Haaga-Helia
ammattikorkeakoulu Oy

Varausjärjestelmä matkailualan yritykselle

Kaisu Karvonen

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2018

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Kaisu Karvonen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Varausjärjestelmä matkailualan yritykselle

Sivu- ja liitesivumäärä
30 + 1

Opinnäytetyön tavoitteena on suunnitella ja toteuttaa matkailualan yritykselle selainpohjainen varausjärjestelmä, joka sujuvoittaa yrittäjän liiketoimintaa asiakasvarausten osalta. Toimeksiantajana opinnäytetyölle toimii Nuuksion Taika oy, joka tarjoaa erilaisia luontopalveluita ja majoitusta Nuuksion keskellä idyllisessä järvimaisemassa. Yritys kohdistaa palvelunsa erityisesti ryhmille ja yrityksille, joille järjestetään kokous- ja virkistyspäiviä sekä ryhmän toiveiden mukaan räätälöityjä luonto -ja majoituspaketteja.

Varausjärjestelmän avulla yrityksen potentiaaliset asiakkaat voivat tehdä toiveitaan vastaavan tarjouspyynnön yrittäjälle. Varausjärjestelmä pyrkii esittämään tarjottavat palvelut aiempaa selkeämmin asiakkaalle. Asiakkaan täytettyä tiedot varausjärjestelmään tarjouspyyntö lähetetään yrittäjän sähköpostiin. Järjestelmä ei tallenna asiakkaan dataa mihinkään, vaan tietojen tallennus ja varauksen varmistus jäävät yrittäjän vastuulle. Toteutus tapahtuu React, Node.js ja MongoDB teknologioita hyödyntäen. Yrittäjän käyttämä Google Kalenteri integroidaan järjestelmän ajanvarauskalenteriin.

Asiasanat
React, Node.js, MongoDB

Sisällys

1 Johdanto

Opinnäytetyön tarkoituksena on suunnitella ja toteuttaa matkailualan yritykselle selainpohjainen varausjärjestelmä. Toimeksiantajana opinnäytetyölle toimii Nuuksion Taika oy. Yritys tarjoaa erilaisia luontopalveluita ja majoitusta Nuuksion keskellä idyllisessä järvimaisemassa. Yritys kohdistaa palvelunsa erityisesti ryhmille ja yrityksille, joille järjestetään kokous- ja virkistyspäiviä sekä ryhmän toiveiden mukaan räätälöityjä luonto- ja majoituspaketteja. Opinnäytetyö toteutetaan yhteistyössä Laurea ammattikorkeakoulun opiskelijoiden kanssa, jotka ovat oman opinnäytetyönsä puitteissa haastatelleet yrityksiä ja selvittäneet yritysten tarpeita yrittäjän tarjoamiin majoitus- ja virkistyspalveluihin. Haastatteluista keräämiensä tietojen pohjalta Laurean opiskelijat ovat luoneet alustavan näkymän varausjärjestelmästä, jota käytetään käyttöliittymän pohjana varsinaisen varausjärjestelmän toteuttamisessa.

Varausjärjestelmän avulla yrityksen potentiaaliset asiakkaat voivat tehdä toiveitaan vastaavan tarjouspyynnön yrittäjälle. Opinnäytetyön tavoitteena on sujuvoittaa yrittäjän liiketoimintaa asiakasvarausten osalta esittämällä tarjottavat palvelut aiempaa selkeämmin ja antamalla alustava hinta-arvio valituista palveluista asiakkaalle. Varausjärjestelmä sisältää varauslomakkeen, jossa listataan kaikki yrittäjän tarjoamat palvelut sekä niiden alustavat hinnat. Asiakas katsoo lomakkeella olevasta ajanvarauskalenterista onko hänen toivomalleen aikavälille vielä mahdollista varata palveluita ja sen jälkeen täyttää varauslomakkeen. Lomakkeen tiedot lähetetään sähköpostin välityksellä yrittäjälle, joka varmistaa varauksen asiakkaalle. Varausjärjestelmä ei tallenna lomakkeesta syntyvää dataa mihinkään, vaan asiakkaan tietojen tallennus ja varauksen varmistus jäävät yrittäjän vastuulle. Varausjärjestelmä toteutetaan React, Node.js ja MongoDB teknologioita hyödyntäen. Yrittäjän käyttämä Google Kalenteri integroidaan järjestelmän ajanvarauskalenteriin.

2 Käytettävät teknologiat

Tässä luvussa esitellään järjestelmän luomisessa ja hallinnoimisessa käytettävät teknologiat.

2.1 Reactin ideologia ja yhteys muihin teknologioihin

React on Facebookin kehittämä avoimen lähdekoodin Javascript-kirjasto, jonka avulla voi luoda käyttöliittymiä selainpohjaisiin sovelluksiin. Reactia voidaan kirjoittaa Javascriptillä tai JSX-syntaksia hyödyntäen. Käytössä olevat funktiot ja tarkka syntaksi riippuvat React-sovelluksen ECMAScript-versiosta. ECMAScript on standardi, joka määrittää Javascript-kielen toteutustavan. Reactissa käyttöliittymäelementit koostuvat komponenteista, joiden avulla sovellus jaetaan itsenäisiin osiin. Samaa komponenttia voi uudelleenkäyttää käyttöliittymän eri osissa. Jokaisella komponentilla on tila, jossa voidaan säilöä komponentin käyttämää dataa. Komponentit voivat välittää toisilleen arvoja propsien eli ominaisuuksien avulla. Komponentti A voi välittää komponentille B propsin ’nimi’, jolloin jälkimmäinen komponentti voi käyttää nimi-propsin arvoa. React-sovelluksen rakentamisessa voi hyödyntää myös npm-paketinhallintajärjestelmän tarjoamia valmiita koodipaketteja. (React 2018a; React 2018b; Tutorialspoint 2018b.)

Reactin tilan hallintaan voidaan hyödyntää Reduxia, jonka avulla useampi komponentti saa käyttöönsä sovelluksen yhteisen tilan. Yhteisessä tilassa voidaan säilöä muun muassa sovelluksen rajapinnasta tulevaa dataa. Rajapintana React-sovellukselle voi toimia Javascriptillä rakennettu Node.js-palvelin. Yhteys sovelluksen tietokantaan muodostetaan palvelimella. Tietokantana voi toimia esimerkiksi NoSQL-mallinen MongoDB. Sovelluksen kehittämistä eri ympäristöissä voi yksinkertaistaa virtualisoimalla sovelluksen Docker-virtualisointialustan avulla. Mahdollistaakseen yhteistyön muiden kehittäjien kanssa React-sovelluksen voi lisätä Githubiin, joka on ohjelmistoprojektien julkaisualusta. (Docker 2018a; Gagliardi 2018; Wodehouse 2015.)

2.2 Komponenttimalli

Reactilla tehty käyttöliittymä koostuu komponenteista, jotka jakavat käyttöliittymän itsenäisiin, erillisiin osiin. Jos komponentti päivittää käyttöliittymää, se määritellään luokaksi. Komponentti voi kuitenkin olla myös pelkkä funktio, joka ei päivitä lainkaan käyttöliittymää. Koska komponentit ovat itsenäisiä, samaa komponenttia voidaan helposti uudelleenkäyttää käyttöliittymän useissa eri osissa, esimerkiksi kerran luotu sivupalkki-komponentti voidaan renderöidä sovelluksen jokaiselle sivulle. React-komponentin rakentamisessa voidaan hyödyntää JSX-syntaksia, joka on XML-ohjelmointia muistuttava Javascript-syntaksi.

JSX:n käyttäminen Reactissa ei ole pakollista ja sen sijaan voidaan käyttää Javascriptiä. Kuten XML:ssä, myös Reactin JSX-komponentilla voi olla nimi, propseja eli ominaisuuksia ja mahdollisesti useita lapsielementtejä. Propsit ovat toiselta React-komponentilta saatuja arvoja. (Madein 2017; React 2018a.)

```
<MyComponent className="custom">
  <div>
    Hello world!
  </div>
</MyComponent>
```

Kuva 1. JSX-syntaksilla tehty React-komponentti

Kuvan 1 komponentilla on props className ja lapsielementti div. Selaimet eivät osaa tulkita JSX:ää, joten komponentti käännetään käyttämällä jotakin kääntötyökalua (esimerkiksi Babel) selaimen ymmärtämäksi Javascriptiksi. Kääntäjien myötä voidaan käyttää uusia syntakseja, välittämättä siitä tuleeko selain vielä niitä. Alla oleva kuva 2 havainnollistaa kääntäjän tuottamaa Javascriptia kuvan 1 React-komponentista. (Madein 2017.)

```
React.createElement(
  MyComponent,
  { className: "custom" },
  React.createElement(
    "div",
    null,
    "Hello world!"
  )
);
```

Kuva 2. Selaimen tulkitsema Javascript kuvan 1 komponentista

2.3 Komponentin tila, propsit ja elinkaari

Jokaisella React-komponentilla on oma paikallinen tilansa, jossa voidaan säilöä kaikkea komponentin tarvitsemaa dataa, esimerkiksi käyttöliittymän syötekenttiin kirjoitettuja arvoja. Komponentti voi muuttaa omaa tilaansa. Kuva 3 esittää React-komponenttia, joka kysyy käyttäjän nimeä ja tervehtii käyttäjää annetun syötteen perusteella. Komponentin handleOnChange-metodin sisällä tilaa muokataan kutsumalla metodia this.setState(), joka aktivoi komponentin render()-metodin. Muuttunut komponentti päivitetään sitten DOMiin eli dokumenttioliomalliin. Tilan asettaminen komponentin ensimmäisen renderöinnin aikana tulee tehdä joko konstruktorissa tai kuvan 3 mukaisesti komponentin luokka-ominaisuutena. Esimerkin komponentti hyödyntää Babel-kääntäjän tuomia ominaisuuksia Reactissa, minkä takia konstruktoria ei ole näkyvillä komponentissa. Tilaa ei saa koskaan

muulloin muokata kutsumalla state = { }, sillä se ei aktivoi renderöintiä komponentille ja voi johtaa tilan epäjohdonmukaisuuksiin. Tilaa käsitellessä tulee myös muistaa, että se päivittyy asynkronisesti. Jos tilassa on esimerkiksi muuttuja number (arvo 0) ja arvoa halutaan suurentaa kolme kertaa peräkkäin kutsumalla this.setState({ number: this.state.number + 1 }), number-muuttujan arvo on silti edelleen 1. React päivittää tilan vasta kun komponentti on renderöity uudelleen. This.state.number on siis joka kerta kutsuttaessa setState()-metodia 0. Vähentämällä turhia renderöintejä voidaan varmistaa että ohjelman suorituskyky ei kärsi. (React 2018b.)

```
class Application extends React.Component {
  state = { name: ''};

  handleOnChange = (event) => {
    this.setState({name: event.target.value});
  }

  render() {
    return <div>
      Mikä on nimesi?
      <input onChange={this.handleOnChange} value={this.state.name} />
      <p>Hei vaan {this.state.name}!</p>
    </div>;
  }
}
```

Kuva 3. React-komponentti, joka kysyy käyttäjän nimeä ja tervehtii käyttäjää annetun syötteen perusteella

React-komponentilla voi olla myös toiselta komponentilta saatuja propseja, joita voidaan hyödyntää käyttöliittymänäkymän rakentamisessa. Propsien jakaminen toisille komponenteille mahdollistaa sen, että komponentteja voi jakaa pieniin osiin ja kukin komponentti saa silti tarvittavan datan käyttöönsä. Komponentti voi saada toiselta komponentilta propsina esimerkiksi listan, ja pystyy lukemaan sen sisällön kutsumalla this.props.lista. Toisin kuin tilaa, komponentin propsien arvoja ei saa muuttaa, vaan komponentti voi ainoastaan lukea niiden arvon. Jos komponentille jaettua propsin arvoa tarvitsee muokata, sen voi tallentaa tilaan ja muokata sieltä käsin. Jos komponentti ei tarvitse lainkaan tilaa, sen voi määritellä funktiona, jolle jaetaan parametrinä propsit (mikäli niitä tarvitaan). Jos tilaa tulee kuitenkin käsitellä, pitää komponentti luoda luokkakomponenttina, jolla on konstruktori ja elinkaarimetodit käytössään. (React 2018a; React 2018c.)

Luokkamuotoisella komponentilla on elinkaarimetodit, joita kutsutaan komponentin syntyessä sekä tilan ja propsien muuttuessa. Elinkaarimetodeja käyttämällä komponentille

voidaan kertoa, että sen tulisi päivittyä uudestaan tai muuttaa tilan arvoa. Jos komponentin täytyy kutsua esimerkiksi jotakin rajapintaa, se tulee tehdä componentDidMount()-metodissa, joka aktivoituu heti komponentin ensimmäisen renderöitymisen jälkeen. (React 2018a; React 2018c.)

2.4 Renderöinti ja virtuaalinen DOM

React-komponenttien renderöinti DOMiin tapahtuu yleensä yhden root-noden kautta. Kaikki HTML-elementit ovat nodeja eli objekteja. Koodissa määritellään node ja haluttu komponentti sekä root-node jaetaan ReactDOM.render()-funktiolle. ReactDOMille jaettu komponentti on yleensä ns. yläkomponentti, joka voi sisältää useita eri komponentteja. Kuvassa 4 on esimerkki React-komponentin renderöimisestä DOMiin. Jos esimerkin React-ohjelma ajettaisiin, selaimelle tulisi näkyviin teksti ’Hi there!’. Sen sijaan että React muokkaisi itse DOMia renderöidäkseen komponenteissa tapahtuvia muutoksia, se hyödyntää virtuaalista DOMia, joka on kuin Reactin paikallinen ja yksinkertaistettu kopio varsinaisesta DOMista. Muutoksen tapahtuessa DOMissa oleva elementti ja kaikki sen mahdolliset lapsielementit joutuvat käymään läpi rakentamis- ja uudelleenpiirto-operaation, joiden aikana kukin node paikannetaan ja muutokset käsitellään. Esimerkki tästä olisi lista, jonka yhtä elementtiä muutetaan, mutta DOM joutuu silti renderöimään kaikki listan jäsenet uudelleen. Käyttäjä eikä käyttöliittymä itse voi tehdä muita toimenpiteitä DOMin päivityksen aikana. Välttääkseen DOMin päivittämisen hitauden ja ylläpitääkseen hyvää suorituskykyä Reactin virtuaalinen DOM pyrkii minimoimaan nämä läpikäytävät vaiheet. (Kurian 2017.)

```
const hello = <h1>Hi there!</h1>;
ReactDOM.render(hello, document.getElementById('root'));
```

Kuva 4. Komponentin renderöinti DOMiin

Virtuaalisen DOMin päivittymistä voidaan havainnollistaa kuvan 5 Laskuri-komponentilla, jossa käyttäjä voi summata tai miinustaa kahta lukua. Kun käyttäjä kirjoittaa syötekenttiin kaksi lukua ja ja painaa Summaa-nappia, ohjelma laskee luvut yhteen, päivittää komponentin tilaan lukujen summan ja näyttää käyttäjälle laskun tuloksen. Kun komponentin tilaa päivitetään ja se aktivoi render() -metodin, Laskuri-komponentti merkitään virtuaalisessa DOMissa likaiseksi eli se sisältää muutoksia. Päivittääkseen muutokset varsinaiseen DOMiin React käy läpi likaiseksi merkityt komponentit läpi ja etsii niistä päivittyneet elementit vertaamalla komponentin aikaisempaa tilaa uuteen tilaan. Lopuksi pelkästään muuttuneet nodet päivitetään DOMiin. (Kurian 2017.)

Kuva 5. Laskuri-komponentti

2.5 Redux

Redux on Javascript-kirjasto sovelluksen tilan hallinnointiin. Se on hyödyllinen erityisesti kun sovelluksessa on useita komponentteja, jotka tarvitsevat samaa dataa tilasta ja halutaan välttää tilan jakaminen propseina useille muille komponenteille. Redux mahdollistaa yhteisen säiliön luomisen tilalle, johon useampi komponentti voi päästä käsiksi. Tilaa ei luoda manuaalisesti, vaan se syntyy Reduxin reducer-funktion avulla. Reducer ottaa vastaan kaksi parametriä: sovelluksen nykyisen tilan ja toiminnon (action). Toiminto on yksinkertainen objekti, jonka avulla Reduxin säiliö saa tiedon muuttuneesta datasta. Toiminto luodaan action creator -funktion kautta, joka palauttaa toiminto-objektin. Objektilla on tekstimuotoinen tyyppi, joka kuvaa toimintoa sekä mahdollinen Reduxin tilaan tallennettava data. Selvyyden vuoksi toiminnon tyyppi tallennetaan vielä erilliseen muuttujaan, johon viitataan action creator -funktiossa. Kuva 6 havainnollistaa toiminnon luomista action creator -function avulla. (Gagliardi 2018.)

```
const ADD_NUMBER = 'NUMBERS/ADD_NUMBER';

export const addNumber = number => ({ type: ADD_NUMBER, number });
```

Kuva 6. Toiminnon tyyppi ja action creator -funktio.

```
const defaultState = { numbers: [] };

export default function (state = defaultState, action) {
  switch (action.type) {
    case ADD_NUMBER:
      return { ...state, numbers: [...state.numbers, action.number] };
    default:
      return { state };
  }
}
```

Kuva 7. Reduxin tilan muutos

Kun toimintoa kutsutaan, säiliö aktivoi reducer-funktion, jossa tilan muutos tapahtuu. Toisin kuin komponentin paikallisen tilan päivitys, Reduxin tilan muutoksessa ei käytetä lainkaan setState()-metodia. Kuva 7 on esimerkki Reduxin tilan päivittämisestä reducer-

funktiossa. Reducer käy switch-funktion avulla läpi mikä toiminto saapui reduceriin ja muuttaa tilaa sen mukaan. Alkuperäiseen tilaan ei kosketa lainkaan, vaan spread-operaattorin (…-syntaksi) avulla tilasta otetaan kopio ja palautetaan se. Reduxin säiliö täytyy vielä yhdistää sovellukseen, jotta sitä voidaan käyttää. Kuvassa 8 nähtävän esimerkin mukaisesti säiliö luodaan createStore-funktion avulla, jolle annetaan parametrinä reducer. Luotu säiliö jaetaan Reduxin Provider-komponentille, joka tekee säiliön sovelluksen käytettäväksi. Reduxin tila ja toiminnot täytyy vielä yhdistää propseihin. Tämä hoituu Reduxin connect-funktion avulla, jossa määritellään propseja vastaavat tilan arvot ja kerrotaan toiminnot. Kuvan 9 esittämä funktio tilan ja toimintojen yhdistämisestä on aukikirjoitettuna connect(mapStateToProps, mapDispatchToProps). mapDispatchToProps-funktion sisällä voidaan yhdistää action creator -funktiot kutsumalla bindActionCreators-funktiota, jolloin kaikki toiminnot tuodaan kerralla komponentin käyttöön. Nyt komponentista voidaan kutsua funktiota this.props.addNumber(2), joka lisää luvun 2 Reduxiin tilan numbers-listaan. (Gagliardi 2018.)

```
const store = createStore(reducer);

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

Kuva 8. Säiliön luominen ja sovelluksen kietominen Provider-komponenttiin

```
export default connect(
  state => ({
    numbers: state.numbers.numbers,
  }),
  dispatch => (bindActionCreators({
    ...numberActions,
  }, dispatch)),
)(App);
```

Kuva 9. Sovelluksen tilan ja toimintojen yhdistäminen

Jos sovellus tekee pyyntöjä api-rajapintaan, voi olla aiheellista ottaa käyttöön Reduxin väliohjelmisto (middleware). Väliohjelmistojen tarkoituksena on muun muassa helpottaa sovelluksen asynkronisten pyyntöjen ja selaimen välimuistin käsittelyä. Myös Reduxin

toiminnot pysyvät selkeinä, kun rajapintapyyntöjä ei käsitellä niiden sisällä vaan väliohjelmistossa. (Redux-Saga.)

2.6 ECMAScript

ECMAScript (ES) on komentosarjoja suorittaville kielille (komentokielet) luotu standardi, jonka pohjalta syntyi Javascript. Standardin on tarkoitus määrittää sitä hyödyntävien kielien perusominaisuudet ja niiden toteutustavan. ECMAScriptistä julkaistaan uusi versio vuosittain, ja useimmat selaimet tukevat nyt ECMAScript 5 -versiota. (Tutorialspoint 2018b.)

Javascript-ohjelma voidaan määritellä käyttämään tiettyä ES-versiota. Uusien versioiden myötä Javascriptiin tulee uusia funktioita käytettäväksi ja syntaksiin voi tulla muutoksia. Monet React-applikaatiot hyödyntävät ES6 -versiota, joka vähentää kirjoitettavan koodin määrää ES5-versioon verrattuna. Koska monet selaimet eivät tue uusimpia ES-versioita, Javascript-ohjelma tulee kääntää aiempaan selaimen tukemaan versioon jonkin kääntäjätyökalun avulla. (Tutorialspoint 2018b.)

2.7 Node.js

Node.js on avoimen lähdekoodin suoritusaikainen ympäristö Javascriptille, jonka avulla voidaan ajaa Javascript-ohjelmia myös palvelinpuolella. Node käyttää V8 Javascript -moottoria koodin suorittamiseen, mikä tekee Nodesta nopean ja tehokkaan. Node hyödyntää lisäksi asynkronista input/output -tekniikkaa, jolloin useat samanaikaiset pyynnöt palvelimelle eivät estä toistensa kulkua. Palvelin voi näin ollen vastaanottaa pyynnön ja siirtyä kutsumaan seuraavaa saamatta vielä ensimmäiseen vastausta. Tämä on etu verrattuna esimerkiksi Apachen palvelimiin, joilla on rajallinen kyky suorittaa samanaikaisia pyyntöjä. (Tutorialspoint 2018a.)

Node-palvelimen pystyttäminen on yksinkertaista ja vaadittavien konfiguraatioiden määrä on hyvin vähäinen. Yksinkertaisimmillaan palvelimen voi pystyttää npm-pakettina ladattavan http-kirjaston avulla kutsumalla http.createServer().listen(3000), jolloin paikallinen palvelin käynnistyy portissa 3000. Erilaisten npm-pakettien avulla palvelin voi suorittaa paljon erilaisia toimintoja. Express on suosittu Noden kanssa käytettävä npm-paketti, jonka avulla Node pystyy käsittelemään erimuotoista dataa kuten JSONia ja reitittämään palvelimelle saapuvat pyynnöt eri päätepisteisiin. Nodea ei suositella käytettävän paljon suorituskykyä vievissä tehtävissä. (Tutorialspoint 2018a.)

2.8 MongoDB

MongoDB on avoimen lähdekoodin tietokantaohjelma, joka määritellään NoSQL -tietokannaksi. MongoDB:n käyttöönotto on varsin helppoa ja vaivatonta, tietokannan saa päälle minuuteissa. Tietokantahaut ovat nopeita ja kanta skaalautuu vaivattomasti. Data tallennetaan tietokantaan JSON-dokumentteina, mikä tekee tietokantarakenteiden hallinnoimisesta ja muokkaamisesta helppoa. Relaatiotietokannan taulua vastaava nimike MongoDB:ssä on kokoelma. MongoDB:ssä ei ole SQL-tietokantojen liitoksia, vaan se hyödyntää dokumentteihin perustuvaa hakukieltä, jonka avulla toisiinsa yhteydessä olevien eri dokumenttien haku onnistuu. Usein dokumenteista tehdäänkin hierarkisia ja dokumentin ominaisuuden alle saatetaan tallentaa esimerkiksi listoja, jotta datahaut pysyisivät mahdollisimman helppoina. (Tutorialspoint 2018c.)

Toisin kuin relaatiotietokannoissa, joissa tietokantataulun rakenne eli skeema tulee olla määriteltynä ennenkuin kantaan voidaan tallentaa dataa, MongoDB:ssä skeeman voi luoda joustavasti samalla kun data tallennetaan kantaan. Skeeman muuttaminen ei vaadi erillisiä tietokannassa tehtävien komentojen läpivientiä, vaan uusia kenttiä voidaan myös tarpeen mukaan tuoda dataa tallennettaessa. Skeeman muokattavuuden helppous saattaa tosin johtaa siihen, että tietokantaa ei suunnitella kunnolla, vaan muutetaan jatkuvasti matkan varrella. Tämä voi tietysti aiheuttaa ongelmia ja epäjohdonmukaisuuksia uudessa ja vanhassa datassa tietokannassa. (Tutorialspoint 2018c.)

2.9 Npm

Npm on paketinhallintajärjestelmä Javascriptille. Sen avulla kehittäjät voivat jakaa tekemiään koodinpätkiä eli Javascript-paketteja muille käytettäväksi ja ladata itselleen muiden tekemiä paketteja. React-ohjelmassa voidaan käyttää npm:n avulla asennettuja paketteja. Npm koostuu kolmesta eri komponentista: nettisivusta, komentoriviohjelmasta ja pakettirekisteristä. Nettisivuilta voi etsiä paketteja ja kunkin paketin sivu sisältää myös usein paketin asennusohjeet. Pakettirekisteri on julkinen tietokanta, joka sisältää yli 600 000 Javascript-pakettia. (Npm 2018a.)

Komentoriviohjelman avulla paketin voi ladata itselleen. Asentaakseen paketin tulee antaa komento npm install <name_of_package>, jolloin paketti asennetaan oletuksena tämänhetkisen kansion alla olevaan node_modules-kansioon. Paketin asennettu versio näkyy projektin package.json -tiedostossa, joka on npm:n käyttämä konfiguraatiotiedosto sovelluksessa ja jonne npm lisää tiedon asennetusta paketista. (Npm 2018b.)

2.10 Github

Github on Git-versionhallintatyökalua käyttävien projektien julkaisualusta, jossa sovelluskehittäjät voivat jakaa tekemiään projekteja ja tehdä yhteistyötä muiden kehittäjien kanssa. Githubin käyttämä Git-työkalu mahdollistaa projektin koodin kehityksen seurannan myös suurissa projekteissa, joissa on paljon kehittäjiä työstämässä koodia samanaikaisesti. Gitin avulla yhteistyön tekeminen helpottuu, kun toinen kehittäjä lataa koodinsa versionhallintasivustolle, josta muut voivat katsoa ja kehittää sitä eteenpäin. Jos projektin koodi jostain syystä hajoaa jossakin vaiheessa, Gitin versiohistoriasta saa palautettua aikaisemman version projektista. Koodin työstö tapahtuu haaroissa, joita voidaan jaotella esimerkiksi seuraavasti: master, dev, feature/xx. Master-haara tulisi olla suojattu, jotta kuka tahansa projektin kehittäjä ei voi lisätä siihen muutoksia ennen kuin ne on tarkistettu ja hyväksytty. Kukin kehittäjä voi työstää muutoksia omassa haarassaan ja yhdistää haaransa sitten deviin eli kehityshaaraan. (Wodehouse 2015.)

Projekti julkaistaan Githubin repositoriossa, josta koodia voi katsella ja jakaa muille nähtäväksi. Tärkeä osa repositorion toiminnallisuutta on issuet, joita projektin kehittäjät ja muuten vain projektista kiinnostuneet voivat lisätä. Issuet ovat lähdekoodista löytyneitä bugeja, toiveita ja kysymyksiä puuttuvista / nice-to-have -ominaisuuksista. Issueiden avulla projektin kulkua voidaan seurata, ja kehittäjät saavat myös kuvan siitä mitä muut työstävät tällä hetkellä projektissa. (Wodehouse 2015.)

2.11 Docker

Docker on alusta ohjelmistojen kehittämiseen ja ajamiseen virtualisoidussa ympäristössä. Dockerin avulla ohjelman voi eristää omasta infrastruktuurista ja viedä kehityksestä pienemmällä vaivalla tuotantoon ilman laitteistoalustan tai käyttöjärjestelmän aiheuttamia ongelmia. Docker mahdollistaa ohjelman paketoinnin ja ajamisen virtualisoidussa ympäristössä, jota kutsutaan virtuaalikontiksi. Kontti on Docker-kuvan ajettava versio, joka voidaan yhdistää useisiin verkkoihin. Kontit ovat kevyempiä kuin virtuaalikoneet, koska ne ajetaan suoraan isäntäkoneen ytimessä. (Docker 2018a.)

Docker hyödyntää virtualisoinnissa kuvaa, joka on eräänlainen malli virtuaalikontin luomisesta. Docker tarjoaa virallisia kuvia esimerkiksi ubuntu-kontin luomiseen, mutta kuvia voi myös luoda itse omiin tarpeisiin. Konttien myötä jatkuvan integraation ja käyttöönoton prosessi on suoraviivainen, kun kehittäjät voivat työstää koodiaan paikallisesti omissa virtuaalikonteissaan, ja tuotantoympäristöön viedään päivitetty Docker-kuva. (Docker 2018a.)

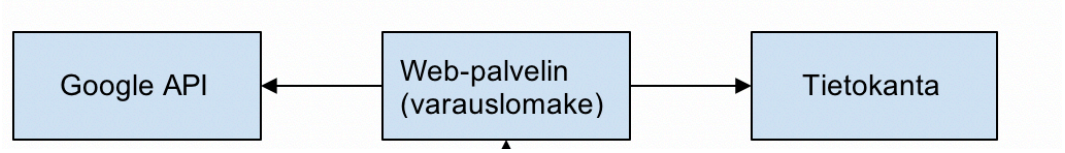
3 Järjestelmän suunnittelu

Tässä luvussa käydään läpi opinnäytetyön myötä syntyvää järjestelmää koskevat vaatimukset sekä käyttöliittymämallit ja tietokantarakenne.

3.1 Vaatimusmäärittely

Opinnäytetyössä toteutetaan varausjärjestelmä, jonka kautta yrityksen potentiaaliset asiakkaat voivat tehdä toiveitaan vastaavan tarjouspyynnön. Järjestelmä sisältää varauslomakkeen, jossa asiakkaalta kysytään yhteystietoja, palveluiden ja majoituksen toivottua ajankohtaa sekä haluttuja lisäpalveluita ja aktiviteetteja. Asiakas täyttää lomakkeen tiedot, ja yrittäjälle lähtee sähköposti annettujen tietojen perusteella. Yrittäjä voi saadun tarjouspyynnön perusteella tehdä tarkentavia kysymyksiä asiakkaalle ja hyväksyä tai hylätä tarjouspyynnön sähköpostin välityksellä. Järjestelmä ei tallenna asiakkaan lähettämiä tietoja mihinkään, koska tarjouspyynnön lähettäminen ei automaattisesti tarkoita, että varaus olisi hyväksytty. Järjestelmän käyttäjäryhmiä ovat yksityishenkilöt ja yritykset, jotka haluavat käyttää yrittäjän tarjoamia palveluita. Yksityishenkilöitä voivat olla ketkä tahansa yksittäiset henkilöt tai ryhmä, jota ei voi määritellä yritykseksi, yhdistykseksi tai organisaatioksi (esim. perhe tai kaveriporukka). Järjestelmän käyttäjinä voivat toimia joko suomea tai englantia ymmärtävät henkilöt.

Järjestelmän käyttöliittymän tulee olla helppokäyttöinen ja selkeä, jotta sen käyttö on luontevaa. Käyttäjän ei oleteta tuntevan yrittäjän tarjoamia palveluita etukäteen käyttääkseen järjestelmää; hänellä tulee kuitenkin olla toimiva internet-yhteys käyttääkseen järjestelmää. Käyttöliittymäelementtien tulee olla loogisesti järjestelty, jotta järjestelmä ei tarvitse erillisiä käyttöohjeita. Varauslomakkeen tulee sisältää kalenteri, josta asiakas näkee varattavissa olevat ajan palveluille ja pystyy valitsemaan ajanjakson vapaiden aikojen pohjalta. Kun yrittäjä on vahvistanut asiakkaan varauksen, hän lisää omaan Google Kalenteriinsa varauksen, josta tieto päivittyy varauslomakkeen kalenteriin. Jos asiakas valitsee lomakkeella olevansa yksityisasiakas, hänelle näytetään eri valintavaihtoehtoja kuin yritysasiakkaalle. Lomakkeella tehtyjen valintojen perusteella asiakas näkee palveluiden alustavan hinnan ja selityksen mistä hinta koostuu. Käyttäjän tulee saada palautetta virheellisesti täytetyistä kentistä ja lomakkeen lähettäminen tulee olla estetty kunnes virhe on korjattu. Kun käyttäjä lähettää lomakkeen, järjestelmän tulee antaa ilmoitus asiakkaalle joko onnistuneesta lomakkeen lähettämisestä tai sattuneesta virhetilasta. Lomakkeen kieleksi voi valita joko suomen tai englannin ja kieli tulee olla luonnollista ja ymmärrettävää.

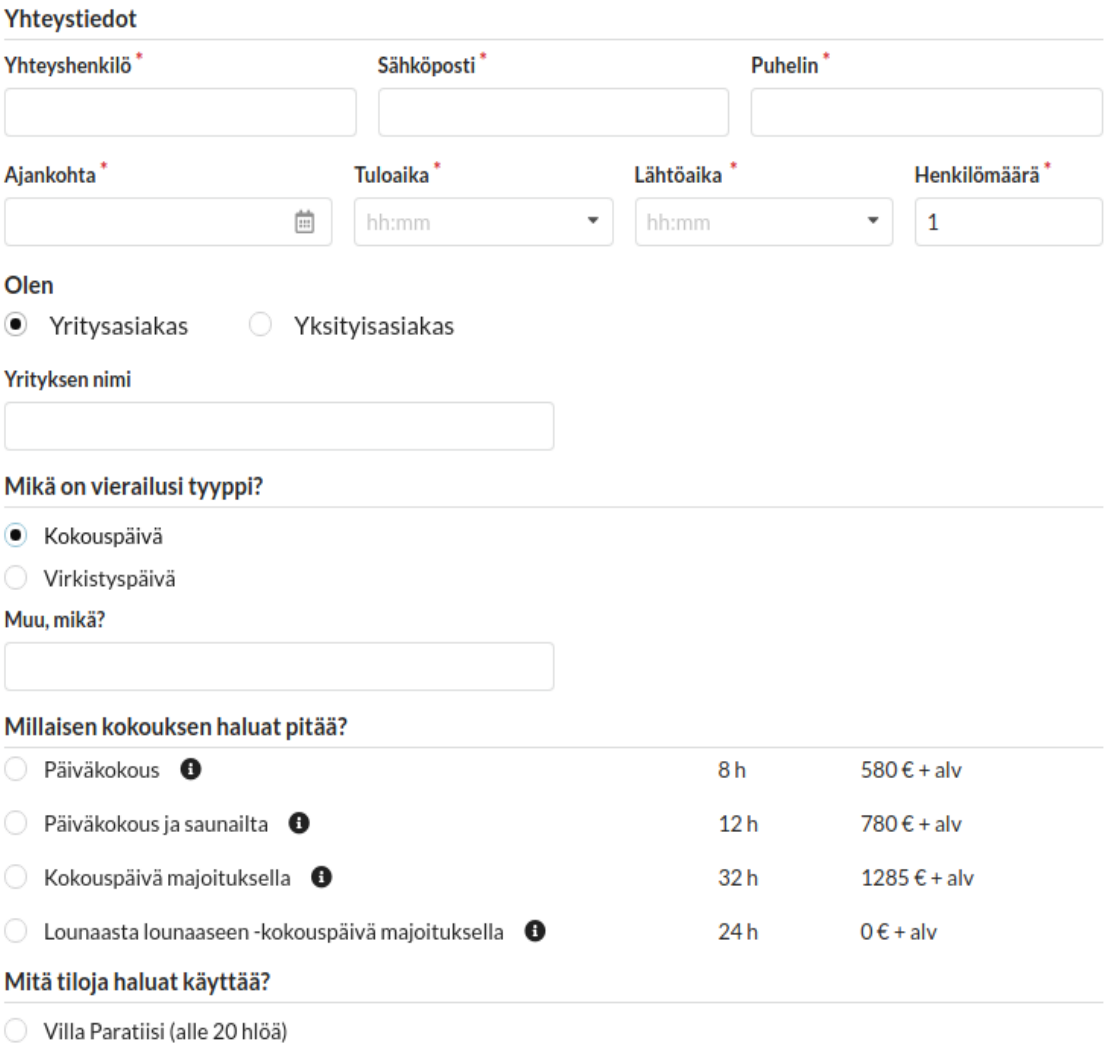


Kuva 10. Varausjärjestelmän toimintaympäristö

Järjestelmän tulee olla yhteensopiva yleisten web-selainten (mm. Chrome, Firefox ja IE) kanssa ja kyetä palvelemaan useampaa käyttäjää samanaikaisesti. Sen ei tarvitse kuitenkaan olla mobiililaitteille skaalautuva. Kuten kuva 10 havainnollistaa, järjestelmä sijaitsee web-palvelimella, josta se ottaa yhteyden tietokantaan hakeakseen käyttöliittymäkomponenttien tiedot. Järjestelmä kutsuu Google API:an näyttääkseen varauslomakkeen kalenterissa yrittäjän Google Kalenterin tiedot vapaista ja varatuista ajoista.

3.2 Käyttöliittymä

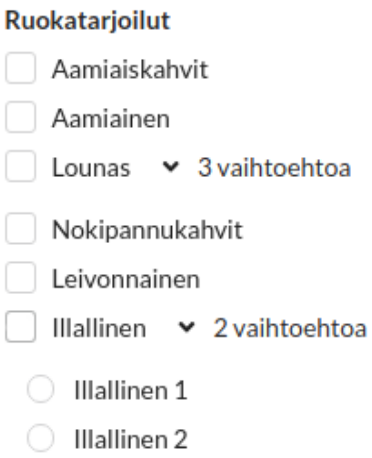
Järjestelmän käyttöliittymä koostuu lomakkeesta, joka voidaan jakaa kolmeen osa-alueeseen: yhteystiedot, asiakastyyppiin liittyvät valinnat ja lisäpalvelut. Lomakkeen käyttöliittymän suunnittelu lähti liikkeelle Laurea ammattikorkeakoulun opiskelijoiden tekemän käyttöliittymämallin pohjalta (liite 1). Malli antoi hyvän pohjan käyttöliittymälle, vaikkakin se sisältää jonkin verran puutteita yrittäjän asettamiin vaatimuksiin nähden. Se havainnollistaa ainoastaan yritysasiakkaan näkymää, eikä mallin perusteella voi tehdä minkäänlaisia päätelmiä kalenterin käyttöliittymästä. Mallissa kalenteri on jaettu kahteen erilliseen kenttään: tulo- ja lähtöpäiväkenttä. Lopullisessa käyttöliittymämallissa ajanvaraus on kuitenkin yhdistetty yhteen kenttään, jotta käyttäjän ei tarvitse erikseen avata kahta ajanvarauskalenteria valitessaan ajanjaksoa vierailulle.



Kuva 11. Malli käyttöliittymän yhteystieto- ja asiakastyyppiin liittyvistä valinnoista

Kuva 11 havainnollistaa suunniteltua käyttöliittymää yhteystietojen ja yritysasiakkaaseen liittyvien valintojen osalta. Tuloaika- ja lähtöaika kentät ovat alasvetovalikkoja, joista asiakas valitsee kellonajan vierailulleen. Punaisella tähdellä merkityt kentät ovat pakollisia, ja lomaketta ei voi lähetttää täyttämättä näitä kenttiä. Pieni i-merkki kokoustyypin (‘Päiväkokous, ‘Kokouspäivä majoituksella’) nimen vieressä kertoo, että kentästä saa vielä lisäinformaatiota painamalla merkin päältä. Tästä aukeaa ponnahdusikkuna, joka sisältää tarkentavaa tietoa kentästä tai esimerkiksi kokouksen hinnasta. Ajankohta-kenttää painamalla käyttäjälle avataan ponnahdusikkuna, jossa näkyy ajanvarauskalenteri kahden kuukauden näkymällä. Kalenteri näyttää vihreänä varattavat päivät ja harmaana varatut ja menneet päivät. Kalenterin näkymää voi selata eteenpäin 6 kuukautta. Käyttöliittymämalli kalenterista on esitetty kuvassa 12.

Kuva 12. Ajankohdan valinta



Kuva 13. Malli lisäpalveluiden Ruokatarjoilut-kohdasta

Lisäpalvelut jaetaan erillisiin alaotsikoihin (mm. Ruokatarjoilut, Aktiviteetit), joiden alta löytyy checkbox-kenttiä, joista asiakas voi valita mieleiset palvelut. Kuvasta 13 nähdään, kuinka alakenttiä sisältävät checkbox-kentät toimivat: lounaan tai illallisen valitessaan asiakas näkee myös suoraan tarjolla olevat ruokavaihtoehdot, ja voi valita jo tässä vaiheessa haluamansa aterian vierailulleen. Hintoja ei haluta näissä kohdissa näyttää, sillä ne saattavat vaihdella palveluntarjoajasta ja osallistujamäärästä riippuen.

Järjestelmän käyttöliittymää suunnitellessa vertailtiin erilaisia Reactin kanssa käytettäviä käyttöliittymäkirjastoja. Tärkeimpiä vaatimuksia käyttöliittymäkirjastolle oli kattava komponenttien määrä, niiden helppo muokattavuus ja komponenttien miellyttävä visuaalinen ilme. Vertailun kohteeksi valittiin Material UI, Semantic UI ja Ant Design niiden saaman suuren Github-suosion (kirjaston saaman tähtien) perusteella. Kyseisistä kirjastoista löytyy paljon valmiita React-komponentteja erilaisten kenttien ja muiden näkymien

rakentamiseen. Kunkin kirjaston dokumentaatiossa esitellään komponentit ja näytetään käyttöesimerkkejä. Dokumentaation perusteella Material UI:lla vaikuttaa olevan kuitenkin kahteen muuhun kirjastoon verrattuna vähemmän varauslomakkeelle sellaisenaan hyödyllisiä komponentteja. Lisäksi jotkin komponenteista näyttävät visuaalisesti hieman kömpelöiltä. Ant Designilta löytyy datepicker-komponentti, joita voisi mahdollisesti hyödyntää lomakkeen ajanvarauskalenterin tekemiseen. Dokumentaatiossa olevien esimerkkien pohjalta näyttäisi kuitenkin siltä, ettei komponentti ole muokattavissa tarpeeksi sopiakseen lomakkeeseen. Semantic UI:lla vastaavaa komponenttia ei ole. Visuaaliselta ilmeeltään Semantic UI on kuitenkin hieman neutraalimpi kuin Ant Design ja sen dokumentaatio on esimerkkien osalta kattavampi ja selkeämpi, minkä takia se päätettiin ottaa käyttöön. (Ant Design; Material UI; Semantic UI React.)

3.3 Tietokanta

Järjestelmän tietokannassa säilytetään käyttöliittymäelementtien tietoja. Varauslomaketta voi käyttää suomeksi ja englanniksi, joten jokaisen näytettävän kentän tekstitiedot tulee olla saatavilla molemmilla kielillä. Tietokanta sisältää yhden kokoelman: ‘Fields’. Jokaisella kokoelman tietueella on tietokannan automaattisesti generoima id ja itseluotu avain, jonka avulla kentän arvoa voidaan käsitellä ohjelmakoodissa. Avain on käyttöliittymäelementtiä kuvaava tekstimuotoinen sana, esim. ‘Email’. Tietueella on lisäksi suomen- ja englanninkielinen arvo. Tietue voi olla yhteen käyttöliittymän kenttään liittyvä, esimerkiksi syötekenttä, johon kirjoitetaan sähköposti. Tällaisella kentällä on attribuutit _id (tietokannan generoima), key (‘email’), fi (‘Sähköposti’) ja en (‘Email’). Taulukko 1 listaa tietueen kaikki mahdolliset attribuutit.

Taulukko 1. Tietueen attribuutit

Kentän nimi	Selitys	Tyyppi
_id (PAKOLLINEN)	Yksilöivä id kentälle	ObjectId
key (PAKOLLINEN)	Kentän arvo	String
fi	Kentän suomenkielinen teksti	String
en	Kentän englanninkielinen teksti	String
options	Lista objekteja	Array
price	Hinta ilman alvia	Integer
alvPrice	Hinta, johon sisältyy alv	Integer
url	Kenttään liittyvä linkki johonkin nettisivulle	String
duration	Kesto (tuntia)	Integer
info	Kentän lisätietoa	String

max	Maksimikapasiteetti	Integer

Tietueen options-attribuutti on listamuotoinen attribuutti, joka sisältää yhden tai useamman kentän tiedot. Listarakenteen myötä käyttöliittymä osaa päivittää automaattisesti varauslomakkeen näkymää. Ylätietue (kuvassa 14 key-arvo 'meetingEquipment') toimii ns.otsakkeena listatyyppiselle kentälle käyttöliittymässä. Yrittäjä pystyy siis itse muokkaamaan käyttöliittymän tiettyjä elementtejä lisäämällä tietokantaan halutun tietueen options-attribuutin listalle uuden jäsenen. Kuvassa 14 on esimerkki kokousvälineet-tietueesta, jolla on listamuotoinen attribuutti.

```
{ key: 'meetingEquipment', fi:'Kokousvälineet', en:'Meeting equipments', options: [
  { key: 'cables', fi: 'Kaapelit (HDMI, VGA)', en: 'Cables (HDMI, VGA)' },
  { key: 'flipchart', fi: 'Fläppitaulu', en: 'Flip chart' },
]}
```

Kuva 14. Malli options-attribuutin sisältämästä tietueesta

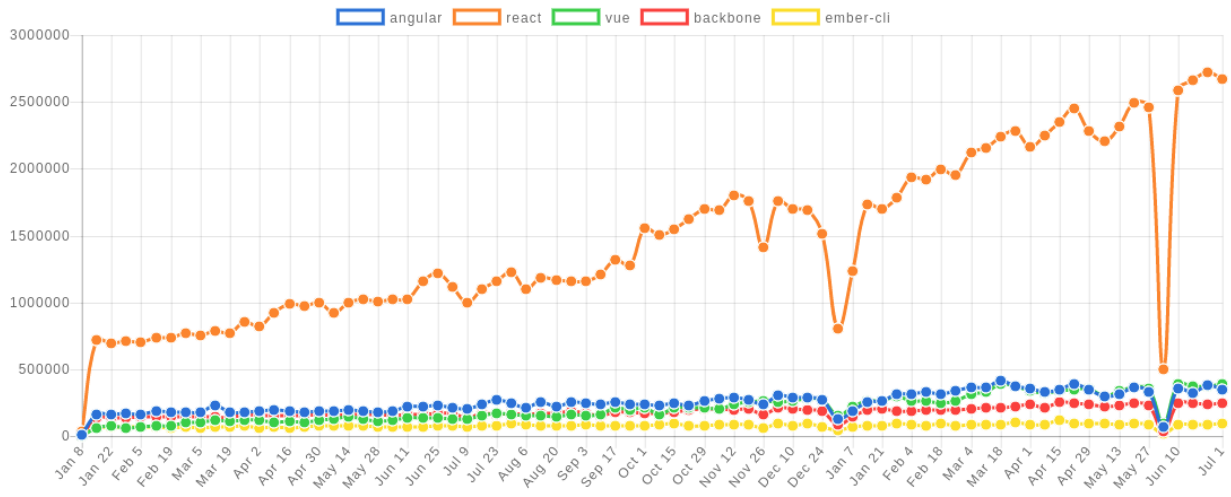
Järjestelmän tietokantaa valittaessa tehtiin vertailua SQL- ja NoSQL-tietokantojen välillä. Kannassa säilytetään ainoastaan käyttöliittymäelementtien tietoja järjestelmän monikielisyyden takia, ja kannan dataa ei ole tarpeen voida muokata käyttöliittymän kautta laisinkaan. Tämän vuoksi tietokantajärjestelmän tärkeimmiksi vaatimuksiksi nousi sen yksinkertaisuus ja vaivattomuus. Käyttöliittymäelementtien tarvittavat tiedot saattavat myös muuttua yrittäjän asettamien vaatimusten mukaan, minkä takia tietokannan rakenteen tulee olla helposti ja nopeasti muokattavissa. NoSQL-tyyppiset tietokannat eivät vaadi tietokantataulun rakenteen määrittämistä ennen kuin kantaan voidaan tallentaa dataa, toisin kuin SQL-tietokannat. Monissa NoSQL-tietokannoissa data tallennetaan tietokantaan avain-arvo-pareja sisältävänä JSON-tyylisinä dokumentteina. Datan tallentaminen JSON-muodossa, jollaisena se halutaan lähettää käyttöliittymälle käytettäväksi, on etu verrattuna SQL-tietokantoihin, joita käytettäessä data pitää aina muuttaa erikseen JSON-muotoon joko manuaalisesti tai jonkin valmiin kirjaston avulla. (Buckler 2015.)

Tietueella voi olla lapsielementtejä sisältävä options-attribuutti, jonka toteuttaminen normalisoidussa SQL-tietokannassa vaatisi lisätauluja. Tämä tuntuu turhan monimutkaiselta pääsääntöisesti staattista dataa sisältävälle tietokannalle, ja SQL-tietokanta toisi vain ylimääräistä työtä kannan rakentamiseen ja päivittämiseen. Tietokannan tietueiden arvojen tulee myös olla helposti yrittäjän muokattavissa, joten kaiken datan säilyttäminen yhden taulun sisällä käyttäen NoSQL-tietokantaa tässä järjestelmässä on järkevin ja vaivattomin ratkaisu. MongoDB on suuressa suosiossa oleva NoSQL-tietokanta ja siitä löytyy erittäin kattavasti dokumentaatiota verrattuna muihin NoSQl-tietokantoihin, minkä takia se valittiin järjestelmän tietokannaksi. (Buckler 2015.)

4 Järjestelmän toteutus

Tässä luvussa käsitellään järjestelmän tekninen toteutus ja lähdekoodin versionhallintaan liittyvät asiat.

4.1 Teknologiavalinnat



Kaavio 1. Javascript-sovelluskehysten npm-pakettien latausmäärät aikavälillä 1/2017 - 7/2018

Varausjärjestelmän käyttöliittymän rakentamiseen tarvittiin Javascript-sovelluskehys, mitä varten tehtiin vertailua tämän hetken suosituista sovelluskehyksistä. Npm trends -sivua hyödyntäen toteutettiin ensin vertailu viimeisen kahden vuoden ajalta yleisesti käytössä olevien Javascript-kehysten npm-pakettien latausmääristä. Kaavio 1 kuvaa npm-pakettien latausmääriä aikavälillä 1/2017 – 7/2018. Siitä voidaan päätellä, että React on selkeästi vertailukohteisiinsa nähden tämän hetken suosituin sovelluskehys. Angularin ja Vuen latausmäärät ovat nousseet melko tasaisesti vuodesta 2017 ja molempien latausmäärät ovat tällä hetkellä suurin piirtein samoissa luvuissa. Myös Backbonen suosio on kasvanut. Kaavion antamien lukujen myötä sovelluskehysten vertailuun valittiin kolme suosituinta kehystä: React, Angular ja Vue. (Npm trends 2018.)

React ja Angular hyödyntävät komponentteja käyttöliittymän rakentamisessa. Toisin kuin Reactissa, Angularissa komponentit käyttävät riippuvuusinjektiota, jonka avulla ne välittävät sovelluksen muille osille niiden tarvitsemat tiedot. Riippuvuusinjektio helpottaa erityisesti komponenttien testausta, mutta voi monimutkaistaa sovelluksen rakennetta. Angularissa komponentti jaetaan tyypillisesti tyylitiedostoon, ulkoasun ja sovelluslogiikan sisältävään tiedostoon. Reactissa komponentti sisältää ulkoasun ja sovelluslogiikan samassa tiedossa. Vuen taas usein sanotaan olevaan Reactin ja Angularin sekoitus. Se hyödyntää

Reactin tavoin komponenttien välisessä kommunikoinnissa propseja ja tilaa. Vuen koko on hyvin pieni ja rakenne yksinkertaisempi verrattuna Angulariin ja Reactiin, mikä mahdollistaa hyvän suorituskyvyn sovellukselle. (Sviatoslav.)

Varausjärjestelmän käyttöliittymän rakenne on melko yksinkertainen, joten Angular todettiin turhan monimutkaiseksi ja suureksi sovelluskehykseksi sen rakentamiseen. Angularin syntaksi on myös melko työläs opetella, joten Vue ja React nousivat ykkösvaihtoehdoiksi. React kehittyy huimalla vauhdilla, minkä takia dokumentaatio ei välttämättä aina ole ajan tasalla. Vuella taas on kattava dokumentaatio, mutta koska sovelluskehys ei ole läheskään yhtä suosittu kuin React, apua ja ratkaisuja ongelmiin ei ole saatavilla yhtä paljon. Suuren suosionsa takia Reactista saatavilla oleva tieto on ylitse muiden ja sille on tarjolla suuri määrä erilaisia valmiita paketteja, minkä takia sen todettiin olevan sopivin sovelluskehys tämän järjestelmän käyttöliittymän rakentamiseen. (Sviatoslav.)

Rajapinnan teknologiaa valittaessa tuli ottaa huomioon järjestelmän tekemisessä käytettävä tietokanta (MongoDB). Rajapinnan kautta tehdään tietokantahaut ja lähetetään lomakkeen datan pohjalta muodostettu tarjouspyyntö yrittäjän sähköpostiin. Backend-kielen tulisi siis olla kevyt ja helposti käytettävissä JSON-muotoisen tietokantadatan kanssa. Vähäisten tarvittavien vaatimusten takia Javascriptiä käyttävä Node.js valittiin rajapintaratkaisuksi, sillä se on nopea ja sen avulla tietokannasta tulleen JSON-muotoisen datan käsittely on äärimmäisen helppoa.

4.2 Projektin luominen ja konfigurointi

React-kehitysalusta laitettiin pystyyn CRA (create-react-app) -npm-paketin avulla. CRA sisältää itsessään valmiit konfiguraatiot Babel-kääntäjän käyttämiseen ES6-syntaksin kanssa. Siihen on lisäksi configuroitu valmiiksi Webpack-paketointityökalu, joka optimoi selainohjelman muodostamalla kaikista projektin Javascript- ja CSS-tiedostoista tiiviitä paketteja. React-projekti luotiin CRA-paketin avulla kutsumalla npm create-react-app rfo-form, joka luo kansioon rfo-form käyttövalmiin koodipohjan ja package.json-tiedoston React-sovellukseen. Sovellus käynnistetään kutsumalla npm start. CRA-paketin luoman koodipohjan myötä sovellukset, jotka eivät vaadi erityisiä konfiguraatioita, voidaan saada käyntiin hyvin nopeasti ja vaivattomasti. Jotta kirjoitettu Javascript-koodi olisi mahdollisimman helppolukuista ja selkeää, otettiin projektissa käyttöön linter-työkalu, jonka avulla voidaan valvoa koodin laatua. Linter-työkalut sisältävät listan sääntöjä koodin tyylittelystä. Linter on erityisen hyödyllinen, kun koodia työstää useammat ihmiset, jotka saattaisivat ilman tyylisääntöjä kirjoittaa hyvin erinäköistä koodia. Sovellukseen valittiin käytettäväksi ESLint linter-työkalu, jonka noudattaa melko tiukkaa sääntökirjastoa. ESLint asennettiin

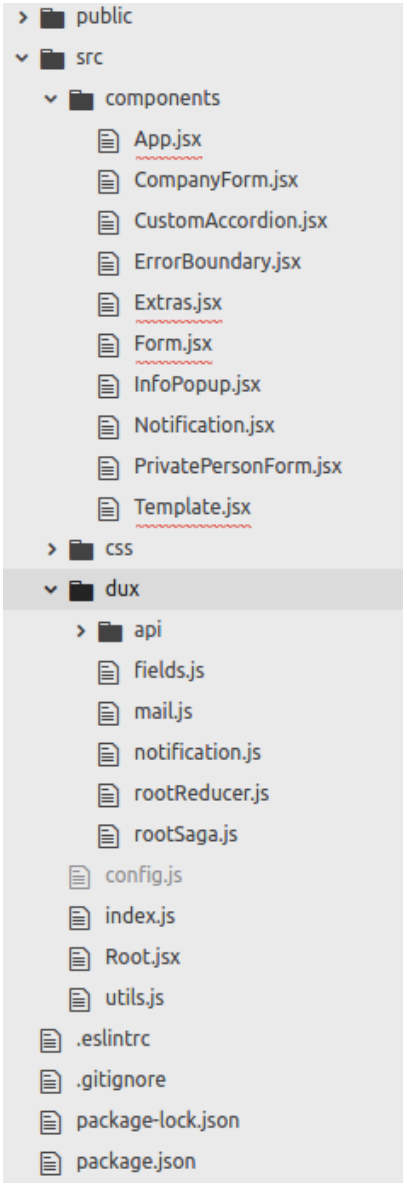
erikseen käyttöjärjestelmään npm-työkalun avulla ja jotta React-sovellus saa käyttöönsä tyylisäännöt, projektiin luotiin .eslintrc -konfiguraatiotiedosto. Kuvassa 15 on esimerkki React-projektin ESLint-konfiguraatiosta, jossa ESLintille kerrotaan mitä sääntökirjastoa sen tulee hyödyntää (airbnb). Lisäksi sääntöjen oletusarvoja voi muuttaa tai ottaa epätoivottuja sääntöjä kokonaan pois käytöstä, esimerkiksi max-len -säännön poistamalla koodissa voidaan kirjoittaa yli 100 merkin pituisia rivejä, ilman että ESLint ilmoittaa tyylivirheestä. (ESLint; Facebook 2018.)

```
{
  "extends": ["react-app", "airbnb"],
  "globals": {
    "document": true,
    "window": true
  },
  "rules": {
    "jsx-a11y/href-no-hash": "off",
    "no-underscore-dangle": "off",
    "react/forbid-prop-types": 0,
    "max-len": "off",
    "react/sort-comp": "off"
    }
}
```

Kuva 15. ESLint-konfiguraatio.

Projektin rajapinnan Node-sovellus käynnistettiin asentamalla npm-komennolla tarvittavat paketit projektin kansion alle. Rajapinnassa hyödynnettiin Noden express-sovelluskehystä, jotta projekti saatiin käynnistettyä paikallisessa palvelimessa. Express asennettiin komennolla npm install express, jonka jälkeen luotiin server-tiedosto. Server-tiedostossa luodaan express-objekti ja kerrotaan, missä portissa ohjelman tulee pyöriä. Server-tiedostoon kirjoitettiin express().listen(3000) ja ohjelma käynnistettiin kutsumalla node server.js, jolloin Node kutsuu server-tiedostossa olevaa express-objektia ja käynnistää ohjelman porttiin 3000. (Expressjs 2017.)

4.3 Käyttöliittymän rakenne



Kuva 16. Käyttöliittymän tiedostorakenne

React-sovelluksen rakenne jaettiin alakansioihin: public, components, css ja dux. Kuva 16 havainnollistaa rakennetta. Public sisältää html-sivun, johon React-sovellus renderöidään. Components-kansiossa ovat kaikki varauslomakkeen hyödyntämät React-komponentit. Komponentit pyrittiin jakamaan mahdollisimman pieniin loogisiin osiin, jotta komponenttien tilan hallinta olisi selkeää. App-komponentti toimiii ns.yläkomponenttina, joka palauttaa varsinaisen varauslomakkeen (Form), virheenkäsittelykomponentin (ErrorBoundary) ja notifikaatiokomponentin (Notification). App-komponentti yhdistetään Reduxin tilaan, ja se antaa propseina palauttamilleen komponenteille tarvittavat funktiot ja muuttujat, jotta muita komponentteja ei tarvitse erikseen yhdistää Reduxiin. Komponentissa kutsutaan lisäksi Reduxin avulla sovelluksen rajapintaa, jotta saadaan tietokannasta käyttöliittymäelementtien suomen-ja englanninkieliset tiedot komponenttien käyttöön.

Form-komponentti sisältää sovelluksen varauslomakkeen. Lomake on vielä jaettu alakomponentteihin yritys- ja yksityisasiakkaan valintojen ja lisäpalveluiden mukaan. Lomakkeen ajanvarauskalenteria varten Form-komponentti käyttää ulkoista npm-pakettia. Reactille tehtyjä kalenteri-komponentteja löytyy paljon ja sopivan komponentin valitseminen tässä tapauksessa toteutettiin hakukoneella löydettyjen suosittujen komponenttien dokumentaatiota vertailemalla. Kaikki komponentit eivät tarjonneet ajanvarauskalenterissa tarvittavia ominaisuuksia, joten ne karsittiin tämän perusteella. Vaihtoehdoiksi jäivät react-dates ja react-day-picker. Ominaisuuksiltaan molemmat komponentit vaikuttivat lupaavilta, mutta react-dates-npm-paketti on riippuvainen 13 muusta paketista, jotka täytyisi myös asentaa projektiin ja projektin koko kasvaisi. Paketeille ei olisi myöskään mitään muuta tarvetta projektissa, joten kalenteri-komponentiksi valittiin mielummin react-day-picker, joka oli riippuvainen vain yhdestä projektissa käytössä olevasta paketista. (React-dates 2018; React-day-picker 2018.)

Sovelluksen App-komponentin palauttama ErrorBoundary-komponentti hyödyntää Reactin 16 -version myötä tullutta elinkaarimetodia componentDidCatch, jonka avulla voidaan näyttää käyttäjälle virheilmoitus, jos komponenttia ei kyetty jonkin virhetilan takia renderöimään. Ennen React 16 -versiota käyttäjä näki virhetilanteessa vain tyhjän ruudun, nyt selaimella voidaan sen sijaan näyttää käyttäjäystävällinen virheilmoitus. Notification-komponentin avulla käyttäjälle näytetään ruudulla ilmoitus, kun varauslomake on lähetetty yrittäjän sähköpostiin. Komponentti kommunikoi Reduxin kanssa tietääkseen milloin ja minkä tyyppinen (onnistunut, virhe) ilmoitus näytetään käyttäjälle. (Abramov 2017.)

Css-kansio sisältää sovelluksen käyttämät css-tyylitiedostot. Dux-kansiossa ovat tiedostot Reduxin tilan hallinnoimiseen, sekä alakansio api, jossa ovat Reduxin tekemät kutsut sovelluksen rajapintaan. Sovelluksessa päätettiin käyttää Reduxia, jotta tilan hallinta olisi mahdollisimman selkeää ja vaivatonta. Jos sovellusta jatkokehitetään myöhemmin ja rajapintapyyntöjä tulee lisää, sovelluksen rakenne tukee jo valmiiksi globaalin tilan hallintaa. Sovellus hyödyntää Saga-väliohjelmistoa rajapintapyyntöjen tekemiseen Reduxista. Reduxiin määriteltiin mitä komponentilta tulevia toimintoja Sagan tulee kuunnella. Kun tiettyä toimintoa kutsutaan, Saga kutsuu ns. Saga worker -funktiota, jossa tehdään rajapintapyyntö ja sen virheenkäsittely. (Redux-Saga.)

```
function* fetchFieldsWorker() {
  try {
    const response = yield call(fetch);
    if (response.status === 200) {
      yield put(fetchedFields(response.data));
    } else {
      // error handling
    }
  } catch (e) {
    // more error handling
  }
}

export const fieldSagas = [
  takeLatest(FETCH_FIELDS, fetchFieldsWorker),
];
```

Kuva 17. Sagan worker-funktio ja yhdistäminen toimintoon

Sovelluksessa Saga kuuntelee kuvassa 17 esitetyn fieldSagas-muuttujan saamaa FETCH_FIELDS -toimintoa ja kun App-komponentissa kutsutaan Reduxin toimintoon yhdistettyä funktiota this.props.fetchFields(), Saga kutsuu fetchFieldsWorker-funktiota. Yield call -funktio saa parametrinä rajapinnan kutsufunktion ja palauttaa rajapinnalta saadun vastauksen. Jos pyyntö onnistui (response.status === 200), kutsutaan fetchedFields-toimintoa, joka päivittää Reduxin reducer-funktion avulla rajapintapyynnön vastauksen sisällä olevan datan säiliöön. Muussa tapauksessa suoritetaan virheenkäsittely ja voidaan esimerkiksi näyttää käyttäjälle virheilmoitus. Saga tulee vielä yhdistää Reduxin säiliöön, jotta sen worker-funktio aktivoituu toimintoa kutsuttaessa. Kuvan 18 mukaisesti Saga yhdistetään säiliöön applyMiddleware-funktion avulla, joka kertoo säiliölle, että sen tulee käyttää Saga-väliohjelmistoa. (Redux-Saga.)

```
const sagaMiddleware = createSagaMiddleware();
const store = createStore(
  rootReducer,
  applyMiddleware(sagaMiddleware),
);
```

Kuva 18. Sagan yhdistäminen säiliöön

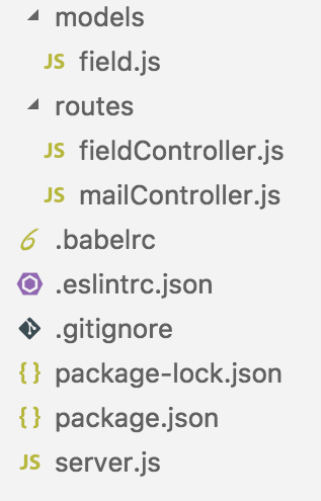
4.4 Integraatio Google Kalenteriin

Varauslomakkeen ajanvarauskalenterin vapaiden aikojen näyttämiseksi tarvittiin integraatio asiakkaan käyttämään Google Kalenteriin. Googlella on oma kalenterirajapinta, jonka avulla voidaan tehdä pyyntöjä kalenteriin. Jotta rajapinta voitiin ottaa käyttöön, se tuotiin React-sovelluksen public-kansion html-tiedostoon skriptinä: <script

src=https://apis.google.com/js/api.js />. Kalenteritietojen saamiseksi rajapinnan kautta projekti tuli vielä lisätä Googlen rajapintapalveluun, josta projektille generoitiin identifioiva Google API-avain. (Google 2018a.)

Rajapintaa kutsutaan Form-komponentissa pyynnöllä https://www.googleapis.com/calendar/v3/calendars/<CALENDAR_ID>/events, jonka parametreinä ovat aiemmin luotu API-avain sekä kalenteritapahtumien aikaisin alkupäivämäärä ja viimeisin loppupäivämäärä. Pyyntö palauttaa tapahtumat listamuotoisena päivämäärineen ja kellonaikoineen. Tapahtumien päivämääriä ja kellonaikoja vertailemalla ajanvarauskalenteriin muodostetaan tieto vapaista ja varatuista päivistä. (Google 2018b.)

4.5 Rajapinnan rakenne



Kuva 19. Rajapinnan tiedostorakenne

Rajapinnan Node-sovelluksen kansiorakenne on esitetty kuvassa 19. Sovelluksen juuressa ovat konfiguraatiotiedostot kuten package.json ja server.js. Sen lisäksi sovellus sisältää kansion tietokantamalleille (models) ja reiteille (routes), joiden avulla React-sovellus voi kutsua rajapintaa. Server-tiedostossa määritellään express-objekti sovelluksen käynnistämistä varten ja yhteys rajapinnan kontrollereihin ja tietokantaan. Tietokantayhteyden muodostamiseksi käytetään mongoose-pakettia, joka on Node.js:n kanssa käytettävä MongoDB-työkalu. Mongoosen avulla luodaan models-kansion alla oleva tietokantamalli fields-tietokantakokoelmasta. Mallin luominen ei ole pakollista MongoDB:tä käytettäessä, mutta kun kannan dokumenteille määritellään sallitut tietokentät ja näiden tietotyypit, tietokannan data pysyy eheänä ja yhtenäisenä. Mallissa määritellään tietokentät fields-kokoelmalle, ja mallia päivitetään aina järjestelmän vaatimusten tarkentuessa tai muuttuessa. (Mongoose.)

Kansion routes alla olevien kontrollereiden kautta rajapintaan voidaan ottaa yhteys React-sovelluksesta. Server-tiedostossa kutsutaan express().use('/fields', fieldController), joka yhdistää sovelluksen osoitteen /fields fieldController-tiedostoon. Tämän kontrollerin sisällä on GET-funktio, jonka avulla haetaan tietokannasta fields-kokoelman tiedot käyttöliittymälle. Kuvassa 18 on esitetty kontrollerissa määritelty router-objektin GET-funktio, joka vastaa osoitetta /fields. Router luodaan express-objektin avulla ja sen avulla sovellus pystyy reitittämään pyynnöt eri osoitteisiin. Field-objekti on Mongoosen avulla tehty tietokantamalli, jota kutsumalla saadaan tietokannasta mallia vastaavat dokumentit. Jos haku onnistuu, palautetaan JSON-muodossa fields-niminen lista tietokantadokumenteista. (Expressjs 2017.)

```
const router = express.Router();

router.get('/', (req, res) => {
  Field.find({}).exec((err, fields) => {
    if (err) throw err;
    res.json(fields);
  });
});

export default router;
```

Kuva 18. GET-funktio kontrollerissa

4.6 Sähköpostin lähettäminen

Kun käyttäjä on täyttänyt varauslomakkeen kentät ja painaa Lähetä-nappia, luodaan html-koodina tarjouspyyntö lähetettäväksi yrittäjän sähköpostiin. Jotta sähköposti olisi hyvin jäsennelty ja helposti luettavissa, haluttiin ottaa käyttöön muutamia html-elementtejä ja tyylejä. Tähän otettiin avuksi npm-paketti react-html-email, jonka avulla voidaan generoida React-komponenteista yksinkertainen taulukkomuotoinen html-koodi. Muita vastaavia paketteja ei löytynyt Reactille. Varauslomakkeen täytettyjen kenttien tiedot haetaan komponentin tilasta ja kutsutaan Template-komponentin funktiota createHTML(). Funktion sisällä kutsutaan react-html-emailin omaa renderEmail-funktiota, joka palauttaa html-koodipohjan. Kuva 19 esittää sähköpostin html-koodin luomista react-html-emailin komponenttien (Email, Item, Span) avulla. Komponenteille voi lisätä css-tyylejä, mutta eri sähköpostiasiakasohjelmat tukevat hieman vaihtelevasti eri css-elementtejä. CreateHTML-funktion palauttama tekstimuotoinen html-koodi annetaan parametrinä Reduxin toiminnolle, joka kutsuu rajapintaa. Rajapinta vastaanottaa html-koodin ja varauslomakkeeseen ilmoitetun sähköpostiosoitteen, johon halutaan vastaus tarjouspyynnön käsittelystä. Rajapinta hyödyntää nodemailer-pakettia sähköpostin lähettämiseen. Nodemailer ei ole riippuvainen

mistään muista npm-paketeista, kuten monet muut npm:stä löytyvät vastaavat sähköpostin lähetykseen tarkoitetut paketit Nodelle. Sähköpostin lähettämistä varten nodemailer tarvitsee transporter-objektin, jolle määritellään yhteyteen vaadittavat tiedot. (Chromakode 2017; Nodemailer a.)

```
function createHTML(data, description) {
  const header = { paddingTop: '15px' };

  return renderEmail(
    <Email title="Tarjouspyyntö" align="left">
      {description && <Item>{description}</Item>}
      { Object.keys(data).map(innerObject =>
        (
          <div>
            <Item style={header}><Span fontSize={17}>{data[innerObject].title}</Span></Item>
            { Object.keys(data[innerObject]).map(key =>
              (
                key !== 'title' && data[innerObject][key] &&
                <Item>…
              ))}
          </div>)
      )}
    </Email>);
}
```

Kuva 19. Sähköpostipohjan luominen React-komponenttien avulla

Nodemailerin transporter-objekti luodaan rajapinnassa, mitä kuva 20 havainnollistaa. Nodemailerilla on tiedot yhteyden muodostamisesta SMTP-protokollan avulla moniin yleisiin sähköpostin tarjoajiin (kuten gmail), joten tällöin riittää vain antaa tarjoajan nimi service-attribuutissa. Sen lisäksi nodemailer tarvitsee autentikointitiedot sähköpostin lähettävältä osapuolelta auth-objektissa. Sähköposti lähetetään kutsumalla transporter.sendMail(options), jossa options-objekti sisältää itse sähköpostin sisällön ja sen vastaanottajan. (Nodemailer b.)

```
const transporter = nodemailer.createTransport({
  service: 'gmail',
  secure: true,
  auth: {
    user: SENDER,
    pass: SENDER_PW
  }
});
```

Kuva 20. transporter-objektin luominen

4.7 Version- ja työtehtävien hallinta

Projektin versionhallintaan käytettiin julkista Githubia-tiliä. Projekti jaettiin kahteen repositorioon: rfo-form ja rfo-form-backend. Projektin jakaminen erillisiin repositorioihin tuntui

loogiselta, koska varauslomakkeen käyttöliittymä ja rajapinta muodostivat omat erilliset Javascript-projektinsa, jotka vain kommunikoivat toisilleen rajapinnan kautta. Koska lähdekoodia oli melko vähän ja sitä työsti vain yksi ihminen, repositorioihin luotiin aluksi vain master-haarat. Uusia ominaisuuksia lisättäessä luotiin kullekin ominaisuudelle haara feature/<ominaisuuden-nimi>, joka poistettiin, kun ominaisuus valmistui ja se voitiin yhdistää master-haaraan.

Projektin edetessä yrittäjä esitti pieniä tarkennuksia varauslomakkeen toiminnallisuuksiin, jotka kirjattiin ylös ja niistä luotiin projektin varauslomakkeen repositorioon issueita. Kullekin issuelle voi antaa sitä kuvaavan nimikkeen, esimerkiksi bugi, parannus tai ominaisuus. Projektin ollessa testausvaiheessa varauslomakkeen testaajat pystyivät lisäämään uusia issueita ja kommentoimaan jo olemassa oleviin issueihin. Kun esimerkiksi bugi-nimikkeellä varustettu issue saadaan korjattua, voidaan kommentoida mihin ratkaisuun päädyttiin ja miksi. Näin voidaan välttää vastaavan kaltaisia ongelmia tulevaisuudessa. Vaikka projektin versionhallinta ja edistymisen seuraaminen onkin yhden kehittäjän työstäessä projektia vielä melko yksinkertaista, dokumentointia ongelmista kannattaa tehdä jo alusta alkaen huolella. Uuden kehittäjän on huomattavasti helpompi aloittaa projektin jatkokehitys kun hän näkee mitä on tähän mennessä tehty versionhallinnan työkalun kautta.

4.8 Virtualisointi

Koko järjestelmä haluttiin virtualisoida, jottei sen toimivuus riippuisi millään tavalla käyttöjärjestelmästä ja siirto palvelimelta toiselle helpottuisi. Erilaisia virtualisointimenetelmiä on tarjolla suuri määrä ja niiden tarjoamat ratkaisut virtualisointiin vaihtelevat. Kontteihin perustuvista virtualisointijärjestelmistä Kubernetes ja Docker ovat suosion kärjessä. Kubernet tarjoaa monipuoliset työkalut virtualisointiin, mutta virtuaaliympäristön luominen eri käyttöjärjestelmille vaatii omat erilliset asetuksensa. Dockerissa luotu virtuaaliympäristö taas on käytettävissä kaikissa käyttöjärjestelmissä ja muutosten tekeminen ympäristöön onnistuu helposti. Varausjärjestelmän virtualisointiin haluttiin mahdollisimman kevyt ja yksinkertainen ratkaisu, jota olisi myös helppo muokata tarpeen vaatiessa. Vaatimuksiin nähden Docker sopi siis paremmin virtualisointimenetelmäksi. (Scott 2018.)

```
FROM node:latest

WORKDIR /usr/src

COPY package.json /usr/src/package.json
RUN npm install
RUN npm install react-scripts@0.9.5 -g

CMD ["npm", "start"]
```

Kuva 21. React-sovelluksen Docker-tiedosto

Varausjärjestelmän React- ja Node-sovelluksille luotiin ensin omat Docker-tiedostot, joiden avulla Docker-kuvien luominen automatisoitiin. Kuvassa 21 on esimerkki varausjärjestelmän React-sovelluksen Docker-tiedostosta, joka koostuu sarjasta komentoja. Docker-kuvan pohjana käytetään tässä tapauksessa valmista node:latest-kuvaa ja kuvalle kerrotaan tiedostossa mitä komentoja sen tulisi ajaa. Tiedoston viimeinen rivi (CMD [”npm”, ”start”]) sisältää komennon React-sovelluksen käynnistämiseen virtuaalikontissa. Varausjärjestelmän virtualisoimiseen tarvittiin omat virtuaalikontit React- ja Node-sovelluksille sekä MongoDB-tietokannalle, joten virtualisointiympäristön hallinnoimiseen otettiin apuvälineeksi Docker Compose. Compose on työkalu, jonka avulla voidaan määritellä ja käynnistää useita kontteja yhden komennon avulla. Kuva 22 on esimerkki docker-compose -tiedostosta, jossa määritellään kunkin kontin tiedot. Build-arvo kertoo kansiopolun Docker-kuvalle, jonka pohjalta kontti luodaan. Volumes-arvo määrittää mikä kansiopolku palvelimella liitetään kontin polkuun. Kun React-sovelluksen tiedostoihin tehdään muutoksia palvelimella, ne päivittyvät liitoksen myötä myös kontin sisällä olevaan sovellukseen. Ports-arvon avulla Docker tietää mihin porttiin kontin sovellus yhdistetään. Virtuaaliympäristö ajetaan kutsumalla docker-compose up -d, joka käynnistää kaikki docker-compose -tiedoston sisällä olevat kontit. (Docker 2018b; Docker 2018c.)

```yaml
version: "3"
services:

  rfo-form-backend:
    container_name: rfo-form-backend
    build: ./server/
    volumes:
      - './server:/usr/src/'
      - './server/package.json:/usr/src/package.json'
    ports:
      - '3001:3001'
    links:
      - mongodb

  rfo-form-client:
    container_name: rfo-form-client
    build: ./client/
    volumes:
      - './client:/usr/src/'
      - './client/package.json:/usr/src/package.json'
    ports:
      - '3000:3000'
```

Kuva 22. Docker-compose -tiedosto

5 Pohdinta

Opinnäytetyön tavoitteena oli tuottaa yritykselle varausjärjestelmä, joka helpottaa yrittäjän ja asiakkaan kohtaamisprosessia. Työn lähtökohtana oli Laurea ammattikorkeakoulun opiskelijoiden tekemä malli käyttöliittymästä, joka oli luotu yrityksille teetettyjen haastattelujen pohjalta. Opinnäytetyön toteutus alkoi järjestelmän vaatimusten määrittelystä käyttöliittymämallin pohjalta. Siitä edettiin lopullisen käyttöliittymän ja tietokantarakenteen suunnitteluun. Järjestelmän selaimella toimiva varauslomake toteutettiin React-teknologiaa hyödyntäen ja rajapinnan teknologiaksi valittiin Node.js. Järjestelmän tietokantana toimi MongoDB.

Opinnäytetyön tavoite saavutettiin, sillä varausjärjestelmä saatettiin julkaisemisvaiheeseen opinnäytetyön puitteissa. Toimeksiantaja oli tyytyväinen järjestelmän visuaaliseen ilmeeseen ja toiminnallisuuksiin. Sain jatkuvasti rakentavaa palautetta työstä, mikä kannusti minua jatkamaan työskentelyä hyvässä tahdissa. Oppimisprosessina opinnäytetyö oli minulle ylipäätään hyvin antoisa ja opettavainen. Laurean opiskelijoiden tekemä käyttöliittymämalli antoi hyvän pohjan järjestelmän suunnittelulle ja vaatimusten määrittelemiselle. Kokonaisen järjestelmän tuottaminen yksin tuntui joka tapauksessa alkuun suurelta haasteelta, mutta jakaessani työmäärän pieneen osiin, prosessi eteni melko sujuvasti ja vältin suurimmat riskit, jotka olisivat voineet vaikuttaa työn edistymiseen. Varasin työn tekemiseen myös riittävästi aikaa, jotta sain tehdä jokaisen työvaiheen huolella loppuun.

5.1 Haasteita

Työn edetessä vastaan tuli myös erilaisia haasteita. Suunnitteluvaiheen alkaessa järjestelmälle oli määritelty vaatimukset, jotka tulisi täyttyä. Vaatimuksiin tuli työn edetessä jonkin verran muutosehdotuksia, ja ohjelmakoodia piti muuttaa useamman kerran uusien vaatimusten täyttämiseksi. Onneksi olin suunnitellut järjestelmän huolella, jolloin muutosten tekeminen ei vienyt liikaa aikaa. Yrittäjä oli myös toivonut että järjestelmä saataisiin mahdollisimman nopeasti testaukseen ja käyttöön. Tämä asetti minulle jonkin verran paineita työn nopealle edistymiselle. Järjestelmän julkaisu päätettiin kuitenkin jättää opinnäytetyön tavoitteista pois.

Tietokannan rakenne ja sen suunnittelu toi oman haasteensa työhön. Kannan käyttäminen järjestelmässä ei ollut pakollista, sillä kannasta ainoastaan haetaan dataa, mutta mitään dataa ei tallenneta tai poisteta sieltä käyttöliittymän kautta. Näin ollen dataa voisi säilyttää myös esimerkiksi tekstitiedostossa. Koin kuitenkin pitkällisen pohdinnan jälkeen parhaimmaksi ratkaisuksi hyödyntää tietokantaa datan säilyttämisessä, jotta yrittäjä pystyy

itse helposti muuttamaan kannassa säilytettävää dataa tietokannan tarjoaman visuaalisen työkalun avulla. Tätä ei oltu asetettu vaatimukseksi, mutta myös yrittäjä piti ajatusta hyvänä.

5.2 Jatkokehitysideoita

Järjestelmän kehittäminen on opinnäytetyön osalta saatettu päätökseen, mutta järjestelmän parantaminen jatkuu mahdollisesti myöhemmin. Tällä hetkellä varauslomake toimii hyvin selaimella, mutta mobiililaitteella hieman puutteellisesti, muun muassa ajanvarauskalenteri ei näy kokonaisuudessaan pienillä näytöillä. Yksi jatkokehityskohteista järjestelmän osalle olisikin varauslomakkeen optimointi mobiilille, jotta asiakkaat pystyisivät lähettämään tarjouspyynnön yrittäjälle päätelaitteesta riippumatta. Yrittäjä ilmaisi myös toiveen, että varauslomakkeen ajanvarauskalenteria käytettäisiin myös ilman varauslomaketta yrityksen nettisivuilla. Kalenteri voitaisiin laittaa esille yrityksen etusivuille, jotta sivuston vierailija näkee heti ensimmäiseksi tämänhetkisen varaustilanteen. Kalenteriin joudutaan kuitenkin tekemään jonkin verran muutoksia, jos sitä halutaan käyttää varauslomakkeesta irrallisena elementtinä.

Järjestelmän nykytilassa yrittäjä käsittelee varauksen loppuun asiakkaan kanssa sähköpostin välityksellä ja tekee vahvistuksen varaukseen. Hyväksytty varaus tallennetaan yrittäjän Google kalenteriin, mikä on kyllä toiminut tähän saakka, mutta prosessia voisi parantaa. Varausten käsittelyyn voitaisiin luoda oma käyttöliittymänsä, johon yrittäjällä ja muilla varauksia käsittelevillä henkilöillä olisi pääsy. Asiakkaan tiedot tallennettaisiin tähän järjestelmään ja tarpeen tullen varausta voisi muokata tai sen voisi poistaa kokonaan peruuntumisen sattuessa. Käyttöliittymä voisi pitää sisällään myös oman kalenterin, joka näyttää kaikki tulevat varaukset.

Lähteet

Abramov, D. 2017. Error Handling in React 16. Luettavissa: https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html . Luettu: 29.7.2018.

Ant Design. Ant Design of React. Luettavissa: https://ant.design/docs/react/introduce. Luettu: 3.8.2018.

Buckler, B. 2015. SQL vs NoSQL: The Differences. Luettavissa: https://www.sitepoint.com/sql-vs-nosql-differences/. Luettu: 27.7.2018.

Docker 2018a. Docker overview. Luettavissa: https://docs.docker.com/engine/docker-overview/. Luettu: 12.6.2018.

Docker 2018b. Dockerfile reference. Luettavissa: https://docs.docker.com/engine/reference/builder/#usage . Luettu: 12.9.2018.

Docker 2018c. Overview of Docker Compose. Luettavissa: https://docs.docker.com/compose/overview/ . Luettu: 14.9.2018.

ESLint. Configuring ESLint. Luettavissa: https://eslint.org/docs/user-guide/configuring. Luettu: 15.8.2018.

Expressjs 2017. 4.x API. Luettavissa: https://expressjs.com/en/4x/api.html#express. Luettu: 27.8.2018.

Facebook 2018. Create-react-app. Luettavissa: https://github.com/facebook/create-react-app . Luettu: 17.8.2018.

Gagliardi, V. 2018. React Redux Tutorial for Beginners: The Definitive Guide (2018). Luettavissa: https://www.valentinog.com/blog/react-redux-tutorial-beginners/. Luettu: 11.6.2018.

Google 2018a. Browser Quickstart. Luettavissa: https://developers.google.com/calendar/quickstart/js . Luettu: 22.8.2018.

Google 2018b. Events:list. Luettavissa: https://developers.google.com/calendar/v3/reference/events/list . Luettu: 23.8.2018.

Kurian, G. 2017. How Virtual-DOM and diffing works in React. Luettavissa: https://medium.com/@gethylgeorge/how-virtual-dom-and-diffing-works-in-react-6fc805f9f84e . Luettu: 9.6.2018.

Madein, T. 2017. What's the Deal with JSX. Luettavissa: https://hackernoon.com/whats-the-deal-with-jsx-9ab2f862bf2b . Luettu: 9.6.2018.

Material-UI. Component API. Luettavissa: https://material-ui.com/. Luettu: 3.8.2018.

Mongoose. Schemas. Luettavissa: https://mongoosejs.com/docs/guide.html . Luettu: 8.8.2018.

Npm 2018a. What is npm?. Luettavissa: https://docs.npmjs.com/getting-started/what-is-npm. Luettu: 11.6.2018.

Npm 2018b. npm. Luettavissa: https://docs.npmjs.com/cli/npm . Luettu: 11.6.2018.

Npm trends 2018. angular vs react vs vue vs backbone vs ember-cli. Luettavissa: http://www.npmtrends.com/angular-vs-react-vs-vue-vs-backbone-vs-ember-cli. Luettu: 06.7.2018.

Nodemailer a. Nodemailer. Luettavissa: https://nodemailer.com/about/#nodemailer-features. Luettu: 28.8.2018.

Nodemailer b. SMTP Transport. Luettavissa: https://nodemailer.com/smtp/ . Luettu: 29.8.2018.

React 2018a. Components and Props. Luettavissa: https://reactjs.org/docs/components-and-props.html . Luettu: 9.6.2018.

React 2018b. State and Lifecycle. Luettavissa: https://reactjs.org/docs/state-and-lifecycle.html . Luettu: 9.6.2018.

React 2018c. React.Component. Luettavissa: https://reactjs.org/docs/react-component.html . Luettu: 9.6.2018.

React-dates 2018. React-dates. Luettavissa: https://github.com/airbnb/react-dates. Luettu 18.8.2018.

React-day-picker 2018. React-day-picker. Luettavissa: http://react-day-picker.js.org/. Luettu: 18.8.2018.

Chromakode 2017. React-html-email. Luettavissa: https://github.com/chromakode/react-html-email. Luettu: 26.8.2018.

Redux-Saga. Beginner Tutorial. Luettavissa: https://redux-saga.js.org/docs/introduction/BeginnerTutorial.html. Luettu: 12.6.2018.

Scott, T. 2018. Kubernetes vs. Docker: Comparing Containerization Systems. Luettavissa: https://technologyadvice.com/blog/information-technology/kubernetes-vs-docker/ . Luettu: 7.9.2018.

Semantic UI React. Elements. Luettavissa: https://react.semantic-ui.com/. Luettu: 3.8.2018.

Sviatoslav A. The Best JS Frameworks for Front End. Luettavissa: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end . Luettu: 08.7.2018.

Tutorialspoint 2018a. Node.js - Introduction. Luettavissa: https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm. Luettu: 11.6.2018.

Tutorialspoint 2018b. ES6 - Overview. Luettavissa: https://www.tutorialspoint.com/es6/es6_quick_guide.htm. Luettu: 02.8.2018.

Tutorialspoint 2018c. MongoDB – Quick Guide. Luettavissa: https://www.tutorialspoint.com/mongodb/mongodb_quick_guide.htm . Luettu: 13.7.2018.

Wodehouse 2015. What Is GitHub and Why Should Your Digital Team Use It? Luettavissa: https://www.upwork.com/hiring/development/what-is-github-and-why-should-your-digital-team-use-it/. Luettu: 11.6.2018.

Liitteet

Liite 1. Varauslomakkeen käyttöliittymämalli

Varaa Villa Paratiisi

Villa Paratiisin varaaminen onnistuu verkossa. Täytä alla olevaan lomakkeeseen toiveesi tapahtumasta. Lähetämme varausvahvistuksen ja mahdolliset pyynnöt lisätiedoista antamaasi sähköpostiosoitteeseen vuorokauden kuluessa.

Nimi*

Sähköpostiosoite*

Puhelin

Asiakastyyppi

○ Kokousasiakas ○ Yksityisasiakas

Henkilömäärä

Tulopäivä **Tuloaika** **Lähtöpäivä** **Lähtöaika**

Kokouksen tyyppi

○ Päiväkokous 8 h ⓘ

○ Kokouspäivä ja saunailta 12 h ⓘ

○ Kokouspäivät majoituksella 32 h ⓘ

○ Lounaasta lounaaseen -kokouspäivä majoituksella 24 h ⓘ

8 tuntia, hinta 580€+alv%
Sisältää kokouspalvelut ja päivän lomassa opastetun tunnin retken metsään tai vesille.
Tarjoilut (aamiainen, lounas, kahvit) alkaen 35€/henk.+alv%

Mitä palveluja tarvitset?

☐ Neuvottelutilat (sis. kokoustekniikan)

Ruokailu:

☐ Aamiainen

☐ Lounas (menu ⓘ)

☐ Nokipannukahvit (kokouskahvit sis. hintaan)

☐ Illallinen (menu ⓘ)

☐ Iltapala

Voit valita kolmesta vaihtoehdosta:
Maahisten hirvimakkaroita ja talon omaa peruna-yrttisalaattia
Ilmattaren kermaista kasvis-, kala- tai riistakeittoa
Katajattaren metsäsieni- ja savuporotäytteisiä uuniperunoita

Mitä aktiviteetteja haluat tilata? (Tunnin opastettu metsäretki sis. hintaan)

☐ Vesiurheilu (melonta, sup-lautailu)

☐ Kalliolaskeutuminen

☐ Tiimikisailu

☐ Kehon hoito (esim. jooga, kahvakuula, yrttikylpy)

☐ Työyhteisökoulutus

☐ Jurttasauna

☐ Elävää musiikkia

☐ Husky-retki kickbikella

VARAA