Mikko Manninen

Ajanvarausjärjestelmän suunnittelu ja toteutus React-kirjastoa ja Firebase-tietokantaa hyödyntäen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

24.10.2024

Tiivistelmä

Tekijä:	Mikko Manninen
Otsikko:	Ajanvarausjärjestelmän suunnittelu ja toteutus React-kirjastoa ja Firebase-tietokantaa hyödyntäen
Sivumäärä:	23 sivua
Aika:	24.10.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistosuunnittelu
Ohjaajat:	Janne Salonen

Tämän insinöörityön tarkoituksena oli tehdä paikallisessa verkossa toimiva ajanvarausjärjestelmä React.js -kehyksellä ja Firebase-tietokannalla. Aikataulullisista syistä johtuen työ tehtiin MVP-periaatteella. Projektissa hyödynnettiin ketterän kehityksen story mapping – ja kanban-menetelmää.

Web-projektin toteutuksessa kerrotaan front end -ohjelmoinnista ulkosasusuunnittelijan tekemien Figma-luonnosten mukaan, back end -puolen käyttöönotosta Firebase-palvelun menetelmillä sekä näiden kahden integroimisesta toisiinsa.

Insinöörityö sisältää johdannon, teoriaosuuden käytetyistä tekniikoista, toteutusosion web-projektin etenemisestä ja yhteenvedon, jossa käydään läpi mitä tehtiin ja miten työssä onnistuttiin.

Avainsanat: Javascript, React, Firebase, full stack, sovelluskehitys

---

Abstract

Author:	Mikko Manninen
Title:	Planning and implementing a booking system using the React library and Firebase database
Number of Pages:	23 pages
Date:	24 October 2024
Degree:	Bachelor of Engineering
Degree Programme:	Degree programme in Information and Communication Technology
Professional Major:	Software Development
Supervisors:	Janne Salonen

The purpose of this engineering thesis was to create a scheduling system operating in a local network using the React.js framework and Firebase database. Due to time constraints, the work was done following the MVP principle. The project utilized agile development methods, specifically story mapping and Kanban.

The implementation of the web project describes front-end programming based on Figma sketches made by the UI designer, the back-end setup using Firebase services, and the integration of these two components.

The thesis includes an introduction, a theoretical section on the techniques used, an implementation section detailing the progress of the web project, and a conclusion summarizing what was done and how well the project was executed.

Keywords: Javascript, React, Firebase, full stack, web development

Sisällys

Lyhenteet

API: Application programming interface, ohjelmointirajapinta, jonka avulla ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoa keskenään.

BaaS: Back-end as a service, tietokannan ulkoistaminen kolmannen osapuolen tarjoajalle.

DOM: Document Object Model, mahdollistaa HTML-elementtien muokkaamisen dynaamisesti Javascriptillä.

JSON: Javascript object notation, kevyt ja helppolukuinen tapa strukturoida tietoa.

MERN: MongoDB, Express.js, React.js ja Node.js, teknologiapino, jolla luodaan ja ylläpidetään back-end -palvelu tietokantoineen.

MVP: Minimum viable product, tuotteen versio, jossa on riittävästi ominaisuuksia, jotta varhaiset asiakkaat voivat käyttää sitä ja antaa palautetta tulevaa tuotekehitystä varten.

NoSQL: Not only SQL, yleinen nimitys tietokannoille, jotka käyttävät muuta kuin relaatiomallia tiedon esittämiseen.

NPM: Node package manager, oletuspaketinhallintatyökalu JavaScript-ohjelmointikielen ajonaikaiselle ympäristölle Node.js:lle.

SQL: Structured query language, yleinen kyselykieli, jota käytetään relaatiotietokannan hallintaan.

1 Johdanto

Tämä opinnäytetyö käy läpi prosessia, jossa suunnitellaan ja toteutetaan full-stack -web-projekti, jonka avulla voidaan varata yhteisessä tilassa toimivan neuvotteluhuoneen käyttö omiin tarkoituksiin. Olennaisena osana varausjärjestelmää on antaa ylläpitäjille oikeudet tarkastella kaikkien varaajien varaustietoja ja tarpeen tulleen poistaa muiden tekemiä varauksia.

Opinnäytetyön tarkoituksena on tuottaa yllä kuvattu palvelu asiakkaan toiveiden mukaisesti mahdollisimman nopealla aikataululla. Kun tarkoituksena on luoda full-stack -projektin MVP, minimum viable product, avuksi tulee React.js ja Google-yhtiön ylläpitämä Firebase-tietokantapalvelu.

Olennaisena osana web-projektia on ketterän kehityksen story mapping - ja kanban-menetelmät. Story mapping -menetelmän myötä syntyy asiakkaan sekä ulkoasusuunnittelijan kanssa yhteisymmärrys siitä, minkälainen valmis projekti tulee olemaan. Kanban-menetelmä helpottaa suuresti työn etenemisen seurantaa aikataulujen suhteen.

Kuten mainittua, aikataulun rajallisuus supistaa käytettävien teknologioiden vaihtoehtoja huomattavasti. Tämän vuoksi front-end -ratkaisuksi valitsin React.js -kehyksen pitkäaikaisten kokemuksieni vuoksi. Back-end -ratkaisuksi valikoitui Google-yhtiön Firebase-palvelu sen suoraviivaisen ja nopean käyttöönoton ja ylläpidon takia.

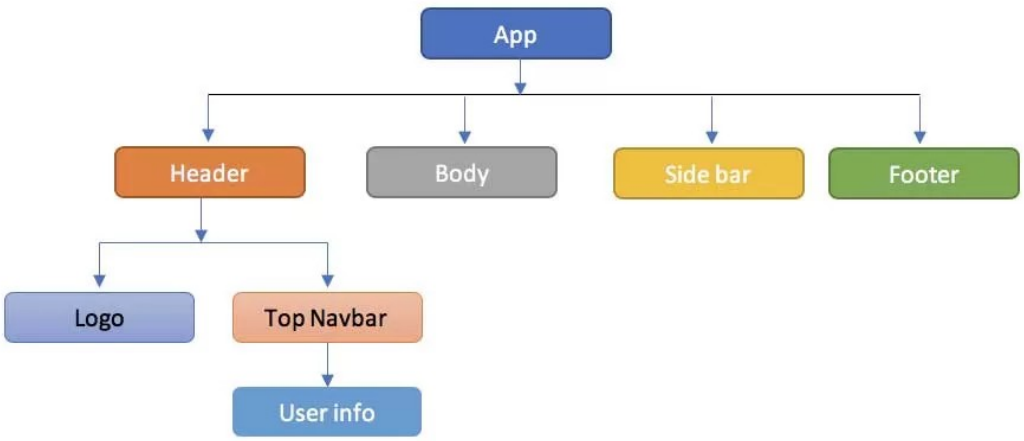
Tämä opinnäytetyö pitää sisällään johdannon, teoriaosuuden käytetyistä tekniikoista, toteutusosion web-projektin etenemisestä ja yhteenvedon, jossa käydään läpi mitä tehtiin ja miten työssä onnistuttiin.

2 React.js-kehys

React.js on laajalti käytössä oleva JavaScript-kirjasto, jota käytetään käyttöliittymien rakentamiseen. Se noudattaa komponenttipohjaista arkkitehtuuria, joka mahdollistaa uudelleenkäytettävien käyttöliittymäkomponenttien kehittämisen ja näiden komponenttien tilan tehokkaan hallinnan. [1.]

React.js-sovellukset rakennetaan käyttäen useita komponentteja. Jokainen komponentti edustaa itsenäistä osaa käyttöliittymässä omine logiikkoineen, merkintöineen ja tyyleineen. Komponentit voidaan jakaa kahteen kategoriaan: funktionaaliset komponentit ja luokkakomponentit. Funktionaaliset komponentit ovat yksinkertaisia JavaScript-funktioita, kun taas luokkakomponentit ovat React.Component-kirjastosta laajennettuja ES6-luokkia. Viimeisten vuosien aikana funktionaaliset komponentit ovat syrjäyttäneet luokkakomponenttien käytön selkeytensä vuoksi. [2.]

Komponentit on järjestetty hierarkkiseen rakenteeseen. Ylintä komponenttia kutsutaan "juuri"-komponentiksi, joka sisältää muita alakomponentteja. Tämä hierarkia muodostaa puun kaltaisen rakenteen, jossa jokaisella komponentilla voi olla useita alakomponentteja. [3.]



Kuva 1. React.js:n hierarkkinen komponenttirakenne

DOM-ohjelmointirajapinnan (Document Object Model) avulla ohjelmointijärjestelmät voivat muuttaa dokumenttien rakennetta, tyyliä ja sisältöä. React.js renderöi ja päivittää komponentteja virtuaalisen DOM:in avulla, joka on kevennetty versio oikeasta DOM-ohjelmointirajapinnasta. Kun komponentin tila muuttuu, React.js vertaa virtuaalista DOM:ia oikeaan DOM:iin ja soveltaa ainoastaan tarvittavat päivitykset oikeaan DOM:iin. [4.]

React.js-sovelluksessa tila (engl. state) edustaa komponentin sisäistä tietoa. Se määrittää, miten komponentti käyttäytyy ja renderöi itsensä. Ominaisuudet (engl. props) ovat syötteitä, jotka välitetään komponentille sen yläkomponentilta. Ominaisuudet ovat muuttumattomia ja niitä käytetään tiedon ja asetusten välittämiseen alakomponenteille. Tieto liikkuu yleisesti ottaen yläkomponentista alakomponenttiin. Jos alakomponentin on päivitettävä yläkomponentin tilaa, se voi tehdä sen kutsumalla takaisinkutsufunktiota, joka on välitetty ominaisuutena yläkomponentista alakomponenttiin. [5.]

Komponenttien väliseen tilanhallintaan käytetään yleisesti joko React.js:n omaa useContext-metodia tai hieman laajempien kokonaisuuksien hallintaan tarkoitettua Redux-kirjastoa. React.js:n useContext-ominaisuus välittää muuttuvan tilan kaikille alakomponenteille kun taas Redux-kirjasto käyttää globaalia tilaa tilanhallinnassa, johon kaikilla komponenteilla on pääsy. [6.]

React Router -reitityskirjasto mahdollistaa siirtymisen eri komponenttien välillä ilman sivujen uudelleenlatausta.

React.js voi vuorovaikuttaa ulkoisten ohjelmointirajapintojen (API) tai backend-palveluiden kanssa hakeakseen tietoja asynkronisesti. Tämä tapahtuu yleensä käyttäen JavaScript-kirjastoja, kuten Axiosia tai React.js:n sisäänrakennettua Fetch API -metodia. React.js:n komponentit voivat tehdä HTTP-pyyntöjä tietojen noutamiseksi ja päivittää sen perusteella tilaansa. [7.]

2.1 Hooks-funktiot

React.js:n versio 16.8 keväällä 2019 toi mukanaan merkittäviä uudistuksia. Näistä mainittavimpia olivat ns. Hooks-funktiot, jotka helpottavat ja yksinkertaistavat komponentin tilanhallinnan ja elinkaaren käsittelyä. [8.]

Ennen React.js:n versiota 16.8 komponentit kirjoitettiin pääasiassa luokkapohjaisina komponentteina, joissa oli elinkaaren hallintaan tarkoitettuja metodeja, kuten *componentDidMount*, *componentDidUpdate* ja *componentWillUnmount*. Tämä lähestymistapa saattoi aiheuttaa monimutkaista ja hajanaista koodia. [8.]

React.js:n Hooks-funktiot perustuvat funktiopohjaisiin komponentteihin. Ne mahdollistavat tilanhallinnan ja muiden React.js-ominaisuuksien, kuten sivuvaikutusten ja kontekstin, käytön suoraan funktiokomponenteissa ilman luokkarakenteen tarvetta. [9.]

Hooksien ansiosta komponentit voivat olla helpommin luettavia, ymmärrettäviä ja testattavia. Ne tarjoavat myös paremman tavan hallita tilaa ja elinkaarta komponenteissa. Tässä insinöörityössä on käytetty React.js:n tarjoamista vakio-Hookseista *useState*, *useEffect* ja *useContext* -funktioita. [9.]

2.1.1 useState

useState-metodilla lisätään tilamuuttuja komponenttiin. Komponentti päivittyy joka kerta, kun *useState*-muuttujaa kutsutaan. Huomion arvoista on, että muuttujan arvo vaihtuu vasta kun komponentti on päivittynyt. [9.]

2.1.2 useEffect

useEffect mahdollistaa sivuvaikutusten suorittamisen komponentin elinkaaren eri vaiheissa. Se voi esimerkiksi suorittaa tiettyä koodia aina, kun komponentti

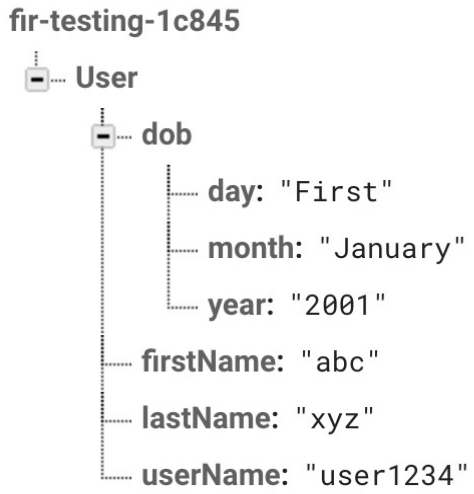
päivittyy tai purkautuu, tai vaihtoehtoisesti jotakin komponentin *useState*-muuttujaa kutsutaan. [9.]

2.1.3 useContext

useContext mahdollistaa globaalin tilanhallinnan käytön komponenttien välillä. Se palauttaa ylemmällä tasolla määritetyn kontekstin arvon. *useContext*-funktiossa määritetyn tilamuuttujan päivittyminen näkyy reaaliajassa kaikissa komponenteissa, joissa kontekstin muuttuja on esitetty. [9.]

3 Firebase-tietokanta

Firebase-tietokanta on NoSQL-tietokanta, mikä eroaa SQL-tietokannoista merkittävästi rakenteeltaan, käyttötarkoitukseltaan ja toimintatavaltaan. [10.] Firebase-tietokannassa data tallennetaan JSON-objekteina tai dokumentteina jaettuna kokoelmiin.



Kuva 2. Esimerkki Firebasen reaaliaikaisesta tietokantamallista.

Rakenteet ovat joustavia, eivätkä vaadi ennalta määriteltyjä skeemoja. SQL-tietokannat ovat relaatiotietokantoja, joissa tiedot tallennetaan tauluihin riveinä

ja sarakkeina. Jokainen taulu noudattaa tarkkaa skeemaa, ja tietueet yhdistetään usein relaation kautta. [11.]

Datan käsittelyssä Firebase-tietokanta toimii reaaliaikaisesti, ja tiedot synkronoituvat automaattisesti kaikille liitetyille asiakaslaitteille. Firebase tarjoaa sisäänrakennetun tavan kuunnella datan muutoksia reaaliajassa. SQL-tietokannoissa data haetaan yleensä perinteisillä kyselyillä (SELECT, INSERT, UPDATE). Reaaliaikaisuus ei ole oletusarvoista, ja useimmissa tapauksissa sitä varten tarvitaan erillisiä ratkaisuja (esim. WebSockets). [11.]

Huomion arvoista on myös se, että Firebase on hallinnoitu palvelu. Käyttäjän ei tarvitse huolehtia tietokannan hallinnasta, skaalautuvuudesta tai varmuuskopioista, koska ne ovat osa palvelua. SQL-tietokantojen hallinta taas vaatii usein manuaalista ylläpitoa, mukaan lukien varmuuskopiointi, suorituskyvyn optimointi ja palvelinresurssien hallinta. [11.]

Backend as a Service (BaaS) on pilvipalvelu, jossa kehittäjät ulkoistavat sovellustensa taustatoimintojen, kuten tietokantojen, käyttäjähallinnan ja pilvitallennuksen hallinnan kolmannen osapuolen tarjoajalle. [12.]

Firebase on Googlen ylläpitämä BaaS-alusta. Se tarjoaa kattavan valikoiman pilvipohjaisia työkaluja ja palveluita, joiden avulla kehittäjät voivat rakentaa ja ottaa käyttöön sovelluksia nopeasti ilman palvelininfrastruktuurin hallintaa. [12.]

Firebase perustettiin alun perin vuonna 2011 nimellä Envolve, joka oli reaaliaikaiseen viestintään keskittynyt startup-yritys. Vuonna 2014 Google hankki omistukseensa Envolve-yhtiön ja sen palvelut ja näin Firebase muuttui osaksi googlea. [13.]

Firebase tarjoaa backend-toimintaa, kuten tietojen tallennusta, reaaliaikaista tietokantaa, käyttäjähallintaa ja muita verkko- ja mobiilisovellusten kehittämistä helpottavia palveluja. [13.]

Tähän insinöörityöhön on otettu käyttöön Firebasen palveluista Authentication ja Cloud Firestore.

3.1 Firebase Authentication

Firebase Authentication -palvelu tarjoaa vaivattoman tavan lisätä käyttäjähallinnan ja käyttäjien tunnistamisen web-pohjaisiin sovelluksiin. [14.]

Firebase Authentication -palvelu tukee useita eri tunnistautumistapoja, kuten sähköposti ja salasana -, puhelinnumero- ja Google/Facebook-kirjautumista. Näistä sovelluskehittäjä voi valita yhden tai useamman tunnistautumistavan ja antaa loppukäyttäjälle mahdollisuuden kirjautua haluamallaan menetelmällä. [14.]

Kun käyttäjä rekisteröityy sovellukseen, Firebase Authentication huolehtii tunnistautumistiedon tallentamisesta ja ylläpidosta. Se tarjoaa myös API-rajapinnan, jonka avulla voi tarkistaa käyttäjän tilan ja suorittaa erilaisia toimintoja, kuten sähköpostin vahvistamisen ja salasanan palauttamisen. [14.]

3.2 Cloud Firestore

Cloud Firestore -palvelu on pilvipohjainen dokumenttitietokanta, joka mahdollistaa reaaliaikaisen tietokantaratkaisun web-pohjaisille sovelluksille. Palvelu tarjoaa skaalautuvan ja helppokäyttöisen tavan tallentaa, synkronoida ja kysellä tietoja tietokannassa. [15.]

Cloud Firestore käyttää dokumentti-kokoelma-malliin perustuvaa tiedontallennusmuotoa. Tieto tallennetaan dokumentteihin, jotka ovat JSON-muotoisia objekteja avain-arvo-pareineen. Dokumentit tallennetaan kokoelmiin, jotka helpottavat tiedon järjestämistä ja kyselyjen suorittamista. Jokaisella dokumentilla on oma tunnistemerkkijononsa, joka mahdollistaa helpon hakemisen ja päivittämisen. [15.]

Yksi Cloud Firestoren tärkeimmistä ominaisuuksista on sen reaaliaikaisuus. Kun tietoa tallennetaan tietokantaan, muutokset välittyvät automaattisesti kaikille sovelluksen käyttäjille samanaikaisesti. Tämä mahdollistaa yhteistyön sovellusten välillä ja helpottaa esimerkiksi chat-sovellusten toteuttamista. [15.]

4 Toteutus

4.1 Ketterän kehityksen hyödyntäminen projektissa

Asiakasprojektissa on tärkeää, että sekä kehittäjällä että asiakkaalla on yhteinen käsitys projektin laajuudesta ja projektin etenemisestä.

Story mapping -menetelmällä käytiin yhdessä asiakkaan kanssa läpi mitä ominaisuuksia ajanvarausjärjestelmässä tulee olla ns. pienimmässä julkaisukelpoisessa tuotteessa, MVP:ssä.

Ajanvaraussivuston käyttäjätarinat kerättiin aktiviteettien alle tärkeys- ja kiirellisyysjärjestyksessä. Aktiviteetit määriteltiin käymällä mielessä läpi käyttökokemus kuinka käyttäjä etenee sivustolla rekisteröityessään, kirjautuessaan, varatessaan aikaa ja poistaessaan varatun ajan.

Kuva 3. Asiakkaan ja ulkoasusuunnittelijan kanssa toteutettu story mapping -taulu.

Huolimatta siitä, että ajanvaraussivuston kehitys oli itsenäistä työtä, päätettiin jo suunnitteluvaiheessa ottaa mukaan projektinhallintataulu, jotta projektin etenemistä voitiin seurata visuaalisesti. Projektinhallintatauluna käytettiin Github-versionhallinnan tarjoamaa Projects-näkymää.

Uudet käyttäjätarinat ja mahdolliset tekniset ratkaisut lisättiin projektinäkymään ajanvaraussivuston Github-varaston Issues-osion “New issue” -ominaisuudella.

Käyttäjätarinan status määritettiin ensiksi backlog-merkinnäksi, jolloin Githubin projektinhallintataulu sijoitti sen Board view -näkymän vasempaan laitaan automaattisesti. Statuksen vaihduttua In progress -, Ready for Review - ja Done-merkintöihin siirtyi käyttäjätarina merkintää vastaavaan pystysuoraan sarakkeeseen.

Githubin projektinhallintataulu mahdollisti myös Roadmap-ominaisuuden käytön visualisoimaan käyttäjätarinoiden etenemistä suhteessa ajan kulkuun.

4.2 Front-end -kehitys React.js-kehyksellä

Web-projektin ulkoasusuunnittelijan luonnosteltua Figma-suunnitelmat aloituspalaverissa esitettyjen näkemysten pohjalta, luotiin ViteJS-työkalulla uusi React.js-projekti. Tässä vaiheessa projekti sisälsi ainoastaan ViteJS:n luomat välttämättömimmät tiedostot ReactJS-sovelluksen suorittamiseen. Sovellusrakenteen selkeyden vuoksi tehtiin tarvittavat kansiot (pages, components, context) juurikansiossa sijaitsevan src-kansion alle.

4.2.1 Ulkoasu

Sivuston ylälaidassa kiinteästi pysyvä navigointipalkki luotiin components-kansioon. Story mapping -suunnitelman mukaisesti yläpalkissa näkyy tilaajan logo sekä kirjautuneen käyttäjän nimi tai “ei kirjautunut”, kun käyttäjä ei ole kirjautunut. Kehityksen alkuvaiheessa sijoitettiin nimen kohdalle nimeä havainnollistava merkkijono. CSS-asetus *position: sticky* asetti yläpalkin määrätylle paikalleen aivan sivuston yläreunaan.

Projektin etusivuna toimi pages-kansioon sijoitettu FrontPage-komponentti. Figma-sovelluksella toteutetun ulkoasusuunnitelman mukaisesti etusivulla on kolme suurta painiketta ajanvaraukseen, ajanvarausten hallinnointiin ja uloskirjautumiseen. Etusivun jälkeen kehitys muuttui hyvin suoravíivaiseksi varaus- ja hallinnointisivun osalta.

Varaus- ja hallinnointisivuilla esiintyvän kalenterikomponentin lähdekoodi on peräisin aiemmasta projektista. Ajanvarausjärjestelmää varten komponenttiin oli tehtävä vain pieniä muutoksia. Projektin alkuvaiheessa kokeiltiin NodeJS-sovellussäilön valmista kalenterikomponenttia (React-calendar), mutta sen rajallinen muokattavuus niin ulkoasun kuin toiminnallisuuden suhteen teki siitä projektiin sopimattoman.

Kuva 4. Ajanvarausjärjestelmän toiminnalisia ja ulkonäöllisiä vaatimuksia varten räätälöity kalenteri

4.2.2 Modaalikomponentit

Ajanvarausjärjestelmä haluaa käyttäjältä varmistuksen ajanvarausta tehtäessä tai poistaessa. Projektin Figma-suunnitelmia noudattaen varmistus nousee ajanvaraussivun tai varauksenhallintasivun päälle modaalikomponenttina.

React.js-kehyksessä on lukuisia kolmannen osapuolen ratkaisuja modaalikomponentin toteutukseen, joista mainittakoon NPM-sovelluskeskuksen React-modal -sovelluspaketti ja eri käyttöliittymäkirjastojen (Bootstrap, Material UI) omat modaalikomponentit.

Koska modaalikomponentti on rakenteeltaan varsin yksinkertainen, päädyttiin ohjelmoimaan se ajanvarausjärjestelmään itse. Tämä ratkaisu varmistaa sen, että kolmannen osapuolen mahdolliset päivitykset eivät tulevaisuudessa riko projektin rakennetta.

Modaali-ikkuna tulee esiin *useState*-metodilla luodun boolean-muuttujan arvon ollessa *true*. Looginen &&-operaattori suorittaa aaltosulkeiden sisällä olevan

koodin ainoastaan silloin, kun molemmat väittämät ovat totta. Jälkimmäinen väittämä on aina totta, joten komponentin näkymä riippuu ensimmäisen väittämän arvosta.

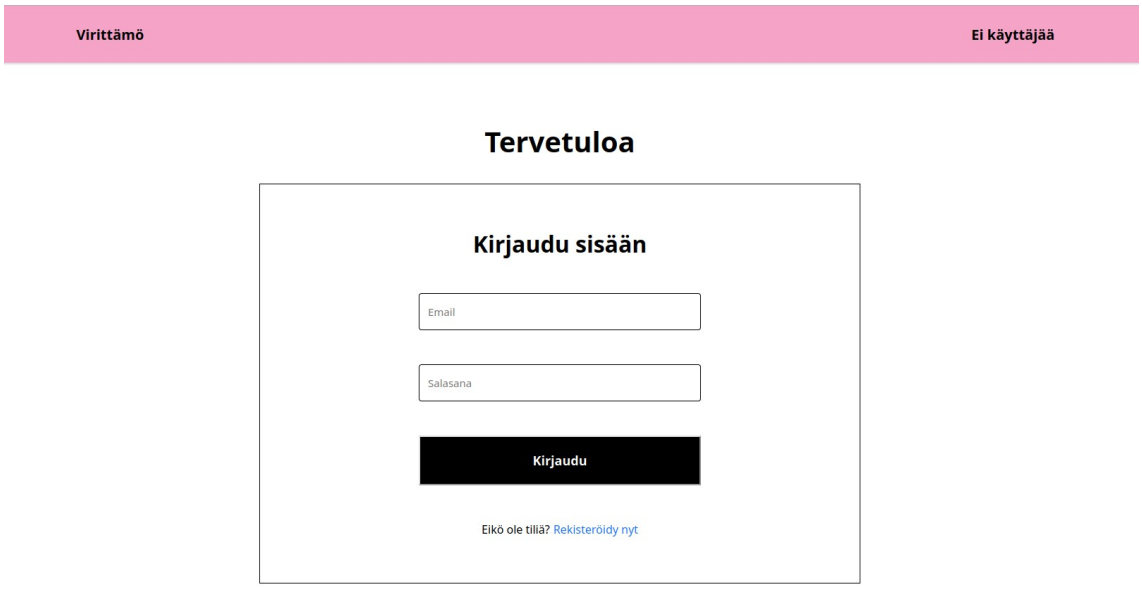
```
{showConfirmWindow && (
   <div className="fullscreen-modal">
      <div className="modal-detail-content">
       …
         <button onClick={() => setShowConfirmWindow(false)}>
              Peruuta
         </button>
      </div>
   </div>
)}
```

Esimerkkikoodi 1. Boolean-muuttujan ja modaalikomponentin muodostama looginen &&-operaattori

Tyylittelyasetuksissa modaalikomponentille asetetaan tarpeeksi suuri päällekkäisten elementtien näkyvyysjärjestystä kuvaava *z-index* -arvo. Modaalikomponentti sulkeutuu, kun komponentin näkyvyydestä vastaava boolean-muuttuja saa arvokseen *false*.

4.2.3 Käyttäjänhallinta

Käyttäjänhallintaan tarkoitetussa LoginPage-komponentissa luotiin ensin tekstinsyöttökentät ja niille omat paikalliset tilamuuttujat ReactJS:n useState-ominaisuudella. Kun komponentti oli kykenevä ottamaan vastaan käyttäjän tekstisyötettä ja käsittelemään syötteiden tilamuuttujaa ReactJS:n *useEffect*-ominaisuudella käyttäjän painaessa sivun alalaidan painiketta, aloitettiin tietokantayhteyden kehitys React.js-komponenttien ja Firebasen välillä.



Kuva 5. Ajanvarausjärjestelmän kirjautumissivu.

Koska käyttäjätiedoissa tapahtuvia muutoksia on päivitettävä useassa komponentissa samanaikaisesti, luotiin tietokannan ja ReactJS-komponenttien välille ReactJS:n *useContext*-ominaisuuteen pohjaava AuthContext-komponentti. Käyttäjähallintaan tarkoitettu LoginPage-komponentti kutsuu AuthContext-komponenttia, joka taas kutsuu Firebase Authentication -palvelun käyttäjähallintatoimintoja asynkronisesti.

NodeJS-sovelluskeskuksen firebase-kirjaston onAuthStateChanged-funktio kuuntelee Firebasen käyttäjähallinnassa tapahtuvia muutoksia ja päivittää muutosten tapahduttua AuthContext-komponentin user- ja admin-muuttujaa. Komponenttihierarkian ylimmällä tasolla toimivan AuthContext-komponentin tilamuutokset siirtyvät niihin alakomponentteihin, joissa on tieto AuthContext-komponentin olemassaolosta. Tällä tavoin kaikilla käyttäjätietoja tarvitsevilla komponenteilla on reaaliaikainen tieto siitä, onko käyttäjä kirjautunut ja onko käyttäjällä admin-oikeudet hallita myös muiden tekemiä varauksia.

```
  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (currentUser) =>
      setUser(currentUser);
    );
    return () => {
      unsubscribe();
    };
  }, []);
```

Esimerkkikoodi 2. onAuthStateChanged-metodi kuuntelee Firebasen Authentication-palvelussa tapahtuvia käyttäjätietomuutoksia ja välittää ne komponenteille AuthContext-komponentin kautta.

React.js:n Route-komponenttiin luotiin mukautettu reititys-metodi ohjaamaan käyttäjä kirjautumista vaativille sivuille vain, jos käyttäjä on kirjautunut.

```
const UserRoute = ({ children }) => {
  const { user } = UserAuth();

  if (!user) {
    return <Navigate to='/login' />;
  }
  return children;
};
```

Esimerkkikoodi 3. Reititys-metodi, joka ei päästä kirjautumatonta käyttäjää kirjautumista vaativaan osioon.

4.3 Back-end -kehitys Firebasella

Insinöörityön pohjana olevan ajanvarausjärjestelmän tietokannaksi valikoitui Google Firebase -pilvitallennuspalvelu hyvin pitkälti aikataulullisista syistä. Omien projektien ja erään aiemman työprojektin kautta Firebasen käytöstä back-end -palveluna oli kertynyt runsaasti kokemusta.

Hyvin suunniteltu ja toteutettu front-end -puoli ei juurikaan ole kiinnostunut siitä, mikä back-end -ratkaisu on käytössä. Tarpeen tullen back-end -puolen tulee olla helposti vaihdettavissa ratkaisusta toiseen. Käytännössä tämä toteutetaan keskittämällä sovelluksen tietokantakutsut React.js:n Context-kirjaston avulla komponenttien ulkopuolelle, tai vaihtoehtoisesti Redux-kirjaston globaalia tilanhallintaa hyödyntämällä.

Koska ajanvarausjärjestelmän MVP-versio oli saatava nopealla aikataululla testikäyttöön, sijoitettiin tietokantakutsut suoraan komponentteihin. Datahaun suorittava Firebase-kutsufunktio päivittää komponentin tilamuuttujaa, jonka myötä sivu päivittää itsensä automaattisesti.

4.3.1 Authentication-palvelu

Google Firebasen suosion taustalla on sen vaivaton käyttöönotto ja ylläpitäminen. Authentication-palvelussa Firebase huolehtii käyttäjätietojen tallennuksesta ja käyttäjätunnisteen palautuksesta, helpottaen täten kehittäjän työtä huomattavasti. Authentication-palvelun tarjoaman sähköposti ja salasana -kirjautumisen katsottiin riittävän hyvin ajanvarausjärjestelmän tarpeisiin.

Sähköposti ja salasana -kirjautumisen yhteyteen toteutettiin Firebase Authentication -palvelun tarjoama sähköpostiosoitteen varmentaminen. Käyttäjän rekisteröinnin jälkeen järjestelmä kirjaa käyttäjän ulos kirjautumispalvelusta ja lähettää käyttäjän antamaan sähköpostiosoitteeseen varmennusviestin. Kun käyttäjä painaa sähköpostiin saapuneessa viestissä olevaa varmennelinkkiä, Firebase Authentication -palvelu merkitsee käyttäjätietoihin sähköpostiosoitteen varmennetuksi. Sisäänkirjautumisen määritin onnistuvan vain silloin, kun sähköpostiosoite on varmennettu.

```
const createUser = async (email, password, group) => {
    try {
      const userdata = await createUserWithEmailAndPassword(auth,
email, password);
      await setDoc(doc(db, "users", userdata.user.uid), {
        email: email,
        group: group,
      });
      signOut(auth);
      await sendEmailVerification(userdata.user);
    } catch (error) {
      window.alert("Käyttäjätilin luominen ei onnistunut:\n\n" + er-
ror);
    }
  };
```

Esimerkkikoodi 4. Käyttäjätietojen luonti, uloskirjautuminen ja varmennussähköpostin lähetys

4.3.2 Cloud Firestore -palvelu

Koska Cloud Firestore -pilvitallennuspalvelu käyttää dokumentti-kokoelma-malliin perustuvaa tiedontallennusmuotoa, tallentuu tieto dokumentteihin JSON-muotoisilla avain-arvo -pareilla. Ajanvarausjärjestelmän tietokantakehityksessä luotiiin varaus- ja hallintakomponentteihin Firebase-kirjaston Firestore-kutsuja käyttävät tallennus- ja hakufunktiot. *setDoc()*-tallennusfunktio ottaa parametreikseen tiedon tietokannasta sekä kokoelma-dokumentti -muodossa tallentuvat merkkijonomuuttujat. Dokumentteihin voi tallentaa tiedon myös JSON-objektina.

```
 const createBooking = async () => {
    try {
      const dbtime = year + "-" + month + "-" + day + "-" + time;
      await setDoc(doc(db, "users", user.uid, "bookings", dbtime), {
        year: selectedDate.getFullYear().toString(),
        month: selectedDate.getMonth().toString(),
        day: selectedDate.getDate().toString(),
        time: time,
        weekday: selectedDate.getDay().toString(),
        topic: topic,
      });
    } catch (error) {
      window.alert("Ongelmia tietokantaan tallentamisessa:\n\n" + er-
ror);
    }
  };
```

Esimerkkikoodi 5. Varaustietojen vieminen tietokantaan päivämäärän perusteella *setDoc()*-funktiolla.

Firestore-kirjaston tallennusfunktiot sisältävät JavaScript-kirjaston Promise-ominaisuuden, joten suoritettava funktio voidaan jättää odottamaan tietokantatapahtuman valmistumista async/await -syntaksia käyttäen ennen kuin funktio suorittaa toimintonsa loppuun.

Firestore-kirjaston *getDoc()*-tietokantahakufunktiolla haetaan dokumenttia, joka sisältää tietyn päivän kaikki varaustiedot. On huomioitava, että yksittäistä dokumenttia haettaessa on tarkistettava, että hakutulos todella palauttaa haetun dokumentin *exists()*-toiminnolla.

Epäonnistunut tai keskeytynyt await-tietokantakutsu palauttaa Firestore-palvelusta virheilmoituksen React.js-komponentille, joka näytetään käyttäjälle window-objektin alert-toimintoa käyttäen async-funktion catch-tilaa. Epäonnistuneen await-kutsun jälkeen funktion suoritus keskeytyy.

Onnistunut tietokantakutsu palauttaa dokumentin sisältämät varaustiedot useState-muuttujaan, jolloin sivu päivittyy reaaliaikaisesti.

```
const fetchBookings = async () => {
  try {
    const docSnap = await getDoc(
      doc(db, "bookings", date.getFullYear().toString(),
   date.getMonth().toString(), date.getDate().toString())
    );
    if (docSnap.exists()) {
      const documents = [];
      let i = 0;
      while (docSnap.data()[i]) {
        documents.push(docSnap.data()[i]);
        i++;
      }
      setBookings(documents);
    } else {
      setBookings(initTimes());
    }
  } catch (error) {
    window.alert("Ongelmia tietokannasta hakemisessa:\n\n" + error);
  }
};
```

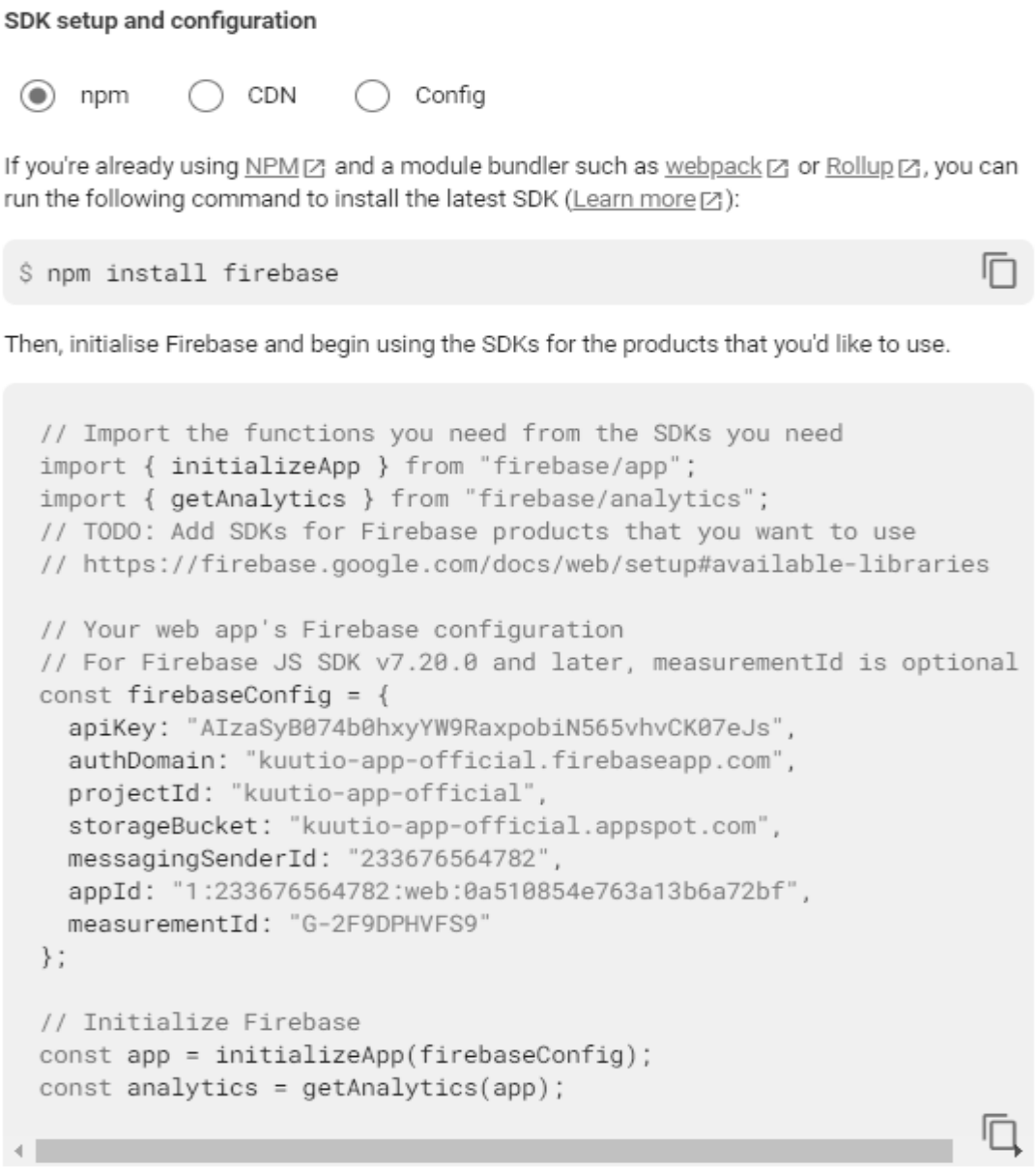
Esimerkkikoodi 6. Cloud Firestore -palvelun palauttaman varaustiedot sisältävän dokumentin avaaminen React.js-komponentin käyttöön.

Kuutio-ajanvarausjärjestelmän varaustiedot tallennettiin Firebase-tietokantaan vuosiluvun ja päivämäärän muodostamaan dokumentti-kokoelma -polkuun. Tällä tavoin tietokannan ylläpitäjän on tarpeen tullen helppo löytää oikea varaus päivämäärän ja kellon ajan mukaan.

JSON-objekti sallii tiedon tallentamisen listoina käyttämällä hyväksi JavaScript-kirjaston spread-funktiota. Varaustietojen tallentaminen tapahtui luomalla kyseisen päivän kaikista kellonajoista listan objekteja, jotka sisälsivät tiedon kellonajasta ja tiedon, onko kyseisessä objektissa tallennettuna varaustietoja.

4.4 React.js:n ja Firebasen integrointi

Google Firebase tietokanta- ja kirjautumispalveluineen otettiin käyttöön luomalla uusi projekti Googlen ylläpitämän Firebase-palvelun konsolinäkymässä. Firebasen liittäminen front-end -puolen kanssa tapahtui luomalla alustustiedosto ReactJS-projektiin. Kun Firebase-tietokannan alusta oli määritelty verkkopohjaiseksi sovellukseksi, tarjosi Firebasen konsoli alustustiedoston vaatimat asetukset käyttövalmiina.



Kuva 6. Firebase-palvelun tarjoama integrointimenetelmä React.js-projektiin.

Firebasen Authentication -kirjautumispalvelu ja Firestore-pilvitallennuspalvelu luotiin Firebase-projektin konsolinäkymässä. Edellä mainitut palvelut liitettiin React.js-projektin Firebase-alustustiedostoon kopioimalla palvelua vastaavat alustusfunktiot front-end -puolen React.js-kehyksen Firebase-kirjastosta.

Firebasen käyttäjähallinta- ja tietokantapalvelut integroitiin React.js-komponentteihin tuomalla Firebasen alustustiedostosta palvelua vastaavat alustuskutsumuuttujat (getAuth ja getFirestore). Tämän lisäksi kutsuttiin komponenteissa Firebasen palvelua vastaavat Firebase-kirjastot.

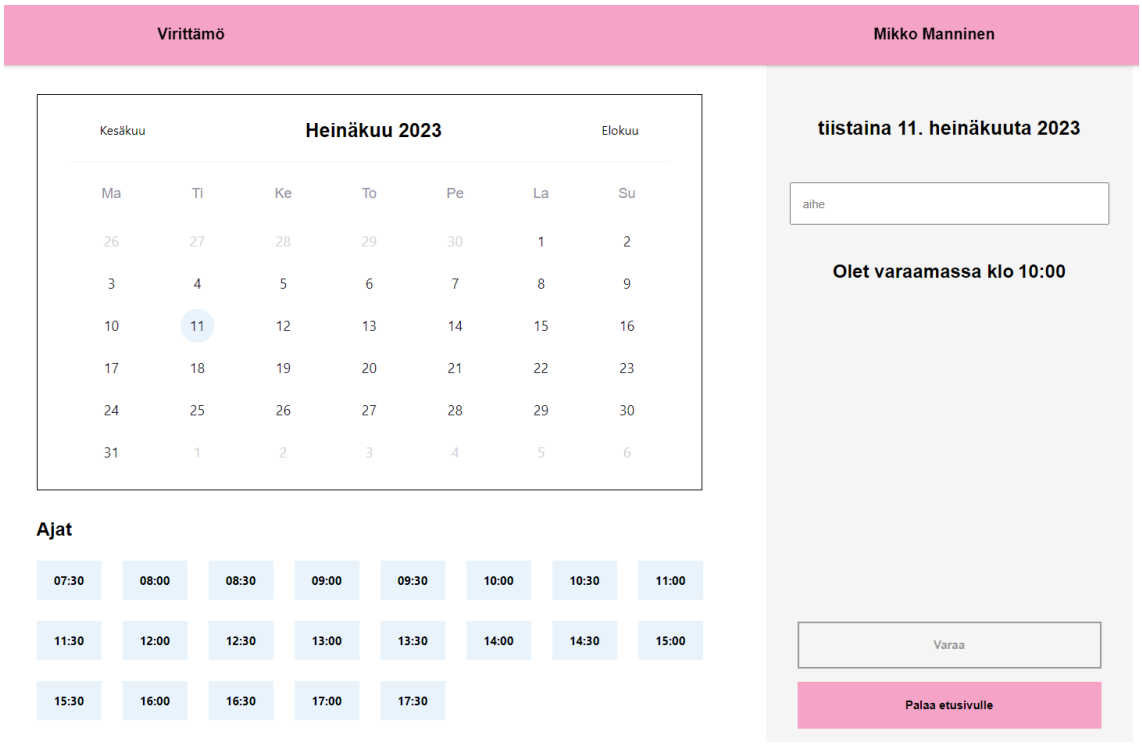
```
import { getFirestore } from "firebase/firestore";
import { getAuth } from "firebase/auth";

const firebaseConfig = {
  ...
};

export const app = initializeApp(firebaseConfig);
export const db = getFirestore(app);
export const auth = getAuth(app);
```

Esimerkkikoodi 7. Firebase-asetusten liittäminen React.js-projektiin

Firebasen dokumenttisivusto tarjosi kattavat dokumentaatiot käyttäjähallinta- ja tietokantapalveluiden käyttöönottoon komponenttitasolla.

4.5 Ulkoasun viimeistely

React.js-komponenttien ja Firebase-tietokannan välisen toiminnallisuuden valmistuttua viimeisteltiin ajanvarausjärjestelmän ulkoasu. Asiakkaan ja ulkoasusuunnittelijan todettua projektin näyttävän Figma-suunnitelmien mukaisesti, oli ajanvarausjärjestelmä valmis tuleviin käyttäjätesteihin.



Kuva 7. Varaussivun valmis ulkoasu

5 Yhteenveto

Tämän insinöörityön tarkoituksena oli tehdä yrityksen tiloissa sijaitsevan neuvotteluhuoneen ajanvarausjärjestelmä React.js -kehyksellä ja Firebase-tietokannalla. Aikataulullisista syistä johtuen työ tehtiin MVP-periaatteella.

Projekti aloitettiin tapaamisella asiakkaan ja ulkoasusuunnittelijan kanssa, jolloin hahmoteltiin sivuston toiminnallisuutta, kuten millainen kirjautumisen tulisi olla, mitä elementtejä pääsivulla näkyy ja miten itse ajanvaraus tehdään. Yhteiset näkemykset kirjoitettiin valkotaululle post-it -lapuille ketterän kehityksen story mapping -menetelmää hyödyntäen. Ennen ohjelmointiurakan aloittamista kirjoitin post-it -lapuista itselleni käyttäjätarinat kanban-tauluun

Front-end -puolen ohjelmointi React.js -kehyksellä oli varsin suoraviivaista ulkoasusuunnittelijan tekemien Figma-luonnosten pohjalta.

Google Firebase-tietokanta oli nopeaa ottaa käyttöön luomalla henkilökohtaiseen Google-tiliin uusi Firebase-projekti. Firebase-palvelu tarjosi

React.js-kehystä varten alustustiedot, jotka liitettiin React.js-projektiin. Tietokanta oli tämän jälkeen käytettävissä.

Tuloksena oli toimiva ajanvarausjärjestelmä, jonka testikäyttö päästiin aloittamaan sovitussa aikataulussa. Testikäyttö sujui lähestulkoon ongelmitta.

Kuutio-ajanvarausjärjestelmä -projektia on myöhemmin jatkettu toisten ohjelmoijien toimesta ja Firebase-tietokanta on vaihdettu toimiston omalla palvelimella toimivaan MERN-tietokantaratkaisuun. Ulkoasu on kuitenkin pysynyt pitkälti samanlaisena.

Työ oli hyvin mielenkiintoinen ja opettavainen. Jos olisi ollut enemmän aikaa käytettävissä, olisin siirtänyt kaikki front-end -puolen tietokantakutsut pois sivujen omista komponenteista yhteen useContext-komponenttiin. Tämä toimenpide helpottaisi ja vähentäisi työnmäärää tietokantaratkaisua vaihdettaessa.

Lähteet

1 What is React | PubNuB. 2024. Verkkoaineisto. <https://www.pubnub.com/guides/react/> Luettu 15.10.2024.

2 React Components – GeeksforGeeks. 2024. Verkkoaineisto. <https://www.geeksforgeeks.org/reactjs-components/> Luettu 15.10.2024.

3 Understanding the React component hierarchy and composition. 2023. Verkkoaineisto. <https://www.learnbestcoding.com/post/137/understanding-the-react-component-hierarchy-and-composition> Luettu 16.10.2024.

4 What is the Virtual DOM in React. 2024. Verkkoaineisto. <https://www.freecodecamp.org/news/what-is-the-virtual-dom-in-react/> Luettu 16.10.2024.

5 Managing State – React. 2024. Verkkoaineisto. <https://react.dev/learn/managing-state> Luettu 17.10.2024.

6 Redux vs Context API: When to use them – DEV Community. 2023. Verkkoaineisto. <https://dev.to/ruppysuppy/redux-vs-context-api-when-to-use-them-4k3p> Luettu 17.10.2024.

7 Axios vs. Fetch API: Selecting the Right Tool for HTTP Requests. 2023. Verkkoaineisto. <https://medium.com/@johnnyJK/axios-vs-fetch-api-selecting-the-right-tool-for-http-requests-ecb14e39e285> Luettu 17.10.2024.

8 Introducing Hooks – React. 2018. Verkkoaineisto. <https://legacy.reactjs.org/docs/hooks-intro.html> Luettu 18.10.2024.

9 React Hooks – GeeksforGeeks. 2024. Verkkoaineisto. <https://www.geeksforgeeks.org/reactjs-hooks/> Luettu 18.10.2024.

10 Firebase Realtime Database. 2024. Verkkoaineisto. <https://firebase.google.com/docs/database> Luettu 19.10.2024.

11 Firebase vs. MySQL: A Database Comparision – Integrate.io. 2023. Verkkoaineisto. <https://www.integrate.io/blog/firebase-vs-mysql/> Luettu 19.10.2024.

12 Overview of Firebase Backend as a Service Platform. 2024. Verkkoaineisto. <https://medium.com/@engrshul/overview-of-firebase-aa8e05710542> Luettu 19.10.2024.

13 Introduction To Google Firebase. 2019. Verkkoaineisto. <https://www.c-sharpcorner.com/article/introduction-to-google-firebase/> Luettu 19.10.2024.

14 Firebase Authentication | Firebase. 2024. Verkkoaineisto. <https://firebase.google.com/docs/auth> Luettu 19.10.2024.

15 Firestore | Firebase. 2024. Verkkoaineisto. <https://firebase.google.com/docs/firestore> Luettu 19.10.2024.