KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Robert Kuhlmann

OPISKELIJATAPAHTUMAKALENTERIN TOTEUTUS MODERNEILLA VERKKOKEHITYSTEKNOLOGIOILLA

Opinnäytetyö
Toukokuu 2019

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Toukokuu 2019
Tietojenkäsittelyn koulutusohjelma

Tikkarinne 9
80200 JOENSUU
+358 13 260 600 (vaihde)

Tekijä
Robert Kuhlmann

Nimeke
Opiskelijatapahtumakalenterin toteutus moderneilla verkkokehitysteknologioilla

Toimeksiantaja
Karelia-ammattikorkeakoulun opiskelijakunta POKA

Tiivistelmä

Opinnäytetyön tarkoituksena oli toteuttaa Karelia-ammattikorkeakoulun opiskelijakunta POKAlle tapahtumakalenteri verkkosovelluksena, jossa tiedotetaan opiskelijatapahtumista. Tavoitteena kehitysprosessissa oli hyödyntää moderneja verkkosovelluskehityksen teknologioita.

Sovellus toteutettiin React-kirjaston avulla hyödyntämällä Firebase-alustaa palvelinrajapintana. Toteutuksen raportoinnissa paneuduttiin syvällisemmin käytettyihin työkaluihin, niiden käyttötarkoitukseen, integroitavuuteen sekä tarkasteltiin niiden avulla toteutuksessa tehtyjä ratkaisuja. Raportin lopussa arvioitiin toteutuksessa saavutettuja tuloksia sekä työkalujen soveltuvuutta verkkosovelluskehitykseen. Raportissa käsitellään myös tietosuojaan liittyviä asioita verkkosovelluksen toteutuksessa.

Lopullinen verkkosovellus saatiin tavoitteiden mukaisesti valmiiksi täyttäen kaikki sille asetetut kriittisimmät vaatimukset. Raportin lopussa esitellään myös sovelluksen kannalta oleellisia jatkokehitysideoita tulevaisuuden varalle.

Kieli	Sivuja 46
suomi	Liitteet 1
	Liitesivumäärä 2

Asiasanat
verkkosovellus, opiskelijatapahtumat, tapahtumakalenteri, JavaScript, Firebase, React, Redux

THESIS
May 2019
Degree Programme in Business Information Technology

Tikkarinne 9
80200 JOENSUU
+ 358 13 260 600 (switchboard)

Author
Robert Kuhlmann

Title
Development of Student Event Calendar with Modern Web Development Technologies

Commissioned by
Student Union POKA of Karelia UAS

Abstract
The purpose of this thesis was to develop a student event calendar as a web application for Student Union POKA of Karelia UAS. The goal of this thesis was also to discover and utilize modern web technologies in the development process.

Application was developed with React and it uses Firebase platform for server-side functionality. The report of the development process described the tools used in the process profoundly and examined their usage, integrability and the solutions provided with the specific toolset. The results of the development process and suitability of the tools used with it were evaluated at the end of the report. The report also examined privacy related subjects considering web application development.

The final web application was successfully prepared in the process and it fulfills all of the critical requirements. Performance, future development ideas and other quality related measurements were also analyzed at the end of the report.

Language	Pages 46
Finnish	Appendices 1
	Pages of Appendices 2

Keywords
web development, student events, event calendar, JavaScript, Firebase, React, Redux

Sisältö

Lyhenteet

BaaS	Back-end as a Service. Palvelun malli, jonka mukaan palvelussa on sisäänrakennettuna tyypilliset palvelinrajapinnan toiminnot, esimerkiksi tietokannan käsittely, kirjautuminen ja tiedostojen käsittely (Roberts 2018).
CSS	Tyylittelykieli, jonka avulla voidaan määritellä, miten HTML-dokumentin elementit renderöidään käyttöliittymään. Kielen avulla voidaan myös hallita HTML- ja XML-dokumenttien elementtien renderöitymistä eri medioissa. (Mozilla ja yksittäiset avustajat 2019b.)
DOM	Document Object Model. Dokumenttioliomalli kuvastaa HTML-dokumentin rakennetta ja tarjoaa rajapinnan dokumentin muokkaukselle (Mozilla ja yksittäiset avustajat 2019c).
ES5, ES6, ES7	ECMAScript-standardin eri versioita. Versioita ilmaistaan myös vuosiluvuilla (esim. ES6 tai ES 2015).
FaaS	Functions as a Service. Palvelun malli, jonka mukaan käyttäjä voi julkaista funktioita palvelussa vastaamaan yksittäisistä toiminnoista. (Roberts 2018).
HTTP	Tiedonsiirtoprotokolla, jonka avulla resursseja voidaan noutaa verkosta, kuten HTML-dokumentteja ja JavaScript-tiedostoja (Mozilla ja yksittäiset avustajat 2019a).

HTML	Englanniksi Hypertext Markup Language. Hyperlinkkejä sisältävän tekstin kuvauskieli, jolla verkkosivustojen käyttöliittymät rakennetaan (Mozilla ja yksittäiset avustajat 2019d).
JSX	React-kirjaston tukema JavaScriptin syntaksi, jossa käyttöliittymäelementtejä voidaan sisällyttää osaksi JavaScriptiä (Facebook Inc. 2019d).
SPA	Single Page Application. Sovellus, jossa sivuston toiminnallisuuden sisältävät resurssit ladataan kerran selaimeen. SPA kommunikoi yleensä palvelinten kanssa reaaliajassa päivittäen dataa suorituksen aikana. (Halme 2018.)

1 Johdanto

Internet on muuttunut viimeisten vuosikymmenien aikana staattisista HTML-dokumenteista koostuvasta verkostosta laajamittaiseksi sovellusalustaksi. Perinteiset sivustot syrjäyttäneet modernit verkkosovellukset taipuvat useille eri päätelaitteille, kommunikoivat reaaliaikaisesti useiden eri rajapintojen kanssa ja vastaavat toiminnallisuudeltaan enemmän tietokoneohjelmia kuin verkkosivuja. Tarve monipuolisille verkkosovelluksille on aktivoinut kehittäjät tekemään uusia avoimeen lähdekoodiin perustuvia sovelluskehyksiä, alustoja ja kirjastoja, jotka ovat mullistaneet koko verkkosovelluskehityksen.

Tässä opinnäytetyössä sovellettiin moderneja verkkosovelluskehityksen teknologioita ja muodostettiin niiden pohjalta toimeksiantoa vastaava verkkosovellus. Toimeksiantona toteutettiin opiskelijatapahtumakalenteri verkkosovelluksena Karelia-ammattikorkeakoulun opiskelijakunta POKAlle. Toimeksiannon pääasiallisena vaatimuksena oli avoin tapahtumakalenteri, johon yksittäiset opiskelijat ja opiskelijajärjestöt pystyvät lisäämään omia tapahtumiaan. Sovelluksella pyritään vastaamaan opiskelijakunnan tarpeeseen saada kaikki tapahtumat keskitetysti yhteen paikkaan ja kalenteri tulee toimimaan yhtenä uutena tiedotuskanavana opiskelijoille.

Varsinainen verkkosovellus toteutettiin JavaScript-ohjelmointikielellä. Sovelluksesta tehtiin yhden sivun sovellus (SPA) ja sovelluksen kehityksessä käytettiin React-kirjastoa sekä muita tarvittavia kirjastoja ja paketteja toiminnallisuuden tuottamisessa. Sovelluksen palvelinrajapinnan toiminnallisuus koostuu kokonaisuudessaan Googlen Firebase-alustan palveluista, joiden toimintaa käsitellään raportissa tarkemmin. Toimeksiannon perusteella tavoiteltiin syvällisempää käsitystä alustan eri toiminnoista sekä toteutuksessa hyödynnetyistä työkaluista.

Opinnäytetyön raportti on erityisesti kohdennettu verkkosovelluskehittäjille, mutta myös muille ohjelmoijille sekä ohjelmistosuunnittelusta ja -arkkitehtuurista vastaaville henkilöille, jotka voisivat mahdollisesti hyödyntää opinnäytetyössä sovellettuja työkaluja.

2 JavaScript

Mozillan dokumentaatiossa JavaScript on määritelty järjestelmäriippumattomana olio-pohjaisena skriptauskielenä, jonka käyttö selaimissa painottuu interaktiivisten verkkosivujen toteuttamiseen. JavaScriptin avulla sivustot kommunikoivat palvelinten ja rajapintojen kanssa ilman, että sivuston resursseja, kuten HTML-dokumentteja, JavaScript-tiedostoja ja CSS-tyylitiedostoja, ladattaisiin uudestaan palvelimelta. Myös grafiikoiden generointi, animaatiot ja sivuston ulkoasun muokkaus ovat mahdollisia JavaScriptin avulla. Sivustojen rakennetta ja ulkoasua hallitaan muokkaamalla sivustojen dokumenttioliomallia (DOM) kutsumalla JavaScriptin selainrajapinnan funktioita. Sivustojen eri toimintoja, kuten näppäinten painalluksia ja lomakkeiden lähetyksiä voidaan käsitellä JavaScriptin avulla. (Mozilla ja yksittäiset avustajat 2019f.)

JavaScript-kielen standardoidusta määritelmästä vastaa Ecma International -yhdistys. Tämän vuoksi nykyaikainen nimitys standardin mukaiselle JavaScript-kielelle on ECMAScript. Nimitys muodostuu suoraan standardin määritelmää ylläpitävän järjestön nimestä, mutta kielen nykyinen nimitys JavaScriptinä on jäännös kielen alkuajoista. Standardia kehitetään TC39-komitean toimesta ja muutokset otetaan vastaan kehittäjäyhteisön esittämistä ehdotuksista, jotka komitea käsittelee ja hyväksyy osaksi standardia. (Rauschmayer 2018.)

ECMAScript 7 -versiosta (myös ECMAScript 2016 tai ES7) alkaen komitea on sitoutunut julkaisemaan vuosittain uuden ECMAScript-version uusimmilla hyväksytyillä ominaisuuksilla. Uusinta standardia noudattavan ohjelmakoodin suorittaminen selaimilla ei kuitenkaan ole aina mahdollista, sillä selainten JavaScript-moottorit ovat itsenäisiä ohjelmistoja ja täten standardin

noudattaminen on selainten kehittäjien vastuulla. Siksi ECMAScript-standardia kehitetään säilyttäen vanhat ominaisuudet ja ottaen uudet ominaisuudet entisten rinnalle. (Rauschmayer 2018.)

3 React

3.1 Taustaa

React on Facebookin kehittämä käyttöliittymien kehitykseen luotu JavaScript-kirjasto. Reactilla luodut käyttöliittymät koostuvat uudelleenkäytettävistä *komponenteista*. React-kirjastolla tehdyt selainsovellukset ovat pääasiassa selaimessa ajettavia yhden sivun sovelluksia (SPA), mutta Reactilla voidaan tehdä myös mobiilisovelluksia ja luoda sivustoja palvelinympäristössä. React-kirjaston avulla luodut komponentit koostuvat käyttöliittymäelementeistä, joita komponentit toiminnallisuudellaan tulostavat käyttöliittymään eli komponentit *renderöivät* sovelluksen käyttöliittymän. (Facebook Inc. 2019e.)

Andrea Pappin artikkelissa ”The History of React.js on a Timeline” kuvatun Reactin aikajanan mukaan Reactin lähdekoodi julkaistiin avoimena jo vuonna 2013, mutta sen suurin suosio alkoi vasta 2016 vuoden aikana (Papp 2018). Nykyään React on muihin käyttöliittymäkehitykseen luotuihin ohjelmistokehyksiin ja kirjastoihin verrattuna yksi kysytyimmistä teknologioista.

React on puhtaasti JavaScript-painotteinen kirjasto ja sitä on tehokasta käyttää vähintään ECMAScript 6-standardin mukaisten ominaisuuksien kanssa. JSX- ja ES6-syntaksit vaativat myös käännöstyökalujen hyödyntämisen laajemman selaintuen saavuttamiseksi, joka vaatii yleensä myös erillisen kehitysympäristön pystyttämisen. Reactia on mahdollista hyödyntää myös pienemmillä asetuksilla lisäämällä sen koodi pienemmän sivuston HTML-dokumenttiin. (Facebook Inc. 2019b.)

3.2 JSX

React-kirjaston kanssa käytettävä ohjelmointikieli on JavaScript, mutta Reactin dokumentaatiossa kehitykseen suositellaan käytettäväksi Reactin omaa *JSX-syntaksia*. JSX on pääasiassa JavaScriptiä, joka mahdollistaa käyttöliittymäelementtien sisällyttämisen osaksi JavaScriptiä. Täten käyttöliittymäelementtien mallinnukseen tarvittava logiikka voidaan toteuttaa yhdessä tiedostossa esimerkiksi erillisten JavaScript- ja HTML-tiedostojen käyttämisen sijaan, kuten kuvan 1 esimerkissä on näytetty. (Facebook Inc. 2019d.)

```
<FilterWrapper>
    <Filters>
        <DateShortcuts show={this.state.dateRange !== 'custom'}>
            <button
                onClick={() => this.setSelectedRange('today')}
                style={this.selectedStyle('today')}><Translate t="TODAY" /></button>
            <button
                onClick={() => this.setSelectedRange('this_week')}
                style={this.selectedStyle('this_week')}><Translate t="THIS_WEEK" /></button>
            <button
                onClick={() => this.setSelectedRange('this_month')}
                style={this.selectedStyle('this_month')}><Translate t="THIS_MONTH" /></button>
            <button
                onClick={() => this.setSelectedRange('custom')}
                style={this.selectedStyle('custom')}><Translate t="CHOOSE_PERIOD" /></button>
        </DateShortcuts>
        <DateRangeSelect show={this.state.dateRange === 'custom'}>
            {this.renderDatePickers()}
        </DateRangeSelect>
    </Filters>
```

Kuva 1. JSX-syntaksia.

3.3 Komponentit

Reactin sisältämät komponentit jaotellaan tyypillisesti kahteen eri kokonaisuuteen: *funktionaalisiin* ja *luokkapainotteisiin komponentteihin*. Myös koko React-sovellus itsessään rakennetaan yhden juurikomponentin sisään, jonka sisälle sovelluksen komponenttipuu rakentuu. (Facebook Inc. 2019a.)

Luokkapainotteiset komponentit ovat suurimman toiminnallisuuden omaavia komponentteja React-kirjastossa. Luokkapainotteiset komponentit ovat koodissa kirjaimellisesti ES6-standardin mukaisia luokkia, mistä myös niiden nimitys muodostuu. Luokka-painotteisilla komponenteilla voi olla funktionaalisista komponenteista poiketen oma tila (state), jota muokataan komponentin *elinkaarimetodeilla*. Elinkaarimetodit ovat luokkapainotteisen komponentin metodeja, joilla voidaan tehdä muutoksia komponentin tilaan esimerkiksi komponentin asetuttua näkymään (componentDidMount) tai komponentin poistuessa näkymästä (componentWillUnmount). Muokkaukset tilaan tehdään komponentin setState-metodilla. Komponentin ulkoasun päivitystä eli renderöintiä varten komponentilla tulee olla render-metodi, joka ajetaan välittömästi tilan muuttuessa setState-metodin kutsusta. Tällöin komponentin uusi ulkoasu muodostetaan komponentin muuttuneen tilan mukaisesti. (Facebook Inc. 2019f.)

Luokkapainotteisia komponentteja käytetään yleensä aina, kun komponentilla pitää olla oma tila, jossa dataa säilytetään. Esimerkiksi sovelluksen tekstikenttä- ja lomakekomponentit tarvitsevat tilan käyttäjän syöttämän tekstin säilyttämiseen. Luokkapainotteista komponenttia tarvitaan myös yleensä tapauksissa, joissa on hyödynnettävä elinkaarimetodeja jonkin toiminnon suorittamiseen. Esimerkiksi kuvitteellinen komponentti, jonka tehtävä on hakea ja listata uusimmat uutiset, kutsuu näkymään asettuessaan elinkaarimetodia (componentDidMount), joka hakee tietokannasta uusimmat uutiset ja lopulta asettaa haetut uutiset osaksi komponentin tilaa kutsumalla komponentin setState-metodia. Tässä esimerkissä komponentti myös renderöityy aluksi käyttöliittymään ilman uutisia, mutta haun päättyessä ja setState-metodin kutsusta se renderöityy uudelleen listaten haetut uutiset.

Funktionaaliset komponentit ovat JavaScript-funktioita, jotka vastaanottavat niille annetut ominaisuudet[1] (props) funktion parametreina ja renderöivät komponentin ulkoasun funktiossa. Funktionaaliset komponentit ovat yleensä

[1] React-komponenteille voidaan antaa dataa ennen niiden renderöintiä asettamalla komponenteille ominaisuuksia JSX:n avulla. Komponenteille annetut ominaisuudet ovat tallennettuina komponentin sisäisessä props-muuttujassa. (Facebook Inc. 2019a.)

hyödyllisiä, kun komponentteja halutaan käyttää vain datan näyttämiseen tai yksinkertaisten käyttöliittymäelementtien tuottamiseen. Funktionaalisilla komponenteilla ei ole omaa tilaa tai elinkaarimetodeja, mikä rajoittaa niiden käytettävyyttä luokkapainotteisiin komponentteihin verrattuna. Hyötyinä funktionaalisissa komponenteissa on niiden kevyt rakenne ja helppo uudelleenkäytettävyys. Funktionaalisia komponentteja voi hyödyntää etenkin tapauksissa, jossa täytyy monistaa useita samanlaisia komponentteja käyttöliittymään. (Facebook Inc. 2019a.)

Esimerkiksi aiemmin kuvailtu uutiset listaava kuvitteellinen luokkapainotteinen komponentti voisi listata komponentteja, joiden tehtävä on näyttää vain yksittäisten uutisten otsikot ja linkit. Koska uutisia on paljon, yksittäisen uutisen näyttävä funktionaalinen komponentti toistuu listauksessa useita kertoja ja kevyen rakenteen ansiosta se on parempi vaihtoehto yksittäisen uutisen tietojen renderöintiin luokkapainotteiseen komponenttiin verrattuna. Kyseinen komponentti ei siis tarvitse omaa tilaa eikä elinkaarimetodeja, vain yksittäisen uutisen tiedot ominaisuuksina.

3.4 Tilanhallinta Redux-kirjastolla

React on yksinomaan tehokas käyttöliittymien toiminnallisuuden toteutuksessa. Se ei varsinaisesti kuitenkaan sisällä valmiita työkaluja datan prosessointiin sovelluksen tasolla ja useasti sovelluksen tilaa kuvastavia arvoja tulee pystyä säilyttämään yhdessä paikassa keskitetysti. Reactin kanssa integroituva Redux on kirjasto, joka vastaa täysin sovelluksen tilan ylläpidosta. Reduxin toiminnallisuus perustuu siihen, että yksittäisen *store-muuttujan* sisällä säilytetään koko sovelluksen tilaa kuvaavat arvot. Redux on täysin itsenäinen kirjasto ja sitä voidaan hyödyntää missä vain JavaScript-sovelluksessa. (Dan Abramov and the Redux documentation authors 2019.)

Reduxin store-muuttujaan tehdään muutoksia käyttämällä *actioneita* ja *reducereita*. Actionit ovat yksittäisiä toimintoja kuvastavia JavaScript-muuttujia, joilla on yleensä jokin käyttäjän määrittelemä tyyppi ja datakuorma. Reducerit

ovat funktioita, jotka aktivoituvat actionien lähetyksestä ja päivittävät niille osoitetun osan sovelluksen tilaa. Actioneita lähetetään store-muuttujan *dispatch-funktiolla*, jolloin action toimitetaan sille osoitettuun reducer-funktioon. Reducer-funktio saa aina parametreikseen reducerin edellisen tilan ja dispatch-metodilla lähetetyn actionin. Reducerit palauttavat aina uuden osan sovelluksen tilaa ja niissä voidaan sulauttaa actionien datakuormia osaksi store-muuttujaa. (Grider 2016.)

Esimerkiksi kuvitteellisessa sovelluksessa blogikirjoituksia palauttava reducer vastaanottaa actionin datakuormasta uuden blogikirjoituksen datan, jolloin reducer sulauttaa sen osaksi aiempia blogikirjoituksia. Lopulta reducer palauttaa niistä muodostetun uuden kokoelman blogikirjoituksia ja ne sulautetaan store-muuttujaan, eli niistä tulee osa sovelluksen tilaa. Blogikirjoituksia voidaan lukea kaikkialla sovelluksessa store-muuttujan kautta.

Redux integroituu Reactin kanssa poikkeuksellisen hyvin, sillä komponentit voidaan kytkeä osaksi Reduxin store-muuttujaa, jolloin React-komponentit renderöityvät itsenäisesti sovelluksen tilan muuttuessa. React-komponenteista voidaan myös lähettää actioneita hyödyntämällä store-muuttujan dispatch-funktiota, jolloin voidaan tehdä sovelluksen tilaan muutoksia. (Grider 2016.)

4 Firebase

Firebase on Googlen ylläpitämä pilvipalvelualusta, joka tarjoaa kehittäjille valmiina yleiset palvelinrajapinnan toiminnot ja auttaa kehittäjiä alkuun nopeasti sovellusten kehityksessä, skaalaamisessa ja ylläpidossa. Firebasessa on valmiina käyttäjien kirjautuminen, tietokannat, toiminnot tiedostojen tallennukselle sekä julkaisu ja ylläpitotoiminnot. Firebasen palvelukirjo kasvaa jatkuvasti ja se tukee useita eri alustoja, esimerkiksi web-, Android- ja iOS-pohjaisia sovelluksia. (Firebase 2016.)

Firebasen kustannukset on jaoteltu palvelukohtaisesti. Uusissa Firebase-projekteissa on alustavasti aktiivisena Spark-maksusuunnitelma, jonka mukaan käyttö on asiakkaalle ilmaista, mutta palveluiden käyttöä on rajoitettu. Esimerkiksi Firebasen Firestore-tietokannan (ks. luku 6.2) käyttöä on Spark-suunnitelmassa rajattu niin, että tietokannassa voi olla korkeintaan yhden gigatavun verran dataa tallennettuna. Firebasessa voidaan myös aktivoida Flame-maksusuunnitelma, jossa sallittu käyttöaste on palveluiden osalta suurempi ja kiinteä kuukausihinta on tasan 25 Yhdysvaltain dollaria. Poiketen aiemmista suunnitelmista, Blaze-maksusuunnitelmassa asiakas voi itse määrittää kullekin palvelulle käyttörajan ja täten asiakas maksaa Firebasen palveluista käytön mukaisesti. Esimerkiksi Blaze-suunnitelmalla Firebasen asiakas maksaisi Firestore-tietokannan datasta 0,18 Yhdysvaltain dollaria jokaista tietokantaan tallennettua gigatavua kohden kuukaudessa. (Google Developers 2019l.)

Firebase edustaa osittain uudenlaista ohjelmistokehityksen arkkitehtuurimallia, jota kutsutaan *serverless-arkkitehtuuriksi*. Mike Robertsin artikkelin ”Serverless Architectures” mukaisesti serverless-arkkitehtuuria noudattavien sovellusten palvelinrajapinnan toiminnallisuus on lähes täysin kolmannen osapuolen pilvipalveluiden tarjoamaa, jolloin ”palvelimettoman” arkkitehtuurin kuvaus täyttyy. Serverless-arkkitehtuuri hakee käsitteenä vielä paikkaansa, mutta Robertsin artikkelissa esitetään arkkitehtuurille kaksi lievästi eriävää määritelmää: *BaaS* (Back-end as a Service) ja *FaaS* (Functions as a Service). (Roberts 2018.)

BaaS-määritelmän mukaan palvelinrajapinta on valmiiksi toteutettu ja sen ylläpito tapahtuu täysin tai osittain kolmannen osapuolen toimesta pilvipalveluna. BaaS-palveluihin voidaan integroida useita eri sovelluksia ja kehityksessä voidaan painottua enemmän sovellustasolle. (Roberts 2018.)

FaaS-määritelmässä kehittäjä tuottaa itse yksittäisiä funktioita, joita ylläpidetään toisistaan eristetyissä ympäristöissä. Funktiot vastaavat palvelinrajapinnan toiminnallisuutta, mutta ne on hajautettu omiin kokonaisuuksiinsa toimimaan yksittäisinä pienimuotoisina palveluina, eli yksittäistä funktiota voidaan hyödyntää useassa järjestelmässä. (Roberts 2018.)

Firebase palveluna vastaa osittain molempia edellä mainittuja määritelmiä, sillä Firebase-toimii BaaS-määritelmän mukaisena palvelinrajapintana, sekä tarjoaa mahdollisuuden hyödyntää FaaS-määritelmän mukaisia funktioita. Firebasen eri ominaisuuksia käsitellään tarkemmin luvussa 6.

5 Lähtötilanne

Projektin aluksi kartoitettiin vaatimukset projektissa toteutettavista ominaisuuksista toimeksiantajan kanssa. Kalenterin vaadittavat ominaisuudet ja niiden pohjalta luodut aika-arviot kirjattiin Trello-projektinhallintajärjestelmään, jonka avulla toimeksiantaja pystyi seuraamaan projektin etenemistä. Työskentelyä dokumentoitiin Toggl-sovelluksen avulla, jolloin pystyttiin seuraamaan aika-arvioiden toteutumista.

Teknistä toteutusta varten tarvittiin kaksi erillistä Firebase-projektia. Ensin luotiin kehityskäyttöä ja testausta varten oma projekti ja varsinaista julkaistavaa sovellusta varten tehtiin erillinen projekti, jota toimeksiantaja tulee käyttämään sovelluksen julkaisun jälkeen. Käyttämällä eri ympäristöjä voidaan mahdollistaa järjestelmän kehittäminen sen ollessa jo käytössä ja varmistaa, ettei testaukseen käytettävä data sekoitu tuotannossa käytettävään dataan.

5.1 React-sovelluksen luominen

React-sovellus koostuu kokonaisuudessaan selaimelle toimitetuista tiivistetyistä kooditiedostoista ja muista resursseista, joiden kautta sovellus käynnistetään selaimessa. Varsinaisessa selainsovelluksessa käytettävä ohjelmakoodi on siis automaattisesti generoitua, mutta sisältää ne toiminnallisuudet, mitä kehityksessä tuotetaan kirjoittamalla Reactin ja JSX-syntaksin mukaista koodia.

Projektin alustamisessa käytettiin Create React App -pakettia, joka sisältää tarvittavat valmiit asetukset koodin kääntämiselle, React-sovelluksen kehityspalvelimen ylläpitämiselle sekä tuotantoon vietävän sovelluksen rakentamiselle. Create React App on NPM-paketinhallintatyökalulla ladattava paketti, joten kehityksessä käytettävällä tietokoneella tulee olla asennettuna Node.js-ohjelmisto ja NPM-paketinhallintatyökalu ennen kuin pakettia voidaan käyttää. (Facebook Inc. 2019b.)

Create React App asennetaan ajamalla komentokehotteessa komento ”npm install –g create-react-app”. React-sovellus luodaan Create React App -paketin komennolla ”create-react-app <sovelluksen nimi>”. Komennossa ilmoitetaan sovelluksen nimi, jolla sovellus sen hetkiseen kansioon generoidaan. Sovelluksen kansiossa on package.json-tiedosto, mihin tallennetaan esimerkiksi sovelluksessa käytettävien pakettien versionumerot, sovelluksen yleisiä tietoja sekä kehittäjän määrittelemiä komentoja. Create React App asettaa valmiiksi sovelluksen kehitystä varten tarvittavat paketit ja komennot, joiden avulla sovelluksen kehityspalvelin voidaan käynnistää sekä luoda lopullinen optimoitu sovellus. (Facebook Inc. 2019c.)

React-sovelluksen kehitys tapahtuu suurimmaksi osaksi Create React App -paketin generoiman src-kansion sisällä. Sovelluksen käynnistystiedostona toimii kansiossa oleva index.js-tiedosto. Kansiossa on myös valmiiksi generoitu App-komponentti, jonka sisään yleensä asetetaan kaikki React-sovelluksen komponentit. (Facebook Inc. 2019c.)

5.2 Sovelluksen navigaatio

React-sovellus on yhden sivun sovellus (SPA), joten sovelluksen linkkien välinen navigointi ei lataa joka kerta uutta sivua palvelimelta. React-sovelluksessa linkkien toiminnallisuutta mallinnetaan erillisellä React Router DOM -paketilla. Paketin avulla voidaan toteuttaa sovelluksen *reititys*, eli suunnitella sovelluksen alasivujen välinen navigaatio linkkien kautta.

Reititys voidaan toteuttaa React Router DOM -paketin avulla eri tavoin, mutta tässä sovelluksessa käytettiin paketin sisältämää *BrowserRouter-komponenttia* reitityksen toteuttamiseksi. BrowserRouter-komponentin tulee sisältää kaikki sovelluksen komponentit, missä hyödynnetään React Router DOM -paketin toiminnallisuutta. Täten BrowserRouter-komponentti on suoraan ensimmäinen alikomponentti sovelluksen juurikomponentille (App-komponentti) ja BrowserRouter-komponentin sisälle rakennetaan koko sovelluksen komponenttipuu. (React Training 2019.)

BrowserRouter-komponentin sisälle rakennetaan jokaiselle sivuston navigoitavalle alasivulle omat komponentit, jotka renderöivät itse alasivun sisällön. Reititys eri alasivukomponentteihin toteutetaan React Router DOM -paketin *Route- ja Switch-komponenteilla*. Route-komponentti tarkistaa, mikä on käyttäjän sijainti sivustolla sovelluksen polun[2] mukaan ja renderöi kyseiselle polulle osoitetun komponentin. Switch-komponentti koostuu Route-komponenteista ja sen tehtävä on renderöidä sovelluksen polun mukaisesti ensimmäinen yksittäinen Route-komponentti, jolle on määritelty vastaava polku. Route-komponentin polku määritellään komponentin *path-ominaisuuden* avulla ja sen renderöitävä komponentti määritellään *component-ominaisuuden* avulla. Route-komponentille voidaan määritellä myös *exact-ominaisuus*, jolloin sille ominaisuutena annettu komponentti renderöidään vain, jos path-ominaisuus vastaa sovelluksen aktiivista polkua täysin. Route-komponentille ei myöskään tarvitse määritellä polkua lainkaan, jolloin sen komponentti renderöidään aina, kun polku ei sovi muihin Route-komponentteihin. Esimerkki reitityksestä sekä Switch- ja Route-komponenttien käytöstä löytyy kuvasta 2. (React Training 2019.)

[2] Polulla (path) viitataan käyttäjän sijaintiin verkossa tai verkkosivulla URL-osoitteen mukaan.

```
/* Sivuston eri polut ja niille osoitetut komponentit */
const Routes = () => (
    <React.Fragment>
        <Switch>
            <Route exact path="/" component={HomePage}></Route>
            <Route exact path="/new" component={CreateEventPage}></Route>
            <Route exact path="/event/:eventid" component={EventSinglePage}></Route>
            <Route exact path="/admin" component={AdminPage}></Route>
            <Route component={NotFoundPage}></Route>
        </Switch>
    </React.Fragment>
);
```

Kuva 2. Switch-komponentti ja Route-komponentit.

Kokonaisuudessaan sovellus koostuu vain muutamasta alasivusta, joiden välille navigaatio toteutettiin. Sovelluksen pääsivulla listataan tapahtumat, joiden kautta voidaan navigoida yksittäisen tapahtuman tiedot näyttävälle sivulle. Navigointi yksittäiseen tapahtumaan toteutettiin niin, että sivuston URL-osoitteeseen sisällytetään tunniste, jolla tapahtuma on yksilöity tietokantaan. Yksilöivä tunniste muodostetaan jokaiselle tapahtumalle erikseen tapahtuman nimestä. Sivustolle rakennettiin myös järjestelmän ylläpitäjille oma sivu, minkä kautta tapahtumia voidaan hallita. Ylläpitäjän sivulle navigointi vaatii järjestelmään kirjautumisen. Sovelluksen eri polut on kuvattu tarkemmin taulukossa 1.

Taulukko 1. Sovelluksen eri polut ja niille osoitetut alasivut.

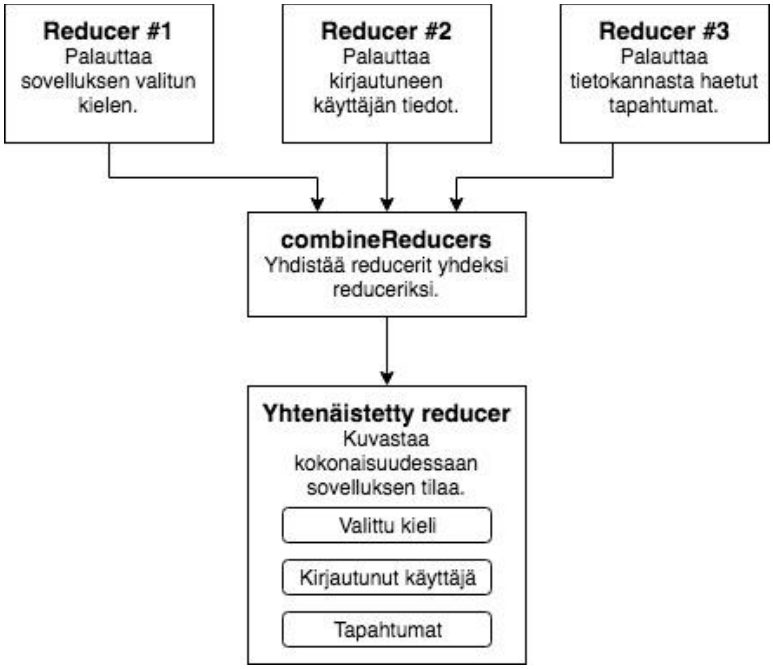
Polku	**Sivu**
`/`	Pääsivu
`/new`	Tapahtuman lisäyslomakkeen sivu
`/event/<tapahtuman tunniste>`	Yksittäisen tapahtuman sivu
`/admin`	Ylläpitäjän sivu
`/yksityisyys`	Sovelluksen tietosuojaseloste
`* (määrittelemätön polku)`	404-sivu

5.3 Reduxin asetukset ja storen luominen

React-komponentteja voidaan yhdistää Reduxin store-muuttujaan, missä ylläpidetään sovelluksen tilaa kuvastavaa dataa. React-komponentit renderöityvät täten itsenäisesti sovelluksen tilan muuttuessa, jolloin

komponenttien kontrollointi tapahtuu keskitetysti Reduxin avulla. Reactin ja Reduxin integraatiota varten tarvitaan erillinen React Redux -paketti, joka sisältää kirjastojen integraatiota helpottavia toimintoja. (Grider 2016.)

Sovelluksen käynnistysvaiheessa on Reduxille määriteltävä uusi store-muuttuja, missä ylläpidetään sovelluksen tilaa kuvastavaa dataa. Uusi store-muuttuja luodaan Reduxin *createStore-funktion* avulla. Funktio vaatii ensimmäisenä parametrinaan reducerin, joten ennen funktion kutsumista tulee määritellä vähintään yksi reducer-funktio, joka vastaa tilan palauttamisesta. Reducereita voidaan luoda kerralla useampi vastaamaan sovelluksen tilan useista eri osista ja ne voidaan yhdistää yhdeksi yhtenäiseksi reduceriksi Reduxin *combineReducers-funktion* avulla, jonka toimintaa on esitelty kuvassa 3. (Grider 2016.)



Kuva 3. CombineReducers-funktion toiminta

CreateStore-funktion toisena parametrina voidaan antaa storen alustava tila. Tässä sovelluksessa alustavalle tilalle ei ole tarvetta, joten toisena parametrina annetaan sovelluksen käynnistyessä tyhjä JavaScript-muuttuja. Kolmannessa parametrissa storeen voidaan kytkeä sen toimintaa laajentavia funktioita, joita kutsutaan Reduxin *väliohjelmistoksi* (middleware). Tässä sovelluksessa hyödynnettiin Reduxin kahta väliohjelmistoa: Redux Thunk ja Redux Logger.

Redux Thunk mahdollistaa funktioiden lähettämisen actioneiden lisäksi storen dispatch-funktiolle, mikä mahdollistaa asynkronisten toimintojen[3], kuten tietokantahakujen ja tiedostojen tallennuksen käsittelyn. Redux Logger -väliohjelmistoa hyödynnettiin vain kehitysvaiheessa sovelluksen tilan muutosten monitorointiin. Väliohjelmistot sulautetaan yhteen Reduxin applyMiddleware-funktion kutsussa, mikä annetaan suoraan kolmantena parametrina createStore-funktiolle. Store-muuttujan luonti koodissa on toteutettu kuvassa 4. (Grider 2016.)

```
/* Tuodaan tarvittavat funktiot storen alustamiseksi */
import { createStore, applyMiddleware } from 'redux';

/* Tuodaan storen middlewaret */
import thunk from 'redux-thunk';
import logger from 'redux-logger';

/* Yhdistetty reducer (luotu combineReducers-funktiolla) */
import reducers from './reducers/index';

/* Storen luominen createStore-funktiolla */
export const store = createStore(
  reducers,
  {},
  applyMiddleware(thunk, logger)
);
```

Kuva 4. Store-muuttujan alustaminen koodissa.

Reactin ja Reduxin integraatiota varten React-sovelluksen komponentit tulee sisällyttää React Redux -paketin *Provider-komponentin* sisään, mikä mahdollistaa myöhemmässä vaiheessa komponenttien kytkemisen Reduxin store-muuttujaan. Myös aiemmin sovelluksen reititystä varten määritelty BrowserRouter-komponentti siirrettiin Provider-komponentin sisälle kuvassa 5 esitetyllä tavalla. Provider-komponentille annetaan sovelluksen käynnistyessä store-ominaisuuden arvoksi createStore-funktiolla luotu store-muuttuja, jolloin React-sovellus on kytketty osaksi Reduxia. (Grider 2016.)

[3] JavaScriptin asynkroniset toiminnot suoritetaan varsinaisen ohjelmakoodin suorituksen rinnalla, koska ne ovat yleensä pitkäkestoisia toimintoja. JavaScriptissä asynkronista koodia käsitellään callback-funktioiden tai Promise-luokan avulla (Mozilla ja yksittäiset avustajat 2019e).

```
/* App-luokka sisältää kokonaisuudessaan React-sovelluksen */
class App extends Component {
  render() {
    return (
      <Provider store={store}>
        <BrowserRouter>
          <Routes></Routes>
        </BrowserRouter>
      </Provider>
    );
  }
}
```

Kuva 5. App-luokka, joka toimii sovelluksen juurikomponenttina.

5.4 Firebase-projektin luominen ja integraatio

Firebase-projekti voidaan luoda Firebasen hallintakonsolissa osoitteessa https://console.firebase.google.com/. Projektin luominen vaatii Google-tunnuksen. Projektille valitaan luomishetkellä nimi, projektille yksilöity tunniste (”project ID”), asiakkaan organisaation sijainti ja Firestore-tietokannan sijainti. Firebase tuottaa käyttäjälleen tarvittaessa myös analytiikkatietoa, teknistä tukea sekä muita palveluita vastineeksi datan jakamisesta Firebasen kanssa. Projektin luomisen jälkeen käyttäjä ohjataan Firebase-projektin hallintasivulle, mistä voidaan tarkkailla projektin dataa ja hallita Firebasen palveluita. Hallintasivulta saadaan myös yksilölliset avaintunnukset (ks. kuva 6), joiden avulla projekti integroidaan osaksi React-sovellusta. (Google Developers 2019a.)

Add Firebase to your web app

Copy and paste the snippet below at the bottom of your HTML, before other `script` tags.

```
<script src="https://www.gstatic.com/firebasejs/5.8.6/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "",
    authDomain: "",
    databaseURL: "",
    projectId: "",
    storageBucket: "",
    messagingSenderId: ""
  };
  firebase.initializeApp(config);
</script>
```

Copy

Kuva 6. Avaintunnukset Firebasen hallintasivulla.

Integraatiota varten React-sovellukseen asennettiin paikallisesti NPM-paketinhallintatyökalulla Firebase SDK -paketti, mikä mahdollistaa Firebasen eri palveluiden käyttämisen sovelluksessa. Firebase SDK asennetaan komennolla ”npm install firebase --save”. Firebase SDK asennetaan ympäristökohtaisesti, joten paketin nimi ja versio tallennettiin projektin package.json-tiedostoon asennuskomennossa argumentilla ’--save’. (Google Developers 2019a.)

Firebasen asetuksia varten määriteltiin tähän sovellukseen oma tiedosto, jossa tuodaan Firebase SDK -paketista tarvittavat palvelut ja muodostetaan niiden pohjalta uusi ES6-standardin luokka, joka toimii sovelluksessa yhtenäisenä rajapintana Firebasen eri palveluille. Yhteys Firebaseen muodostetaan käyttämällä Firebase SDK:n sisältämää initializeApp-funktiota, jolle annetaan parametriksi Firebase-projektin avaintunnukset. Firebasen yksittäiset palvelut ovat alikansioina Firebase SDK -paketissa ja niitä voidaan tuoda koodiin yksittäisillä ES6-standardin import-lauseilla, kuten on esitelty kuvassa 7. Tällä tavalla vältetään myös selaimen konsoliin tulostuva varoitusviesti, missä ohjeistetaan välttämään koko Firebase SDK:n tuomista useita kertoja projektissa. (Google Developers 2019a.)

```
import firebaseApp from 'firebase/app';
import 'firebase/firestore';
import 'firebase/auth';
import 'firebase/functions';
import 'firebase/storage';
import { config, functionsURL } from './keys';

class Firebase {
    constructor() {
        /* Firebase-sovelluksen käynnistys */
        firebaseApp.initializeApp(config);

        /* Firebase-palveluiden asettaminen luokkakohtaisiksi muuttujiksi */
        this.firestore = firebaseApp.firestore();
        this.auth = firebaseApp.auth();
        this.functions = firebaseApp.functions();
        this.storage = firebaseApp.storage();

        /* Firebase-funktioiden osoite */
        this.functionsURL = functionsURL;
    }

    /* Firebasen palveluiden asetukset */
    init() {
        this.firestore.settings({ timestampsInSnapshots: true });
        this.auth.setPersistence(firebaseApp.auth.Auth.Persistence.LOCAL);
    }

    /* Firebase-palveluiden palautus */
    getFirestoreInstance() { return this.firestore; }
    getAuthInstance() { return this.auth; }
    getFunctionsInstance() { return this.functions; }
    getStorageInstance() { return this.storage; }

    /* Firebase-funktioiden URL-osoitteen palautus */
    getFunctionsURL() { return this.functionsURL; }
}

/* Luodaan uusi Firebase-luokan olio ja alustetaan se */
const firebase = new Firebase();
firebase.init();

/* Tiedosto palauttaa olion, joka toimii sovelluksessa rajapintana Firebasen palveluille */
export default firebase;
```

Kuva 7. Sovelluksessa hyödynnettävä Firebase-luokka soveltaen Robin Wieruchin ohjeartikkelia (Wieruch 2018).

6 Sovelluksen ominaisuudet

6.1 Kirjautuminen

Kalenterisovelluksen tapahtumien hallintaa varten sovellukseen tuli kehittää kirjautuminen, joka päästää käyttäjän sovelluksen hallintatilaan. Kirjautuneilla käyttäjillä on oikeudet hyväksyä, lukea, poistaa ja muokata kaikkia kalenterin tapahtumia. Ainoastaan kirjautuneen käyttäjän hyväksynnällä kalenteriin voidaan vahvistaa muiden käyttäjien lisäämiä tapahtumia.

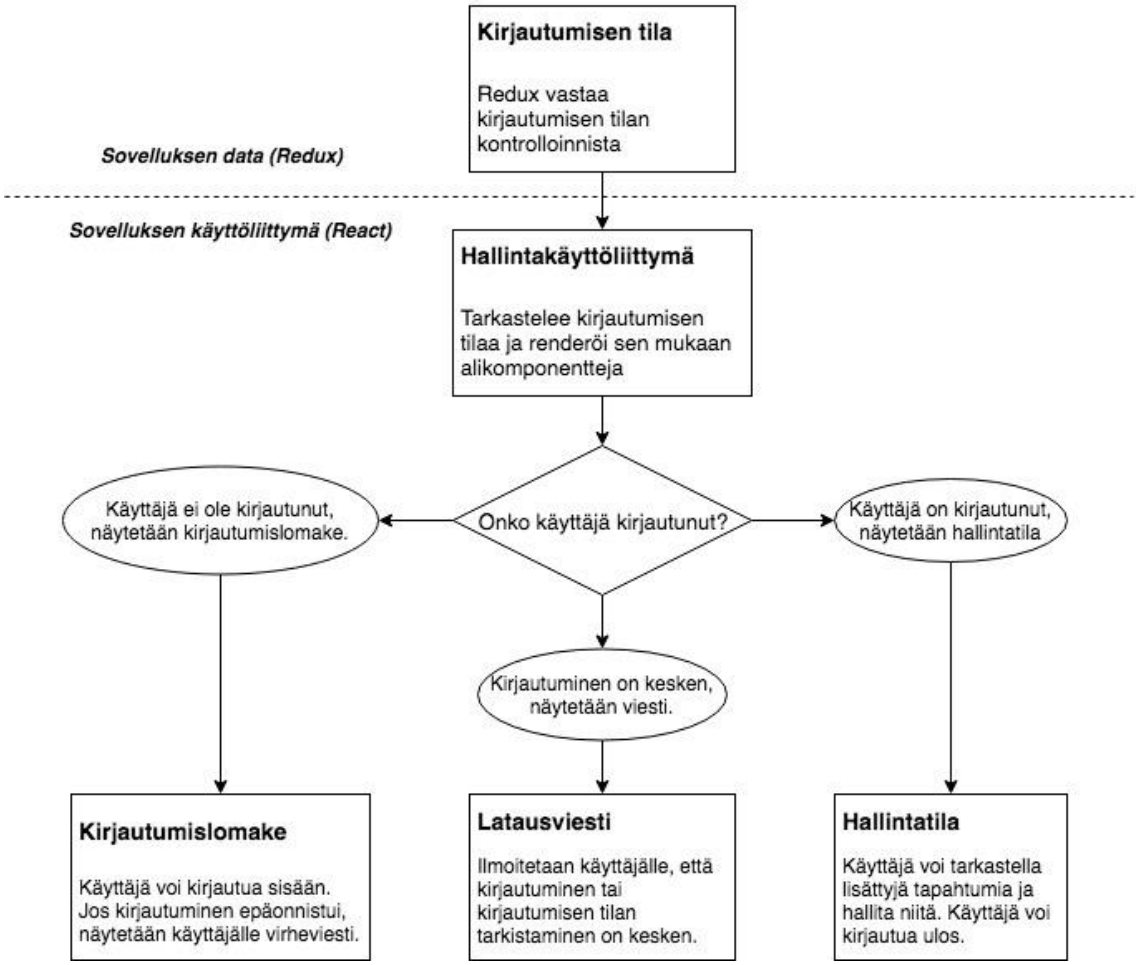
Firebase tukee valmiiksi useita eri kirjautumistapoja, kuten esimerkiksi Facebook-, Twitter-, Google- ja GitHub-tunnuksilla kirjautumisen. Eri kirjautumismetodit voidaan aktivoida Firebase-projektin hallintakonsolissa Authentication-välilehden kautta. Tässä sovelluksessa kirjautuminen on sallittua vain Firebase-projektin ylläpitäjän lisäämillä käyttäjätunnuksilla, joten projektiin kytkettiin ainoaksi kirjautumistavaksi sähköpostilla ja salasanalla kirjautuminen. Kun kirjautumistapa on aktivoitu Firebase-konsolissa, voidaan uusia käyttäjiä lisätä suoraan Users-välilehdellä antamalla uuden käyttäjän sähköposti ja salasana. (Google Developers 2019h.)

Sovelluksessa kirjautuminen käynnistetään kuvassa 8 esitellyn authenticateUser-funktion avulla, joka lähettää Reduxin actioneita kirjautumisen edetessä. Funktiota kutsutaan sovelluksen kirjautumislomakkeella, kun käyttäjä on kirjoittanut tunnuksensa ja käynnistänyt kirjautumisen lomakkeen lähetyksestä. Kirjautumisen alussa sovelluksen tilaan päivitetään kirjautumisen olevan vaiheessa ja kirjautumisen epäonnistuessa päivitetään kirjautumislomakkeelle virheviesti, jossa ilmoitetaan syy kirjautumisen epäonnistumiselle.

```
/* Tämä funktio kirjaa käyttäjän sisään */
export const authenticateUser = (email, password) => async dispatch => {
    // Lähetetään action, joka ilmaisee kirjautumisen olevan käynnissä
    await dispatch(authenticatingUser());
    // Aloita kirjautuminen Firebasen kirjautumispalvelun avulla.
    // Jos kirjautuminen onnistuu, React komponentit renderöityvät itsestään
    // tilan muuttuessa.
    await auth.signInWithEmailAndPassword(email, password).catch(e => {
        // Tänne mennään virheen tapahtuessa
        console.log(e);
        // Lähetetään action, joka ilmaisee kirjautumisen epäonnistuneen
        dispatch(authenticationFailed(e.message));
    });
}
```

Kuva 8. Kirjautumisen käynnistävä funktio.

Sovelluksessa kuunnellaan kirjautumisessa tapahtuvia muutoksia jatkuvasti. Kuvassa 9 kuvataan, miten eri käyttöliittymäkomponentteja renderöidään käyttöliittymään kirjautumisen tilan muuttuessa. Tilan muutoksia kontrolloidaan Reduxin actionien avulla.



Kuva 9. Kirjautuminen hallintakäyttöliittymään sovelluksessa.

Kirjautumisen toteutuksessa ilmeni myös ongelma istunnon pysyvyydessä: kun sivu päivitetään selaimessa, kirjautumisistunto ei ole enää voimassa. Tämä johtuu siitä, että selaimen JavaScript-koodissa voimassa oleva kirjautumisistunto katoaa, kun sovelluksen JavaScript-koodi suoritetaan selaimessa uudelleen sivun päivittyessä. Tätä varten sovelluksessa on oltava toiminto, jonka avulla voidaan tarkistaa kirjautumissession voimassaolo sivun päivittyessä. Firebasen kirjautumispalvelu sisältää tätä varten onAuthStateChanged-funktion, jolle voidaan antaa parametriksi *kuuntelijafunktio*, joka suoritetaan aina kirjautumisistunnon muuttuessa. Kun kirjautumisistunto muuttuu, kuuntelijafunktio päivittää sovelluksen tilan sekä tilan päivittyessä renderöi uudelleen käyttöliittymän komponentit. OnAuthStateChanged-funktio aktivoidaan hallintakäyttöliittymän renderöityessä, jonka jälkeen se autonomisesti kutsuu kuuntelijafunktiota. (Google Developers 2019h.)

Firebasen kirjautumispalvelulle voidaan erikseen määrittää kirjautumisistunnon *pysyvyys* (persistence). Pysyvyydellä tarkoitetaan sitä, miten kauan kirjautumisistunto voi olla voimassa ja mitkä eri toiminnot päättävät voimassa olevan istunnon. Istunnon pysyvyystyyppejä on Firebasessa yhteensä kolme:

1. LOCAL: Istunto päättyy selaimen sulkeutumisesta tai erillisestä uloskirjautumisesta.
2. SESSION: Istunto päättyy selaimen välilehden sulkeutumisesta.
3. NONE: Istunto säilytetään vain sovelluksen välimuistissa ja se päättyy sivuston päivittyessä tai muuten sovelluksen suorituksen lakkautuessa.

Kirjautumisistunnon pysyvyystyypiksi sovellukseen määriteltiin SESSION. Istunnon pysyvyys asetetaan sovellukseen aiemmin luvussa 5.4 esitetyn Firebase-luokan init-metodissa ja se näytetään tarkemmin kuvassa 10. (Google Developers 2019b.)

```
init() {
    this.firestore.settings({ timestampsInSnapshots: true });
    /* Kirjautumisistunnon pysyvyyden määritys */
    this.auth.setPersistence(firebaseApp.auth.Auth.Persistence.SESSION);
}
```

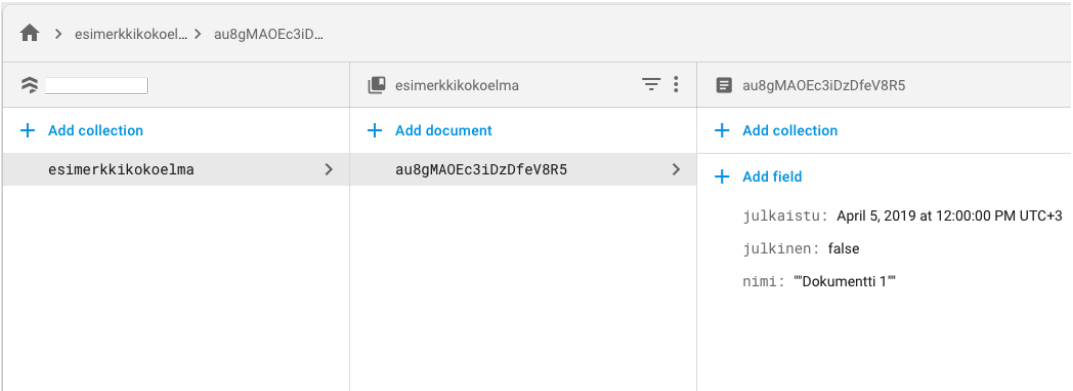
Kuva 10. Kirjautumisistunnon pysyvyyden määritys Firebase-luokassa.

6.2 Tietokanta

6.2.1 Firestoren perustoiminnot

Firebase-projektissa on mahdollista käyttää kahta erilaista tietokantaa: Realtime Database -tietokantaa, joka päivittää aktiivisesti muutoksia kaikille tietokantaa kuunteleville sovelluksille sekä Firestore-tietokantaa, jossa data tallennetaan dokumentteihin niille osoitettujen kokoelmien sisälle. Tässä toimeksiannossa hyödynnetään vain Firestore-tietokantaa, sillä data tuli saada jaoteltua kokoelmakohtaisesti.

Firestoren dokumentit ovat *skeemattomia*, eli niille ei erikseen määritellä dokumentin tai kokoelman rakennetta kuvaavaa skeemaa ennen dokumenttien tallentamista ja dokumenttien rakenne voi vaihdella saman kokoelman sisällä. Firestoreen voidaan luoda dataa suoraan Firebasen hallintakonsolin kautta (ks. kuva 11) tai sovelluksen kautta. Edellytyksenä uuden dokumentin luomiselle on se, että uusi dokumentti sisältää dataa jossain muodossa ja dokumentilla tulee olla jokin tunniste (ID), minkä avulla dokumenttiin voidaan viitata. Firestore luo automaattisesti dokumentille ID:n, jos sellaista ei sille erikseen määritellä. Firestoren automaattinen ID on kirjaimista ja numeroista koostuva yksilöllinen merkkijono. (Google Developers 2019c.)

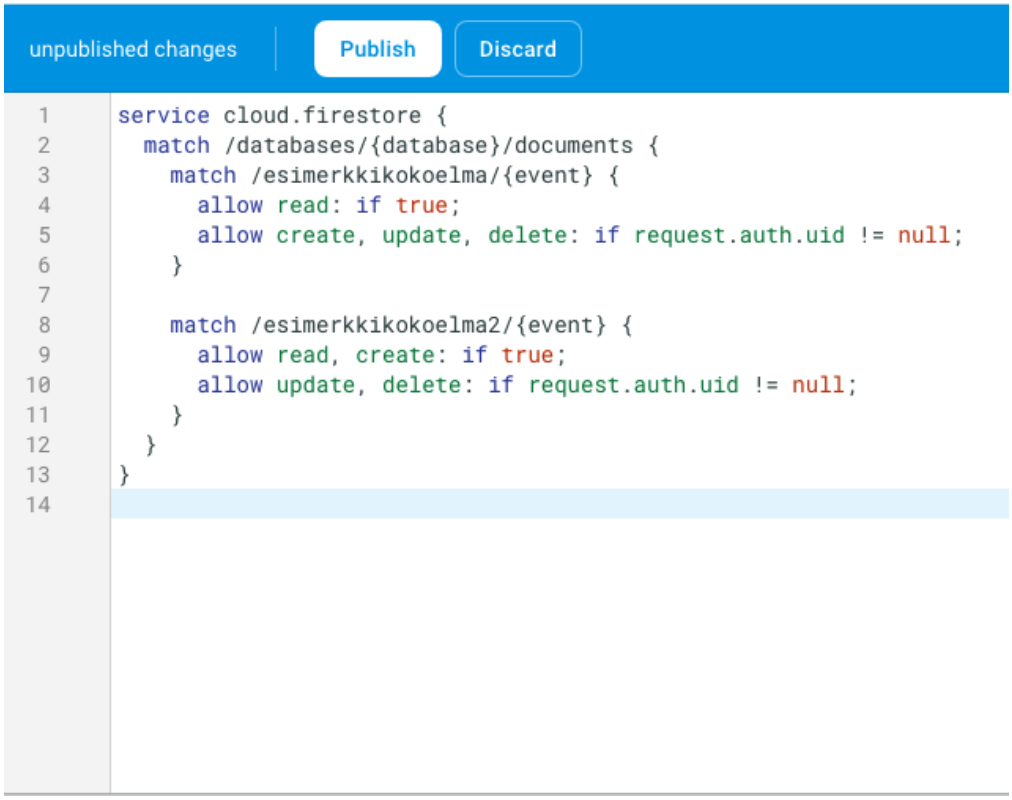


Kuva 11. Firestore-tietokannan hallintanäkymä.

Vaikka dokumenttien rakenteen voi määrittää sovelluksen suorituksessa dokumenttikohtaisesti, itse kokoelmia varten tulee määritellä erikseen *turvallisuussäännöt* ennen kuin uusia kokoelmia luodaan. Jos kokoelmakohtaisia sääntöjä ei ole erikseen määritelty Firebasen hallintakonsolissa uudelle kokoelmalle, kokoelman datan lukeminen ja kirjoittaminen on oletuksella kiellettyä Firestorea käyttävissä sovelluksissa. Firebasen hallintakonsolin Database-sivun Rules-välilehdellä voidaan määritellä säännöt tietokantakokoelmalle Firebasen omassa sääntöeditorissa. Säännöillä on omanlainen Firebasen määrittelemä syntaksi, jolla käyttöä voidaan rajoittaa kokoelmien ja dokumenttien polkujen mukaisesti. (Google Developers 2019n.)

Kuvassa 12 on määritelty esimerkkisäännöt kahdelle esimerkkikokoelmalle. Ensimmäisen kokoelman säännöissä sallitaan dokumenttien luominen, päivittäminen tai poistaminen vain, jos käyttäjä on kirjautunut sisään, mutta

dokumenttien lukeminen sallitaan kaikille. Toisen esimerkkikokoelman säännöissä vuorostaan sallitaan dokumenttien päivittäminen ja poistaminen vain kirjautuneille käyttäjille, mutta luku- ja kirjoitusoikeus on kaikilla käyttäjillä. Säännöt voidaan asettaa voimaan Publish-painikkeella.



Kuva 12. Firestore-tietokannan sääntöeditori Firebasen hallintakonsolissa.

6.2.2 Kokoelma- ja dokumenttiviitteet

Tietokannan dataa luetaan sovelluksessa lähettämällä Firestoren omia *tietokantakyselyitä*. Kyselyt muotoillaan Firebase SDK:n sisältämän Firestore-palvelun avulla. Firestoren dokumentaatiossa ohjeistetaan tekemään tietokantakyselyitä niin, että Firestoren kokoelmasta tai kokoelman dokumentista tehdään *viite*, jolle voidaan erikseen asettaa rajaavia ehtoja sekä suorittaa erilaisia tietokantatoimintoja. (Google Developers 2019c.)

Esimerkiksi jos Firestoresta halutaan hakea kuvitteellinen kokoelma kaupungeista, se saadaan haettua tekemällä kokoelmasta viite koodissa seuraavasti (db-muuttuja on Firestore-palvelu):

```
let citiesReference = db.collection(’cities’);
```

Yllä olevassa koodirivissä asetetaan muuttujan arvoksi *kokoelmaviite* (Collection Reference), joka viittaa suoraan Firestoren kokoelmaan, jolle on annettu nimeksi ”cities”. Yllä olevaa esimerkkiä voidaan laajentaa lisäämällä siihen *where-hakuehto*, jonka mukaan haetaan vain kaupungit, joiden ”country”-kentän arvo on ”Finland”. Kyseinen ehto asetetaan jatkamalla edellisen esimerkin jälkeen seuraavasti:

```
citiesReference = citiesReference.where(’country’, ’==’,
’Finland’);
```

Jos esimerkin kokoelmaan on tallennettu dokumentteina kaikki kaupungit niin, että kaupungin nimi on dokumentin ID, voidaan kokoelmasta hakea yksittäinen dokumentti esimerkiksi seuraavasti:

```
const cityReference = db.collection(’cities’).doc(’helsinki’);
```

Yllä olevassa koodirivissä muuttujan arvoksi asetetaan *dokumenttiviite* (Document Reference). Tämä viittaa yksittäiseen dokumenttiin, joka voi olla tallennettuna kokoelmaan. Dokumenttiviitettä voidaan myös käyttää, vaikka sitä ei olisi vielä tallennettu kokoelmaan. Dokumenttiviite toimii tällöin viitteenä uuteen dokumenttiin, joka voidaan tallentaa kyseiseen kokoelmaan. (Google Developers 2019c.)

Varsinaiset tietokantakyselyt suoritetaan kutsumalla kokoelma- tai dokumenttiviitteiden metodeja. Esimerkiksi kokoelmaviitteen avulla voidaan hakea kaikki kokoelman dokumentit kutsumalla viitteen get-metodia ja kokoelmaan voidaan lisätä uusi dokumentti kutsumalla kokoelmaviitteen add-metodia, jolle annetaan parametriksi tallennettavan dokumentin data.

Dokumenttiviitteen metodeilla voidaan toteuttaa yhtä dokumenttia kohden eri toimintoja, kuten esimerkiksi update-metodilla voidaan päivittää olemassa olevan dokumentin dataa tai delete-metodilla voidaan poistaa dokumentti.

6.2.3 Tietokannan käyttö sovelluksessa

Lopullisessa kalenterisovelluksessa Firestore-tietokannassa on käytössä kolme eri kokoelmaa, joihin dataa tallennetaan:

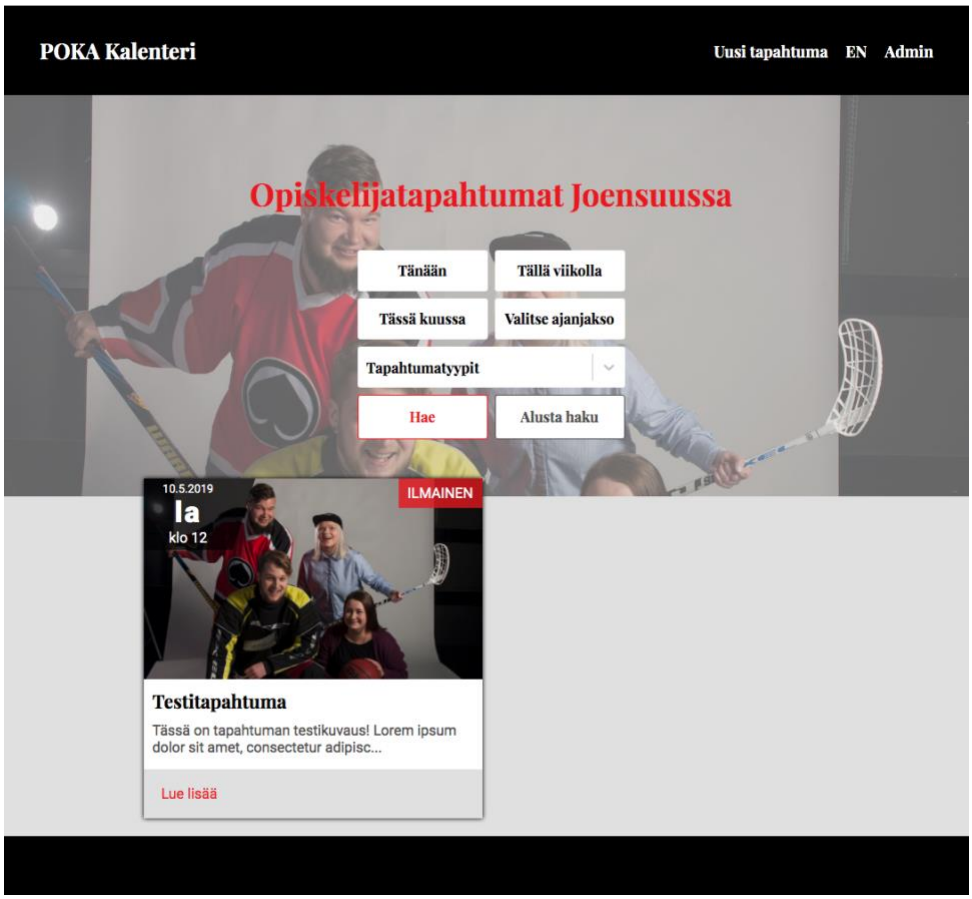
1. **Hyväksymättömät tapahtumat:** Sovelluksen käyttäjien lisäämät tapahtumat, joita ei ole vielä hyväksytty ylläpitäjien toimesta. Hyväksymättömiä tapahtumia voidaan tarkastella vain hallintanäkymässä.
2. **Hyväksytyt tapahtumat:** Ylläpitäjien hyväksymät tapahtumat, jotka näytetään kalenterin etusivulla. Kaikilla sivuston käyttäjillä on mahdollisuus tarkastella näitä tapahtumia.
3. **Yhteystiedot:** Jokaisen tapahtuman yhteydessä lisätään tapahtuman järjestäjän yhteystiedot. Ne on eriytetty omaan kokoelmaan, josta ne voidaan hakea vain hallintanäkymässä.

```
// Luodaan dokumenttiviite
const eventRef = await db.collection('pendingEvents').doc(id);
// Haetaan dokumentti get-metodilla
const eventDoc = await eventRef.get();
// Otetaan dokumentin sisältämä data eventData-muuttujaan
const eventData = eventDoc.data();
```

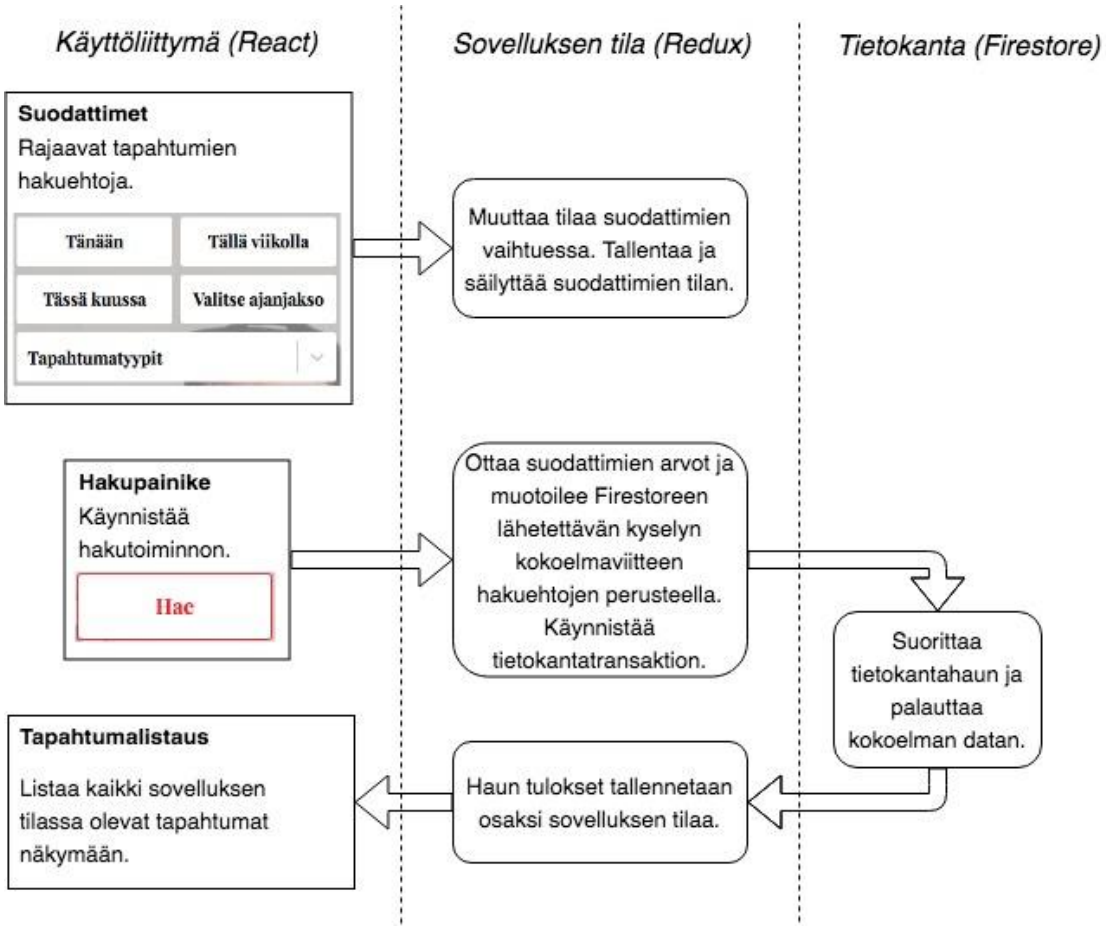
Kuva 13. Tapahtuman hakeminen Firestoresta.

Kuvassa 13 on esimerkki yksittäisen hyväksymättömän tapahtuman hausta sovelluksessa. Tietokantahauista tuli sovelluksessa suhteellisen monimutkaisia, koska niihin asetettiin myös erilliset hakuehdot tapahtuman ajankohtaa ja tyyppiä koskien where-ehdoilla. Hakuehtojen kontrollointi tapahtuu kuvassa 14 esitellyllä sovelluksen etusivulla käyttäjän toimesta aktivoimalla eri hakusuodattimia. Suodattimien arvot tallennetaan etusivulla käyttäjän painalluksista sovelluksen tilaan hakuehdoiksi ja haun käynnistyessä niiden avulla voidaan rajata haettavia

tapahtumia. Kuvassa 15 on esitettynä, kuinka tapahtumien hakeminen sovelluksessa toimii.



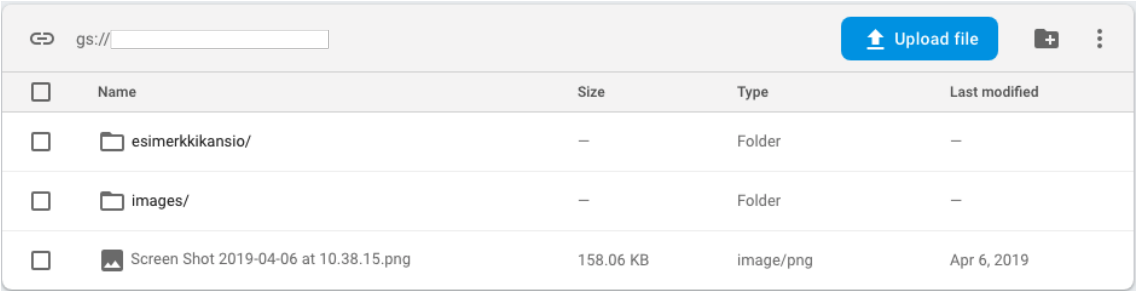
Kuva 14. Sovelluksen etusivu.



Kuva 15. Tapahtumien haku sovelluksen etusivulla.

6.3 Tiedostojen hallinta

Yksi kalenterisovelluksen vaatimuksista oli myös mahdollisuus lisätä tapahtumille kuvia tapahtuman luomisen aikana. Firebasessa on oma Cloud Storage -palvelu, jonka avulla käyttäjän tiedostoja voidaan tallentaa ja hallita Firebasessa. Cloud Storage -palvelua käytetään sovelluksessa samoin, kuten Firestorea ja tunnistautumista, eli Firebase SDK:n kautta. Tiedostoja voidaan myös lisätä Firebasen hallintakonsolin kautta Storage-välilehdeltä. Kuvassa 16 on hallintakonsolin tiedostoselain, jolla tiedostoja voidaan hallita. (Google Developers 2019d.)



Kuva 16. Cloud Storagen tiedostoselain Firebasen hallintakonsolissa.

Cloud Storagen tiedostoille tulee myös erikseen määrittää turvallisuussäännöt Firebasen hallintakonsolissa, jotka noudattavat lähes samaa syntaksia kuin Firestore-tietokannassa käytettävät turvallisuussäännöt, jotka esiteltiin luvussa 6.2.1. Turvallisuussäännöillä voidaan rajata esimerkiksi tiedostojen formaatin, koon, nimen ja muiden kuvan *metatietojen* perusteella tiedostojen hallintaa sekä säännöillä voidaan rajoittaa käyttöoikeutta vain kirjautuneille käyttäjille tiedosto- ja kansiokohtaisesti. Kuvan 17 esimerkissä on määritelty turvallisuussäännöt, jotka sallivat kaikilta käyttäjiltä kuvien lisäämisen alikansioon nimeltä ”images” sekä rajoitetaan tiedostotyypiksi kuvat ja tiedostojen maksimikooksi 5 megatavua. Esimerkin säännöt on otettu Firebasen Cloud Storage -palvelun dokumentaatiosta. (Google Developers 2019j.)

```
service firebase.storage {
 match /b/{bucket}/o {
   match /images {
     match /{allImages=**} {
       allow read;
     }

     match /{imageId} {
       allow write: if request.resource.size < 5 * 1024 * 1024
                    && request.resource.contentType.matches('image/.*')
                    && request.resource.contentType == resource.contentType
                    && imageId.size() < 32
     }
   }
 }
}
```

Kuva 17. Firebasen Cloud Storage -palvelun turvallisuussäännöt.

Tiedostojen lisäämistä varten sovelluksessa hyödynnettiin erillistä react-firebase-file-uploader -pakettia. Paketti sisältää valmiin React-komponentin, jonka avulla tiedoston lataaminen Firebaseen on toteutettavissa vähäisillä asetuksilla. Komponentille voidaan suoraan määritellä viite Cloud Storage -palvelun sisäiseen kansioon, johon tiedostot komponentin kautta tallennetaan. Komponentissa on myös valmiina metodeja, joiden avulla voidaan tarkkailla

tiedoston latauksen edistymistä ja suorittaa eri funktioita sen mukaan. Tiedoston lataus tapahtuu React-komponentin kautta suoraan Cloud Storage -palveluun ja latauksen onnistuessa tallennetun tiedoston viitteestä saadaan suora URL-osoite kuvatiedostoon. Jokaiselle tapahtumalle tallennetaan erikseen tapahtuman luonnin aikana tallennetun kuvatiedoston URL-osoite sekä tiedoston nimi, jonka avulla kuva voidaan poistaa Cloud Storagesta tapahtuman poistuessa. Komponentti on esitelty kuvassa 18. (Sprmn 2018.)

```
<FileUploader
    accept="image/*"
    name="fileName"
    randomizeFilename
    storageRef={storage.ref('images')}
    onUploadStart={this.handleUploadStart}
    onUploadError={this.handleUploadError}
    onUploadSuccess={this.handleUploadSuccess} />
```

Kuva 18. React-firebase-file-uploader-komponentti.

6.4 Sovelluksen julkaisu Hosting-palvelulla

Sovelluksen kooditiedostot voidaan julkaista käyttämällä Firebasen Hosting-palvelua. Täten sovellusta voidaan käyttää julkisesti verkon yli. Sovelluksen tiedostot lähetetään Hosting-palvelulle käyttämällä Firebasen komentorivityökalua, jolla voidaan määritellä julkaistavan sovelluksen kansio. Firebasen komentorivityökalut asennetaan NPM-paketinhallintatyökalulla komennolla ”npm install –g firebase-tools”. (Google Developers 2019i.)

Projektin JSX-syntaksilla kirjoitetut koodit on käännettävä selainten tukemaan muotoon ennen kuin ne voidaan lähettää Firebasen Hosting-palvelimelle. Valmis tuotannon sovellus rakennetaan komennolla ”npm run build”. Komento käynnistää prosessin, jossa sovelluksen koodit käännetään ja valmiin sovelluksen koodit asetetaan build-kansioon. Kansio voidaan suoraan toimittaa sellaisenaan palvelimelle, sillä sen juuresta löytyvästä index.html-tiedostosta React-sovellus käynnistetään selaimessa.

Firebasen komentorivityökalu vaatii käyttäjältä kirjautumisen ennen kuin sovellus voidaan julkaista. Komentorivityökalulla tunnistaudutaan ajamalla komento ”firebase login”, jonka jälkeen käyttäjä voi kirjautua selaimen kautta omilla Google-tunnuksilla. Tämän jälkeen asetukset Hosting-palvelua varten voidaan määrittää ajamalla komento ”firebase init” ja määrittämällä sen jälkeen asetukset yhden sivun sovelluksen osalta. Sovelluksen juurikansioon generoidaan komennon suorituksen aikana kaksi uutta tiedostoa: *.firebaserc* ja *firebase.json*. (Google Developers 2019i.)

Tiedostossa .firebaserc määritellään projektin *alias*, eli käytettävän Firebase-projektin nimi. Firebasessa on mahdollista käyttää useita eri projekteja samanaikaisesti, kuten myös tässä toimeksiannossa tehtiin ja Firebasen komentorivityökalulla voidaan lisätä jokaista Firebase-projektia kohden oma alias. Tiedostossa firebase.json määritellään julkaisun kannalta oleellisia asetuksia, kuten esimerkiksi tiedostot, joita ei lähetetä Firebase Hosting-palvelulle sekä projektin julkisten resurssien kansion polku. (Google Developers 2019i.)

Kun asetukset sovelluksen julkaisulle on määritelty, sovellus voidaan julkaista komennolla ”firebase deploy”. Tämän jälkeen sovellus on julkisesti käytettävissä ja komentokehotteeseen tulostuu sovelluksen URL-osoite. Firebasen hallintakonsolissa voidaan myös määrittää URL-osoitteen tilalle asiakkaan oma domain. (Google Developers 2019i.)

6.5 Cloud Functions

Firebasen Cloud Functions -palvelun avulla voidaan laajentaa Firebasen toiminnallisuutta lisäämällä palvelimelle kehittäjän määrittelemiä funktioita, joita voidaan kutsua Firebasen eri toimintojen laukaisemina tai Firebasen palvelimelle osoitetuista HTTP-pyynnöistä. Funktiot kirjoitetaan JavaScriptillä ja niitä ylläpidetään ja ajetaan Firebasen palvelimella. Funktiot mahdollistavat palvelimen päässä tapahtuvan prosessoinnin, jolloin voidaan myös keventää selainsovelluksen päässä tapahtuvaa prosessointia sekä suorittaa koodia, jota ei

voida julkisesti näyttää selaimessa. Funktioilla on myös mahdollista ylittää Firestore-tietokannan ja Cloud Storage -palvelun turvallisuussäännöt, mikä mahdollistaa tarkemman kehittäjän määrittelemän tiedonkäsittelyn sovelluksessa. (Google Developers 2019o.)

Cloud Functions -funktioita julkaistaan luvussa 6.4 esiteltyä Firebasen komentorivityökalua hyödyntäen. Funktiot lisätään projektiin suorittamalla komento ”firebase init” ja valitsemalla komentokehotteen valikosta vaihtoehto ”Functions”. Funktioiden asetusten määrittämisen jälkeen komentorivityökalu generoi niille projektiin oman kansion. Funktiot ovat omana kokonaisuutenaan projektissa täysin erillään React-sovelluksen koodista, joten niillä on myös omat NPM-paketit ja oma erillinen package.json-tiedosto. Funktioiden kansiossa on index.js-tiedosto, josta kaikki funktiot viedään Firebaseen. Valmiit funktiot voidaan julkaista suorittamalla komento ”firebase deploy --only functions”. Yksittäinen esimerkki Cloud Functions -funktiosta on esitelty kuvassa 19. (Google Developers 2019k.)

```
const firebase = require('./firebase');
const storage = firebase.getStorageInstance();
const functions = require('firebase-functions');

module.exports = (data, context) => {
    const imagePath = data;
    return storage.bucket().deleteFiles({prefix: imagePath})
    .then(() => {
        return { success: true };
    })
    .catch(e => {
        throw new functions.https.HttpsError('internal', e.message);
    });
}
```

Kuva 19. Cloud Functions -funktio, joka poistaa sille parametriksi annetun kuvatiedoston Cloud Storage -palvelimelta.

Sovelluksessa Cloud Functions -funktioita hyödynnettiin esimerkiksi seuraavien toimintojen toteutuksessa:

- reCAPTCHA-tarkistuksen[4] suoritus palvelimella
- tapahtumiin liittyvien yhteystietojen ja kuvien automaattinen poistaminen tapahtuman poistuessa
- käyttäjän lisäämän tapahtuman peruminen, eli tapahtuman poistaminen yksilöllisen linkin kautta
- sähköposti-ilmoituksen lähettäminen sovelluksen hallintakäyttäjille tapahtuman lisäyksen yhteydessä.

Cloud Functions -palvelussa voidaan määrittää erikseen *ympäristömuuttujia*, eli vain palvelinympäristössä käytettäviä muuttujia, joita voidaan hyödyntää funktioissa. ReCAPTCHA-palvelun tarkistusta sekä sähköpostin lähetystä varten palvelimelle määriteltiin ympäristömuuttujat, joihin tallennettiin reCAPTCHA-palvelun vaatimat avaintunnukset sekä sähköpostin lähetystä varten tarvittava tunnus ja salasana. Ympäristömuuttujia voidaan asettaa Firebasen komentorivityökalun avulla, kun käyttäjä on kirjautunut komentorivin kautta sisään ja viimeistellyt projektin asetukset sillä paikallisesti. Ympäristömuuttujia voidaan asettaa esimerkiksi komennolla ”firebase functions:config:set esimerkki.avain=”arvo””. (Google Developers 2019f.)

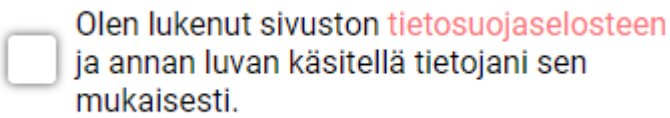
7 Tietosuoja

Euroopan komission verkkosivuilla on linjattu tietosuoja-asetuksen kohdistuvan ”yritykseen tai yksikköön, joka käsittelee henkilötietoja osana yhden EU:ssa toimivan sivuliikkeensä toimintoja riippumatta siitä, missä tietoja käsitellään” (Euroopan komissio 2019a). Tämä kohdistuu toimeksiannon sovellukseen, sillä sovelluksessa käyttäjiltä kerätään henkilötietoja tapahtuman lisäyksen yhteydessä ja tapahtumista pidetään rekisteriä, jonka rekisterinpitäjänä toimii opiskelijakunta POKA. Sovelluksella täytyy täten olla tietosuojaseloste, jossa käyttäjälle tehdään selväksi henkilökohtaisten tietojen käsittelyä koskevat asiat ja

[4] ReCAPTCHA tekee sovellukseen pienen haasteen, jonka suorittamalla haasteen onnistumisen ja käyttäjän päätelaitteen tietojen perusteella pyritään vahvistamaan käyttäjä ihmiseksi (Google Developers 2019e).

sovelluksen toiminnoissa pitää huomioida tietosuojan hallinta. Tietosuoja-asetuksen mukaiset toiminnot ja tietosuojaseloste toteutettiin osana opinnäytetyöprosessia ja tietosuojaseloste on raportissa liitteenä.

Henkilötietojen käsittelylle tulee olla jokin peruste, joka oikeuttaa henkilötietojen käsittelyyn sovelluksessa (Euroopan komissio 2019b). Tämän sovelluksen osalta tietojen keräämisen oikeudellinen peruste on käyttäjän suostumus, joka kerätään tapahtuman luomisen yhteydessä ennen kuin käyttäjä antaa tietojaan sovellukselle. Tapahtumalomakkeelle tehtiin erikseen kuvan 20 mukainen tarkistuslaatikko, jolla käyttäjä voi antaa luvan henkilötietojensa käsittelylle sekä tutustua sovelluksen tietosuojaselosteeseen ennen tietojensa antamista.



Kuva 20. Käyttäjän suostumuksen varmistus sovelluksen tapahtumalomakkeella.

Sovelluksessa yksityisyys huomioitiin kehityksen aikana, esimerkiksi varmistamalla käyttäjälle mahdollisuus poistaa juuri lisätty tapahtuma ja toteuttamalla automaattisia toimintoja tietojen poistamiseksi, kuten kuvien ja yhteystietojen automaattinen poistotoiminto. Sovelluksessa myös hyödynnettiin Firebasen tietoturvaominaisuuksia, kuten Firestore-tietokannan ja Cloud Storagen turvallisuussääntöjä ja Firebasen kirjautumista henkilötietoihin pääsyn rajoittamiseksi.
Sovelluksessa käytettävän reCAPTCHA-palvelun osalta käyttäjän lupa tietojen keräämiselle kysytään ennen reCAPTCHA-haastetta, sillä ennen haasteen näyttämistä käyttäjältä tulee jo olla lupa tietojen keräämiseksi. ReCAPTCHA-palvelun omista käyttöehdoista ja tietosuojaselosteista sisällytettiin linkit osaksi tietosuojaselostetta.

Firebase-palvelun ylläpitäjä Google toimii sovelluksessa datan prosessoijana; Google ei itse kontrolloi dataa, vaan tiedot ovat täysin Firebasen asiakkaan (opiskelijakunta POKAn) hallinnassa ja vastuu yksilöiden henkilötiedoista on Firebasen asiakkaalla (Google Developers 2019m). Tietosuojaselosteessa

käytiin läpi Firebasen eri palvelut ja pääpiirteittäin Firebasen merkitys sovelluksessa. Linkki Firebasen omaan tietosuojaselosteeseen sisällytettiin myös osaksi tietosuojaselostetta.

8 Tulokset

8.1 Vaatimusten täyttyminen

Lopullinen verkkosovellus täytti kaikki kriittiset vaatimukset sekä lähes kaikki pienemmän prioriteetin vaatimukset, joita toimeksiannon alussa kartoitettiin. Kokonaisuudessaan verkkosovellus saatiin valmiiksi ja julkaistua versiota ylläpidetään asiakkaan omistamassa Firebase-projektissa. Sovelluksen alustavan prototyypin toteutukseen kulunutta aikaa ei mitattu erikseen, mutta toimeksiantajan kanssa pidetyn ensimmäisen palaverin ja vaatimusten kartoituksen jälkeen sovelluksen toteutukseen meni noin 118 tuntia.

Kehityksen aikana toteutetut vaatimukset jaoteltuina eri kokonaisuuksiin:

- **Käyttöliittymä:** Kalenterisovelluksesta saatiin vaatimusten mukaisesti *responsiivinen*, eli sovellus on käytettävissä useilla eri päätelaitteilla. Sovellus noudattaa graafisesti POKAn ilmettä fonttien ja värien osalta. Käyttöliittymä on kaksikielinen (suomi ja englanti).
- **Tapahtumanäkymä, tapahtumien lisäyslomake:** Käyttäjä voi itse lisätä tapahtumia ja selata muiden lisäämiä tapahtumia hakusuodattimien avulla kalenterissa. Tapahtumien lisääminen tapahtuu käyttäjien tietosuojaa kunnioittaen, perustuen käyttäjän antamaan lupaan. Tapahtumista kerätään ja näytetään vaaditut tiedot sovelluksessa. Tapahtumien lisäystä rajoitetaan reCAPTCHA-palvelun avulla.
- **Tapahtumien hallintatila (vain kirjautuneet käyttäjät):** Tapahtumien lisäämien, poistaminen, tarkastelu ja hallinta on toteutettavissa sekä vahvistettujen että vahvistamattomien tapahtumien osalta. Tapahtuman

hallintatila on suojattu kirjautumisella ja käyttäjätilejä voidaan hallita erikseen Firebasen kautta.

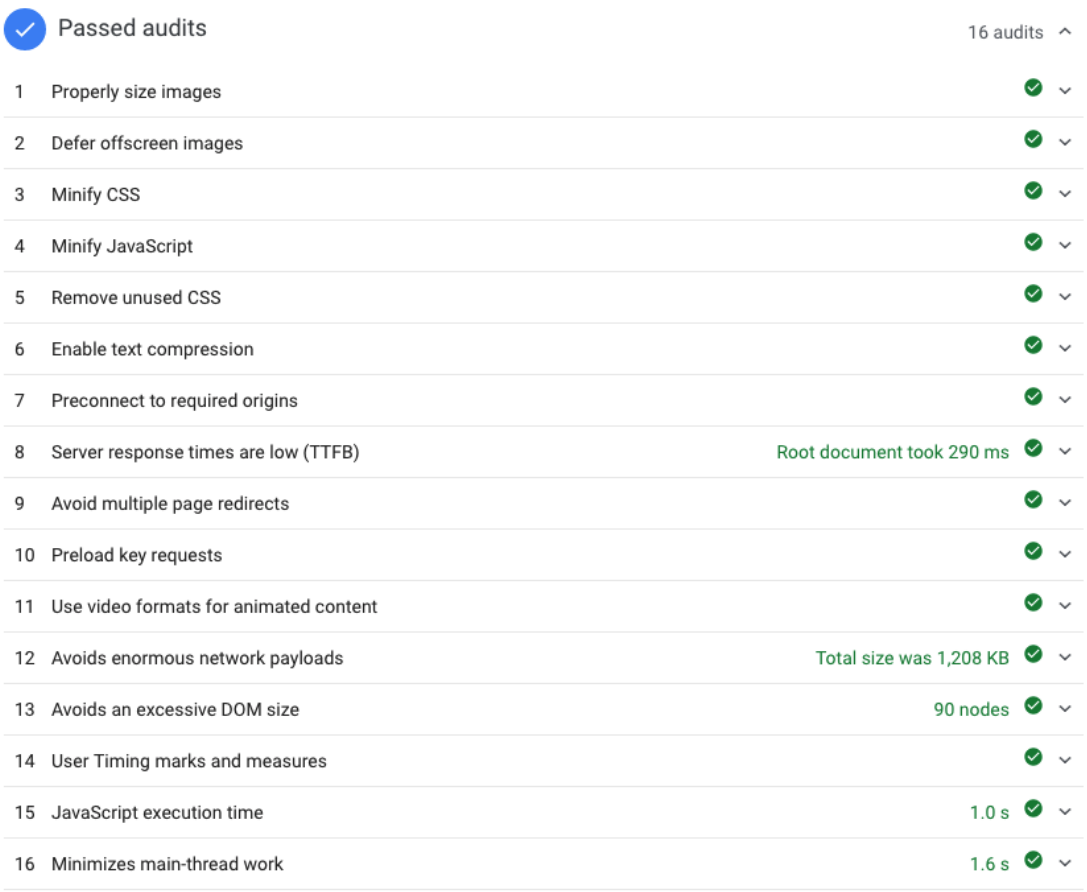
- **Itsenäisiä toimintoja:** Ylläpitäjille lähetetään automaattisesti sähköposti-ilmoitus, kun uusia tapahtumia lisätään kalenteriin. Tietojen poistamista varten toteutettiin myös itsenäisiä toimintoja.
- **Tietosuoja:** Sovellukseen toteutettiin tietosuojaseloste kaksikielisenä sekä sen edellyttämät toiminnot toteutettiin sovellukseen, kuten käyttäjän luvan kysyminen käyttäjän tietojen käsittelyyn ja tietojen automaattinen poistuminen.

Kehitysprosessin osalta jäi myös muutamia toimintoja, joita ei ajan puitteissa saatu toteutettua:

- **Facebook-tapahtumien suora lisääminen kalenteriin:** Toiminnon avulla tapahtumia voitaisiin hakea Facebookin rajapinnan kautta suoraan kalenteriin. Facebookin rajapinnan käyttöehdoista tällaisen toiminnallisuuden osalta ei ehditty tekemään selvitystä.
- **Pienemmän kalenterisovelluksen/upotteen tekeminen POKAn verkkosivuille:** Erillisen sovelluksen toteuttaminen ja sitä varten toteutettavan rajapinnan kehitys koitui ajallisesti haasteeksi, joten se jätettiin kehityskohteeksi tulevaisuutta varten.
- **Kuvien laadun karsiminen kuvan lisäyksen yhteydessä:** Sovelluksen nopeamman latautumisen ja suorituskyvyn osalta tapahtumien kuvien laadun automaattinen karsiminen olisi hyvä tapa tehdä sovelluksesta optimaalisempi. Toiminto jätettiin kehityskohteeksi tulevaisuutta varten.
- **Tapahtumien lisäämien ulkoiseen kalenteriin:** Tapahtuman lisääminen esimerkiksi iCalendar-standardin mukaisesti sitä tukeviin kalenterisovelluksiin. Toiminto jätettiin kehityskohteeksi tulevaisuutta varten.
- **Tapahtumien jakamistoiminnot:** Tapahtumien jakaminen eri sovelluksissa ja alustoilla, esimerkiksi sosiaalisen median sovelluksissa. Toiminto jätettiin kehityskohteeksi tulevaisuutta varten.

8.2 Sovelluksen tekniset tulokset

Sovelluksen tehokkuutta mitattiin Googlen PageSpeed Insights -työkalulla. Sovellus sai testissä tulokseksi 80/100 pistettä, mikä merkitsee sovelluksen olevan keskivertotasoa nopeudeltaan. Pääpiirteittäin sivustolla kehitettävää jäi kuvien optimoinnin, sivuston fonttien latauksen ja staattisten tiedostojen välimuistin käytön osalta. Kuvien optimoinnin osalta suorituskykyä voitaisiin parantaa kuvien optimoinnilla latauksen yhteydessä, mikä jätettiin kehityskohteeksi sovelluksen osalta. Fonttien latausasetuksia voidaan korjata muokkaamalla sovelluksen tyylitiedostoja ja välimuistin käyttöasetuksia voidaan korjata Hosting-palvelun asetusten kautta. Sovellus läpäisi yhteensä 16 eri tarkastusta, joita testissä suoritettiin. Testissä myös huomataan, että selaimelle toimitettu JavaScript- ja CSS-tiedostojen tiivistäminen vaikuttaa positiivisesti sovelluksen latausnopeuteen. Kuvassa 21 on esiteltynä sovelluksen tuloksia PageSpeed Insights -testistä.



Kuva 21. Sivuston läpäisemät testit PageSpeed Insights -testissä.

8.3 Firebasen toimintojen hyödyntäminen toteutuksessa

Firebasen toiminnoista hyödynnettiin sovelluksessa Firebasen kirjautumista, Firestore-tietokantaa, Cloud Storage -palvelua, Hosting-palvelua sekä Cloud Functions -palvelua. Näiden palveluiden hyödyntämisen lisäksi käsiteltiin Firebasen ja web-sovelluksen integrointia sekä Reactin ja Reduxin integraatiota Firebasen eri palveluiden kanssa. Toimeksiannon aikarajojen sisällä sovellus saatiin käytettyjen työkalujen avulla valmiiksi saakka ja lopullisen järjestelmän ylläpito on täysin hallittavissa Firebasen kautta asiakkaan toimesta ilman fyysisiä palvelimia tai virtuaalikoneiden hallinnointia. Lopullisen sovelluksen hallinnoinnin kannalta tehtiin jo toteutuksessa täten säästöjä hallinnointiin vaadittavien resurssien osalta. Järjestelmän käytön ja hallinnoinnin perehdyttämisen kannalta Firebasen palvelut ja hallintakäyttöliittymä helpottavat järjestelmän käytön perehdyttämistä myös uusille opiskelijakunnan työntekijöille.

9 Pohdinta

9.1 Kehitysprosessi

Opinnäytetyö kokonaisuudessaan osoittautui odotettua suuremmaksi ja kehitystyö venyi hieman arvioidusta aikamääreestä työmäärän kasvaessa. Lopputuloksen valmistumisen viivästymiseen vaikutti osittain sovelluksen tietosuojaan liittyvät muokkaukset teknisessä toteutuksessa sekä tietosuojaselosteen toteutus, joita ei pystytty vielä toimeksiannon alussa realistisesti arvioimaan. Aika-arviot vaatimusten osalta myös venyivät arvioitua pidemmiksi ja kehitysprosessin aikana ilmeni uusia tarpeellisia vaatimuksia, jotka jouduttiin priorisoimaan muiden kehitystehtävien edelle. Riski oli kuitenkin huomioitu kehityksen alussa ja tämän pohjalta vaatimuksia muokattiin yhdessä toimeksiantajan kanssa niin, että lopputuloksena saatiin valmis ratkaisu. Osa vaatimuksista jätettiin tulevaisuuden kehityskohteiksi.

9.2 Työkalut

Opinnäytetyössä hyödynnettyjen työkalujen toimintaa olisi voitu käsitellä syvällisemmin erillisinä kokonaisuuksinaan, mutta tämän opinnäytetyön tavoitteiden mukaisesti niitä käsiteltiin toimeksiannon vaatimusten näkökulmasta. Toimeksiannon teknisestä toteutuksesta valittiin raporttiin esiteltäväksi ne ratkaisut, joita voitaisiin soveltaa parhaiten myös muissa samankaltaisissa sovelluksissa. Teknisten ratkaisujen toteutuksessa pyrittiin myös huomioimaan parhaat käytännöt työkalujen käytön osalta hyödyntämällä pääasiassa kunkin työkalun omaa dokumentaatiota sekä kriittisesti soveltaen erilaisten ajankohtaisten ja luotettavien lähteiden ja opetusmateriaalien tarjoamia ratkaisuja. Raportissa esiteltiin työkaluista myös yleisellä tasolla lukuisia esimerkkejä niiden soveltuvuudesta ja käyttötarkoituksesta tavoitteiden mukaisesti.

Firebase mahdollisti erityisesti käyttöliittymään ja sovelluksen toiminnallisuuteen painottuvan kehityksen. Firebasen käyttäminen sovelluksen palvelinrajapintana säästi järjestelmän tuottamiselta huomattavasti aikaa, sillä vastaavan palvelinrajapinnan toteuttaminen olisi ollut kokonaisuutena täysin erillinen toimeksianto. Palvelinrajapinnan toteutus olisi lisäksi vaatinut myös virtuaalikoneiden tai fyysisten palvelimien käyttöönoton, jolloin palvelinten hallinnointiin olisi pitänyt varata resursseja ja osaamista, joita toimeksiantajalla ei olisi ollut käytettävissä. Firebasen käytön myötä kaikki palvelut ovat keskitettynä yhdessä paikassa, jolloin kustannukset sovelluksen käytöstä muodostuvat ainoastaan Firebasen palveluiden osalta. Esimerkiksi sovelluksen tietokantaa, tiedostopalvelinta ja sovelluksen resurssien ylläpitoa varten olisi saatettu joutua hyödyntämään useita eri palveluita, jolloin kustannuksia olisi muodostunut erillisten palveluiden myötä huomattavasti enemmän. Firebase soveltuu hyvin toimeksiannon kaltaiseen pieneen sovellukseen, jolla ei tule olemaan suuria käyttäjämassoja eikä sovellukselta odoteta monimutkaista tiedon prosessointia tai useiden eri palveluiden hyödyntämistä.

Firebasen käyttäminen palvelinrajapintana tuo myös rajoitteita sovelluksen laajennettavuuden, siirrettävyyden ja toimivuuden kannalta. Seuraavanlaisia rajoitteita ilmeni kehityksen aikana:

- Tietokantahakujen muodostamisessa ilmeni huomattavia rajoitteita hakuehtojen muodostamisessa, jotka saattaisivat olla esteenä Firebasen soveltuvuudelle järjestelmässä, joka vaatii monimutkaisempia hakuja. Esimerkiksi tekstihakua ei voitu toteuttaa sovellukseen, sillä Firestoren tietokantahaussa ei voida hakea dataa joka muistuttaa käyttäjän syöttämää tekstiä.
- Firebasen oma kirjautuminen ei tue suoraan eri käyttäjätasojen määrittämistä käyttäjätileille. Tämän ominaisuuden avulla olisi voitu tehdä sovellukseen eri tasoisia käyttäjiä esimerkiksi opiskelijakunnan työntekijöille ja eri kumppanijärjestöjen edustajille, jotka lisäävät tapahtumiaan kalenteriin.
- Välillä Cloud Functions -funktioissa ja muissa Firebasen palveluissa ilmeni pieniä viiveitä käytössä.
- Firebasen käytön myötä myös toteutettu sovellus on kokonaan Firebasen varassa ja täten tulevaisuudessa tapahtuvat muutokset palvelun ehdoissa ja toiminnassa vaikuttavat suoraan sovellukseen.

Reactin avulla käyttöliittymäkehitys oli tehokasta ja käyttöliittymien mallintaminen oli huomattavasti vaivattomampaa toteuttaa Reactin toiminnallisuuden avulla kuin se olisi ollut pelkästään JavaScriptin ja HTML:n avulla. Reactin käyttö pakottaa kehittäjää noudattamaan myös parempia kehityskaavoja esimerkiksi hyödyntämällä olio-ohjelmoinnin periaatteita, ES6-moduuleita sekä komponenttirakennetta sovelluksen kehityksessä. Redux oli hyvin olennainen osa React-sovelluksen datan hallintaa ja vastaavanlainen kirjasto on lähes pakollinen lisä etenkin monimutkaisten React-sovellusten toteutuksessa. Reactin ja Reduxin avulla voidaan tehdä hyvin suorituskykyisiä, monipuolisia ja interaktiivisia verkkosovelluksia.

React ja Redux olivat aluksi hieman monimutkaisia kokonaisuuksia ja niiden periaatteiden omaksuminen voi olla uusille kehittäjille haaste alussa. Kirjastot generoivat yhdessä myös paljon JavaScript-koodia selaimelle, joten

yksinkertaisilla sivustoilla Reactin ja Reduxin käyttäminen voi olla tarpeettoman raskasta sivun toimintoihin nähden.

Hakukoneet eivät myöskään kykene kunnolla lukemaan JavaScriptillä dynaamisesti renderöityä HTML-dokumenttia. Tämän vuoksi React-sovellusten hakukonenäkyvyys on heikkoa. Parannuksena tähän React-sovelluksen ulkoasut voitaisiin renderöidä palvelinympäristössä, jolloin hakukoneet pystyisivät lukemaan valmiiksi renderöityjä HTML-dokumentteja paremmin.

9.3 Prosessin aikana kerrytetty osaaminen

Oppimiskokemuksena opinnäytetyö oli hyvin kokonaisvaltainen, käsittäen sovelluksen vaatimusten määrittelyn, teknologioiden valinnan, suunnittelun, toteutuksen, julkaisun, tietosuojan ja ylläpidon suunnittelun. Toteutus edisti erityisesti ohjelmointiosaamista JavaScriptin ja Reactin avulla sekä tietämystä JavaScriptin eri kirjastoista kuten Reduxista. Toteutuksen aikana myös Firebasen perustoiminnot tulivat tutuiksi ja niiden käytettävyyttä on helpompaa arvioida jatkossa vastaavien sovellusten toteutuksen osalta. Opinnäytetyöprosessi edisti myös osaamista sovelluksen julkaisemiseen liittyvistä asioista sekä EU:n tietosuoja-asetuksen asettamista vastuista, joita tulee huomioida jo sovelluksen kehitysvaiheessa.

9.4 Jatkokehitysideat

Osa sovelluksen vaatimuksista jäi toteuttamatta ja ne on kuvattu tarkemmin luvussa 8.1. Tässä luvussa on lisäksi kuvattuna kehityskohteita, joilla saavutettaisiin toimeksiantoa paremmin palveleva lopputulos.

Sovelluksen nopeuttamiseksi ja hakukonenäkyvyyden parantamiseksi sovellukseen tulisi tehdä erillinen Node.js-palvelin, joka vastaisi kokonaan

sovelluksen alasivujen renderöinnistä. Erillisellä palvelimella voitaisiin hyödyntää Reactin työkaluja, joilla renderöidään käyttöliittymä palvelinympäristössä. Tämä parantaisi sovelluksen hakukonenäkyvyyttä ja vähentäisi selaimelle toimitettavaa JavaScript-koodia nopeuttaen sovelluksen toimintaa.

Firebase-projektiin voitaisiin jatkossa toteuttaa Cloud Functions -funktio, joka lähettäisi kutsuttaessa Firestore-tietokannasta tapahtumia ulkoisille järjestelmille. Funktio toimisi rajapintana, jonka kautta uudet tapahtumat voitaisiin hakea ja listata myös sovelluksen ulkopuolella. Tämä vaatisi myös sovelluksen tietosuojaehtojen uudistamista ja muokkausta.

Lähteet

Dan Abramov and the Redux documentation authors. 2019. Three Princibles. Redux. https://redux.js.org/introduction/three-principles. 23.1.2019.

Euroopan Komissio. 2019a. Keneen tietosuojalakia sovelletaan?. Euroopan komissio. https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/application-regulation/who-does-data-protection-law-apply_fi. 18.4.2019.

Euroopan Komissio. 2019b. Milloin henkilötietoja voidaan käsitellä?. Euroopan komissio. https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/legal-grounds-processing-data/grounds-processing/when-can-personal-data-be-processed_fi. 18.4.2019.

Facebook Inc. 2019a. Components and Props. React. https://reactjs.org/docs/components-and-props.html. 24.4.2019

Facebook Inc. 2019b. Create a New React App. React. https://reactjs.org/docs/create-a-new-react-app.html. 24.4.2019

Facebook Inc. 2019c. Docs. Create React App. https://facebook.github.io/create-react-app/docs/getting-started. 14.4.2019.

Facebook Inc. 2019d. Introducing JSX. React. https://reactjs.org/docs/introducing-jsx.html. 23.4.2019.

Facebook Inc. 2019e. Rendering Elements. React. https://reactjs.org/docs/rendering-elements.html. 24.4.2019.

Facebook Inc. 2019f. State and Lifecycle. React. https://reactjs.org/docs/state-and-lifecycle.html. 24.4.2019.

Firebase. 2016. Introducing Firebase. Youtube. https://www.youtube.com/watch?v=O17OWyx08Cg. 24.1.2019.

Google Developers. 2019a. Add Firebase to your JavaScript project. Firebase. https://firebase.google.com/docs/web/setup. 24.4.2019.

Google Developers. 2019b. Authentication State Persistence. Firebase. https://firebase.google.com/docs/auth/web/auth-state-persistence. 24.4.2019.

Google Developers. 2019c. Cloud Firestore Data Model. Firebase. https://firebase.google.com/docs/firestore/data-model. 24.4.2019.

Google Developers 2019d. Cloud Storage. Firebase. https://firebase.google.com/docs/storage/. 24.4.2019.

Google Developers. 2019e. Developer’s Guide. reCAPTCHA. https://developers.google.com/recaptcha/intro. 18.4.2019.

Google Developers. 2019f. Environment configuration. Firebase. https://firebase.google.com/docs/functions/config-env. 24.4.2019.

Google Developers. 2019g. Firebase CLI Reference. Firebase.https://firebase.google.com/docs/cli/. 17.4.2019.

Google Developers. 2019h. Get Started with Firebase Authentication on Websites. Firebase. https://firebase.google.com/docs/auth/web/start. 24.4.2019.

Google Developers. 2019i. Get Started with Firebase Hosting. Firebase. https://firebase.google.com/docs/hosting/quickstart. 24.4.2019.

Google Developers. 2019j. Learn to Secure Files. Firebase. https://firebase.google.com/docs/storage/security/secure-files. 24.4.2019.
Google Developers. 2019k. Manage functions deployment and runtime options. Firebase. https://firebase.google.com/docs/functions/manage-functions. 24.4.2019.
Google Developers. 2019l. Pricing Plans. Firebase. https://firebase.google.com/pricing/. 13.4.2019.
Google Developers. 2019m. Privacy and Security in Firebase. Firebase. https://firebase.google.com/support/privacy/. 20.4.2019.
Google Developers 2019n. Structuring Cloud Firestore Security Rules. Firebase. https://firebase.google.com/docs/firestore/security/rules-structure. 24.4.2019.
Google Developers. 2019o. What can I do with Cloud Functions?. Firebase. https://firebase.google.com/docs/functions/use-cases. 24.4.2019
Grider, S. 2016. Modern React With Redux. Udemy, Inc. https://www.udemy.com/react-redux/. 23.1.2019.
Halme, A. 2018. Mikä on Single Page App ja mihin sitä käytetään?. City Dev Labs. https://citydevlabs.fi/single-page-app/. 23.4.2019
Mozilla ja yksittäiset avustajat. 2019a. An overview of HTTP. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. 22.4.2019.
Mozilla ja yksittäiset avustajat. 2019b. CSS: Cascading Style Sheets. MDN Web Docs. https://developer.mozilla.org/fi/docs/Web/CSS. 22.4.2019
Mozilla ja yksittäiset avustajat. 2019c. Document Object Model (DOM). MDN Web Docs. https://developer.mozilla.org/fi/docs/DOM. 29.1.2019.
Mozilla ja yksittäiset avustajat. 2019d. HTML: HyperText Markup Language. MDN Web Docs. https://developer.mozilla.org/fi/docs/Web/HTML. 29.1.2019.
Mozilla ja yksittäiset avustajat. 2019e. Introducing asynchronous JavaScript. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing. 24.4.2019.
Mozilla ja yksittäiset avustajat. 2019f. What is JavaScript?. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#What_is_JavaScript. 23.1.2019.
Papp, A. 2018. The History of React.js on a Timeline. https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/. 23.1.2019.
Rauschmayer, A. 2018. Exploring ES6. Axel Rauschmayer. http://exploringjs.com/es6.html. 23.1.2019.
React Training. 2019. Basic Components. React Training. https://reacttraining.com/react-router/web/guides/basic-components. 24.4.2019.
Roberts, M. 2018. Serverless Architectures. Martin Fowler. https://martinfowler.com/articles/serverless.html. 24.1.2019.
Sherman, P. 2017. A Simple React Router v4 Tutorial. Blog.pshrmn. https://blog.pshrmn.com/entry/simple-react-router-v4-tutorial/. 10.3.2019.

Sprmn. 2018. React-firebase-file-uploader. NPM. https://www.npmjs.com/package/react-firebase-file-uploader. 24.4.2019.

Wieruch, R. 2018. A Firebase in React Tutorial for Beginners [2018]. RWieruch. https://www.robinwieruch.de/complete-firebase-authentication-react-tutorial/. 10.3.2019.

Tietosuojaseloste

POKA Kalenteri tietosuojaseloste

Rekisterinpitäjä
Karelia-ammattikorkeakoulun opiskelijakunta POKA (Palveluntarjoaja)

Rekisterinpitäjän edustaja
Nina Yletyinen
Tikkarinne 9, 80200 Joensuu
paasihteeri@pokapoka.fi
044 060 6472

Mitä tietoa kerätään

Sähköposti
Kerätään mahdollista yhteydenottoa varten. Ei näytetä jukisesti sivustolla.
Puhelinnumero
Kerätään mahdollista yhteydenottoa varten. Ei näytetä jukisesti sivustolla.
Käyttäjän laitekohtaiset tiedot
Sivuston asianmukaisen käytön turvaamiseksi sivustolla on käytössä Googlen reCAPTCHA-palvelu. ReCAPTCHA kerää tietoa käyttäjän laitteesta ja ohjelmistoista varmistaakseen, että palvelua käyttää ihminen. ReCAPTCHAn luomaa dataa ei säilytetä sivustolla.

Tutustu Googlen palveluehtoihin ja tietosuojakäytäntöön ennen kuin tallennat palveluun tapahtumia.
Googlen palveluehdot: https://policies.google.com/terms?hl=fi
Googlen tietosuojakäytäntö: https://policies.google.com/privacy?hl=fi

Tiedostot
Tapahtumille voidaan vapaaehtoisesti lisätä kuva. Kuvatiedostot näytetään julkisesti sivustolla tapahtumien yhteydessä.

Tapahtumakohtaiset tiedot
Tapahtumien luonnin yhteydessä kerätään seuraavaa tietoa:
- Tapahtuman nimi suomeksi ja englanniksi
- Tapahtuman kuvaus suomeksi ja englanniksi
- Tapahtumatyyppi
- Tieto tapahtuman ilmaisesta osallistumisesta
- Tapahtuman ajankohta, sijainti, verkkosivu ja järjestäjä

Miten tietoa käytetään
Kaikki sivulla kerättävät tiedot liittyvät sivustolla julkaistaviin tapahtumiin. Tietojen pääasiallinen käyttötarkoitus on sivustolla toteutettava tapahtumaviestintä.

Missä tietoa säilytetään
POKA Kalenterin ylläpito tapahtuu **Google Firebase**-palvelussa. Tapahtumien tiedot, kuvat ja niihin liitetyt yhteystiedot säilytetään Google Firebase-palvelun

Tietosuojaseloste

tietokannassa. Sivustolle tallentamaasi tapahtumakohtaista tietoa, tiedostoja ja yhteystietoja säilytetään ja käsitellään Euroopan Unionin-alueella.

Seuraavia Google Firebase-alustan palveluita hyödynnetään sivustolla:

- Cloud Functions for Firebase (palvelimen eri toiminnot tiedon prosessoinnissa)
- Firebase Authentication (koskee vain palveluntarjoajan kirjautumista)
- Firebase Hosting (sivuston staattisten tiedostojen ylläpito)
- Cloud Firestore (tapahtumien tietojen säilytys, yhteystietojen säilytys)
- Cloud Storage for Firebase (tiedostojen säilytys)

Tutustu Google Firebase-palvelun yksityisyys- ja turvallisuusselosteeseen ennen kuin tallennat palveluun tapahtumia.
Yksityisyys ja turvallisuus Firebase-palvelussa (vain englanniksi): https://firebase.google.com/support/privacy/

Miten pitkään tietoa säilytetään
Tapahtumien tietoja ja niihin liitettyjä kuvia ja yhteystietoja säilytetään viikon ajan tapahtuman päättymisajankohdasta.

Miten tiedot saa poistettua
Juuri lisätty tapahtuma voidaan poistaa tapahtuman lisäyksen jälkeen näytettävästä painikkeesta tai navigoimalla painikkeen osoittamaan osoitteeseen. Jos tapahtuma on jo vahvistettu, ole yhteydessä rekisterinpitäjän edustajaan. Tapahtuman poistaminen hävittää tapahtumaan liittyvän kuvan sekä yhteystiedot.

Tietojen korjaus ja tarkastusoikeus (rekisteröidyn oikeudet)
Rekisteröidyllä on oikeus pyytää rekisterinpitäjältä pääsyä häntä itseään koskeviin henkilötietoihin ja oikeus pyytää kyseisten tietojen oikaisemista, poistamista, käsittelyn rajoittamista tai vastustaa käsittelyä sekä oikeus siirtää tiedot järjestelmästä toiseen. Rekisteröidyllä on oikeus milloin tahansa perua antamansa suostumuksensa henkilötietojensa käsittelyyn. Lisäksi rekisteröidyllä on oikeus tehdä valitus valvontaviranomaiselle. Yhteyshenkilö rekisteröidyn oikeuksiin liittyvissä asioissa on rekisterinpitäjän edustaja.

Tietoturva
Sivustolla on käytössä SSL-suojaus tietojen siirrossa. Sivuston ei-julkisten tietojen tarkastelu ja käsittely on suojattu rekisterinpitäjän kirjautumisella.

Tietojen käsittelyn oikeusperuste
Peruste tietojen keräämiselle on käyttäjän antama suostumus. Hyväksymällä tämän tietosuojaselosteen annat suostumuksen tietojen keräämiselle.