

Julia Virtanen, Noora Turunen, Esa Ryömä, Markus Ojajärvi

## Loppuraportti

moiccu.

---

Metropolia Ammattikorkeakoulu

Tieto- ja viestintätetekniikka

Ohjelmistotuotanto

Ohjelmistotuotantoprojekti 2

8.5.2019



## Sisällyös

1	Suunnittelu	1
1.1	Visio	1
1.2	Ulkoasu	1
2	Toteutus	2
2.1	Arkkitehtuuri	2
2.2	Käyttöliittymä, React	3
2.3	Back end	6
2.4	Evästeet, sisäänsijautuminen ja JSON Web Tokenit	7
2.5	Tietokanta	8
2.6	Lokalisointi	9
2.7	Lopputilanne	9
3	Testaus	11
3.1	Yksikkötestaus, Jenkins	11
3.2	Käyttöliittymätästaus, Sideex	12
3.3	Manuaalinen testaus	14
4	Reflektointi	16
4.1	Onnistumiset	16
4.2	Kehittävä	16
4.3	Yhteenvetö	17
	Lähteet	18



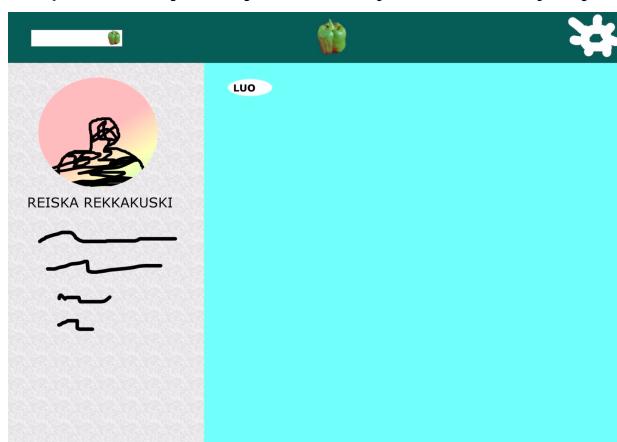
## 1 Suunnittelu

### 1.1 Visio

Visiona oli luoda kalenterisovellus, jossa olisi helppokäyttöisiä ja selkeitä ominaisuuksia ryhmätapaamisten ja tapahtumien luomiseksi. Ajatuksena oli, että pohjalla olisi jokaisen käyttäjän henkilökohtainen kalenteri, johon käyttäjä voisi luoda omia merkintöjä ja linkittää niihin muistiinpanoja. Sovelluksessa olisi mahdollista myös luoda tapahtumia, joihin pystyisi kutsumaan muita sovelluksen käyttäjiä. Käyttäjien olisi mahdollista muodostaa ryhmiä, jonka jäsenien kalenterit saisi sulautettua yhteen ryhmäkalenteriin yhdellä klikkauksella. Nämä käyttäjät olisi helppo katsoa kaikille sopiva ajankohta tapaamiselle. Meillä oli alusta asti tiedossa, että sovelluksessamme tulee olemaan paljon samaa kuin Googlen omassa kalenterissa. Halusimme kuitenkin luoda oman itsenäisen ja helppokäyttöisen sovelluksen, joka palvelisi optimalisesti juuri meidän tarpeitamme.

### 1.2 Ulkoasu

Tarkoitus oli tehdä niin sanottu "single page application", eli sovellusta käytäessä ladataan vain yksi html-sivu. Sovellusta käytäessä sisältö luotaisiin toimintojen mukaan dynaamisesti JavaScriptin avulla. Sovelluksella olisi kiinteät sivu- ja yläpalkit, joiden toiminnot klikkaamalla osa sivun näkymästä muuttuu tai popup ilmestyy. Sivupalkissa olisi valittavissa kuvakkeet profiilin tarkasteluun, muistiinpanon tekemiseksi, ryhmän tarkasteluun ja hallintaan sekä tapahtuman ja merkinnän luomiseksi. Painikkeita klikkaamalla sivupalkki laajensi ja siinä näytettiisiin käyttäjän klikkaaman kuvakkeen toiminnot. Käyt-



täjän kalenteri olisi koko ajan näkyvissä. Yläpalkkiin siirrettäisiin toiminnot, jotka eivät ole samalla tavalla käyttäjän aktiivisuutta vaativia toiminnoja kuin sivupalkin toiminnot. Yläpalkkiin sijoitettaisiin kuvake asetuksille, hakukenttä sekä käyttäjän ilmoitusten hallinta. (Kuva 1).

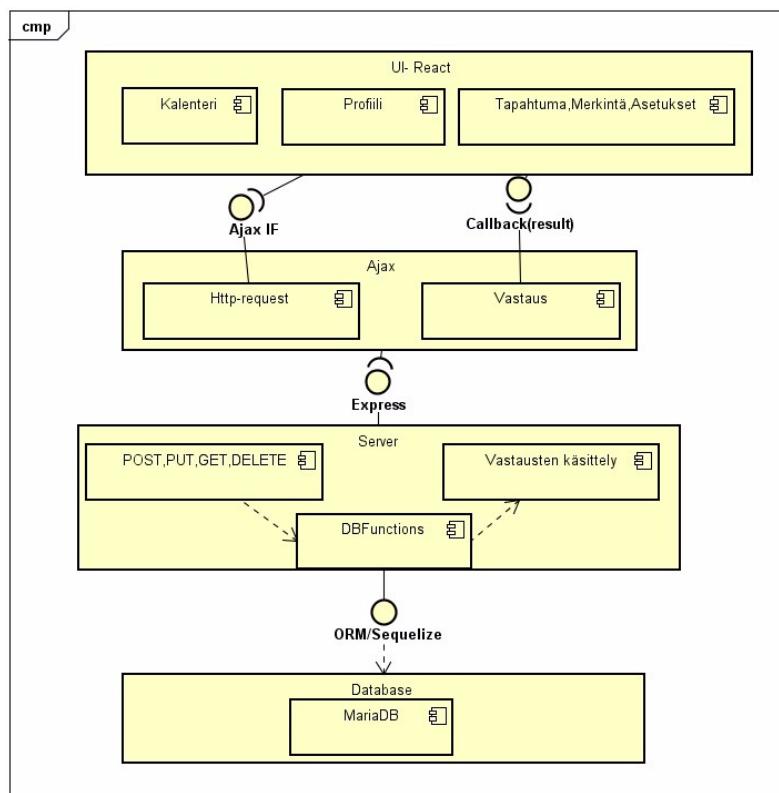
Kuva 1 Suunnitelma sovelluksen ulkoasusta



## 2 Toteutus

### 2.1 Arkkitehtuuri

Tuotteen arkkitehtuuri koostuu neljästä kerroksesta ja näiden välisistä rajapinnoista (kuva 2). Arkkitehtuuri on pysynyt suhteellisen samana alusta asti.



Kuva 2 Arkkitehtuurikaavio

Ylimpänä osana on React-pohjainen käyttöliittymä käyttäjän selaimessa. Selaimesta pyynnöt etenevät itsetehtyyn Ajax-moduuliin, jonka metodeilla lähetetään palvelimelle get, put, update, delete -pyyntöjä tiettyihin osoitteisiin eli polkuihin.

Palvelimella pyynnön vastaanottaa Express-moduuli, jolle on määritelty polkuja eri pyytöjä varten. Nämä on tarkasti määritelty, sillä tietty pyyntö (esim. get-pyyntö osoitteeseen domain.com/profiles) selaimelta etenee tiettyä polkuoa omanlaiseensa suoritukseen (kuva 3).



```

app.get('/entries', middleware.addAccountIdToRequest, async (req, res) => {

    //console.log(req.query);
    let accId = req.app.locals.accId;
    let sdate = req.query.sdate;
    let edate = req.query.edate;

    let entries = await database.getCalendarEntries(accId, sdate, edate);
    let jsonEntries = JSON.stringify(entries);
    console.log(jsonEntries);
    res.send(jsonEntries);
    // Todo: A separate GET-route to have more parameters (date/time scopes)

});

```

### Kuva 3 Yksittäinen express-polku (get-pyyntö osoitteeseen domain.fi/entries)

Lisäksi ennen varsinaista polkua käytämme esikäsittelymoduuleja (ns. *middleware*), jotka muokkaavat pyynnön sisällön tietyissä tapauksissa helpommin käsiteltäväksi tai lisäävät pyyntöön tarvittavia asioita (esim. Body-parser -moduuli muokkaa post-pyynöistä json-tyyppisen datan automaattisesti muuttujaksi, eikä tätä tarvitse erikseen ohjelmoida).

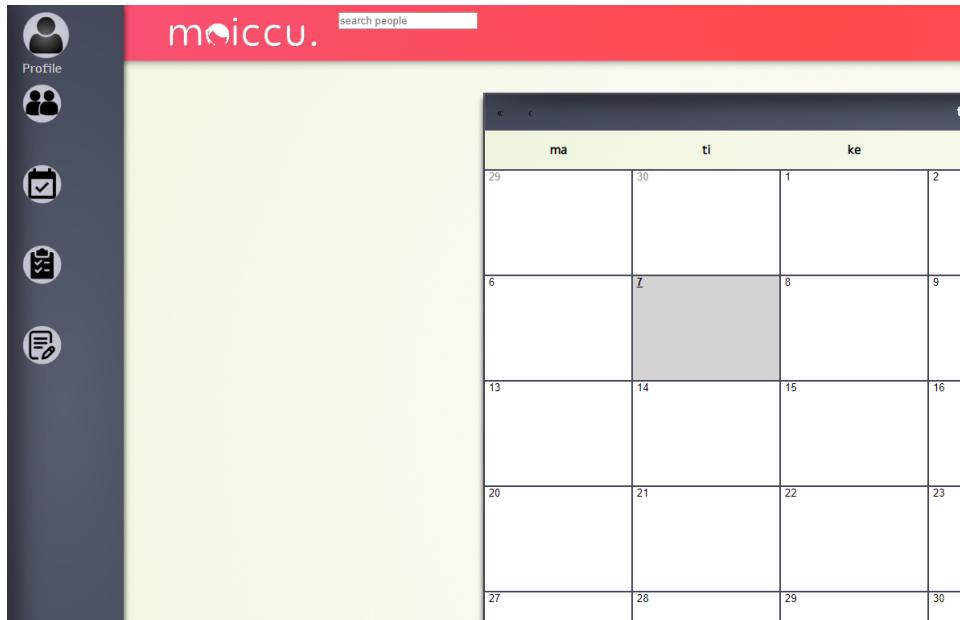
Express-polkujen koodissa käsittelemme pyynnön tekemällä tietokantaan CRUD-operaatioita. Tässä työkaluna toimi ORM-moduuli Sequelize, jolla MariaDB - tietokanta alustettiin ja sitä käytettiin operoimaan kantaa ilman SQL-kieltä.

Pyyntöjen vastaukset lähetetään takaisin frontille JSON-muodossa, jossa callback-mетодi hoitaa vastauksen saapumisen, sitten kun sellainen on saatavissa. Projektissamme ei ole hyödynnetty Promise-rakennetta, vaan osittainen asynkronisyys on toteutettu callbackien avulla.

## 2.2 Käyttöliittymä, React

Käyttöliittymä on toteutettu Reactilla, HTLM:llä ja CSS:llä. React valikoitui frontin toteutustekniikaksi, sillä halusimme kokeilla jotain uutta tekniikkaa. Lisäksi mahdollisuus asentaa ympäristöön avoimen lähdekoodin komponentteja houkutteli. Käyttöliittymän kalenteri on toteutettu Reactin oman kalenterikomponentin, react-calendarin, avulla. Sivusto on lisätty muun muassa react-simple-tooltip komponentin avulla toteutetut tooltipit. (kuva 4)





Kuva 4 Kalenteri sekä profiili-ikonin tooltip

Vaikka kalenteri olikin helppo asentaa, ja sen ulkoasua muokata, sen sisällön lisäämiseen ja muokkaamiseen ei löytynyt mitään materiaalia yksittäisiä keskustelupalstan viestitketjuja lukuun ottamatta. React.js -ympäristön asennuksessa oli aluksi vaikeuksia. Jo valmiiksi asennettujen NodeJS -kirjastojen päälle ei ollut mahdollista asentaa Reactia, vaan se vaati manuaalista kikkailua. Tähän kului ylimääräisiä työtunteja projektin alussa. Useampi työviikko kului itse uuden kielen opetteluun ja simppelit asiat vaativat aktiivista tutkimustyötä internetistä. Komponentit eivät toimineet oikein eikä niihin talletettu data päivittynyt. Syynä oli JSX -kielen väärinkäyttö, kun komponentit aseteltiin sivulle (esimerkki 1).

```
ReactDOM.render(profile.render(), document.getElementById('profile'));
```

Ennen - toimimaton ratkaisu

```
ReactDOM.render(<Profile/>, document.getElementById('profile'));
```

Jälkeen - toimiva ratkaisu

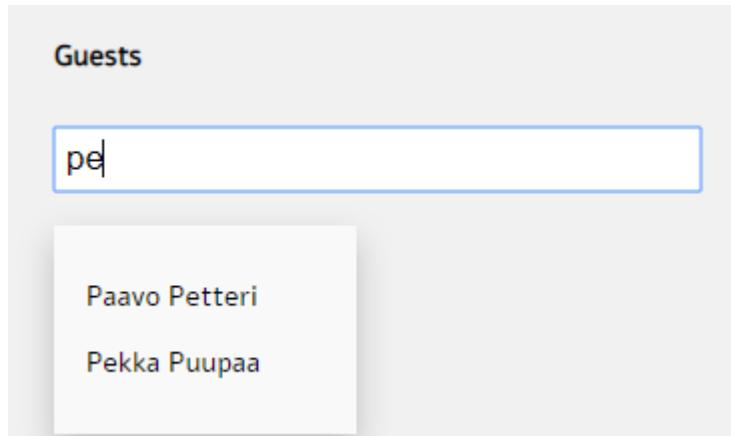
### Esimerkki 1 Koodiesimerkki JSX-kielen ongelmasta.

Tässä jaksossa uutena vaatimuksena saimme käyttöliittymän saavutettavuuteen liittyviä vaatimuksia. Näiden pohjalta kiinnitimme enemmän huomiota saavutettavuuteen ja käytettävyyteen ja lisäsimme elementtejä ja ominaisuuksia sivustoomme. Yksi tärkeimmistä



elementeistä on jo aiemmin mainittu tooltip. Sovellus on yleisesti ottaen selkeä, ja esimerkiksi lomakkeiden kentät ovat hyvinkin selkeitä, ja koimme ettei niihin tarvita tooltippejä. Kuitenkin vaseman sivupalkin ja yläpalkin ikonit halusimme pitää pelkistettyinä, ja näihin lisättiinkin yksinkertaiset tooltip-tekstit.

Tämän lisäksi lisäsimme ominaisuuksia, joiden tarkoitus oli minimoida käyttäjän virheestä aiheutuvat virhetilanteet. Yksi lisätyistä elementeistä oli hakukenttiin täydennys (kuva 5). Hakukenttä hakee käyttäjän antaman syöteen mukaan listan mahdollisista hakuehdoista, joista käyttäjän on valittava haluamansa. Toiminnallisuuden toteutus jäi siinä mielessä kesken, ettei köyttäjän ole mahdollista hakea tietuetta, jota hakukenttä ei ehdota. Tosin, silloin käyttäjä ei myöskään saisi minkäänlaista tulosta, sillä hakukenttä ehdottaa kaikista mahdollisista hakutuloksista kyseisellä hakusanalla löytyvät tietueet.



Kuva 5 Hakukentän automaattinen syöttö

Ohjelmointiprojekti 1:n lopussa meillä oli ongelmana se, että tietokanta, backend ja frontend olivat erillään toisistaan, eivätkä tiedot siirtyneet näiden osien välillä. Tämä taasen aiheutti sen, että sovellus oli käytökelvoton. Jokainen osa toimi omana osanaan, mutta mitään toimintoa ei saatu vietyä kerralla alusta loppuun. Tiedontallennusongelmiien ratkettua toteutuksesta saatuiin fiksumpi ja toimivampi. Myös tietokantayhteys saatuiin toimimaan ja yhdistettyä komponentteihin talletettuun dataan.



```

return (
  <article id="profileSection" ref={this.profileSection}>
    <div className="imgPreview">
      {image}
    </div>

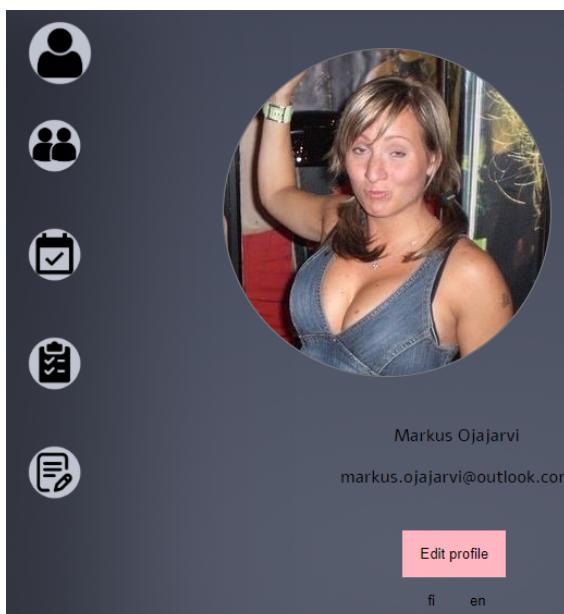
    <br/>
    <p id="profileName" ref={this.profileName}>{this.state.profileName} {this.state.profileLName}</p>
    <p id="profileEmail" ref={this.profileEmail}>{this.state.profileEmail}</p>
    <br/>
    <button id={"editProfile"} onClick={this.editProfile}>t('profile:edit')</button> <br><br>

    <button onClick={() => i18n.changeLanguage('fi')}>fi</button>
    <button onClick={() => i18n.changeLanguage('en')}>en</button>
  </article>
);

```

Kuva 6 Tietokantakutsut päivittävät komponentin sisällön automaattisesti

React renderöi komponentin uudelleen onnistuneen tietokantakutsun jälkeen (kuvat 6 ja 7).



Kuva 7 Päivittynyt profiili

### 2.3 Back end

Ohjelmointiprojekti 1:ssä vaivannut Ajax- ja JavaScript import-ongelmat saatiin hallintaan tämän jakson alussa, ja meininki olikin paljon vauhdikkaampi. Express-polkuja tuli vielä laajentaa uusien ominaisuuksien tarpeen mukaan (esimerkiksi lisää ryhmä, poista ryhmä). Ainakin osa backendin NodeJS-koodista muutettiin luokkarakenteiseksi (käyt-



täen Class-syntaksia), jotta se olisi vielä tutumpi Java-ympäristöstä. Tämä aiheutti hienman harmaita hiukkien import ja export -toimintojen kanssa, sillä nähtävästi esimerkiksi osa valmiista kirjastoista luo suoraan uuden olion pelkällä require-metodilla, mutta tehdyt kotikutoiset luokat vaativat vielä erikseen instassioliin luomista requiren jälkeen. (Kuva 8).

```
const Middleware = require('./middleware');
const middleware = new Middleware(),
module.exports = Middleware;
```

#### Kuva 8 Middlewareen importtaus ja exporttaus

## 2.4 Evästeet, sisäänskirjautuminen ja JSON Web Tokenit

Tämä ominaisuus on 95%:sti valmis, mutta jää niin viime tippaan, että testaus on koko-naan tekemättä suppeita kehityksenaikaisia manuaalisia integraatiotestejä lukuun ottamatta. Vielä epäselvää saadaanko ominaisuus toimivaksi todettuna mukaan viimeiseen projektin master-pushiin.

Tarkoituksena oli, että käyttäjän kirjautuessa sisään, palvelin lähetetään käyttäjän tietokoneelle eväste, johon on tallennettu tämän sähköpostiosoite sekä JSON Web Token (edempänä JWT) merkkijono. JWT on käyttäjän tunnistamista varten kehitetty malli, jonka token lähetetään joka ikisessä sivuston pyynnössä. Palvelin käyttää tokenia todentakseen, että käyttäjä on se, joka väittää olevansa, eikä pyynnön ja sen vastaanottamisen välillä ole päässyt kolmansia osapuolia. Tätä ominaisuutta varten projektiin asennettiin JWT- sekä cookie-parser -kirjastot, joiden avulla palvelinpäädyn NodeJS-koodissa saatiin JWT:t helpommin luotua ja varmistettua (dekoodattua).

Tämä ei tosin onnistunut ilman yli kymmenen tunnin yritys/erehdys-rumbaa, sillä JWT:n muuten hyvillä verkkosivuilla oli hyvin vähän esimerkkikoodia. Tämän vuoksi käteväät valmis-funktiot jäivät aluksi huomiotta, ja projektissa yritettiin tokenien en- ja dekoodausta täysin ”hartiavoimin”, kunnes loppujen lopuksi tarkemmalla haulla löytyi helpompiakin keinuja. Tämä täysin hukkaan heitetty aika näkyi siinä, ettei toiminnallisuutta saatu valmiaksi ennen esitystä.



## 2.5 Tietokanta

Aluksi tietokannasta tehtiin ER-kaavio, jonka avulla hahmoteltiin tauluja ja tietueita, jotka ovat tietokannassa tarpeellisia. Kaaviosta muodostettiin SQL-scripti. Tietokantascriptiä olemme joutuneet muokkaamaan projektin edetessä huomatessamme, että joitain tarpeellisia tauluja tai tietueita puuttuu, tai jokin ratkaisu ei toimi kuten toivoimme. Tietokanta on rakennettu NodeJS:n Sequelize-ORM-moduulilla, joka mahdollistaa tietokannan käsitteilyn NodeJS:llä.

```
const Profile = sequelize.define('profile', {
  ProID: {type: Orm.INTEGER, autoIncrement: true, primaryKey: true},
  FirstName: {type: Orm.STRING, allowNull: false},
  LastName: {type: Orm.STRING, allowNull: false},
  Email: {type: Orm.STRING, allowNull: false},
  Password: {type: Orm.STRING, allowNull: false}
});

Account.Profile=AccounthasOne(Profile);
```

Kuva 9 Sequelize-koodi profiili-taulun luomiseksi.

Kuvassa 9 esitetään Sequelize-koodi, jolla luodaan uusi profiilitaulu tietokantaan rekisteröitymisen yhteydessä. Käyttäjän syöttämät tiedot tallennetaan muuttuihin, joiden avulla ne viedään tietokantaan (kuva 10). Salasana kryptataan passwordhash-kirjaston avulla, ja tallennetaan kryptattuna tietokantaan. Salasanaa ei pysty dekryptaamaan.

ProID	FirstName	LastName	Email	Password	createdAt	updatedAt	accountAccID
1	Mircu	Mircu	mircu.mircu@mail.com	sha1\$cc2ca601\$1\$f9edbd74aa831c5d32e9fba0ad48ea5...	2019-05-04 16:51:50	2019-05-06 06:04:43	1
2	Teppo	Terava	terava@email.fi	sha1\$ef69d8c\$1\$abfbfb9b498a63f490a7d7a3b25c41a...	2019-05-04 16:52:47	2019-05-04 16:52:47	2
3	Paavo	Petteri	paavo@email.com	sha1\$d4fc0bb6\$1\$bb773e48fd7e1991b383b091d839c6d...	2019-05-04 16:53:18	2019-05-04 16:53:18	3
4	Jaana	Jontunen	jontunen@palika.fi	sha1\$dd7640e41\$9a8f25b9440d83b9ea5df6261dc567...	2019-05-04 16:53:48	2019-05-04 16:53:48	4
5	Kukka	Dino	dino@vaasalandia.fi	sha1\$abd81d45\$1\$6da2d6850f0b77fb5a65af3452f857...	2019-05-05 10:21:53	2019-05-05 10:21:53	5
6	Pekka	Puupaa	puupaa@masku.fi	sha1\$96cc1587\$1\$00f8aebc5ee9871656e04606fd9b859...	2019-05-05 10:29:15	2019-05-05 10:29:15	6
7	Hurta	Hettunen	huhe@email.com	sha1\$d96ed052\$1\$fc95912c648ef31992b8dafc5f194d6...	2019-05-05 10:32:06	2019-05-05 10:32:06	7

Kuva 10 Tietokannan profiili-taulu



## 2.6 Lokalisointi

Lokalisointi oli yksi saamistamme uusista vaatimuksista. Sovellus tuli käänää englanniksi ja suomeksi. Tähän käytimme Reactin valmista komponenttia nimeltään react-i18next. Lokalointia varten luotiin molemmille kielille (en, fi) omat kielikirjastot, johon sanastot luotiin (kuva 11). Lokalisoitaviin React-komponentteihin importattiin i18next:in 'withTranslation' funktio, sekä muuttuja t. Tämän lisäksi tekstikentissä viitattiin näihin kirjastojen tiedostoihin (kuva 12).

```
{
  "title": "Create a new entry",
  "formtitle": "Entry title",
  "date": "Entry date",
  "time": "Entry time",
  "remind": "Remind me",
  "private": "Private entry",
  "tag": "tag",
  "save": "Save"
}
```

Kuva 11 Kielikiraston englanninkielinen entry-tiedosto

```
<input type="text" ref={this.entryName} placeholder={t('entry:formtitle')} pattern=".{2,}" required/>
<br/><br/>
<p>{t('entry:date')}</p>
<input type="date" defaultValue={date} ref={this.entryDate} pattern=".{8,}" />
<br/>
<p>{t('entry:time')}</p>
<input type="time" ref={this.entryTime} pattern=".{6,}" />
<br/><br/>
{t('entry:remind')} <input type="checkbox" ref={this.remindCheck}/>
<br/>
{t('entry:private')} <input type="checkbox" ref={this.privateCheck}/>
<br/>
<input type="text" ref={this.entryTag} placeholder={t('entry:tag')} pattern=".{2,}" />
<br/><br/><br/>
<button id="save" type="button" onClick={this.saveEntry}>{t('entry:save')}</button>
```

Kuva 12 Lokalisoitu entry-komponentti

## 2.7 Lopputilanne

Viimeisen sprintin lopetuspalaverin jälkeen tuotteeseen oli toteutettu tärkeimmät ominaisudet, mutta osa puuttui vielä ja hiottavaa jäi runsaasti. Käyttäjä pystyy luomaan käytäjätunnukseen ja kirjautumaan sillä sisään. Käyttäjä pystyy luomaan itselleen tapahtumia ja merkintöjä, ja ne näkyvät kalenterissa (päivittyytä hieman viiveellä ajoittain, mutta nä-



kyvät kuitenkin). Käyttäjä pystyy luomaan itselleen ryhmiä ja kontakteja, ja tarkastelemaan näiden yhteisiä kalentereita, jossa näkyvät kaikkien jäsenien menot. Kun käyttäjä kirjautuu seuraavan kerran sisään, palautetaan kaikki nämä tiedot tietokannasta. Lokalisointi toimii myös kaikilla käännytyillä komponenteilla.

Nämä ollen saimme kutakuinkin toteutettua tärkeimmät ominaisuudet ja käyttäjätarinat sovelluksesta. Muutamaa käyttäjätarinaa lukuun ottamatta toteuttamatta jääneet tarinat olivat enemmän 'nice-to-have' -tyyppisiä ominaisuuksia, esimerkiksi juhlapyhien automaattinen merkintä kalenteriin tai käyttäjien blokkaaminen. Jo ohjelmistotuotantoprojekti 1:n alussa, oli selvää, että tuskin ehdimme kaikkia ominaisuuksia toteuttamaan. Halusimme merkata ne kuitenkin ylös, jotta saisimme kokonaiskuvan, millaisen tuotteen haluamme rakentaa.



### 3 Testaus

Sovelluksen testausta suoritettiin jatkuvan integroinnin palvelimen (Jenkins), käyttöliittymätestien (Sideex) ja manuaalisen testauksen avulla.

#### 3.1 Yksikkötestaus, Jenkins

Viimeisimpien sprinttien aikana ei ole tehty uusia mocha-testejä back end -puolella tietokantafunktioita varten. Tietokantatestejä ei useista yrityksistä huolimatta saatu Jenkinsin puolella toimimaan ja tällä hetkellä kaikki Jenkins buildit ovat jäyneet aborted -tilaan. Koska vanhojenkaan tietokantatestien pyörittäminen Jenkinsissä ei onnistunut, koimme, ettei tällä aikataululla ole mielekästä käyttää enempää aikaa niihin. Totesimme palavamme tietokantatestien pariin, jos ongelmat Jenkinsin suhteen saataisiin selvitettyä.

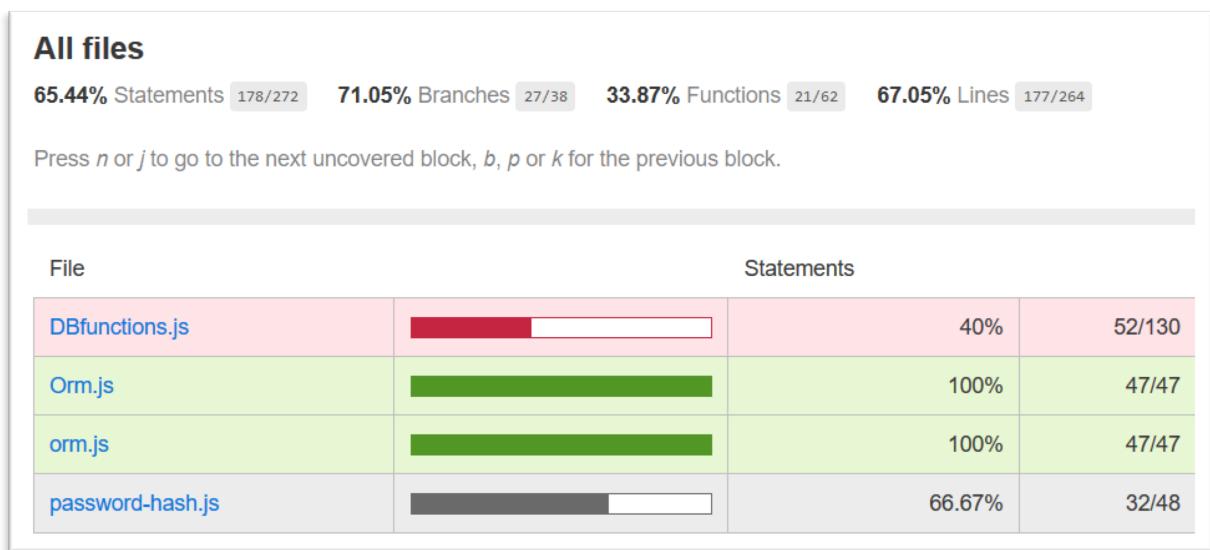
```
0 passing (229ms)
2 failing

1) DB tests
  "before all" hook:
  SequelizeDatabaseError: Table 'database_r10.notifs' doesn't exist
```

**Kuva 13 Jenkinsin virheilmoitus**

Jenkinsissä tietokantaskriptiä ei saatu etänä ajettua eikä kurssivastajankaan panostus auttanut. Tämän vuoksi syntyy kuvassa näkyvä virhe, joka kertoo, ettei yhtä tietokannan tauluista löydy (kuva 13). Kyseisen taulun nimi oli aiemmin liian pitkä, eikä sen luonti tämän takia Jenkinsin puolella onnistunut. Ongelma korjattiin, mutta edelleenkään tietokanta ei tule Jenkinsiin oikein. Omassa koodissa tietokannan käyttö on kuitenkin sujunut ongelmitta.





Kuva 14 Mocha-testien kattavuusraportti

Yritämme vielä mahdollisesti ajaa tietokantaskriptin suoraan Jenkinsissä, mutta todennäköisesti emme Jenkinsin kautta tule saamaan testejä läpi. Tämän vuoksi myöskaän Sonarqubesta ei saatu kattavuustietoja, vaikka nekin olivat paikallisesti helposti saatavilla. Uudet ominaisuudet testattiin manuaalisesti ja toiminnot tulivat testatuksi myös käyttöliittymätestien ohella. Tällä hetkellä Mochalla on toteutettu testit suurimmille tietokantafunktioille, kuten käyttäjän luonnille, tapahtuman ja menon lisäämiselle ja käyttäjien haulle (kuva 14). Kuitenkin olemme parhaamme mukaan yrittäneet paikata yksikkötestien puutteellisuutta ja Jenkinsin toimimattomuutta panostamalla erityisen paljon käyttöliittymän testaukseen Sideexillä sekä manuaaliseen testaukseen.

### 3.2 Käyttöliittymätestaus, Sideex

Sideexiä käytimme käyttöliittymän toiminnallisuuksien testaamiseen. Sideexillä oli helppo ja nopea tehdä käyttöliittymätestejä, mutta aivan mutkatonta se ei ollut. Vaikka työkalun avulla löytyikin useampi bugi ohjelmistosta, sen käyttöön ja säätämiseen käytetty aika sai pohtimaan, oliko se sen arvoista. Erityisesti ongelmat email-kentän ja alert-vikaviestien tulkinnassa haittasi testausta. Taulukossa (taulukko 1) on kuvattu muutama esimerkki Sideexillä tuotetuista testisarjoista. Näiden lisäksi tuotettiin muutamia yksittäisiä testitapauksia, joilla testattiin täsmällisesti tiettyjä ominaisuuksia. Koimme, ettei kaikien testitapauksien luetteleminen ja taulukoiminen olisi tarkoituksenmukaista raportin pituus huomioon ottaen.



Testisarja	Testitapaukset	Tulokset	Huomioita
Merkinnän luonti	1. Tiedot oikein 2. Nimi puutteellinen 3. Aika puuttuu 4. Syötetty päivämäärä mennyt jo 5. Erikoismerkkejä kuvauksessa	1. Passed 2. Fail, fixed, passed 3. Fail 4. Passed 5. Passed	Kohta 2; nimen vähimmäismerkkimäärä on 2, mutta jos kentän jättää tyhjäksi meni testi läpi. Korjattiin. Kohta 3, Sideex ei osannut käsitellä alerteja oikein.
Muistiinpanon luonti	1. Tiedot oikein 2. Nimi puuttuu 3. Teksti puuttuu 4. Syötteenä erikoismerkkejä 5. Muistiinpanossa 1000 merkkiä	1. Passed 2. Fail 3. Fail 4. Passed 5. Passed	2. ja 3., työkalu ei osannut käsitellä oikein alerteja -> vaikka syöte vastasi odotettua, ei testi mennyt läpi
Profiilin tietojen muokkaus	1. Nimen vaihto 2. Sähköpostin vaihto 3. Tietojen poisto	1. Passed 2. Passed 3. Passed	
Ryhmän muodostaminen	1. Uuden ryhmän luonti 2. Tyhjän ryhmän luonti 3. Sama käyttäjä useaan kertaan samassa ryhmässä 4. Sama käyttäjä useassa eri ryhmässä	1. Passed 2. Passed 3. Passed 4. Passed	
Kontaktin luominen	1. Uuden kontaktin luominen 2. Kontaktoutuminen jo olemassa olevan kontaktin kanssa	1. Passed 2. Passed	

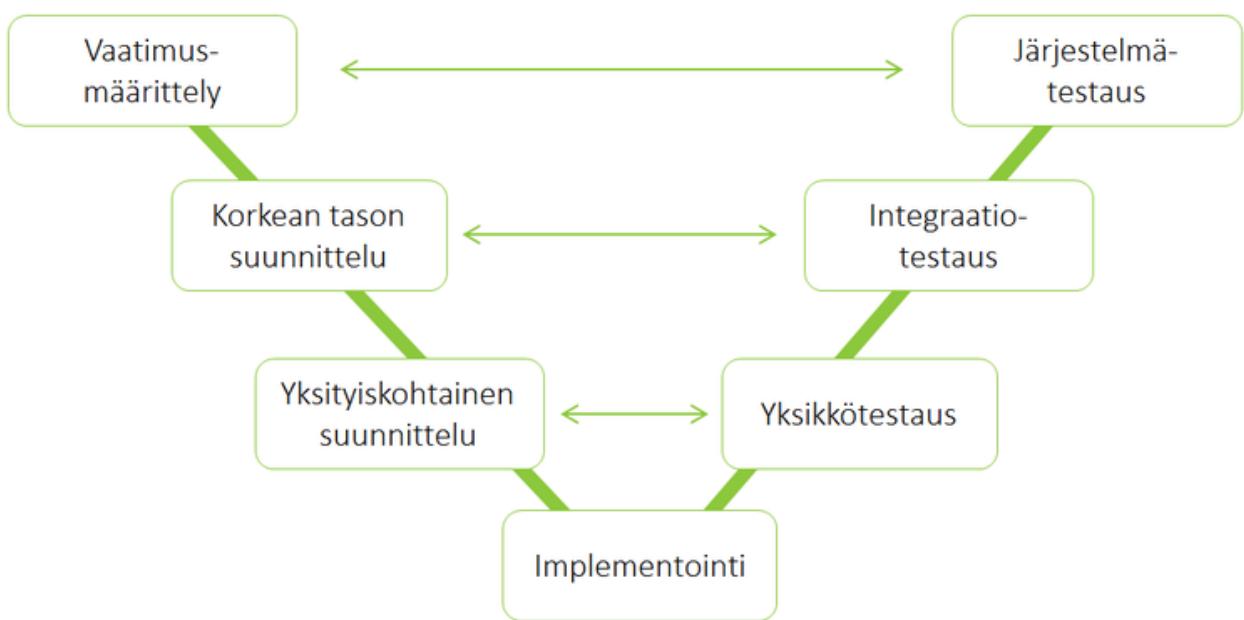
Taulukko 1 Esimerkkejä Sideexillä luoduista testisarjoista ja -tapausklistista.



Sideexillä tuotetuissa testeissä oli ongelmana myös koko ajan muuttuva käyttöliittymä. Käyttöliittymään lisättiin lähes päivittäin elementtejä sekä niiden sijainteja ja toiminnallisuksia muutettiin. Tämä aiheutti sen, että aiemmin tehdyt Sideex-testit eivät enää hetken päästä toimineetkaan, vaan koko testisarja tuli päivittää. Muutaman kerran ne päivitetiinkin projektin aikana, mutta lopulta testien ylläpitämiseen käytetty aika alkoi kasvaa niin suureksi, että oli kannattavampaa panostaa manuaaliseen testaukseen.

### 3.3 Manuaalinen testaus

Manuaalinen testaus on ollut iso (ja luultavasti merkittävin osa) ohjelmistomme testausta. Testaus on jakautunut yksikkö-, integraatio-, järjestelmätestaukseen (kuva 15) sekä vikojen korjaamisen jälkeisiin uudelleen- ja regressiotestaukseen.



Kuva 15 Testauksen tasot esitettynä V-mallissa.

Yksikkötestauksessa testattiin kehitetty komponentti tai ohjelman osa, ennen kun se implementoitiin koko ohjelmaan. Yleensä yksikkötestattava osa saattoi olla esimerkiksi käyttäjätarinasta luodun taskin toteutus. Tämän osan toteutti yleensä sama henkilö, joka on testattavan osan luonut. Siinä vaiheessa, kun yksikkötestit menivät halutulla tavalla läpi, voitiin kyseinen ominaisuus/ohjelmiston osa integroida sovellukseen. Yksikkötestien



tekeminen kuuluu osana käyttäjätarinoiden toteutukseen, joten siitä ei kenellekään erikseen merkityt tunteja Agilefantiin, vaan ne kuuluivat osana kyseisen taskin toteutusta.

Integraatiotestit suoritettiin siinä vaiheessa, kun yksikkötestit läpäissyt ominaisuus voitiin liittää muuhun ohjelmistoon. Tällöin testattiin, että sovellus toimii kokonaisuutena, kun osat liitetään yhteen. Muutamia erikoisia bugeja löytyi, ja osaa emme saaneet korjattua. Esimerkiksi loimme lomakkeen asetusten tallennusta varten, jossa käyttäjä pystyi valitsemama, kuka pystyy tarkastelemaan hänen tietojaan. Lomakkeessa valinnat tehtiin radiobuttoneilla, ja jokaisessa asetuksessa oli kolme vaihtoehtoa, joista valittiin yksi, oletuksena vaihtoehtoista keskimmäinen. Lomake toimi html-ympäristössä ongelmitta, mutta kun se integroitiin React-komponenttiin, radiobuttonit hajosivat. Jos valinnan halusi vaihtaa keskimmäisestä painikkeesta ylimpään, tuli ylintä painiketta klikata kahdesti. Jos tämän jälkeen valinnan halusi siirtää alimpaan valintaan, siirtyi kerran klikkaamalla valinta keskimmäiseen painikkeeseen, jonka jälkeen vasta toisella klikkauksella se saatiin siirrettyä alimpaan vaihtoehtoon. Vian syytä yritettiin jäljittää, mutta aikataulupaineiden vuoksi totesimme sen olevan niin matalan prioriteetin vika, että päätimme keskittyä graavimpien vikojen korjaamiseen.

Kun osat on saatu integroitua onnistuneesti, testasimme vielä koko järjestelmää manuaalisesti. Kokeilimme kuinka uuden käyttäjän rekisteröiminen, sisäänkirjautuminen ja ohjelmiston toimintojen käyttö onnistuu, ja mitä virheitä käytön aikana mahdollisesti esiintyy.

Vikojen korjaamisen jälkeen teimme uudelleentestaukset nähdäksemme, että vika on korjaantunut niin kuin oli tarkoitus. Kun uudelleentestauksesta saatiin halutut tulokset, suoritettiin regressiotestit. Regressiotestien tarkoituksesta on testata korjattua toimintoa ympäröiviä komponentteja ja niiden toimintaa, sekä varmistaa, ettei siellä hajonnut mitään. Regressiotesteissä harvemmin ilmeni mitään mitä ei heti huomannut Reactin elementirakenteen vuoksi. Jos korjatessa oli käynyt joku virhe, ei komponentteja usein ladattu ollenkaan. Joten virheen pystyi toteamaan heti alkunsa, kun joko PHP-Storm kieltyyi buildaamasta sivua, tai sen tehtyään sivusta puuttuvat elementit.



## 4 Reflektointi

### 4.1 Onnistumiset

Koemme, että opimme hyvin projektissa eri kieliä, ja saimme näillä hyvin toteutettua projektia. Uusia tekniikoita ja työtapoja tuli paljon, mutta onnistuimme sisäistämään näistä ison osa, ja hyödyntämään niitä projektissamme.

Opimme myös paljon projektinhallinnasta; scrum-menetelmästä ja ketterästä kehityksestä. Tiimityömme kehittyi koko ajan, ja kommunikointimme parani loppua kohden. tässä riittää kyllä vielä kaikilla töitä. Tiimissä oli kaikkien mielestä helppo työskennellä; kaikille jäi sellainen olo, että jokainen osallistui täysillä projektin tekoon ja teki parhaansa. Kaikki tekivät oman osansa, pyysivät apua tarvittaessa ja auttoivat toisiaan. Vaikka jo-kaisella oli tavallaan oma osa-alueensa, niin autettiin toista, jos toinen oli jumissa. Paljon teimme myös niin sanottua parikoodaamista.

### 4.2 Kehitettävää

Kommunikointi ryhmän sisällä osoittautui välillä yllättävän vaikeaksi. saatoimme keskus-tella jostain asiasta kauan ja olla siitä eri mieltä, kunnes lopulta kävi ilmi, että olemme puhuneet aivan eriasioista. Myös kiire ja aikataulupaineet vaikuttivat tähän; aina ei huo-mannut pysähtyä kuuntelemaan kunnolla mitä toinen sanoo. Puhuimme kyllä paljon, ja kerroimme mitä teemme ja mitä ongelmia on. emme vaan aina kuunnelleet toisiamme kunnolla.

Työmäärän arvointi meni myös alussa pieleen. Yliarvioimme, kuinka paljon ehdimme tekemään kevään aikana. Myös muutama useamman viikon kestänyt ongelma romutti aikataulua, jonka takia paljon ominaisuuksia jäi toteuttamatta.

Myös uusien teknikkoiden opettelu aiheutti välillä ongelmia. Tietoa oli ajoittain vaikea löy-tää, ja aikaa kului hirveästi tiedon etsimiseen.



#### 4.3 Yhteenveto

Loppujen lopuksi kaikki oppivat ja kehittyivät paljon projektin aikana. Aikatauluongelmien vuoksi projekti tuntui jäävän hieman kesken, mutta olemme silti tyytyväisiä siitä, mitä saimme aikaan. Ajoittaisia stressin ja väsymyksen aiheuttamia kiukkuksia ja ärsyyntymisiä lukuun ottamatta koko ryhmä on nauttinut projektin teosta yhdessä ja sen tuomista onnistumisista (kuva 16). Jatkossa osaamme toimia kehityskohteiden osalta paremmin, ja osaamme välttää (ainakin osan) tekemistämme virheistä.



Kuva 16 Ryhmä 10 viettämässä vapaaailtaa koodauksesta vappurientojen merkeissä.



## Lähteet

1. Kuva 13, Jyväskylän yliopisto, <http://smarteducation.jyu.fi/projektit/systech/Periaatteet-suunnittelun-periaatteet/testaus/testaus-lyhyesti>

