TAMPEREEN
AMMATTIKORKEAKOULU

LOMPSA

Maatalouslomituksen palvelusuunnitelmien hallinta-järjestelmä

Teemu Kanerva

Opinnäytetyö
Toukokuu 2016
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

KANERVA TEEMU:
Maatalouslomituksen palvelusuunnitelmien hallintajärjestelmä

Opinnäytetyö 41 sivua
Toukokuu 2016

Opinnäytetyön tarkoituksena oli toteuttaa aikaisemmasta maatalouslomituksen palvelusuunnitelmien hallintajärjestelmästä uusi versio nykyaikaisilla tekniikoilla. Tekniikoiksi valittiin Meteor-verkko -ohjelmistokehys ja käyttöliittymän toteutukseen React JavaScript-kirjasto. Toimeksiantajana työlle oli Tampereen ammattikorkeakoulu.

Maatilan palvelusuunnitelmien hallintajärjestelmä LOMPSA on tarkoitus ulkoistaa ja digitalisoida maatilojen lomittajien rekrytoimiseen ja lomitustarpeisiin vaadittu työ kolmannelle osapuolelle.

LOMPSA-järjestelmän käyttäjät jakautuvat kolmeen eri käyttäjärooliin: yrittäjiin, lomittajiin ja palveluohjaajiin. Tässä opinnäytetyössä on toteutettu palveluohjaajien osuutta sovelluskokonaisuudesta.

Opinnäytetyössä esitellään sovelluksen käyttökohde, toteutettu sovellus, käytetyt ohjelmistoympäristöt, ohjelmointikielet ja kolmannen osapuolen moduulit.

Asiasanat: meteor, react, lompsa, nykyaikainen verkkosovelluskehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT engineering
Software engineering

KANERVA TEEMU
Farm substitute management system

Bachelor's thesis 41 pages, appendices 0 pages
May 2016

In this final thesis the goal was to redevelop and update a new farm substitute management system from a previous version using the latest technologies. Meteor was chosen for the web framework and React JavaScript library was chosen for developing the user interface. Customer for development of the software was Tampere University of Applied Sciences.

The purpose of farm substitute management system LOMPSA is to outsource and digitalize the task of finding a qualified substitute worker for a required job.

LOMPSA users are divided to three user roles: farm owner, substitute worker and service officer. This thesis includes the implementation of the role of the service officer.

Final thesis introduces software domain, developed software, frameworks, programming languages, and third party modules.

Key words: meteor, react, lompsa, modern web development

SISÄLLYS

ERITYISSANASTO

Atmosphere	Meteorin oma paketinhallintajärjestelmä
ES6	JavaScript-kielen päivitys, joka tuo mukanaan uusia ominaisuuksia JavaScript-ohjelmointiin
Full-Stack	Ohjelmistokohteen kerrokset: asiakas, palvelin ja tietokanta
Importtaus	Tuodaan funktioita tai olioita tiedostoon ulkopuolisista moduuleista
JavaScript	Netscape Communications Corporationin kehittämä verkko-ohjelmointikieli
LOMPSA	Maatalouslomituksen palvelusuunnitelmien hallintajärjestelmä
Meteor	Full-Stack-ohjelmistokehys modernien verkkosovellusten toteuttamiseen.
MongoDB	Dokumenttipohjainen, ei relaatiomallinen tietokanta
Node.js	Asynkroninen tapahtumapohjainen ja ajonaikainen JavaScript-palvelinpohjaisten sovellusten alusta
NoSQL	Ei relaatiomallinen tietokanta
NPM	Paketinhallinta järjestelmä kolmannen osapuolen JavaScript-paketeille
Parsia	Muuntaa syötetiedon toiseen rakenteeseen
PHP	Verkkoympäristössä käytettävä dynaamisten verkkosivujen ohjelmointikieli
React	Facebookin ja Instagramin kehittämä käyttöliittymän toteutukseen kehitetty JavaScript-kirjasto
Robomongo	Graafinen mongoDB tietokannan hallintatyökalu
SQL	Relaatiomallinen tietokanta

1 JOHDANTO

Maatalouden lomituspalveluja varten kehitettiin LOMPSA-järjestelmän ensimmäinen versio vuonna 2014. Siinä käytettiin PHP- ja JavaScript-ohjelmointikieliä. Tietokantana käytettiin SQL-tietokantaa. Järjestelmästä toteutettiin palveluohjaajien osuus ja sitä käytiin esittelemässä kohdeyritykselle.

Opinnäytetyön tarkoitus on LOMPSA-järjestelmän uudelleen toteutus, käyttäen Meteor -ohjelmistokehystä ja React JavaScript-kirjastoa. Aikaisempaa versiota käytetään uuden version toteutuksessa referenssinä. Järjestelmää lähdettiin toteuttamaan osana TAMKin ohjelmistotekniikan kurssia.

Järjestelmä haluttiin toteuttaa uudelleen nykyaikaisilla tekniikoilla. Niiden tarkoituksena on eliminoida aikaisemmassa versiossa havaitut puutteet ja ongelmat. Meteor-ohjelmistokehyksen etuna on se, että se toimii full-stack JavaScript-ohjelmistokehyksenä, jolla tarkoitetaan koko sovelluksen kehitysalueita.

Opinnäytetyön toisessa luvussa käsitellään LOMPSA-järjestelmän tarkoitusta ja selvitetään mikä on järjestelmän käyttökohde.

Kolmannessa osiossa kuvataan lyhyesti työssä käytetyt ohjelmointikielet ja ohjelmistot kuten editorit ja versionhallintatyökalut.

Neljännessä osiossa esitellään Meteor-ohjelmistokehyksen rakennetta tarkkailemalla sen arkkitehtuuria ja toimintatapoja.

Viidennessä osiossa selostetaan React-kirjaston rakennetta ja tarkastellaan React-luokan sisältöä esimerkin avulla.

Kuudennessa osiossa käydään läpi LOMPSA-sovelluksen toteutusta, näkymien ja kehitettyjen toimintojen osalta.

Viimeisessä osiossa tarkastellaan LOMPSA-järjestelmän jatkokehitykseen liittyviä tekijöitä.

2 MAATALOUSLOMITUKSEN DIGITALISOINTI

Projektin tavoitteena on kehittää julkisen palvelun puolelle sovellus, jonka avulla voidaan lisätä palvelujen tuottavuutta kunnallisissa ja yksityisissä maatalouden lomituspalveluissa.

Projekti-ideaa on kehitelty jo 1990–luvulta lähtien. Aihe ja siihen liittyvät tarpeet ovat tulleet esiin lukuisissa koulutuksissa, joita Ahlmanin ammatti- ja aikuisopisto on järjestänyt valtakunnallisesti maatalousalalla niin työnjohdon, työntekijöiden kuin palvelun asiakkaiden puolelta.

Lomituspalvelut ovat osa julkista hyvinvointipalvelujen tuottamista ja avainasemassa maaseudun hyvinvoinnin kehittämisessä. Tähän mennessä haastavaksi palvelun tuottamisen on tehnyt juuri kommunikaation puute järjestelmän käyttäjien välillä. Erityisen haastavaa palvelujen tuottamisessa on ollut maatilan tietojen saaminen lomittajalle, mistä on aiheutunut mm. hengenvaarallisiakin työturvallisuusongelmia sekä palautteen saaminen viljelijöiltä lomittajien toiminnasta ennen kuin ongelmat kehittyvät. Maatalousyrittäjän, julkisen palvelun tuottajan (lomitusyksikkö) ja palvelutyöntekijän (lomittaja) välisessä yhteistyössä kulminoituvat useat julkisen ja yritysyhteisön palvelutuotannon haasteet haja-asutusalueella.

2.1 Projektin kohderyhmä

Projektin kohderyhmänä on kunnallinen tai yksityinen palveluntuottaja, lomituspalvelu, jonka toimintaa pyritään tehostamaan toteutetun järjestelmän käytöllä. Lomituspalveluissa on kolme kohderyhmäjoukkoa. Näitä ovat:

1. Lomittajat
2. Lomitusohjaajat, jotka toimivat työn koordinoijina ja työnantajina lomittajille
3. Maatalousyrittäjät

Projektin avulla keskustelu ja yhteistyö näiden kolmen tahon välillä lisääntyy ja tehostuu merkittävästi. Lomituspalvelusopimus on lomituspalvelun ja yrittäjien välinen sopimus, joka sisältää myös lomittajalle tarpeellisia tietoja, joita hän voi hyödyntää työssään.

3 KÄYTETYT TYÖKALUT JA TEKNOLOGIAT

Tässä osiossa käsitellään sovelluksen tuottamiseen käytettyjä ohjelmointikieliä ja aputyökaluja. Tämän lisäksi selvitetään työkalujen käyttötarkoitusta.

3.1 Sublime Text-tekstieditori

Projektiin valittiin käytettäväksi Sublime Text-tekstieditori, koska se on kevyt ja helposti muokattavissa. Siihen pystyy lisäämään sovelluksen omalla paketinhallintajärjestelmällä lisämoduuleita. Moduulit helpottavat kehitystä antamalla käyttäjälle pikanäppäimet koodileikkeleitten luontiin ja värityksen tietylle syntaksille. (Sublime Text: Sublime Text.)

3.2 Robomongo-sovelluskehitin

Robomongo on MongoDB-tyyppisten tietokantojen käyttöön tarkoitettu graafinen sovellus. Robomongolla pystytään tekemään MongoDB-kannan komentoja graafisessa ympäristössä. Tämä helpottaa tietokantaan tehtyjen muutoksien tarkastelua ja testausta. (Robomongo: 2016.)

Robomongoa on käytetty tässä projektissa helppokäyttöisyytensä takia. Sovelluksella on ollut nopeaa ja yksinkertaista suunnitella ja selvittää toteutettavan tietokannan rakenteita.

3.3 Versionhallinta

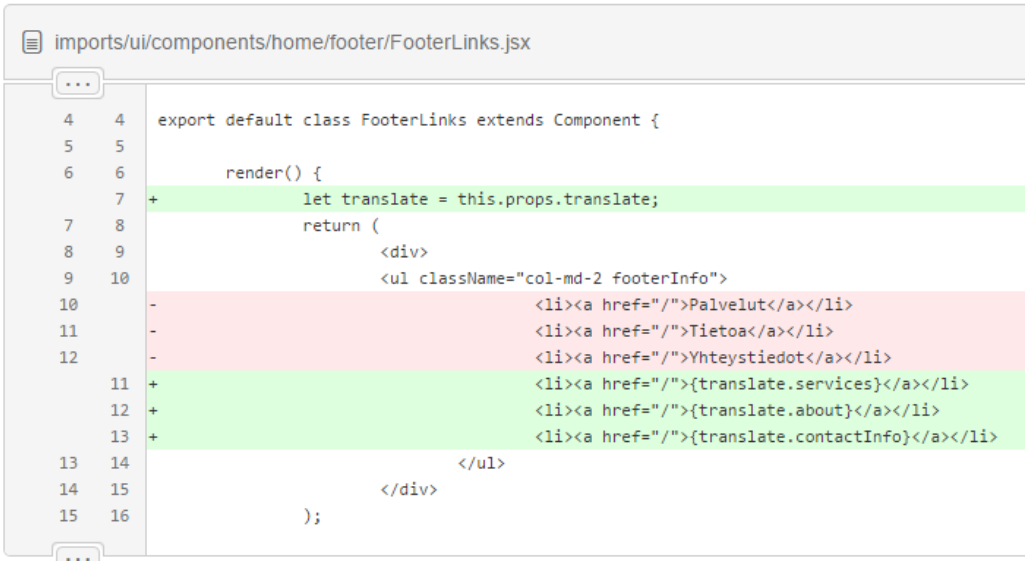
Versionhallintatyökaluina on käytetty projektissa Bitbucket- ja Git-ohjelmistoympäristöjä. Bitbucket on pilvipalvelussa sijaitseva tallennustila, johon voidaan tallettaa versioita sovelluksesta Git-versionhallintatyökalua käyttäen.

Ohjelmaa toteuttaessa on omalla tietokoneella versio sovelluksesta ja Bitbucket-pilvipalvelimella toinen. Näitä kahta versioita Git vertaa toisiinsa ja sen mukaan tiedostoja päällekirjoitetaan pilvipalvelun versioon ohjelmasta. Bitbuckettiin lähetetään muokattu versio

talletettavaksi, kun osa toteutuksista on saatu tehtyä. Tällöin uusin versio on aina varmuuskopioitu ja muut projektissa olevat käyttäjät pääsevät lähdekoodiin helposti käsiksi. Pilvessä sijaitsevasta projektin versiohistoriasta (kuva 1) voidaan selvittää ohjelman muutoshistoria. Mikäli toteutettavassa ohjelmassa on ilmennyt ongelmia, pystytään siis palaamaan takaisin aikaisempaan versioon. (Git. 2016.)

Versionhallinnan hyvinä puolina on se, että useampi kehittäjä pääsee tekemään samanaikaisesti toteutettavaa ohjelmistoa. Tällöin sovelluksen ylläpitäminen ja toteutus helpottuvat.

Git on ilmainen ja avoimen lähdekoodin versionhallintajärjestelmä. Git on helppokäyttöinen ja nopea ja sitä käytetään komentotilassa. Git-työkaluun on saatavilla graafisilla käyttöliittymillä varustettuja lisäosia ja tässä projektissa on käytetty TortoiseGit-versionhallintatyökalua.



KUVA 1. Bitbucket-pilvipalvelun muutoshistoria

3.4 HTML5- ohjelmointikieli

HTML5 (HyperText Markup Language) on standardi merkintäkieli verkkosivujen kehitykseen. HTML-teknologiaa käytetään verkkosivujen luomisen lisäksi käyttöliittymien toteutukseen mobiililaitteissa ja verkkosovelluksissa. HTML-elementteihin voidaan lisätä JavaScript-toimintoja ja elementtejä voidaan tyylitellä CSSn (Cascading Style Sheets) tyyliohjeilla.

3.5 JavaScript-ohjelmointikieli

JavaScript on kevyt ja korkeantason dynaaminen tulkattu ohjelmointikieli. Se on yksi tunnetuimmista ohjelmointikielistä verkkoympäristössä. Se on standardoitu EC-MAScript-kielen spesifikaatioiden mukaisesti. JavaScript on yksi kolmesta keskeisimmästä ohjelmointikielestä, joita käytetään verkkosovellusten kehittämiseen.

JavaScriptiä ei tule sekoittaa Java-ohjelmointikieleen. Kummatkin Java ja JavaScript ovat tavaramerkkinä Oracle tuoteryhmää, mutta niiden syntaksi, semantiikka ja käyttö eroavat toisistaan.

3.6 JSON-formaatti

JSON (JavaScript Object Notation) on kevyt tiedon käsittelyn formaatti. Sitä on helppo kirjoittaa ja lukea. Sitä on myös helppo koneiden parsia ja generoida. JSON pohjautuu JavaScript-ohjelmointikielen osajoukkoon. JSON on tekstiformaattina kieliriippumaton, mutta se käyttää tapoja, jotka ovat tuttuja C-ohjelmointikieliperheelle, kuten C, C++, C#, Java, JavaScript, Perl, Python ja moni muu. Nämä ominaisuudet tekevät JSONista ideaalisen datan käsittelykielen. (JSON.org: Introducing JSON.)

JSON muodostetaan kahdesta rakenteesta.

1. Kokoelmasta avain-arvo-pareja. Useissa ohjelmointikielissä tätä kutsutaan olioksi.
2. Järjestetystä listasta arvoja. Useimmissa ohjelmointikielissä näitä kutsutaan taulukoksi, vektoriksi, listaksi tai sekvenssiksi.

3.7 Sass-tyyliohjelaajennus

Sass on laajennus CSS (Cascading Style Sheets) tyyliohjeisiin, joka lisää tehokkuutta ja tyylikkyyttä CSS-syntaksiin. Sass mahdollistaa tavan käyttää muuttujia, perintäsääntöjä, sisäkkäisyyttä ja uudelleen käytettävyyttä ilman tarvetta HTML-viittauksiin. Sass auttaa pitämään isot tyylitiedostot hyvin organisoituina ja nopeuttaa saamaan pienet tyylitiedostot käyttöön. (Sass: Sass Basics.)

3.8 MongoDB-tietokanta

MongoDB on ilmainen ja avoimen lähdekoodin dokumenttityylinen tietokanta. MongoDB on tyypiltään NoSQL-tietokanta, jolla kuvataan perinteisestä relaatiotietokannasta poikkeavaa teknologiaa. Relaatiotietokannat soveltuvat huonosti suurten dokumenttimäärien käsittelyyn ja pilvilaskennan käyttöön. NoSQL-tyyppiset tietokannat sopivat hyvin applikaatioihin, joissa generoidaan suuria määriä uutta ja jatkuvasti muuttuvaa dataa. (MongoDB: Introduction to MongoDB.)

NoSQL-tietokannat jakautuvat neljään eri tietokantatyyppiin.

1. Dokumenttikantoihin, jossa määritetään jokaiselle avaimelle kompleksi datastruktuuri, joka tunnetaan dokumenttina. Dokumentti sisältää useita eri avain-arvo-pareja, avainarvotaulukkopareja tai sisäkkäisiä dokumentteja.
2. Kuvaajatiloihin, joita käytetään varastoimaan verkostoitua dataa, kuten sosiaalisia yhteyksiä
3. Avain-arvo-parin tallennustiloihin, jotka ovat yksinkertaisin muoto NoSQL-tietokannasta. Jokainen olio tietokannassa on tallennettuna attribuuttina, jolla on avaimen nimi ja arvo.
4. Leveisiin kolumnin tallennustiloihin, jotka ovat optimoitu suuriin hakuihin

3.9 Paketinhallintajärjestelmä npm

Npm on paketinhallintajärjestelmä, joka on toteutettu JavaScript-kehittäjille koodin jakamiseen ja koodin uudelleen käyttämistä varten. Hyötynä JavaScript-kehittäjille paketinhallintajärjestelmässä on se, että kehittäjä voi helposti hakea omissa töissä ilmenneisiin ongelmiin ratkaisuja ulkopuolisista moduuleista ja tarkastella moduuliin tehtyjä päivityksiä paketinhallintajärjestelmän omilla toiminnoilla. (npm: What is npm?.)

npm on kuitenkin enemmän kuin pelkkä kolmansien osapuolien moduuleitten lataustyökalu. npm-sivuilta voidaan etsiä moduuleita projektiin ja tarkastella pakettien latausmääriä, moduuleitten käyttöönotto-ohjeita ja paketin kehittäjiä.

Projektissa käytetyistä moduuleista npm muodostaa JSON-muotoisen pakettitiedoston nimeltä package.json. Tiedoston tehtävänä on pitää sisällään dokumentaationa projektissa käytettyjä npm-paketteja, tietyn moduulin version ja muodostaa sovelluksesta helposti uudelleen käytettävän, jolloin sen jakaminen helpottuu muille käyttäjille.

4 METEOR-OHJELMISTOKEHYS

Meteor on full-stack JavaScript-ohjelmistokehys, modernien verkkosovellusten toteutukseen, joka on tehty Node.jsn päälle. Full-stack-termillä tarkoitetaan, että Meteor-projekti sisältää asiakas-, palvelin- ja tietokanta-alueen yhden sovelluksen alla. Meteorissa käytetään JavaScript-ohjelmointikieltä koko sovellusalueella.

Meteorilla pystytään toteuttamaan useammalle eri alustalle verkkosovelluksia kuten Android- ja iOS-alustoille. Meteor sisältää avainasemassa olevat kehittyneet teknologiat reaktiivisten verkkosovellusten kehittämiseen kuten: ohjelman käännöstyökalun, node.js-paketit ja JavaScript-yhteisön tuen. (Meteor.com: Introduction.)

4.1 Julkaisu- ja tilaaja-arkkitehtuuri

Meteor käyttää tilaaja / julkaisu-ohjelmistoarkkitehtuuria. Tämä näkyy Meteorissa siten, että palvelinpuolelta julkaistaan dataa asiakkaille ja asiakkaat tilaavat näitä julkaisuja tarpeidensa mukaisesti. Tällä voidaan eksplisiittisesti määrittää, mitä dataa palvelin tarjoaa asiakkaalle. (Meteor.com: Publish and subscribe.)

Järjestelmässä jokaisessa näkymässä, jossa käytetään tietokannan dataa, pitää asiakkaan tilata kyseiseen datankäsittelyyn tarvittava data palvelimelta.

4.2 Meteor-projektirakenne

Jotta Meteor-ohjelmistokehystä voitaisiin käyttää tehokkaasti hyödyksi ja varmistaa että käytettävää koodia ajetaan vain silloin kun on tarve, niin on hyvä noudattaa Meteorin omaa tiedostorakennetta. Meteori noudattaa seuraavaa tiedostojen latausjärjestystä.

1. HTML-mallit
2. Tiedostot, joiden nimet alkavat main sanalla ladataan viimeiseksi
3. Tiedostot, jotka ovat ’lib/’-kansion alla ladataan seuraavaksi
4. Tiedostot, joissa polkujen pituudet ovat suuria
5. Loput tiedostot ladataan aakkosjärjestyksessä

Meteor noudattaa siis selkeää kansiorakennetta. Kuvasta 2 voidaan nähdä millä tavalla projektin kansiorakenne koostuu. Kansiorakenne jakautuu kahteen pääkansioon, ui ja api. Ui-kansiossa sijaitsee kaikki asiakaspuolen kooditiedostot kuten näkymät ja yksittäiset apukomponentit. Api-kansio sisältää palvelinpuolen koodit, joihin lukeutuu tietokannan rajapintatoiminnot ja julkaisumetodit.

```
imports/
  startup/
    client/
      index.js                # import client startup through a single index entry point
      routes.js               # set up all routes in the app
      useraccounts-configuration.js # configure login templates
    server/
      fixtures.js             # fill the DB with example data on startup
      index.js                # import server startup through a single index entry point

  api/
    lists/                    # a unit of domain logic
      server/
        publications.js       # all list-related publications
        publications.tests.js # tests for the list publications
      lists.js                # definition of the Lists collection
      lists.tests.js          # tests for the behavior of that collection
      methods.js              # methods related to lists
      methods.tests.js        # tests for those methods

  ui/
    components/               # all reusable components in the application
                              # can be split by domain if there are many
    layouts/                  # wrapper components for behaviour and visuals
    pages/                    # entry points for rendering used by the router

client/
  main.js                     # client entry point, imports all client code

server/
  main.js                     # server entry point, imports all server code
```

KUVA. 2 Meteori-projektin kansiohierarkia

4.3 DDP-asiakaspalvelinprotokolla

Meteor käyttää DDP (Distributed Data Protocol) asiakaspalvelinprotokollaa tietokannan kyselyihin ja päivityksiin. DDP käyttää julkaisu-tilaaja ohjelmistoarkkitehtuuria ja se on luotu Meteor-ympäristöön. DDP-toiminta perustuu kahteen osa-alueeseen. Se käsittelee RPC-kutsuja (Remote Procedure Call) ja hallitsee dataa.

Kuvassa 3 näkyy, kuinka asiakaspuolelta kutsutaan palvelinpuolen rajapinnan metodia Meteor.call-kutsulla, jolle annetaan parametrina palvelimella määritetyn metodin nimi ja lähetettävä data.

```
// Tätä koodia ajetaan vain asiakaspuolella
if(Meteor.isClient){
    var name = "Example data";
    // Kutsutaan palvelinpuolen metodia exampleFunctio ja annetaan parametrina name muuttuja
    Meteor.call("exampleFunction",name);
}

// Tätä koodia ajetaan vain palvelinpuolella
if(Meteor.isServer){
    Meteor.methods({

        //Palvelinpuolen metodi jota ajetaan asiakaspuolelta
        exampleFunction(data){
            console.log("This is your data", + data);
        }
    });
}
```

KUVA 3. DDP käyttö Meteor ympäristössä

4.4 Asiakaspuoli

Meteorin asiakaspuolella eli selainympäristössä toimivat koodit jakautuvat kaikkiin sovelluksen graafisiin käyttöliittymän toimintoihin. Näitä ovat React-luokat, blaze-mallit, reititinmoduuli ja SCSS-tyylitiedostot.

4.5 Blaze-käyttöliittymä kirjasto

Blaze on osana Meteorin sisäänrakennettua reaktiivista käyttöliittymän toteutukseen tarkoitettua kirjastoa. Yleensä näkymäkomponentit ovat kirjoitettu käyttäen Meteorin Spacebars-mallikirjastoa mukaillen, tai sen vaihtoehtoista tapaa Handlebars, jotka ovat suunniteltu käyttämään Meteorin reaktiivisen datan käsittelyn Tracker-moduulia.

Blazea ei tarvitse käyttää Meteor-applikaatioon, vaan sen sijasta voidaan helposti käyttää React- tai Angular-kirjastoja käyttöliittymän toteutukseen. Blazea on käytetty LOMPSA-projektin sähköpostiviestien ulkoasun toteuttamisessa. (Meteor.com: Blaze.)

4.6 Palvelinpuoli

Meteorin palvelinpuolta ajetaan käyttäen Node.js. Palvelinpuolella määritetään kaikki palvelimella toimivat Meteori-metodit, joilla toteutetaan tietokannan rajapinta. Tämän lisäksi palvelinpuoli pitää sisällänsä julkaisumetodit tietokannan hakuoperaatioihin ja MongoDB-yhteyksien muuttujat.

4.7 Atmosphere-paketinhallintajärjestelmä

Atmosphere on Meteorin oma paketinhallintajärjestelmä, josta saa helposti käyttöön kolmannen osapuolen Meteor-paketit. Atmosphere-paketit ovat spesifioitu Meteorin käyttöön ja niissä on useita etuja verrattuna npm-paketinhallintajärjestelmään. Etuihin lukeutuu DDPn, Blazen ja Meteor-kääntötyökalun suora tuki. Tämän lisäksi etuja ovat eksplisiittisesti JavaScript, Less, Sass ja muiden tiedostojen käyttöönotto toteutettavassa ohjelmassa.

Meteorin version 1.3 julkaisun jälkeen Meteor tukee täysin npm-paketteja ja lupaa, että tulevaisuudessa paketit siirtyvät kokonaan npm:n ladattaviksi moduuleiksi. (Meteor.com: Atmosphere vs. npm.)

5 REACT-KÄYTTÖLIITTYMÄ

React on JavaScript kirjasto, joka on suunniteltu verkkosovellusten käyttöliittymän toteutukseen. Reactin on kehittänyt Facebook ja Instagram kehittäjät. Reactia voitaisiin ajatella MVC-ohjelmistoarkkitehtuurin V-osana, joka tulee sanasta view ja sillä tarkoitetaan käyttöliittymänäkymää.

JavaScriptillä luodaan React-luokkia, joita kutsutaan komponenteiksi. Reactin toimintatapa perustuu uudelleen käytettäviin komponentteihin, mikä osaltaan helpottaa koodin käyttöä ja testausta.

React-tiedostoissa suositellaan käytettävän JSX JavaScript-syntaksia. JSX suositellaan käytettävän sen kompaktiuden ja tutun käytön puurakenteiseen malliin. JSX on XMLn kaltainen syntaksin jatke ECMAScriptiin ilman määritettyjä semanttisuuksia. (React: Why React?.)

5.1 React-luokan rakenne Meteor ympäristössä

Meteorin 1.3 versiossa kehystä muokattiin aiemmasta siten, että React-kirjasto ladataan käyttäen npm-paketinhallintajärjestelmää. React-tiedostoissa ladataan kolmannen osapuolen moduulit käyttöön käyttäen JavaScriptin import komentoa.

Tutkimalla kuvan 4 React-luokan rakennetta, voidaan selvittää luokan muodostamista, käyttöä ja ulkopuolisten moduuleitten käyttämistä.

React-komponenttitiedosto koostuu kolmesta eri osa-alueesta, käyttöön otettavien kolmannen osapuolten moduuleista, React-luokista ja reaktiivisen datan käsittelyyn tarvittavasta metodista, jota käsitellään toisessa React-esimerkissä (kuva 4).

JavaScript ES6 uudet määrittelykomennot React-ympäristössä:

1. import komennolla tuodaan tiedoston käyttöön kolmannen osapuolen moduulit
2. export vie luodun moduulin käyttöön applikaatioalueelle

3. extends on ES6 tapa perintään, jolla komennolla peritään Reactin komponenttiosuus luotuun luokkaan.
4. default-parametrilla määritetään luokka ladattavaksi oletusarvona
5. super-komennon avulla saadaan vanhempi komponentin data käyttöön lapsikomponentissa.

```
// Tuodaan luokalle käyttöön ulkopuoliset moduulit;
import ExampleClass from '/client/ExampleClass.jsx'
import React, { Component, PropTypes } from 'react'
import ReactDom from 'react-domain'
//Luokan määrittely
export default class ParentClass extends Component{
    //Renderointi metodi, pitää sisällänsä käyttäjälle näkyvän osuuden ohjelmasta
    render(){
        return(
            <div>
                <ExampleClass message="This message is from parent component"/>
            </div>
        );
    }
}
```

KUVA 4. Esimerkki ylemmän tason React-luokasta

Kuvassa 5 käytetään rakentajametodia, jolle annetaan parametrina vanhempi luokan antama data. Näin toteutetaan komponenttien välinen kommunikaatio. Kuvassa 5 on myös käytössä reaktiivisen datan käsittelyyn tarvittava metodi. Tälle metodille voidaan antaa parametreja esimerkiksi reititinmoduulilta. CreateContainer-metodi on uusi ominaisuus Meteorin 1.3 versiossa, jossa luovuttiin Reactin käyttöönotosta Atmosphere- paketilla.

Metodia käytetään, kun halutaan käyttää reaktiivista dataa, ja kun metodin palautusosuuteen lisätään haluttuja muuttujia. Näitä ovat usein tietokantahakujen tulokset.

```
// Tuodaan luokalle käyttöön ulkopuoliset moduulit;
import ExampleClass from '/client/ExampleClass.jsx'
import React, { Component, PropTypes } from 'react'
import { Meteor } from 'meteor/meteor'
import { createContainer } from 'meteor/react-meteor-data'

//Luokan määrittely
export default class ExampleClass extends Component{

    constructor(props){
        super(props)
    }

    render(){
        let data = this.props;
        console.log(data);
        return(
            <div>
                <p>{data.text}</p>
                <p>{data.message}</p>
            </div>
        );
    }
}

// Reaktiivisen datankäsittelyyn tarvittava komponentti
export default createContainer(() =>{
    let text = "Hello World";
    return{
        text,
    };
},ExampleClass);
```

KUVA 5. Esimerkki alemman tason React-luokasta

Konsolitulostus esimerkkikuvan 5 perusteella tuottaa kuvan 6 tulostuksen. Alemman tason komponentti saa käyttöönsä olion, jossa on ylemmän tason komponentin viesti ja reaktiivisessa metodissa annettu tekstimuuttuja. Näin data liikkuu React-komponenttien välillä eri tavoilla. React suosittelee datan käsittelyä siten, että ylemmän tason komponentissa tehdään haut ja käsitellään data, joka annetaan parametreina alemman tason komponentille, jossa se renderoidaan. Näin ohjelma voidaan hajauttaa pieniin komponenttiosuuksiin ja sen ylläpitäminen ja testaus helpottuvat.

```
▼ Object {message: "This message is from parent component", text: "Hello World"}
    key: (...)
  ▶ get key: function ()
    message: "This message is from parent component"
    ref: (...)
  ▶ get ref: function ()
    text: "Hello World"
  ▶ __proto__: Object
> |
```

KUVA 6. Esimerkki alemman tason komponentille lähetettävästä datasta

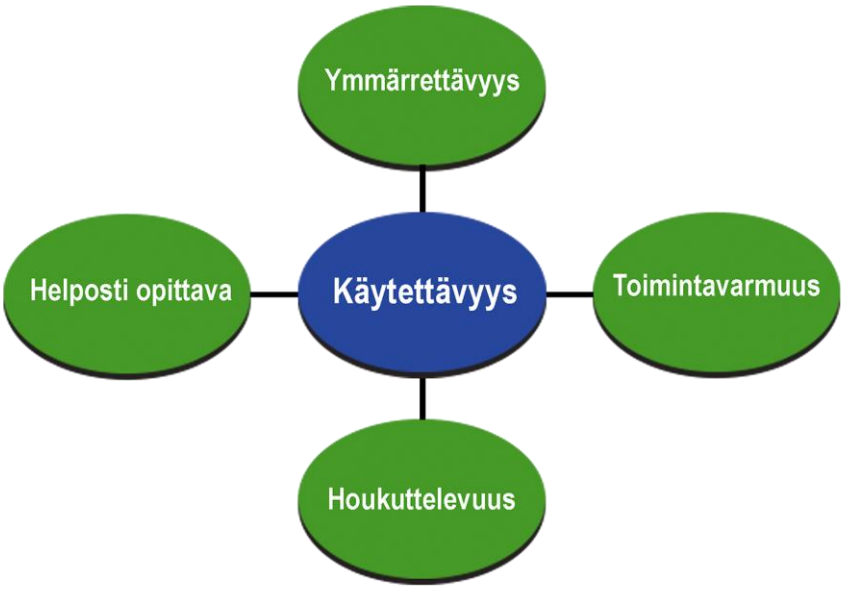
6 LOMPSA-SOVELLUS

Tässä luvussa käsitellään ja tutkitaan LOMPSA-sovelluksen palveluohjaaja roolin osuuden toteutusta ja toimintaa. Lisäksi pohditaan järjestelmän suunnitteluvaihetta ja sovelluksen laatuominaisuuksia.

6.1 Suunnittelu ja laatuattribuuttien määritys

Sovelluksesta oli tehty jo aikaisemmin eri ohjelmistotekniikoilla versio, jota käytettiin referenssinä ja jonka perusteella tehdään uutta versiota. Aikaisemman version käyttöliittymästä sai palvelusuunnitelman täytettävien kenttien tiedot, eli mitä tietoja palvelusuunnitelma sisältää.

Järjestelmän suunnitteluvaiheessa tuli määrittää toteutettavan järjestelmän laatuominaisuudet. Järjestelmän kahdeksi pääominaisuudeksi määräytyivät käytettävyys (kuva 7) ja skaalautuvuus. Käytettävyys tulee huomioida järjestelmän käyttäjän näkökulmasta siten että, palvelusuunnitelmien täyttäminen on mahdollisimman yksinkertaista, helposti opittavaa, järjestelmässä on korkea toimintavarmuus ja että järjestelmä on houkutteleva käyttäjälle.



KUVA 7. Järjestelmän käytettävyys

Toiseksi laatuattribuutiksi lukeutuu järjestelmän skaalautuvuus. Suomessa on noin 53 000 maatilaa (mtk.fi: Terveellistä ruokaa kestävin menetelmin). Järjestelmän pitää pystyä käsittelemään usean maatilan tietoja samanaikaisesti. On epätodennäköistä, että järjestelmää käyttäisivät kaikki Suomen maatilat, mutta järjestelmän pitää pystyä käsittelemään mahdollisimman tehokkaasti ja jatkuvasti muuttuvaa dataa.

6.2 MongoDB-tietokannan toteutus

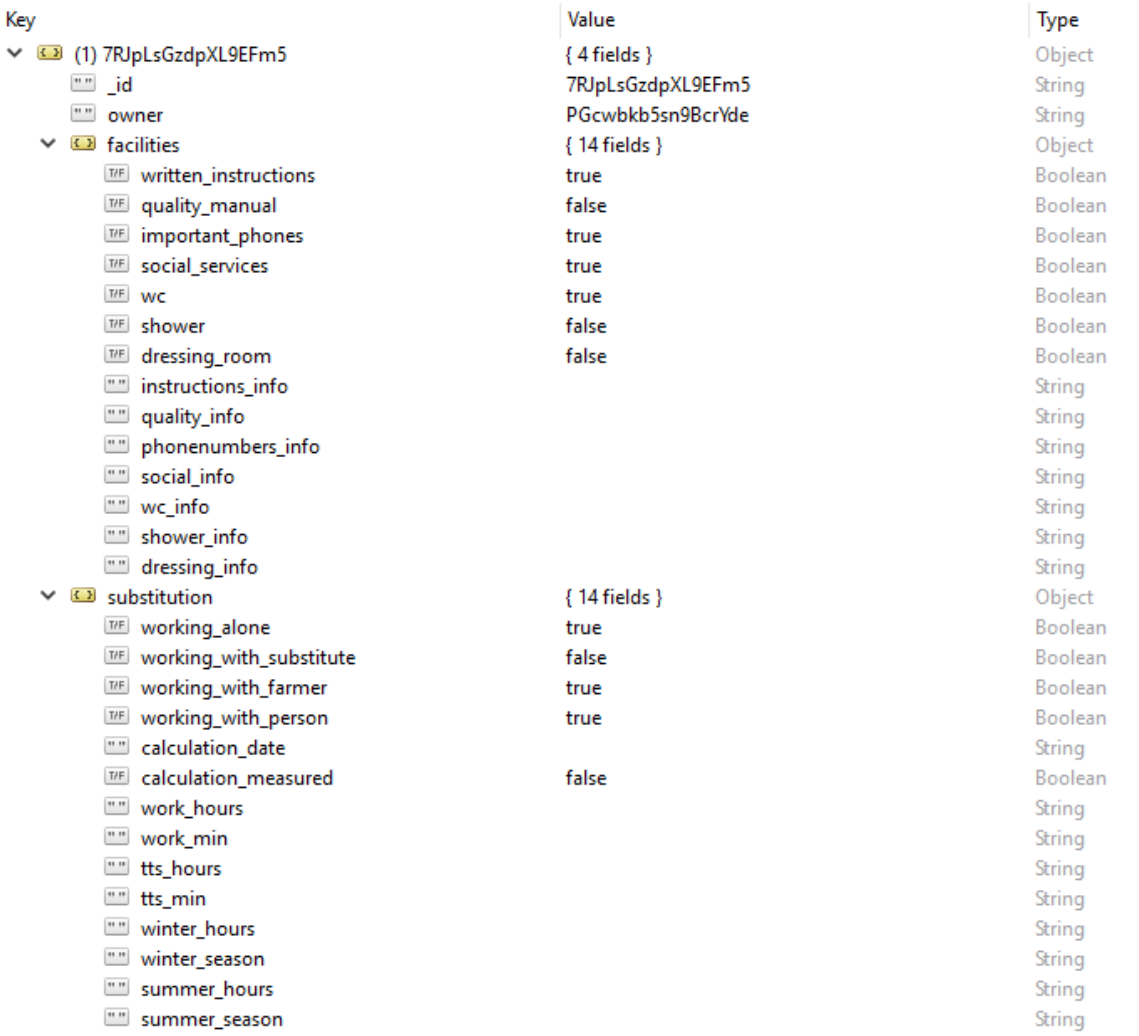
Sovellusta toteuttaessa tietokanta toteutetaan käyttöliittymän mukaisesti. Järjestelmässä käyttäjillä on oma kokoelma, joka sisältää käyttäjän tilin nimen ja salasanan. Tilin nimenä käytetään tilin omistajan sähköpostiosoitetta. Tällä menetelmällä ratkaistaan ongelma, että käyttäjänimiä ei voi olla kahta samanlaista.

Maatiloille luodaan myös oma kokoelmansa joka pitää sisällänsä maatilan nimen, osoitteen, kunnan, alueen ja luontipäivän.

Palvelusuunnitelmakokoelmat jakautuvat palvelusuunnitelman pääotsikoiden mukaisesti: yleisiin tietoihin, lypsykarjaan, lihakarjaan, sikalaan, lomitustarpeisiin ja perehdytyksiin ja yhteenvetoihin.

MongoDB ei ole relaatiomallinen tietokanta, joten jokaiselle palvelusuunnitelmaan liittyvälle kokoelmalle pitää tallentaa maatilan omistajan identifikaatioparametri.

Kuvassa 8 voidaan nähdä yhden yleisten tietojen maatilaolion rakenne. Tämä dokumentti luodaan siten, että kentät ovat tyhjinä. Tämä tarkoittaa sitä, että dokumenttiin ei lisätä dataa vaan sitä pelkästään muokataan. Dokumentilla on omistaja eli maatila, jolle nämä yleiset tiedot kuuluvat. Tämän lisäksi dokumentti sisältää maatilan olosuhde-, lomittaja- ja työtiedot.



Key	Value	Type
(1) 7RJpLsGzdpXL9EFm5	{ 4 fields }	Object
_id	7RJpLsGzdpXL9EFm5	String
owner	PGcwbkb5sn9BcrYde	String
facilities	{ 14 fields }	Object
written_instructions	true	Boolean
quality_manual	false	Boolean
important_phones	true	Boolean
social_services	true	Boolean
wc	true	Boolean
shower	false	Boolean
dressing_room	false	Boolean
instructions_info		String
quality_info		String
phonenumbers_info		String
social_info		String
wc_info		String
shower_info		String
dressing_info		String
substitution	{ 14 fields }	Object
working_alone	true	Boolean
working_with_substitute	false	Boolean
working_with_farmer	true	Boolean
working_with_person	true	Boolean
calculation_date		String
calculation_measured	false	Boolean
work_hours		String
work_min		String
tts_hours		String
tts_min		String
winter_hours		String
winter_season		String
summer_hours		String
summer_season		String

KUVA 8. Esimerkki yleisten tietojen MongoDB-dokumentista

6.3 Kolmannen osapuolen moduulit

LOMPSA-sovelluksessa käytetään ulkoisia moduuleita helpottamaan ja nopeuttamaan sovelluksen kehittämistä. Moduulit jakautuvat kahdelle eri paketinhallintajärjestelmälle, Atmosphere:lle ja npm:lle. Tässä osiossa selvitetään Meteor- ja React-pakettien lisäksi käytettäviä moduuleita ja niiden käyttötarkoitusta sovellusalueella.

Atmosphere-paketit:

Accounts-password lisää sovellukseen käyttäjätilijärjestelmän, jonka avulla määritetään kirjautuneet käyttäjät ja oikeudet näkymiin.

Alanning-roles-paketti on laajennus accounts-password pakettiin, ja sen avulla lisätään rekisteröityneille käyttäjille roolit. Roolien avulla mahdollistetaan käyttäjälle eri oikeuksia näkymiin.

Reywood bootstrap3-sass-paketti tuo sovellukseen Bootstrap3 ja Sass tuen.

Fourseven scss-paketti on scss-tiedostotyyppien kääntämiseen kehitetty moduuli. Moduulin avulla voidaan kääntää projektissa käytettyjä scss-tyylitiedostoja Meteor-projektin käyttöön.

Kadira FlowRouter on URL (Uniform Resource Locator) reititin, jonka avulla voidaan vaihtaa sovelluksen näkymiä. FlowRouter moduulin avulla voidaan antaa parametreja URL-merkkijonossa.

npm-paketit:

Griddle on React JavaScript-kirjaston käyttämiseen suunniteltu taulukkokomponentti. Komponentin avulla voidaan näyttää tietokannan dataa selkeässä formaatissa, tehdä hakuja tietokannan hakutulokseen ja sivuttaa hakutulokset.

react-mounter-paketti on sivun näkymän pohjan luontiin kehitetty moduuli. Moduulin avulla voidaan jakaa näkymä eri osiin, kuten toteutetussa sovelluksessa otsikkoon, sisällykseen ja sivupaneeliin.

react-modal-pakettia käytetään projektissa tietokannan tietojen poiston varmistusikkunoiden luomisessa. Moduulista luodaan oma komponentti, jolle annetaan parametreina poistotoimintoja ja tekstejä.

react-notification-system on paketti, jonka avulla voidaan ilmoittaa käyttäjälle tietokantaan tehtyjen päivitysten onnistumisista ja epäonnistumisista.

6.4 Reititys ja sivun asemointi

Sovelluksessa on useita eri käyttöliittymänäkymiä, joiden kautta sovellusta käytetään. Jotta eri näkymien hallitseminen helpottuisi, käytettiin Meteorin paketinhallintajärjestelmästä ladattavaa reititinpakettia FlowRouter. Flowrouter on asiakaspuolelle kehitetty reititin, jonka avulla voidaan vaihtaa näkymiä ja antaa komponenteille parametreja url-merkkijonosta. Tämän lisäksi käytetään react-mounter -moduulia, jonka avulla voidaan luoda käyttäjälle näkymä monesta eri komponentista.

Sovelluksen jokainen näkymä rakennetaan samalla React-pohjakomponentilla kuvan 9 mukaisesti. Komponentille annetaan parametrina ylätunniste-, sivupaneeli- ja sisältökomponentit parametreina.

```
// Luodaan uusi reitti ohjelmaan
// Reitti aktivioituu kun sivuston juuri URL-merkkijonoon lisätään /serviceplan/ ja merkki
FlowRouter.route('/serviceplan/:_id',{

    // Kun osoitteeseen ollaan siirrytty toteutetaan action metodi
    // params muuttuja pitää sisällänsä merkkijonosta haetun datan
    action(params){
        // Renderoidaan käyttäjälle näkymä ja annetaan komponenteille parametrina URL-data
        mount(MainLayout,{
            header: (<Header/>),
            sidepanel: (<ServicePlanSidepanel params={params}/>),
            content: (<ServicePlanHome params={params}/>)
        });
    }
});
```

KUVA 9. FlowRouter esimerkki

Kuvassa 10 esitetään näkymäkomponentin toiminnallisuutta. Komponentille annetaan reitittimeltä parametreina komponentteja. Näin pystytään rakentamaan käyttäjälle näkymä eri komponenttikokonaisuuksista. Tämä osaltaan helpottaa järjestelmän ylläpitämistä siten, että kooditiedostot pysyvät kompakteina.

```
import React from 'react'
/*
*   Ohjelman näkymäpohja josta kaikki ohjelman näkymät rakennetaan
*/
export const MainLayout = ({header,sidepanel,content}) => {
    <div className="container-fluid layout">
        {header}
        {sidepanel}
        {content}
    </div>
}
```

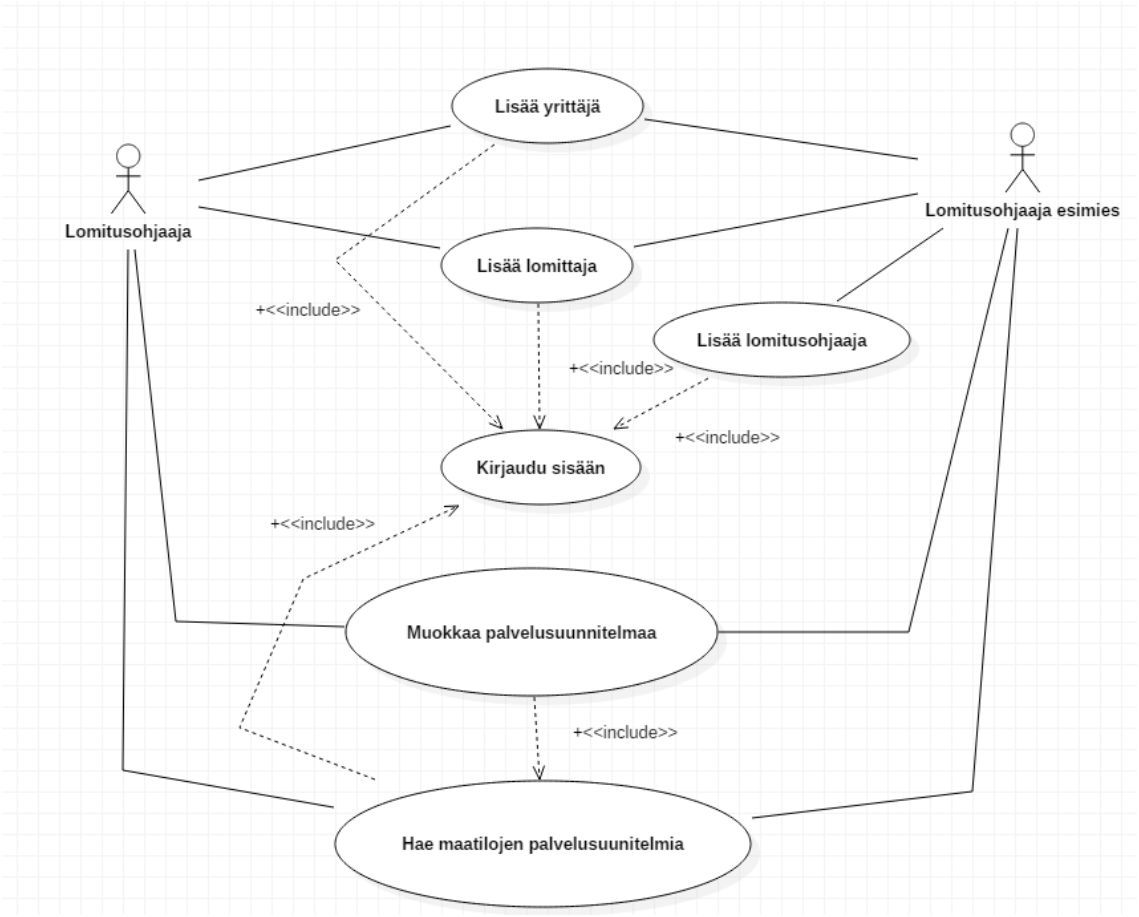
KUVA 10. Sivun ulkoasun määrittely komponentti

6.5 Käyttäjätilit ja roolit

LOMPSA-järjestelmään ei voida rekisteröityä tavanomaisin keinoin vaan lomitusohjaajat ja järjestelmän ylläpitäjät luovat tilejä käyttäjille.

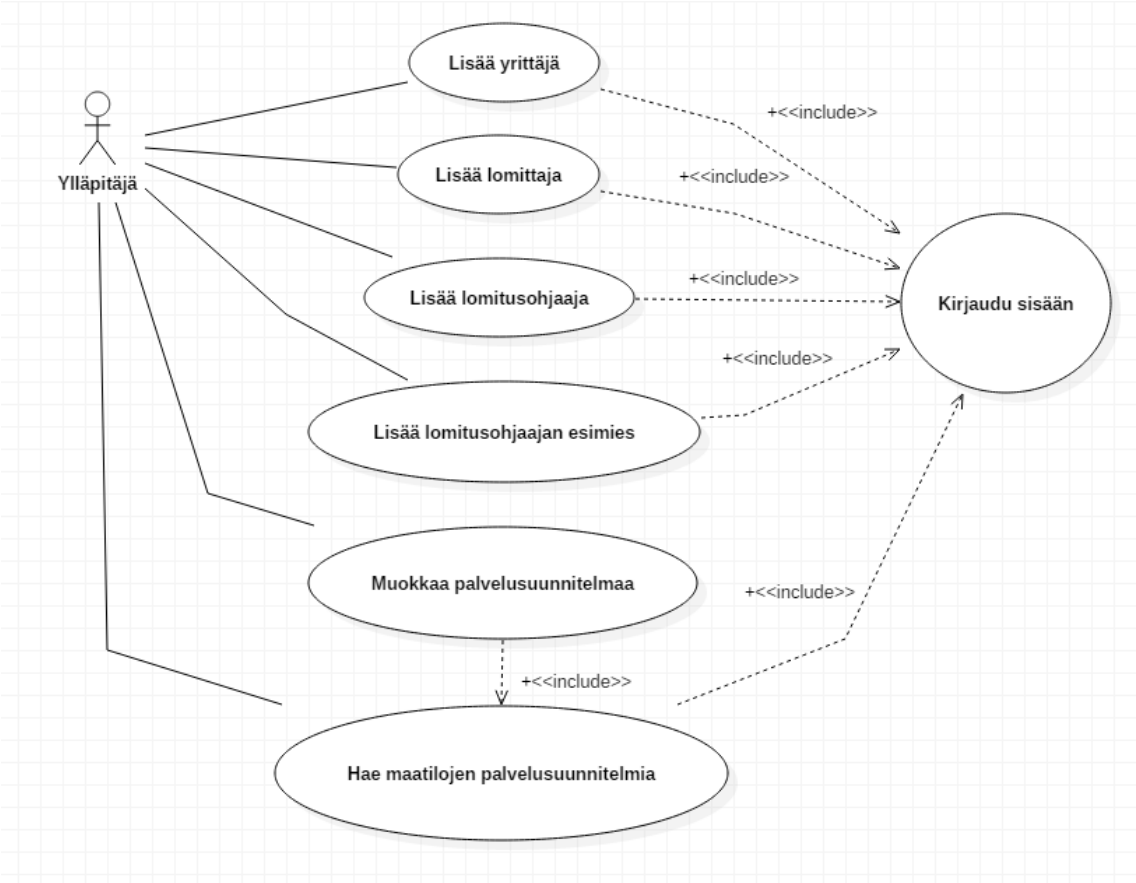
Järjestelmän lomitusohjaajat jakautuvat kahdelle käyttöoikeusroolille, lomitusohjaajat ja lomitusohjaajien esimiehet. Lomitusohjaajat voivat lisätä järjestelmään yrittäjiä ja lomittajia. Lomitusohjaajien esimiehillä on oikeus lisätä lomittajien ja yrittäjien lisäksi lomitusohjaajia. Järjestelmässä on ylläpitäjä, joka voi lisätä yrittäjien ja lomittajien lisäksi lomitusohjaajia ja niiden esimiehiä.

Käyttötapauskaavio kuva 11 kuvaa lomitusohjaajien perus- ja palvelusuunnitelmaan liittyviä toimintoja. Kummatkin käyttäjäroolit voivat tehdä järjestelmän perustoimintoja ja lisätä järjestelmään uusia käyttäjiä, mutta vain lomitusohjaajien esimiehet voivat lisätä uusia lomitusohjaajia.



KUVA 11. Käyttötapauskaavio Lomitusohjaajien rooli

Ylläpitäjän rooliin kuvassa 12 kuuluu samat ominaisuudet kuin lomitusohjaajien. Lisäksi ylläpitäjät voivat lisätä järjestelmään uusia lomitusohjaajien esimiehiä.



KUVA 12. Käyttötapauskaavio Ylläpitäjän rooli

Kuvan 13 metodille annetaan parametrina kirjautunut käyttäjä, jonka avulla navigaatiopaneeliin palautetaan käyttäjän roolin mukaiset linkit roolikohtaisiin toimintoihin. Tämän lisäksi komponenteille annetaan parametreina kielikäännökset.

```
// Metodi jonka avulla haetaan navigaatiopaneeliin käyttäjänroolin mukaiset linkit
// Annetaan parametrina komponenteille kielikäännökset linkki teksteihin
getUserRoleLinks(currentUser){
    let translate = this.props.translate;
    if(Roles.userIsInRole(currentUser, 'administrator'))
        return <AdminLinks translate={translate}/>

    if(Roles.userIsInRole(currentUser, 'administrator'))
        return <HeadServiceOfficerLinks translate={translate}/>

    if(Roles.userIsInRole(currentUser, 'service_officer'))
        return <ServiceOfficerLinks translate={translate}/>
}
```

KUVA 13. Lomitusohjaajien roolien määrittely

Kuvan 14 komponentissa renderoidaan käyttäjän roolin mukaiset linkit. Järjestelmän toteutuksen alkuvaiheessa lomitusohjaajien ja järjestelmän ylläpitäjien roolilinkit ovat samat, mutta järjestelmän kasvaessa ylläpitäjille ja ohjaajille tulee erilaisia toimintoja.

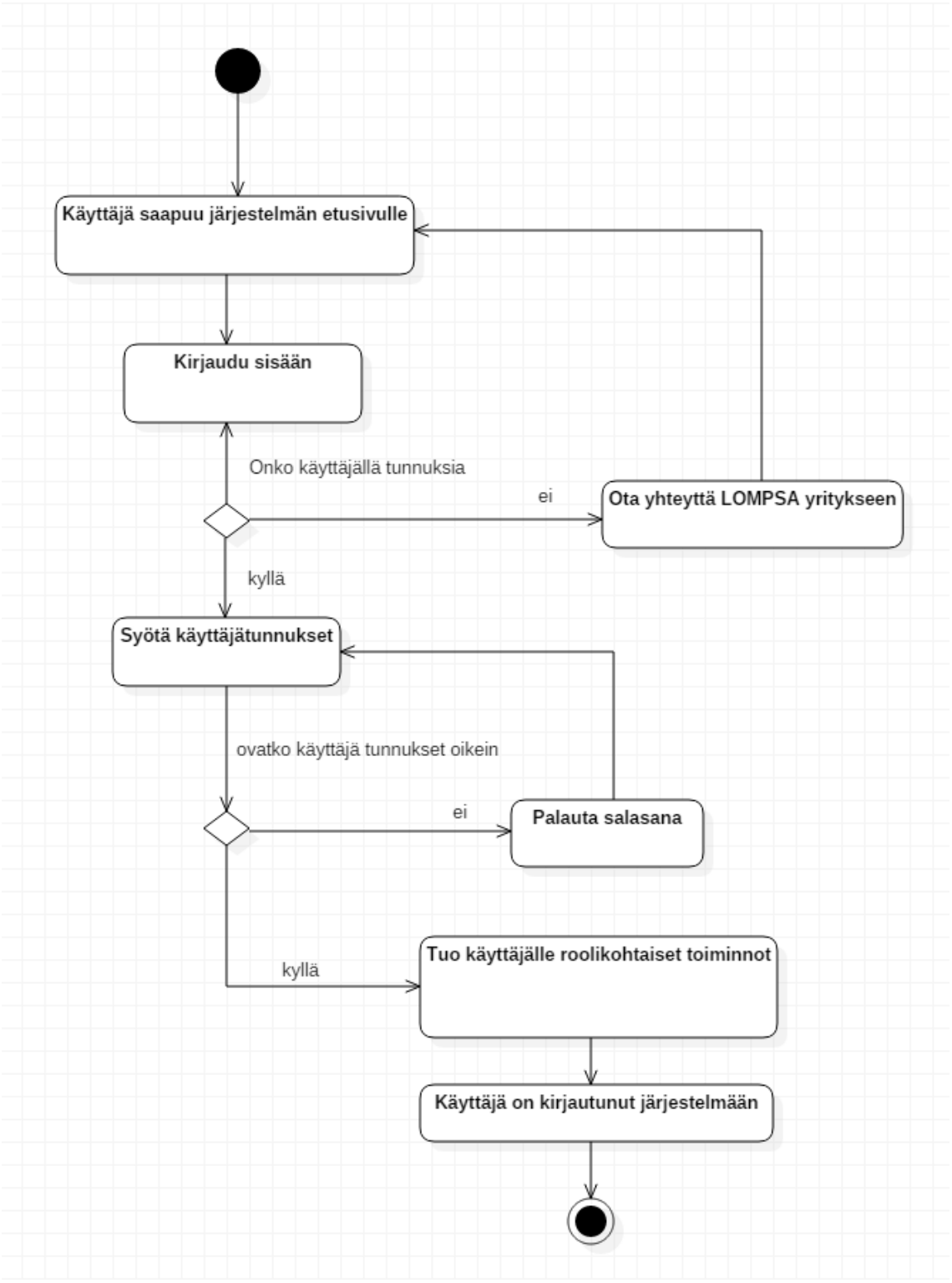
```
// Luodaan linkki komponentti
export default class HeadServiceOfficerLinks extends Component {

    constructor(props){
        super(props);
    }

    render(){
        // Talletetaan muuttujaan kielikäännös olio vanhempi komponentilta
        let translate = this.props.translate;
        return(
            <div>
                <ul className="nav navbar-nav">
                    <li><a href="/newuser"><strong>{translate.newUser}</strong></a></li>
                    <li><a href="/newfarm"><strong>{translate.newFarm}</strong></a></li>
                    <li><a href="/farms"><strong>{translate.servicePlans}</strong></a></li>
                    <li><a href="/settings"><strong>{translate.settings}</strong></a></li>
                </ul>
            </div>
        );
    }
}
```

KUVA 14. Esimerkki komponentti roolien linkeistä

LOMPSA-järjestelmään ei rekisteröidytä tavanomaisin keinoin, vaan otetaan yhteyttä järjestelmän palveluohjaajiin, jotka luovat tilejä yrittäjille ja lomitusohjaajille. Kuvassa 15 on aktiviteettikaavio järjestelmään kirjautumisen prosessista.

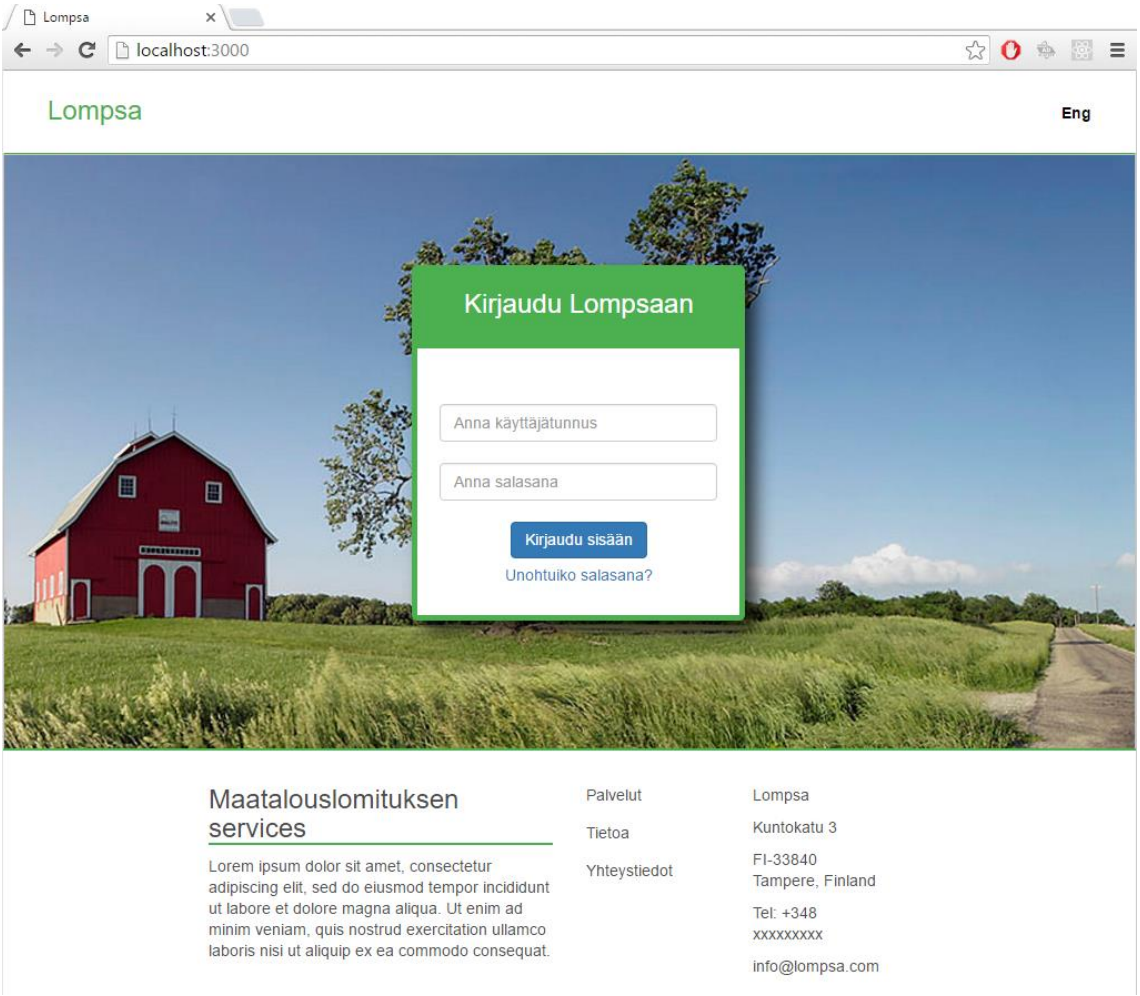


KUVA 15. Aktiviteettikaavio kirjautumisesta

6.6 Päänäkymä

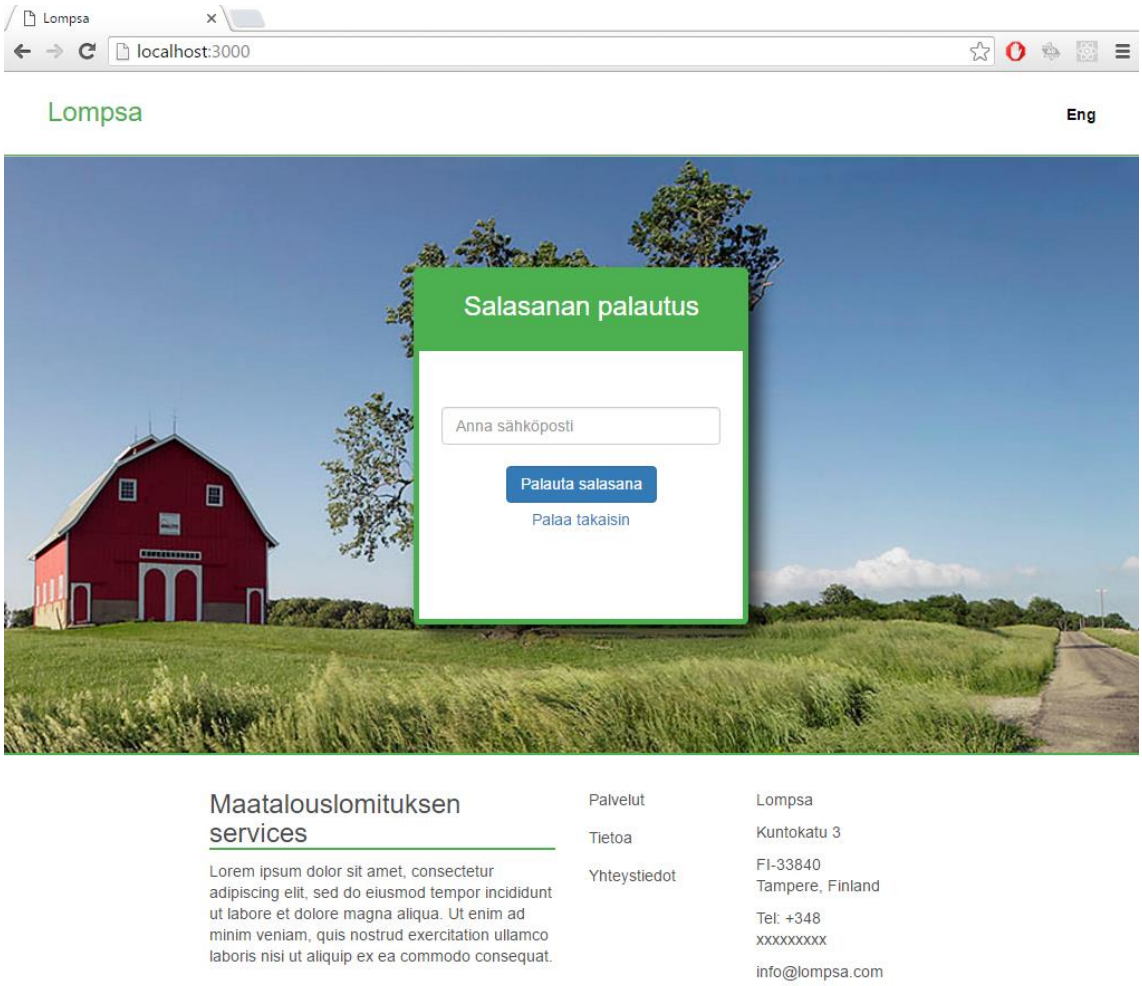
Käyttäjän tullessa sovelluksen aloitussivulle, renderoidaan sovelluksen kotinäkymä. Tässä näkymässä käyttäjälle näytetään LOMPSA-järjestelmän palvelu, tietoa yrityksestä ja yhteys tietolinkit, joista käyttäjä pääsee tutustumaan yrityksen tietoihin. Tämän lisäksi käyttäjälle näytetään järjestelmän kirjautumisen ikkuna ja salasanan palauttamisen linkki. Käyttäjä voi vaihtaa järjestelmän kieliasetuksia painamalla kielen vaihtoasetuspainiketta suomen- ja englanninkielen välillä.

Kuvassa 16 on sisäänkirjautumisen ikkuna, jossa käyttäjälle tuodaan syöterivit, johon täytetään käyttäjätunnus ja salasana. Kirjautumisen aikana järjestelmä tarkistaa löytyykö tietokannasta käyttäjän nimellä ja salasanalla kyseessä olevaa henkilöä. Tarkistuksen perusteella näytetään käyttäjälle sisään kirjautuneen käyttäjän toiminnot tai sovelluksen päänäkymä.



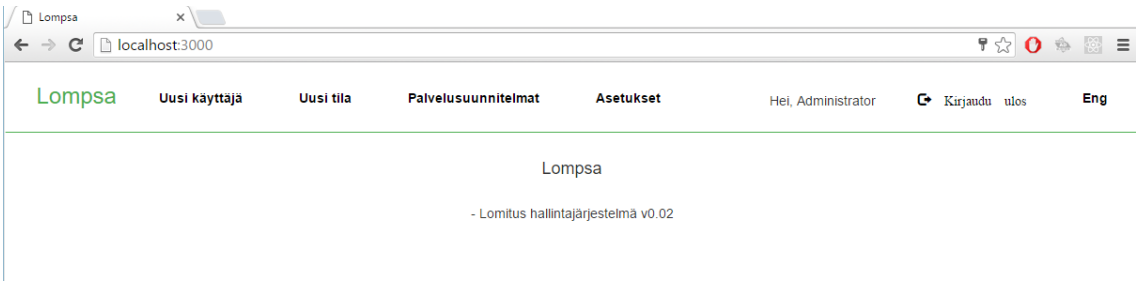
KUVA 16. Järjestelmään kirjautuminen

Kun sähköposti on syötetty kenttään (kuva 17), järjestelmä tarkistaa onko tietokannassa käyttäjää annetulla sähköpostiosoitteella. Mikäli sähköpostiosoite löytyy järjestelmästä, niin käyttäjälle lähetetään sähköpostitse linkki uuden salasan määrittämiseen. Käyttäjän avatessa linkin sähköpostistansa käyttäjälle avautuu uusi selainikkuna LOMPSA-sovelluksen sivuille, jossa käyttäjälle näytetään lomake. Sen avulla käyttäjä voi vaihtaa käyttäjätilinsä salasanan.



KUVA 17. Salasanan palautus

Kirjautuneelle käyttäjälle (kuva 18) tuodaan roolikohtaiset linkit navigaatiopaneeliin, josta pystytään roolin mukaan tekemään järjestelmän toimintoja. Kun käyttäjä on kirjautunut järjestelmään, niin käyttäjälle näytetään uloskirjautumispainike navigaatiopaneeliin. Tämän lisäksi käyttäjälle ilmoitetaan kyseisen järjestelmän versio ja tulevaisuudessa mahdollisesti järjestelmän päivitysten uutiset.



KUVA 18. Kirjautunut käyttäjä

6.7 Perustoiminnot

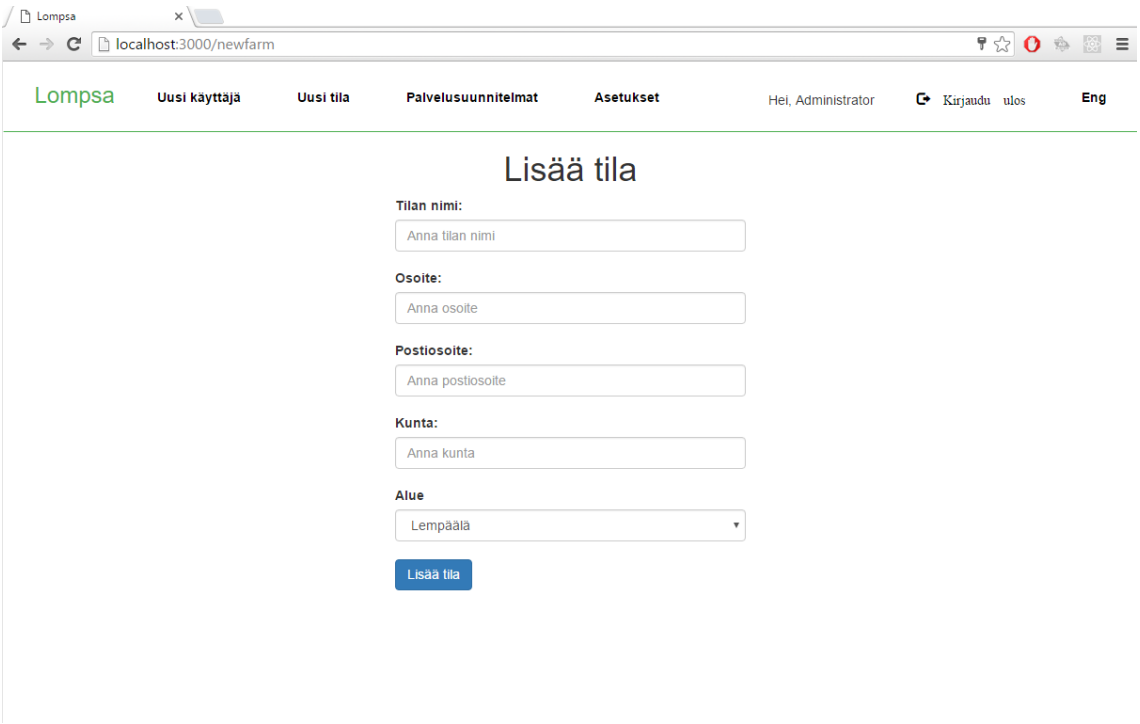
Sovelluksen lomitusohjaajien ja ylläpitäjän perustoimintoihin lukeutuu uuden käyttäjän ja uuden maatilan luominen. Tämän lisäksi käyttäjä voi tarkastella ja muokata palvelusuunnitelmia tai omia käyttäjäasetuksia.

Jokaisessa perustoiminnon näkymässä on syötevarmistukset, jotta jokainen kenttä tulee täytetyksi. Tällä osaltaan varmistetaan se, että virheellisiä lisäyksiä tietokantaan ei tule.

Päivityksen onnistuminen tai epäonnistuminen ilmoitetaan tekstinä käyttäjälle.

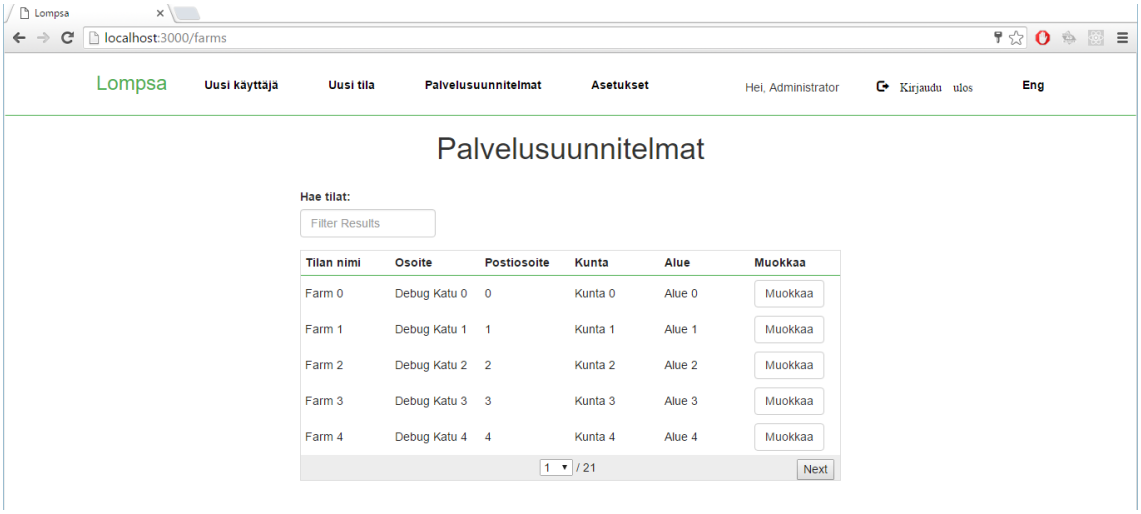
Tarkastellaan seuraavaksi kahta lomituspalvelijan perusnäkymää.

Kuvan 19 tilan lisäyskomponentti on hyvin yksinkertainen. Käyttäjälle tuodaan syötteet, joiden perusteella luodaan uusi maatila järjestelmään. Maatilan lisäystoiminnossa tarkastetaan, onko kaikki kentät täytetty, jonka perusteella käyttäjälle palautetaan viesti lisäyksen onnistumisesta tai epäonnistumisesta.



KUVA 19. Tilan lisäysnäkymä

Palvelusuunnitelma-näkymässä (kuva 20) tuodaan käyttäjälle tietokannan maatilat taulukkona, joka näytetään käyttäjälle Griddle-moduulikomponenttien avulla. Taulukosta voidaan tarkastella maatilan nimiä, osoitteita, kuntia, alueita sekä aloittaa maatilan palvelusuunnitelmien muokkaus.



KUVA 20. Palvelusuunnitelmien näkymä

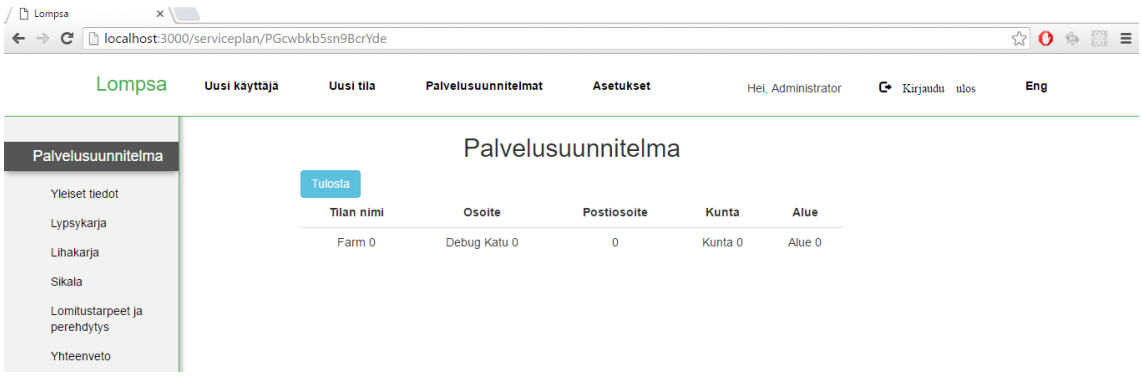
6.8 Maatalouslomituksen palvelusuunnitelma

Maatalouslomituksen palvelusuunnitelmat sisältävät maatilan lomitustarpeisiin liittyvää tietoa. Palvelusuunnitelman kokonaistietomäärä on hyvin suuri, jolloin käyttöliittymän toteutuksessa tulee ottaa huomioon sovelluksen käytettävyys. Palvelusuunnitelma jakautuu useisiin eri otsikkoalueisiin kuten: yleisiin tietoihin, lypsykarjaan, lihakarjaan, sikalaan, lomitustarpeisiin ja perehdytyksiin ja yhteenvetoihin. Nämä kaikki osa-alueet sisältävät suuren määrän toisistaan eroavaa tietoa, joka pitää saada tallennettua järjestelmään.

Jokaisesta palvelusuunnitelman pääalueesta luodaan oma MongoDB-kokoelma. Näin pystytään pitämään kehittäjän näkökulmasta tietorakenteet selvinä. Näkymissä haetaan tarkasteltavan maatilan identifikaatioparametrilla palvelusuunnitelman tietoa ja se tuodaan käyttäjälle muokattavaksi.

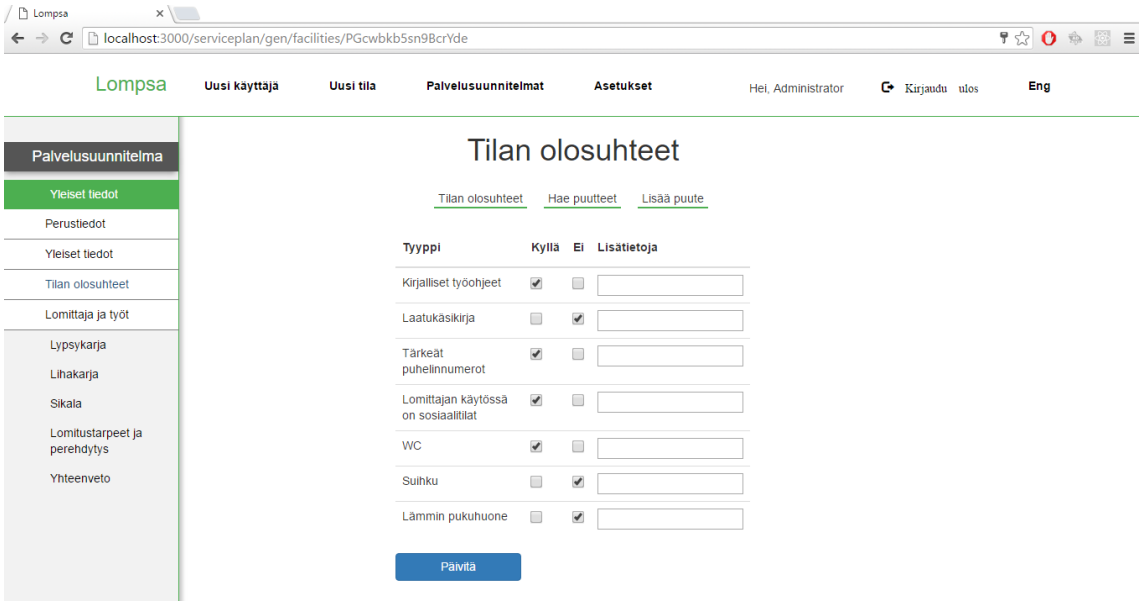
Palvelusuunnitelma sisältää useita eri näkymiä käyttäjälle. Seuraavassa osiossa tarkastellaan paria näkymää esimerkin valossa.

Palvelusuunnitelman pääsivulla (kuva 21) esitetään käyttäjälle maatilan perustiedot kuten, nimi, osoite, kunta ja alue. Tällä sivulla näytetään jatkokehityksen toiminnot palvelusuunnitelman pdf-version luomiseen paperiversion tulostusta varten. Lisäksi käyttäjälle näytetään palvelusuunnitelmien navigaatiopaneeli vasemmalle puolelle selain-näkymää.



KUVA 21. Palvelusuunnitelman pääsivu

Tilan olosuhdenäkymässä (kuva 22) käyttäjä voi muokata tilan olosuhdetietoja, lisätä tilalla olevia puutteita, joita voivat olla esimerkiksi vaaralliset paikat tai valaistuksen puutteet. Näkymästä voidaan hakea tilan puutteita ja poistaa niitä, kun puuteet on korjattu.



KUVA 22. Tilan olosuhteiden näkymä

6.9 Sähköpostipalvelut

Sovelluksessa käyttäjät eivät itse rekisteröidy järjestelmään vaan lomitusohjaajat lisäävät käyttäjiä. Uuden käyttäjätilin luomisprosessin päätteeksi uudelle käyttäjälle lähetetään käyttäjätilin tiedot sähköpostiin. Sähköpostin lähettämiseen tarvitaan kolmannen osapuolen sähköpostipalvelua. Projektissa on käytetty Meteorin omaa sähköposti moduulia. Moduulin käyttöönotto edellyttää sähköpostipalvelun tarjoajaa.

Projektin sähköpostien lähettämiseen käytetään Mailgun-palveluntarjoajaa. Palvelu tarjoaa kehittäjälle rajapinnan sähköpostiviestien lähetys- ja vastaanottopalveluihin.

Käyttäjälle saapuu sähköpostiin (kuva 23) viesti, kun käyttäjän tili on luotu järjestelmään. Sähköposti sisältää käyttäjätunnuksen ja satunnaisesti luodun salasanan. Salasana kehotetaan vaihtamaan mahdollisimman pian.

Hei, Teemu !

Oheisessa viestissä on käyttäjätunnuksesi ja salasana, on suositeltavaa että salasana tulisi vaihdettua ensimmäisellä kirjautumiskerralla.

Käyttäjätunnus	teemu.kanerva@eng.tamk.fi
Salasana	0v6Tn2iuHdRA72vD

Terveisin Lompsa väki.

KUVA 23. Sähköpostin ulkoasu

6.10 Monikielisyyden tuki

Järjestelmään haluttiin monikielisyyden tuki, joka on suomen- ja englanninkielen tuki. Kielituki toteutettiin Meteorin session-muuttujalla. Session luo globaalin olion asiakaspuolelle, johon voi tallettaa mielivaltaisesti JSON avain-arvo-pareja. Session data toimii reaktiivisesti, jolla tarkoitetaan sitä, että kun session muuttujan arvoa muutetaan, päivitetään sen muuttunut arvo käyttäjän näkymään. Session muuttuja toimii hyvin ratkaisuksi monikielisyyden tueksi.

Kielidata sijaitsee omassa tiedostossa, jonka sisällä määritellään JSON-muuttuja. Muuttuja pitää sisällänsä kaksi JSON-aladokumenttia, suomen ja englannin kieliset käännökset. Tiedosto sisältää myös metodit kielen vaihtamiseen. Käyttöliittymässä käyttäjä kutsuu kielitiedoston metodia ja sen avulla vaihtaa session-olioon kielen.

Sovelluksen kielenvaihto on tehty omalla React-komponentilla, joka tuodaan navigaatiopaneeliin erillisenä osana. Komponentissa säädetään sovelluksen kieli suomeksi tai englanniksi navigaatiopaneelissa sijaitsevalla napilla. Kielenvaihtologiikka on toteutettu siten, että siihen voidaan helposti lisätä muitakin käytettäviä kieliä.

6.11 Tietoturva

Järjestelmässä käsitellään maatilojen arkaluontoista tietoa, jolloin tietoturvallisuus on yhtenä tärkeänä osana sovellusta. Sovelluksessa estetään kirjautumattomien käyttäjien

pääsy näkymiin, joissa kirjautuminen vaaditaan. Esto tehdään sovelluksessa komponenttitasolla (kuva 24), jossa tarkistetaan ovatko käyttäjätiedot käsiteltävissä. Haettaessa käyttäjän tiedot tietokannasta näytetään latauksen aikana näytöllä latausikonia.

```
// Tuodaan luokalle käyttöön ulkopuoliset moduulit;
import React, { Component, PropTypes } from 'react';
import ReactDOM from 'react-dom';
import { Meteor } from 'meteor/meteor';
import { createContainer } from 'meteor/react-meteor-data';

// Tuodaan luokalle käyttöön virhenäkymä, lataus ja näkymäkomponentti
import NotAuthorizedUser from '/imports/ui/components/utility/NotAuthorizedUser.jsx';
import Loading from '/imports/ui/components/utility/Loading.jsx';
import ServicePlanHomeView from '/imports/ui/components/service_panel/serviceplan_home/ServicePlanHomeView.jsx';

export default class ServicePlanHome extends Component {

    getContent(){

        let params = this.props.params._id;
        return <div> {this.props.currentUser? <ServicePlanHomeView params={params}/> : <NotAuthorizedUser/>} </div>
    }

    // Renderoidaan latauskomponenttia niin kauan kunnes käyttäjädata on haettu
    // Mikäli käyttäjä data on tyhjä renderoidaan virhenäkymä muutoin päästetään jatkamaan oikeaan näkymään
    render() {
        return (
            <div>
                {this.props.authInProcess? <Loading/> : this.getContent()}
            </div>
        );
    }
}
// Haetaan reaktiivisesti käyttäjätieto dataa
export default createContainer(() => {
    return {
        currentUser: Meteor.user(),
        authInProcess: Meteor.loggingIn(),
    };
}, ServicePlanHome);
```

KUVA 24. Komponenttitason näkymän esto

6.12 Testaus

Järjestelmän toteutuksen aikana sovelluksen toimintoja testattiin jatkuvasti tekemällä eri tyyppisiä syötteitä järjestelmään. Tällöin voitiin tarkastella mahdollisia virhetilanteita. Tämän lisäksi sovelluksen tehokkuutta testattiin. Suomessa on noin 57300 maatilaa, joten tällä määrällä oli järkevää testata järjestelmän suorituskykyä. Suorituskykytestauksessa huomattiin, kuinka maatilojen datan reaktiivisuus käyttäytyy isojen datamäärien kanssa. Järjestelmän datan käsittely alkaa hidastua, kun maatilojen datamäärä saavuttaa 20000 rajan. Parantaakseen järjestelmän nopeutta tulisi datan käsittely reaktiivisesti ottaa pois-

S

7 JATKOKEHITYS

LOMPSA-järjestelmän jatkokehitys jakautuu kolmeen eri pääalueeseen: yrittäjä-, lomittaja- ja palveluohjaajaosion jatkokehitykseen. Palveluohjaajien jatkokehityksessä järjestelmään tuodaan mahdollisuus tulostaa sähköisiä palvelusuunnitelmia paperille. Tämän lisäksi palveluohjaajien perustoimintoja kehitetään ja parannetaan entisestään.

LOMPSA-Järjestelmän tulee käsitellä paljon dataa samanaikaisesti. Käyttäjäkannan kasvaessa skaalautuvuus tulee huomioida erityisesti siten, että tietokanta hakuja tehdään vain silloin kun dataa halutaan tarkastella ja käsitellä.

Tietoturvallisuuden osuus on suuri osa järjestelmää. Palvelusuunnitelmat pitävät sisällään arkaluontoistatietoa maatiloista. Sivuston turvallisuuden takaamiseksi tulisi käyttöönottaa SSL tietoverkkosalausprotokolla.

Yrittäjille luodaan mobiiliversio, josta yrittäjät voivat mobiililaitteilla käyttää sähköistä palvelusuunnitelmaa. Yrittäjille luodaan myös toiminnot tehdä sähköisiä ohjeita lomittajille liittyen työntekoon, työtapoihin ja työturvallisuuteen.

Lomittajille luodaan oma mobiilikäyttöliittymä, jonka avulla lomittajat voivat tarkastella etukäteen lomitustyöpaikkansa tietoja kuten osoitetta, ajo-ohjeita ja muita työpaikkaan liittyviä asioita. Tämän lisäksi järjestelmän avulla lomittajat voivat käydä läpi video- tai tekstiohjeita heidän työohjeistuksista, työtiloista ja työtavoista. Lomittajat voivat myös tarpeen niin vaatiessa tarkastella työohjeita ja työmenetelmiä mobiililaitteestaan maatilalla työpisteessään.

8 POHDINTA JA YHTEENVETO

LOMPSA-järjestelmä on kunnianhimoinen projekti, jonka tarkoitus on ratkaista maatilojen lomitustyöhön liittyvät ongelmat. Sovelluksen alle jakautuu kaikki maatilan lomitukseen liittyvät tekijät ja osapuolet.

Projektina LOMPSA-sovellus on massiivinen, sen sisältöön liittyy eri osapuolien käyttäjät, heidän toiminnot ja koko sovelluksen käytettävyys. Sovelluksen kehittämisen aikana on tullut ilmi omana kokemuksena käytettävyyden huomioon ottaminen. Se on näkynyt varsinkin testatessa sovellusta siten, että millä tavalla on itse kokenut dokumenttien täyttämisen yksinkertaisuuden tai näkymien välisen navigoimisen. Testaaminen on osoittautunut yhdeksi tärkeimmistä osa-alueista sovelluksen kehittämisessä.

Uudet kehittyneet ohjelmointimenetelmät kuten Meteor ja React ovat tuoneet sovelluksen kehitykseen nopeutta. Tämä osaltaan tarkoittaa sitä, että kehittäjän ei tarvitse tehdä kaikkea itse alusta.

Kolmannen osapuolen pakettien avulla on sovelluksen kehitysnopeus pysynyt tasaisena. Ongelmaksi pakettien käytössä voi kuitenkin ilmetä käyttöönotto-ongelmia tai muita virhetilanteita, joita paketin käyttöönoton alkuvaiheessa ei havaitse. Pakettien käyttämistä kannattaakin tutkia kriittisesti jatkokehityksen osalta.

Omana kokemuksena LOMPSA-sovelluksen kehittäminen on ollut mielekästä. Koska sovellusta lähdettiin uudelleen toteuttamaan alusta, pääsin ensikädessä tutustumaan ja opettelemaan koko ohjelmistoalueen kehitysalueet, joita tämän projektin osalta on ollut asiakaspuoli, palvelinpuoli ja tietokanta.

Tämän projektin osalta oma ymmärrys kehittyneiden verkkosovellusten kehittämisestä on kasvanut ja tuonut itseluottamusta tulevaisuuden työmahdollisuuksia varten.

LÄHTEET

Git. 2016. Luettu 18.5.2016. https://git-scm.com/

Json.org. 2015. Introducing JSON. Luettu 12.5.2016. http://www.json.org

MongoDB. 2016. Introduction to MongoDB. Luettu 8.5.2016. https://docs.mongodb.com/manual/introduction/

Meteor.com. 2016. Application Structure. Luettu 10.5.2016. http://guide.meteor.com/structure.html

Meteor.com. 2016. Atmosphere vs. npm. Luettu 31.5.2016. http://guide.meteor.com/atmosphere-vs-npm.html

Meteor.com. 2016. Blaze. Luettu 30.5.2016. http://guide.meteor.com/blaze.html

Meteor.com. 2016. Introduction. Luettu 10.5.2016. http://guide.meteor.com/

Meteor.com 2016. Publish and subcribe. Luettu 12.5.2016. https://docs.meteor.com/api/pubsub.html

mtk.fi. 2016. Turvallista ruokaa kestävin menetelmin. Luettu 28.5.2016. https://www.mtk.fi/maatalous/maatilat_suomessa/fi_FI/maatilat_suomessa/

npm. 2016. What is npm?. Luettu 4.5.2016. https://docs.npmjs.com/getting-started/what-is-npm

React, 2016. Why React?, Luettu 11.5.2016. https://facebook.github.io/react/docs/why-react.html

Robomongo. 2016. Luettu 9.5.2016. https://robomongo.org/

Sass. 2016. Sass Basics. Luettu 22.5.2016. http://sass-lang.com/guide

Sublime Text. 2016. Sublime Text. Luettu 2.5.2016. https://www.sublimetext.com/