SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

LAITELIITTYMIEN HALLINTAPORTAALI

TEKIJÄ/T: Ville Nuutinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Ville Nuutinen	
Työn nimi Laiteliittymien hallintaportaali	
Päiväys 15.04.2017	Sivumäärä/Liitteet 27 / 0
Ohjaaja(t) lehtori Keijo Kuosmanen, lehtori Jussi Koistinen	
Toimeksiantaja/Yhteistyökumppani(t) Mastercom Oy	

Tiivistelmä

Opinnäytetyön aiheena oli kehittää laiteliittymien relaatiotietokannan hallintaportaali Mastercom Oy:lle. Tietokanta koostuu liittymän tiedoista ja viittauksesta liittymään liitettyyn IoT laitteeseen. Portaalin käyttöliittymän avulla voi hakea, tarkastella, muokata ja vaihtaa laitteisiin liitettyjen liittymien ominaisuuksia.

Työn tavoitteena oli kehittää toimiva, selkeä ja helppokäyttöinen sovellus, jonka avulla pystytään hallinnoimaan usean eri operaattorin liittymiä saman portaalin kautta. Ohjelmointiympäristönä käytössä oli VIM tekstieditori ja LAMP palvelinkokonaisuus. Sovellus rakennettiin MVC - mallin muokaisena Web pohjaisena ohjelmistona käyttäen hyödyksi Doctrine - sekä Vue.js - ohjelmointikehyksiä.

Vaikka kaikkia haluttuja rajapintoja operaattorien palveluihin ei saatu toteutettua, oli tämän insinöörityön lopputulos onnistunut, ja tuotoksena syntyi toimiva sovellus, joka on yrityksellä käytössä.

Avainsanat
PHP, Doctrine, Vue.js, relaatiotietokannat

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Ville Nuutinen			
Title of Thesis Web Portal for Managing Data Subscription Database			
Date	15 April 2017	Pages/Appendices	27 / 0
Supervisor(s) Mr Keijo Kuosmanen, Lecturer, Mr Jussi Koistinen, Lecturer			
Client Organisation /Partners Mastercom Oy			

Abstract

The purpose of this thesis was to develop a user friendly web portal for managing and controlling data subscription database. The database holds information about the current state of the subscription and reference to the corresponding Iot device to which the subscriber identity module is installed. The main purpose of the portal was to be able to manage a large amount of subscriptions and change the state of subscription of multiple operators. The work was commisioned by Mastercom Oy.

The development environment consisted of the Vim text editor and LAMP software stack.
The software was built using the MVC architectural pattern and using Doctrine and Vue.js frameworks.

Although all the APIs for the operators could not be implemented the thesis was successful and the final product is used by Mastercom Oy.

Keywords
MVC, PHP, Javascript, Doctrine, Vue.js

SISÄLTÖ

1 JOHDANTO

Työn toimeksiantajana on Mastercom Oy. Mastercom vuonna 2003 perustettu pohjoissavolainen yritys, joka tuottaa automatisoituja tiedonkeräys- ja jalostuspalveluita. Työn idea syntyi tarpeesta hallita ajoneuvoihin liitettyjen telematiikkalaitteiden dataliittymiä. Yrityksellä oli käytössä useiden operaattorien liittymiä, joten jos liittymiä piti muuttaa, täytyi muutos toteuttaa jokaisen operaattorin omassa portaalissa. Tästä syystä yrityksessä tuli tarve luoda liittymien hallintaportaali, joka keskittäisi kaikki liittymät yhden portaalin alle ja jossa tehdyt tilan muutokset siirtyisivät automaattisesti operaattorin palveluun. Työn toinen tarkoitus on varmistaa, että kaikki liittymät, jotka on tarkoitus olla suljettuja, olisivat myös suljettuina.

Portaalin avulla yritys pystyisi näkemään yksityiskohtaisesti SIM-kortin tietoja sekä hallitsemaan liittymien tilaa. Portaalista tehdyt muutokset piti pystyä tallentamaan muutoslokiin, jotta tehtyjä muutoksia pysyttäisiin seuramaan tehokkaasti. Työhön kuulu myös kehittää jo olemassa olevaan taulukkomuotoisen tiedon tuonti työkaluun uusi ominaisuus, jolla saataisiin tuotua operaattorien palveluista ladatut tiedot tähän uuteen portaaliin ja samalla tarkistettua, että tuotujen liittymätietojen tila vastaisi yrityksen tietokannassa olevia tietoja.

Työssä käsitellään työssä käytettyjä ominaisuuksia Doctrinen ORM–kirjastosta sekä käsitellään Vue.js:n ominaisuuksia. Työn toteutuksessa käsitellään käyttöliittymän ominaisuuksia ja liittymän tilan muutosta operaattorien palveluun.

2 KÄYTETYT TEKNIIKAT, LYHENTEET SEKÄ MÄÄRITELMÄT

Käytetyt tekniikat määräytyivät hyvin pitkälle sen mukaan, miten yrityksen tuoteperheen tuotteet oli rakennettu. Yrityksessä oli käytössä LAMP - palvelinkokonaisuus, jolla WWW-palvelimen ohjelmistoympäristö oli muodostettu. Yrityksessä oli myös käytössä Doctrine - kirjastokokoelma, josta käytettiin ORM-kirjastoa. Vue.js:n valitsin käyttöliittymän rakentamiseen, suurimmaksi osaksi oman mieltymykseni vuoksi ja osaksi siksi, että se oli myös otettu yrityksessä käyttöön. Editorina oli palvelimella toimiva VIM-tekstieditori.

2.1 Doctrine

Doctrine - projekti on kokoelma PHP-kirjastoja, joiden tärkeimpinä projekteina ovat ORM - Object Relational Mapping sekä DBAL - Database Abstraction Layer.

2.2 Vue.js

Vue.js on avoimen lähdekoodin Javascript-ohjelmointikehys käyttöliittymien rakentamiseen.

2.3 SOAP

SOAP (Simple Object Access Protocol) on tietoliikenneprotokolla jonka pääasiallisena tehtävänä on mahdollistaa proseduurien etäkutsu (RPC). Se on toimintaperiaatteeltaan samantapainen kuin muut RPC-protokollat, mutta sen erityispiirteinä on pohjautuminen XML-kieleen ja toimiminen useiden eri protokollien yli. Sitä käytetään pääasiassa HTTP-protokollan yli. (Wikipedia, SOAP)

2.4 VIM

VIM on avoimen lähdekoodin tekstieditori, joka on laajasti muokattava ja joka mahdollistaa kaikenlaisen teksin luomisen ja muokkaamisen tehokkaasti.

2.5 DOM

Document Object Model on tapa kuvata rakenteisen dokumentin, kuten HTML:n, XML:n rakenne puuna, jonka eri olioita voi hakea, tutkia ja manipuloida. (Wipedia, DOM)

2.6 SIM

Subscriber Identity Model on kortti, joka pitää sisällään IMSI numeron, jotta käytetään päätelaitteen käyttäjän tunnistamisessa. (Wipedia, SIM)

2.7 HTML

Hypertext Markup Language on ohjelmointikieli, joka on erityisesti tarkoitettu verkkosivujen luontiin.

2.8 Apache

Apache on avoimen lähdekoodin ohjelmisto, joka on maailman käytetyin HTTP-palvelinohjelma. (Wikipedia, Apache)

2.9 Ajax

Asynchronous JavaScript and XML on käsite, joka on muodostettu kuvamaan usean eri teknologian käyttöä yhdessä, joilla saadaan palvelimelta haettua nopeasti tietoja päivittämättä selaimen sivua. (MSDN, Ajax)

2.10 PHP

Hypertext PreProcessor on ohjelmointikieli, jota käytetään erityisesti Web-palvelinympäristöissä dynaamisten web-sivuen luonnissa. (Wikipedia, PHP)

2.11 SQL

Structured Query Language on ohjelmointikieli suunniteltu relaatiotietokantojen hallintaan. (Wikipedia, SQL)

2.12 MySQL

MySQL on relaatiotietokantaohjelmisto, joka on hyvin suosittu web-palveluiden tietokantana. MySQL tarjoaa nopean usean säikeen, usean käyttäjän SQL tietokantapalvelimen. (Oracle Corporation, MySQL)

2.13 LAMP

LAMP on akronyymi avoimen lähdekoodin ohjelmistoista, joita käytetään muodostamaan WWW-palvelinkokonaisuus. LAMP muodostuu sanoista Linux, Apache, MySQL, sekä PHP.

3 DOCTRINE

Doctrine on kokoelma valittuja PHP-kirjastoja, jotka tarjoavat samankaltaisia toiminnallisuuksia. Sen avainprojektina on ORM–työkalu (Object Relational Mapper), eli oli-relaatiomallintaja, joka luo virtuaalisen oliotietokannan olemassa olevasta tietokannasta. Tämän avulla oliohjelmointi paragdimaa tukevat kielet voivat käyttää tietokannan tauluja luokkina ja tietokannan sarakkeita luokan ominaisuuksina. Toisena avainprojektina on DBAL (Database Abstraction Layer) eli tietokannan abstraktointi rajapinta, joka yhdistää tietokannan ohjelmakoodiin. DBAL tarjoaa johdonmukaisen rajapinnan piilottaa tietokantaan yhdistämiseen liittyvät toiminnallisuudet. (ORM, DBAL)

Doctrine sisältää oman oliokyselykielen, DQL:n (Doctrine Query Language), joka tarjoaa monipuoliset kyselymahdollisuudet oliomalliin. Syntaksiltaan kyselykieli on hyvin lähellä SQL-kieltä, mutta kyselyt eivät kohdistu tauluihin eivätkä sarakkeisiin vaan olioon ja sen ominaisuuksiin. (DQL)

3.1 ORM

ORM eli olio-relaatiomallintaja on rakennettu toimimaan PHP-version 5.4 tai uudemman kanssa. ORM on suunniteltu Data Mapper suunnittelumallin pohjalta, ja sen tarkoitus on huolehtia olion tilan muutoksien tallennuksista tietokantaan, jolloin ohjelmoija voi keskittyä muuhun sovelluslogiikkaan. (ORM)

Luokkaa, joka määrittelee relaatiomallin, kutsutaan entiteetiksi. Entiteetit ovat PHP-olioita, jotka voidaan tunnistaa yksilöllisen tunnisteen tai pääavaimen avulla ja joiden ominaisuuksien tilaa voidaan vaihtaa ja tallentaa. Entiteetin määritys alkaa luomalla PHP-luokka, joka kuvastaa tietokantataulua kentiltään. Entiteetille määritetään metatiedoilla, kuinka olion ominaisuudet ja viittaukset pitäisi tallentaa joko sisällytettynä ohjelmakoodiin luokan ja metodien päälle käyttäen Doctrinen Docblock-annotaatioita, XML tai YAML -formaatissa. (ORM) ORM olettaa, että luokan nimi ja sen ominaisuuksien nimet vastaavat tietokannasta löytyviä taulun ja sarakkeiden nimiä. Jos kuitenkin halutaan käyttää toisia nimiä pitää ORM:lle kertoa metatiedoissa tietokannasta löytyvä nimi. Ominaisuuksille voidaan antaa tieto tietueen tietotyypistä ja tieto siitä mihin toiseen entiteetti luokkaan ominaisuuden arvo viittaa. (ORM) Kuvassa 1 on yksinkertainen esimerkki entiteetistä, joka on merkitty annotaatioilla ja joka sisältää kolme tietotyypiltään erilaista ominaisuutta sekä yhden ominaisuuden, joka on viittaus toiseen olioon.

```php
use Doctrine\Common\Collections\ArrayCollection;

/**
 * @Entity @Table(name="users")
 **/
class User {

    /** @Id @Column(type="integer") @GeneratedValue **/
    protected $id;

    /** @Column(type="string") **/
    protected $name;

    /** @Column(type="datetime") **/
    protected $dateOfBirth;

    /**
     * @OneToMany(targetEntity="Task")
     **/
    protected $tasks

    public function __construct() {
        $this->tasks = new \ArrayCollection();
    }
}
```

Kuva 1 Esimerkki ORM - entiteetin PHP- ohjelmakoodista

Column-annotaatiolla kerrotaan muuttujan arvoon liittyviä ominaisuuksia. Sillä voidaan kertoa esimerkiksi tietotyyppi (type), tietokannan kentän nimi (name) ja tietokannan kentän pituus (length). Koska jokaisella entiteetillä pitää olla yksilöllinen avain, id-ominaisuudelle määritetään Id-annotaatio. ORM:lle pitää kertoa kuinka tämä avain on luotu ja tähän on useitakin ratkaisuja, mutta yleensä riittää GeneratedValue–annotaatio, joka määrittää ominaisuuden käyttävän tietokannan tapaa luoda yksilöllinen avain. Esimerkiksi MYSQL:n tapauksessa AUTO_INCREMENT.

Doctrinessä viittaukset ovat aina viittauksia toisiin oliohin eikä ohjelmoijan tarvitse huolehtia viittausavaimista, jotka liittävät olioita vastaavat taulut yhteen. Viittausmetatiedot merkitään, joko OneToOne- , ManyToOne- , OneToMany- tai ManyToMany-annotaatioilla.

```
public function save( $obj ) {
    $this->_em->persist( $obj );
    $this->_em->flush();
}
```

Kuva 2 EntityManagerin persist ja flush metodien kutsu

3.2 EntityManager

Kun entiteetti on luotu sitä voidaan käyttää aivan kuin normaaleissa oliohjelmoinnin perusolemukseen liittyvissä toimenpiteissä. Koska entiteetit kuitenkin ovat rinnastettavissa oikeisiin tietokannan riveihin niiden tilaa ja olemassaoloa hallitsee Doctrinen ohjelmointirajapinta EntityManager. Uuden olion tai jo olemassaolevan olion tietojen tallennus tapahtuu persist-metodin avulla, joka ilmoittaa EntityManagerille että kyseisen olion tiedot pitäisi tallentaa tietokantaan. Tämä metodi ei kuitenkaan vielä tallenna tietoja vain asettaa ne valmiiksi tallennukseen. Jotta tieto oikeasti tallentuisi tietokantaan täytyy kutsua EntityManagerin flush-metodia. Nämä kaksi metodia ovat olemassa, jotta useita eri oliota, joihin on kohdistunut erillaisia muutoksia voidaan tallentaa yhtäaikaa. Koska kaikki muutokset, jotka on rekisteröity persist–metodilla, toteutuvat vasta kun flush–metodia kutsutaan. (ORM) Kuvassa 2 on entiteetin tallennus toteutettu yhden metodin taakse.

3.3 DQL

DQL tulee sanoista Doctrine Query Language ja se on oliopohjainen kyselykieli. DQL:llä tehdyt kyselyt voi kuvitella kohdistuvan tietokantaan, joka sisältää olioita. Sen avulla voidaan hakea yksittäisiä olioita tai useita olioita käyttäen luokkien nimiä, luokkien välisiä suhteita tai luokan ominaisuuksien nimiä. Kielellisiä yhtäläisyyksiä löytyy paljon SQL–kieleen, mutta hakulauseet kohdistuvat luokkiin ja niiden ominaisuuksiin tietokannan kenttien sijaan. (DQL) Kuvassa 3 on esimerkki DQL–kyselystä, jossa haetaan User-entiteettiä pääavaimen avulla.

```
$query = $em->createQuery('SELECT u FROM \User u WHERE u.id = :userId')
$query->setParameter(['userId', 1001])
$result = $query->getResult();
```

Kuva 3 Yksinkertainen kysely DQL-kielellä

3.3.1 Kyselyt

DQL sisältää SELECT- , UPDATE- ja DELETE-kommennot, jotka vastaavat SQL:n komentoja. INSERT-komento ei ole sallittu DQL:ssä, koska entiteetit ja niiden suhteet pitää tallentaa EntityManagerin persist–metodin kautta. DQL tukee sekä nimellisiä parametrejä sekä indeksoituja parametrejä. Indeksoituja parametreja käytettäessä kyselyn parametrit korvataan kysymerkillä, nimellisiä parametrejä käytettäessä kyselyn parametrit korvataan kaksoispisteellä ja parametrin nimellä. DQL:ssä voidaan käyttää matemaattisia ilmauksia ja SQL:stä tuttuja BETWEEN- , IN-, NOT IN- , LIKE- ja EXISTS-ilmauksia. DQL ymmärtää taulujen liitokset automaattisesti, joten ohjelmoijan ei tarvitse erikseen määritellä kyselylle taulujen viittauksia. (DQL)

3.3.1.1 QueryBuilder

Doctrinen QueryBuider tarjoaa rajapinnan, joka sisältää luokkia ja metodeja, jolla voidaan rakentaa DQL kyselyitä vaiheittain. Rajapinta sisältää apumetodeja, jotka vastaavat DQL:n komentoja. Esimerkiksi SELECT- , FROM-, WHERE-, ORDER BY- ja JOIN-kommennot ovat kaikki omia metodejaan voita voidaan putkitaan olio-ohjelmoinnin Builder Pattern -suunnitellumallin tavoin. (QueryBuilder) Kuvassa 4 on esimerkki QueryBuilder–luokan käytöstä. Opinnäytetyössä hyödynnettiin QueryBuilder-luokkaa rakentamaan kysely, jonka käyttäjä oli muodostanut valitsemalla ennalta arvaamattoman määrän hakuehtoja.

3.3.2 Paginator

Kun haetaan tuhansia rivejä tietokannasta, se voi johtaa muistin loppumiseen ja huonoon käytettävyyteen loppukäyttäjälle. Tätä varten on olemassa sivutus (Paginator), joka tarjoaa rajapinnan tietokannan rivien hauille sivuittain. Paginator luokalle annetaan ensin normaalisti luotu DQL – kysely, jonka jälkeen annetaan sivun numero ja haluttujen rivien määrä.

```
$qb = $em->createQueryBuilder('u')
        ->leftJoin('u.tasks', 't')
        ->where('t.id = :id')
        ->setParameter('id', 2002);

$user = $qb->getQuery()->getResult();
```

Kuva 4 Esimerkki QueryBuilder rajapinnan käytöstä

4 VUE.JS

Vue.js on avoimen lähdekoodin ohjelmointikehys, interaktiivisten käyttöliittymien rakentamiseen. Vuen yksi merkittävin piirre on reaktiivisuus. Sivun DOM–elementtiin liitetty tieto päivittyy automaattisesti, jos Vue–olion sisältämät tiedot muuttuvat. (GitHub, Vue.js) Vue.js on verrattavissa muihin ohjelmointikehyksiin, kuten React.js- ja Angular.js-ohjelmointikehyksiin. Vue.js tarjoaa uusia sekä samoja ideoita, kuin verrattavat ohjelmointikehykset.

4.1 Sapluunat

Vue.js käyttää HTML-pohjaista sapluunasyntaksia, millä Vue olion tiedot voidaan kiinnittää sivun DOM–elementteihin. (GitHub, Vue.js) Kuvassa 5 näkyy sapluunasyntaksin yksinkertaisin esimerkki, jossa kaarisulkeiden sisään kirjoitetut olion ominaisuudet kirjoitetaan DOM–elementtiin. Vue.js tukee Javascriptin lausekkeita kaarisulkeiden sisällä, joten siellä voidaan toteuttaa ehtolausekkeita, funktion kutsuja ja muita operaatioita. (GitHub, Vue.js)

4.2 Direktiivit

Direktiivit ovat attribuutteja, joita voi antaa elementeille. Nämä attribuutti kertovat Vuen kirjastolle, että kyseiselle elementille pitää tehdä jotain muutoksia. Vue.js:n direktiivit alkavat aina v-etuliitteellä, kuten v-for-direktiivi. Vue.js mahdollistaa elementtien luonnin ohjelmointikielistä tutuilla ehtolauseilla ja silmukoilla, jolloin HTML – elementteihin voidaan lisätä if- , else- ja else if-ehtolauseita ja for-silmukoita. Direktiiveille voidaan antaa argumentti, joka merkitään direktiivin perään kaksoispisteellä ja argumentin nimellä. Tämä kertoo Vue.js:lle, että argumentiksi tuleva olio tai sen ominaisuus, pitää yhdistää elementin attribuuttiin. Valmiiden direktiivien käytön lisäksi voidaan luoda omia direktiivejä, mutta Vue.js 2.0 version myötä uudelleen käytettävien koodin pääasiallinen tapa on käyttää komponentteja.

```
<div id="main">
    {{ message + user.name }}
</div>

<script type="javascript">
    var main = new Vue({
        el: "#main",
        data: {
            message : "Hello, ",
            user : { name : "Ville" }
        }
    })
</script>
```

Kuva 5 Yksinkertainen Vue instanssi ja sapluunasyntaksi

4.3 Komponentit

Vue.js:n avulla voidaan luoda komponentteja, jotka laajentavat HTML - elementin ominaisuuksia uudelleen käytettävässä muodossa. (GitHub, Vue.js) Komponentit voivat sisältää HTML- , CSS- ja Javascript-koodia, eli käytännössä komponentit voivat olla jopa omia sivujaan, joita ohjelmoija voi käyttää uudelleen. Globaali komponentti määritellään kutsumalla Vuen component-metodia. Metodille annetaan komponentin nimi ja asetukset. Komponentin asetukset vaativat vähintään kuvassa 6 nähtävät ominaisuudet. Template–ominaisuus määrittää elementin HTML-koodin ja tämä voidaan antaa joko suoraan merkkijonona tai viittaamalla HTML–elementtiin. Props–ominaisuus määrittää parametrit, jotka annetaan komponentille sen määritysvaiheessa HTML–koodiin, käyttäen v-bind-direktiiviä, mikä yleensä lyhennetään kirjoittamalla pelkästään kaksoispiste ja parametrin nimi. Kuvassa 6 table-component–elementille annetaan rows- ja columns-ominaisuudet. Komponentteja voi määrittää, joko kuvassa 6 näkyvällä tavalla tai kapsuloimalla kaikki HTML- , CSS- ja Javascript-koodit yhden tiedoston alle (single-file component), joka merkitään .vue-tiedostopäätteellä. Näin luodut ohjelmakoodit eivät ole globaalissa nimiavaruudessa. Tämä kuitenkin vaatii Javascriptin koodin kääntämistä, esimerkiksi käyttäen Webpack-kirjastoa.

4.4 Animaatiot

Vue.js tarjoaa tapoja asettaa animaatioita, kun elementtejä lisätään, muokataan ja poistetaan. Tämä tapahtuu lisäämällä transition-komponentti animoitavan elementin ympärille. Tälle komponentille voidaan antaa attribuutteja, jotka kertovat millaisen animaation elementille pitää suorittaa. Attribuuteilla voidaan antaa CSS-kielellä tehtyjä muunnoksia tai animaatioita tai integroida toiminnallisuuksia kolmannen osapuolen kirjastoista.

```
<div id="main">
    <table-component :rows="rows" :columns="columns"></table-component>
</div>

<script type="javascript">
    Vue.component('table-component' {
        template: ` <table>
                      <tr v-for="row in rows">
                         <td v-for="column in columns"> row[column] </td>
                      </tr>
                    </table>`,
        props : { rows : Array, columns : Array }
    })
    var main = new Vue({
        el: "#main",
        data: {
            columns : [ 'name', 'number', 'ts' ],
            rows : [ { name : 'ville', number : '1001', ts : '1494475029' } ]
        }
    })
</script>
```

Kuva 6 Yksinkertainen Vue instanssi ja globaalin komponentin käyttö

4.5 Instanssi

Jokainen Vuen näkymä alustetaan kutsumalla Vuen rakentaja metodia (Kuva 6). Vuen alustuksessa metodille pitää antaa olio, joka sisältää tarvittavat asetukset. Toimiakseen Vue instanssi tarvitsee kaksi ominaisuutta el- sekä data-ominaisudet. el-ominaisuudella kerrotaan mille tietylle HTML-osiolle halutaan tämän kyseisen instanssin toiminnallisuudet kohdistuvan. Data-minaisuus sisältää kaikki tiedot, jotka halutaan olla reaktiivisia ja jotka halutaan pystyä kiinnittämään DOM elementteihin. On otettava huomioon, että vain data-olioon määritetyt ominaisuudet ovat reaktiivisia ja myöhemmin lisättyjä ominaisuuksia Vue ei ota huomioon. Uudet ominaisuudet voidaan kuitenkin rekisteröidä erikseen Vue.set-metodia käyttäen.

Vue instanssiin voi myös määritellä laskettuja arvoja (computed), metodeja (methods) ja tarkkailu metodeja (watchers). Methods-ominaisuuteen voidaan lisätä funktioita, joita halutaan kutsua joko sapluunoissa tai kiinnittää elementtien tapahtumiin. Lasketut arvot ovat metodeja, joita voi käyttää DOM:ssa muuttujan tapaan, mutta ne niiden arvo perustuu metodin toteutuksesta palautettavaan arvoon. Tarkkailumetodit ovat funktioita, joilla reaktiivisen ominaisuuden muutokset voidaan ottaa kiinni. Tarkkailumetodit nimetään sen reaktiivisen ominaisuuden mukaan, jonka muutokset halutaan ottaa kiinni. Lasketun arvon funktio, sekä tarkkailumetodit käydään läpi aina kun niiden toteutuksessa käytetty reaktiivinen tieto muuttuu. (GitHub, Vue.js)

5 PORTAALIN TIETOKANTA JA ENTITEETIT

Sovelluksen tietokanta toimii MySQL-relaatiotietokanta. Tietokannasta löytyy taulut liittymien ja lokimerkintöjen tallentamiseen, sekä liitostaulu tunnisteiden ja liittymien välille. Tauluihin käytettiin viiteavainmäärittelyä vaikka Doctrinelle oli määritetty taulujen väliset suhteet.

5.1 Mallien luonti

Entiteetit luotiin kaikista tauluista, jotka työhön tarvittiin. Päätaulun entiteetti on Sim-luokka (Kuva 8), lokitieto taulun entiteetti on SimLog-luokka ja lisäksi tunnistetaulun liitostaulun entiteetti on SimLabel-luokka. Sim-luokka määritettiin kertomalla luokan olevan entiteetti, kirjoittamalla kohdetaulun nimi. Tämän jälkeen luokan ominaisuuksille määritetään tietotyyppi, mihin tietokannan vastaavan sarakkeen arvo halutaan muutettavaksi. Id-kenttä on merkitty vielä erikseen Id- sekä GeneratedValue-metatiedoilla, koska ORM:n halutaan käyttävän MySQL:n luomaa pääavainta. (ORM) Kuvassa 8 on merkitty viittaus device- sekä labels-ominaisuuksiin. Jokaisella laitteella on vain yksi liittymä, joten device–ominaisuus on merkitty OneToOne-metatiedolla. Labels-ominaisuus on taulukko, joka pitää liittymälle lisättyjä tunnisteita. Tämä ominaisuus on merkitty OneToMany-metatiedolla, sillä liittymällä voi olla useita eri tunnisteita.

SIM-luokasta on tehty abstrakti ja jokainen eri operaattorin luokka perii tämän luokan. Tämä kerrotaan ORM:lle merkitsemällä InheritanceType-metatietoon SINGLE_TABLE-perintätyyppi. Nimensä mukaisesti tämä luo taustalle yhden virtuaalisen taulun, johon kaikki perivien luokkien ominaisuudet listataan. DiscriminatorColumn-metatieto, kertoo minkä tietokannan kentän perusteella olio luodaan. DiscriminatorMap-metatieto määrittää, että mikä tietokannan arvo vastaa tiettyä Sim–luokasta periytyvää luokkaa. Kuvassa 7 on sim taulun malli SQL-lauseena, joka on sim-luokkaa vastaava relaatiotietokannan taulu.

```
CREATE TABLE `sim` (
    `id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    `simId` VARCHAR(30) NOT NULL DEFAULT '0' COMMENT 'SIM - tunniste, ICCID' COLLATE 'utf8_unicode_ci',
    `number` VARCHAR(30) NULL DEFAULT NULL COMMENT 'Puhelinnumero, MSISDN' COLLATE 'utf8_unicode_ci',
    `PIN` VARCHAR(4) NULL DEFAULT NULL COLLATE 'utf8_unicode_ci',
    `state` ENUM('Activated','Deactivated','Terminated') NOT NULL DEFAULT 'Deactivated' COLLATE 'utf8_unicode_ci',
    `type` ENUM('tele2','sonera-b2b','sonera-m2m','dna') NOT NULL COMMENT 'Palveluntarjoaja' COLLATE 'utf8_unicode_ci',
    `plan` VARCHAR(70) NULL DEFAULT NULL COMMENT 'Palveluntarjoajan liittymätyyppi' COLLATE 'utf8_unicode_ci',
    `description` VARCHAR(100) NULL DEFAULT NULL COLLATE 'utf8_unicode_ci',
    PRIMARY KEY (`id`)
)
```

Kuva 7 Sim taulun rakenne

```php
<?php

namespace Sim;

use \Doctrine\ORM\Mapping as ORM;
use Doctrine\Common\Collections\ArrayCollection;

/**
* @ORM\Entity( repositoryClass="\Repositories\Sim" )
* @ORM\Table(name="sim")
* @ORM\InheritanceType("SINGLE_TABLE")
* @ORM\DiscriminatorColumn(name="type", type="string")
* @ORM\DiscriminatorMap({"sonera-m2m" = "\Sim\SoneraM2M", "sonera-b2b" = "\Sim\SoneraB2B", "dna" = "\Sim\Dna", "tele2" = "\Sim\Tele2"}),
**/
abstract class Sim {
    /**
    * @ORM\Id
    * @ORM\Column(type="integer")
    * @ORM\GeneratedValue
    **/
    protected $id = null;

    /** @ORM\Column(type="string") */
    protected $simId;

    /** @ORM\Column(type="string") */
    protected $number;

    /** @ORM\Column(type="string") */
    protected $PIN;

    /** @ORM\Column(type="string") */
    protected $state;

    /** @ORM\Column(type="string") */
    protected $plan;

    /**
    * @ORM\OneToOne(targetEntity="\Device\GPSDevice", mappedBy="SIM")
    **/
    private $device;

    /**
    * @ORM\OneToMany(targetEntity="\Sim\SimLabel", mappedBy="sim")
    **/
    private $labels;

    /** @ORM\Column(type="string") */
    protected $description = '';

    public function __construct() {
        $this->labels = new ArrayCollection();
    }
```

Kuva 8 Sim entiteetin ohjelmakoodi

5.2 Haku

Portaalin käyttöliittymä koostuu melkein kokonaan taulukosta, jossa liittymiä voidaan hakea erillaisilla hakuehdoilla. Koska käyttöliittymästä käyttäjä pystyy valitsemaan halutun määrän hakuehtoja, sisältäen haut kohdistuen joko tiettyyn tauluun tai vapaan haun kohdistuen kaikkiin taulun kenttiin, käytettiin liittymien hakuun hyödyksi Doctrinen QueryBuilder-luokkaa, jolla saatiin pala palalta luotua kysely ja toteutettua se. Kuvassa 9 otetaan vastaan käyttöliittymästä lähetetyt taulukon arvot, jotka välitetään Sim-entiteetin Repository-luokalle suoritettavaksi. Tulokset palautetaan JSON muodossa takaisin pyynnön suorittajalle.

```php
public function getSIMs() {

    $settings = array();
    $settings['sort']['sortKey'] = $this->_request->getValue("sortKey", null);
    $settings['sort']['sortOrder'] = $this->_request->getValue("sortOrder", null);
    $settings['filter']['filterKey'] = $this->_request->getValue("q", null);
    $settings['filter']['filterKeys'] = $this->_request->getValue("filterKeys", null);
    $settings['page']['index'] = $this->_request->getValue("page", null);
    $settings['page']['limit'] = $this->_request->getValue("limit", null);

    $result = $this->getRepository('Sim\Sim')->fetchSims($settings);

    return $this->jsonResponse($result);
}
```

Kuva 9 Controllerin metodi liittymien hakuun

```
public function fetchSims($setting) {
    $qb = $this
    ->createQueryBuilder('s')
    ->leftJoin('s.device', 'd')
    ->leftJoin('d.company', 'c')
    ->leftJoin('s.labels', 'sl')
    ->leftJoin('sl.label', 'l');
    ...
    $qb->OrderBy($name, $setting['sortOrder']);
    foreach($setting['filterKeys'] as $name => $searchItem) {
            $value = $searchItem['value'];
            ....
            if($value == 'null') $qb->andWhere($short.'.'.$name.' IS NULL');
            else $qb->andWhere($short.'.'.$name.$accuracy.':'.$name)->setParameter($name, $value);
       }
    }
    ...
    $qb->andWhere('d.DeviceID = :q OR d.DeviceName LIKE :joker OR
        c.FirstLevelCounter = :q OR c.CompanyName LIKE :joker OR
        s.description LIKE :joker OR l.name LIKE :joker')
        ->setParameter('joker', '%' . $filterKey. '%')
        ->setParameter('q', $filterKey);
    }
    ....
    $qb->setFirstResult($start);
    $qb->setMaxResults($limit);
    ....
    $query = $qb->getQuery();

    ...
}
```

Kuva 10 Typistetty versio metodista, joka luo kyselyn annettujen arvojen mukaan

Kuvassa 10 on huomattavasti tiivistetty versio metodista, joka muodostaa QueryBuilderin avulla kyselyn annettujen parametrien perusteella. Tiivistelmästä on jätetty pois esimerkiksi kaikki tarkistukset, jossa tarkistetaan onko arvoa annettu ja erityistapausten käsittely. Kyselyn rakentaminen alkaa liittämällä viitatut oliot kutsumalla leftJoin-metodia. Kyselyyn liitetään telematiikkalaite device-entiteetti, asiakas company-entiteetti, sekä tunnisteet. Tämän jälkeen iteroidaan kaikki sarakekohtaiset hakuehdot (filterKeys) ja lisätään vapaan haun arvo (filterKey) hakemaan useista sarakkeista. SetFirstResult-metodilla määritetään mistä kohtaa eteenpäin tuloksista palautetaan arvot ja setMaxResults-metodilla määritetään, kuinka paljon arvoja palautetaan. Näillä metodeilla saadaan aikaan sivutus.

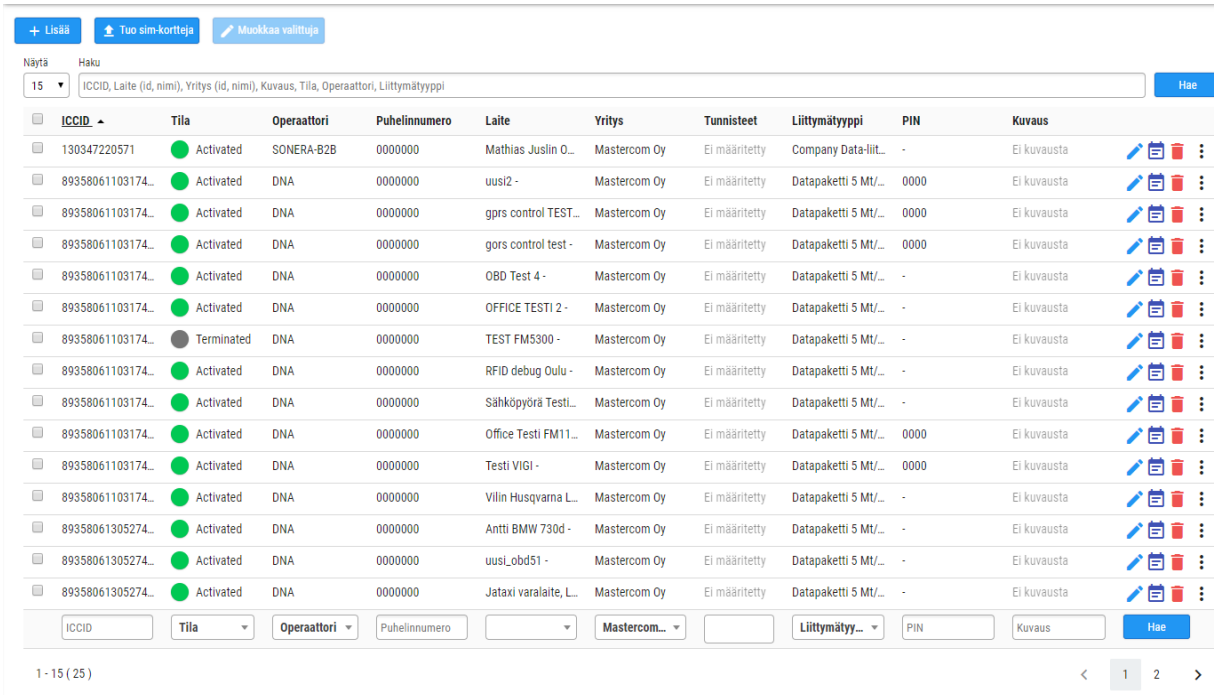
6 PORTAALI

Sovelluksen käyttöliittymä koostuu melkein täysin yhdestä, tätä työtä varten tehdystä, uudelleen käytettävästä, Vue.js komponentista. Taulukkokomponentin ominaisuuksien lisäksi on mahdollisuus lisätä liittymä, tuoda liittymiä taulukkomuotoisista tiedostoista ja muokata taulukosta valittuja liittymiä.

6.1 Taulukkokomponentti

Taulukkokomponentti on suunniteltu alusta lähtien käytettäväksi myös tämän työn ulkopuolella. Komponentin ominaisuuksiin kuuluu hakukenttä vapaan haun suorittamiseksi, kohdistuen kaikkiin taulukossa nähtäviin sarakkeisiin, rivi määrän valinta, sarakekohtainen haku, sekä sivun valinta (Kuva 11). Valitun sarakkeen voi järjestellä laskevaan tai nousevaan järjestykseen. Komponentin alustuksessa voi määritellä halutaanko taulukon järjestely, haku ja sivutus toteuttaa selaimessa vai palvelimella. Tässä työssä työn tekee palvelin, sillä tietomäärät ovat niin suuria, että tuhansien tietojen lataaminen selaimelle suoritettavaksi ei olisi tehokasta. Jos valitsee palvelimen suorituksen, taulukosta valitut hakuparametrit siirtyvät Ajax-pyynnöllä palvelimelle.

Jokaisen taulukon soluun voi käyttää komponenttia pelkän tekstin sijaan ja jokainen sarakekohtainen hakuelementti on myös kustomoitu komponentti, jotta hakuparametrit voidaan valita sopivimmalla tavalla kyseiselle sarakkeelle. Taulukkoon on myös mahdollista määrittää actionsItems-ominaisuus, joka asettaa jokaisen rivin viimeiseksi sarakkeeksi ohjelmoijan määrittämät komponentit, jotka kaikki ottavat parametrina rivin tietueen. Tässä työssä nämä komponentit ovat liittymän muokkaus, muutoshistoria ja rivin poisto ominaisuus.



Kuva 11 Portaalin käyttöliittymä

```
33            <tfoot v-if="rows.footer">
32                <tr>
31                    <th v-if="rows.lineNumbers"></th>
30                    <th v-if="rows.selected"> </th>
29                    <template v-for="(column, index) in rows.columns">
28                        <th :class="getHeaderClass(column, index)">
27                            <component v-if="column.editTemplate" v-bind:is="column.editTemplate"
26                                :column="column"
25                                @update="updateFilterKey">
24                            </component>
23                            <template v-else>
22                                {{ column.header }}
21                            </template>
20                        </th>
19                    </template>
18                    <th>
17                        <div class="table-control">
16                            <button @click='filterServer'><?= __('Hae') ?></button>
15                        </div>
14                    </th>
13                </tr>
12            </tfoot>
```

Kuva 12 Taulukkokomponentin hakukomponenttien HTML - ohjelmakoodi

Kuvasta 15 näkyy portaalin näkymän Vue instanssin alustuksen ohjelmakoodi. Kuvasssa columns-ominaisuudelle on annettu taulukko, joka sisältää jokaisen sarakkeen asetukset aina kentän nimestä kustomoituun komponenttiin. Kuvassa 14 on ote taulukosta, joka määrittelee näytettävät sarakkeet. Kuvassa näkyy, että olioon määritellään kentän nimi (key), sarakkeen otsikko (header) ja komponentti, joka on sarakkeen hakuelementti (editTemplate). Koska jokainen solu voi olla myös komponentti, se määritellään template-ominaisuuteen. Kuvassa 14 näkyy esimerkki tästä ominaisuudesta, missä tila-sarakkeelle on määritetty stateIndicator-komponentti (Kuva 13), joka sisältää liittymän tilaa indikoivan värikoodauksen sekä tekstin.

Kuvassa 12 on ote taulukkokomponentin HTML koodista, jossa luodaan taulukon alaotsikkojen kentät, jotka ovat tässä työssä hakuelementtejä. Rivillä rivillä 29 käydään silmukka läpi, joka iteroi kaikki annetut sarakkeet ja asettaa sarakkeelle, joko komponentin tai pelkän tekstin sen perusteella onko column-oliolle määritetty hakukomponetti vai ei. Kuvassa 12 rivillä 27 näkyy kuinka column.editTemplate-ominaisuus asetetaan is-argumentin arvoksi. Tästä seuraa, että component-elementin arvoksi tulee se komponentti, joka olion ominaisuudessa on. Samanlaisella logiikalla luodaan myös taulukon solut. Taulukolle annetut tiedot ja sarakkeet iteroidaan ja solut muodostetaan näiden tietojen sisältämien ominaisuuksien perusteella.

```
/*
 * State indicator
 */
let stateIndicator = Vue.component('state-indicator', {
    template: `<div><span :title='entry[column.key]' class='state-indicator' :class='entry[column.key]'></span>
                <p class='col_9' style='float:left'>{{ entry[column.key] }}</p></div>`,
    props : { column : Object, entry : Object }
});
```

Kuva 13 Vue.js – komponentin, joka ilmaisee liittymän tilaa.

```
gColumns = [
    {
        key : "simId",
        header : "ICCID",
        editable : true,
        editTemplate : new VueTableInput().getComponent()
    },
    {
        key : "state",
        header : __("Tila"),
        template : {
            component : stateIndicator,
            action : function() {  }
        },
        editTemplate : selectStates,
        canModifyForMultiple : true,
        editable : true
    },
```

Kuva 14 Ote Javascript koodista, joka kartoittaa näytettävän tiedon sarkkeiden asetukset

```
new Vue({
    el: '#sim-management',
    data : {
        sims : {
            name : 'sim-management',
            data : [],
            ajax : "admin/getSIMs",
            columns : gColumns,
            idColumn : "simId",
            searchPlaceholder: 'ICCID, Laite (id, nimi), Yritys (id, nimi), Kuvaus, Tila, Operaattori, Liittymätyy
            footer: true,
            loadingMessage : __("Haetaan SIM tietoja") + "...",
            zeroResultMessage : __("SIM - kortteja ei löytynyt"),
            actionItems : [
                { template : modify },
                { template : showLog },
                { template : remove }
            ],
            serverSide : true,
            selected : {},
            pagination: true,
            classes : {}
        }
    },
    methods: {
        addNew: function() {
            new Create(gColumns).show();
        },
        importing: function(e) {
            initImportTableStuff(e);
        },
        modifySelected: function() {
            new Editor(
                gColumns,
                this.sims.selected,
                true
            ).show();
        }
    }
});
```

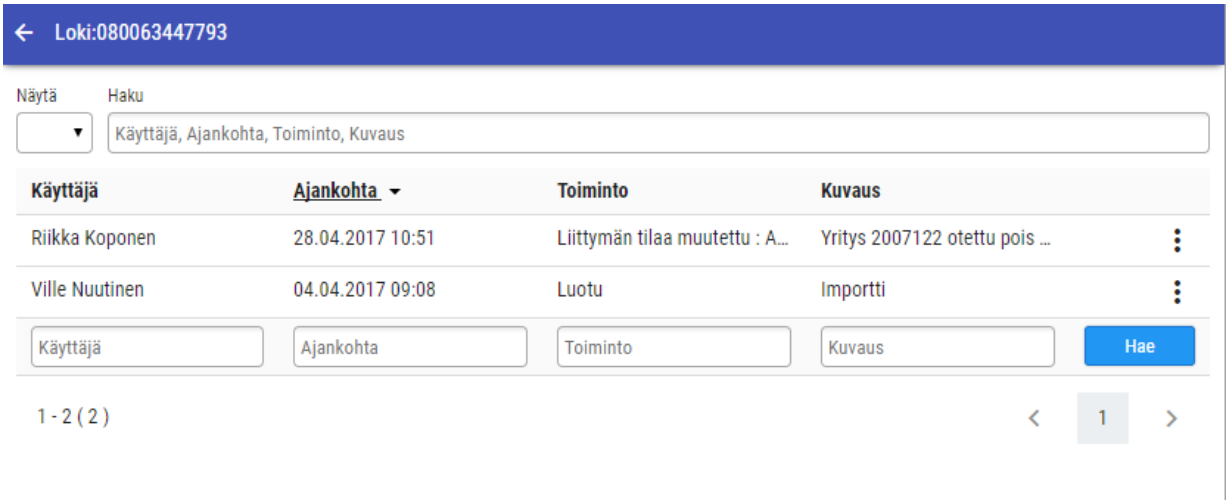
Kuva 15 Portaalin näkymän alustus Javascript-ohjelmakoodissa

6.2 Loki

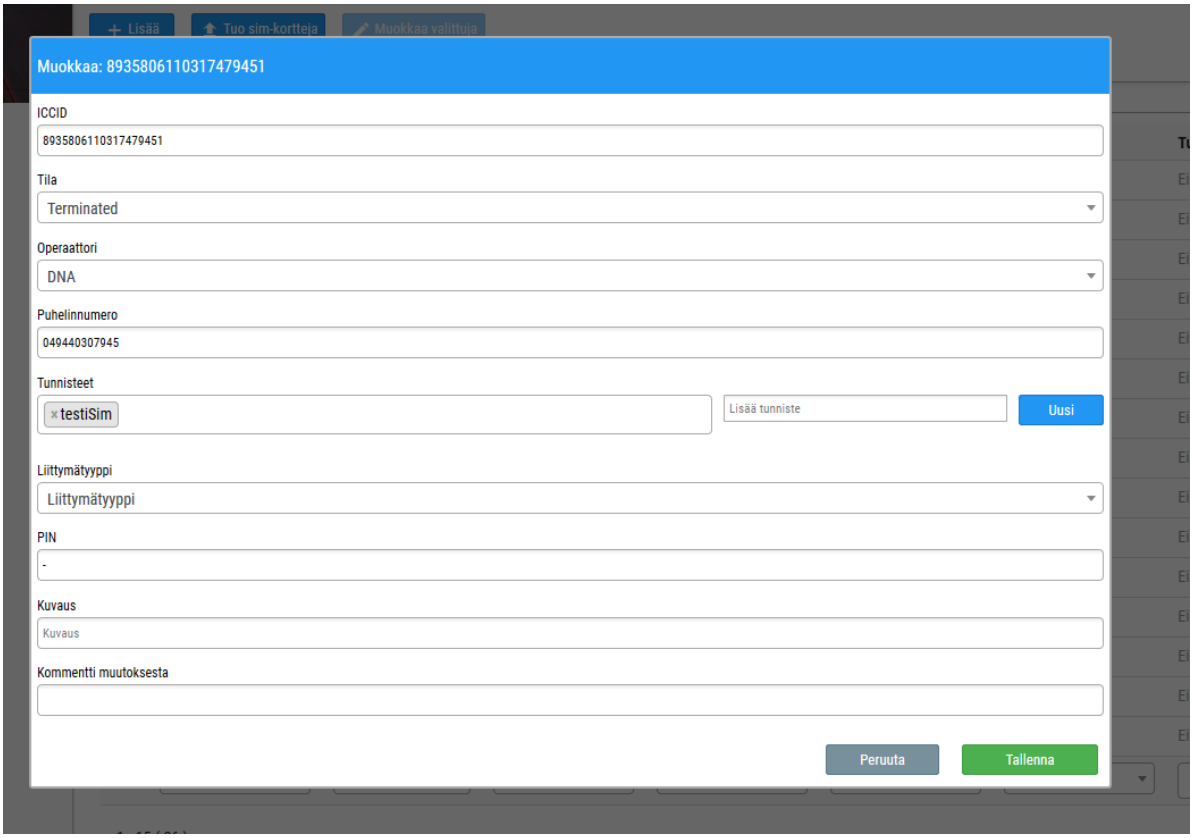
Yrityksen toive oli että kaikkia liittymiin tehtyjä muutoksia pystyttäisiin seuraamaan. Kaikki muutokset, joita pystytään tekemään muokkausnäkymästä (Kuva 17) tallennetaan lokitauluun. Lokitauluun tulee tieto muutoksen tekijästä, päivämäärästä, tieto vanhasta arvosta, tieto uudesta arvosta ja mahdollisesta kommentista. Lokiin kirjoituksen hoitaa, Sim-luokasta periytyvä Operator-luokka, jonka kaikki eri operaattoriluokat perivät. Tällä luokalla on metodit, kaikkien Sim-luokan ominaisuuksien lokiin kirjoittamiseen. Kuvassa 18 näkyy kuinka updateState-metodi kutsuu perittävän luokan samaa metodia, joka kirjoittaa lokiin. Lokimerkintöjen näyttämiseen luotiin lokinäkymä, jossa hyödynnettiin samaa taulukkokomponenttia kuin päänäkymässä (Kuva 16).

6.3 Tuonti

Työn määrittelyssä haluttiin myös ominaisuus, jolla operaattorien palveluista voitaisiin tuoda yrityksen liittymät taulukkomuodossa luotuun tietokantaan. Tätä ominaisuutta varten käytin yrityksessä jo luotua tuontityökalua, jolla tuodun taulukkomuotoisen tiedon sarakkeet kartoitettiin tietokannan vastaaviin sarakkeisiin. Yksi suuri syy tämän työn toteutukseen oli tarve varmistaa, että mikään liittymä, joka pitäisi olla suljettu ei olisi sulkematta. Ennen uuden liittymienhallinnan tekoa, yrityksessä oli tieto liittymästä ja sen tilasta samassa tietokantataulussa, missä telematiikkalaitteita pidettiin yllä. Yrityksessä oltiin varmoja siitä, että joitakin liittymiä oltiin merkitty suljetuksi, mutta joko inhimillisistä tai teknillisistä syistä näitä liittymiä ei koskaan suljettu. Tämän vuoksi toteutettiin ominaisuus, joka tarkisti että tuontityökalusta tuotu tieto liittymästä ja sen tilasta vastaisi tilaa, joka oli merkitty laitetaulun tietoihin. Myöhemmin tätä ominaisuutta käytetään varmistamaan, että liittymätaulun liittymien tila vastaa operaattorilta tuotujen liittymä tietojen tilaa. Jos ristiriitoja löytyi ne näytettiin taulukossa, jossa näkyi tuodun rivin tila ja tietokannassa olevan rivin tila, sekä liittymän ICCID.



Kuva 16 Liittymien loki



Kuva 17 Liittymän lisäys - ja muokkausnäkymä

6.4 Lisäys ja muokkaus

Vaikka yksittäisiä liittymiä tuskin lisätään, sillä jos uusia liittymiä tulee niin niitä tulee usein useita kerralla ja ne on helpoin lisätä tuontityökalua käyttäen, ei portaaliin olisi välttämättä tarvinnut tehdä ominaisuutta liittymän lisäykseen. Mutta koska liittymiä piti pystyä muokkamaan ja tähän tarkoitukseen piti tehdä käyttöliittymä, pystyi liittymien lisäyksen tekemään pienellä vaivalla. Liittymien lisäykseen ja muokkaamiseen käytetään samaa dialog–ikkunaa (Kuva 17). Liittymän muokkauksessa voi lisäksi kirjoittaa kommentin muutoksesta, joka tallennetaan lokimerkinnän yhteyteen. Portaalissa on ominaisuus usean liittymän muokkaamiseen, joilloin dialogissa tehdyt muutokset vaikuttavat kaikkiin valittuihin liittymiin. Tässä tapauksessa käyttöliittymässä ei kuitenkaan voi muokata puhelinnumero- , PIN- ja operaattori-kenttiä.

```php
public function updateState($state) {

    parent::updateState($state);

    if(!isset($state))
        $state = self::STATES['Deactivated'];

    $settings = array(
        'simId' => $this->getSimId(),
        'target' => self::STATES[$state],
        'type' => 3
    );

    $this->editTerminal($settings);
}

private function editTerminal($settings) {

    $service = new \SoapClient(self::TERMINAL,  array('exceptions' => TRUE));

    $params = array(
        'licenseKey' => static::LICENSE_KEY,
        'messageId' => '',
        'version' => '',
        'iccid' => $settings['simId'],
        'targetValue' => $settings['target'],
        'changeType' => $settings['type']
    );

    $header = $this->getAuthHeader();
    $service->__setSoapHeaders(array($header));

    try {
        $result = $service->EditTerminal($params);
    }
    catch(\SoapFault $e) {
        $this->getError($e);
    }
}
```

Kuva 7 Ohjelmakoodi yhden operaattorin liittymien tilan muuttamiseen

```xml
<!--  Change the status of a terminal to Deactivated  -->
<soapenv:Envelope xmlns:sch="http://api.jasperwireless.com/ws/schema" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-16847597">
        <wsse:Username>user</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <sch:EditTerminalRequest>
      <sch:messageId>?</sch:messageId>
      <sch:version>?</sch:version>
      <sch:licenseKey>xxxxxxx-xxxx-xxxx-xxxx-xxxx</sch:licenseKey>
      <sch:iccid>8901650600000009465</sch:iccid>
      <!-- Optional: -->
      <sch:targetValue>DEACTIVATED_NAME</sch:targetValue>
      <sch:changeType>3</sch:changeType>
    </sch:EditTerminalRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Kuva 19 XML-kuvaus EditTerminalRequest-pyynnöstä

7 TILAN MUUTOS OPERAATTORIN PALVELUUN

Jokaisen liittymien tietokantataulun rivin perusteella luotiin entiteetti, joka luokka määräytyi operaattorin perusteella. Näin ohjelmakoodissa pystyttiin, jokaisen operaattorin rajapintapyynnöt sisäistämään omiin luokkiinsa ja koska kaikki operaattoriluokat perivät Sim-luokan, pystyttiin tilan muutos vaiheessa kutsumaan updateState-metodia, joka suoritti oikeat rajapinnat pyynnöt operaattoriluokan perusteella.

Operaattorien rajapintoja käytettiin vain liittymien tilan muutoksiin ja ne toteutettiin esimerkiksi yhden operaattorin kohdalla SOAP-protokolan avulla. Kuvasta 18 näkyy ohjelmakoodi, joka lähettää pyynnön PHP:n SOAPClient-luokkaa käyttäen operaattorin rajapintaan. SOAPClient kutsuu EditTerminal-metodia, joka on tämän kyseisen operaattorin metodi liittymän ominaisuuksien muuttamiseen. Kuvassa 19 on operaattorin tarjoaman SOAP protokolan XML esimerkki, jonka mukaan SOAPClient-kutsun parametrit on määritetty.

Monista syistä kaikkien operaattorien rajapintoja ei voitu toteuttaa, joten jos sellaisen operaattorin liittymää muutettiin, jolla ei ollut rajapintatoteutusta, portaali esitti taulukon muutetuista liittymistä ja avasi sen operaattorin verkkoportaalin, johon liittymä kuului. Koska muutokset liittymän tilaan oli jo tässä vaiheessa tallennettu liittymätietokantaan, oli käyttäjästä kiinni että muutokset todella toteutettiin operaattorin palveluun. Tätä varten esitettyyn taulukkoon muutettavista liittymistä tehtiin pieni ominaisuus, joka pakotti käyttäjän merkitsemään jokainen muutettu liittymä valmiiksi, ennen kuin käyttäjä voisi jatkaa takaisin portaaliin.

8 YHTEENVETO JA POHDINTA

Työn tavoitteena oli toteuttaa portaali, joka kokoaisi usean operaattorin liittymät yhden käyttöliittymän alle ja jossa liittymien tilan muuttaminen tekisi pyynnön operaattorien palveluun. Tähän lopputulokseen päästiin ja ohjelmisto on nyt käytössä yrityksessä, vaikka kaikkien operaattorien rajapintoja ei saatu toteutettua.

Työ pysyi aikataulussa, mutta joihinkin osa-alueisiin kului enemmän aikaa kuin oli suunniteltu. Esimerkiksi taulukkokomponentin luonti yleispäteväksi ratkaisuksi vei aikaa verrattuna siihen, että olisi käytetty jotain valmista ratkaisua monipuolisen taulukon luontiin. Lopputulos kuitenkin palvelee ja on jo palvellut muissa projekteissa. Aikaa kului myös operaattorien rajapintojen tutkimiseen ja niiden ymmärtämiseen.

Työn toteuksessa tuli opittua enemmän Vue.js- ohjelmointikehyksen käyttöä, sillä työssä joutui tutustumaan uusiin ominaisuuksiin varsinkin taulukkokomponenttia tehdessä. Doctrinessä tutustuin uusiin ominaisuuksiin ja tuli opittua enemmän SOAP-protokolan käyttöä ja ymmärtämään sen toimintalogiikkaa.

Työhön jäi vielä jatkokehityksen aiheita. Liittymien muiden ominaisuuksien kuin tilan muuttaminen operaattorien palveluun pystyisi toteuttamaan pienellä vaivalla, esimerkiksi päivittämään liittymätyyppi käyttämään erillaista tiedonkäyttö rajoitetta ja sopimusta. Operaattorien rajapinnoista saisi myös haettua liittymän datan käytön tietoja, jolloin portaaliin voisi lisätä esimerkiksi raportointiominaisuuden.

LÄHTEET

DOCTRINE TEAM. ORM . 2006 – 2017. [Viitattu 2017-4-15] Saatavissa: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/index.html

DOCTRINE TEAM. DBAL . 2006 – 2017. [Viitattu 2017-4-15] Saatavissa: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/index.html

DOCTRINE TEAM. DQL . 2006 – 2017. [Viitattu 2017-4-18] Saatavissa: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html

DOCTRINE TEAM. QueryBuilder. 2006 – 2017. [Viitattu 2017-4-20] Saatavissa: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html

GITHUB. Vue.js. [Viitattu 2017-4-15] Saatavissa: https://github.com/vuejs/vuejs.org/tree/master/src/v2/guide

ORACLE CORPORATION. MySQL. 2017. [Viitattu 2017-4-15] Saatavissa: https://dev.mysql.com/doc/refman/8.0/en/introduction.html

MSDN. AJAX. [Viitattu 2017-4-15] Saatavissa: https://developer.mozilla.org/en-US/docs/AJAX

WIKIPEDIA. APACHE. [Viitattu 2017-4-15] Saatavissa: https://en.wikipedia.org/wiki/Apache_HTTP_Server

WIKIPEDIA. SOAP. [Viitattu 2017-4-15] Saatavissa: https://fi.wikipedia.org/wiki/SOAP

WIKIPEDIA. SIM. [Viitattu 2017-4-16] Saatavissa: https://fi.wikipedia.org/wiki/Subscriber_Identity_Module

WIKIPEDIA. PHP. [Viitattu 2017-4-15] Saatavissa: https://fi.wikipedia.org/wiki/PHP

WIKIPEDIA. DOM. [Viitattu 2017-4-15] Saatavissa: https://en.wikipedia.org/wiki/Document_Object_Model