

# **ASIAKASKONSOLI KAHDEN RAJAPINNAN INTEGRAATIOTA HYÖDYNTÄEN**



Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäki, Tieto- ja viestintätekniikka

Syksy, 2018

Mikko Tuppurainen

Tieto- ja viestintätekniikka  
Riihimäki

---

<b>Tekijä</b>	Mikko Tuppurainen	<b>Vuosi</b> 2018
<b>Työn nimi</b>	Asiakaskonsoli kahden rajapinnan integraatiota hyödyntäen	
<b>Työn ohjaaja</b>	Petri Kuittinen	

---

## TIIVISTELMÄ

Opinnäytetyön tavoitteena on rakentaa asiakaskonsolin ensimmäinen versio Louhi Networksille heidän verkkokauppaansa. Määritellyt ominaisuudet, mitä ensimmäinen versio tulee pitämään sisällään, on asiakkaan tili- ja yhteystietojen hallinta, palveluiden ja sopimusten seuraaminen, sekä laskujen seuraaminen ja PDF-version lataus asiakkaan tietokoneelle.

Toteutukseen käytetään samaa teknologiaa, jota on käytetty verkkokaupan toteutukseen, sekä lisäksi verkkokaupan ja Louhen oman rajapinnan välistä integraatiota asiakastietojen tuontia varten. Käyttöliittymä koostuu pääsääntöisesti HTML5-, JavaScript- ja CSS-teknologiasta ja niiden kirjas-toista ja web-sovelluskehysistä. Rajapinnat on rakennettu PHP-ohjelmointikielellä ja tietokannat on toteutettu PostgreSQL:llä.

Opinnäytetyö tehdään toimeksiantona Pilvi Cloud Companylle, jolta Louhi Networks on tilannut asiakaskonsolin. Pilvi on jo aikaisemmin rakentanut Louhelle heidän verkkokaupansa, jota Pilvi vielä ylläpitää ja kehittää.

**Avainsanat** API, integraatio, käyttöliittymä, tietokanta, Web-sovelluskehitys.

**Sivut** 52 sivua

Information and Communication Technology  
Riihimäki

---

<b>Author</b>	Mikko Tuppurainen	<b>Year</b> 2018
<b>Subject</b>	Customer console, utilizing integration between two APIs	
<b>Supervisor</b>	Petri Kuittinen	

---

ABSTRACT

The aim of this project was to build a first version of customer console for Louhi Networks on their e-commerce site. The features that were developed in the first version includes tools for a customer to manage his or her account and contact information, tools for the customer to view his or her services and contracts, and tools for customer to view his or hers invoices and as needed downloading a PDF version of it to their computer.

The same technologies used for the e-commerce implementation of Louhi Networks, as well as the integration between the e-commerce site and Louhi's own API for customer data import, were used in this project. The user interface mainly consisted of HTML5, JavaScript js CSS technologies and their libraries and web application frameworks. The APIs were built in the PHP programming language, and the databases were implemented with PostgreSQL.

The project was commissioned by Pilvi Cloud Company, which supplied the customer console for Louhi Networks. Pilvi had already built the e-commerce site for Louhi Networks, which Pilvi still maintains and develops.

**Keywords** API, database, integration, user interface, Web software development.

**Pages** 52 pages

# SISÄLLYS

1	JOHDANTO.....	1
2	PROJEKTISSA KÄYTETYT TEKNOLOGIAT .....	2
2.1	PHP .....	2
2.1.1	Tausta .....	2
2.1.2	Syntaksi.....	3
2.1.3	Tietokantatuki.....	4
2.2	Tietokanta: PostgreSQL.....	5
2.2.1	Tausta .....	5
2.2.2	Ominaisuudet .....	5
2.3	JavaScript.....	6
2.3.1	Tausta .....	6
2.3.2	Syntaksi.....	6
2.3.3	jQuery .....	8
2.4	Backbone.js .....	8
2.4.1	Model, collection & view .....	9
2.4.2	Event-tapahtuma ja ListenTo()-funktio .....	9
2.5	HTML .....	9
2.5.1	Tausta .....	10
2.5.2	Syntaksi.....	10
2.6	CSS/SCSS.....	11
2.6.1	Syntaksi.....	12
2.6.2	SCSS .....	13
2.7	Muita mahdollisia teknologioita .....	13
2.7.1	Angular.js.....	14
2.7.2	React.js .....	14
2.7.3	MySQL.....	14
2.7.4	MariaDB.....	14
2.7.5	Python.....	15
2.7.6	Java .....	15
3	PROJEKTISSA KÄYTETYT TYÖKALUT .....	16
3.1	Sublime Text.....	16
3.2	Eclipse.....	16
3.3	Versiohallinta .....	17
3.3.1	Git .....	17
3.3.2	Apache Subversion – SVN.....	18
3.4	PgAdmin .....	18
3.5	Kehitysympäristö.....	18
3.5.1	Oracle – Virtualbox .....	18
3.5.2	Vagrant .....	18
3.6	Muita mahdollisia työkaluja .....	18
3.6.1	Notepad++ .....	19
3.6.2	GNU Emacs .....	19
3.6.3	Navicat for Postgre SQL.....	19

3.6.4	Mercurial .....	19
4	KONSOLIN SUUNNITTELU .....	20
4.1	Konsolin tarve ja toimeksianto.....	20
4.2	Ohjelmistokehityksen periaatteita.....	20
4.3	Konsolin toteutuksen suunnittelu.....	22
4.3.1	Tilin asetukset-osio .....	22
4.3.2	Palvelut-osio .....	23
4.3.3	Laskut-osio.....	24
4.4	Suunnitelman dokumentointi ja esittäminen asiakkaalle.....	26
5	KONSOLIN TOTEUTUS.....	27
5.1	Rajapinnat ja niiden integraatiot.....	27
5.1.1	PAITA – rajapinta .....	27
5.1.2	Service Integration API .....	28
5.1.3	Service Management API .....	28
5.1.4	Hostmaster v2.0 – rajapinta .....	28
5.2	Kaupan API .....	29
5.2.1	Tilin asetukset-osion API toteutus.....	29
5.2.2	Palvelut-osion API toteutus .....	33
5.2.3	Laskut-osion API toteutus.....	35
5.2.4	Virhetilanteiden näyttäminen .....	36
5.3	Käyttöliittymä.....	37
5.3.1	Tilin asetukset-osion käyttöliittymä .....	38
5.3.2	Palvelut-osion käyttöliittymä .....	40
5.3.3	Laskut-osion käyttöliittymä .....	41
5.3.4	Virhetilanteiden näyttäminen käyttöliittymässä.....	42
5.4	Konsolin toiminnollisuuksien testaaminen .....	42
6	YHTEENVETO .....	44
	LÄHTEET .....	45

## 1 JOHDANTO

Tämän opinnäytetyön tavoitteena on toteuttaa uuden asiakaskonsolin ensimmäinen versio Louhi Networks Oy:n verkkokauppaan. Tavoitteina on myös luoda selkeällä koodilla toimiva pohja asiakas konsolin jatkokehitystä varten, sekä käydä läpi uuden konsolin ominaisuuksia ja toimintoja. Opinnäytetyön toimeksiantaja on Pilvi Cloud Company Oy, jolta Louhi Networks tilasi uuden asiakaskonsolin. Olin suorittanut harjoitteluni Pilvi Cloud Companylla vuonna 2017 tuotekehitystiimissä ja jatkoin harjoitteluni jälkeen heillä työskentelyä. Työnkuvaani kuului erilaiset asiakasprojektit, Pilven verkkokauppa-tuotteen kehittäminen ja asiakkaiden tilaamien verkkokauppojen ylläpitäminen. Louhi Networksin verkkokauppa on kustomoitu versio Pilven verkkokauppa-tuotteesta, jota myös ylläpidetään ja kehitetään jatkuvasti.

Ensimmäinen versio asiakaskonsolista toimii pohjana konsolin tulevalle jatkokehitykselle. Se pitää sisällään muutamia ennalta määrättyjä toiminnollisuuksia. Konsolin käyttöliittymä ei tule olemaan viimeistelty versio, sillä siihen liittyvät UI-määritelmät ovat vielä työn alla.

Asiakaskonsolia Louhen verkkokauppaan on suunniteltu jo ennen projektin alkua. Sainkin tehtäväksi toteuttaa sen ensimmäisen version annettujen määritelmien mukaan. Tämä oli oivallinen tilaisuus tehdä opinnäytetyö toimeksiannosta. Projekti on monipuolinen, ja pitää sisällään useita web-sovelluskehityksen osa-alueita front-end ja back-end tasolla.

Koska projekti toteutettiin valmiiseen ympäristöön, määräytyvät siihen käytetyt teknologiat, ja osa työkaluista, sen mukaan. Toteutuksessa on myös otettu huomioon verkkokaupan aiempaa toteutusta, jotta saadaan yhteneväisyys uuden koodin ja aiemman koodin välille. Olen aikaisemmin tehnyt projekteja liittyen Louhi Networksin verkkokauppaan, joten se oli tullut tutuksi koodin ja ominaisuuksien tasolla.

Opinnäytetyössä tullaan käymään läpi konsolin toteutukseen käytettyjä teknologioita ja työkaluja. Työssä käydään myös läpi projektin alkuvaiheen suunnittelua ja omia lähtökohtiani tähän projektiin sekä muutaman web-sovelluksen kehityisperiaatteita. Suunnittelun jälkeen käyn läpi projektin toteutusta ja tehtyjä testauksia liittyen API:hin ja käyttöliittymään. Lopuksi arvioin projektin lopputulosta ja asiakaskonsolin ominaisuuksien käytettävyyttä sekä sitä, miten opinnäytetyö onnistui.

## 2 PROJEKTISSA KÄYTETYT TEKNOLOGIAT

### 2.1 PHP

PHP (PHP: Hypertext Preprocessor) on scriptikieli, joka on pääsääntöisesti suunniteltu dynaamisten web-sivujen toteuttamiseen. PHP toimii suuressa roolissa projektissa. Sitä on käytetty molemmissa rajapinnoissa, sekä näiden rajapintojen väliseen integraatioon. Projektiin liittyvät API-funktiot, joilla dataa saadaan välitettyä rajapintojen välillä, toteutettiin PHP:lla. PHP:lla myös tuodaan ja vastaanotetaan dataa asiakaskonsolin käyttöliittymästä.

#### 2.1.1 Tausta

Nykypäivänä tunnettu PHP-ohjelmointikieli sai alkunsa, kun 1994 tanskalais-kanadalainen ohjelmoija Rasmus Lerdorf loi scriptejä, mitä kutsui nimellä "Personal Home Page Tools", joita hän käytti niitä muun muassa omilla kotisivuillaan. Tuolloin Personal Home Page Toolsia kutsuttiin usein nimellä PHP Tools. Ajan kanssa Rasmus Lerdorf lisäsi toiminnollisuuksia PHP Tools scripteihin. Näitä olivat esimerkiksi tietokannan integrointimahdollisuus ja web-sovelluskehitys, jolla voitiin tehdä yksinkertaisia dynaamisia verkkosivuja. (The PHP Group n.d.)

Vuonna 1995 Rasmus Lerdorf julkaisi PHP Toolsin lähdekoodin avoimesti maailmalle. Näin muut ohjelmistokehittäjät ympäri maailmaa pääsivät käyttämään teknologiaa omiin tarpeisiinsa, sekä kehittämään ja korjaamaan PHP Toolsista löytyneitä ongelmia. PHP on kirjoitettu muutamaa otteeseen uudestaan sen alku vuosina. (The PHP Group n.d.)

Vuonna 1997 julkaistiin PHP/FI 2.0, jonka yhtenä suurimpana vahvuutena pidettiin sen helppoa yhdistämistä HTML-koodin kanssa. Tuolloin PHP oli jo saavuttanut korkean suosion ja sen kautta myös korkean käyttäjämäärän. Vuonna 1998 julkaistu PHP 3.0 onkin ensimmäinen versio, joka muistuttaa nykypäiväistä PHP:ta ja käyttäjiä oli tuolloin arviolta jo yli 50 000. (The PHP Group n.d.)

PHP:n yhtenä suurimpana läpimurtona voidaan pitää vuotta 2000, kun PHP 4.0 julkaistiin ja sen suosio yritystasolla kasvoi huimasti. Yritysten ottaessa käyttöön PHP:ta sen käyttäjämäärät nousivat yli 3,6 miljoonaan käyttäjään. PHP 4.0 toi paljon uusia ominaisuuksia ja parannuksia vanhaan versioon. Uusia ominaisuuksia tuli muun muassa natiivituki Java-ohjelmointikielelle, natiivisessioden (HTTP) hallintaan sekä parempi resurssien hallinta ja tuki olio-ohjelmoinnille. Tällä hetkellä uusin saatavilla oleva versio PHP:sta on 7.1. (The PHP Group n.d.)

## 2.1.2 Syntaksi

PHP:lla tehdyt sivut ovat käytännössä XHTML-sivuja, joiden tiedosto päätteinä käytetään .php-loppua. PHP-sivun ohjelmointikoodi kirjoitetaan `<?php *Koodi* ?>` elementtien sisälle. Myös pelkkä `<? *Koodi* ?>` toimii. PHP:n syntaksista löytyy myös yhtäläisyyksiä Java-ohjelmointikieleen. Esimerkiksi lauseet päättyvät puolipisteeseen, lohkomerkinnät merkataan aaltosulkeilla ja rivien kommentointi tapahtuu samalla tavalla `"/"-` ja `"/* */"-` merkkejä käyttäen. (Kollanus n.d.)

```
// $foo = "this line is commented out";  
  
/*$foo = "this line is commented out";  
$bar = "also this line is commented out"; */
```

Kuva 1. Esimerkki rivien kommentoinnista PHP:ssa.

PHP:n omiin syntaksiominaisuuksiin kuuluu muuttujien muodostaminen siten, että `$`-symboli on ensimmäisenä merkinä eikä niille tarvitse erikseen määritellä tieto-tyyppiä kuten esimerkiksi string, integer tai boolean. Muuttujia nimitessään voi myös käyttää skandinaavisia kirjaimia, kuten "ä" ja "ö". Muuttujissa myös kirjainkoot otetaan huomioon sen nimeämisessä. Esimerkiksi seuraavat muuttujat ovat kaksi eri muuttujaa:

```
$foo;  
  
$Foo;
```

Kuva 2. Esimerkki PHP muuttujista.

PHP:n funktiot luodaan ilman, että niihin määritellään vastaanotettavalle parametrille tai palautusarvolle tietotyyppiä. Funktioiden sijainnilla koodissa ei myöskään ole merkitystä siihen, missä sitä voidaan käyttää. Esimerkki PHP funktiosta:

```
function Example($foo, $bar){  
    $result = $foo + $bar;  
    return $result;  
}
```

Kuva 3. Esimerkki PHP funktiosta.

Funktiot eivät myöskään ole riippuvaisia kirjainkoosta, jonka perusteella seuraavat esimerkit viittaavat samaan funktioon:



```
Example($foo, $bar);
example($foo, $bar);
```

Kuva 4. Esimerkki PHP funktioiden yhtäläisyydestä.

PHP funktion parametreille voidaan määritellä oletus arvot, joita funktio käyttää, jos parametriin ei lähetetä omaa arvoa. Esimerkki:

```
function Example($foo, $bar = 2){
    $result = $foo + $bar;
    return $result;
}
```

Kuva 5. Esimerkki PHP funktion parametreista.

### 2.1.3 Tietokantatuki

PHP:n valmis tuki tietokantoihin tekee siitä suositun työkalun web-ohjelmistokehityksessä. PHP tukee tällä hetkellä seuraavia tietokantoja: (Kollanus n.d.)

- Adabas D
- dBase
- Empress
- FilePro(read-only)
- Hyperwave
- IBM DB2
- Informix
- Ingres
- InterBase
- FrontBase
- mSQL
- Direct MS-SQL
- MySQL
- ODBC
- Oracle (OCI7 and OCI8)
- Ovrinos
- PostgreSQL
- Solid
- Sybase
- Velocis
- Unix dbm

PHP:lla pystytään helposti ottamaan yhteys tietokantaan, hakemaan sieltä dataa sekä muokkaamaan ja tallentamaan dataa.

## 2.2 Tietokanta: PostgreSQL

PostgreSQL on avoimeen lähdekoodiin perustuva olio- relaatiotietokannan hallintajärjestelmä, jota käytetään projektin tietokantojen hallintaan, ylläpitoon ja niiden luomiseen. PostgreSQL tunnetaan hyvistä ominaisuuksistaan ja sitä pidetään yleisesti luotettavana tietokantana. Se myös pystyy suoriutumaan suurestakin käyttökuormituksesta, jos käytössä on useita palvelimia. (PostgreSQL 2014.)

### 2.2.1 Tausta

PostgreSQL on saanut alkunsa vuonna 1986 POSTGRES-nimisestä projektista, jota johti professori Michael Stonebraker. Projektia rahoittivat muun muassa Defense Advanced Research Projects Agency (DARPA), the Army Research Office (ARO) ja the National Science Foundation (NSF). Ideana oli luoda järjestelmä, jolla pystyttiin vastaamaan senaikaisten tietokantojen yleisiin ongelmiin. Vuonna 1988 POSTGRES-projektista luotiin ensimmäinen toimiva prototyyppi ja vuonna 1989 julkaistiin ensimmäinen testiversio pienelle käyttäjäkunnalle. Tämän jälkeen julkaistiin vielä kolme versiota lisää aina vuoteen 1993 asti. Projekti lopetettiin, kun se saavutti niin suuren käyttäjämäärän, ettei enää pystytty vastaamaan käyttäjien toivomiin tuki- ja ominaisuuspyyntöihin. (PostgreSQL 2014.)

Vuonna 1994 Andrew Yu ja Jolly Chen muokkasivat POSTGRES-projektin koodia muun muassa lisäämällä siihen SQL:lle tuen, jolla korvattiin edellinen kyselykieli PostQuel. He julkaisivat sen nimellä Postgres95. Vuonna 1996 uuden kyselykielen inspiroimana POSTgres95 nimeksi muutettiin PostgreSQL. PostgreSQL perustuu vielä tänäkin päivänä avoimeen lähdekoodiin ja siitä julkaistaan uusia versioita ajoittain. Tällä hetkellä uusin versio on PostgreSQL 11 Beta 2. (PostgreSQL 2018.)

### 2.2.2 Ominaisuudet

PostgreSQL:llä voidaan kirjoittaa palvelimessa ajettavaa ohjelmakoodia. Näitä koodeja voidaan kirjoittaa muun muassa SQL-kielellä tai pgSQL-kielellä. Ne sisältävät valmiita funktioita, joita hyödynnetään myös tässä projektissa. Palvelinpuolen ohjelmointiin voidaan käyttää myös muita kieliä postgresSQL:lää käytettäessä, kuten C/C++-kieliä, Javaa tai PHP-kieltä. (PostgreSQL 2018.)

PostgreSQL:ssä voidaan käyttää joko sisäänrakennettuja indeksejä tai käyttäjän itse määrittämiä indeksejä. Indeksejä pystyy myös selaamaan taaksepäin ilman erillisiä funktiolausekkeita. Myös tuki osittaisindekseille sekä bittikarttaindeksille löytyy valmiina. (PostgreSQL 2018.)

PostgreSQL:n kyky käsitellä useampaa erilaista dataa asettaa sen muiden avoimeen lähdekoodiin perustuvien tietokantojen yläpuolelle. Mikä myös

mahdollistaa sen, että se soveltuu moneen käyttötarkoitukseen, kun kyse on tietokannoista. (PostgreSQL 2018.)

## 2.3 JavaScript

JavaScript on ohjelmointikieli, jota käytetään usein dynaamisissa web-ohjelmointitoteutuksissa front-endin tasolla. JavaScriptillä voidaan muun muassa luoda funktioita ja toiminnallisuuksia käyttöliittymälle. Verkkokaupan asiakaskonsolin käyttöliittymän toiminnoiden toteutukseen käytettiin JavaScriptiä, johon on lisätty kirjastoja sekä Backbone.js-framework.

### 2.3.1 Tausta

World Wide Web eli WWW oli 90-luvun alussa suhteellisen uusi ilmiö ihmisten jokapäiväisessä arjessa. Netscape Communication Corporation niminen yhtiö oli tuolloin kovasti esillä WWW:hen liittyvien projektien ja tuotteiden kehityksessä. Netscape Communication Corporation yhtiön perustaja Marc Anderssenilla oli visio, että webiin tarvittaisiin lisää dynamiikkaa muun muassa animaatioilla, käyttäjäinteraktiolla ja yleisellä automaatiikalla. Tähän tarvittiin scriptauskieli, joka pystyi vaikuttamaan ja kommunikimaan sen aikaisen DOM:in kanssa. Tuolloin HTML-kieli oli vielä hyvin alkeellinen ja sitä pystyi helposti kuka vaan opettelemaan ja kirjoittamaan. Samaa ominaisuutta haluttiin myös JavaScriptille, koska ajateltiin, että sillä saataisiin nopeasti webin staattinen ilme muuttumaan dynaamiseksi. (Peyrott 2017.)

Vuonna 1995 ensimmäinen prototyyppi JavaScriptistä integroitiin Netscapen web-selaimeen. Tuolloin sitä kutsuttiin Mocha-nimellä, mutta lyhyen ajan päästä se nimettiin uudelleen LiveScriptiksi markkinoinnin takia. Vuoden 1995 joulukuussa nimi muutettiin jälleen. Tällöin päädyttiin nykyiseen JavaScript nimeen. Tuolloin Java oli suuressa suosiossa ammattilaisten keskuudessa ja JavaScriptille haluttiin luoda samanlaista imagoa. (Peyrott 2017.)

### 2.3.2 Syntaksi

JavaScriptin syntaksi muistuttaa hyvin paljon Javaa. Muun muassa lauseet päätetään puolipisteellä ja rivien kommentointi tapahtuu samalla tavalla. Sitä voidaan kirjoittaa HTML-kielen sekaan käyttämällä `<script type="text/javascript" language="JavaScript"> *Koodi* </script>` elementtejä. Joitakin yksinkertaisia JavaScript toiminnollisuuksia voidaan käyttää myös suoraan HTML-koodissa ilman erillistä `<script>`-elementtiä. Ne toimivat kaikissa uudemmissa selaimissa, missä on JavaScript tuki. Esimerkki valmiista toiminnollisuudesta: (W3Schools n.d.)

```
<li onclick="alert('Foo')">"Bar"</li>
```

Kuva 6. Esimerkki HTML:n JavaScript valmiudesta

JavaScriptissä muuttujille ei erikseen tarvitse määrittää tieto-tyyppiä, vaan se määritellään muuttujan saadun arvon mukaan: (W3Schools n.d.)

```
var foo = 1 //integer
var Foo = "Bar" //string
var bar = true //boolean
var Bar = {x: "y"} //Object
var fooBar = ["a", "b", "c"] //Array
```

Kuva 7. Erilaisia JavaScript muuttujien arvoja.

Jos muuttujalla ei ole arvoa, lukee JavaScripti sen "undefined" arvoisena. JavaScriptin muuttujissa pätee myös kirjainkoot, joten samannimiset muuttujat eri kirjainkoolla ovat todellisuudessa eri muuttujia.

JavaScriptiä kirjoittaessa käytetään yleensä käytäntöä "lower camel case", jossa useampi sanaiset muuttujien ja funktioiden nimet kirjoitetaan yhteen. "Lower camel case" tyyliässä funktioiden ja muuttujien ensimmäinen kirjain kirjoitetaan pienellä ja loppujen sanojen ensimmäinen kirjain kirjoitetaan isolla. Esimerkkejä "lower camel case" tyylistä: (W3Schools n.d.)

```
function exampleFunction(){
    return;
}

var exampleVariable;
```

Kuva 8. Esimerkki "lower camel case" tyylistä.

Mallipohjan k JavaScriptin muuttujat voidaan myös määrittellä const tai let alkuisiksi. Const alkuisten muuttujien arvo ei voi muuttua missään kohtaa koodin suoritumista ja let alkuisten muuttujien arvo pysyy samana, paitsi funktioissa, joissa sitä halutaan mahdollisesti muokata. Esimerkki let-muuttujan toiminnosta: (Mozilla n.d.)

```
let x = 1;

if(x === 1){
    x = x + 1;
    console.log(x); //printtaa x:n arvon 2
}

console.log(x); //printtaa x:n arvon 1
```

Kuva 9. Esimerkki let-muuttujasta.

### 2.3.3 jQuery

Yksi JavaScriptin käytetyimmistä kirjastoista on jQuery. Se sisältää valmiita funktioita, joiden avulla JavaScript koodissa pystytään helposti viittamaan HTML-elementteihin, tekemään animaatioita, hallitsemaan eventtejä sekä tekemään ajax-kutsuja. jQuery kirjastoa käytettiin konsolin käyttöliittymän toteutuksessa. Esimerkkejä jQuery toiminnollisuuksista: (jQuery n.d.)

```
$("#p.text").html("New text");
```

Kuva 10. Esimerkki jQuery:n html() funktiosta.

Aiemman kuvan funktio muuttaa kaikkien <p class="text"> sisällön "New text"-tekstiksi.

```
$( "#button button" ).on( "click", function( event ) {  
    someFunction();  
});
```

Kuva 11. Esimerkki jQuery:n on() funktiosta.

Aiemmassa kuvassa id #button nappia painaessa kutsutaan funktiota someFunction().

Yksinkertainen ajax-kutsu jQueryllä:

```
$.ajax({  
    url: "/api/getTargetFile",  
    data: {  
        parameter: "foo"  
    },  
    success: function( result ) {  
        console.log("success", result);  
    }  
});
```

Kuva 12. Esimerkki jQuery:n ajax() funktiosta.

## 2.4 Backbone.js

Backbone.js on JavaScript-sovelluskehys, jota käytetään paljon projektin toteutuksessa. Backbone.js käyttää MVP-arkkitehtuuria (Model-view-presenter), jonka avulla API:sta saatava data voidaan käsitellä collection- ja model-objekteina käyttöliittymässä. Koska Backbone.js pohjautuu REST-ohjelmointirajapintaan, se mahdollistaa monien valmiiden toimintojen tuomisen projektiin. Backbone.js vaatii underscore.js kirjaston käytön projektissa jQueryn lisäksi. (Backbone.js n.d.)

### 2.4.1 Model, collection & view

Backbone.js mahdollistaa palvelimelta tulevan datan tallennuksen model-objekteihin. Se myös tallentaa dataa palvelimelle. Kaikki logiikka, mikä tehdään datan käsittelyyn, pysyy myös erillään käyttöliittymästä. Käyttöliittymässä dataa esitetään view-toiminnon avulla, joka kuuntelee model-objekteja ja renderöi käyttöliittymää sen mukaan, miten dataa muutetaan. View myös tuo uutta dataa model-objekteille.

Collection-objektit pitävät sisällään useita model-objekteja, joita hallinnoidaan samaan aikaan. Näillä toimintoperiaatteilla tullaan rakentamaan projektin käyttöliittymään editorit, joiden avulla näytetään haluttua dataa ja käyttöliittymän käyttäjä pääsee muokkaamaan omia tietojaan. (Backbone.js n.d.)

### 2.4.2 Event-tapahtuma ja ListenTo()-funktio

Backbone.js mahdollistaa myös event-tapahtumien luomisen model- ja collection-objekteille sekä ListenTo() funktion käytön. Näillä toiminnollisuuksilla voidaan esimerkiksi objektia muuttamalla kutsua siihen tarkoitukseen liittyvää funktiota. Event-tapahtumat ja ListenTo()-funktio tulee olemaan suuressa roolissa projektin toteutuksessa. Esimerkki objektin event-tapahtumasta: (Backbone.js n.d.)

```
var object = {};

_.extend(object, Backbone.Events);

object.on("sync", function() {
  console.log("triggered");
});
```

Kuva 13. Esimerkki Backbone.js:n event hallinnasta.

Aiemmassa kuvassa on esimerkki jossa aina, kun objektiin viitataan, tulee selaimen konsoliin "triggered" printtaus.

## 2.5 HTML

HTML eli HyperText Markup Language on web-ohjelmistokehityksessä käytetty kuvauskieli, jolla pystytään kirjoittamaan runko web-sivulle. Se on avoimesti standardoitu ja se on käytetyin kuvauskieli web-sivujen kehittämisessä.

Tässä projektissa käytettiin underscore.js kirjastoa, jolla voidaan luoda HTML-sivuja koodin kautta määrittämällä underscore.js:n `template()`-funktion parametriksi HTML-koodia, joka sitten renderöidään web-sivuksi.

### 2.5.1 Tausta

HTML-kielen kehitys alkoi 1990 CERN konsernin alaisuudessa, jonka jälkeen sen kehitys siirtyi IETF:lle. Kun World Wide Web Consortiumin (W3C) perustettiin, siirtyi HTML-kielen kehitys sen vastuulle. HTML-standardi on vieläkin W3C:n hallinnoima. (W3C 2017.)

HTML:ää kehitettiin aina versioon 4.01 asti aktiivisesti. Sitten W3C halusi alkaa kehittämään sille XML pohjaista korviketta nimeltä XHTML. XHTML ei kuitenkaan koskaan pystynyt korvaamaan HTML:ää webissä ja W3C jatkoi HTML:n kehittämistä vuonna 2007. Vuonna 2014 virallisesti julkaistiin HTML5, joka on uusin versio standardista. (W3C 2017.)

### 2.5.2 Syntaksi

HTML-kielen syntaksi koostuu HTML-elementeistä, joilla voidaan määrittää muun muassa tekstin sijaintia ja muotoa web-sivulla. HTML5-pohjainen web-sivu tarvitsee myös pakolliset elementit toimiakseen. Esimerkki minimaalisesta HTML5 web-sivusta: (W3C 2017.)

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>sivun title</title>
  </head>
  <body>
    <!-- sivun sisältö -->
  </body>
</html>

```

Kuva 14. Yksinkertainen HTML rakenne.

Esimerkistä huomaa myös, että HTML-elementit alkavat `<*elementin nimi*>` merkinnästä ja päättyvät `</*elementin nimi*>` merkintään. Kommentointi tapahtuu käyttäen `<!-- *kommentti* -->` merkintää. Elementeille pystytään määrittelemään erilaisia attribuutteja, joilla voidaan merkitä, määritellä tai tehostaa elementtejä. Esimerkkejä muutamasta yleisestä attribuutista ja miten niitä voidaan käyttää: (W3C 2017.)

```

<p class="foo"></p>

```

Kuva 15. HTML:n class-attribuutti.

Aiemmassa kuvassa merkitään p-elementille class-attribuutin arvolla foo, jolloin voidaan viitata siihen käyttämällä sitä esimerkiksi jQuery-koodissa. Usealla HTML-elementillä voi olla sama class-attribuutti määriteltynä.

```
<h1 id="bar"></h1>
```

Kuva 16. HTML:n id-attribuutti.

Aiemmassa kuvassa merkitään h1-elementille id-attribuutin arvolla bar. Toiminnollisuus on sama kuin class-attribuutilla, mutta vain yhdellä elementillä voi olla samanarvoinen id attribuutti määriteltynä.

```
<p style="color:red;">foo</p>
```

Kuva 17. HTML:n style-attribuutti.

Aiemmassa kuvassa on esimerkki miten style-attribuutilla voidaan vaikuttaa elementin ulkoasuun. Esimerkissä attribuutti muuttaa p-elementin "foo" tekstin punaiseksi.

```
<a href="www.hamk.com">hamk</a>
```

Kuva 18. HTML:n href-attribuutti.

Aiemmassa kuvassa href-attribuutilla saadaan tehtyä hyperlinkki haluttuun web-sivuun. Esimerkissä linkkaus veisi HAMKin kotisivuille, kun "hamk" tekstiä painettaisiin kursorilla.

## 2.6 CSS/SCSS

CSS eli Cascadian Styling Sheet on pääsääntöinen työkalu, kun luodaan ulkoasua web-sivulle. Sillä voidaan määrittää esimerkiksi fontin kokoa, väriä ja tyyliä. Sillä voidaan myös hallinnoida elementtien kokoja ja sijoitusta web-sivulla. Vaikkakin projektissa ei ollut tarkoitus kiinnittää huomiota erityisesti ulkoasuun, käytetään CSS:ää sijoittamaan elementtejä paikalleen sekä luomaan ensimmäinen versio teemasta konsolille.

SASS on CSS:n esikäsittelijä, joka tuo siihen lisää ominaisuuksia kuten kyvyn käsitellä funktioita. SCSS on syntaksimuoto SASS:lle, jota käytetään projektissa teeman luontiin ja hallintaan. SCSS tiedostot käännetään normaalisti CSS koodiksi, kun web-sivu viedään verkkoympäristöön.



### 2.6.1 Syntaksi

CSS:n syntaksi on hyvin yksinkertaista. Tiedostot eivät tarvitse ylimääräisiä merkkejä indikoimaan tiedoston tyyppin luonnetta, vaan pelkkä .css-pääte tiedostossa riittää. Kun halutaan vaikuttaa web-sivun elementtien teemaan, niihin pitää viitata CSS tiedostossa. Viittaaminen tapahtuu muun muassa nimeämällä elementti, sen id-attribuutin arvo tai sen class-attribuutin arvo. Esimerkkejä näihin kolmeen tapaukseen CSS-tiedostossa viitatessa: (W3Schools n.d.)

```
p {  
    color: red;  
}
```

Kuva 19. Esimerkki CSS syntaksista.

Aiemmassa kuvassa syntaksi muuttaa kaikki HTML-tiedostosta löytyvien p-elementtien tekstit punaiseksi.

```
.foo {  
    color: blue;  
}
```

Kuva 20. Esimerkki CSS syntaksista.

Aiemmassa kuvassa on esimerkki kun halutaan viitata class-attribuuttiin, merkitään sitä pisteellä, joka on ennen attribuutin arvoa. Tässä tapauksessa kaikkien elementtien, joiden class-attribuutin arvo on "foo", tekstit muuttuvat sinisen värisiksi.

```
#bar {  
    color: green;  
}
```

Kuva 21. Esimerkki CSS syntaksista.

Aiemmassa kuvassa on esimerkki kun halutaan viitata id-attribuuttiin, merkitään sitä #-symbolilla joka on ennen attribuutin arvoa. Tässä tapauksessa vain elementillä, jonka id-attribuutin arvo on "bar", on vihreä teksti.

Aiemmista esimerkeistä huomataan, että kun viittaus haluttuun kohteeseen on merkitty, suljetaan teemaan liittyvä syntaksi aaltosulkeilla. Itse teemaan vaikuttava syntaksi koostuu halutusta vaikutuksesta, kuten "color" ja sille annetusta arvosta kuten "red". Nämä erotetaan toisistaan kaksoispisteellä ja lause päätetään puolipisteeseen.

## 2.6.2 SCSS

SCSS mahdollistaa monipuolisen teeman kehityksen tehokkaasti. Sillä voidaan määrittää muun muassa muuttujia ja funktioita, joiden avulla teeman kehitys käy nopeammin. Esimerkki muuttujien hyödyntämisessä SCSS tiedostossa: (Sass-lang 2018.)

```
$main-color: red;
$main-font: 20px;

#main-title {
  color: darken($main-color, 20%);
  font-size: $main-font * 2;
}

.content {
  color: $main-color;
  font-size: $main-font;
}
```

Kuva 22. Esimerkki SCSS syntaksista.

Aiemman kuvan esimerkissä on määritelty muuttujille `$main-color` ja `$main-font` kiinteät arvot, joita käytetään, kun määritellään `#main-title` ID:lle ja `.content` luokalle teema muutoksia. `#main-title`:n `color`-attribuuttiin arvoon on käytetty SASS `darken()`-funktioita, joka tässä tapauksessa tummentaa annettua väriä 20 %. `#main-title` fontin kokoa on myös säädetty kertomalla annettu arvo kahdella, jolloin sen fontti on kaksi kertaa suurempi, kuin `.content` fontti. Jos halutaan muuttaa koko sivun tekstin fontin kokoa tai väriä, tässä tapauksessa se voidaan tehdä helposti muuttamalla vain muuttujien arvoa. Näiden ominaisuuksien avulla suurten teemakokonaisuuksien ylläpitäminen helpottuu huomattavasti, kun käytetyimmät arvot on tallennettu muuttujiin, joita voidaan tarpeen tullen muuttaa. Samalla sivun yhtenäisyys säilyy paremmin uusien muutosten myötä.

## 2.7 Muita mahdollisia teknologioita

Vaikka teknologiat projektiin oli ennalta määrätty verkkokaupan aiemman toteutuksen mukaisesti, olisi projektin toteutuksessa voitu käyttää vaihtoehtoisesti muita web-sovelluskehitykseen käytettyjä yleisiä teknologioita. Seuraavaksi käyn läpi muutaman yleisimmän web-kehitysteknologian ja vertaan niitä toteutuksessa käytettyihin teknologioihin. On hyvä huomioida, että jotkin seuraavista teknologioista olisi vaatinut myös verkkokaupaan erilaisen toteutuksen.

### 2.7.1 Angular.js

Käyttöliittymässä käytetyn backbone.js:n sijaan voitaisiin käyttää angular.js JavaScript-ohjelmistokehystä. Angular.js käyttää MVC-arkkitehtuuria (sanoista model-view-controller), joka mahdollistaa muun muassa käyttöliittymän erottamisen muista sovellukseen tarvittavista tiedoista. Angular.js:n arkkitehtuuri rakenne on lähes samanlainen kuin backbone.js:n MVP-malli, jonka takia se soveltuisi hyvin ajamaan samaa tarkoitusta data-objektien hallinnassa. Angular.js ei myöskään lisäksi tarvitse erillisiä kirjastoja toimiakseen, toisinkuin backbone.js tarvitsee underscore.js ja jQuery kirjastot jotta sitä voidaan käyttää. Angular.js on myös ollut pidempään olemassa, joten siihen löytyy kattavampi dokumentaatio mikä helpottaa ohjelmistokehityksen käyttöä ja soveltamista. Angular.js ei kuitenkaan ole yhtä kevyt kuin backbone.js. (Google n.d.)

### 2.7.2 React.js

React.js on JavaScript kirjasto, joka tuo jokseenkin samanlaisen view-näkymä ja mahdollistaa datan renderöinnin HTML muotoon käyttöliittymälle. React.js pystyy myös luomaan virtuaalisia DOM kerroksia, joilla voidaan muun muassa päivittää käyttöliittymän tekstikenttiä dynaamisesti niiden muokkausten jälkeen. React.js:n käyttäminen projektissa olisi vaatinut todennäköisesti muidenkin kirjastojen käyttämistä, sillä se yksinään ei yllä samalle tasolle ominaisuuksissa kuin Backbone.js. React.js vaatii myös paljon muistia, ja voi olla raskas pyörittää laajassa verkkosivussa, kun taas Backbone.js on kevyt ja nopea. (Facebook inc. n.d.)

### 2.7.3 MySQL

MySQL on yleinen web-kehityksessä käytetty tietokanta teknologia, joka sisältää paljon samoja ominaisuuksia kuin PostgreSQL. MySQL:n suosiota kasvattaa muun muassa sen helppo käyttöön otto ja käyttäminen. Sitä tukee kattava dokumentaatio ja hyvät perus ominaisuudet, kuten kattava SQL-funktio tuki ja sisään rakennetut turvallisuus ominaisuudet. Tätä projektia ajatellen MySQL on kuitenkin heikompi valinta, kuin PostgreSQL. MySQL:n luotettavuus ei yllä samalle tasolle kuin PostgreSQL:llä, sekä se toimii paremmin datan näyttämiseen tietokannoista, kuin datan tallentamiseen ja muokkaamiseen. PostgreSQL on myös täysin ilmainen, toisin kuin MySQL:llä on maksullisia versioita, jota tarjoavat enemmän ominaisuuksia sen mukaan mitä enemmän ne maksavat. PostgreSQL tarjoaa myös tuen ohjelmointikielille, mitä MySQL ei tarjoa. (Oracle Corporation n.d.)

### 2.7.4 MariaDB

MariaDB on relaatiotietokantajärjestelmä, joka pohjautuu MySQL:ään. Se perustuu avoimeen lähdekoodiin, ja on laajalti käytössä suurilla yrityksillä,

kuten PostgreSQL:llä. MariaDB:ssä on hyvä yhteensopivuus MySQL:llän kanssa, joka mahdollistaa laajemmat integraatiot eri järjestelmissä. MariaDB tukee myös enemmän ohjelmointikieliä, kuin PostgreSQL, sekä se on saatavilla eri käyttöjärjestelmille, kuten Windowsille ja Linuxille. Ominaisuuksiltaan ja käyttötarkoitukseltaan MariaDB ei hirveästi eroa PostgreSQL:stä, joten se olisi ollut varteenotettava vaihtoehto projektiin, jos kaikki tietokannat olisi sillä rakennettu. Vaikka MariaDB on nuorempi teknologia, kuin PostgreSQL, on siihen saatavilla hyvin dokumentaatiota ja ohjeita. (MariaDB n.d.)

### 2.7.5 Python

Python on suosittu ohjelmointikieli, jota käytetään usein web-sovellushityksessä back-end tasolla. Python on laajasti dokumentoitu, ja siihen saatavilla muutamia sovelluskehysä tukemaan eri toimintoja, vaikkakin vähemmän kuin PHP:lle. Pythonin ja PHP:n väliset erot tulevat esiin suorituksessa, sekä kielten syntakseista. Moni pitää Pythonin syntaksia selkeämpänä ja helpompina kehittää, kun taas joissain tapauksissa PHP:n parempi nopeus suorittaa annettuja tehtäviä on syy sen käyttöön. Pythonia olisi voitu käyttää projektissa, mutta se olisi vaatinut suurempia muutoksia verkkokaupan jo olemassa olevaan rajapintaan. Vaikka Pythonin käyttäminen olisi ollut mahdollista, ei se olisi ollut tarpeellista rajapintojen yksinkertaisuuden takia. Pythonin käyttöä suositellaan suurille ja monimutkaisille ohjelmille, jotka tarvitsevat paljon laskenta- ja suoritustehoa. (Python software foundation n.d.)

### 2.7.6 Java

Java on suosittu ohjelmointikieli, jota käytetään myös web-ohjelmoinnissa back-end tasolla. Sen suurimpiin etuihin kuuluu sen laaja toimivuus alue, missä sitä voidaan käyttää. Java on suuressa suosiossa myös mobiiliohjelmistojen ja Android-sovellusten rakennuksessa. Javaa käytetään yhdessä PHP:n kanssa verkkokaupan back-end toteutuksessa, ja siihen ei tarvittu muutoksia tähän projektiin liittyen. Javan käyttäminen PHP:n sijasta olisi vaatinut myös suurempia muutoksia verkkokaupan olemassa olevaan PHP rajapintaan, vaikkakin kielet eivät eroa toisistaan hirveästi syntaksissa ja suorituksessa. Suurin ero tulee muista teknologioista palvelinpuolella, joiden ympärillä kielet toimisivat. (Marsh n.d.)

### 3 PROJEKTISSA KÄYTETYT TYÖKALUT

#### 3.1 Sublime Text

Projektissa käytän Sublime Text-tekstieditoria konsolin koodin kirjoittamiseen. Sublime Text-editorissa on natiivituki moneen ohjelmointi- ja merkintäkieleen. Siihen on saatavilla myös useita lisäosia kuten syntaksikorostajia ja versiohallintatyökaluja. Editorissa on myös lokaalimuisti, johon se tallentaa avatut tiedostot ja niihin tehdyt muutokset. (López-Anglada 2013.)

Projektin aikana käytössä oli muun muassa seuraavat lisäosat:

- BrackerHighlighter
- GitGutter
- Git
- GitStatusBar
- Gutter Color
- HTML Underscore Syntax
- Package Control
- Sass
- SublimeLinter
- SublimeLinter – PHP

Lisäosien lisäksi käytössä oli Sublime Text-editorin natiivit syntaksikorostukset seuraaville kielille:

- HTML
- CSS
- JavaScript
- jQuery
- PHP
- JSON
- SQL

#### 3.2 Eclipse

Louhi Networksin oman rajapinnan kehittämiseen käytetään Eclipse-sovelluskehitystä, jossa käytetään SVN:ää versiohallinta työkaluna. Eclipse on suosittu ilmainen sovelluskehitin, joka perustuu avoimeen lähdekoodiin. Se tukee natiivisti useita eri ohjelmointikieliä kuten PHP, Java, C ja C++. Louhi Networksin rajapinta on toteutettu PHP-kielillä, joten Eclipse antaa siihen valmiudet syntaksikorostukseen ja virheiden näyttämiseen.



### 3.3.2 Apache Subversion – SVN

SVN on Apachen julkaisema avoimeen lähdekoodiin perustuva versionhallintatyökalu. Projektissa SVN on integroitu Eclipse-sovelluskehittimeen, jolla kehitetään Louhi Networksin omaa rajapintaa. SVN on ollut versionhallintatyökaluna kyseisellä rajapinnalla jo ennen projektin alkamista. (Apache n.d.)

### 3.4 PgAdmin

PgAdmin on PostgreSQL:lle tarkoitettu kehitys- ja ylläpito-ohjelma, joka perustuu avoimeen lähdekoodiin. PgAdmin:lla hallinnoitiin myös projektin aikana kaupan ja Louhi Networksin rajapinnan tietokantoja. Projektissa oli käytössä työpöytäsovellus pgAdmin III ja selaimessa toimiva phpPgAdmin 4.2.

### 3.5 Kehitysympäristö

Projektin kehitysympäristönä käytettiin Oracle VM Virtualboxilla ja vagrantilla luotua virtuaalikoneita. Projektin tarvittava lähdekoodi oli kopioitu yhteisen kehitysympäristön Git-branchistä, jota ylläpidetään sille tarkoitettulla palvelimella.

#### 3.5.1 Oracle – Virtualbox

Virtualbox on Oraclen virtualisointiohjelma, jolla voidaan muun muassa luoda virtuaalikoneita lokaaliin verkkoympäristöön. Se on ilmainen ja se perustuu avoimeen lähdekoodiin, mitä muun muassa käyttäjät päivittelevät ja ylläpitävät. Virtualbox toimii kaikilla käyttöjärjestelmillä ja se tuo muun muassa mahdollisuuden käyttää muita käyttöjärjestelmiä samankaltaisesti tietokoneella. Projektissa virtualboxilla luodulla virtuaalikoneella käytettiin Linux käyttöjärjestelmää. (Oracle n.d.)

#### 3.5.2 Vagrant

Vagrant on HashiCorpin julkaisema työkalu, joka on suunniteltu helpottamaan työympäristöjen luontia virtuaalikoneilla. Se tuo mukaan automaatiota, jolla kehitysympäristö saadaan luotua ennalta määrättyjen asetusten mukaan. Projektissa vagrantin avulla on esimerkiksi kopioitu yhteisen kehitysympäristön Git-branchi lokaalille virtuaalikoneelle. (HashiCorp n.d.)

### 3.6 Muita mahdollisia työkaluja

Projektin toteutukseen olisi voitu käyttää muitakin työkaluja, kuin mitä oli valittu. Jotkin projektissa käytetyistä työkaluista oli valittu, koska olin tot-

tunut työskentelemään niiden kanssa sekä oppinut käyttämään niihin saatavia lisäosia. Jotkin työkalut taas oli ennalta määrätty käytettäväksi versiohallinta syistä, tai koska jotkin teknologiat vaativat niiden kaltaisten ohjelmien käyttöä projektissa. Käyn läpi seuraavaksi muutamia tekstieditointi työkaluja, sekä joidenkin teknologioiden kehitykseen tarkoitettuja työkaluja, ja vertaan niitä projektissa käytettyihin.

### 3.6.1 Notepad++

Notepad++ on avoimeen lähdekoodiin perustuva tekstieditori, joka on suosittu ilmainen koodaus työkalu. Projektissa sitä olisi voitu käyttää Sublime Text-editorin sijasta. Siinä on sisäänrakennettu tuki suurelle määrällä eri ohjelmointi- ja skriptaus-kielille, mutta lisäosien saatavuudessa se häviää Sublime Text-editorille. (Notepad n.d.)

### 3.6.2 GNU Emacs

GNU Emacs on tekstieditori, jolla pystytään kehittämään suurinta osaa ohjelmointikielistä. GNU Emacsia käyttäjä pystyy muokkaamaan juuri sellaiseksi kuin haluaa, mikä on sen yksi parhaista ominaisuuksista. Se tarjoaa myös valmiiksi laajan tuen ja syntaksi korostuksen erilaisille ohjelmointikielille, ja unicode – merkeille. GNU Emacs olisi ollut varteen otettava vaihtoehto Sublime Text-editorin tilalle projektin toteutuksessa. (Free Software Foundation n.d.)

### 3.6.3 Navicat for Postgre SQL

Navicatiltä on saatavilla PostgreSQL tietokantojen hallintaan tarkoitettu työkalu. Työkalu tarjoaa graafisen käyttöliittymän tietokantojen ja taulujen hallinnointiin sekä muokkaukseen. Navicat tuo mukanaan myös omia ominaisuuksiaan ohjelmaan, kuten hyvän SQL-tuen ja SSL-tietoturvan. Navicat:n työkalu olisi ollut hyvä vaihtoehto PgAdminin tilalle tietokantojen hallintaan. (Navicat n.d.)

### 3.6.4 Mercurial

Mercurial on hajautettu versionhallintajärjestelmä ohjelmistokehitykseen. Se on alustariippumaton, sekä sopii hyvin kun työskennellään tiimeissä. Mercurialin toimintaperiaatteet vastaavat Git-versionhallintajärjestelmän toimintoja, sekä sitä voidaan käyttää suoraan tietokoneen terminaalista, kuten Gittiä. Jos Git ei olisi ollut valmiiksi käytössä verkkokaupan versionhallinnassa, olisi Mercurial ollut siihen hyvä toinen vaihtoehto. (Mercurial n.d.)



## 4 KONSOLIN SUUNNITTELU

### 4.1 Konsolin tarve ja toimeksianto

Louhi Networks tilasi asiakaskonsolin tämän hetkiseltä työnantajaltani Pilvi Cloud Companyltä. Pilvi oli aiemmin rakentanut Louhi Networksille verkko-kaupan, jonka osaksi asiakaskonsoli tulisi. Asiakaskonsolilla halutaan uudistaa ja parantaa Louhi Networksin asiakkaiden käyttökokemusta Louhi Networksin palveluista. Ajatuksena on myös tuoda kaikki Louhi Networksin tarjoamat palvelut yhden web-sivun alle.

Uusi asiakaskonsoli tulee tulevaisuudessa korvaamaan nykyisen oma.louhi.fi-sivuston, josta tällä hetkellä Louhi Networksin asiakkaat voivat hallinnoida asiakastietojaan, sekä tuotteitaan kuten domaineja ja webhotelleja. Uusi asiakaskonsoli tulee myös pitämään enemmän toimintoja sisällään kuin nykyinen oma.louhi.fi-portaali.

Toimeksianto pitää sisällään asiakaskonsolin pohjan suunnittelun ja rakentamisen sekä seuraavat ominaisuudet, joilla hallinnoidaan asiakastietoja:

- käyttäjätilin ja yhteystietojen näyttäminen ja muokausmahdollisuus
- asiakkaan sopimusten ja tuotteiden listaus ja yksittäisten sopimus- ja tuotetietojen näyttäminen
- asiakkaan laskujen listaus ja yksittäisten laskujen näyttäminen sekä laskunlataus PDF-muodossa.

Konsolin käyttöliittymän teemat ovat yksinkertaisia ja pelkistettyjä ja niitä tullaan parantamaan tulevaisuudessa enemmän kuin niihin liittyvää UI-dokumentaatiota ja suunnitelmat ovat valmiina.

### 4.2 Ohjelmistokehityksen periaatteita

Ohjelmistokehitystä voidaan ajatella jatkuvana opiskeluna. Oppimista tapahtuu aina, kun ohjelmistoa suunnitellaan, rakennetaan ja käytetään. Ohjelmistokehitys ei ole kuitenkaan niin sanottua liukuhihna-tuotantoa vaan jokainen ohjelmisto vaatii sille räätälöidyn toteutuksen, jotta lopputulos olisi laadukas ja toivottu. (Kelly 2008, 1.)

Ohjelmistoala myös muuttuu jatkuvasti. Vanhat teknologiat päivittyvät tai korvataan uudella ja ohjelmistoja päivittävät järjestelmät muuttuvat ja tuovat lisää mahdollisuuksia uusille ohjelmistoille. Tämän takia ohjelmistokehittäjän on pysyttävä muutoksen mukana sopeutumalla uuteen sekä mahdollisesti olla mukana kehittämässä uutta teknologiaa. (Kelly 2008, 2.)

Ennen projektia olin jo tutustunut Louhi Networksin verkkokauppaan muiden tehtävien aikana. Aiempaa oppia oli siis tullut alustasta, johon konsolia lähdettiin rakentamaan. Mutta suunnitteluvaiheessa piti ottaa selvää useasta uudesta asiasta sekä ottaa huomioon, miten uudesta asiakaskonsolistä saadaan käyttöliittymän käyttäjälle helppokäyttöinen ja selkeä.

Ohjelmistokehittäjän luoma koodi tulee olla selkeää, helposti ylläpidettävää ja jatkokehitettävää. Koodin rakenteen on hyvä noudattaa konventiota, jolla aiemmat projektit on toteutettu. Näistä periaatteista sai hyvän suunnan, miten projektin toteutusta lähdettiin suunnittelemaan ja myöhemmin toteuttamaan.

Alexander Dawson määrittelee neljä tärkeää kohtaa, jotka on hyvä ottaa huomioon, kun kehitetään web-sovelluksia: (Dawson 2011, 10.)

- Tunne ympäristösi.
- Suunnittele etukäteen.
- Opi ja mukaudu tilanteisiin.
- Ratkaise ongelmia sitä mukaan, kun niitä ilmenee.

Kun suunnitellaan web-sovellusta tai web-sivua on tärkeää tunnistaa ympäristö, johon sitä kehitetään. Ympäristöstä pitää myös löytää ne tekijät, jotka joko lisäävät tai laskevat projektin näkyvyyttä ja toimia. Käyttöliittymän käyttäjän näkökulma pitää myös ottaa huomioon, kun suunnitellaan toiminnollisuuksia, mitä tarjotaan. Toiminnollisuuksia suunnitellessa on hyvä tiedostaa, mikä on mahdollista ja mikä mahdotonta toteuttaa. (Dawson 2011, 10.)

Hyvä ympäristöntuntemus parantaa projektin suunnittelua. Suunnitteluvaiheessa tietämys ympäristöstä ja käyttäjistä auttaa kartoittamaan eri ratkaisuvaihtoehtoja, joilla projektia voidaan lähteä toteuttamaan. Samalla voidaan ennaltaehkäistä virheitä, joita projektin aikana voi ilmentyä. Kun parhaaksi todettu toteutus tapa on selvitetty, voidaan keskittyä suunnitelman luomiseen. (Dawson 2011, 18.)

Web-sovellusta tai -sivua kehittäessä tulee usein vastaan tilanteita, joissa vanhan teknologian korvaaminen uudella parantaa projektin lopputulosta. Web-teknologioita kehitetään koko ajan lisää ja niiden soveltaminen vaatii jatkuvaa opiskelua. Jos uudempi teknologia parantaa projektin toiminnollisuuksia tai suoritusta, on hyvä miettiä vaihtoehtoja, joilla projektia pysyttäisiin parantamaan kyseisellä teknologialla. (Dawson 2011, 23.)

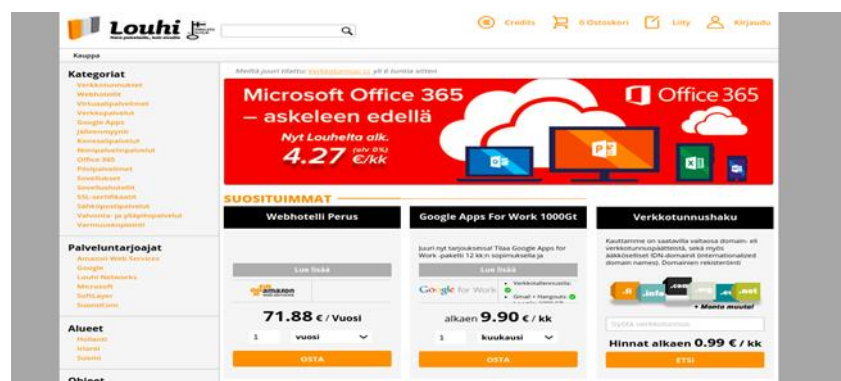
Ongelmien ratkaiseminen heti, kun niitä ilmenee projektin aikana, pitää kehittäjän ajan tasalla web-sovelluksen tai -sivun toiminnollisuudesta. Jos ongelmat jäävät huomioimatta, ne nopeasti johtavat uusiin ongelmiin ja

kasaantuvat kunnes pahimmillaan sovellus tai sivu ei toimi ollenkaan. Ongelmien ratkaiseminen kehitysvaiheessa parantaa myös jatkokehitystä, koska uusia toiminnollisuuksia ei suunnitella rikkinäiselle alustalle. (Dawson 2011, 32.)

### 4.3 Konsolin toteutuksen suunnittelu

Konsolin toteutusta suunniteltiin yhdessä Louhi Networks ja Pilvi Cloud Companyn johtavan ohjelmistokehittäjän kanssa. Toteutusta varten pidettiin suunnittelu palavereita, joissa käytiin läpi muun muassa, mitä ominaisuuksia asiakas konsolin ensimmäinen versio pitäisi sisältää.

Konsolin ulkoasun ensimmäinen versio suunniteltiin käyttämään samaa teemaa ja rakennetta kuin Louhi Networks verkkokauppa. Ikkunan vasemmalle puolelle tulee pystysuora valikko, jonka kautta loppukäyttäjä navigoi konsolissa. Konsolin oikeaan yläkulmaan tulee sisään- ja uloskirjautumista hallinnoiva valikko. Konsolin footer-osio tulee olemaan samanlainen kuin verkkokaupassa.



Kuva 24. Louhi Networks verkkokaupan etusivu.

#### 4.3.1 Tilin asetukset-osio

Asiakaskonsolin tilin asetukset-osio pitää sisällään samoja ominaisuuksia kuin oma.louhi.fi palvelun yhteystieto-osio. Asiakkaan on mahdollista muokata, lisätä tai poistaa tilin yhteystietoja. Myös asiakkaan tilin tiedot näytetään. Lisäksi verkkolaskutuksen operaattorin ja osoitteen voi määrittää osiosta ja valita verkkolaskulle toimitusmuodon.

**Yhteystiedot**

**Laskutus**

Voit kytkeä verkkolaskutuksen verkkopankkisi kautta.

Toimitustapa: Sähköposti / Oma Louhi ▼

Tallenna

Lisää uusi yhteystieto >>

Nimi: Mikko Tuppurainen

Osoite: Tupalantie 13 D 33

Postinumero: 04400

Toimipaikka: Järvenpää

Maa: Suomi

Email: mikko.tuppurainen@pilvi.com

Puhelin:

Roolit: laskutus, tilaaja, uutiskirje, yhteyshenkilö

Muokkaa Poista

Kuva 25. Oma.louhi.fi portaalin yhteystieto-sivun näkymä.

#### 4.3.2 Palvelut-osio

Toinen haluttu ominaisuus oli listanäkymä asiakastiliin kuuluvista palveluista, niihin liittyvistä sopimuksista ja näille omat yksityiskohtaisemmat sivut. Palveluosion etusivun listausnäköymän tulisi näyttää jokaisesta sopimuksesta sopimuksen numero, mikä on sopimuksen sen hetkinen tila, mitä palveluita siihen kuuluu, mihin asti siitä on laskutettu, mihin asti sitä on maksettu ja kuinka pitkä on laskutusjakso. Sopimuksen numeroa painamalla pääsisi sopimuksen omalle sivulle, jossa näytettäisiin samat tiedot kuin etusivulla ja tarkemmat tiedot sopimukseen kuuluvista palveluista, joita ovat tuotteiden yksittäiset ja jaksotetut hinnat, sekä tuotteisiin määritellyt tai liitetyt domainit. Oma.louhi.fi-sivun sopimustenhallinta sivusta tullaan tuomaan loputkin ominaisuudet uuteen asiakaskonsoliin tulevaisuudessa.

**Sopimukset**

Menemällä aktiivisen sopimuksen sivulle löydät toiminnon, jolla voit jatkaa sopimusta.

[Tulosta PDF voimassa olevista sopimuksista.](#)

Sopimusno	Tuotteet	Avattu	Voimassa	Tila
103690	Verkkotunnus (.com) paketin yhteydessä: mikkotestaa.com	4.5.2018	3.5.2019	Aktiivinen

[Näytä myös päättyneet sopimukset >>](#)

Kuva 26. Oma.louhi.fi portaalin sopimukset-sivun näkymä.

Aiempi kuva oma.louhi.fi-sivun sopimukset-osion etusivu näkymästä, josta asiakas näkee sopimuksensa.

**Sopimuksen tiedot**

Sopimusnumero:	103690
Sopimuksen tila:	Aktiivinen
Avattu:	4.5.2018
Maksettu asti:	4.5.2019
Laskutusjakso:	12 kk

[Irtisano sopimus](#) [Jatka sopimusta](#)

**Toimintoja**

- [Tulosta sopimus](#)

**Sopimuksen tuotteet**

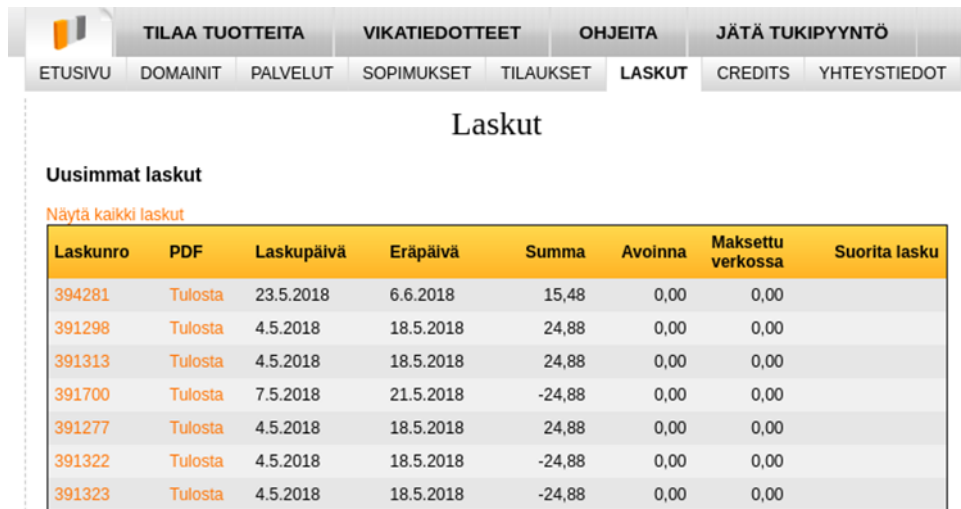
Palvelu	Domain	Palvelin	Määrä	Hinta/kk	Hinta/jakso
Verkkotunnus (.com) paketin yhteydessä	mikkotestaa.com	wp1.louhi.net	1	0,49	5,88
<b>Yhteensä</b>				<b>0,49</b>	<b>5,88</b>

Kuva 27. Oma.louhi.fi portaalin sopimuksen tiedot-sivun näkymä.

Aiempi kuva oma.louhi.fi-sivun yksittäisen sopimuksen näkymästä, josta asiakas voi tarkastella jokaisen sopimuksen tarkempia tietoja.

#### 4.3.3 Laskut-osio

Kolmas haluttu ominaisuus käsittelee käyttöliittymän käyttäjän laskuja. Lasku-osion etusivulle tulee listausnäkymä kaikista laskuista, joista näytetään laskun numero, laskun luontipäivä, laskun eräpäivä, laskun loppusumma sekä paljonko laskusta on maksamatta. Laskujen numeroa painamalla pääsee kyseisen laskun omalle sivulle, jossa on listattu samat tiedot kuin etusivulle sekä eritelty laskuun kuuluvat palvelut. Palveluista on lue-teltu määrä, laskutusjakson yksikkö sekä palvelun hinta. Laskun omalta si-vulta on myös mahdollista ladata PDF-versio laskusta.



**Laskut**


Uusimmat laskut

[Näytä kaikki laskut](#)

Laskunro	PDF	Laskupäivä	Eräpäivä	Summa	Avoinna	Maksettu verkossa	Suorita lasku
394281	<a href="#">Tulosta</a>	23.5.2018	6.6.2018	15,48	0,00	0,00	
391298	<a href="#">Tulosta</a>	4.5.2018	18.5.2018	24,88	0,00	0,00	
391313	<a href="#">Tulosta</a>	4.5.2018	18.5.2018	24,88	0,00	0,00	
391700	<a href="#">Tulosta</a>	7.5.2018	21.5.2018	-24,88	0,00	0,00	
391277	<a href="#">Tulosta</a>	4.5.2018	18.5.2018	24,88	0,00	0,00	
391322	<a href="#">Tulosta</a>	4.5.2018	18.5.2018	-24,88	0,00	0,00	
391323	<a href="#">Tulosta</a>	4.5.2018	18.5.2018	-24,88	0,00	0,00	

Kuva 28. Oma.louhi.fi portaalin laskut-sivun näkymä.

Aiempi kuva oma.louhi.fi-sivun laskut-osion etusivu näkymästä, josta asiakas voi tarkastella laskujaan.



**Laskun tiedot**

Laskunumero:	394281	Laskun toimitustapa:	Sähköposti / Oma Louhi
Viitenumero:	1839 42810	Pankkiyhteydet:	
Laskupäivä:	23.5.2018		
Eräpäivä:	6.6.2018		
Laskun summa:	15,48 EUR		
Avoim summa:	0,00 EUR		

[Tulosta PDF](#)

**Tuotteet**

Sellite	Määrä	Yksikkö	Yksikköhinta	Ale%	Alv%	Yhteensä
Verkkotunnus (.fi): mikkotestioso.fi	12	kk	1,04	0	24	15,48
<b>Yhteensä</b>						<b>15,48</b>

Kuva 29. Oma.louhi.fi portaalin laskun tiedot-sivun näkymä.

Aiempi kuva oma.louhi.fi-sivun yksittäisen laskun näkymästä, josta asiakas voi tarkastella laskun tarkempia tietoja, sekä tulostaa laskusta PDF-version.

#### 4.4 Suunnitelman dokumentointi ja esittäminen asiakkaalle

Asiakaskonsoli-projektin suunnitelma dokumentoitiin Pilvi Cloud Companyn prosessien mukaisesti. Suunnitteludokumentti pitää sisällään seuraavia asioita projektin toteutuksesta:

- yleisnäkymä
- tarkoitus
- kuvaus
- tavoitteet
- termit
- huomiot suuremmista muutoksista
- käyttötapauksia
- käyttäjätarinoita
- komponenttien määritelmiä
- mallikuvia käyttöliittymästä
- käyttöliittymän muutosten toteutussuunnitelma
- API:n ja tietokantojen muutosten toteutus suunnitelma
- aikataulutukset projektiin

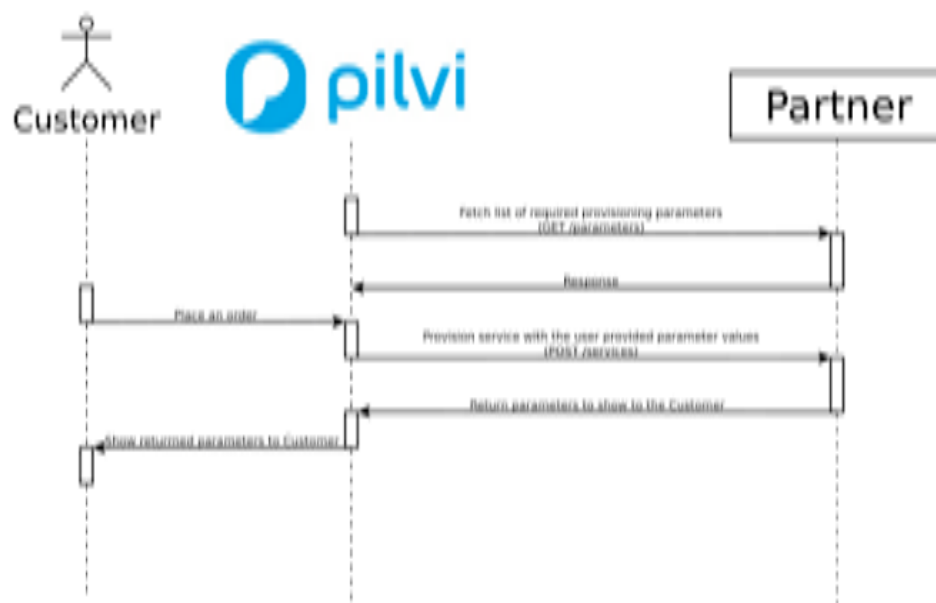
Jokaiselle osiolla määriteltiin omat suunnitteludokumentit yksi kerrallaan. Aina, kun yksi osio saatiin valmiiksi, alettiin laatimaan seuraavalle osiolla dokumenttia.

Dokumenttien valmistuessa ne esiteltiin Louhi Networks edustajalle ja Pilvi cloud companyn johtavalle kehittäjälle ja Pilven johdolle. Kun suunnitteludokumentti hyväksyttiin, voitiin aloittaa asiakaskonsolin kehittäminen siltä osalta.

## 5 KONSOLIN TOTEUTUS

### 5.1 Rajapinnat ja niiden integraatiot

Pilvi Cloud companyn tarjoaman ja ylläpitämän verkkokaupan sekä Louhi Networksin väliseen integraatio on toteutettu pilven PAITA-rajapinta integrointiprosessin kautta. Mukana on kuitenkin hyvin paljon Louhi Networksille tehtyä kustomointia johtuen Louhen laajasta tuote- ja palvelutarjonnasta. Pilven ja Louhen rajapintojen välinen integraatio oli valmiina ennen projektin alkua, mutta projektin aikana rajapintoihin lisättiin uusia funktioita tukemaan asiakaskonsolin ominaisuuksia, jotka käyttävät rajapintojen integraatiota provisiointi-datan siirtämiseen niiden välillä.



Kuva 30. Integration sequence (Pilvi Cloud Company n.d).

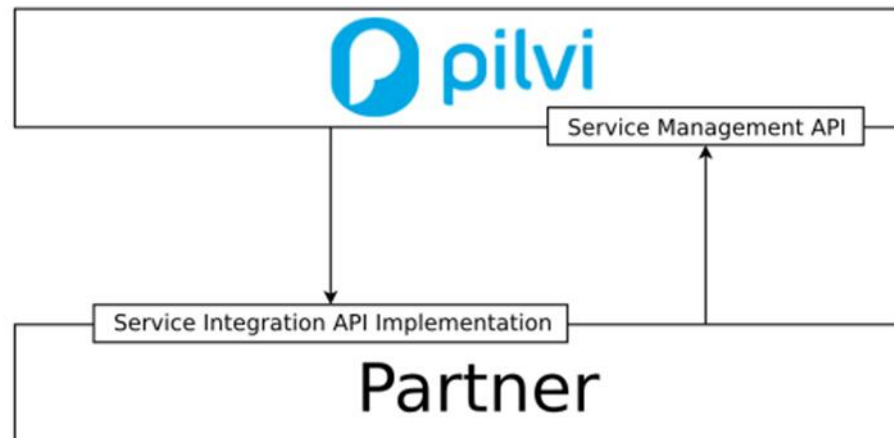
Aiemmassa kuvassa on esitetty yksinkertaisesti miten palvelun provisiointi tapahtuu PAITA-rajapinnan avulla.

#### 5.1.1 PAITA – rajapinta

Pilvi cloud company tarjoaa asiakkailleen Pilvi Automatic Integration Technology API eli PAITA-rajapintaa. Rajapinnan avulla asiakkaat pystyvät integroimaan tarjoamiaan palveluita Pilven tarjoamaan verkkokauppaan. PAITA-rajapinta koostuu kahdesta muusta Pilven rajapinnasta; Service Integration API:sta ja Service Management API:sta.



## PAITA - Pilvi Automatic Integration Technology API



Kuva 31. PAITA-Pilvi Automatic Integration Tehcnology API (Pilvi Cloud Company n.d).

Aiemmpi kuva kuvastaa PAITA-rajapinnan rakennetta suhteessa palvelun tarjoajaan.

### 5.1.2 Service Integration API

Service Integration API-integrointi mahdollistaa palvelun aktivoimiseen ja käyttämiseen tarvittavien parametrien tuonnin Pilven verkkokaupasta palveluntarjoajalle, jolloin käyttöliittymän käyttäjä saa ostamansa palvelun automaattisesti käyttöön. Palvelun tarjoajalla on myös mahdollisuus veloittaa käyttäjää palvelun käytön perusteella verkkokaupan ja Service Integration API:n avulla. (Pilvi Cloud Company n.d.)

### 5.1.3 Service Management API

Service Management API-integrointi antaa palvelun tarjoajalle mahdollisuuden muun muassa muuttaa Pilven verkkokaupassa myytävän palvelun parametreja automaattisesti, kun palvelua muutetaan tarjoajan päässä. (Pilvi Cloud Company n.d.)

### 5.1.4 Hostmaster v2.0 – rajapinta

Hostmaster v2.0 on Louhi Networks oma rajapinta, jolla muun muassa hallinnoidaan asiakkaita, tuotteita, tilauksia ja sopimuksia. Hostmasterilla on oma tietokanta, josta tuodaan dataa asiakaskonsolia varten. Hostmaster on integroitu Pilvi Cloud Companyn rajapinnan kanssa.

## 5.2 Kaupan API

Kaupan API:iin tarvittiin uusia funktioita hakemaan tarvittavaa dataa Louhi Networks rajapinnalta ja tietokannasta. Rajapinnat ovat REST-tyyppisiä joten data kulkee JSON-objekteina niiden välillä. Kaupasta lähetetään sen rajapinnalle request-kutsu tarvittavilla parametreilla, mitkä sitten käsitellään rajapinnassa niihin luoduilla uusilla funktioilla. Kun kaupan rajapinta on saanut vastauksen Louhi Networks rajapinnalta, se lähettää datan eteenpäin käyttöliittymälle objektimuodossa. Käyttöliittymä pilkkoo datan sopivaan muotoon ja näyttää ne käyttöliittymän käyttäjälle niihin rakennetuissa editoreissa. Käyttöliittymästä lähetetään myös url-muuttuja, jolla ohjataan kutsuja oikeisiin funktioihin API:ssa.

### 5.2.1 Tilin asetukset-osion API toteutus

Tilin asetukset-osion pitää pystyä hakemaan käyttöliittymän käyttäjän tilin tiedot ja yhteystiedot sekä muokkaamaan niitä. Näihin tarkoituksiin kaupan API:n luotiin GET-, PUT-, POST-, ja DELETE-funktiot. Käyttäjän ID:tä käytetään yhtenä parametrina tunnistamista varten. Se myös käy validoinnin läpi API:ssa, kun sinne lähetetään requesti käyttöliittymästä.

Tilin asiakastietoihin pystytään vaikuttamaan vain GET- ja PUT-funktioilla. Sillä ei haluta, että asiakas voi poistaa oman tilinsä tai lisätä uutta tiliä konsolin käyttöliittymän kautta. Tilin asiakastiedot pitävät sisällään tilin haltijan nimen, käyttäjätunnuksen, verkkolaskuosoitteen, verkkolaskuoperaattorin ja toimitustavan verkkolaskuille.

Tilin asiakas tietojen GET request- ja response-objektit:

```
▶ beforeSend: f (xhr)
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
▶ model: child (length: 1, models: Array(1), _byId: {}, parent: child, deferred: {}),
  parse: true,
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "GET"
  url: "http://api.local.pilvi.com/users/50567/account"
▶ _proto: Object
```

Kuva 32. GET-kutsun request-objekti.

Aiempi kuva kaupan API:lle lähetetystä GET-pyyntöstä.

```

▼ {, ...}
  ▼ data: {id: "50567", customerName: "Mikko Tuppurainen"
    customerName: "Mikko Tuppurainen"
    delivery_method: "none"
    finvoice_address: "asdasdasd"
    finvoice_routercode: "AABAFI22"
    id: "50567"
    username: "mikkotu"
  }

```

Kuva 33. GET-kutsun response-objekti

Aiempi kuva response-objektista mitä API palauttaa käyttöliittymän GET-kutsuun.

PUT- ja POST-requesteihin käyttöliittymä lähettää erillisen JSON-objektin, joka pitää sisällään tarvittavat arvot muutokseen.

Tilin tietojen PUT request- ja response-objektit:

```

▶ attrs: {finvoice_address: "fooBar"}
▶ beforeSend: f (xhr)
  contentType: "application/json"
  data: '{"request":{"finvoice_address":"fooBar"}}'
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
▶ model: child {cid: "c2809", attributes: {...}, collection: child, _changing: false, _previousAttributes: {...}, ...}
  parse: true
  patch: true
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "PUT"
  url: "http://api.local.pilvi.com/users/50567/account/50567"
  validate: true
  wait: true

```

Kuva 34. PUT-kutsun request objekti.

Aiempi kuva PUT-requestin objektista, jolla halutaan muuttaa asiakas tietojen verkkolaskuosoitetta.

```

▼ {, ...}
  ▼ data: {id: "50567", customerName: "Mikko Tuppurainen"
    customerName: "Mikko Tuppurainen"
    delivery_method: "none"
    finvoice_address: "fooBar"
    finvoice_routercode: "AABAFI22"
    id: "50567"
    username: "mikkotu"
  }

```

Kuva 35. PUT-kutsun response objekti.

Aiempi kuva API:n palauttamasta response-objektista PUT pyyntöön, joka näyttää päivitettyt asiakas tiedot.

API palauttaa päivitettyt asiakas tiedot käyttöliittymälle, kun PUT-funktio on ajettu rajapintojen kautta onnistuneesti.

Tilin asetukset-osion toinen ominaisuus oli yhteystietojen hallinnointi. Näitä asiakkaan pitää pystyä lisäämään, muokkaamaan ja poistamaan jotta käyttöön tulee kaikki aiemmin mainitut request-muodot. Asiakas pystyy määrittämään yhteystietoihin nimen, sähköpostin, puhelinnumeron, katuosoitteen, postinumeron, kaupungin, maan, onko yhteystieto laskutus henkilön, ovatko yhteystiedot yhteyshenkilön, onko yhteystieto tilaajan, ja lähetetäänkö yhteystiedon omaavalle henkilölle uutiskirje.

Yhteystietojen GET request- ja response-objektit:

```

▶ beforeSend: f (xhr)
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
▶ model: child {length: 1, models: Array(1), _byId: {_,}, parent: child, deferred: {_, _}}
  parse: true
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "GET"
  url: "http://api.local.pilvi.com/users/50567/contacts"
▶ proto : Object

```

Kuva 36. GET-kutsun request-objekti.

Aiempana kuva API:lle lähetetystä GET pyynnöstä, kun haetaan tietoja tilin asetukset-osioon.

```

▼ {contacts: [{_,...}]}
  ▼ contacts: [{_,...}]
    ▼ 0: {_,...}
      ▼ contact: {id: "64340", name: "Mikko Tuppurainen"
        address: "Tupalantie 13 D 33"
        city: "Järvenpää"
        country: "Suomi"
        created: "2018-01-10 09:43:41"
        email: "mikko.tuppurainen@gmail.com"
        id: "64340"
        modified: "2018-01-16 12:15:14"
        name: "Mikko Tuppurainen"
        name2: ""
        notes: ""
        phone: "+358505610259"
        postalcode: "04400"
        role_billing: true
        role_customer: true
        role_email: true
        role_tech: true
      }
    }
  }

```

Kuva 37. GET-kutsun response-objekti.

Aiempana kuva API:n palauttamasta response-objektista GET pyyntöön, kun haetaan tietoja tilin asetukset-osioon.

```

▼ attrs:
  ▼ contact:
    address: "Street 123"
    city: "Järvenpää"
    country: "Suomi"
    email: "mikko+lapilvi.com"
    name: "John Normandy"
    phone: "051234567"
    postalcode: "04400"
    role_billing: 0
    role_customer: 0
    role_email: 0
    role_tech: 0
    ▶ __proto__: Object
    ▶ __proto__: Object
  ▶ beforeSend: f (xhr)
  contentType: "application/json"
  data: '{"request":{"contact":{"name":"John Normandy","address":
  dataType: "json"
  ▶ error: f (jqXHR, textStatus, errorThrown)
  ▶ model: child {cid: "c7946", attributes: {__}, _changing: false,
  parse: true
  retryLimit: 2
  ▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "POST"
  url: "http://api.local.pilvi.com/users/50567/contacts"
  validate: true

```

Kuva 38. POST-kutsun request-objekti.

Aiempana kuva API:lle lähetetystä POST pyynnöstä, kun lisätään uusi yhteystieto.

```

▼ {__}
  ▼ contact: {id: "64511", name: "John Normandy"
    address: "Street 123"
    city: "Järvenpää"
    country: "Suomi"
    created: null
    email: "mikko+lapilvi.com"
    id: "64511"
    modified: "2018-07-27 16:56:55"
    name: "John Normandy"
    name2: null
    notes: null
    phone: "051234567"
    postalcode: "04400"
    role_billing: false
    role_customer: false
    role_email: false
    role_tech: false

```

Kuva 39. POST-kutsun response-objekti.

Aiempana kuva API:n palauttamasta response-objektista POST pyyntöön, kun lisätään uusi yhteystieto. Objektista nähdään, mitä on tallennettu tietokantaan ja varmistetaan, että uusi yhteystieto on lisätty onnistuneesti.

### 5.2.2 Palvelut-osion API toteutus

Palvelut-osion etusivun listausnäkyymään API:i tarvitsee käyttäjän ID:n parametriseksi, jonka avulla tiedot haetaan rajapintojen kautta tietokannasta. Etusivun listausnäkyymää käyttäjän ei ole tarkoitus muokata mitenkään, joten tässä tapauksessa pelkän GET-funktion luominen kaupan API:n riittää. Etusivulle tarvitaan seuraavat tiedot-sopimukset ID, sopimukseen liittyvät palvelut tai tuotteet, sopimuksen tila, laskutusjakso, mihin asti sopimusta laskutetaan ja mihin asti sopimusta on maksettu.

Palvelut-osion GET request- ja response-objektit:

```

▶ beforeSend: f (xhr)
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
▶ model: child {length: 14, models: Array(14), _byId: {_,}, parent: child, _listenId: "11225", _}
  parse: true
  reset: true
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "GET"
  url: "http://api.local.pilvi.com/profiles/58567/contracts"
▶ proto: Object

```

Kuva 40. GET-kutsun request-objekti.

Aiempana kuva API:lle lähetystä GET pyynnöstä, kun haetaan asiakkaan palveluita.

```

▼ {contracts: [{contract: {_,}}, {contract: {_,}}, {contract: {_,}}, _]}
  ▼ contracts: [{contract: {_,}}, {contract: {_,}}, {contract: {_,}}, _]
    ▼ 0: {contract: {_,}}
      ▼ contract: {_,}
        billing_period: "1kk"
        contract_id: "97379"
        id: "97379"
        invoiced_till: "2018-07-18"
        paid_till: "2018-07-18"
        products: "Nexetic Shield Backup Office 365: Lisenssi 10 kpl"
        start_date: "2018-06-18"
        status: "active"
      ▶ 1: {contract: {_,}}
      ▶ 2: {contract: {_,}}
      ▶ 3: {contract: {products: "Domain (.com): testidomainiiiiia.com", id: "7
      ▶ 4: {contract: {_,}}
      ▶ 5: {contract: {_,}}
      ▶ 6: {contract: {_,}}
      ▶ 7: {contract: {_,}}
      ▶ 8: {_,}
      ▶ 9: {_,}
      ▶ 10: {_,}
      ▶ 11: {_,}
      ▶ 12: {contract: {_,}}
      ▶ 13: {contract: {_,}}

```

Kuva 41. GET-kutsun response-objekti



Aiempana kuva API:n palauttamasta response-objektista GET pyyntöön, kun haetaan asiakkaan palveluita. Contracts-objekti pitää sisällään kaikki tilin ID:llä löytyvät sopimukset omina objekteinaan.

Palvelut-osion yksittäisten sopimusten sivua varten piti tehdä kaksi erillistä GET-funktiota: yksi sopimusten tiedoille ja yksi sopimukseen kuuluvien palveluiden tiedoille. Koska sopimuksen ja palveluiden tiedot tullaan näyttämään erilaisilla editoreilla, piti niille saada luotua oikean muotoiset response-objektit. Yksittäisten sopimusten sivulla asiakkaalle ei anneta oikeuksia muokata tietoja, joten GET-funktiot API:n riittävät tietojen näyttämiseksi.



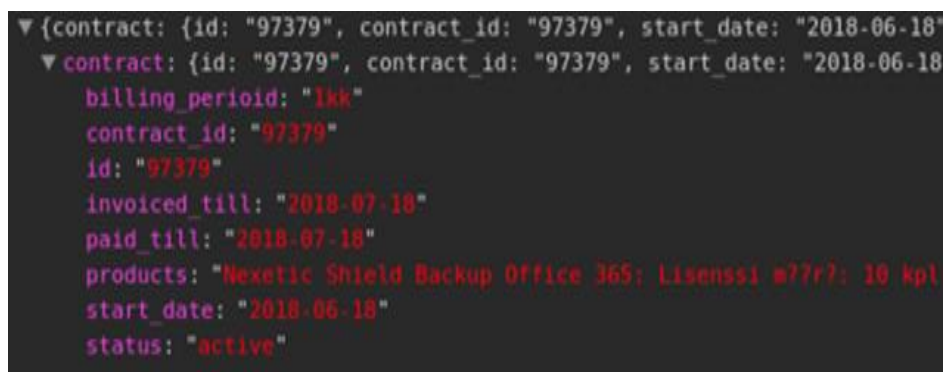
```

▶ beforeSend: f (xhr)
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
▶ model: child {cid: "c1243", attributes: {}, collection: child, _changing: false,
  parse: true
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "GET"
  url: "http://api.local.pilvi.com/profiles/50567/contracts/97379"

```

Kuva 42. GET-kutsun request-objekti.

Aiempana kuva API:lle lähetetystä request-objektista, kun haetaan dataa yksittäisen sopimuksen omaa sivua varten.



```

▼ {contract: {id: "97379", contract_id: "97379", start_date: "2018-06-18"}
  ▼ contract: {id: "97379", contract_id: "97379", start_date: "2018-06-18"}
    billing_period: "1kk"
    contract_id: "97379"
    id: "97379"
    invoiced_till: "2018-07-18"
    paid_till: "2018-07-18"
    products: "Nexetic Shield Backup Office 365; Lisenssi m77r7: 10 kpl"
    start_date: "2018-06-18"
    status: "active"

```

Kuva 43. GET-kutsun response-objekti.

Aiempana kuva API:n response-objektista yksittäisen sopimuksen oman sivun GET pyyntöön. Objektia tullaan käsittelemään model-muodossa joten sen rakenne on yksinkertaisempi.

```

▶ beforeSend: f (xhr)
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
▶ model: child {length: 2, models: Array(2), _byId: {…}, parent: child, _listenId:
  parse: true
  reset: true
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "GET"
  url: "http://api.local.pilvi.com/profiles/50567/contracts/97379/contractrows"
▶ __proto__: Object

```

Kuva 44. GET-kutsun request objekti.

Aiempana kuva API:lle lähetetystä request-objektista, kun haetaan sopimukseen liittyviä palveluita tai tuotteita.

```

▼ {, ...}
  ▼ contractrows: [{, ...}, {contractrow: {id: "164235", price: "0.00",
    ▼ 0: {, ...}
      ▼ contractrow: {id: "164235", price: "0.00",
        domain: "hjhghjghj"
        id: "164235"
        name: "Nexetic Shield Backup Office 365"
        period_price: "0.00"
        price: "0.00"
      ▼ 1: {contractrow: {id: "164236", price: "0.00",
        ▼ contractrow: {id: "164236", price: "0.00",
          domain: "Lisenssi m77r?: 10 kpl"
          id: "164236"
          name: "Nexetic Shield Backup Office 365"
          period_price: "0.00"
          price: "0.00"
        }
      }
    }
  }

```

Kuva 45. GET-kutsun response-objekti

Aiempana kuva API:n response-objektista sopimuksen palveluiden GET-pyyntöön. Objektin rakenne pitää sisällään muita objekteja ja response-objektia käsitellään collectionina käyttöliittymässä.

### 5.2.3 Laskut-osion API toteutus

Laskut-osion etusivun API toteutus tehtiin samalla tavalla kuin sopimusten ja palveluiden etusivu. Käyttäjälle ei ole oikeutta muokata laskujen tietoja etusivulla joten API:n puolelle tarvittiin vain GET-funktiot tietojen tuomista varten, jotka haetaan käyttäjän ID:llä. Laskut-osion etusivulle tarvitaan seuraavat tiedot laskuista eli laskun numero, laskun ID, laskupäivä, eräpäivä, laskun summa, ja paljonko laskusta on maksamatta.



```

▶ beforeSend: f (xhr)
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
▶ model: child {length: 16, models: Array(16), _byId: {...}, parent: child, _listenId:
  parse: true
  reset: true
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "GET"
  url: "http://api.local.pilvi.com/profiles/50567/invoicesHM2"
▶ __proto__: Object

```

Kuva 46. GET-kutsun request-objekti.

Aiempana kuva API:n request – objektista, kun haetaan asiakkaan ID:n perusteella laskuja rajapintojen läpi tietokannasta.

Yksittäisten laskujen omalle sivulle tiedot laskuista ja laskujen palveluista tai tuotteista tuodaan kahdella eri GET-funktiolla aivan kuten palvelut – osion toteutuksessa. Näin saadaan jälleen tuotua data oikeanlaisessa muodossa API:sta käyttöliittymään.

Laskun PDF tulostusta varten käytetään myös GET-funktiota API:ssa, joka hakee PDF:stä koodatun version rajapintojen kautta tietokannasta. API:lle lähetetään GET-kutsussa print-parametri, jonka arvon perusteella API joko hakee PDF:än tai ei hae.

```

▶ beforeSend: f (xhr)
▼ data:
  print: true
  ▶ __proto__: Object
  dataType: "json"
▶ error: f (jqXHR, textStatus, errorThrown)
  retryLimit: 2
▶ success: f (data, textStatus, jqXHR)
  tryCount: 0
  type: "GET"
  url: "http://api.local.pilvi.com/profiles/50567/invoicesHM2/319687"
▶ __proto__: Object

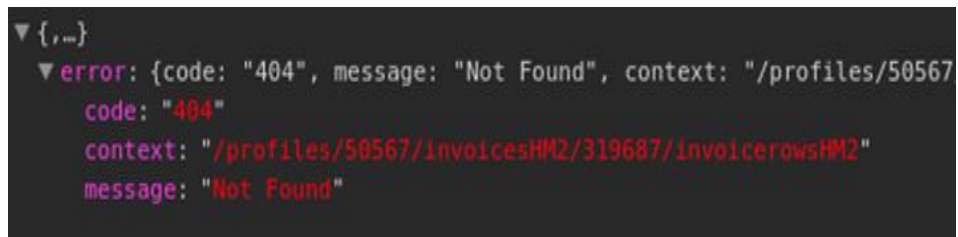
```

Kuva 47. GET-kutsun request-objekti.

Aiempi kuva API:lle lähetetystä GET-pyyynnöstä, jossa print-parametrin arvo "true" on merkki API:lle hakea ja palauttaa laskun PDF-versio käyttöliittymälle.

#### 5.2.4 Virhetilanteiden näyttäminen

Jos API ei löydä tarvittavaa dataa tietokannoista tai tarvittavat parametrit ovat virheellisiä. On kyseessä virhetila. Virhetilanteissa API keskeyttää funktion ja palauttaa response-objektin, joka pitää sisällään virheen koodin, kontekstin, missä yhteydessä virhe tuli ja viestin, mikä selittää vireen.



Kuva 48. API:n response-objekti virhe tilanteissa.

Aiempana kuva API:n palauttamasta response-objektista, joka ilmoittaa, että haluttua tietoa ei löydetty.



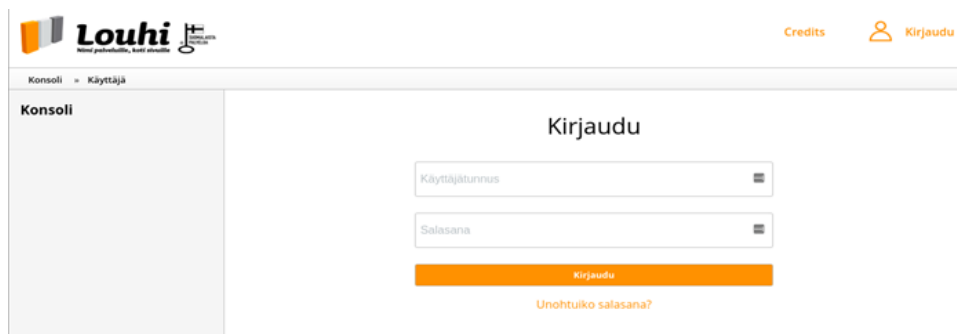
Kuva 49. Web-selaimen esittämä API virhe.

Aiempana kuva miten selain ilmoittaa API:n palauttaman virheen.

Virhetilanteiden ilmoittaminen on tärkeää ongelman löytämisen kannalta. Myös toimintojen kehittämisen aikana virhetilanteiden kartoittaminen helpottaa kehitystä ja mahdollistaa niiden estämisen tulevaisuudessa. Virhetilanteista jää myös lokimerkintä palvelimelle, josta ne voidaan käydä lukemassa jälkikäteen.

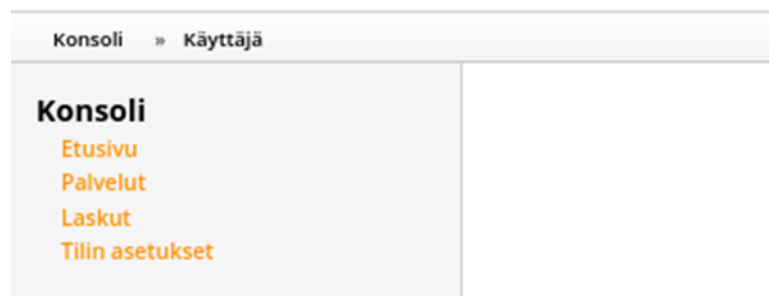
### 5.3 Käyttöliittymä

Käyttöliittymää varten konsolille ja sen jokaiselle osiolle piti rakentaa omat web-sivut. API:n tuoma data näytetään niille kustomoiduilla editoreilla, jotka hyödyntävät backbone.js web-sovelluskehityksen model- ja collection-objektien käsittely toiminnollisuuksia. Konsoliin mentäessä asiakasta pyydetään kirjautumaan sisään verkkokauppaan, jos hän ei ole jo valmiiksi kirjautuneena.



Kuva 50. Konsolin kirjaudu-sivu.

Kirjautumisnäkymässä konsolin sivumenun linkit on piilotettu. Konsolille luotiin myös etusivu, mutta siihen ei haluttu vielä toistaiseksi mitään sisältöä. Asiakas johdetaan etusivulle kirjautumisen jälkeen ja samalla sivuvalikkoon tulee palveluiden linkit näkyviin.

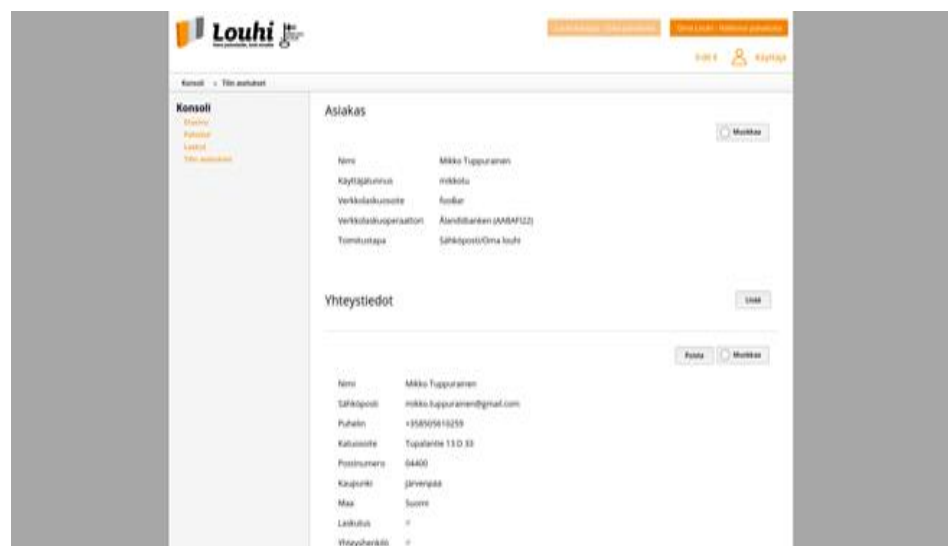


Kuva 51. Konsolin sivuvalikko.

Aiempana kuva konsolin sivuvalikosta, kun käyttäjä on kirjautunut sisään.

### 5.3.1 Tilin asetukset-osion käyttöliittymä

Tilin asetukset-osion sivulle rakennettiin kaksi editoria näyttämään tarvittavat tiedot. Ensimmäinen editori näyttää API:n palauttaman model-objektin asiakkaan tilin tiedoista. Toinen editori näyttää API:n palauttaman collection-objektin tilin yhteystiedoista.



Kuva 52. Konsolin tili asetukset-osio.

Aiempana kuva Tilin asetukset-osion käyttöliittymästä.

Asiakas editorin oikealla puolella olevasta muokkaa-painikkeesta asiakas voi laittaa editorin muokkaustilan päälle, jolloin hän pääsee muokkaamaan sallittuja kenttiä.

Asiakas

Tallenna Muokkaa ☐

Nimi	Mikko Tuppurainen
Käyttäjätunnus	mikkotu
Verkkolaskuosoite:	fooBar
Verkkolaskuoperaattori:	Ålandsbanken (AABAFI22) ▼
Toimitustapa:	Sähköposti/Oma louhi ▼

Kuva 53. Editorin muokkaus-tila.

Aiempana kuva asiakas-editorin muokkaus tilasta, josta asiakas hallitsee tilin tietoja.

Muokkaus tilassa asiakkaan määriteltävissä olevat kentät muuttuvat tekstikentiksi tai alasetoivalikoiksi. Muokkaus tilassa tulee tallenna-painike muokkaus-painikkeen viereen, jota painamalla tallennetaan muutokset.

Yhteystieto-editori muokkaus toimii samalla periaatteella kuin asiakas-editorin. Tämän lisäksi käyttäjällä on mahdollisuus lisätä tai poistaa yhteystietoja. Poistopainike sijoitettiin editorin muokkaustilaan tallenna-painikkeen viereen ja yhteystietojen lisäyspainike sijoitettiin editorin oikeaan yläkulmaan.

Yhteystiedot

Lisää

Poista Tallenna Muokkaa ☐

Nimi:	Mikko Tuppurainen
Sähköposti:	mikko.tuppurainen@gmail.com
Puhelin:	+358505610259
Katuosoite:	Tupalantie 13 D 33
Postinumero:	04400
Kaupunki:	Järvenpää
Maa:	Suomi ▼
Laskutus:	<input checked="" type="checkbox"/>
Yhteyshenkilö:	<input checked="" type="checkbox"/>
Tilaaja:	<input checked="" type="checkbox"/>
Uutiskirje:	<input checked="" type="checkbox"/>

Kuva 54. Editorin muokkaus-tila.

Aiempana kuva yhteystiedot-editorin muokkaus tilasta.

Yhteystietojen lisää-painike avaa uuden ikkunan, johon asiakas täyttää tarvittavat kentät uuden yhteystiedon luomiseen. Kun tarvittavat tiedot on täytetty ja vahvistettu, päivittyy uusi yhteystieto editoriin saman tien.

**Lisää yhteyshenkilö**

Nimi:

Katuosoite:

Postinumero:

Kaupunki:

Maa:

Sähköposti:

Puhelin:

Laskutus: ☐

Yhteyshenkilö: ☐

Tilaaaja: ☐

Uutiskirje: ☐

**Vahvista**

**Peruuta**

Kuva 55. Yhteystieto lomake.

Aiempana kuva lomake, joka täytetään kun halutaan lisätä uusi yhteystieto.

### 5.3.2 Palvelut-osion käyttöliittymä

Palvelut-osion etusivulle rakennettiin yksi editori näyttämään API:n palauttaman collection-objektin listaus näkymänä. Listaus näyttää sopimuksen ID:n, joka toimii samalla linkkinä sopimuksen ja sen palveluiden omalle sivulle. Etusivun editorissa on myös teksti kenttä, johon voi kirjoittaa hakusanoja löytääkseen halutun sopimuksen tai palvelun listauksesta.

**Palvelut**

Sopimus	Palvelut	Tilä	Alkupvm	Laskutusjakso	Laskutettu asti	Maksettu asti
103690	Verkkotunnus (.com) paketin yhteydessä: mikkotestaa.com	Aktiivinen	4.5.2018	12kk	4.5.2019	4.5.2019
103702	Sovellushotelli Wordpress Normi: mikkotestaa2.com	Suljettu	4.5.2018	1kk	4.6.2018	4.6.2018
103710	Sovellushotelli Wordpress Normi: mikkotestaa3.com	Suljettu	4.5.2018	1kk	4.6.2018	4.6.2018
103689	Sovellushotelli Wordpress Normi: mikkotestaa.com	Suljettu	4.5.2018	1kk	4.6.2018	4.6.2018

1 - 4

Kuva 56. Konsolin palvelut-osio.

Aiempana kuva palvelut-näkymän etusivun editorista, joka pitää sisällään listat sopimuksista ja palveluista.

Sopimuksen omalle sivulle rakennettiin kaksi editoria: yksi näyttämään API:n palauttaman model-objektin sopimuksen tiedoista ja toinen näyttämään API:n palauttaman collection-objektin tiedot sopimukseen sisältyvistä palveluista.


<b>Konsoli</b> <a href="#">Etusivu</a> <a href="#">Palvelut</a> <a href="#">Laskut</a> <a href="#">Tilin asetukset</a>		<b>Sopimus</b>			
		Sopimusnumero	103690		
		Tila	Aktiivinen		
		Alkupvm	4.5.2018		
		Laskutusjakso	12kk		
		Laskutettu asti	4.5.2019		
		Maksettu asti	4.5.2019		
		<b>Nimi</b>	<b>Verkkotunnus</b>	<b>Hinta €</b>	<b>Hinta/jakso €</b>
		<a href="#">Verkkotunnus (.com) pakettin yhteydessä</a>	mikkotestaa.com	0,49	5,88
		<b>Yhteensä:</b>		0,49	5,88
		1 - 1			

Kuva 57. Konsolin sopimus-osio.

Aiempana kuva sopimuksen omasta sivusta ja sen editoreista, jonka kautta käyttäjä näkee tarkemmat tiedot sopimuksista.

### 5.3.3 Laskut-osion käyttöliittymä


Laskut-osion etusivulle tarvittiin myös vain yksi editori näyttämään collection-objektin, jonka API palauttaa GET pyyntöön. Laskun numero toimii linkkinä yksittäisen laskun omalle sivulle, josta näkee myös mitä tuotteita tai palveluita lasku koskee sekä niiden yksittäiset hinnat ja yhteen lasketun hinnan. Laskujen omille sivuille rakennettiin kolme editoria näyttämään kaksi model- ja yksi collection-objektien editoria. Model editorit näyttävät laskun tiedot sekä verkkolaskuun liittyvät tiedot. Collection-editori näyttää laskuun sisältyvien tuotteiden tai palveluiden tiedot.



**Louhi**  
Kone palveluilla, ääli ensillä

Louhi Kauppa - Oma palvelu

Oma Louhi - Halinnoi palvelua

0.00 €  Käyttäjä

Konsoli » Laskut

**Konsoli**

[Etusivu](#)

[Palvelut](#)

[Laskut](#)

[Tilin asetukset](#)

Laskut

Lasku	Laskupäivä	Eräpäivä	Summa €	Avoinna €
<a href="#">394282</a>	23.5.2018	6.6.2018	-15,48	-15,48
<a href="#">394281</a>	23.5.2018	6.6.2018	15,48	15,48
<a href="#">391700</a>	7.5.2018	21.5.2018	-24,88	0,00
<a href="#">391313</a>	4.5.2018	18.5.2018	24,88	24,88
<a href="#">391322</a>	4.5.2018	18.5.2018	-24,88	0,00
<a href="#">391323</a>	4.5.2018	18.5.2018	-24,88	0,00
<a href="#">391277</a>	4.5.2018	18.5.2018	24,88	0,00
<a href="#">391298</a>	4.5.2018	18.5.2018	24,88	24,88

1 - 8

Kuva 58. Konsolin laskut-osio.

Aiempana kuva laskut-osion etusivun editorista, joka listaa käyttäjän kaikki laskut.



Kuva 59. Konsolin lasku-osio.

Aiempana kuva yksittäisen laskun omasta sivusta ja sen editoreista, joista käyttäjä näkee laskun tarkemmat tiedot.

Yksittäisen laskun sivun oikeassa yläkulmassa on PDF-laskun lataus painike, jota painamalla lasku latautuu käyttäjän koneelle.

#### 5.3.4 Virhetilanteiden näyttäminen käyttöliittymässä

Virhetilanteen tullessa käyttöliittymä avaa virhetilanneikkunan, joka kertoo virhekoodien mukaan lyhyesti, mistä virhe johtuu.



Kuva 60. Käyttöliittymän virhe-viesti.

Aiempana kuva virhe ilmoituksesta, kun laskua ei löydy tietokannasta.

Kun API palauttaa oman virheilmoituksensa, käyttöliittymä avaa virhetilanne ikkunan ja näyttää syyn API:n palauttaman virhe koodin perusteella.

#### 5.4 Konsolin toiminnollisuuksien testaaminen

Konsolin toteutusta testattiin useassa eri vaiheessa. Ensimmäiseksi testasin konsolia omassa lokaalilla kehitysympäristössä luomalla testi tunnuksia ja niihin tilauksia verkkokaupan kautta. Lokaaleja testauksia tehtiin kehityksen aikana ja sen loppuvaiheella.

Lokaalin testauksen jälkeen koodimuutokset annettiin muille kehittäjille luettaviksi ja tarkastettaviksi. Kehitysehdotusten jälkeen koodia parannettiin niiden mukaan lokaalissa ympäristössä ja uudet muutokset testattiin lokaalisti. Kun koodi muutokset olivat läpäisseet tarkastuksen, ne päivitettiin kehitystiimin yhteiseen kehitysympäristöön, jossa muut kehitystiimin jäsenet pääsivät testaamaan konsolin toimintoja käytännössä, etsimään virheitä ja antamaan palautetta.

Kun tarvittavat korjaukset yhteisen kehitysympäristön testien jälkeen oli tehty ja konsoli todettu korjatuksi, siirrettiin se tuotantoympäristöön, mutta kuitenkin vielä piilotettuna suurelta asiakaskunnalta. Tuotantoympäristössä testejä pääsivät tekemään myös Louhi Networksin työntekijät.

Jos missä tahansa vaiheessa huomattiin virhe, se testattiin yrittämällä toistaa se lokaalissa kehitysympäristössä. Kun virhe saatiin toistettua ja korjattua, vietiin se eteenpäin kaikkiin ympäristöihin ja testattiin, että virhe ei enää toistu niissä. Myös korjaukseen liittyvät koodimuutokset tarkastettiin muiden kehittäjien toimesta.



## 6 YHTEENVETO

Opinnäytetyön tavoitteena oli rakentaa ensimmäinen versio asiakaskonsolista Louhi Networksin verkkokauppaan annettujen määritelmien mukaan, joka hyödyntää Louhen omaa rajapintaa ja tietokantaa. Tavoitteena oli myös luoda selkeää ja helposti luettavaa koodia, josta olisi helppo lähteä jatkokehittämään asiakas konsolia tulevaisuudessa. Projekti tehtiin Pilvi Cloud Companyn toimeksiantona, jolta Louhi oli tilannut asiakaskonsolin. Louhen verkkokauppa oli tilattu kustomoituna Pilveltä, ja Pilvi ylläpitää ja kehittää verkkokauppaa.

Asiakaskonsoli antaisi käyttäjälle mahdollisuuden tarkastella ja muokata omia asiakas- ja yhteystietoja, tarkastella asiakkaan palveluita ja sopimuksia sekä tarkastella asiakkaan laskuja ja ladata niistä PDF-versio. Toteutus tehtäisiin samoilla teknologioilla ja työkaluilla, millä Louhen verkkokaup-pakin on toteutettu.

Projektin suurin haaste oli sen laajuus. Koska kyseessä oli ensimmäinen versio asiakaskonsolista, joka toimii pohjana jatkokehitykselle. Piti miettiä ratkaisuja API-funktioiden ja käyttöliittymän rakenteen yhteensopivuuden kanssa tulevaisuuden kannalta. Myös rajapintojen integraation kanssa oli paikoin haasteita saada oikea data tuotua eteenpäin käyttöliittymälle asti.

Haasteista huolimatta opinnäytetyöni onnistui mielestäni hyvin ja toimeksiantajat olivat myös tyytyväisiä asiakaskonsolin ominaisuuksien toimivuuteen. Konsolin ominaisuudet saatiin toteutettua selkeällä konventiolla verkkokaupan muuhun toteutukseen verraten. Kirjoitettu koodi myös pyrittiin tekemään mahdollisimman yksinkertaiseksi ja yhtenäiseksi muun toteutuksen kanssa, jossa onnistuttiin hyvin. Konsolin käyttäminen asiakkaan näkökulmasta katsottuna tuntui myös helpolta ja selkeältä. Projektin aikana oma osaaminen web-sovelluskehittäjänä myös kasvoi suuresti. Mitä pidemmälle projektissa edettiin sitä varmemmin pystyin toteuttamaan sitä.

Asiakas konsolia tullaan jatkokehittämään vielä useita kertoja ensimmäisen version jälkeen. Jatkokehitettävänä on ainakin tuoda loput ominaisuudet oma.louhi.fi-palvelusta konsoliin sekä tuoda täysin uusia ominaisuuksia sinne. Myös konsolin teema ja käyttöliittymä tulee saamaan todellisen ulkomuodon tulevaisuudessa, kunhan siihen liittyvät suunnitelmat ja dokumentaatiot on saatu valmiiksi.

## LÄHTEET

Apache (n.d). Apache Subversion FAQ. Haettu 18.7.2018 osoitteesta <https://subversion.apache.org/faq.html#why>

Auth0 (2017). A Brief History of JavaScript. Haettu 16.7.2018 osoitteesta <https://auth0.com/blog/a-brief-history-of-javascript/>

Backbone.js (n.d). Backbone.Collection. Haettu 16.7.2018 osoitteesta <http://backbonejs.org/#Collection>

Backbone.js (n.d). Backbone.Events. Haettu 16.7.2018 osoitteesta <http://backbonejs.org/#Events>

Backbone.js (n.d). Backbone.js Haettu 16.7.2018 osoitteesta <http://backbonejs.org/>

Backbone.js (n.d). Backbone.Model. Haettu 16.7.2018 osoitteesta <http://backbonejs.org/#Model>

Backbone.js (n.d). Backbone.View. Haettu 16.7.2018 osoitteesta <http://backbonejs.org/#Collection>

Dawson, A. (2011). Future-Proof Web Design. Yhdysvallat: John Wiley & Sons.

Facebook inc. (n.d) Getting started. Haettu 11.8.2018 osoitteesta <https://reactjs.org/docs/getting-started.html>

Free Software Foundation (n.d) GNU Emacs. Haettu 13.8.2018 osoitteesta <https://www.gnu.org/software/emacs/index.html#features>

Git-scm (n.d). Getting Started - A Short History of Git. Haettu 18.7.2018 osoitteesta <https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git>

Guillermo López-Anglada & The Sublime Text Community (2013). Extending Sublime Text. Haettu 17.7.2018 osoitteesta <http://sublime-text-unofficial-documentation.readthedocs.io/en/sublime-text-2/extensibility/extensibility.html>

HashiCorp (n.d) Introduction to Vagrant. Haettu 23.7.2018 osoitteesta <https://www.vagrantup.com/intro/index.html>

Jennifer Marsh (n.d) PHP vs. Java. Haettu 13.8.2018 osoitteesta <https://www.upwork.com/hiring/development/php-vs-java/>

jQuery (n.d). What is jQuery? Haettu 16.7.2018 osoitteesta <https://jquery.com/>

Jyväskylän yliopisto (n.d). JOHDATUS PHP-KIELEEN. Haettu 13.7.2018 osoitteesta [http://users.jyu.fi/~kolli/ITK215\\_05/php/?sivu=johdanto](http://users.jyu.fi/~kolli/ITK215_05/php/?sivu=johdanto)

Kelly, A. (2008). Changing Software Development : Learning to Become Agile. Yhdysvallat: John Wiley & Sons.

MariaDB (n.d) About MariaDB. Haettu 13.8.2018 osoitteesta <https://mariadb.com/about-us>

Mercurial (n.d) Work easier Work faster. Haettu 13.8.2018 osoitteesta <https://www.mercurial-scm.org/>

Mozilla (n.d). Let. Haettu 16.7.2018 osoitteesta <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

Navicat (n.d) Navicat for PostgreSQL. Haettu 13.8.2018 osoitteesta <https://www.navicat.com/en/products/navicat-for-postgresql>

Notepad (n.d) About. Haettu 13.8.2018 osoitteesta <https://notepad-plus-plus.org/>

Oracle (n.d) Chapter 1. First steps. Haettu 23.7.2018 osoitteesta <https://www.virtualbox.org/manual/ch01.html#virtintro>

Oracle Corporation (n.d) MySQL Documentation. Haettu 11.8.2018 osoitteesta <https://dev.mysql.com/doc/>

Pilvi Cloud Company (n.d.). Integration sequence. Haettu 26.7.2018 osoitteesta <http://developer.pilvi.com/docs/apis/paita/service-integration-api/1.0/>

Pilvi Cloud Company (n.d). Pilvi Automatic Integration Tehcnology API. Haettu 26.7.2018 osoitteesta <http://developer.pilvi.com/docs/apis/paita>

Python software foundation (n.d) Python/C API Reference Manual. Haettu 13.8.2018 osoitteesta <https://docs.python.org/3/c-api/index.html#python-c-api-reference-manual>

Sass-lang (2018). Intro to SCSS for Sass Users. Haettu 17.7.2018 osoitteesta [http://sass-lang.com/documentation/file.SCSS\\_FOR\\_SASS\\_USERS.html](http://sass-lang.com/documentation/file.SCSS_FOR_SASS_USERS.html)

Google (n.d) What is AngularJS?. Haettu 11.8.2018 osoitteesta <https://docs.angularjs.org/guide/introduction>

The PHP Group (n.d). History of PHP. Haettu 12.7.2018 osoitteesta <http://php.net/manual/en/history.php.php>

The PostgreSQL Global Development Group (n.d). A Brief History of PostgreSQL. Haettu 13.7.2018 osoitteesta <https://www.postgresql.org/docs/8.4/static/history.html>

W3Schools (n.d). JavaScript Syntax. Haettu 16.7.2018 osoitteesta [https://www.w3schools.com/js/js\\_syntax.asp](https://www.w3schools.com/js/js_syntax.asp)

W3C (2017). HTML 5.2. Haettu 17.7.2018 osoitteesta <https://www.w3.org/TR/html52/introduction.html#introduction-history>

W3Schools (n.d). CSS Syntax. Haettu 17.7.2018 osoitteesta [https://www.w3schools.com/js/js\\_syntax.asp](https://www.w3schools.com/js/js_syntax.asp)