Jani Saareks

HTML5-SOVELLUSKEHITYS
Case: Ilmavirtalaskuri

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2015

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** **30.11.2015**
Tekijä(t) Jani Saareks	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittelyn koulutusohjelma

Nimeke

HTML5 Sovelluskehitys – Case: Ilmavirtalaskuri

Tiivistelmä

Opinnäytetyön tavoitteena on HTML5-sovelluksen toteutus ja suunnittelu Jeven Oy:lle. Sovelluksen on tarkoitus helpottaa ilmavirtojen laskemista ja näin olla apuvälineenä työntekijöille. Sovellus on tulossa yrityksen päivittäiseen käyttöön työvälineeksi, joten toimintaan ja värimaailmaan tarvitsee kiinnittää huomiota.

Tässä opinnäytetyössä käsitellään eri toteutustapoja ja arkkitehtuurimalleja, joilla HTML5-sovelluksia voidaan kehittää. Työssä avataan myös käsitteitä, jotka liittyvät HTML5-sovelluskehitykseen. Samalla käydään läpi myös yleisesti, mitä HTML5 tarkoittaa ja mitä kehittäjän tulee ottaa huomioon, jos halutaan kehittää HTML5-sovellus.

Työn tuloksena toimeksiantajalle valmistui sovellus, jota voidaan tarvittaessa jatkokehittää. Toimeksiantaja voi myös julkaista työn, koska sovellus toimii myös mobiililaitteilla.

Asiasanat (avainsanat)

Ohjelmointi, JavaScript, sovellukset, HTML, Ajax-ohjelmointi, jQuery

Sivumäärä 31	**Kieli** Suomi

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi Arto Väätäinen	**Opinnäytetyön toimeksiantaja** Jeven Oy

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** November 30, 2015
Author(s) Jani Saareks	**Degree programme and option** Business Information Techology

Name of the bachelor's thesis

HTML5 Application development – Case: Air flow calculator

Abstract

The subject of my theses was to build and design an HTML5 application for Jeven Oy. The application was meant to serve as a tool for the company, so then they could more easily calculate air flows for their hoods. The purpose of this study was to upgrade their old application to an HTML5 application.

This thesis dealt with the different implementation methods and architectural designs that HTML5 applications can be developed with. Also the key concepts related to the HTML5 application development were explained. HTML5 was also explained in general terms and what a developer should take into account when developing an HTML5 application.

As a result of this thesis an HTML5 application was made for further development. The results achieved in this thesis will make it possible for the company to continue the development of the air flow calculator and to publish it when they see fit.

Subject headings, (keywords)

Programming, JavaScript, Applications, HTML, Ajax programming, jQuery

Pages 31	**Language** **Finnish**

Remarks, notes on appendices

Tutor Arto Väätäinen	**Bachelor's thesis assigned by** Jeven Oy

SISÄLTÖ

1 JOHDANTO

Tässä opinnäytetyössä tutkitaan millainen HTML5 on sovelluskehitysympäristönä. Työn tavoitteena on käsitellä eri toteutustapoja ja arkkitehtuurimalleja, joilla HTML5-sovelluksia voidaan kehittää. Työn kehittämistehtävänä on toteuttaa toimeksiantajalle uusi ilmavirtalaskurisovellus. Single-Page Application, kolmikerrosmalli ja MVC-malli mainitaan usein HTML5:n liittyvässä tekstissä. Tässä opinnäytetyössä selvitetään, mitä nämä termit tarkoittavat ja miten front-end ja back-end eroavat toisistaan.

Luvussa 2 käydään läpi termejä, jotka liittyvät HTML5-sovelluskehitykseen. Alussa käsitellään HTML5:sta HTML-kielen osana, jonka jälkeen siirrytään erilaisiin HTML5-sovellusten toteutustapoihin, joita käsitellään tarkemmin. Luvussa käsitellään myös arkkitehtuurimalleja, joiden mallien mukaan sovelluksia rakennetaan sekä mitä termit back-end ja front-end tarkoittavat ja mitä ne pitävät sisällään. Muutamaa tekniikkaa käsitellään tarkemmin, koska niitä on käytetty itse työssä.

Luvussa 3 esitellään toimeksiantaja yrityksenä sekä kerrotaan, miten sovelluksen kehitys edistyi ja mitä ongelmia oli muuttaa vanhaan kolmikerrosmalliin pohjautuva sovellus HTML5-sovellukseksi. Samalla esitellään vanha ilmavirtalaskuri ja kerrotaan esimerkein mitä siinä oli vialla ja mitä ominaisuuksia uusi sovellus pitää sisällään. Kerron myös, miksi projekti aloitettiin ja mitä hyötyä siitä oli. Viimeisestä luvusta löytyy mietelmät työn kulusta ja onnistumisesta.

2 HTML5-SOVELLUSKEHITYS

Sovellus on toiminnallinen kokonaisuus, joka sisältää omat ohjaustoiminnot ja on suunniteltu käyttäjälle (Beal 2015; Lehdonvirta & Korpela 2013, 13). Sovelluskehitys tarkoittaa prosessia, jonka tuloksena syntyy valmis sovellus (Rouse 2010). Moni varmasti yhdistää termin HTML5 webkehitykseen tai valmisteilla olevaan standardimalliin, eli termillä on selvästi kaksi eri merkitystä. Toinen merkitys tarkoittaa HTML-kielen standardin uutta kehitysvaihetta. Webkehitykseen liittyvänä terminä sillä tarkoitetaan sovellusten toteuttamistapaa, joka perustuu web-tekniikoiden käyttöön eli sovellus, joka käyttää HTML:ää (Hypertext Markup Language), CSS:ää (Cascading Style Sheets) ja JavaScriptiä (Lehdonvirta & Korpela 2013, 12).

2.1 HTML5 HTML-kielen kehitysvaiheena

HTML5 on uusin versio HTML-kielestä ja sisältää uusia elementtejä, esimerkiksi <*canvas*>, joka mahdollistaa grafiikan käsittelyn JavaScriptillä. HTML5 tarkoituksena on myös tehdä kielestä selvempää ja siksi elementtejä on pyritty nimeämään paremmin. (HTML5 2015), esimerkiksi tagi <*video*>, jonka sisään voidaan laittaa video. CSS3 pyrkii laajentamaan vanhaa CSS2.1 ja tuo lisäominaisuuksia, jolla on mahdollista tehdä visuaalisesti näyttävämpiä sovelluksia (CSS3 2015). CSS3 mahdollistaa elementin kulmien pyöristyksen sekä koon muokkauksen, tosin elementin kokoa ei pysty muokkaamaan Internet Explorer- tai Opera-selaimilla, koska selaimet eivät tue tätä ominaisuutta (Resize 2015).

Canvas-elementistä voidaan poimia muun muassa yksittäisen pikselin väri (kuva 1) ja muuttaa sen tai muiden pikselien väriarvoja skripteillä. Tämä mahdollistaa kuvankäsittelyn selaimessa ja erilaisia filttereitä on helppo pistää 2d-kuville canvasin avulla.

```
//haetaan canvas elementti
var canvas = document.getElementById('canvas');
//otetaan canvaksen 2d sisältö muuttujaan
var tausta = canvas.getContext('2d');
function vari(event) {
    var x = event.layerX;
    var y = event.layerY;
    var pixel = tausta.getImageData(x, y, 1, 1);
    var data = pixel.data;
    //pikselin väriarvo RGBA(red, green, blue, alpha)
    var rgba = data[0] + ',' + data[1] +',' + data[2] + ',' + data[3];
    console.log(rgba);
}
//hiiren liikkuessa kutsutaan funktiota
canvas.addEventListener('mousemove', vari);
```

KUVA 1. Pikselin väri hiiren osoittimessa canvaksessa

Uudet HTML5-elementit ja ominaisuudet siis mahdollistavat hyvinkin monimutkaisten sovellusten tekemisen selaimeen. Uusilla ominaisuuksilla on myös pyritty siihen, että sovelluksista saataisiin toimivia lähes kaikilla uusilla alustoilla, koska ohjelman ajamiseen riittää selain ja mitään selaimen lisäosia ei tarvita.

2.2 Single-Page Application

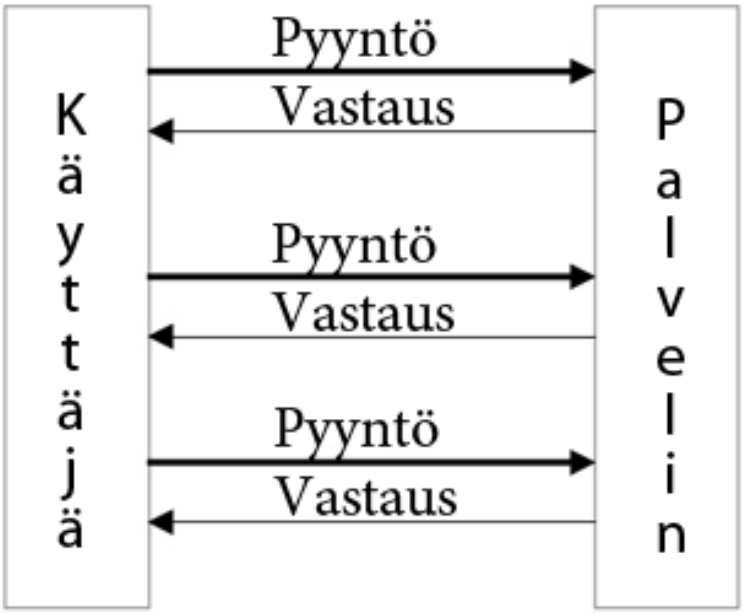
Perinteisissä web-sovelluksissa aina kun sovellus kutsuu palvelinta, näytetään käyttäjälle uusi HTML-sivu, josta aiheutuu sivuston uudelleenlataus selaimessa. Tämä tyyli rasittaa palvelinta ja vaatii käyttäjältä verkkoyhteyttä, koska muuten sovellus ei pysty näyttämään sivua. (Wasson 2013).

HTML5-sovellus koostuu yhdestä HTML-dokumentista (kuva 2), jonka sisältöä muutellaan dynaamisesti ja yleisesti puhutaankin Single-Page Applications eli ”yksisivuisista sovelluksista”. Yksisivuinen sovellus käyttää Ajaxia sivujen päivittämiseen niin, että jonkin sivun osa päivitetään ja tämä ei aiheuta käyttäjälle sivuston uudelleen latausta. Ajaxia käytetään tiedon hakemiseen palvelimelta, jolloin HTML5-sovelluksen käsitteeseen ei sisälly vaatimusta että sen on toimittava täysin paikallisesti (Lehdonvirta & Korpela 2013, 14). Tosin paikallista toimivuutta yleensä tavoitellaan ja verkkoyhteyttä käytetään vasta kun on aivan pakko. HTML5-sovellus myös pyrkii noudattamaan RESTin periaatteita (Lehdonvirta & Korpela 2013, 54; Wasson 2013).

KUVA 2. HTML5-sovelluksen idea (Lehdonvirta & Korpela 2013, 14)

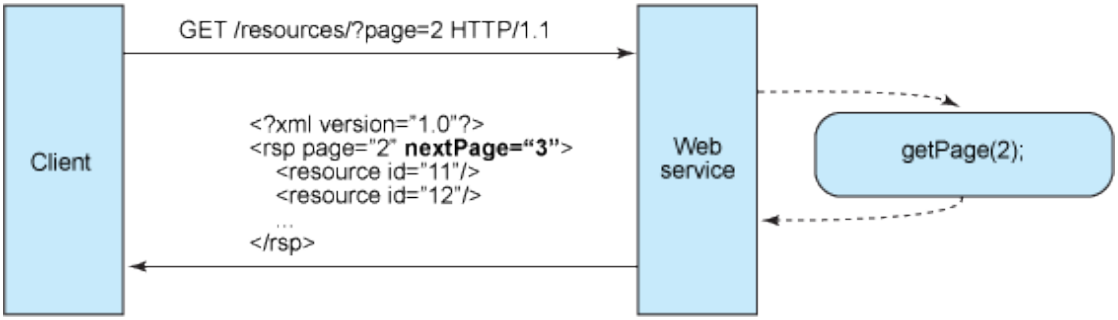
REST (Representational State Transfer) on arkkitehtuuri tyyli, jolla voidaan toteuttaa web-palvelu, joka käyttää HTTP-pyyntöjä eri laitteiden välillä (kuva 3). REST-palvelussa palvelimelle ei tallennu mitään dataa asiakkaasta, minkä seurauksena REST on erittäin yksinkertainen ja kevyt. REST käyttää neljää eri metodia GET, POST, PUT ja DELETE. GET tarkoittaa tietojen tulostusta eli käyttäjä haluaa lukea jotain dataa. POST tarkoittaa uuden tiedon lisäämistä. PUT:tia käytetään muokkaa-

maan jo olemassa olevaa dataa ja DELETE:ä käytetään tiedon poistoon. (Rodriguez 2015). Webohjelmoinnissa, jos rajanpinta (API) noudattaa RESTin periaatteita puhutaan siitä nimellä REST rajapinta, vaikka ohjelmointirajapinta ei olisikaan täysin RESTin mukainen. Rajapintojen tarkoitus on tarjota asiakassovelluksille ohjelmointirajapinta palvelimilla sijaitsevien resurssien käyttöön.



KUVA 3. Miten REST toimii

Hyvin tehdyssä web-palvelussa, jos halutaan saada käyttäjän tiedot, jolla on id 2345, voitaisiin kutsu suorittaa seuraavalla tavalla www.service.fi/users/2345. Tällainen kutsu on REST-rajapinnan mukainen. Jos palvelua ei ole suunniteltu hyvin, kutsu voitaisiin tehdä seuraavasti www.service.fi/get?type=user&id=2345. Hyvänä URL suunnitteluna voidaan pitää sitä, jos hakemistot ovat monikkomuodossa esimerkiksi www.service.fi/users/2345/images/95 eikä www.service.fi/user/2345/image/95. Myös nimet tulisi pitää mahdollisimman selkeinä ja yksinkertaisina, koska tämä helpottaa palvelun käyttöä. Samoin erikoismerkkien kuten viivojen käyttöä tulisi välttää, koska kaikki palvelimet eivät tue niitä, ja kaikkien kirjaimien pitäisi olla pienellä eli hakemistoissa ei saa käyttää suuria alkukirjaimia (REST API Quick Tips 2015).



KUVA 4. Tilattoman palvelimen toiminta (Rodriquez 2015)

REST-periaatteita noudattavat sovellukset siis käyttävät HTTP-pyyntöjä ja ovat tilattomia eli sovelluksen täytyy lähettää palvelimelle kaikki se tieto, jota palvelin tarvitsee, jotta se pystyy suorittamaan pyynnön. Eli palvelin ei pidä mitään historiaa aikaisemmista pyynnöistä. Palvelinta ei siis voida pyytää näyttämään edellistä sivua, vaan täytyy pyytää suoraan sivua, joka halutaan näyttää käyttäjälle esimerkiksi haeSivu(5); (kuva 4). Palvelimen vastaus yleensä sisältää vaadittavan datan JSON- tai XML-muodossa, mutta se voi myös olla kuva tai jokin tekstimuoto. Datan tyyppi yleensä valitaan käyttötarkoituksen mukaan. HTML5-sovelluksissa yleensä palvelimen vastaus on JSON-muodossa. (Lehdonvirta & Korpela 2013, 54; Rodriquez 2015.)

XML eli Extensible Markup Language on kieli, joka on suunniteltu datan välitykseen, ei sen näyttämiseen. XML ei sisällä itsessään mitään ennalta määrättyjä tageja, vaan käyttäjä saa itse nimetä käyttämänsä tagit. Tagillä tarkoitetaan elementtiä, joka pitää sisällään dataa eli sillä on aloitus ja lopetus. XML:ssä on kuitenkin rajoituksia esimerkiksi ei voida suoraan käyttää > merkkiä vaan se täytyy merkitä >, jos sitä halutaan käyttää tagin sisässä. XML-tiedostossa kaikki data on aina, jonkin juuri elementin alla. Kuvassa 5 henkilöt tagi toimii juurena, jonka alle on luotu henkilö elementti, joka sisältää tietoja henkilöstä. Tagilla nimi on kaksi lapsielementtiä etunimi ja sukunimi.

```
<?xml version="1.0" encoding="UTF-8"?>
<henkilot>
    <henkilo>
        <nimi>
            <etunimi>Teppo</etunimi>
            <sukunimi>Virtanen</sukunimi>
        </nimi>
        <ika>19</ika>
    </henkilo>
    <henkilo>
        <nimi>
            <etunimi>Matti</etunimi>
            <sukunimi>Aalto</sukunimi>
        </nimi>
        <ika>45</ika>
    </henkilo>
</henkilot>
```

KUVA 5. Esimerkki XML-tiedostosta

JSON eli JavaScript Object Notation on kevyt datan siirtomuoto ja ihmisten on helppo lukea sitä. JSON koostuu objekteista, jotka voivat myös sisältää taulukoita. JSON-objekti luodaan aaltosulkeiden avulla. (JSON 2015; Lengstorf 2015). Kuvassa 6 on esitettynä sama data JSON muodossa kuin mitä kuvassa 5 on XML-tiedostona. JSON-objekti on paljon kevyempi käsitellä suurta määrää dataa kuin XML-tiedosto. JSON-objektin sisältämää tietoa on myös helppo tarkastella koodilla. Jos halutaan saada ensimmäisen henkilön etunimi selville, tapahtuu se seuraavasti JavaScriptillä: *json.henkilot.henkilo[0].nimi.etunimi*. JSON-dataa on siis hyvinkin helppo käyttää.

```
var json={
  "henkilot": {
    "henkilo": [
      {
        "nimi": {
          "etunimi": "Teppo",
          "sukunimi": "Virtanen"
        },
        "ika": "19"
      },
      {
        "nimi": {
          "etunimi": "Matti",
          "sukunimi": "Aalto"
        },
        "ika": "45"
      }
    ]
  }
};
```

KUVA 6. Esimerkki JSON-objektista

Ajax eli Asynchronous JavaScript and XML mahdollistaa sovelluksen kommunikoinnin palvelimen kanssa ilman erillistä sivun latausta. Nimestä riippumatta XML:n käyttö ei ole mitenkään olennaista vaan palvelimen vastaus voi olla muussakin muodossa, nykyään tavallisesti JSON, koska JSON-tiedostot ovat pienempiä kuin vastaavat XML-tiedostot (Ajax 2015). Pyyntö voidaan myös suorittaa HTTPS-protokollan mukaan, jolloin saadaan salattua data.

Ajax on niin sanotusti epäsynkroninen, mikä tarkoittaa sitä että Ajax-kutsut eivät keskeytä ajettavaa lähdekoodia, vaan pyyntö suoritetaan taustalla. Tämän jälkeen ajetaan funktiot, joilla käsitellään palvelimelta saatua tietoa. Ajax-kutsut voidaan tehdä JavaScriptillä (kuva 7) tai erilaisilla JavaScript-kirjastoilla kuten jQuery tai AngularJS, joista lisää seuraavassa luvussa. Kuvassa 8 esitetään jQuery-kirjastolla tehty Ajax-kutsu, jossa käytetään metodin omaa callback-funktiota *done* tai *fail* riippuen pyynnön onnistumisesta. Callback-funktiot ovat tapahtumia, jotka suoritaan tietyn tapahtuman jälkeen eli Ajax-kutsussa sitten kun on saatu palvelimelta tarvittavat tiedot takaisin (jQuery Callback Functions 2015).

```
var xhr = new XMLHttpRequest();
xhr.open('GET', encodeURI('main.html'));
xhr.onload = function() {
    if (xhr.status === 200) {
        //Suoritetaan jos pyyntö onnistui
    }
    else {
        //Suoritetaan jos pyyntö epäonnistui
    }
};
xhr.send();
```

KUVA 7. Esimerkki Ajax kutsusta natiivi JavaScriptillä

Nykyisin Ajaxilla tarkoitetaan XHR eli XMLHttpRequest-objektia (Lehdonvirta & Korpela 2013, 55), jota käytetään jQuery-kirjastossa functiolla *$.ajax({})* (kuva 8). Kaikki selaimet tukevat nykyään XHR.ää (Van Kesteren ym. 2014).

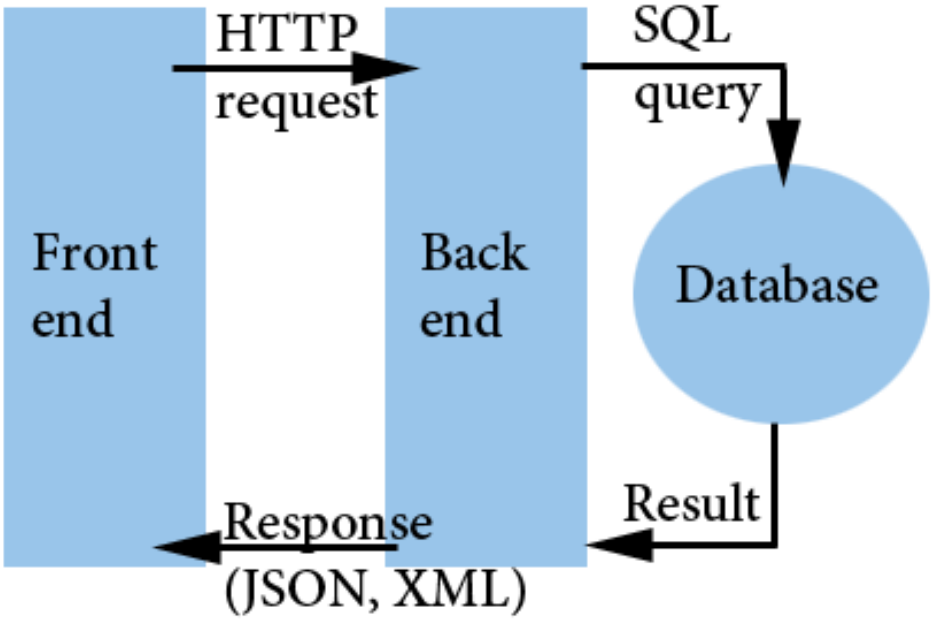
```
$.ajax({
  url: "main.html"
}).done(function() {
    //Suoritetaan jos pyyntö onnistui
  })
  .fail(function() {
   //Suoritetaan jos pyyntö epäonnistui
  });
```

KUVA 8. Esimerkki Ajax-kutsusta jQuery kirjastolla

jQueryn avulla tehty Ajax-kutsu on syntaksiltaan paljon helpompi, koska käyttäjän ei tarvitse tietää numerokoodeja, joilla kerrotaan onnistuiko kysely vai ei.

2.3 Back-end ja front-end

Back-end tarkoittaa palvelinpuolen ohjelmointia, mikä ei suoraan näy loppukäyttäjälle. Ilman back-endiä, front-end sovellukset joiden tarvitsee tallentaa tietoa, eivät olisi mahdollisia (Wales 2014). Back-end siis tukee front-end sovelluksia. Back-end koostuu kolmesta osasta palvelimesta, sovelluksesta ja tietokannasta (Girdley 2015). Esimerkiksi verkkokaupassa asiakkaiden ja tuotteiden tiedot tulevat yleensä tietokannasta. Jos asiakas haluaa ostaa jonkin tuotteen, niin voidaan tuotteen tarkemmat tiedot pyytää palvelimelta, vaikka Ajax-kutsulla. Ajaxilla voidaan kutsua tiettyä ohjelmaa palvelimella, joka on voitu toteuttaa PHP:lla, Rubyllä, Javalla tai jollain muulla palvelinpuolen ohjelmointikielellä. Ohjelma hakee tietokannasta asiakkaan haluaman tuotteen tiedot ja palauttaa ne takaisin front-end sovellukselle (kuva 9).



KUVA 9. Kaavio Front-end, back-end toiminnasta

PHP (PHP:Hypertext Preprocessor) on palvelinpuolen ohjelmointikieli, jota käytetään dynaamisten verkkosivujen tekemisessä. PHP on skriptikieli ja sitä voidaan upottaa HTML-sivuille (kuva 10). Kaikki yleisimmät käyttöjärjestelmät tukevat PHP:ta kuten Linux, Microsoft Windows ja Mac OS X sekä lähes kaikki webhotellit nykyisin tukevat myös PHP:ta (What can PHP do? 2015).

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>PHP esimerkki</title>
    </head>
    <body>
        <?php
        print "<h1>PHP</h1>".
              "<p>Sisältöä sivulle</p>";
        ?>
    </body>
</html>
```

KUVA 10. Esimerkki PHP-koodista HTML-sivulla

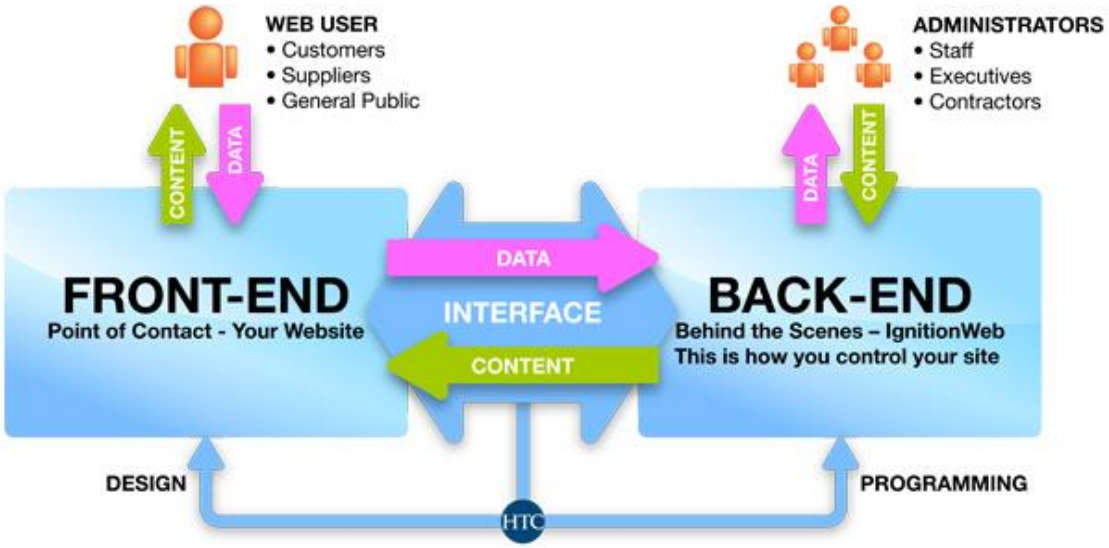
Tyypitys ei ole PHP:n vahvin puoli eli muuttujille ei tarvitse erikseen määrittää, minkä tyyppisiä arvoja ne tulevat sisältämään, joten se voi olla yksi tekijä, jonka takia kieli on helppo oppia. Toisaalta tyypityksen puute on myös yksi PHP:n heikkous, koska kääntäjä ei löydä kaikkia virheitä niin helposti kuin vahvemmin tyypitetyissä kielissä. PHP sisältää myös valmiiksi paljon ennalta määriteltyjä funktioita, joita on helppo käyttää ja ne nopeuttavat koodausta. PHP:n valmiiden funktioiden ongelma on se, että niitä ei ole kaikkia nimetty samalla tavalla. Osa voi olla kirjoitettu suoraan yhteen esimerkiksi *strlen()*, osa sisäänrakennetuista funktioista on taas nimetty niin, että sanat on erotettu toisistaan alaviivalla kuten *mysql_connect()*.

PHP on erittäin yleinen kieli ja esimerkiksi suosittu sisällönhallintajärjestelmä Wordpress on rakennettu PHP:n päälle. Myös työpöytäsovelluksia on mahdollista tehdä PHP:lla, mutta ensisijaisesti se on kuitenkin tarkoitettu palvelinpuolen ohjelmointiin (What is PHP-GTK? 2015). PHP:lle löytyy monia sovelluskehyksiä (framework), joista suosituin varmasti on Zend Framework.

Zend Framework on avoimeen lähdekoodiin perustuva olio-pohjainen sovelluskehys, jota käytetään paljon PHP 5-version kanssa. Zend Frameworkistä on myös olemassa Zend Framework 2, joka vaatii toimiakseen PHP-version 5.3 (Zend Technologies

2015). Zend Frameworkissä on mahdollista käyttää vain niitä komponentteja, joita tarvitsee ja haluaa käyttää eli kehittäjä saa itse valita, haluaako hän käyttää Zendin komponentteja vai jotain muita projektissaan, johon on tuotu Zend. Zendin tarkoituksena on helpottaa ja nopeuttaa PHP:n käyttöä (Lehdonvirta & Korpela 2013, 58). Zend esimerkiksi tarjoaa HTML5-formin käsittelyyn komponentit, joten esimerkiksi tietojen validointi hoidetaan olion kautta. Zend Frameworkin pääsponsori on Zend Technologies, mutta myös Google, Microsoft ja Strikelron toimivat Zendin kanssa yhteistyössä (Zend Technologies 2015).

Front-endistä puhuttaessa tarkoitetaan sitä osaa verkkopalvelusta, jonka käyttäjä näkee, ja jolla käytetään palvelua (kuva 11). Ennen web-kehityksessä työskenteli useita eri ihmisiä eri osa-alueilla. Osa työskenteli ainoastaan Photoshopin kanssa ja osa koodasi sivuja käyttäen HTML ja CSS. Nykyisin front-end kehittäjän on hallittava Photoshop, HTML:ää, CSS:ää ja JavaScriptiä, koska lähes kaikki sivut koostuvat näiden yhdistelmistä. (Girdley 2015). Front-end on siis selaimessa toimiva osa järjestelmää. Front-endin yhteydessä voidaan myös puhua HTML5-sovelluksesta, jos front-end sovellus on toteutettu käyttäen HTML, CSS ja JavaScript ja se toimii selainmoottorissa (Lehdonvirta & Korpela 2013, 48). On mahdollista myös toteuttaa front-end sovellus, joka ei tarvitse yhtään back-end ohjelmointia. Esimerkiksi staattiset sivut, joiden sisältöä ei tarvitse muuttaa tai sivusto, jonka ei tarvitse tallentaa tietoa, voidaan toteuttaa ilman back-end koodaamista (Kavourgias 2015).

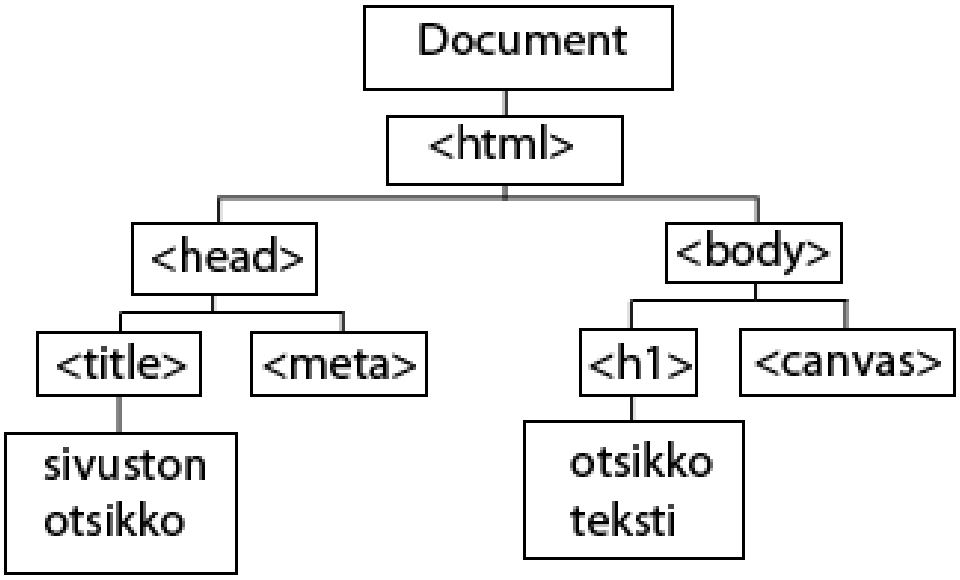


KUVA 11. Käyttäjien roolit Front-end ja back-end (Website Front-end and Back-end 2015)

Front-endiin tehtävät muutokset koodissa eivät vaikuta mitenkään back-endin toimintaan vaan sivuston ulkoasua ja toteutustapaa voidaan muutella ilman, että mikään toiminnallisuus kärsii, koska kyseessä on kaksi aivan eri osa-aluetta. Back-endiin tehtävät muutokset sen sijaan voivat vaikuttaa front-end sovelluksen toimintaan. Jos muutetaan palvelimenvastauksen muoto XML:stä JSON:iin, niin tämä täytyy huomioida sovelluksen koodissa. Tietokannan tyypin vaihto tai palvelinpäänohjelmointi kielen vaihto ei vaikuta front-endiin millään tavalla, kunhan vastaukset ja pyynnöt toimivat samalla periaatteella kuin ennenkin.

Selainpuolen ohjelmoinnissa käytetään yleensä jotain JavaScript-kirjastoa, mikä yksinkertaistaa käytettävää JavaScript koodia. Tunnetuin kirjasto on varmasti jQuery, joka on julkaistu vuonna 2006. JavaScript-kirjastot kuten jQuery myös yleensä huolehtivat selainten välisistä eroavaisuuksista, mikä helpottaa ylläpitoa (Lehdonvirta & Korpela 2013, 59). Tärkein jQueryn ominaisuus kuitenkin on DOM-manipulointi.

DOM (Document Object Model) on dokumentin esitystapa, jossa rakenne esitetään puumallisena (kuva 12). HTML-dokumentissa DOM-manipulaatio mahdollistaa sivun elementtien muokkauksen ilman sivuston uudelleenlatausta. Yleensä DOM-manipulaatio suoritetaan JavaScriptin avulla, mutta sen pystyy tekemään myös muilla kielillä, koska DOM itsessään ei ole osa JavaScript kieltä (Document Object Model (DOM) 2015).



KUVA 12. Esimerkki DOM-puusta

jQueryn avulla voidaan etsiä tiettyä DOM-elementtiä esimerkiksi id:n, luokan tai tagin perusteella tai vaihtoehtoisesti voidaan valita kaikki elementit. jQueryssä elementin id:seen viitataan seuraavasti *$(”#otsikko1”);* edellinen koodin pätkä valitsee elementin, jolle on annettu id otsikko1. Id:n valitsin on siis jQueryssä #, luokkaan viitataan pisteellä esimerkiksi $(”.*luokka”.)*. Tagiin viitatessa käytetään vain tagin nimeä *$(”title”);* (kuva 13), jos halutaan valita kaikki dokumentin elementit käytetään valitsimena *$(”*”);* (jQuery 2015). Elementtejä siis valitaan samoilla valitsimilla kuin CSS.

```
// Dokumentin titlen muutos JavaScriptillä
document.getElementsByTagName("title")[0].
        innerHTML = "JavaScript";
// Sama muutos jQueryllä
$("title").text("jQuery");
```

KUVA 13. DOM hallinta JavaScript ja jQuery

Kuten kuvasta 13 käy ilmi, voidaan jQueryn avulla lyhentää merkittävästi tarvittavan koodin käyttöä ja elementtiin viittaaminen on helpompaa kuin JavaScriptillä. Koska

dokumenteilla on vain yksi title-elementti, ei sille tarvitse määritellä erikseen id:tä vaan voidaan käyttää tagi valitsinta. Jos haluttaisiin muokata vain tietyn otsikon tekstiä ja dokumentti sisältää useamman otsikon, on hyvä tarkentaa valitsinta, koska muuten muutos tehdään kaikkiin elementteihin. Käytännössä siis muutos tehtäisiin niin, että annetaan otsikolle oma id, jota käytetään valitsimena. Toinen vaihtoehto olisi valita tietyn elementin lapsielementti esimerkiksi, jos haluttaisiin div-elementin sisältä otsikko tehtäisiin valinta *$("div > h1");*. Tämä valitsin tosin valitsee kaikki div-elementtien sisässä olevat h1-elementit, joten parempi olisi määritellä *$("#container > h1")*, jolloin muutos tapahtuu vain tietyn id:n lapsielementteihin.

Itse valitsimien käyttö ei suoraan helpota koodausta, vaan jQueryssä on myös valmiita metodeja, joilla voidaan käsitellä selaimen tapahtumia (event). Front-end koodaus perustuu JavaScriptin osalta siihen, että kuunnellaan ja käsitellään selaimen tapahtumia, joihin sitten reagoidaan koodin mukaan (kuva 14). Esimerkiksi napin painaminen voi aiheuttaa Ajax-kutsun palvelimelle ja käyttäjälle haetaan verkkokaupan tuotteiden listaus. Tai jos käyttäjä vie hiiren jonkin elementin päälle, muutetaan elementtiä.

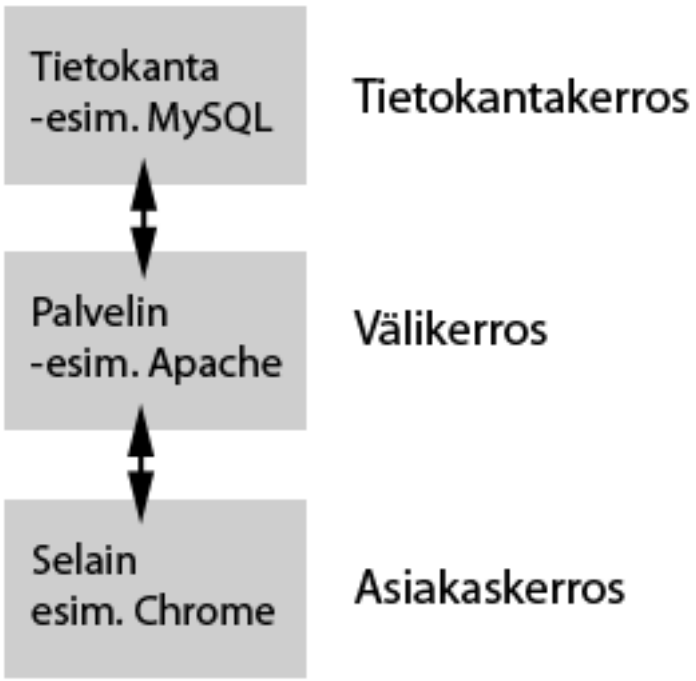
```
$("#nayta_tuote").on("click",function(){
    //Ajettava koodi tänne

});
```

KUVA 14. jQuery tapahtuma klikatessa elementtiä

Yksi vaihtoehto tehdä front-end koodausta on käyttää jotain valmista JavaScript ohjelmistokehystä kuten AngularJS. AngularJS on avoimeen lähdekoodiin perustuva, Googlen ylläpitämä ohjelmistokehys, joka on tarkoitettu yksisivuisten web-sovellusten tekemiseen. AngularJS on suosituin JavaScript ohjelmistokehys (Most Popular JavaScript Frameworks 2015). AngularJS mahdollistaa MVC-arkkitehtuurin käytön sovellusten teossa. MVC-arkkitehtuurista lisää seuraavassa luvussa. AngularJS yksi periaate on modulaarisuus ja sillä pyritään siihen, että kaikki sovelluksen eri osat liitetään moduuleina sovellukseen (Kuivamäki 2015, 19). Tällä pyritään helpottamaan koodin ylläpitoa.

2.4 Kolmikerrosmalli ja MVC-malli

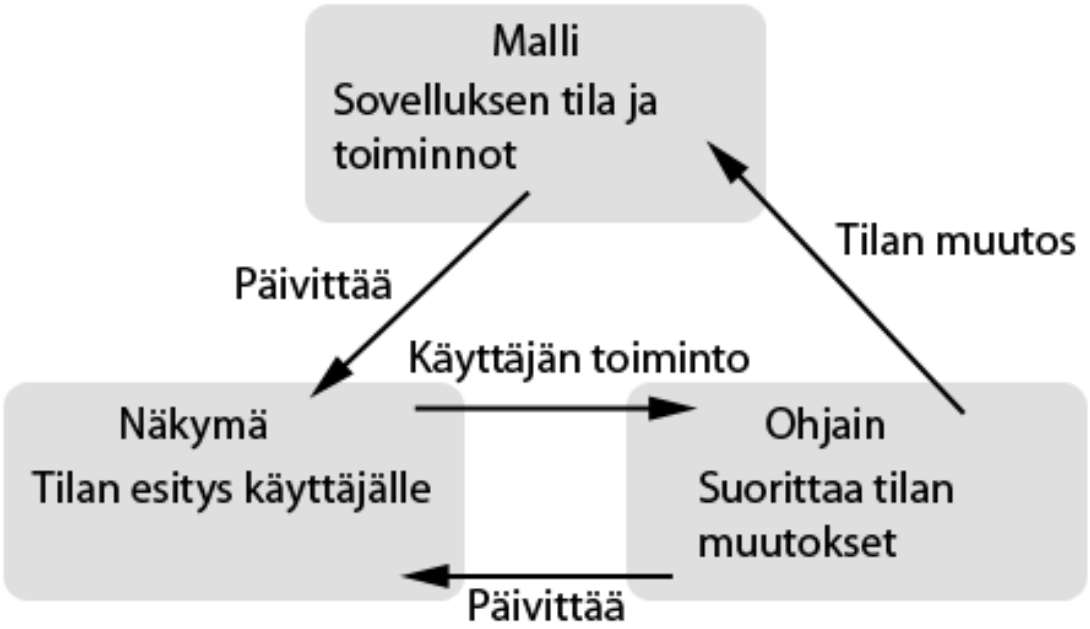
Kolmikerrosmalli (three-tier model) on verkkopalvelun arkkitehtuurimalli, jossa ajatellaan palvelun koostuvan kolmesta kerroksesta (kuva 15). Kolmikerrosmalli koostuu seuraavasti osista: asiakaskerros eli käyttäjän selain, välikerros jossa pyörii palvelin, sekä tietokantakerros (N-Tier Data Applications Overview 2015). Tietokantapohjaiset sovellukset on pitkälti toteutettu käyttäen kolmikerrosmallia.



KUVA 15. Kolmikerrosmalli

Kolmikerrosmallissa asiakaskerrokselta tulee käskyjä välikerrokselle, joka taas lähettää pyynnön tietokantakerrokselle. Tietokanta palauttaa vastauksen palvelimelle, joka näyttää tiedot asiakkaalle. Koska asiakas ei ole suoraan yhteydessä tietokantaan, voidaan palvelimella esimerkiksi ensin tarkistaa onko asiakkaalla oikeutta hakea tietoa, jota se on pyytänyt näytettäväksi.

MVC (Model View Controller)-arkkitehtuurimallissa sovellus jaetaan kolmeen osaan: malliin (model), näkymään (view) ja ohjaimeen (controller). MVC-mallia käytetään usein käyttöliittymien teossa ja siten se on suosittu malli myös websovelluksissa (MVC architecture 2015). Kuvassa 16 on esitetty MVC-arkkitehtuurin eri osat.



KUVA 16. MVC:n perusajatus

Malli määrittää mitä tietoa sovellus sisältää, jos tila muuttuu niin malli ilmoittaa näkymään, jolloin käyttäjän näkemä sisältö muuttuu. Myös ohjain voi päivittää näkymän tarvittaessa (MVC architecture 2015). Näkymä esittää tiedot käyttäjälle, jotka se saa mallilta. Joten näkymä on riippuvainen sovelluksen tilasta ja mallin tiedoista. Ohjain lukee käyttäjän antamia syötteitä, esimerkiksi tekstin syöttöä ja muokkaa mallin sisältämää tietoa, joka taas vaikuttaa näkymään. Eli aina kun sovelluksen tila muuttuu, niin näkymä päivittyy.

HTML5-sovelluksessa näkymä ja ohjain toteutetaan yleensä selainmoottorissa (Lehdonvirta & Korpela 2013, 67). Voidaan myös ajatella, että HTML-dokumentin puurakenne on osa mallia. MVC-arkkitehtuurin mukaisten sovellusten tekoon on olemassa omat työkalunsa kuten AngularJS tai Ember. MVC-malliin on myös erilaisia variaati-

oita, joissa osassa ohjain ei päivitä näkymää vaan se tehdään aina mallin kautta (Google Developers 2015).

MVC-arkkitehtuurin tarkoituksena on luoda uudelleenkäytettävää koodia, sekä pitää koodi helposti muokattavana (MVC architecture 2015). Samalla erotetaan näkymä toimintamallista, jolla mahdollistetaan samanaikainen työskentely sovelluksen ulkoasun ja toimintojen kanssa.

3 ILMAVIRTALASKURISOVELLUS

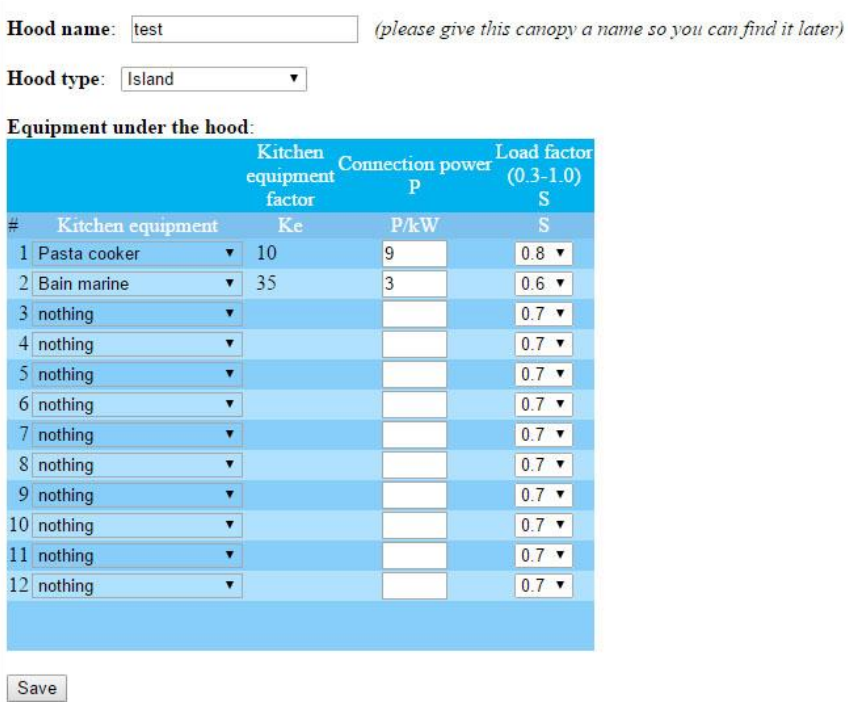
Jeven Oy on ammattikeittiöiden ilmanvaihtolaitteita valmistava ja suunnitteleva yritys, jonka pääkonttori sijaitsee Mikkelissä. Ilmavirtalaskuri on aluksi yrityksen sisäiseen käyttöön tuleva apuväline, jolla voidaan laskea ilmavirtoja huuville eri standardien mukaan. Laskuri on jo aloitettu aikaisemmin, mutta sitä ei ole ikinä tehty valmiiksi. Sovellus on myös ollut alkeellisempana käytössä.

3.1 Ilmavirtalaskurin nykytilanne

Vanha laskuri pohjautuu kolmikerros malliin ja käyttää mysql-funktioita, jotka eivät enää ole tuettuina uusissa PHP-versioissa (MySQL Function 2015). Tavoitteena on päivittää vanha sovellus (kuva 17) ja käyttää mahdollisimman paljon vanhaa koodia hyväksi uuden kehittämisessä.

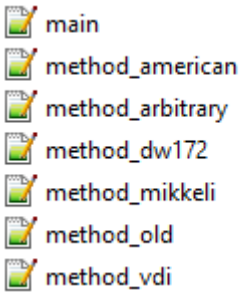
KUVA 17. Vanhan laskurin etusivu

Vanhaa ohjelmaa ei ole suunniteltu käyttäjäystävälliseksi ja käyttöliittymä on kehno. Vanhassa laskurissa toimintaperiaate oli seuraava: valikosta valitaan millä standardilla halutaan laskea ilmavirta, jonka jälkeen käyttäjä antaa tarvittavat tiedot ja tallentaa huuvan tietokantaan (kuva 18). Tosin ”save”-nappi ei toimi, koska sille ei ole tehty mitään toimintoja.



KUVA 18. Vanha laskuri, Mikkeli method

Tiedostojen nimeäminen on tehty hyvin vanhaan projektiin ja tiedostojen nimistä voi päätellä, mitä kyseisessä PHP-tiedostossa pitäisi olla (kuva 19). Tiedoston sisässä koko projekti sitten hajoaa todella vaikeasti ymmärrettäväksi ja hitaasti korjattavaksi, josta lisää seuraavaksi.



KUVA 19. Vanhan sovelluksen tiedostonimet

Projektissa on jokaisessa PHP-tiedostossa sekaisin JavaScriptillä luodut skriptit, jotka liittyvät laskentaan tai lomakkeen validointiin, sekä PHP-koodit, joilla luodaan koko näkyvä HTML-dokumentti. Samoin tyylit annetaan suoraan koodissa eikä niitä ladata erillisestä CSS-tiedostosta. (kuva 20).

```
function notvalidnumber(n) {
    if (n==null) return true;
    if (n=='') return true;
    n=n+'';
    if (!n.match(/^\d+$/)) return true;
    return false;
}</script>
<h1>Create a hood using the Mikkeli method</h1>
<form name='form' method='post' onsubmit='return
<input type='hidden' name='checked' value='no'>
<b>Hood name</b>:   <input type='text' name=
<br>
<b>Hood type</b>:   <select name='type'>
<option value='0' selected>choose hood type...</o
<option value ='1'>Wall</option>
<option value ='2'>Island</option>
</select>
<br>
<br>
<b>Equipment under the hood</b>:
<table border=0 style='border-collapse:collapse;'
<tr bgcolor='#00B2EE'><td> </td><td> </
<tr bgcolor='#7EC0EE'><td>#</td><td style='color:
```

KUVA 20. Vanhan sovelluksen PHP-koodia

Vanhaa PHP-koodia ei ole kommentoitu kunnolla ja muuttujien nimetkään eivät auta koodin ymmärtämistä. Kuvassa 21 näkyy laitteita, jotka on laitettu arrayksi eli taulukoksi, josta voidaan sitten myöhemmin poimia indeksin mukaan tiedot laitteesta, tosin tietoja ei ole mitenkään kommentoitu eli on vaikea tietää mitä lukuja laitteille on annettu tähän taulukkoon. Taulukosta tulee myös ilmi, että se ei ole täysin valmis, koska osalla laitteista on arvoina 999999, kun taas muilla arvot ovat alle tai yli yksi.

```
//FLAME COOKING
$ap[101] = array('Griddle (mild steel)',0.3,0.25);
$ap[102] = array('Griddle (chrome)',0.45,0.4);
$ap[103] = array('Conveyor pizza oven',0.45,0.4);
$ap[104] = array('Heavy duty deep fat friex',0.5,0.45);
$ap[105] = array('Heavy duty bratt pan',0.55,0.45);
$ap[106] = array('Solid top oven range',0.6,0.51);
$ap[107] = array('Upright or chain broiler',0.75,0.55);
$ap[108] = array('Salamander or steakhouse grill',0.75,0.55);
$ap[109] = array('Rotisserie',0.75,0.55);
$ap[110] = array('Chargrill/charbroiler',0.95,0.52);
$ap[111] = array('Chinese wok range',1.1,999999);
$ap[112] = array('Chinese wok range (induction)',999999,0.4);
$ap[113] = array('Mesquite grill',1.2,999999);
```

KUVA 21. Laitteita vanhasta sovelluksesta

Taulukossa laitteille on annettu tietyt ennalta määrätyt paikat, mihin ne sijoittuvat. Tämä ei helpota taulukon käyttöä, koska pitäisi ennalta tietää missä kohdin taulukkoa laitteet sijaitsevat, jotta tietoja voi käyttää oikein.

3.2 Uusi sovellus

Uuden laskurin kehityksessä otetaan huomioon, että sovelluksen tulee olla responsiivinen, koska sovellusta on voitava käyttää myös mobiililaitteella. Sovelluksen värimaailmassa taas on käytettävä Jevenin värejä, jotka on määritelty yrityksen antamassa ohjeistossa. Uusi sovellus saatetaan myös tulevaisuudessa liittää yrityksen sivuille, joka on tehty Wordpress-alustalla, joten senkin takia värimaailman on oltava sama. Tällöin koodiin saatetaan joutua tekemään pieniä muutoksia, koska tällä hetkellä sovelluksesta on tulossa itsenäinen osa.

Uusilla sivuilla kaikki JavaScript-koodit on toteutettu jQueryn avulla, joten tässäkin projektissa käytetään jQueryä. Tavoitteena on myös luopua vanhasta kolmikerrosmalliin pohjautuvasta tekniikasta ja tehdä sovellus, joka mukailee MVC-arkkitehtuurimallia toimintaperiaatteeltaan. MySQL-tietokannasta on myös tavoitteena luopua ja alkaa käyttää JSON-tiedostoa tarvittavien tietojen tallentamiseen tai lataamiseen palvelimelta. Mahdollisimman paljon käskyjä pyritään suorittamaan käyttäjän omassa selaimessa, jotta palvelinta ei tarvitsisi rasittaa niin paljoa ja samalla sovelluksesta saadaan toimiva ilman internetyhteyttä. Tavoitteena on myös kasata projektin

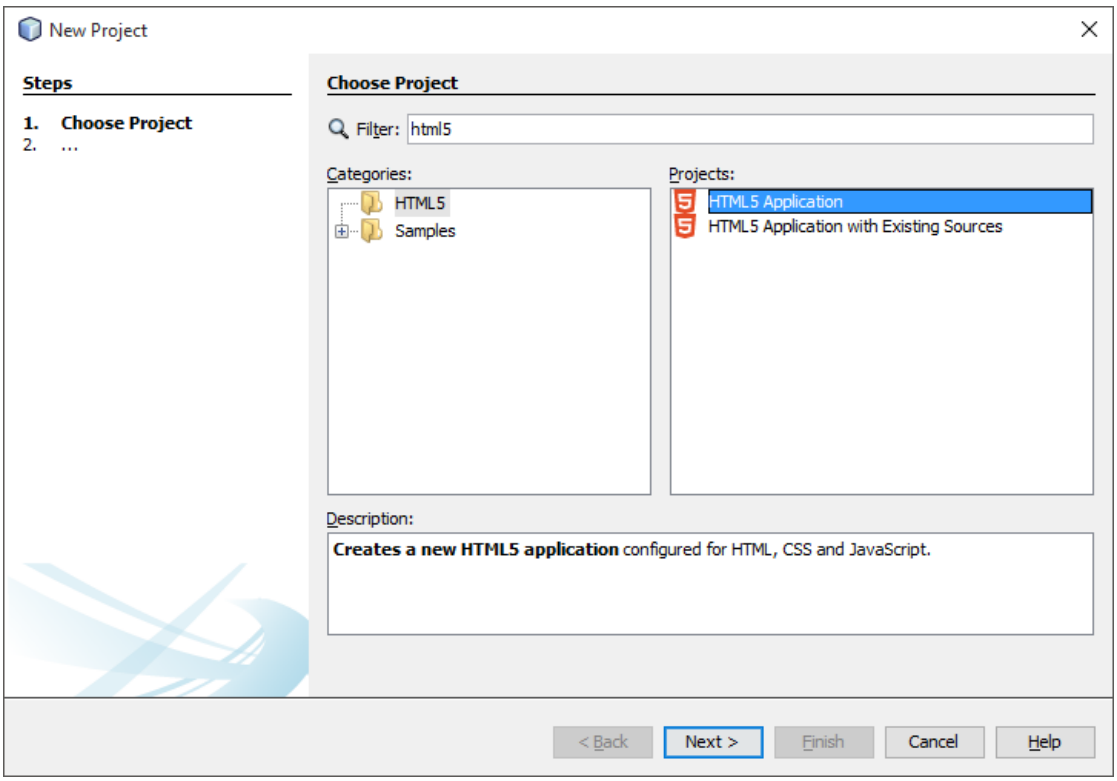
tiedostot helpommin ymmärrettävään rakenteeseen, jossa tyylit, skriptit ja itse sisältösivut ovat eroteltuina kokonaisuuksina.

Sovelluksen tekemisessä otetaan myös huomioon muuttujien ja funktioiden nimeäminen sekä koodin kommentointi, jotta jatkokehitys ja koodin muokkaus olisi helpommin tehtävissä kuin vanhalla sovelluksella. Kehityksessä on myös otettava huomioon ulkoasun selkeys ja sovelluksen käytön ei pitäisi vaatia erillistä kurssia.

3.3 Toteutus

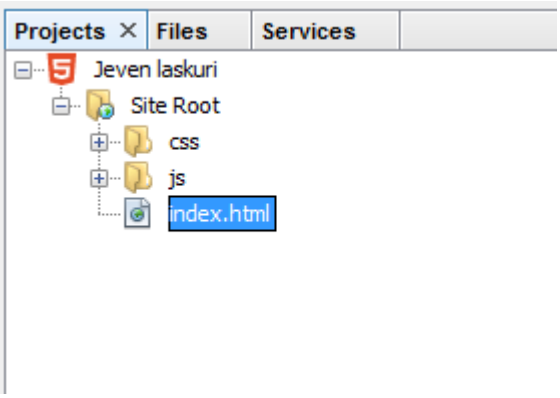
Sovelluksen kehittämisessä käytettiin NetBeans IDE 8.0.2-versiota, joka on avoimeen lähdekoodiin perustuva kehitysympäristö, jolla on mahdollista kehittää sovelluksia eri ohjelmointikielillä. NetBeansillä pystyy kirjoittamaan, kääntämään, debuggaamaan ja ottamaan käyttöön sovelluksia. Siihen voidaan myös tuoda moduuleja, jolloin kehitysympäristöä pystyy laajentamaan.

Projekti aloitettiin luomalla uusi HTML5-projekti NetBeansillä (kuva 22). Projektin luontivaiheessa voidaan myös tuoda jokin valmis sivupohja projektiin tai ladata tietty pohja. Lopuksi projektiin voidaan myös ladata JavaScript-kirjastoja.



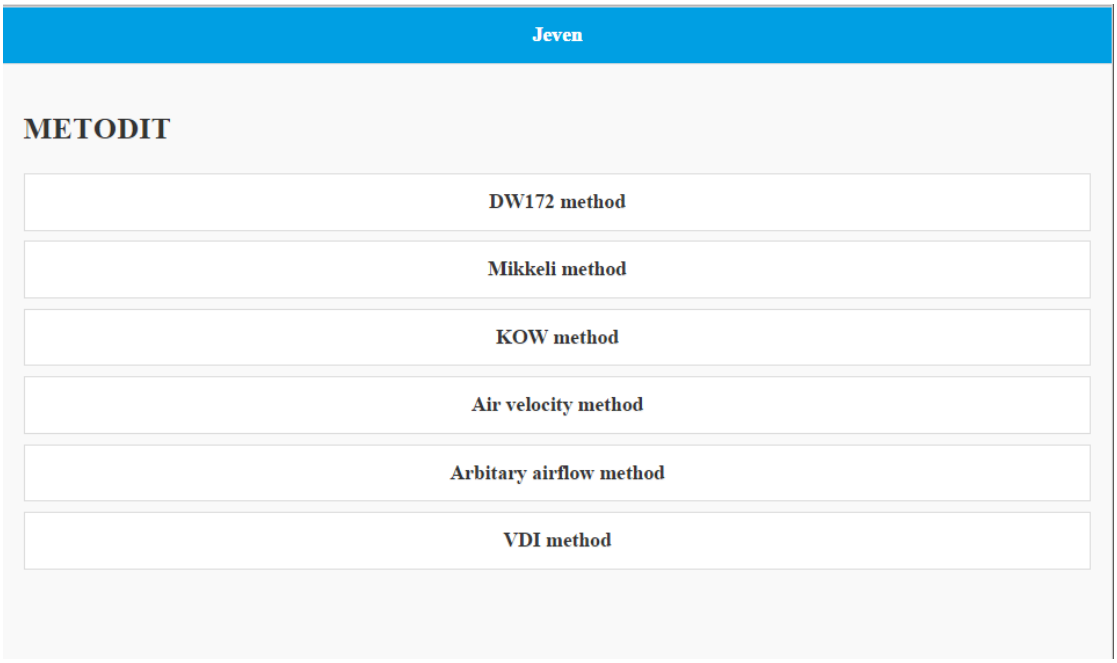
KUVA 22. Uuden projektin luominen

Ohjelma luo automaattisesti selkeän rakenteen, jossa JavaScript-tiedostot ja CSS-tiedostot ovat eri kansioissa kuin HTML-tiedostot (kuva 23). Käyttäjän on myös helppo tuoda tai luoda projektin kansioihin uusia tiedostoa, jolloin rakenne pysyy kuitenkin samana. Projektiin tuotiin ulkopuolisina tiedostoina jQuery-kirjasto ja FontAwesome, sekä valmis grid CSS-tiedostona, jolla voidaan jakaa sivua paremmin osiin antamalla tietty luokka elementeille. FontAwesomen avulla sovelluksessa voidaan käyttää erilaisia ikoneita, joita ei normaalisti olisi käytettävissä.



KUVA 23. Uuden projektin oletusrakenne

Uuden sovelluksen etusivu (kuva 24) on samantapainen kuin vanhassakin eli käyttäjän on ensin valittava haluamansa tapa, jolla ilmavirtoja tullaan laskemaan, jonka jälkeen sovellus vaihtaa näkymän haluttuun laskuriin. Erona vanhaan on myös erilaiset värimäärittelyt elementeille ja vanhassa sovelluksessa olevat linkit on muutettu napin kaltaisiksi elementeiksi.

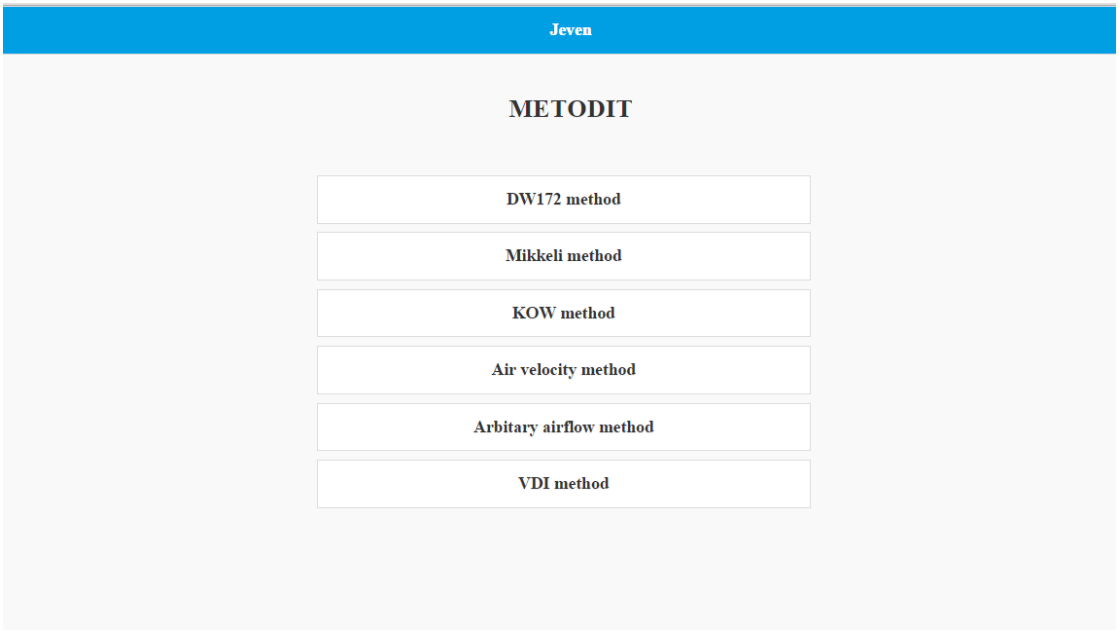


KUVA 24. Uuden sovelluksen etusivu

Kuvasta huomataan, että valikon napit ovat yhtä leveitä kuin pohjaelementtikin. Tämä on toimiva ratkaisu mobiililaitteella, jolla pieniin nappeihin voi olla vaikea osua, jolloin käyttömukavuus laskee huomattavasti, mutta työpöytäkäytössä suuremmalla näytöllä sovellusta käytettäessä laidasta laitaan menevät napit eivät ole hyvännäköisiä. Joten elementeille on määriteltävä oma leveys mobiililaitteella käytettäessä ja työpöytä käytössä. Sovelluksessa tämä on toteutettu CSS3-ominaisuudella *@media*, jolla on helppo säätää laitekohtaisia tyylejä.

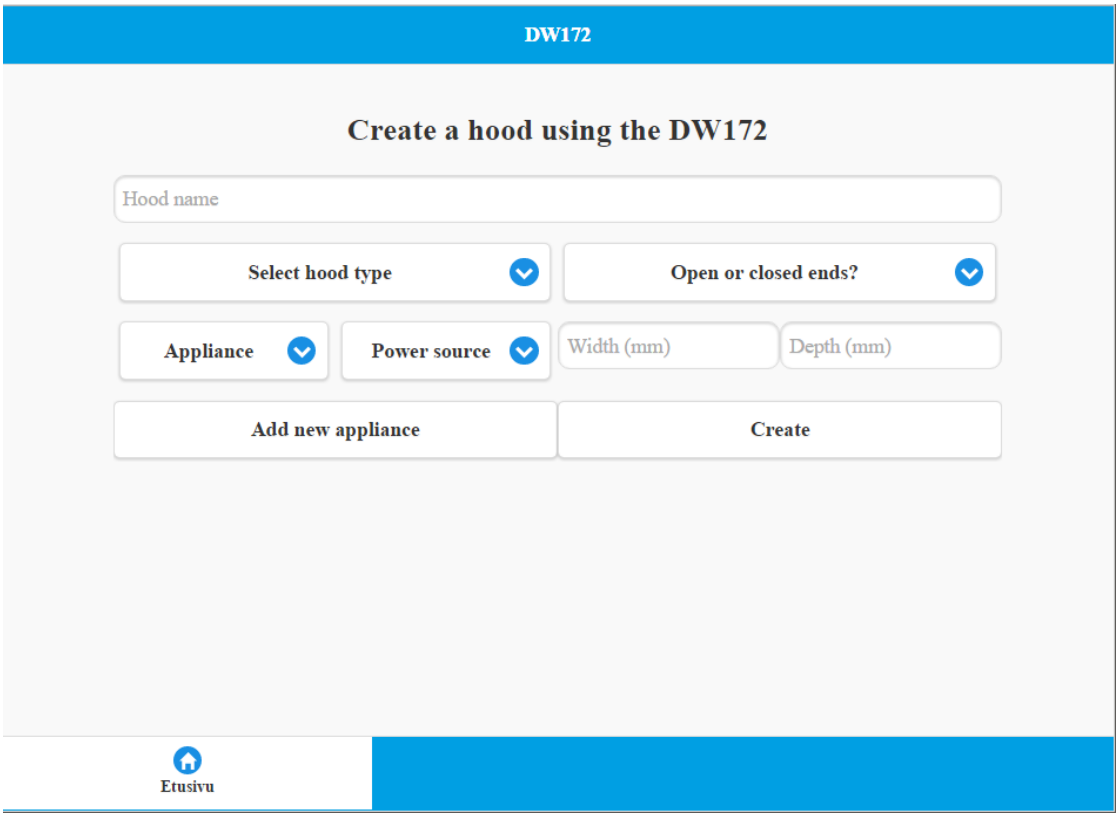
```
@media screen and (min-width: 768px){
    .ui-btn-main{
        width: 25%;
    }
    html{
            text-align: center;
            width:80%;
            margin-left:auto;
            margin-right:auto;
    }
}
@media only screen and (max-width: 768px) {
    /* For mobile phones: */
    .ui-btn-main{
        width: 100%;
    }}
```

Edellä olevalla tyylimäärittelyllä siis säädetään nappielementtien maksimileveydeksi 25 % ruudun leveydestä kaikilla muilla paitsi mobiililaitteilla, joilla leveys on 100 %. Tekstit määritetään myös keskelle dokumenttia, jolloin lopputulos on tyylikkäämmän näköinen ja napit eivät leviä koko ruudun leveydelle (kuva 25).



KUVA 25. Uuden sovelluksen korjattu etusivu (työpöytä sovellus)

Kun käyttäjä klikkaa jotain metodia, niin sovellus vaihtaa näkymän kyseiseen laskentataulukkoon. Laskentataulukossa (kuva 26) tulee esiin, miten käytin valmista gridiä hyväksi tehdessäni sovellusta. Kaikilla sivuilla on sama rakenne: ylhäällä tietyn värinen sininen palkki, joka toimii sivun headerina, ja alhaalla on tarvittaessa navigointi takaisin edelliselle sivulle. Gridin avulla elementit on jaettu sivulle vierekkäin ja viereiset elementit ovat aina saman levyisiä, jolloin ulkoasu pysyy siistinä.



KUVA 26. Uuden sovelluksen DW172 (työpöytä sovellus)

Laskentametodissa kysytään tarvittavat tiedot, jotta voidaan laskea arvioitu ilmavirta huuvalle. Käyttäjää pyydetään myös nimeään huuva, koska nimen perusteella laskelma voidaan tallentaa ja tarvittaessa löytää. Lomakkeeseen on myös mahdollista lisätä uusi laite, koska laitteita voi olla useampi. Tällöin sovellus vain muokkaa DOMia ja luo sinne uuden rivin lomakkeeseen, jossa on samat kentät. Lähes kaikki tiedot lomakkeeseen saadaan JSON-tiedostoista.

```
//get appliances from json file
$.ajax({
    url: "appliances.json",
    type: "GET",
    dataType: "json"
}).done(function (appliances) {
    for (var i = 0; i < appliances.length; i++) {
        $("#apps_dw").append("<option>" + appliances[i][0] + "</option>");
    }
})
      .fail(function () {

      });
```

Koodissa tehdään Ajax-kutsu, jolla haetaan laitteet JSON-tiedostosta, josta ne sitten jQueryn append -metodilla lisätään elementtiin <*select*> vaihtoehtoina. Näin pidetään HTML-tiedosto siistimpänä ja muutoksia on helpompi tehdä JSON-tiedostoon.

Laitteen leveys ja syvyys kentät myös tarkistetaan, jos käyttäjä painaa *Create*-nappia. Arvot eivät saa olla kirjaimia ja vain positiiviset luvut kelpaavat mitoiksi. Tarkistus suoritetaan omalla funktiollaan, joka palauttaa *true* tai *false* riippuen tarkistuksesta. Funktiossa tarkistetaan funktiolle tullut arvot säännöllisen lausekkeen (Regular Expression) avulla, jolle on määritelty arvot, että vain positiiviset luvut kelpaavat, joten vaikka käyttäjä pystyisikin liittämään kenttään tekstiä, ei se mene läpi lopullisesta tarkastuksesta. Jos kaikki arvot ovat ok, niin ohjelma laskee ja näyttää käyttäjälle tarvittavan ilmavirran kyseisille laitteille.

```
function checkValues(number) {
    if (number > 0) {
        var regex = new RegExp("^(?: *)?[0-9]+$");
        if (regex.test(number)) {
            return true;
        }
        else {
            return false;
        }
    } else {
        return false;
    }
}
```

Sovellus näyttää virheilmoituksen käyttäjille, joilla ei ole JavaScript käytössä ja pyytää käyttäjää sallimaan JavaScriptin käytön sivulla, koska ilman JavaScriptiä sovellus ei toimi oikein. Tällä hetkellä ohjelma ei tallenna mitään tietoja JSON-tiedostoon, mutta se ominaisuus on helposti lisättävissä tulevaisuudessa eli käyttäjälle vain näytetään tarvittava tieto.

4 PÄÄTÄNTÖ

HTML5-sovelluksen teossa on siis huomioita monta eri asiaa. Tärkeimpänä on oikean tekniikan valitseminen, sekä ohjelmistokehyksen valinta, jos sellaista haluaa käyttää. Myös eri arkkitehtuurimallit on hyvä ymmärtää, koska tämä helpottaa sovelluksen suunnittelua. Usein eniten esiin nousevat termit MVC ja Single-Page Application, joten kehittäjän tulee tietää mitä nämä termit tarkoittavat.

Opinnäyte työn tekeminen osoittautui haastavaksi ja monimuotoiseksi projektiksi. Haastavuutta lisäsi vanhan sovelluksen huonot koodit ja oma tietämättömyys LVI-

alasta. Toisaalta työ onnistui ja jonkinlainen toimiva versio sovelluksesta saatiin aikaiseksi, vaikka aikaa meni jonkin verran hukkaan tutkiessani vanhan sovelluksen koodeja. Lisäksi koska sovellukselle ei ollut määritelty tarkempaa suunnitelmaa, oli vaikea tietää millainen on valmis sovellus. Eli tarvitseeko sovelluksen esimerkiksi tallentaa tietoa vai ei, ja onko tarvetta päästä tulevaisuudessa muokkaamaan vanhoja laskuja.

Olen kuitenkin tyytyväinen omaan suoritukseeni, vaikka sovellusta joudutaankin luultavasti jatkokehittämään ja lisäämään uusia ominaisuuksia, sekä korjaamaan laitteiden tietoja. Joitain pieniä muutoksia myös joudutaan tekemään, jos sovellus joskus lisätään Wordpress-sivuille. Jos tulevaisuudessa päädytään ratkaisuun, ettei sovellusta tulla koskaan lisäämään sivuille, olisi se parasta kehittää suoraan HTML5-sovelluskehitykseen soveltuvalla ohjelmistokehyksellä esimerkiksi AngularJS:n avulla, jolloin sovelluksesta saadaan suoraan MVC-mallin mukainen yksisivuinen sovellus. Toisaalta sovellus toimii tällä hetkellä hyvin mobiililaitteella ja työpöytä käytössä, joten responsiivisuus onnistui mielestäni hyvin.

Työ pysyi kuitenkin aikataulussa ja uskon, että yritykselle on hyötyä sovelluksesta, jonka sain aikaiseksi. Uskoisin myös, että yritys tulee kehittämään sovellusta lisää, jotta siihen saadaan ominaisuudet eri standardien ylläpitoon ja CRUD-ominaisuudet laitteiden ylläpidolle.

LÄHTEET

Ajax. 2015. Mozilla Developer Network. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/AJAX. Päivitetty 12.6.2015. Luettu 27.9.2015.

Beal, Vangie 2015. Application (application software). WWW-dokumentti. http://www.webopedia.com/TERM/A/application.html. Ei päivitystietoa. Luettu 18.9.2015.

CSS3. 2015. Mozilla Developer Network. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3. Päivitetty 2.9.2015. Luettu 10.10.2015.

Girdley, Michael 2015. Front-end vs Back-end. Blogi. http://codeup.com/different-types-of-programmers-front-end-vs-back-end/. Päivitetty 24.1.2014. Luettu 3.10.2015.

Document Object Model (DOM). 2015. Mozilla Developer Network. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model. Päivitetty 16.9.2015. Luettu 23.10.2015.

Google Developers 2015. MVC Architecture. WWW-dokumentti. https://developer.chrome.com/apps/app_frameworks. Ei päivitystietoa. Luettu 24.10.2015

HTML5. 2015. Mozilla Developer Network. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. Päivitetty 4.9.2015. Luettu 10.10.2015.

jQuery Callback Functions. 2015. W3Schools. WWW-dokumentti. http://www.w3schools.com/jquery/jquery_callback.asp. Ei päivitystietoa. Luettu 27.9.2015.

jQuery. 2015. jQuery. WWW-dokumentti. https://jquery.com/. Ei päivitystietoa. Luettu 23.10.2015.

JSON. 2015. JSON Group. WWW-dokumentti. http://www.json.org/. Ei päivitystietoa. Luettu 11.10.2015.

Kavourgias, Callie 2015. What's the Difference Between the Front-End and Back-End. Blogi. http://blog.digitaltutors.com/whats-difference-front-end-back-end/. Päivitetty 29.1.2015. Luettu 3.10.2015.

Kuivamäki, Tomi 2015. Mobiilisovelluksen toteustus ionic-sovelluskehyksessä Case:Aika24. Mikkelin Ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö. PDF-dokumentti. https://www.theseus.fi/bitstream/handle/10024/91715/ontTomiKuivamaki.pdf?sequence=1. Päivitetty 8.5.2015. Luettu 24.10.2015.

Lengstorf, Jason 2015. JSON: What It Is, How It Works, & How to Use It. Blogi. http://www.copterlabs.com/blog/json-what-it-is-how-it-works-how-to-use-it/. Ei päivitystietoa. Luettu 11.10.2015.

Lehdonvirta, Pyry & Korpela, Jukka K. 2013. HTML5 sovellusalustana. Helsinki: RPS-yhtiöt.

Most Popular JavaScript Frameworks. 2015. I'm programmer. WWW-dokumentti. http://www.improgrammer.net/most-popular-javascript-frameworks-2015/. Päivitetty 17.4.2015. Luettu 24.10.2015.

MVC architecture. 2015. Mozilla Developer Network. WWW-dokumentti. https://developer.mozilla.org/en-US/Apps/Build/Modern_web_app_architecture/MVC_architecture. Päivitetty 14.3.2015. Luettu 24.10.2015.

MySQL Function. 2015. The PHP Group. WWW-dokumentti. https://secure.php.net/manual/en/function.mysql-connect.php. Ei päivitystietoa. Luettu 28.10.2015.

N-Tier Data Applications Overview. 2015. MSDN. WWW-dokumentti. https://msdn.microsoft.com/en-us/library/bb384398.aspx. Ei päivitystietoa. Luettu 24.10.2015.

Resize. 2015. Mozilla Developer Network. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/CSS/resize. Päivitetty 22.9.2015. Luettu 10.10.2015.

REST API Quick Tips. 2015. Rest API Tutorial. WWW-dokumentti. http://www.restapitutorial.com/lessons/restquicktips.html. Ei päivitystietoa. Luettu 2.10.2015.

Rodriguez, Alex 2015. RESTful Web services: The basics. WWW-dokumentti. http://www.ibm.com/developerworks/library/ws-restful. Päivitetty 9.2.2015. Luettu 26.9.2015.

Rouse, Margaret 2010. Learn IT: Software development. WWW-dokumentti. http://whatis.techtarget.com/reference/Learn-IT-Software-development. Päivitetty 4.2010. Luettu 18.9.2015.

Van Kesteren, Anne, Aubourg, Julian, Song, Jungkee & R.M.Steen, Hallvord 2014. XMLHttpRequest Level 1. WWW-dokumentti. http://www.w3.org/TR/XMLHttpRequest. Päivitetty 30.1.2014. Luettu 27.9.2015.

Wales, Michael 2014. 3 Web Dev Careers Decoded: Front-End vs Back-End vs Full Stack. Blogi. http://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html. Päivitetty 8.12.2014. Luettu 3.10.2015.

Wasson, Mike 2013. ASP.NET – Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. WWW-dokumentti. https://msdn.microsoft.com/en-us/magazine/dn463786.aspx. Päivitetty 11. 2013. Luettu 20.9.2015.

Website Front-end and Back-end. 2015. 3ny Technology. Blogi. http://www.3nytechnology.com/website-frontend-and-backend/. Päivitetty 4.8.2015. Luettu 4.10.2015.

What can PHP do?. 2015. The PHP Group. WWW-dokumentti. https://secure.php.net/manual/en/intro-whatcando.php. Ei päivitystietoa. Luettu 16.10.2015.

What is PHP-GTK?. 2015. The PHP Group. WWW-dokumentti. http://gtk.php.net/faq.php#1. Päivitetty 8.4.2015. Luettu 16.10.2015.

Zend Technologies 2015. About. WWW-dokumentti. http://framework.zend.com/about/. Ei päivitystietoa. Luettu 23.10.201