LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Asiakaspohjainen käyttöliittymä yritykselle

Case: Lahden ATK Klinikka

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikka
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2017
Anu Halmejärvi

Lahden ammattikorkeakoulu
Tietotekniikka

HALMEJÄRVI, ANU:

Asiakapohjainen käyttöliittymä yritykselle
Case: Lahden ATK Klinikka

Ohjelmistotekniikan opinnäytetyö, 42 sivua

Kevät 2017

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa asiakastietokanta sekä web-pohjainen käyttöliittymä Lahden ATK Klinikalle. Lahden ATK Klinikka voisi näin tallettaa asiakkaiden sekä heidän laitteidensa tiedot palvelimelle, mikä helpottaa työntekijöiden työtä ja minimoi mahdollisia riskejä tietojen katoamiselle.

Tietokanta suunniteltiin ensin relaatiokaavioiden avulla sekä tämän jälkeen toteutettiin MySQLi:llä erillisessä HeidiSQL-ohjelmassa. Web-käyttöliittymän suunnittelussa käytettiin apuna rautalankamalleja, jotka tehtiin Pencil Project –ohjelmassa. Ohjelmointi tapahtui Sublime Text 2 -editorissa käyttäen Laravel PHP -kehystyöalustaa.

Projektissa vältyttiin suurilta ongelmilta. Laravelin opettelu vei aikaa projektin aloittamiselta, minkä takia se ei valmistunut niin nopeasti kuin oli ajateltu. Projektia tehtiin koulun ohella, mikä vaikutti myös aikataulusta viivästymiseen. Suurin ongelma oli palvelimen kanssa, koska se lopetti ajoittain toimintansa. Palvelin oli yrityksen työntekijän vastuualueella.

Ohjelmaa testattiin yrityksen työntekijöillä. Tämän ansiosta käyttöliittymästä tuli toimiva sekä helppokäyttöinen kokonaisuus. Lopputulos oli yritykselle mieleinen.

Asiasanat: tietokanta, käyttöliittymä, Laravel, MySQL

Lahti University of Applied Sciences
Degree Programme in Information Technology

HALMEJÄRVI, ANU:

Client-based user interface for a company Case: Lahden ATK Klinikka

Bachelor's Thesis in software engineering,

42 pages

Spring 2017

ABSTRACT

The purpose of this thesis was to plan and produce a customer database and a web-based user interface for Lahden ATK Klinikka. In this way Lahden ATK Klinikka could save information on its customers and their devices to the server. It would make employees' work easier and minimize possible risks of losing data.

The database was first planned with the relational model and then it was produced in MySQLi with a separate program named HeidiSQL. Wireframes were used to design the web-based user interface in the Pencil Project –program. Programming was done in the Sublime Text 2 - editor using the Laravel PHP framework.

Major problems were avoided in this project. Learning about Laravel took time at the beginning of the project. This was the reason why the program was not completed when it should have. The project was made along with school, which also had an effect on delay of the schedule. The biggest problem was with the server, because it stopped working occasionally. One of the employees was responsible for the server.

The program was tested with the employees of the company. This is why the user interface became functional and user friendly. The final result was satisfying to the company.

Key words: database, user interface, Laravel, MySQL

TERMISTÖ

CSS	*Cascading Style Sheets*, tyyliohjeiden laji. CSS-tiedostossa määritellään sivun tyyli.
Foreign key	Viiteavain, jota käytetään, kun halutaan estää taulujen välisten yhteyksien tuhoutuminen.
Heuristiikka	Lista sääntöjä tai ohjeita.
HTML	*Hypertext Markup Language*, kuvauskieli hypertekstin kuvaamiseen.
ID	*Identifier*, tunniste. Käytetään usein tietokantoja tehdessä.
Localhost	Standardi nimitys käyttäjän tietokoneelle.
Metadata	Tietoa jostakin tietoyksiköstä, esimerkiksi valokuvan koko, päivämäärä ja tiedoston nimi.
MVC-sovelluskehys	*Model-View-Controller*, ohjelmistoarkkitehtuurityyli, jota käytetään etenkin graafisten käyttöliittymien tuottamisessa.
MySQL	Maailman toiseksi suurin relaatiotietokantaohjelmisto.
Objekti	Olio-ohjelmoinnissa käytetään objekteja eli olioita. Olio on tietorakenne, joka koostuu tiedosta ja sitä käsittelevästä toiminnallisuudesta.
ORM	*Object Relational Mapper*, eli oliopohjainen kartoitin.
PDO	*PHP Data Objects,* tietokannan ohjelmointimalli.
PHP	*PHP: Hypertext Preprocessor*, ohjelmointikieli.
Primary key	Pääavain, joka tunnistaa jokaisen tietueen tietokantataulussa.

Relaatio	Relaatioita käytetään, kun halutaan selvittää suhteita kahden asian välillä.
Tietokanta	Tietotekniikassa tietokantaan varastoidaan dataa, joka on toisiinsa yhteydessä.
Tietokantataulu	Tietokantataulut sijaitsevat tietokannassa. Tauluihin voidaan määritellä rivejä ja tietoa, jotka halutaan säilyttää.
URI	*Uniform Resource Identifier*, tietyn tiedon paikan kertova merkkijono.

SISÄLLYS

1 JOHDANTO

Projektin toimeksiantaja on Lahden ATK Klinikka, joka on vuonna 2013 perustettu yritys ja se keskittyy tietokoneiden huoltoon ja korjaukseen. Opinnäytetyö aloitettiin syyskuussa 2014. Yrityksen toimenkuvaan kuuluu tietokoneiden sekä muiden ATK-laitteiden vianhaku, korjaus sekä huolto. Ongelmatilanteet voivat liittyä niin asiakkaan tietokoneen ulkoisiin osiin kuin ohjelmiinkin. Yrityksessä tehdään tämän lisäksi myös puhelinhuoltoja. Työntekijöitä yrityksessä on noin 5. Lukumäärä vaihtelee, sillä pääosin työntekijät ovat työharjoittelijoita tai suorittavat työkokeilua.

Asiakkaiden tiedot kerättiin aikaisemmin yksittäisiin Excel-tiedostoihin, mutta asiakaskunnan kasvamisen takia syntyi tarve kehittää tähän parempi ja turvallisempi ratkaisu. Yritykselle on tärkeää pitää asiakkaiden tiedot ajan tasalla sekä dokumentoida kaikki tehdyt työt tarkasti. Dokumentointi on erityisen tärkeää, sillä huollettavia laitteita on paljon sekä työntekijöiden vaihtuvuus on suuri.

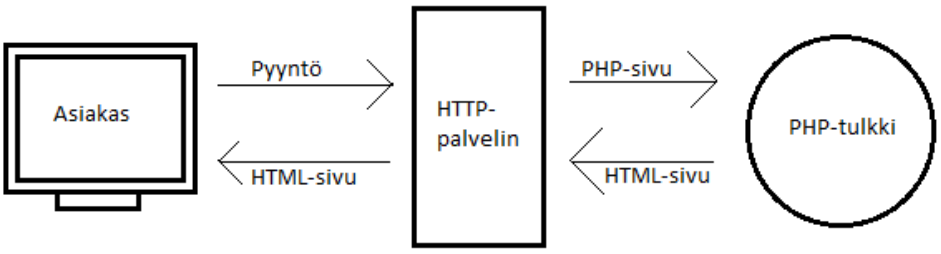
Tämän työn tavoite on kehittää yritykselle oma tietokanta sekä käyttöliittymä, jonka kautta asiakkaiden tietoihin pääsisi helposti käsiksi. Työssä käydään tarkemmin läpi tietokannan ja käyttöliittymän suunnittelua sekä toteutusta. Tässä työssä myös tutkitaan, mitä asioita tulee huomioida helppokäyttöisen ja käytännöllisen ohjelmiston toteuttamiseksi.

2 PHP: HYPERTEXT PREPROCESSOR

PHP (PHP: Hypertext Preprocessor) on ohjelmointikieli, jota käytetään paljon dynaamisten web-sivujen luomisessa. Tämän lisäksi sitä voidaan käyttää komentorivisovelluksia ja graafisia käyttöliittymiä tehdessä. (Johdatus PHP-kieleen 2015). PHP on open source –tuote, sen käyttö ei maksa, vaikka kyseessä olisikin kaupallinen tarkoitus.

2.1 PHP:n käyttäminen

PHP-koodi sisälletään HTML-koodin sekaan. Nämä tiedostot nimetään ”.php”-päätteisiksi. HTTP-palvelin tunnistaa PHP-sivun sitä kutsuttaessa (kuvio 1). Tämän jälkeen palvelin käyttää apunaan PHP-tulkkia, joka kääntää ja suorittaa PHP-koodin. HTTP-palvelimelle palautuu pelkkä HTML-koodi, jonka käyttäjä näkee selaimessa web-sivuna. PHP-koodi näkyy ainoastaan palvelimella, sitä ei näe ulospäin sivulta.



KUVIO 1. Asiakkaan ja PHP-tulkin välinen toiminta

HTML-koodin sekaan ei voi suoraan kirjoittaa PHP-komentoja. Kaikki PHP-koodi tulisi olla PHP:n omien tagien sisällä. Koodi alkaa ”<?php” –tagilla ja päättyy ”?>” –tagiin (kuvio 2).

```
1  <html>
2  <head></head>
3  <body>
4  <h1>Tulostetaan tekstiä PHP:lla</h1>
5
6  <p><?php echo "Testi" ?></p>
7
8  </body>
9  </html>
```

KUVIO 2. Esimerkki PHP:n toiminnasta HTML-koodissa

2.2 Tietotyypit

Jokainen ohjelmointikieli sisältää tietotyyppejä. Tietotyyppejä on jokaisella vakiolla ja muuttujalla. Nämä tietotyypit määräävät, mitä kyseisellä vakiolla tai muuttujalla voi tehdä. PHP-kielen tietotyyppejä on kahdeksan: boolean, integer, float, string, array, object, NULL sekä resource.

Boolean on totuusarvotietotyyppi. Boolean-tietotyypin muuttujalla voi olla vain kaksi arvoa: TRUE tai FALSE (tosi tai epätosi).

Integer on kokonaisluku, jonka maksimipituus on riippuvainen käyttöjärjestelmästä. Integer-arvon ollessa liian suuri muuttuu kyseinen muuttuja float-tyyppiin (Johdatus PHP-kieleen 2015.)

String on merkkijonotyyppi. Merkkijono voidaan sisällyttää lainausmerkkien sekä heittomerkkien sisälle. Näillä kahdella on kuitenkin eroa. Heittomerkkien sisällä olevaa merkkijonoa ei tulkita samalla tavalla kuin lainausmerkkien sisällä olevaa. Jos heittomerkkien sisälle laitetaan esimerkiksi ’$jotain\n’, jonka muuttujan arvo on aikaisemmin määritelty, tulostuu tämä samanlaisena. Lainausmerkkien sisällä ”$jotain\n” näkyy sinä muuttujan arvona, mikä on aikaisemmin määritelty sekä tähän perään tulee myös rivinvaihto.

Array on taulukko, joka voidaan määritellä ”array()”-rakenteella tai käyttämällä hakasulkuja.

Object-tietotyyppi on suomennettuna olio. Olioiden avulla voidaan selkeyttää koodia. Olio-ohjelmoinnissa luokat (class) määrittelevät olion (object) tiedon ja sen, miten oliota käsitellään.

NULL on arvo, jota ei ole määritelty. Muuttuja voi olla NULL, jos se on niin määritelty, sille ei ole annettua arvoa vielä tai sen arvot on tuhottu käyttämällä esimerkiksi ”unset()”-toimintoa. (Johdatus PHP-kieleen 2015)

Tiedostotyyppi resource sisältää viittauksen ulkopuoliseen lähteeseen. Resourcet on luotu ja tarkoitettu erikoisten funktioiden käyttöön.

2.3 Muuttujat

Muuttujan tunnistaa PHP-koodista sillä, että sitä edeltää $-merkki. Ainoa poikkeus tästä on, jos on käytetty define()-funktiota. Tällöin muuttujan arvo annetaan define()-funktion sisällä ja sitä voidaan kutsua ilman $-merkkiä. Muuttujan tietotyyppi määräytyy muuttujan arvon mukaan.

PHP-kieli sisältää myös esimääriteltyjä muuttujia, joita ei tarvitse erikseen määritellä. Näitä ovat $_SERVER, $_GET, $_POST, $_COOKIE sekä $_SESSION. (Johdatus PHP-kieleen, 2015)

$_SERVER on taulukko, joka sisältää otsikoita, reittejä sekä koodin sijainnit.

$_GET on taulukko muuttujia, jotka kuljetetaan koodiin URL-parametrien kautta ja $_POST on joukko muuttujia, jotka kuljetetaan koodiin HTTP POST –metodin kautta. GET-toimintoa voidaan käyttää, kun lähetetään julkista tietoa. Lomake, joka lähetetään käyttämällä GET-toimintoa, näyttää kaikki muuttujien nimet ja arvot URL-osoitteessa. Jos halutaan, ettei lomakkeen tietoja voida lukea sivun URL-osoitteessa selaimessa, tulee käyttää POST-toimintoa.

$_COOKIE-taulukon kautta voidaan lukea evästeitä. Evästeet ovat dataa, joka tallentuu palvelimen kautta käyttäjän tietokoneelle. Evästeet voivat olla lyhytaikaisia istuntokohtaisia. Tällöin evästeet poistuvat automaattisesti istunnon jälkeen. Pitkäaikaiset, pysyvät evästeet säilyvät joko tietyn ajan tai kunnes käyttäjä itse ne tuhoaa.

Istuntoja käytettäessä tarvitaan superglobaalia $_SESSION-taulukkomuuttujaa, jossa istunnon muuttujia käsitellään. Istunnoissa käyttäjälle luodaan satunnainen id-tunnus, joka tallentuu evästeeseen tai URL:iin. Istunnossa käytetyt muuttujat tallentuvat tiedostoon, joka vastaa id-tunnusta määritellyssä hakemistossa.

3 MYSQL

Tietokanta on kooste talletettua dataa. Jokaisella tietokannalla on yksi tai useampi sovellusohjelmointirajapinta.

PHP-kieli tukee monia tavallisesti käytettyjä tietokantoja. Näistä yksi yleisimmistä on MySQL.

MySQL on relaatiotietokantaohjelmisto, jota käytetään www-sovelluksissa. Ohjelmisto on ilmainen, joten sen käyttö on kaikille vapaata. MySQL-tietokantaa käyttävät muun muassa Facebook ja Wikipedia.

3.1 Yhteyden luominen ja tietokannan valinta

Jotta tietokannan tietoja voidaan käyttää PHP-koodissa, täytyy ensin siihen luoda yhteys. Yhteys muodostetaan mysqli_connect()-funktiolla (kuvio 3). Funktion sulkujen sisälle annetaan parametreina palvelimen URL sekä käyttäjätunnus ja salasana palvelimeen. Nämä parametrit eivät kuitenkaan ole pakollisia.

```
1  <?php
2      $connection = mysqli_connect('localhost', 'username', 'password');
3
4      mysqli_select_db("customerdb", $connection);
5  >?
```

KUVIO 3. Yhteys tietokantapalvelimelle sekä tietokannan valinta

Palvelimen yhteyden hankkimisen jälkeen valitaan tietokanta, josta halutaan tietoa. Tässä käytetään mysqli_select_db()-funktiota. Tämän funktion sulkujen sisälle laitetaan tietokannan nimi sekä aiemman mysqli_connect()-funktion toteuttava muuttuja (kuvio 3).

Sekä yhteyden muodostumisessa että tietokannan haussa voi tapahtua virheitä. Käyttäjä saa tiedon virheestä. Virhetekstit on määritelty sopiviksi, jotta käyttäjä tunnistaa virheen.

3.2 Kysely tietokantaan

Kun halutaan valita tiettyä dataa tietokantataulusta, käytetään mysql_query()-funktiota. Kuviossa 4 valitaan customers-taulusta rivit, joissa etunimi on Maija. Lopussa on parametri, jossa on määritelty mysql_connect()-funktio.

```
1  <?php
2
3  $name = mysql_query("SELECT * FROM customers WHERE firstname like 'Maija
       '", $connection);
4
5  ?>
```

KUVIO 4. Haku tietokantataulusta

3.3 Rivien lisääminen tietokantaan

Kuten kyselykin, myös rivien lisääminen tapahtuu mysql_query()-funktiossa. Lisääminen tapahtuu INSERT-lauseella. INSERT-lausetta käytetään mysql_query()-funktion sulkujen sisällä (kuvio 5).

```
1  <?php
2
3  mysql_query("INSERT INTO customers VALUES ('Maija', 'Meikäläinen')")
4      or die("Inserting values failed: ".mysql_error());
5
6  ?>
```

KUVIO 5. Rivin lisäys tietokantatauluun

Jos jostakin syystä rivin lisäämisessä tapahtuu virhe, tulee siitä ilmoitus käyttäjälle.

3.4 MySQLi ja PDO

MySQLi on uudempi, paranneltu versio MySQL-tietokannasta. Kirjain ”i” sanan lopussa viittaakin sanaan ”improved” eli suomennettuna parempi. Vanhaa versiota ei suositella enää käytettävän.

MySQLi kehitettiin PHP 5 -version myötä ja se onkin yleensä automaattisesti sisälletty php5 mysql –pakettiin. Tämä laajennus tukee olioilla ohjelmointia sekä Javan Prepared Statements –tietokantakirjastoa.

Tämä suojaa web-pohjaisia sovelluksia SQL-injektioilta eli siltä, ettei tietokannan kautta voitaisi tunkeutua järjestelmiin. (Pontikis 2013.)

PDO, tietokannan ohjelmointimalli, on myös kehitetty tukemaan PHP 5:tä ja sitä uudempia versioita. Kuten MySQLi, myös PDO tukee olioilla ohjelmointia sekä Prepared Statements -kirjastoa. PDO tarjoaa tuen sekä MySQLi-tietokannalle että monelle muullekin.

3.5 Tietoturva

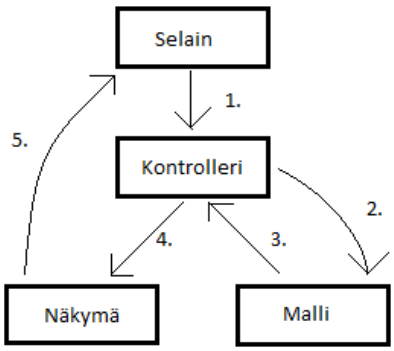
Tietoturva on yksi keskeisimmistä asioista www-sovelluksien kehittämisessä. Web-sivustoilla on mahdollista ohjata internet-hakuja eri osoitteisiin, lähettää sähköpostia väärillä tiedoilla, kopioida käyttäjätunnuksia salasanoineen sekä luottokorttien numeroita. (Suomen Internetopas 2017.) Huonosti suunniteltu ja toteutettu tietoturva lisää riskiä tietojen väärinkäytöksiin sekä virusten tunkeutumista tietokantaan.

Tietokannan suunnittelussa on tärkeää tietoturvan kannalta pohtia, mitä tietoa säilytetään, kuka tietokantaa käyttää ja mitä varten se on kehitetty. Tämän pohjalta on hyvä aloittaa suunnittelu, jotta myöhemmin ei tule vastaan ongelmia.

Tietokantaan voi antaa useille käyttäjille käyttöoikeudet. Käyttäjällä voi olla oikeus katsoa, muuttaa ja poistaa dataa tietokannasta. Tietoturvan ylläpitämiseksi on suositeltavaa antaa käyttäjille mahdollisimman vähäiset oikeudet. Näin voidaan välttää tahalliset sekä tahattomat vahingot. Käyttöoikeuksia voi lisätä tarpeen tullen.

4 LARAVEL

Laravel PHP Framework on helmikuussa 2012 julkaistu MVC PHP-sovelluskehys. MVC koostuu sanoista model, view sekä controller. Kyseessä on ohjelmistoarkkitehtuuri, jonka avulla koodi voidaan jakaa kolmeen osaan: malleihin, näkymiin sekä kontrollereihin. Selain lähettää aineistopyynnön kontrollerille, joka pyytää ja päivittävää tietoja mallilta. Tämän jälkeen kontrolleri lähettää tiedot näkymälle, joka palauttaa selaimelle HTML-sivun (kuvio 6).



KUVIO 6. MVC-arkkitehtuurin toiminta

Laravel helpottaa web-pohjaisten projektien yleisiä tehtäviä, kuten todentamista, reitittämistä sekä sessioiden ja välimuistin käsittelyä (Laravel 2015). Työssä käytettiin Laravelin versiota 4.

4.1 Laravelin käyttöönotto

Laravel hyödyntää Composer-ohjelmaa hallitakseen riippuvaisuuksiaan. Composer on työkalu, jolla hallitaan riippuvaisuuksia PHP-projektissa. Sen avulla voidaan esitellä tarvittavat kirjastot ja Composer asentaa ne projektiin käyttäjän puolesta. Composerin voi asentaa Windows-käyttöjärjestelmässä sen omalla asennusohjelmalla.

Kun Composer on asennettu, ladataan Laravel-sovelluskehyksen uusin versio ja puretaan sen sisältö palvelimen hakemistoon. Seuraavaksi ajetaan Laravel-sovelluksen juuressa Composerin asennus.

Projektiin tarvitsee myös luoda composer.json –tiedosto. Tiedosto kuvailee projektin riippuvaisuuksia sisältäen muutakin metadataa.

4.2 Projektin rakenne

Composerin asennus Laravel-projektissa luo juureen seuraavanlaisen tietorakenteen.

- app/
- bootstrap/
- vendor/
- public/
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- phpunit.xml
- server.php

App-kansio sisältää projektin peruskoodin. Tämä kansio on Laravel-projektin ydin, ja tässä on sen sisältämä tiedostorakenne:

- commands/
- config/
- controllers/
- database/
- lang/
- models/
- start/
- storage/
- tests/
- views/
- filters.php

- routes.php

Kansion alla olevat alikansiot ja tiedostot sisältävät kaiken web-sovelluksen taustalla olevista asetuksista itse web-sovelluksen näkymään asti.

Bootstrap-kansio sisältää tiedostot ”autoload.php”, ”paths.php” sekä ”start.php”. Nämä tiedostot liittyvät sovelluskehyksen käynnistys-toimintoihin ja muutosten tekoa kyseisiin tiedostoihin suositellaan vain kokeneille Laravel-ohjelmoijille.

Kaikki web-puolen tiedostot tulisivat sijaita public-kansiossa. Näihin luetaan CSS, Javascript sekä kuvat. Public-kansio alla ovat tiedostot ”.htaccess”, ”favicon.ico”, ”index.php” sekä ”robots.txt”. ”.htaccess”-tiedosto sisältää yleisimpien asetusten toimintaohjeet. Favicon on 16x16 pikselin kokoinen kuvake, joka näkyy web-selaimen välilehdessä sivun otsikon vieressä. ”index.php” käynnistää sovelluskehyksen toiminnan. ”Robots”-niminen tekstitiedosto on olemassa hakuroboteille, jotka käyvät läpi web-sovelluksen lähteitä.

Git on versionhallinta-työkalu, jonka asetukset ovat ”.gitattributes”-tiedostossa. Näin projektista voidaan tallentaa jokainen versio, mikä voi helpottaa ohjelmoijaa ongelmatilanteissa, esimerkiksi virheiden tunnistamisessa ja korjaamisessa. ”.gitignore”-tiedosto sisältää tiedon siitä, mitä kansioita ei haluta versioida.

Artisan-ohjelma on exe-tiedosto. Se tarjoaa useita hyödyllisiä komentoja, jotka voivat auttaa sovelluksen toteuttamisessa.

Kun vendor-kansio sisältää composer-paketit, niin ”composer.json”- sekä ”composer.lock” –tiedostot sisältävät tiedot kyseisistä paketeista.

”phpunit.xml”-tiedosto tarjoaa PHP Unit Testing –sovelluskehyksen yleiset asetukset.

”server.php”-tiedostoa käytetään luomaan helppoja tapoja ajaa web-sovellusta.

4.3 Reitittäminen

Laravelissa reitit määritellään app-kansion alla olevassa ”routes.php”-tiedostossa. Tämä helpottaa ohjelmoijan työtä, sillä kaikki reititysten muutokset voidaan tehdä yhteen paikkaan. Myös reititysvikojen paikantaminen helpottuu. Reititys-tiedostossa määritellään muun muassa painikkeista tapahtuvat ohjaukset toisille sivuille sekä linkkien toiminta.

```
1  <?php
2
3  Route::get('my/testpage', function() {
4      return 'This is a test!';
5  });
6
7  ?>
```

KUVIO 7. Reitityksen toiminnan osoittava esimerkki

Kuvion 7 esimerkkiä voidaan testata avaamalla web-selain ja kirjoittamalla osoitekenttään ”http://domain.com/my/testpage”, tosin domainin osoite korvataan oman projektin osoitteella. Tässä voidaan käyttää myös ”localhostia”. Jos kaikki on kunnossa, tulisi selaimessa näkyä teksti ”This is a test!”.

Reitit esitellään käyttämällä Route-nimistä luokkaa. Get on route-luokassa metodi, jolla haetaan URL-osoitetta. Post-metodia käytetään, kun halutaan lähettää dataa näyttämättä sitä URL-osoitteessa.

Metodin jälkeen sulkumerkkien sisällä mainitaan URI, johon viitataan. Yllä olevassa esimerkissä tämä sivu on ”my/testpage”. Tämän jälkeen on pilkulla erotettu anonyymi funktio.

Reitityksen apuna voidaan käyttää myös muuttujia (kuvio 8). Tässä tapauksessa $test-muuttuja sisältää funktion ja siirtää sen Route::get()-metodille.

```php
<?php

$test = function() {
    return 'This is a test!';
};

Route::get('my/testpage', $test);

?>
```

KUVIO 8. Reititys käyttämällä apuna muuttujaa

4.4 Kontrollerit

Kontrolleri yhdistää mallin ja näkymän sekä toimii niiden välillä datan välittäjänä. Kontrolleri on luokka, joka yleensä sisältää julkisia metodeja eli toimintoja. Kontrollerien ansiosta tiettyjä reitittämisiä voidaan järjestellä omissa kontrollereissaan. Jokaisen kontrollerin täytyy periä Basecontroller- tai Controller-luokkaan. Kyseiseen luokkaan voidaan asettaa kaikille kontrollereille yhteiset ominaisuudet.

Kontrollerille voi antaa minkä tahansa nimen ja tiedosto löytyy app-kansion alla olevasta controllers-alikansiosta.

```php
<?php

class TestController extends BaseController
{
    public function showPage()
    {
        return View::make('test');
    }
}

?>
```

KUVIO 10. Esimerkki kontrollerin sisällöstä

Kontrollerin luokan sisällä olevien metodien täytyy olla julkisia, että Laravel osaisi hoitaa niiden reitityksiä (kuvio 10). Tässä tapauksessa toiminta ei ole vielä taattu, routes.php-tiedostoon tarvitsee myös tehdä lisäys (kuvio 11).

```php
<?php

Route::get('test', 'TestController@showPage');

?>

```

KUVIO 11. Reititys-tiedostoon määritellään kontrollerin tapahtuma

Kontrollerin luokan nimi on TestController ja kontrollerissa toiminto, joka haluttiin reitittää on showPage. Nämä yhdistetään @-merkillä. Näin reititys-tiedostossa olevia metodeja voidaan yhdistää kontrollereihin.

4.5 Blade ja sen käyttö

Blade-termille ei ole suoraa suomennosta. Nimitys tulee siitä, kun .NET -sovellusalustalla on työkalu nimeltä ”Razor”, josta Bladekin on johdettu ja näin siitä saadaan yhdistelmä ”Razorblade” ja suomennettuna se tarkoittaa partaterää. (Code Bright 2015, 105.)

Tiedosto, jossa on käytetty pelkästään HTML:ää, on usein siisti ja helppolukuinen. Usein mukaan joudutaan lisäämään kuitenkin myös PHP:ta. Tällöin PHP-tagit täytyy sisällyttää tähän puhtaiden HTML-rivien joukkoon. Bladen käyttö on tässä vaiheessa suotavaa. Tiedosto ei tällöin ole pelkkä ”testi.php” vaan se vaatii blade-laajennuksen. Silloin tiedoston nimi olisi ”testi.blade.php”.

```php
<p><?php echo $customerJobs; ?></p>

<p>{{ $customerJobs }} </p>
```

KUVIO 12. Sama toteutus kahdella tavalla

Bladessa käytettään aaltosulkumerkkejä ja kaikki sulkumerkkien sisällä oleva data muuntuu kaiutukseksi (kuvio 12). Tämä on siistimpi ja helpompi toteutustapa.

4.6 Eloquent ORM ja tietokanta

ORM (Object-relational mapping) eli olio-relaatio-mallinnus yhdistää tietokannan tietotyypit ohjelman tietotyyppeihin. Valmiiksi rakennettua

tietokantaa ja tauluja on helpompi käsitellä ORM:n avulla. Jos kuvataan työn kokonaisuuksia objekteina, on myös järkevämpää tallentaa ne objekteina sekä hakea niitä objekteina. (Laravel 2015.)

Eloquent on Laravelin ORM-komponentti. Tämä komponentti helpottaa ohjelmoijaa vuorovaikuttamaan sovelluksen tietokantakerroksen kanssa.

4.6.1 Uusien mallien luonti

Käyttäjän luodessa uutta mallia projektiin, täytyy luoda php-tiedosto models-kansion alle. Tämän voi nimetä miten haluaa. Tiedoston sisälle lisätään rivi ”use Illuminate\Database\Eloquent\Model;”, jonka jälkeen luodaan luokka. Näin on luotu Eloquent-malli. Jotta tätä voitaisiin hyödyntää, täytyy luoda instanssi kyseisestä mallista ja asettaa sille attribuutit, jotka ohjaavat taulun sarakkeet sellaisiksi, joita tarvitaan. Tämän jälkeen kutsutaan save()-metodia objektille ja tieto tallentuu palvelimen määriteltyyn tietokantatauluun.

4.6.2 Olemassa olevan mallin lukeminen

Jos halutaan palauttaa yksi malli-instanssi tietokannasta, voidaan siinä käyttää apuna id-saraketta. Tähän tarvitaan avuksi find()-metodia.

```
1  <?php
2
3  Route::get('/', function()
4  {
5      $book = Book::find(1);
6      return $book->name;
7  });
8
9  ?>
```

KUVIO 13. Haetaan kirja, jonka id on 1

Staattisen find()-metodin avulla saadaan tieto, mikä kirja on tietokannassa rivillä, jonka id on 1.

4.6.3 Olemassa olevan mallin poistaminen

Mallin poistamisessa voidaan myös käyttää find()-metodia, jotta voidaan poistaa se instanssi, joka halutaan poistaa (kuvio 14).

```php
<?php

Route::get('/', function()
{
    $book = Book::find(1);
    $book->delete();
});

?>
```

KUVIO 14. Yhden instanssin poisto

Jos kuitenkin halutaan poistaa enemmän kuin yksi instanssi mallin tietokannasta, käytetään destroy() metodia (kuvio 15).

```php
<?php

Route::get('/', function()
{
    Book::destroy(1, 2, 3);
});

?>
```

KUVIO 15. Instanssien poisto käyttäen destroy()-metodia

4.7 Views-näkymät

Näkymät ovat työn visuaalinen osa. Näkymä on yksinkertainen tekstimalli, joka todennäköisesti sisältää HTML:ää. Näkymät ovat kuitenkin PHP-tiedostoja, joten PHP-ohjelmointikieltä voidaan käyttää näkymän sisällä. (Laravel 2015.)

```php
<html>
    <body>
        <p>Nice to meet you, <?php echo $name; ?></p>
    </body>
</html>
```

KUVIO 16. Yksinkertaisimmillaan näkymä voi näyttää tältä

5 KÄYTETTÄVYYS

Tärkeintä käyttöliittymän suunnittelussa oli saada aikaiseksi onnistunut kokonaisuus. Tässä tapauksessa käytettävyys tarkoittaa sitä, kuinka hyvin tämän työn toimintoja voidaan käyttää tarkoitukseensa (Johdatus käytettävyyteen 2015a).

Käyttöliittymästä piti tehdä tarpeeksi yksinkertainen, jotta sen käyttöä ei juurikaan tarvitse opiskella vaan yrityksen työntekijät voisivat käyttää sitä vaivattomasti. Mitä helppokäyttöisempi käyttöliittymä on, sitä parempi se on yritykselle. Virhetoimintoja tulee vähemmän, käyttö on nopeampaa eikä käyttöliittymän käyttämiseen tarvita erillistä koulutusta tai tukihenkilöstöä.

5.1 Nielsenin heuristiikat

Jo käyttöliittymän suunnittelussa oli hyvä ottaa huomioon käyttöliittymän heuristinen arviointi. Tämän perusteella voidaan päätellä, kuinka hyvä käyttöliittymä on. Monesti käyttöliittymän arvioinnin perustana käytetään Jakob Nielsenin kehittämää listaa web-käyttöliittymien käytettävyydestä. Nielsenillä on tohtorin tutkinto ihmisen ja tietokoneen välisestä vuorovaikutuksesta. (Johdatus käytettävyyteen 2015b.) Seuraavaksi käsitellään, miten nämä heuristiikat vaikuttivat tähän työhön.

5.1.1 Yksinkertainen ja luonnollinen dialogi

Yksinkertaisuus oli tämän työn yksi keskeisimmistä asioista. Käyttöliittymästä piti karsia pois kaikki, mikä voisi hankaloittaa käyttäjän työskentelyä. Tämä tarkoitti sitä, että esillä tulee olla vain välttämätön tieto, vältetään ylimääräisiä painikkeita ja linkkejä, käytetään muutamaa perusväriä visuaalisessa ilmeessä sekä järjestyksen näytöllä tulee olla looginen. Tärkeintä oli, että käyttäjän ei tarvitse nähdä vaivaa käyttöliittymää käytettäessä. Asiakkaan ja töiden lisäys täytyi olla yksinkertaista ja nopeaa.

5.1.2 Käyttäjien oma kieli

Käyttöliittymän kielen valinta ei ollut vaikea, sillä kaikki yrityksen työntekijät puhuvat ja ymmärtävät suomea. Täytyi tosin muistaa, että harjoittelijoiden koulutustaustat olivat erilaisia, joten ammattitermistöä haluttiin välttää mahdollisten ongelmatilanteiden minimoimiseksi. Tällä hetkellä kaikki linkit ovat käyttöliittymässä painikkeita tai tekstiä, mutta mahdollisesti myöhemmin ne voisi korvata myös erilaisilla symboleilla. Näiden symbolien tulisi olla myös yksinkertaisia sekä sellaisia, joita kaikki varmasti ymmärtävät.

5.1.3 Käyttäjän muistikuorman minimointi

Käyttöliittymän käyttämisestä on tehty mahdollisimman helppoa juuri minimoimalla käyttäjän muistikuormaa, onhan tietokoneelle helppo tallettaa tietoja. Käyttäjä voi lisätä asiakkaita ja laitteita sekä päivittää tietoja helposti ilman, että tarvitsee muistaa, minkäänlaisia sääntöjä näille tehtäville. Sääntöjä on toki tehty, mutta käyttäjän ei tarvitse niitä muistaa. Esimerkiksi asiakasta lisättäessä on määritelty annettujen tietojen maksimipituudet sekä se, onko kyseessä numereellinen arvo vai ei. Vaikka käyttäjä laittaisikin väärin jonkin tiedon, kuten sähköpostiosoitteessa ei olekaan @-merkkiä, ilmoittaa käyttöliittymä virheestä ja käyttäjän on siten helppo huomata ja korjata virhe.

Yrityksellä on myös myynnissä tiettyjä palveluita, jotka on lueteltu tietokannassa omassa taulussa. Töihin voi tällöin lisätä palvelut, jotka työlle tehtiin ja samalla se antaa hinnan, mikä asiakkaalle tulee maksettavaksi. Tämäkin helpottaa käyttäjän työtä, sillä käyttäjän ei tarvitse itse kirjoittaa, mitä tietokoneelle on tehty ja samalla vältetään myös mahdolliset hinnoitteluvirheet.

5.1.4 Yhdenmukaisuus

Käyttöliittymässä ei ole montaa erillistä sivua, mutta kaikki ovat silti jollakin tavalla yhdenmukaisia. Navigointilaatikko on kaikissa sivuissa samassa

kohdassa, jotta sivujen välillä liikkuminen olisi mahdollisimman sujuvaa. Myös sivuilla on paluu-painikkeet edelliselle sivulle ja nämäkin ovat sijoitettu jokaisella sivulla samalle paikalle.

5.1.5 Riittävä palaute

Käyttäjälle täytyy myös nähdä ja tietää, onnistuiko vai epäonnistuiko mahdollinen tehtävä. Tietojen tallennuksessa tämä on erityisen tärkeää. Vaikka virheitä ei tulisikaan ja tiedot tallentuisivat tietokantaan, niin käyttäjälle on hyvä ilmoittaa tästäkin. Jos ilmoitusta ei tule, käyttäjä voi jäädä epävarmaksi siitä, menikö kaikki hyvin vai ei. Virhetilanteissa täytyy näkyä, missä sekä minkälainen virhe on, ettei käyttäjän tarvitse sitä etsiä ja arvuutella.

Myös tietojen poistossa ja päivittämisessä on hyvä olla ilmoituksia. Tietojen päivittämisessä kelpaa ilmoitus siitä, onnistuiko päivitys vai ei. Poistossa taas on hyvä varmistaa, haluaako käyttäjä varmasti, että tiedot poistuvat pysyvästi.

Käyttäjällä tulee olla tunne, että hän hallitsee järjestelmää. Palautteen puute voi johtaa useisiin päällekkäisiin talletuksiin ja tiedot saattavat sekoittua toistensa joukkoon. (Johdatus käytettävyyteen 2015.) Jos käyttäjä tallettaa monta saman nimistä asiakasta samoilla tiedoilla, voi toisten käyttäjien olla vaikea tietää, mikä niistä on oikea ja päivitetty versio.

5.1.6 Selkeä poistumistapa eri tilanteista ja tiloista

Jokaisella sivulla on paluu-painike. Lomake-sivuille on lisätty myös peruuta-painike talletuksen viereen. Navigointi-laatikosta pääsee myös kätevästi siirtymään etusivulle asiakas-näkymään.

5.1.7 Selkeät virheilmoitukset

Käyttöliittymä ilmoittaa, missä virhe on ja minkälainen virhe on kyseessä. Virhe-viesti näkyy punaisena ja tietoja ei pysty tallentamaan tai päivittämään, jos virheitä tulee.

5.1.8 Virheiden estäminen

Virheiden estämiseksi käyttöliittymä on suunniteltu mahdollisimman käyttäjäystävälliseksi. Tämä tarkoittaa sitä, että käyttäjän ei itse tarvitse tehdä paljoakaan työtä asiakkaiden ja töiden lisäämiselle. Käyttäjän tarvitsee vain antaa tiedot ja muistaa tallennus. Palvelut on annettu myös valmiiksi ja niitä käyttäjä voi lisätä töihin vain klikkaamalla myyty palvelu.

6 TIETOKANNAN SUUNNITTELU

Ennen kuin voidaan alkaa työskentelemään itse ohjelmoinnin ja käyttöliittymän parissa, täytyy suunnitella tietokanta. Hyvin rakennettu tietokanta antaa mahdollisuuden tuottaa monipuolinen ja selkeä käyttöliittymä sen ympärille.

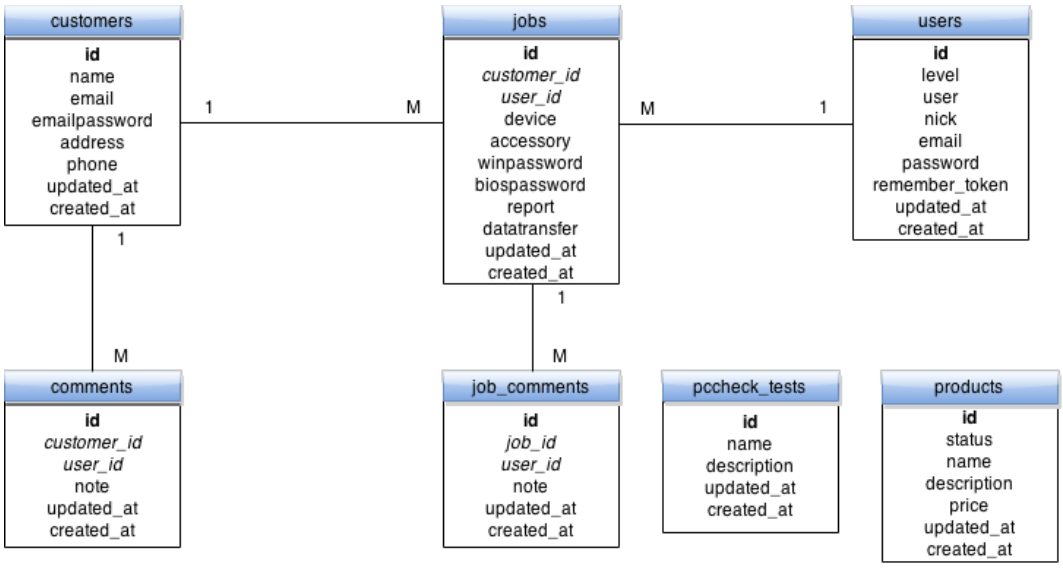
6.1 Asiakasvaatimukset

Aluksi täytyi ottaa selvää, mitä asiakas haluaa tietokannalta sekä käyttöliittymältä. On hyvä kysyä, mitkä ominaisuudet ovat välttämättömiä ja niiden perusteella mietitään sopivia tietokantaratkaisuja. Tässä työssä oli kuitenkin mahdollisuus toteuttaa omia ideoita melkein minkä tahansa suhteen. Tärkeintä oli, että käyttöliittymän kautta voitaisiin tallentaa palvelimelle asiakkaan sekä töiden tietoja. Tietojen mahdollinen tulostus oli myös tärkeä ominaisuus. Erityisesti haluttiin helpottaa työntekijöiden työtaakkaa, joten se täytyi myös ottaa huomioon.

6.2 Relaatiokaavio

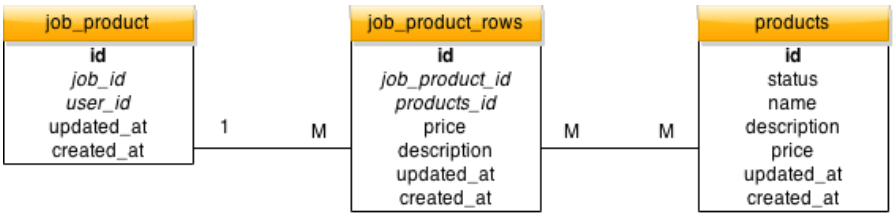
Tässä työssä tarvittiin tietokanta, mikä hallitsee asiakkaiden, laitteiden ja tuotteiden sekä palveluiden suhteita toisiinsa. Taulujen nimet näkyvät seuraavalla sivulla olevassa kuviossa laatikoiden päällä sinisellä pohjalla. Tämän jälkeen tummennettuna ovat Primary Keyt. Kursivoituna näkyvät Foreign Keyt. Näiden avulla muodostuu suhteita, esimerkiksi yhdellä asiakkaalla voi olla monta työtä, mutta työ voi olla vain yhdellä asiakkaalla. Vähintään kahden kohteen välinen riippuvuus muodostaa suhteen. (Käsitteellinen mallintaminen 2004.)

Kohteiden välisen suhteen voi huomata ER-kaaviossa omalla merkinnällään. "1-M" tarkoittaa yhden suhdetta moneen, "1-1" tarkoittaa yhden suhde yhteen ja "M-M" taas tarkoittaa monen suhdetta moneen.

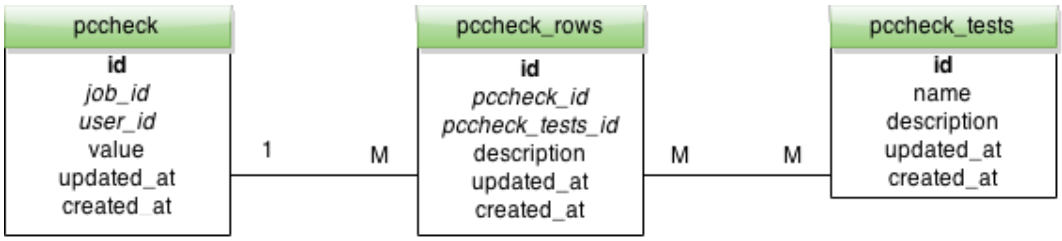


KUVIO 17. Relaatiokaavio tietokannasta

Tietokannan kokoamisessa täytyi miettiä, mitä tauluja tarvitaan ja miten ne yhdistetään (kuvio 17). Taulujen sisällön tuottaminen ei ollut vaikeaa, sillä tiedettiin, mitä tietoja kussakin taulussa täytyy olla. Hankaluuksia tuli eteen vasta, kun tarvitsi yhdistää monia tauluja toisiinsa. Esimerkiksi yhteen työhön voidaan liittää monia palveluita. Liitetyt palvelut haluttiin saada työn id:n alle ja samalla haluttiin tietää käyttäjän id:n perusteella, kuka palveluita on lisännyt. Tämän vuoksi täytyi rakentaa products-taulu sisältäen palveluiden nimet, kuvaukset sekä hinnat, job_product-taulu, mihin tallentuivat työn ja käyttäjän id:t ja job_product_rows-taulu, missä yhdistettiin valitun palvelun eli products-taulun id sekä job_product-taulun id (kuvio 18).



KUVIO 18. Palveluiden tauluista tehty relaatiokaavio



KUVIO 19. PCCheck:n tauluista tehty relaatiokaavio

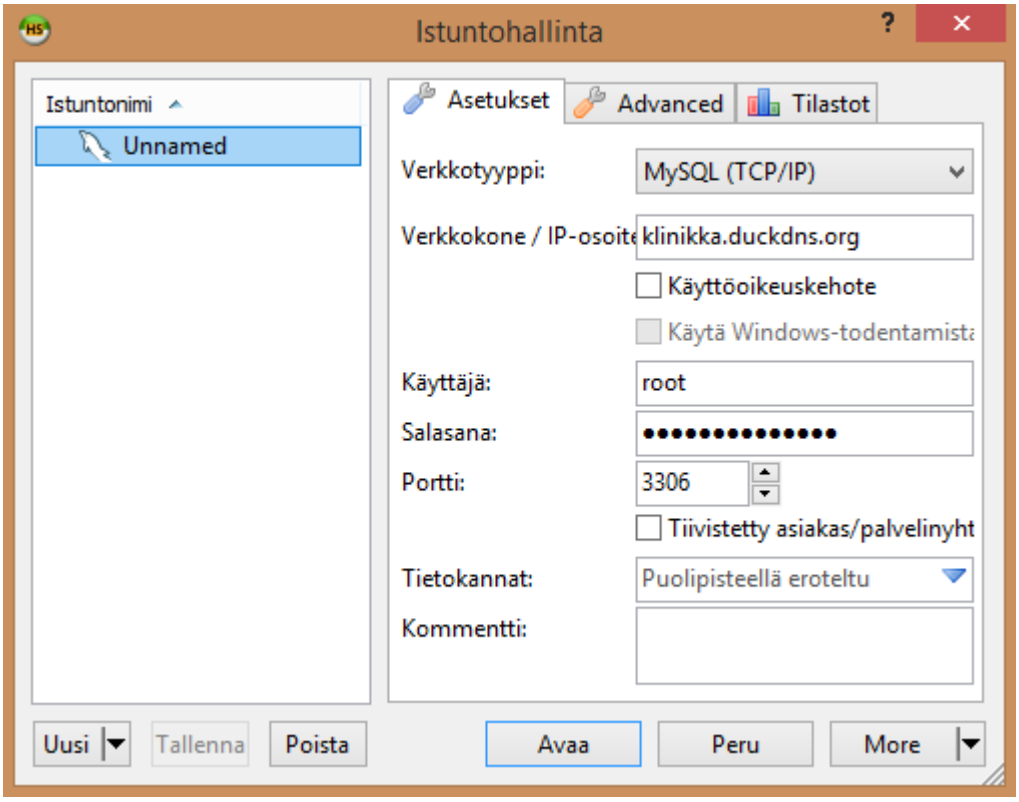
Palveluiden tietokantatauluissa (kuvio 19) käytetty yhdistämismenetelmä toimii myös PCCheck:n kanssa. PCCheck-ohjelman tarkastamat osiot näkyvät pccheck-tests-taulussa. Työn ja käyttäjän id:t sekä PCCheck:n arvo ovat pccheck-taulussa. PCCheck:n arvolla tarkoitetaan sitä, oliko mekaanisessa tarkastuksessa ongelmia. Jos ongelmia ilmenee, arvo on 1. Kaiken ollessa kunnossa arvo on 0. Näiden kahden taulun annetut tiedot kootaan pccheck_rows -tauluun sijoittamalla siihen taulujen id:t.

7 TIETOKANNAN TOTEUTUS

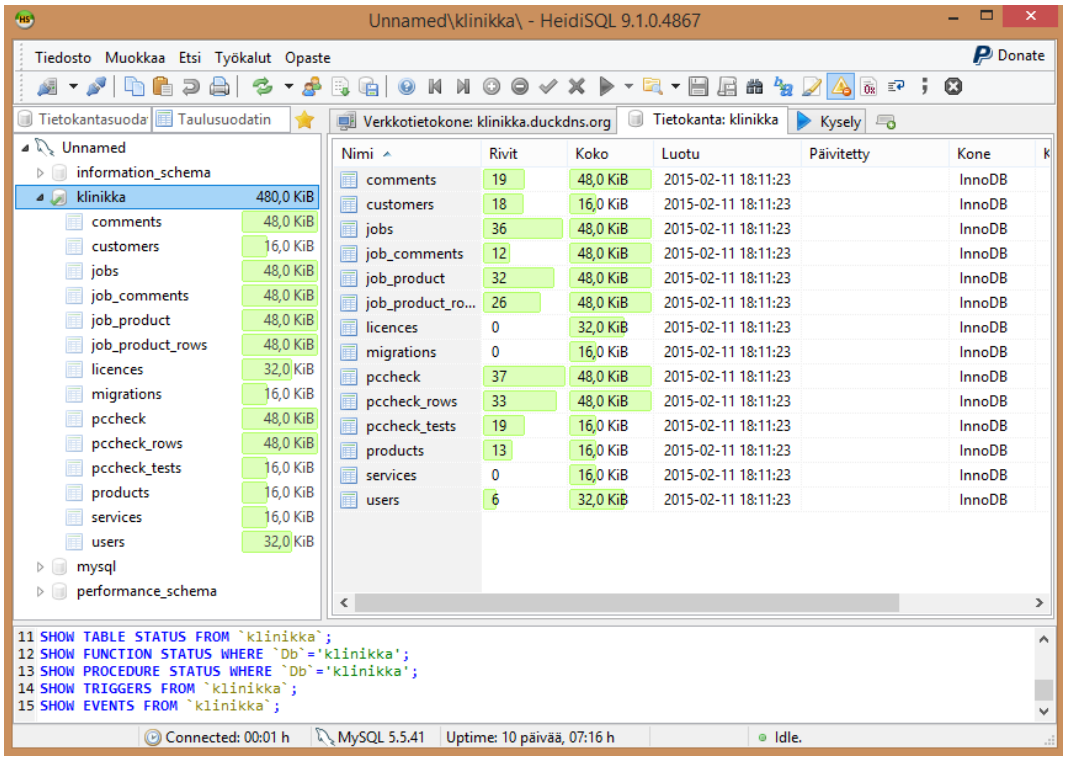
Hyvässä tietokannassa tietoa on helppo lisätä, hakea ja muuttaa. (Ohjelmointiputka 2009). Työn tietokantaohjelmistoksi valittiin MySQL. Vaikka kyseinen ohjelmisto onkin ilmainen, on se silti luotettava valinta. MySQL:ää käyttävätkin muun muassa Facebook, Wikipedia ja Google. (Wikipedia 2015.)

7.1 Tietokannan luominen

Tietokannan luomisessa käytettiin HeidiSQL-nimistä ilmaisohjelmaa. Tässä ohjelmassa tietokannan ja taulujen luominen on resurssitehokasta. Taulun rivejä pystyy päivittämään kätevästi ja nopeasti.
Ohjelmaa käynnistäessä täytyy ensin kirjautua palvelimelle (kuvio 20). Tämän jälkeen avautuu ohjelman päänäkymä (kuvio 21). Vasemmalla laidalla näkyvät palvelimella olevat tietokannat. Klikkaamalla tietokannan nimeä, näkee mitä tauluja kyseinen tietokanta sisältää.



KUVIO 20. Kirjautuminen tietokantaan



KUVIO 21. HeidiSQL:n päänäkymä

7.2 Taulujen luominen tietokantaan

Kuten aikaisemmin mainittiin, tietokannan nimeä klikkaamalla näkee tietokannassa olevat taulut. Taulua klikkaamalla taas tulee esille taulun tiedot. Rivejä pystyy lisäämään, päivittämään sekä poistamaan helposti. MySQL-komentoja ei siis tarvita.

8 KÄYTTÖLIITTYMÄN SUUNNITTELU

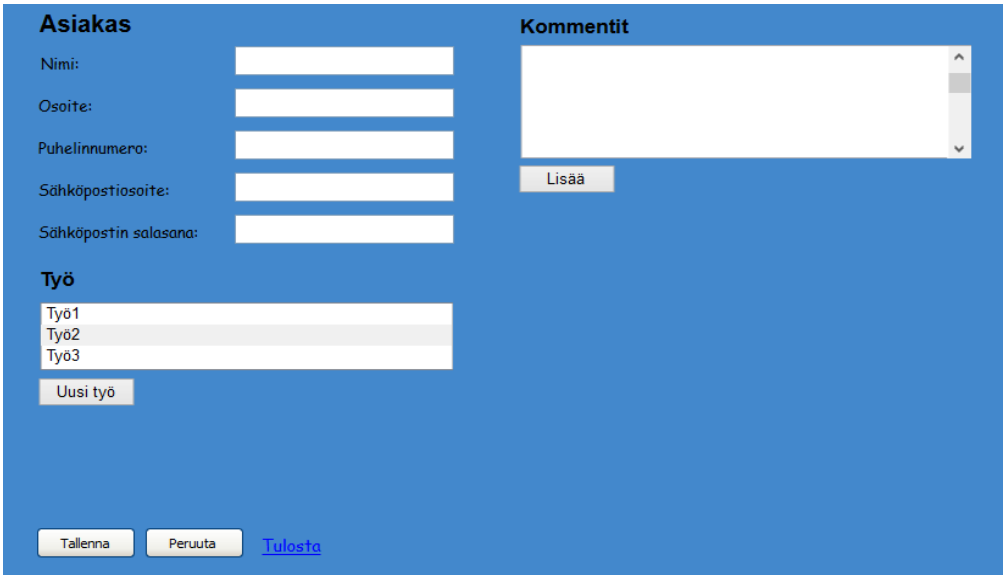
Käyttöliittymä on olennainen osa tätä opinnäytetyötä. Tämän suunnittelu ja toteuttaminen vei eniten aikaa, sillä lopputuloksen täytyi olla tarpeeksi yksinkertainen, mutta silti monipuolinen sisällöltään.

8.1 Rautalankamalli

Erilaisten verkkosivustojen ja -palvelujen visuaalisen ilmeen suunnittelussa on hyvä käyttää rautalankamallia. Rautalankamalli on yksinkertainen esitys sivuston rakenteesta ja sen avulla voidaan testata käyttöliittymän toimivuutta. Tämä malli ei kuitenkaan näytä lopullista visuaalista ilmettä, mutta mallin avulla on käyttöliittymän toteuttajan helpompi päästä alkuun. (Rautalankamalli 2014.) Tässä työssä käytettiin Pencil Project -nimistä ohjelmaa rautalankamallin suunnitteluun.

8.2 Sivujen suunnittelu

Seuraavaksi käydään läpi kolmen sivun rautalankamallit. Nämä eivät vastaa työn lopullista ilmettä, mutta tämän avulla hahmotettiin lomakkeiden ulkoasu ja painikkeiden mahdolliset sijainnit.

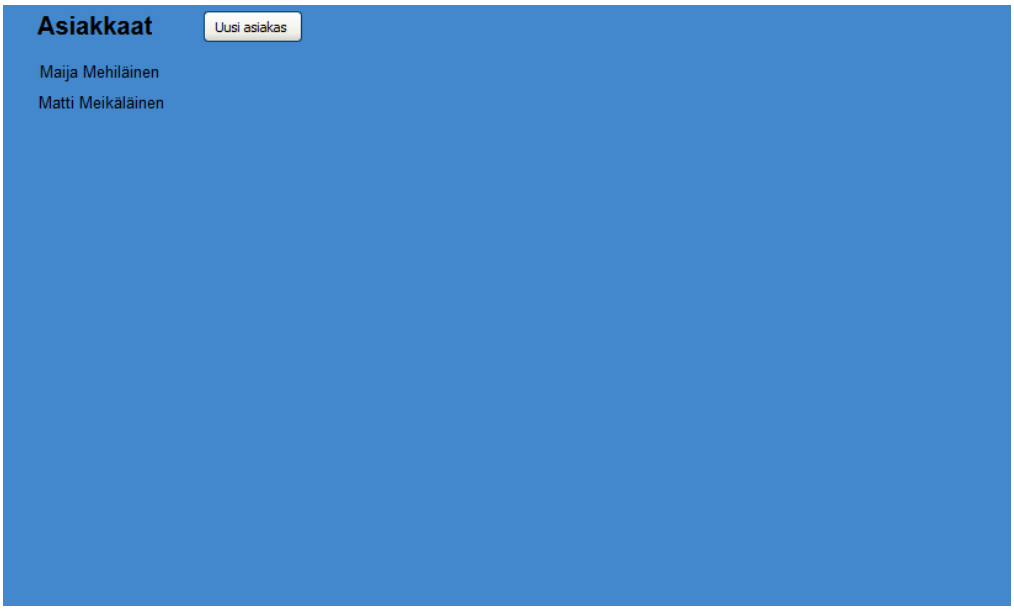


KUVIO 21. Asiakkaan lisäys -lomakkeen näkymä

Kuviossa 21 näkyy ensimmäinen luonnos asiakkaan lisäys -lomakkeelle. Asiakkaalta kysytään perustiedot, joista pakollisia ovat nimi, osoite sekä puhelinnumero. Tämän jälkeen voidaan lisätä asiakkaalle yksi tai useampi työ. Asiakkaan tietojen tallennuksen jälkeen voidaan tulostaa dokumentti, mistä näkyy asiakkaan sekä töiden annetut tiedot. Näitä dokumentteja tulostetaan kaksi kappaletta, joista toinen annetaan asiakkaalle ja toinen jää yritykselle.

KUVIO 22. Uuden työn lisäys -lomakkeen näkymä

Työn lisäämisessä kuviossa 22 on tärkeää, että kaikki tarvittavat tiedot on täytetty oikein. Pakolliset tiedot ovat laitteen malli, vastaanottaja sekä selvitys. Tiedonsiirrossa käytettiin valintaruutua, jonka taustalla on boolean-tietotyppi. Jos tiedonsiirtoa tarvittiin työssä, aktivoitiin valintaruutu painamalla sitä, tällöin vastaus oli TRUE eli tarvitaan. Jatkamalla ilman valintaruudun valitsemista vastaus olisi FALSE, jolloin tiedonsiirtoa ei tarvita.



KUVIO 23. Etusivu sekä asiakas näkymä

Asiakas-näkymä (kuvio 23) toimii myös etusivuna. Näin työntekijät pääsevät nopeasti ja helposti käsiksi asiakkaiden tietoihin kirjauduttuaan käyttäjätunnuksellaan palvelun sisälle. Asiakkaat näkyvät siinä järjestyksessä, kun heidät on lisätty.

9 KÄYTTÖLIITTYMÄN TOTEUTUS

Käyttöliittymästä tehtiin asiakkaan toiveiden mukainen. Toteutusvaiheessa jouduttiin välillä tietenkin testaamaan sivuston toimivuutta, minkä takia esimerkiksi ulkoasu toteutuneessa versiossa eroaa rautalankaversiosta.

9.1 Toteutuksessa käytetyt resurssit

Käyttöliittymän toteutusta varten tarvittin työtietokone. Tämän työkoneen täytyy olla tarpeeksi tehokas ja siihen tarvitsi asentaa tarvittavat ohjelmat. Mahdollisten ylimääräisten ongelmien estämiseksi, tämä työkone oli ainoastaan tämän ohjelmiston suunnittelua ja toteutusta varten.

Laravelin opiskelumateriaalina oli käytössä kaksi e-kirjaa kyseisestä aiheesta. Ensimmäinen kirjoista oli Dayle Reesin Code Bright vuodelta 2014 ja toinen Jeffrey Wayn Laravel Testing Decoded vuodelta 2013.

Aluksi työkoneelle asennettiin toteutukseen käytettävät ohjelmat. Ensimmäiseksi asennettiin työkoneelle Laravel, minkä käyttöönotto selvitettiin tässä työssä jo aikaisemmin. Tämän jälkeen asennettiin tekstieditori, jota käytettiin itse ohjelmointiin. Tässä työssä se oli Sublime Text Editor 2. Samaan aikaan asennettiin myös tietokannan toteutuksessa tarvittavat ohjelmat.

Yksi tärkeä resurssi ohjelmiston toteutuksessa on myös työympäristö. Vaikka yrityksen tilat olivatkin melko pienet ja ahtaat, oli tarvittava työtila kuitenkin saatavilla.

9.2 Mallit projektissa

Mallin php-tiedostoon luodaan ensin luokka, jonka nimen voi itse päättää. Luokan täytyy perii Eloquent-instanssin.

Projektissa käytettiin usein find()-metodia (kuvio 24) yhden malli-instanssin palauttamiseksi tietokannasta käyttäen apuna id-saraketta. Jokaisella

asiakkaalla oli oma id, samaa id:tä ei voinut käyttää kahdesti. Käyttäjän ei tarvitse itse keksiä id-numeroa vaan se tulee automaattisesti asiakkaalle.

```php
<?php

Route::get('/', function()
{
    $customer = Customer::find(1);
    return $customer->name;
});

?>
```

KUVIO 24. find()-metodin käyttö projektissa

9.2.1 Eloquent-komponentin käyttö

Projektissa käytettiin erilaisia malleja, joissa esitettiin tietokanta ja sen relaatiot muihin tauluihin. Customer-mallissa esiintyy Customer-luokka, minkä sisällä määriteltiin customers-taulu. Tämän jälkeen määriteltiin funktioilla relaatiot jobs- sekä comments-tauluihin, eli tässä tapauksessa asiakkaalla voi olla monta työtä sekä monta kommenttia.

Otetaan esimerkkinä kaksi tapaa lisätä customers-tauluun nimi sekä osoite. Ensiksi käsitellään, kuinka tämä tehtäisiin SQL-lausekkeella.

```php
<?php

$query = "
    INSERT INTO
        customers
    VALUES (
        '{$customer->name}',
        '{$customer->address}'
    );
";

?>
```

KUVIO 25. Rivien lisääminen customers-tauluun

Kuviossa 25 nähdään, miten SQL-lausekkeella lisätään rivejä customers-tauluun. Tämä on vielä melko yksinkertainen lauseke, kun kyseessä on vain kaksi riviä. Tauluun voidaan kuitenkin haluta lisätä kymmeniä rivejä,

jolloin lausekkeesta tulee pitkä ja sekava. Objektien avulla sama työ saadaan helpommin tehtyä (kuvio 26).

```
<?php

$customer = new Customer;
$customer->name = 'Maija Mehilainen';
$customer->address = 'Auringonkukkakatu 1';
$customer->save();

?>
```

KUVIO 26. Rivien lisääminen customers-tauluun objektien avulla

9.3 Näkymät

Näkymissä käsitellään projektin visuaalinen osuus. Kuviosta 27 nähdään osio asiakas-näkymästä. Tässäkin käytetään objekteja, kun halutaan hakea tietokannasta tietoa. Tässä tapauksessa haluttiin nähdä asiakkaalle kuuluvat työt. $customerInfo-objekti määrittelee, että haetaan tietylle asiakas id:lle kuuluvat työt jobs-taulusta. Kyseinen objekti on määritelty Customer-kontrollerissa. Tämän jälkeen asiakkaan työt listataan allekkain siten, että näkyviin tulee laitteen nimi ja samalla siitä tulee linkki kyseisen työn sivulle.

```
<div id='jobs'>
    <h2>Työt</h2>

    <?php
    $customerJobs = $customerInfo->jobs()->get();

    foreach($customerJobs as $key => $value){
        print "<a href=\"../job/".$value->id."\">".$value->device."</a>"."<br>";
    }
    ?>

    <br><a href="{{ URL::route('newJob', array('customerId' => $customerInfo->id)); }}">Lisää työ</a>
</div>
```

KUVIO 27. Asiakkaan työt listattuna sekä työn lisäys asiakas-näkymässä

Alimpana kuvassa on vielä Lisää työ -linkki. Kuten voidaan huomata, linkin muoto ei ole perus HTML-muodossa. Osoitteen kohdalla näkyy reitti, mikä ohjaa newJob-näkymään ja samalla määritellään, mille asiakkaalle tämä uusi työ tehdään.

9.4 Kontrollerit

Tässä työssä asiakkaalle, työlle, kommenteille sekä tuotteille oli omat kontrollerit. Asiakkaan kontrollerissa käsiteltiin muun muassa asiakkaan lisäys, muokkaus sekä poisto. Töiden kontrollerissa käsiteltiin taas työn lisäys, muokkaus sekä poisto. Tämän lisäksi töiden kommentointi on työ-kontrollerissa, asiakas-sivun kommentoinnille on oma kommentti-kontrolleri. Myös palveluille sekä käyttäjille on omat kontrollerinsa.

```
public function updateCustomer($id)
{
    $rules = array(
        'name'    => 'required|max:40',
        'address' => 'required|max:50',
        'phone' => 'required|min:9',
        'email' => 'email'
    );
    $validator = Validator::make(Input::all(), $rules);

    if ($validator->fails()) {
        return Redirect::to('customer/' . $id . '/editCustomer')
            ->withErrors($validator)
            ->withInput(Input::except('password'));
    } else {
        $customer = Customer::find($id);
        $customer->name = Input::get('name');
        $customer->address = Input::get('address');
        $customer->phone = Input::get('phone');
        $customer->email = Input::get('email');
        $customer->save();

        Session::flash('message', 'Tallennus onnistui!');
        return Redirect::action('CustomerController@showCustomer',array('customerId'=>$id));
    }
}
```

KUVIO 28. Asiakkaan tietojen päivitys kontrollerissa

Asiakkaan tietojen päivitys (kuvio 28) näyttää melko yksinkertaiselta. Tässä haluttiin tehdä muutoksia tietokannassa olevaan customer-tauluun. Aluksi annettiin säännöt, minkälaisessa muodossa taulun rivien tulee olla asiakkaan tietoja muutettaessa. Nimi, osoite ja puhelinnumero ovat pakollisia tietoja. Nimen maksimipituus on 40 merkkiä, osoitteen 50 merkkiä ja puhelinnumero voi minimissään olla 9 merkkiä. Sähköpostiosoite tulisi antaa myös oikeassa muodossa.

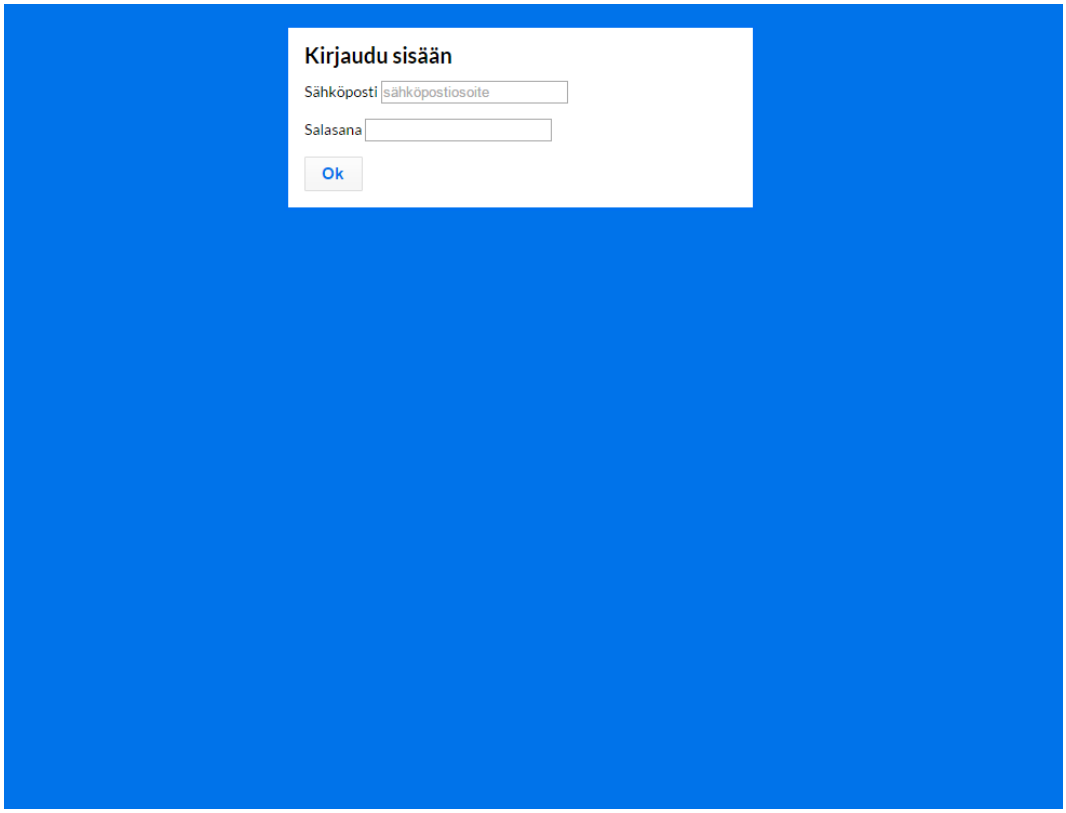
Asiakkaiden tietojen syötön jälkeen tapahtuu validointi. Jos jokin säännöistä ei täsmää käyttäjän antaman tiedon kanssa, ohjataan käyttäjä haluamalle sivulle ja käyttöliittymästä näkee, minkälainen virhe ilmeni. Kaiken ollessa kunnossa, käyttäjän antamat päivitetyt tiedot talletetaan

tietokantaan asiakkaan vanhojen tietojen päälle. Asiakas löytyy oman id:n avulla. Onnistuneesta tallennuksesta tulee viesti käyttäjälle sillä sivulla, mihin käyttäjä on ohjattu.

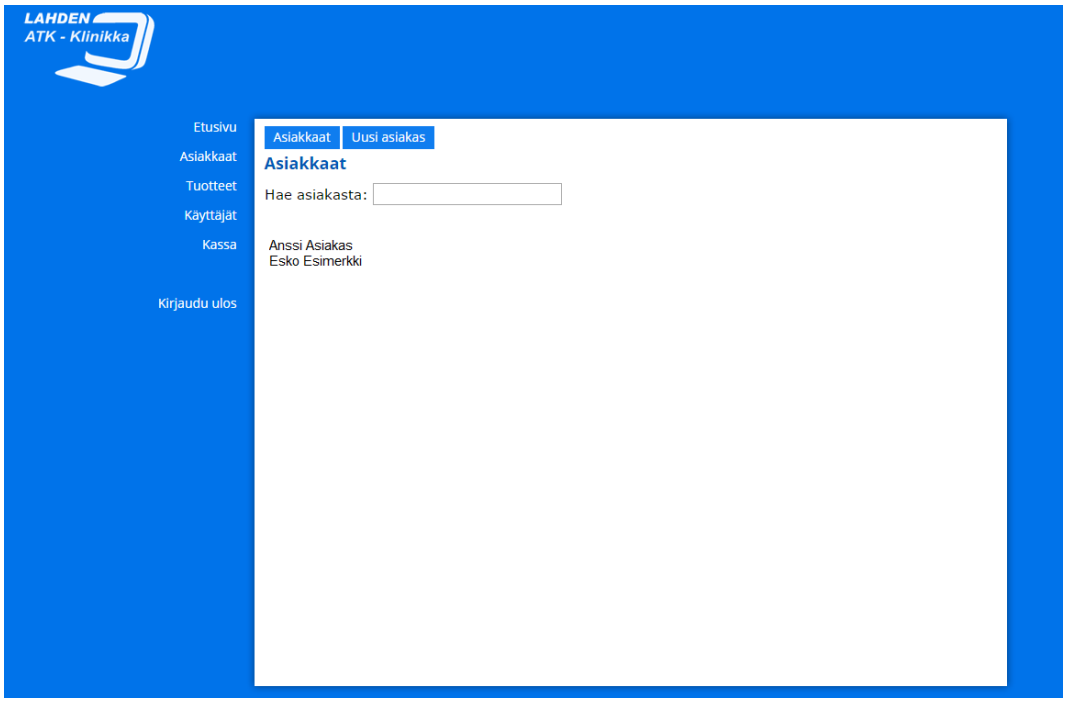
Jotta voidaan yhdistää URI kontrolleriin, täytyy määritellä uusi reitti routes.php-tiedostoon. Merkkijono sisältää kaksi osaa, joiden välissä on @-merkki. Näin saadaan muodostettua kuviossa 1 näkyvä ”CustomerController@showCustomer”. Asiakkaan kontrolleri sisältää showCustomer-funktion, johon käyttäjä ohjataan päivitettyään asiakkaan tiedot.

9.5 Valmis käyttöliittymä

Yrityksen jokaisella työntekijällä on käytössään oma sähköpostiosoite sekä salasana, joiden avulla voidaan kirjautua sisään sivustolle kuvion 29 mukaisesti.

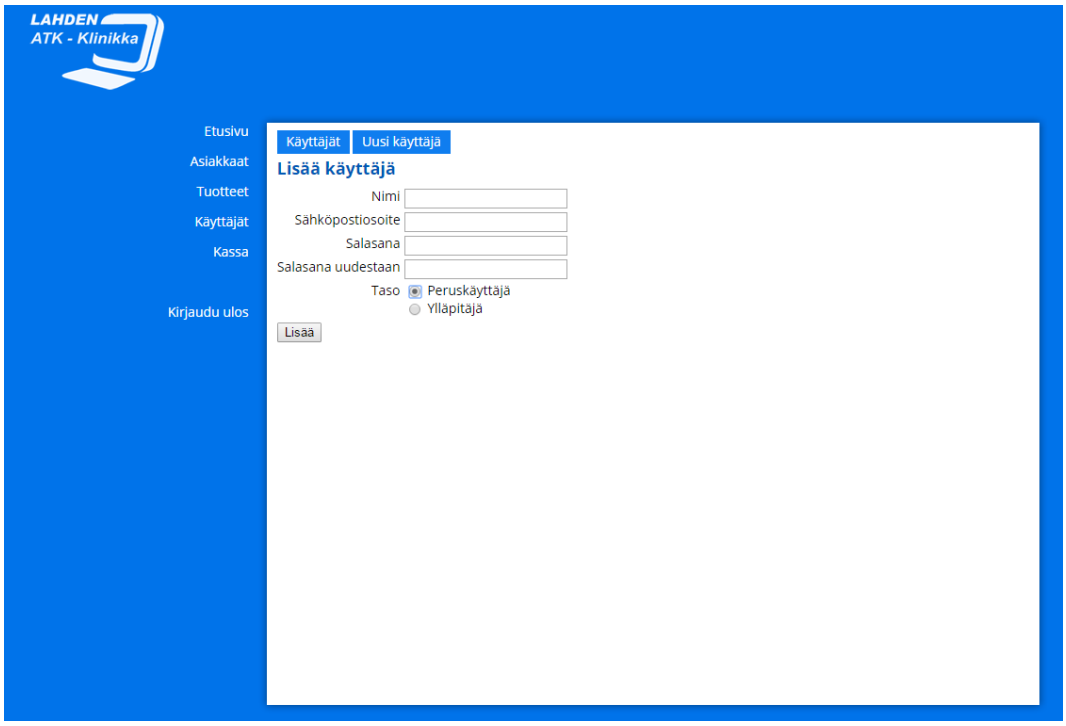


KUVIO 29. Kirjautuminen sivustolle



KUVIO 30. Etusivun näkymä käyttöliittymässä

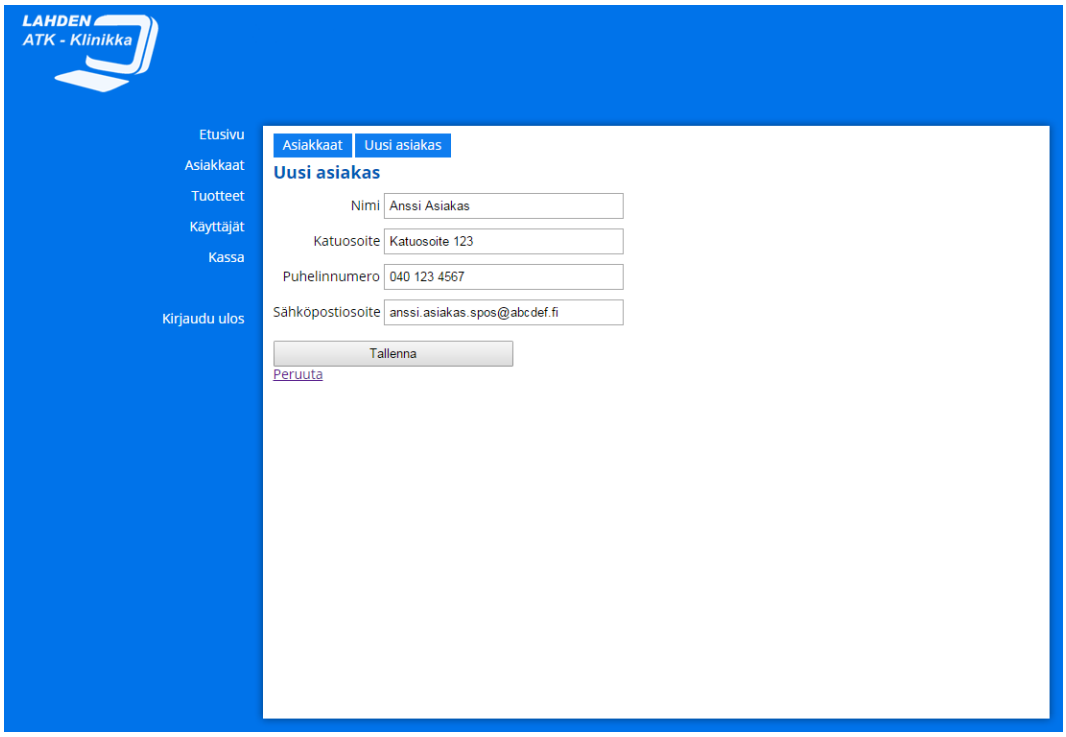
Kuten kuviossa 30 nähdään, etusivusta tehtiin mahdollisimman yksinkertainen. Tämän takia siellä tulisi näkyä vain navigaatiolinkit muille sivuille sekä asiakasnäkymä. Asiakkaat näkyvät allekkain siinä järjestyksessä, kun heidät otettiin vastaan. Työntekijä voi hakea asiakasta nimellä sekä lisätä uuden asiakkaan tältä sivulta. Navigaatiolinkeistä pääsee myös katsomaan yrityksen tuotteita, tarkastelemaan muita työntekijöitä sekä kirjautumaan ulos. Jokainen käyttäjä kirjautuu sivustolle omalla sähköpostiosoitteellaan sekä salasanalla. Käyttäjän nimi näkyy jokaisessa hänen tekemässään asiakkaan tai työn lisäyksessä sekä kommenteissa. Tämä helpottaa yrityksen toimintaa etenkin vilkkaana aikana. Jos tietokoneita on paljon korjattavana ja työntekijöitä on monia, on hankala tietää, kuka on tehnyt mitäkin. Sivuston tallentaman tiedon ansiosta tiedetään, keneltä työntekijältä voidaan kysyä tietokoneen korjauksen edistymisestä esimerkiksi asiakkaan tiedusteltaessa asiaa.



KUVIO 31. Uuden käyttäjän lisääminen lomakkeella

Yrityksen työntekijöiden vaihtuvuus on suuri, sillä työntekijät ovat suurimmaksi osaksi työharjoittelijoita. Tämän vuoksi käyttäjien lisääminen ja poistaminen täytyy olla helposti tehtävissä sivuston kautta. Käyttäjän voi lisätä ja poistaa ainoastaan ylläpitäjä eli yrityksessä vastuussa olevat henkilöt. Lisäys on helppoa lomakkeen avulla (kuvio 31) mihin ylläpitäjä tallentaa uuden työntekijän tiedot. Työntekijä itse saa päättää salasanansa turvallisuussyistä.

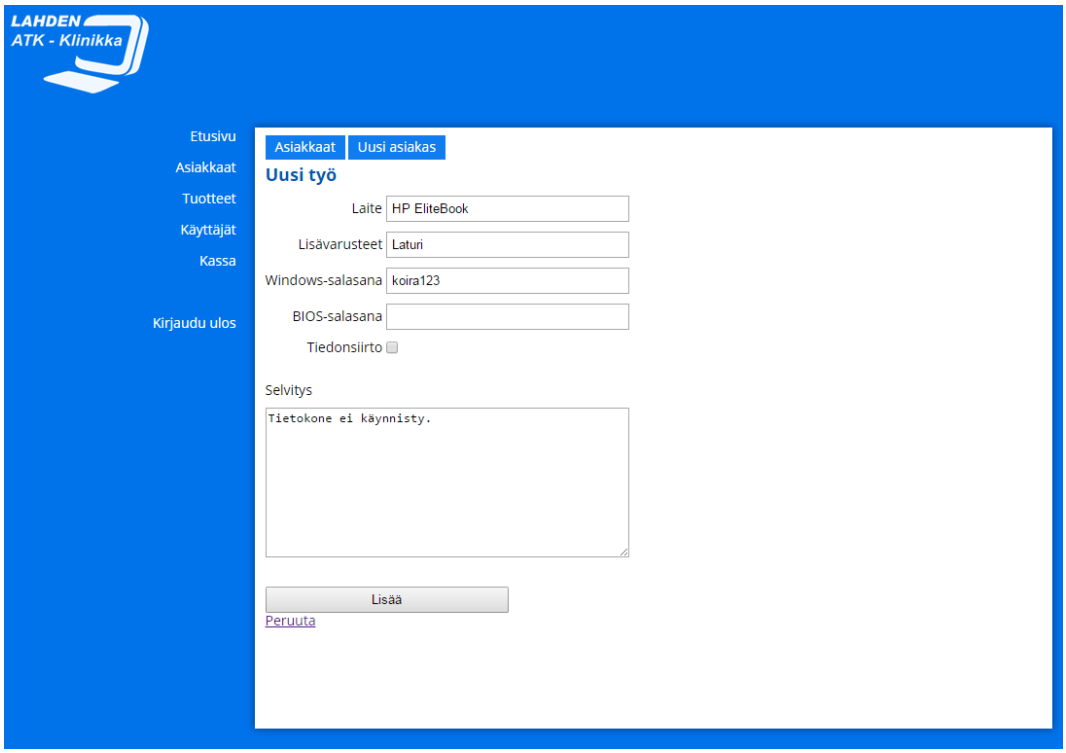
Työntekijät voivat käyttäjät-sivulta tarkastella toisia työntekijöitä. Sivulla näkyvät vain käyttäjien nimet linkkeinä. Linkkiä painamalla päästään sivulle, missä näkyy käyttäjän nimi, sähköpostiosoite sekä käyttäjän taso. Työntekijät voivat nähdä sivulta myös kaikki käyttäjän jättämät kommentit asiakkaan tai työn tietoihin.



KUVIO 32. Asiakkaan lisääminen lomakkeella

Käyttäjät pystyvät lisäämään asiakkaita vaivattomasti. Aikaisemmin asiakkaan tiedot tallentuivat Excel-tiedostoina päätietokoneelle. Tämä oli kuitenkin riski, sillä tietokoneen kovalevy voi hajota melko yllättäenkin. Silloin asiakkaidenkin tiedot katoaisivat. Nyt tiedot tallentuvat palvelimen tietokantaan, johon pääsee käsiksi miltä tahansa tietokoneelta.

Asiakkaan tietoihin (kuvio 32) täytyy tallentaa nimi, katuosoite sekä puhelinnumero. Sähköpostiosoite on valinnainen. Puhelinnumerossa täytyy olla tietty määrä numeroita sekä sähköpostin pitää olla oikeassa muodossaan, jotta virhesanomaa ei tule. Jos virhesanoma ilmestyy, tulee näkyviin kenttä, missä mainitaan virheen muodosta. Esimerkiksi virhesanoma voi olla ”Osoite-kenttä on pakollinen”.



KUVIO 33. Työn lisääminen lomakkeella

Työn lisääminen (kuvio 33) ei myöskään eroa kovinkaan paljoa asiakkaan lisäämisesta. Työ lisätään asiakkaan tietoihin ja siihen merkitään korjattava laite, asiakkaan tuomat lisävarusteet, mahdolliset salasanat, tarvitaanko tiedonsiirtoa sekä selvitys viasta. Erityisen tärkeää on laittaa kaikki tieto oikein. Tietokoneita voi olla samaan aikaan monia kappaleita korjattavana. Yrityksellä itsellään on myös asiakkaiden tietokoneita varten varalatureita tai muita lisävarusteista, jos asiakkaalla on laturi jäänyt kotiin. Siksi onkin tärkeää laittaa ylös, jos asiakas on tuonut jotakin laitteensa lisäksi, jotta varusteet eivät mene sekaisin muiden asiakkaiden tai yrityksen varusteiden kanssa. Tapana onkin ollut laittaa asiakkaan nimi kaikkiin varusteisiin, mutta tämä on hyvä lukea myös työn tiedoissa.

Tiedonsiirto-laatikko voidaan raksia, jos asiakkaalla on tärkeitä tietoja tietokoneellaan. Jos tietokoneen käyttöjärjestelmä täytyy uudelleenasentaa, voidaan asiakkaalta vielä tarkistaa, mitä hän tarkalleen ottaen haluaa säilyttää.

Selvitys tehdään myös mahdollisimman tarkasti. Hän, kuka tietokoneen ottaa vastaan, ei välttämättä ole seuraavana päivänä töissä. Siksi tarkka

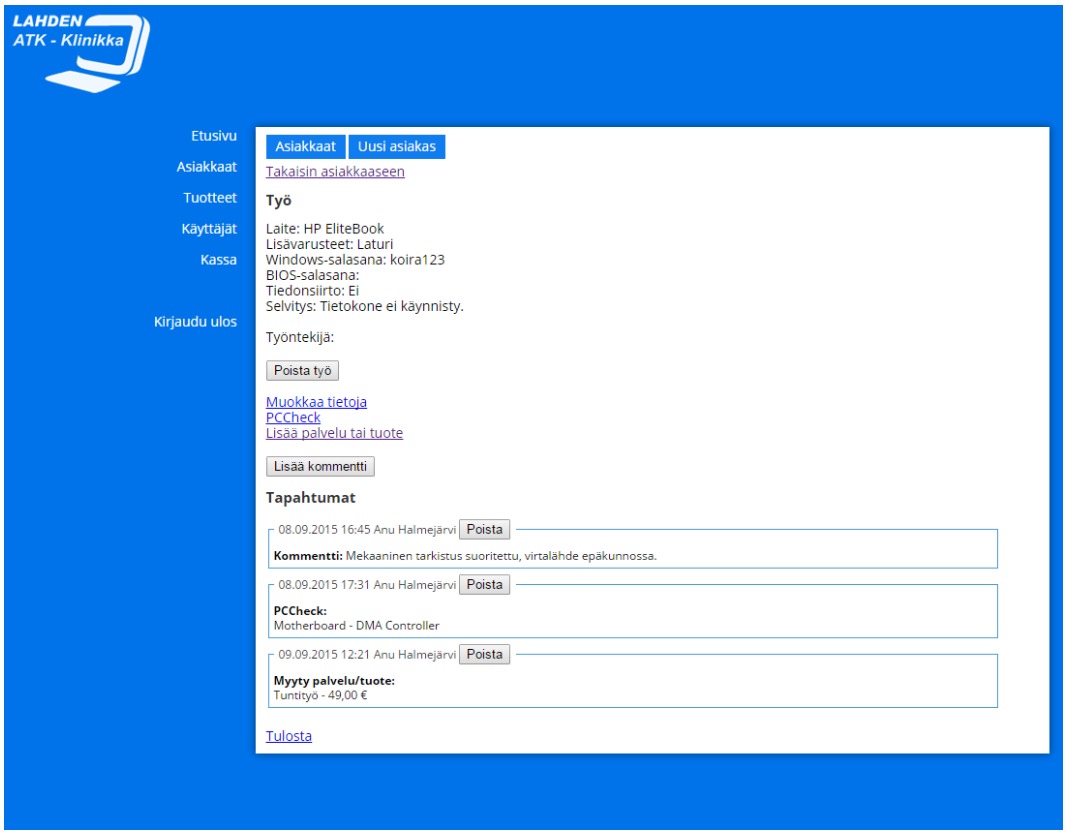
selvitys onkin tärkeä, jotta muut työntekijät voivat tietää ongelman laadusta ja heidän on helpompi lähteä selvittämään vikaa.

Työn tallentamisen jälkeen avautuu näkymä työstä (kuvio 34). Työn tiedot näkyvät ensimmäisenä ja alempana ovat muut tärkeät linkit sekä kommentit. Työn voi poistaa ainoastaan ylläpitäjä-tason käyttäjä. ”Poista työ”-painike ei näy peruskäyttäjille. Peruskäyttäjät voivat kuitenkin muokata työn tietoja.

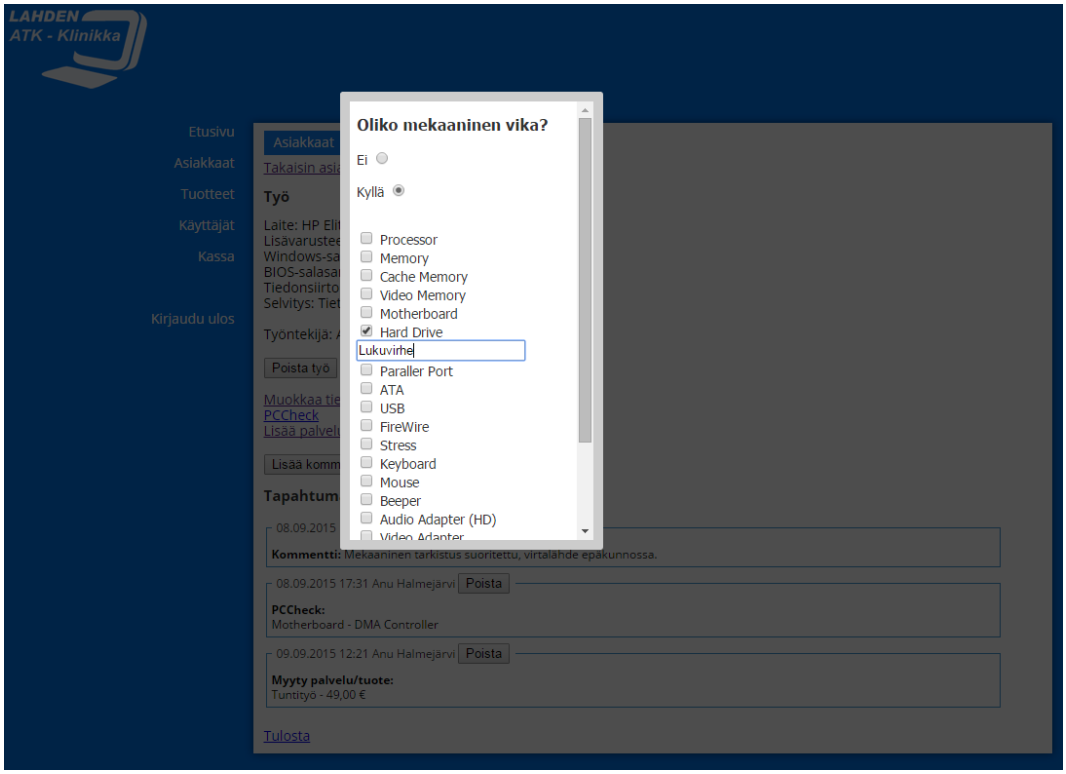
Tietokoneen tarkistus on myös toimenpide, mikä tehdään melkein jokaiselle huoltoon tulleelle tietokoneelle. Tämä on mekaaninen tarkistus, jossa katsotaan, onko tietokoneen vika laitteistossa. Käytössä on PcCheck-niminen ohjelma, joka tarkastaa ennalta määritellyt osat. Nämä on myös lisätty tietokantaan omaan tauluunsa ja käyttäjä voi käyttöliittymässä myös tässäkin kohdassa klikata mahdolliset virhetulokset työn sisällä. Tämä minimoi kirjoitusvirheiden mahdollisuudet ja helpottaa myös työntekijöiden työtä (kuvio 35).

Työhön kannattaa myös lisätä kommentteja tehtyjen vianselvitysten jälkeen. Näin muutkin työntekijät tietävät, mitä tietokoneelle on tehty ja mitä on selvinnyt.

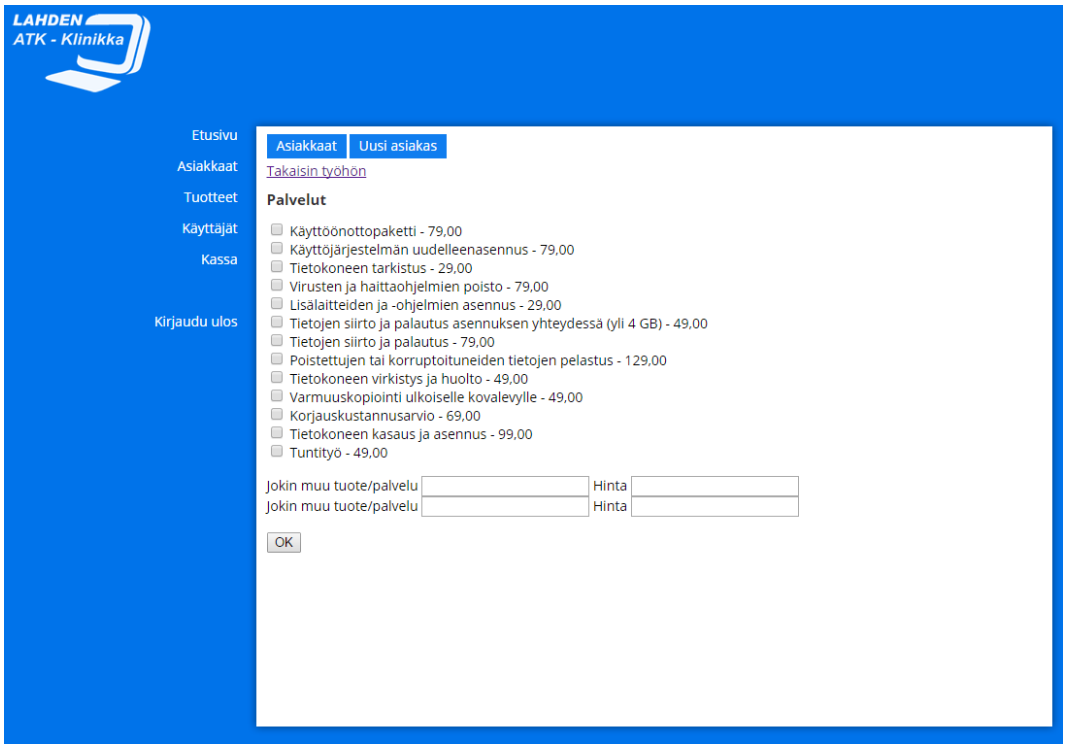
Kun työntekijä on saanut työn tehdyksi, hän lisää palvelut tai tuotteet työn alle (kuvio 36). Tässä näkymässä lisätään kaikki palvelut, mitä asiakkaan tietokoneelle on tehty. Myös tuotteita voidaan lisätä. Esimerkiksi, jos asiakkaan tietokoneessa on jokin osa mennyt rikki, voidaan hänelle tilata uusi vastaava osa. Asiakas saa työstään tulostetun version, missä näkyvät kommentit ja työn tiedot. Työntekijän tehtävänä on myös tehdä tarkempi kuvaus, mitä tietokoneelle on tehty. Myöhemmin saman asiakkaan tietokoneeseen voi ilmentyä ongelmia, joten on hyvä merkitä muistiin kaikki, mitä tietokoneelle on aiemmin tehty.



KUVIO 34. Työn näkymä sivustolla



KUVIO 35. Mekaanisen tarkistuksen lisäys työn alle



KUVIO 36. Palvelun tai tuotteen lisäys työn alle

9.6 Ohjelmiston testaaminen

Ohjelmiston tuli olla sellainen, mitä yritys halusi. Testaajaksi ei voinut laittaa ketä tahansa, sillä ohjelmisto oli ainoastaan yrityksen omaan käyttöön. Ulkopuolisia ei voinut tähän siis pyytää testaajaksi.

Ohjelmistoa ei testattu ainoastaan sen ollessa täysin valmis. Yrityksen työntekijät testasivat ohjelmistoa myös sen toteutuksen eri vaiheissa. Tämän avulla ainakin isommat virheet huomattiin heti ja saatiin korjattua. Näin päästiin taas eteenpäin eivätkä nämä kyseiset virheet enää vaivanneet myöhemmin.

Vaikka virheitä ei olisikaan ollut, niin testaajat myös huomasivat, jos jokin asia kannatti esittää eri tavalla. Testauksen ansiosta ohjelmistosta tuli sellainen, mikä oli mahdollisimman tehokas ja helppokäyttöinen työntekijöiden näkökulmastakin.

10 YHTEENVETO

Tavoitteena tietokannan ja käyttöliittymän kehitys kuulosti yksinkertaiselta toteuttaa, mutta projekti oli ajateltua laajempi ja monimutkaisempi. Ohjelmiston työstäminen aloitettiin syyskuussa 2014.

Ohjelmiston toteuttamiseen saatiin melko vapaat kädet. Yritys kertoi, mitä halusi ohjelmistoon ja nämä asiat toteutettiin. Lisäksi työhön tehtiin muita helpottavia lisäyksiä. Isoista muutoksista aina neuvoteltiin. Koska työskentely tapahtui yrityksen tiloissa, oli neuvottelu ja kommunikointi helppoa.

Tässä työssä haluttiin tutkia, kuinka tämä ohjelmisto saadaan myös toimivuuden lisäksi turvalliseksi ja helpoksi käyttää. Työssä on otettu huomioon yrityksen työntekijöiden tarpeet ja heidän työnsä helpottaminen. Tämä oli helppo toteuttaa, sillä kokemusta kyseisen yrityksen työntekijänä olemisesta löytyy.

Ongelmia työn kanssa ei juurikaan ollut. Ainoa isompi vika ilmeni välillä yrityksen palvelimessa, joka useasti lakkasi toimimasta, mutta sekin saatiin yrityksen puolesta korjattua. Välillä ylimääräistä aikaa kului ohjelmoinnissa jonkin tarvittavan tiedon etsimiseen, sillä Laravelin käyttö ei ollut entuudestaan tuttua.

Lopputulos oli sitä mitä yritys halusikin. Ohjelmisto on tällä hetkellä käytössä yrityksen toiminnassa joka päivä. Yrityksessä on ohjelmiston tunteva henkilö, joka pystyy tarvittaessa päivittämään tai tekemään lisäyksiä joko tietokantaan tai käyttöliittymään.

Ohjelmaan on vielä kehitteillä lisäosa, jonka avulla työntekijät voivat kartoittaa varaston tilannetta, eli kuinka paljon on erilaisia osia ja laitteita. Tämä helpottaa esimerkiksi yrityksen inventaariota ja laitteiden myynnin seuraamista.

LÄHTEET

Auer, L. 2005a. Käytettävyydestä. Virtuaali ammattikorkeakoulu [viitattu 31.3.2015]. Saatavissa: http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/03030811116763481 38/1111677021119/1111677206424/1111677569162.html

Auer, L. 2005b. Nielsenin säännöt. Virtuaali ammattikorkeakoulu [viitattu 31.3.2015]. Saatavissa: http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/03030811116763481 38/1111677021119/1161290796532/1161290917294.html

Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2004. Käsitteellinen mallintaminen. Jyväskylän yliopiston IT-tiedekunta ja avoin yliopisto [viitattu 1.12.2014]. Saatavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/suunnittelu/index2.html

Johdatus PHP –kieleen. 2015. [viitattu 31.3.2015]. Saatavissa: http://users.jyu.fi/~kolli/ITK215_05/php/?sivu=johdanto

Otwell, T. 2015. Laravel Introduction [viitattu 31.3.2015]. Saatavissa: http://laravel.com/docs/4.2/introduction

Pontikis, C. 2013. How to Use PHP Improved MySQLi extension (and Why You Should) [viitattu 20.4.2017]. Saatavissa: http://www.pontikis.net/blog/how-to-use-php-improved-mysqli-extension-and-why-you-should

Rees, D. 2012-2014. Code Bright [viitattu 31.3.2017].

Ristikangas, M. 2005. Paikkatietojen suojausmenetelmät tietoteknisestä näkökulmasta. Teknillinen korkeakoulu. Diplomityö [viitattu 18.4.2017]. Saatavissa: http://builtenv.aalto.fi/fi/midcom-serveattachmentguid-1e3bf13a3eadf2abf1311e391bc4f13d8c720f320f3/2005_ristikangas_m.pd

Suomen Internetopas. 2017. [viitattu 18.4.2017]. Saatavissa: http://www.internetopas.com/yleistietoa/tietoturva/

Vanhala-Nurmi, V. 2014. Rautalankamalli. Haaga-Helia [viitattu 31.3.2015]. Saatavissa: http://myy.haaga-helia.fi/~vanvu/www/suunnittelu/rautalankamalli.html