jamk.fi

Potilaskirjausjärjestelmä LAMP-alustalla

Joonas Hujanen

Opinnäytetyö
Toukokuu 2016
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Hujanen, Joonas	Opinnäytetyö, AMK	Toukokuu 2016
	Sivumäärä 44	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Potilaskirjausjärjestelmä LAMP-alustalla

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Ari Rantala

Toimeksiantaja(t)
Serecom Ky

Tiivistelmä

Opinnäytetyönä toteutettiin Serecom Ky:n tilaama potilaskirjausjärjestelmä-sovellus. Tuotteen potentiaalisena käyttäjäryhmänä ovat kotikäyntejä tekevät sairaanhoitajat, jotka ovat aiemmin kirjanneet potilaskäynnin aikana suoritetut toimenpiteet paperille, josta tiedot on sitten haettu laskutusta varten.

Potilaskirjausjärjestelmän tavoitteena oli tuottaa palvelu, jossa käyntien kirjaus tapahtuu sähköisesti, jolloin esimerkiksi asiakasta laskuttaessa tiedot ovat helposti saatavilla yhdessä paikassa. Serecom Ky toivoi, että sovellusta voi käyttää helposti myös mobiililaitteilla, mikä mahdollistaa helpon kirjauksen paikan päällä. Toteutustavaksi valittiin verkkosivuston ohjelmointi käyttäen kehitykseen valmista MVC-kehysmallia ja relaatiotietokantaa, johon asiakkaiden käynnit kirjattiin.

Lopputuloksena saatiin verkkosivusto, johon asiakas pystyi rekisteröitymään käyttäjäksi, kirjautumaan sisään ja ulos, lisäämään uusia käyntejä, tarkastelemaan aiemmin kirjattuja käyntejä joko suppeasti kaikkia kerralla tai yksittäisiä käyntejä yksityiskohtaisemmin. Lisäksi käyttäjä pystyy lisäämään valitulle käynnille uusia toimenpiteitä ja tarkastelemaan niitä osana käynnin yksityiskohtaisempaa tarkastelua. Toteutettu sivusto sovitetaan dynaamisesti käyttäjän päätelaitteen ruudulle sopivaan muotoon kolmannen osapuolen kirjaston avulla. Näin sivusto soveltuu hyvin myös mobiilikäyttöön.

Toteutuksella täytettiin kaikki asiakkaan vaatimukset sekä kehittäjän omat parannusehdotukset ja muodostettiin hyvä pohjasovellus mahdollista jatkokehitysprojektia varten.

Avainsanat (asiasanat)
ORM, MVC, LAMP

Muut tiedot

jamk.fi

Description

Author(s) Hujanen, Joonas	Type of publication Bachelor's thesis	Date May 2016
	Number of pages 44	Language of publication: Finnish
		Permission for web publication: x

Title of publication **Patient record system using LAMP platform** Possible subtitle
Degree programme Software engineering
Supervisor(s) Rantala, Ari
Assigned by Serecom Ky

Description

Currently, nurses who make visits to patients' homes record their visit and all the corresponding operations done during said visit with a paper-based memo. The information is then used for invoicing or other similar activities.

The aim of this project was to create a record system where the nurse is able to register as a user and record their visits to patients electrically in an easy and straightforward way. This way, all the patient records are kept in one easily accessible place. The client, Serecom Ky stressed that the system should be usable with a mobile device to facilitate the recording during the visit.

The system was implemented by programming a web page using technologies and techniques such as an MVC-framework and a relational database which stored the users' visit data. The completed product gave the clients an ability to register as a new user, log in and out, add new visits to the system and check them either by accessing general information of all the visits at once or detailed information of one visit at a time. The user could also add new actions to the chosen visit and check them in the details view of said visit.

The completed program fulfilled all of the customer requirements as well as the developer's own improvement ideas for the system. It also formed a good base for possible improvements and expansions.

Keywords (subjects)
ORM, MVC, LAMP

Miscellanous

Sisällys

1. Johdanto 3

2. Vaatimusmäärittely 3

2.1 Tavoite 3

2.2 Vaatimukset 4

3. Teknologiat 4

3.1 ORM 4

3.2 MVC 4

3.3 LAMP 5

3.4 Doctrine2 9

3.5 Zend Framework 2 12

3.6 Oracle VirtualBox 13

3.7 Twitter Bootstrap 13

4. Teknologiavalintojen perusteluja 14

4.1 LAMP 14

4.2 Doctrine 2 15

4.3 Zend Framework 2 15

4.4 Twitter bootstrap 16

4.5 Composer 16

4.6 Oracle VirtualBox 16

5. Toteutus 17

5.1 Työn aloitus 17

5.2 Rekisteröityminen 20

5.3 Sisäänkirjautuminen 23

5.4 Uloskirjautuminen 25

5.5 Käyntien tarkastelu 27

5.6 Käynnin lisäys 30

5.7 Käynnin yksityiskohtien tarkastelu 32

5.8 Käynnin toimenpiteen lisäys 34

5.9 Tietoturva 36

6. Tulokset 38

7. Pohdinta 40

Lähteet 43

Kuviot

Kuvio 1. Apachen markkinaosuus huhtikuussa 2015-2016. (Usage statistics and market share of Apache for websites n.d.) 5

Kuvio 2. Tietokannan käsitemalli 19

Kuvio 3. Käyttötapauskaavio uuden käyttäjän rekisteröitymisestä 21

Kuvio 4. Sekvenssikaavio uuden käyttäjän rekisteröitymisestä 23

Kuvio 5. Käyttötapauskaavio sisäänkirjautumisesta 24

Kuvio 6. Sekvenssikaavio sisäänkirjautumisesta 25

Kuvio 7. Käyttötapauskaavio käyttäjän uloskirjautumisesta 26

Kuvio 8. Sekvenssikaavio käyttäjän uloskirjautumisesta 27

Kuvio 9. Käyttötapauskaavio käyntien tarkastelusta 28

Kuvio 10. Sekvenssikaavio käyntien tarkastelusta 29

Kuvio 11. Käyttötapauskaavio käynnin lisäyksestä 31

Kuvio 12. Sekvenssikaavio käynnin lisäyksestä 32

Kuvio 13. Käyttötapauskaavio käynnin yksityiskohtien tarkastelusta 33

Kuvio 14. Sekvenssikaavio käynnin yksityiskohtien tarkastelusta 34

Kuvio 15. Käyttötapauskaavio toimenpiteen lisäämisestä 35

Kuvio 16. Sekvenssikaavio toimenpiteen lisäämisestä 36

1. Johdanto

Työn toimeksiantajana toimi Serecom Ky. Serecom on kuopiolainen startup -yritys, jonka tuleva liiketoiminta nojaa tuotetun potilaskirjausjärjestelmän tarjoamisesta sairaanhoitajille.

Työn teoriaosassa esitellään työssä käytetyt työkalut ja perustelut niiden valintaan. Tämän jälkeen käydään tarkemmin läpi valittujen työkalujen käyttöön liittyviä seikkoja, etenkin Doctrine 2:n ja Zend Framework 2:n kohdalla.

Toteutusosassa käydään läpi sovelluksen kehitysprosessi. Toteutus alkoi asiakkaan kanssa käydyllä alkupalaverilla ja vaatimusmäärittelyllä. Työn aloitusosiossa käydään myös läpi tietokannan rakennetta sekä kehityspalvelimen pystytysprosessi. Lisäksi käydään läpi jokaisen sivuston osan toteutus.

Opinnäytetyötä aloittaessa Serecom Ky, jolle työ tehtiin, oli tehnyt markkina-analyysin ja todennut, että vastaavia töitä ei ole, vaan kohdeyleisö hoitaa käyntien kirjaukset edelleen kynällä ja paperilla.

2. Vaatimusmäärittely

2.1 Tavoite

Tavoitteena oli tuottaa kirjanpito-ohjelma kotikäyntejä tekeville sairaanhoitajille. Hoitaja voi potilaskäynnin aikana tai sen jälkeen kirjata käynnin tapahtumapäivämäärän, käynnin keston ja mahdolliset lisätoimenpiteet. Ohjelma pitää kirjaa täytetyistä tiedoista tietokannassa ja esittää käyttäjälle käyntihistorian haluttaessa. Käyntihistoriasta ilmeni käyntipäivämäärä, asiakkaan tiedot, käynnin kesto tunneissa ja minuuteissa ja käynnin kokonaishinta mukaan lukien kaikki käynnin aikana suoritetut lisätoimenpiteet.

Käyttäjä voi myös valita yhden käynnin ja tarkastella siihen liittyviä tietoja tarkemmin. Käynnin yksityiskohtasivulla käyttäjälle näytetään edellä mainittujen tietojen lisäksi käynnin tuntitaksa, onko käynti tapahtunut sunnuntaina ja/tai illalla sekä kaikki käynnin aikana suoritetut lisätoimenpiteet, kuten siteiden vaihdot, lääkkeen jakamiset tai muut samankaltaiset, käyttäjän määrittämät toimenpiteet. Yksityiskohtasivulla käyttäjä pystyy myös lisäämään uusia toimenpiteitä valitsemaansa käyntiin.

2.2 Vaatimukset

Asiakkaalla oli useita vaatimuksia ja toiveita sovelluksen kehittämiseen liittyen: sovelluksen itsessään tulee olla käyttäjän itse hallinnoima tuntikirjaustyökalu, joka sisältää eräänlaisen taksakatalogin erilaisille toimille, kuten vaikkapa potilaskäynnille. Käyttäjä voi lisätä käyntikirjaukseen merkintöjä, jotka eritellään käyttäjän vapaasti määrittelemillä tunnisteilla, esimerkiksi ”Siteen vaihto”. Näille merkinnöille käyttäjä voi lisätä hinnan, joka lasketaan kokonaishintaan mukaan. Varsinaiseen käyntikirjaukseen käyttäjä voi lisätä keston, tuntitaksan ja mahdolliset lisät kuten sunnuntai- ja/tai iltalisän, joita käytetään käynnin pohjahinnan laskentaan.

Lisäksi asiakas toivoi, että sovellus toimii hyvin mobiililaitteilla: kohderyhmän henkilöt tekevät asiakkaidensa luona kotikäyntejä, jolloin tietojen kirjaaminen käynnin aikana tulisi tehdä mahdollisimman helpoksi.

3. Teknologiat

3.1 ORM

ORM eli Object-relational mapping on ohjelmointitekniikka, jossa käytetään hyväksi olion metadataa kytkemään oliokoodi relaatiotietokantaan. Olio ohjelmoidaan perinteisten olio-ohjelmointikäytäntöjen mukaisesti sillä erotuksella, että oliolle lisätään metadata, joka sisältää tiedot siitä, miten olion muuttujat kytkeytyvät relaatiotietokannan tauluihin. ORM-kirjasto toimii välikätenä relaatiotietokannan ja oliotason välillä ja muuntaa keskenään yhteensopimattomat tiedot käyttökelpoiseen muotoon. (Object-relational mapping n.d.)

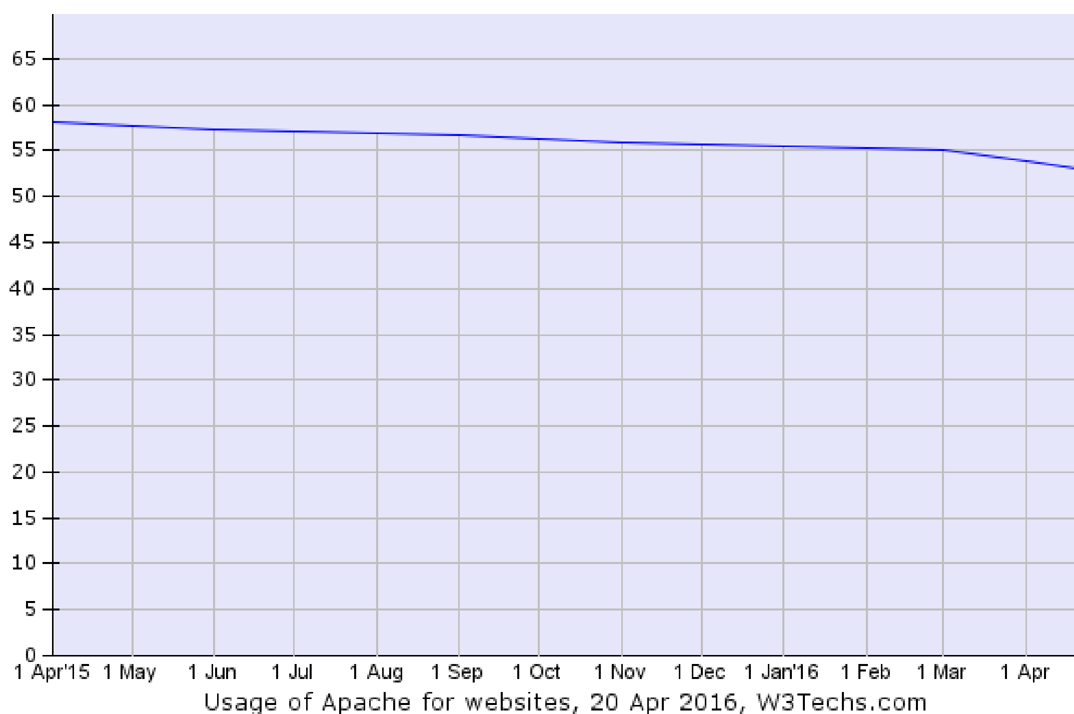
3.2 MVC

MVC on ohjelmistosuunnittelumalli web-sovellusten kehitykseen. Lyhenne MVC tulee sanoista model, view ja controller, jotka kuvaavat MVC:n kolmea komponenttia. Model on matalimman tason malli, joka vastaa tietojen säilyttämisestä. Esimerkiksi sovelluksen oliot voivat olla osa MVC:n model-komponenttia. View-komponentti vastaa modelin säilyttämien tietojen esittelystä käyttäjälle. Web-kehityksessä verkkosivut kuuluvat view-komponentin alle. Controller toimii näiden kahden välikätenä ohjaten view-komponentilta tulevat muutospyynnöt modelille ja modelin tiedot view-komponentille näytettäväksi.

MVC:n suosio perustuu käyttöliittymän ja sovelluslogiikan erotukseen: vain controller voi muokata modelin tietoja ja hakea tiedot view'lle. Vain view voi esittää tietoja käyttöliittymätasolla ja ohjata käyttäjän pyynnöt controllerille. Yksi MVC-mallin mukainen kehysmalli on Zend Framework 2. (Basic MVC Architecture n.d.)

3.3 LAMP

LAMP eli Linux, Apache, MySQL ja PHP on palvelinalusta, joka sisältää Linux-käyttöjärjestelmän, HTTP-palvelimen, MySQL-tietokannan ja PHP-sovelluksen. Tämä yhdistelmä sisältää kaiken tarvittavan verkkosovelluksien kehittämiseen. LAMP on mahdollista asentaa lähes kaikille Linux-jakeluille kuten esimerkiksi Ubuntulle tai RedHat Linuxille. Tällä hetkellä LAMP on hyvin suosittu alusta. W3techs.com-sivuston mukaan noin 53,2% maailman verkkosivustoista toimii jonkinlaisen Apache-pohjan päällä (Ks. kuvio 1) (Usage statistics and market share of Apache for websites n.d.). Linuxin markkinaosuus maailman verkkosivuista taas on noin 36%:n luokkaa (Usage statistics and market share of Linux for websites n.d.).



Kuvio 1. Apachen markkinaosuus huhtikuussa 2015-2016. (Usage statistics and market share of Apache for websites n.d.)

Apache tukee virtual host -käytäntöä, jonka avulla voidaan tarjota monta verkkosivustoa samalta verkkopalvelimelta käsin. Lisäksi, koska virtual hostin voi määrittää mihin tahansa kansioon verkkopalvelimella, voidaan verkkosivuston juurihakemisto määrittää esimerkiksi git-repositorioon, jolloin sivuston uusien versioiden julkaisu on helpompaa. Virtual host määritellään lisäämällä kansioon **/etc/apache2/sites-available/** uusi tiedosto, joka sisältää halutun url-osoitteen, esimerkiksi tässä projektissa käytössä olevaa tiedostonimeä potilaskirjaus.com.conf, joka sisältää seuraavat tiedot:

```
<VirtualHost *:80>
   ServerName potilaskirjaus.com
   ServerAlias www.potilaskirjaus.com
   DocumentRoot /path/to/index/page
   SetEnv APPLICATION_ENV ”development”
   <Directory /path/to/index/page>
                  DirectoryIndex index.php
                  AllowOverride All
                  Order allow, deny
                  Allow from all
                  Require all granted
   </Directory
</VirtualHost>
```

Virtual hostin määrittely aloitetaan aina <VirtualHost>-tunnisteella. Tämä tunniste voi esimerkin tavoin sisältää mahdollisen ip-osoitteen ja TCP-portin, jota kyseinen virtual host kuuntelee. Tässä esimerkissä ainoastaan portti 80 on määritelty eli verkkosivu toimii kaikilla verkkopalvelimen ip-osoitteilla käytettäessä yhteyden muodostamiseen porttia 80. (Apache Core Features, <VirtualHost> Directive n.d.)

Seuraavana esitellään palvelimen nimi. Tämä sisältää verkkosivuston verkkotunnuksen eli domainin. Mikäli verkkosivustolle ei ole tarjolla domainia, ServerName-tietoa voidaan käyttää myös lokaalisti muokkaamalla esimerkiksi Linuxin hosts-tiedostoa ja reitittämällä palvelimen ip-osoite ServerNamen osoittamaan domainiin lisäämällä hosts-tiedostoon uusi rivi: **192.168.0.102 potilaskirjaus.com**, joka kertoo palveli-

melle, että kaikki potilaskirjaus.com-osoitteeseen tulevat pyynnöt ohjataan ip-osoitteeseen 192.168.0.102. ServerAlias sisältää nimensä mukaisesti vaihtoehtoisen palvelimen nimen, johon navigoimalla saadaan myös sama verkkosivusto, kuin ServerNamen osoitteella. (Apache Core Features, ServerName Directive n.d.)

DocumentRoot määrittää verkkosivuston juurihakemiston sijainnin verkkopalvelimella. DocumentRootin avulla apache tietää, mistä verkkosivusto tarjotaan käyttäjille. Ilman kyseistä tietoa käyttäjä ei voi selata sivustoa. DocumentRoot osoittaa aina verkkosivuston etusivun sisältävään hakemistoon. (Apache Core Features, DocumentRoot Directive n.d.)

SetEnv asettaa virtual hostiin ympäristömuuttujia kehittäjän määrittämillä arvoilla. Esimerkissä SetEnviä käytetään asettamaan APPLICATION_ENV-muuttujalle arvo development. Tätä ei välttämättä tarvitse käyttää, mutta ympäristömuuttujien määrittäminen voi auttaa esimerkiksi virheilmoitusten käsittelyssä, jos sovelluksen halutaan ilmoittavan virheistä eri tavalla kehitys- ja tuotantoversioissa. (Apache Module mod_env, SetEnv Directive n.d.)

Directory-osio määrittää palvelimen tiedostojärjestelmän osiin tiettyjä sääntöjä, pääosin käyttöoikeuksiin liittyen. Directory-tunnisteen sisällä määritellään hakemisto, jolle halutaan määrittää lisäsääntöjä. Directory-osiossa DirectoryIndex määrittää hakemiston index-tiedoston nimen, tässä esimerkissä tiedostonimi on index.php. (Apache Core Features, <Directory> Directive n.d.)

AllowOverride määrittää, mitkä virtual host-ohjeet voidaan ylikirjoittaa sivuston omalla .htaccess-tiedostolla tarvittaessa. Sallitut arvot ovat All, None tai jokin tietty ohjetyyppi. None on vakioarvo. Kun AllowOverriden arvo on None, verkkosivuston .htaccess-tiedosto jätetään täysin huomiotta. Mikäli arvo on All, ylikirjoittaa .htaccess kaikki ohjeet, jotka tiedostosta löytyvät. Mikäli AllowOverrideen on määritetty jokin tietty ohjetyypi, voi .htaccess ylikirjoittaa vain sen tyyppisiä ohjeita. (Apache Core Features, AllowOverride Directive n.d.)

Order määrittää, missä järjestyksessä Allow ja Deny-ohjeet käydään läpi käyttöoikeuksia tarkasteltaessa. Esimerkissä järjestys on allow, deny eli ensin palvelin käy läpi määritellyt Allow-ohjeet, joista tulee löytyä vähintään yksi osuma tai asiakkaan tekemä pyyntö palvelimelle hylätään. Tämän jälkeen palvelin käy läpi määritellyt Deny-

ohjeet ja jos aiempi pyyntö täyttää jonkin näistä ehdoista, pyyntö jälleen kerran evätään. Allow ja Deny-ohjeet määritellään erikseen. (Apache Core Features, Order Directive n.d.)

Allow-ohje määrittelee, millä domaineilla on pääsy Directoryn määrittelemään hakemistoon. Esimerkin Allow from all sallii kaikkien domainien pääsyn hakemistoon (Apache Module mod_access_compat, Allow Directive. N.d.). Require-ohjeella määritellään ne vaatimukset, joilla tunnistetut käyttäjät pääsevät verkkosivustolle. Esimerkin Require all granted takaa pääsyn kaikille ilman erillisiä ehtoja (Apache Module mod_authz_core, Require Directiven n.d.).

Kun virtual host konfigurointitiedosto on tallennettu, voidaan sivusto ottaa Ubuntussa käyttöön komennolla **sudo a2ensite /path/to/virtualhost/config**. Komento linkittää valitun konfiguroinnin /etc/apache2/sites-enabled/-kansioon, josta Apache 2 selvittää, mitkä sivut ovat käytettävissä. Kun sivusto on otettu käyttöön, tulee Apache 2 vielä käynnistää uudelleen komennolla **sudo service apache2 restart**, jotta saadaan Apache 2 lataamaan uudet asetukset.

Joskus valitut verkkosivustomallit, kuten Zend Framework 2 saattavat tarvita käyttöönsä moduleita, joita Apachessa ei oletusarvoisesti ole käytössä. Tällöin kyseiset modulit voidaan ottaa Ubuntussa käyttöön komennolla **sudo a2enmod modname**. Zend Framework 2:n tapauksessa tarvitaan rewrite-nimistä modulia, jolloin komennon lopullinen muoto on **sudo a2enmod rewrite**. Kun moduli on otettu käyttöön, täytyy Apache 2:n ladata konfiguroinnit uudelleen komennolla **sudo service apache2 reload**.

PHP on lyhenne sanoista PHP Hypertext Processor. Se on avoimen lähdekoodin skriptikieli, joka on suunniteltu eritoten verkkokehitykseen ja sitä voidaan upottaa HTML dokumentteihin. Sen syntaksi on saanut vaikutteita C-kielestä, Javasta ja Perlistä. Kielen päätavoite on mahdollistaa nopea dynaamisten verkkosivustojen kehittäminen. (Preface n.d.)

MySQL on Oraclen ylläpitämä avoimen lähdekoodin relaatiotietokantasovellus, joka mahdollistaa erinäisten tietojen, kuten käyttäjätietojen, säilömisen verkkopalvelimelle myöhempää käyttöä varten. Usein MySQL käytetään mm. PHP:n, joka sisältää valmiita metodeja MySQL-tietokannan käyttöön, kanssa.

MySQL:ssa tiedot tallennetaan erillisiin tietokantoihin, joita tietokannan ylläpitäjä voi luoda ja poistaa tarvittaessa. MySQL tukee useita tietokantoja samassa järjestelmässä. Uusi tietokanta luodaan MySQL-kyselyllä **CREATE DATABASE dbName**. (Creating and selecting a database n.d.)

3.4 Doctrine2

Doctrine2 on PHP:lle kehitetty ORM eli Object Relational Mapping-kirjasto. Doctrine2 mahdollistaa relaatiotietokannan taulujen muodostamisen olion lähdekoodiin kirjoitetun metatiedon perusteella. Metatietoon sisältyy mm. olion muuttujatasolla määritellyt tietokantataulun sarakkeen tyyppi, kuten esimerkiksi merkkijono, sekä siihen liittyvät lisätiedot, esimerkiksi merkkijonon pituus. Esimerkiksi, kun tietokantataululle halutaan muodostaa automaattisesti kasvava tunniste, muodostetaan lisäämällä olion lähdekoodiin seuraavanlainen metatieto:

```
/**
 * Käynnin ID
 *
 * @Id()
 * @Column(type=”integer”)
 * @GeneratedValue(strategy=”AUTO”)
 */
private $id;
```

Yllä oleva ote lähdekoodista tuottaa oliosta muodostetun tietokantataulun id-kenttään kokonaislukutyyppisen, automaattisesti luodun arvon nollasta alkaen. (Basic Mapping, Identifiers / Primary Keys n.d.)

Eri tietokantataulujen välisten yhteyksien luonti on tästä hieman hankalampi toimenpide: tuolloin olioon tulee lisätä yhteyttä kuvaava muuttuja, esimerkiksi $user, joka kuvastaa olioon sidottua käyttäjää. Myös user-olioon tulee lisätä vastamuuttuja yhteyden muodostavaan olioon. Tämän jälkeen molempien olioiden muuttujien metatietoihin lisätään tietokantayhteyden tarkempi määrittely.

Jos halutaan muodostaa esimerkiksi yksi-moneen-yhteys kahden olion välillä, tapahtuu määrittely seuraavasti:

```
/**
 *
 * @ManyToOne(targetEntity=”User”, inversedBy=”visits”)
 * @JoinColumn(name=”user_id”, referencedColumnName=”id”)
 */
private $user;
```

Ja vastaavasti user-olion lähdekoodissa:

```
/**
 * @OneToMany(targetEntity=”Visit”, mappedBy=”user”, cascade=”remove”)
 *
 */
private $visits;
```

Kun nämä oliot kirjoitetaan Doctrine2:n avulla tietokantaan, muodostetaan yksi-moneen tietokantayhteys, jossa visit-olion tietokantataulussa on user_id-kenttä, jossa visit-olion omistavan käyttäjän id on merkitty. Kun Doctrinen avulla lähetetään tietokannalle kysely user-olioista, sisältää palautettu olio $visits-muuttujassa kaikki user-olioon sidotut visit-oliot. (Association mapping, One-To-Many, Bidirectional n.d.)

Doctrine 2 tarvitsee toimiakseen tietoja tietokannasta, kuten tietokannan nimen, mitä tietokanta-ajuria käytetään sekä tietokannan käyttäjän tunnuksen ja salasanan. Nämä tiedot voidaan säilöä esimerkiksi PHP-taulukossa seuraavassa muodossa:

```
<?php
return array(
   'database' => array(
         'driver'   => 'pdo_mysql',
         'user'     => 'user',
         'password' => 'pword',
         'dbname'   => 'pkrj',
));
```

Database-taulukon tietoja käytetään Doctrine 2:n entitymanager-luokan luonnissa. Esimerkiksi tässä työssä tiedot kirjataan ensin erilliseen Storage-luokkaan, joka vastaa globaalien muuttujien säilömisestä. Kun tietokantakonfigurointi on säilötty Storageen, luodaan Doctrinen entitymanager-luokka lukemalla ensin juuri säilötyt järjestelmän konfiguraatiot Storage-luokasta get-metodilla $appConfig-muuttujaan. Tämän jälkeen määritellään tietokantaolioiden polku $paths-muuttujaan. Tässä tapauksessa poluksi tuli: **array('src/Application/Model')**. Seuraavaksi haetaan tietokantakonfiguraatio $appConfig-muuttujasta $dbParams-muuttujaan seuraavasti: **$dbParams = $appConfig['database'];.** Kun nämä muuttujat on määritelty, voidaan luoda entitymanagerille konfiguraatio Doctrinen metodilla: **Setup::createAnnotationMetadataConfiguration($paths, true);.** Ensimmäinen parametri sisältää aiemmin määritellyn olioiden polkutiedon, toinen puolestaan kertoo Doctrinelle, onko käytössä kehitystila. Tässä työssä ohjelmointivirheiden korjaamisen helpottamiseksi kehitystila on otettu käyttöön konfigurointia luotaessa. Kun konfiguraatio on luotu, voidaan viimein luoda entitymanager-luokka. Luonti tapahtuu Doctrinen metodilla **EntityManager::create($dbParams, $config);,** jossa ensimmäinen parametri on aiemmin $appConfigista haettu tietokantakonfiguraatio ja toinen parametri juuri luotu Doctrinen konfiguraatio (Installation and Configuration, Obtaining an EntityManager n.d.). Kun entitymanager on luotu, säilötään se Storage-luokkaan metodilla **Storage::set(Storage::KEY_ENTITY_MANAGER, $entityManager);,** jonka jälkeen entitymanageria voidaan tarvittaessa pyytää Storage-luokalta, eikä sitä tarvitse luoda jokaisella käyttökerralla uudestaan.

Luotaessa tietokantaan tauluja ensimmäistä kertaa on suositeltavaa luoda erillinen tietokannan luontiskripti, jossa Doctrinelle esitellään tietokantaan kirjattavien olioiden metatiedot, joilla taulut luodaan. Tähän tarvitaan Doctrinen entitymanageria ja SchemaToolia. Ensin luodaan SchemaTool-olio seuraavasti: **$tool = new \Doctrine\ORM\Tools\SchemaTool($entityManager);**. Kun SchemaTool on luotu, luetaan tietokantaolioiden metatiedot PHP-taulukkoon entitymanagerin metodilla **getClassMetadata('Path\To\Entity')**, jossa parametrinä on polku haluttuun tietokantaolioon. Kun kaikkien olioiden metatiedot on luettu taulukkoon, voidaan tietokannan taulut luoda SchemaToolin metodilla **createSchema($classes);**. Jos halutaan ensin hävittää mahdolliset vanhat taulut, joissa tiedon esitysmuoto on vanhentunut,

suoritetaan ensin SchemaToolin metodi **dropSchema($classes);**. Tämän jälkeen tietokantaan on luotu jokaiselle $classes-taulukossa olleelle oliolle oma taulu, jonka kentät määräytyvät oliossa olleen metatiedon mukaan ja olioita voidaan lisätä tietokantaan entitymanagerin persist-metodilla. (Tools, Database Schema Generation n.d.)

3.5 Zend Framework 2

Zend Framework 2 MVC- eli Model View Controller-arkkitehtuuri on sovellusarkkitehtuurityyli, joka jakaa ohjelmistosovelluksen kolmeen osaan. Ensimmäinen osa on malli, jonka voi mieltää esimerkiksi PHP-olioiksi. Toinen on näkymä, joka määrittää, millainen sovelluksen käyttöliittymä on ja miten mallien tieto näytetään kyseisessä käyttöliittymässä. Kolmas on kontrolleri, joka vastaa mallien ja näkymien välisestä viestinnästä, esimerkiksi mallin tietojen päivittämisestä tai näkymän päivittämisestä vastaamaan mallin muuttunutta tietoa.

MVC-arkkitehtuuri irrottaa mm. mallin riippuvuuden näkymästä, jolloin voidaan helposti kehittää useita erilaisia näkymiä esittämään tietoa samasta mallista erilaisilla tavoilla. Tämä säästää kehitystyötä, sillä jokaiselle näkymälle ei tarvitse luoda samasta mallista useita eri versioita, vaan voidaan luoda yksi malli, jonka tiedosta esitetään osia eri näkymissä. Hyvä esimerkki voisi olla käyttäjä-olio, jonka perustiedot esitettäisiin peruskäyttäjälle näkyvässä näkymässä ja tarkemmat tiedot, kuten kirjautumishistoria yms. järjestelmän ylläpitäjälle. Lisäksi riippuvuuden purku mahdollistaa mallien itsenäisen testauksen riippumatta muiden sovelluksen komponenttien tilasta, jolloin esimerkiksi yksikkötestien kirjoittaminen malleille on helpompaa.

MVC-arkkitehtuurin eri osia käytetään Zend Framework 2:ssa niin kutsutun routing-menetelmän avulla. Kun sivustolle luodaan uusi näkymä, se tulee reitittää oikein, jotta Zend voi ohjata palvelimen näyttämään oikean näkymän käyttäjälle. Reititys tapahtuu Zend-modulin asetustiedostosta, joka sisältää eri näkymien ja kontrollerien osoitteet, niihin liittyvät rajoitukset sekä vakioarvot. Tavanomainen reititys voi olla vaikkapa About-näkymään, jolloin asetustiedostossa olevaan listaan lisätään uusi arvo ”about”. Arvolle määritellään reitityksen tyyppi ja lisäasetukset, kuten reitti kokonaisuudessaan, tässä tapauksessa muotoa ”/about[/:action][/:id]”. Lisäasetuksiin lisätään myös rajoitukset kontrollerin ja toiminnan muodostamiselle. Tämä määritellään yleisellä regular expressionilla eli säännöllisellä lausekkeella. Rajoitusten jälkeen

määritellään vakioarvot kontrollerille ja toiminnalle, tässä esimerkissä vakiokontrolleriksi valitaan ”Application\Controller\About” ja vakiotoiminnaksi valitaan ”about”. Lisäksi uusi kontrolleri täytyy lisätä modulin asetustiedoston invokables-taulukkoon, jotta Zend Framework voi luoda kontrollerit ilman konstruktoriparametrejä. Tämän lisäksi uudet phtml-sivupohjat kannattaa lisätä template_map-taulukkoon, jotta niiden käyttö esimerkiksi uudelleenohjauksessa tai ViewModel-olion sivupohjan asettamisessa onnistuu helpommin. Näiden tietojen lisäämisen ja asetustiedoston tallentamisen jälkeen uusi näkymä on valmis käytettäväksi erinäisillä kontrolleri-toimintayhdistelmillä. (Routing and controllers n.d.).

Täytyy kuitenkin muistaa, että mikäli halutaan käyttää jotain muuta kontrolleria tai toimintaa kuin vakioarvoissa määriteltyjä, tulee niiden olla olemassa ja ohjelmoituina. Toiminnat ohjelmoidaan kontrollerien sisälle. Jos aiemmassa esimerkissä mainittuun AboutControlleriin halutaan lisätä adminAbout-toiminta, ohjelmoidaan AboutControllerin sisälle aboutActionin alle adminAboutAction, jossa määritellään, mitä kyseisen toiminnan tulee tehdä kutsuttaessa. Toimintojen määrää ei ole rajoitettu kontrolleria kohden, joten uusia toimintoja voi aina lisätä tarpeen vaatiessa.

3.6 Oracle VirtualBox

VirtualBox on Oraclen ylläpitämä virtualisointisovellus, joka mahdollistaa virtuaalisen ”vieraskoneen” käynnistämisen ja suorittamisen pääasiallisen eli isäntäjärjestelmän päällä. Vieraskoneelle voidaan määrittää muun muassa virtuaalinen kiintolevy ja

-välimuisti, jotka varataan isäntäkoneen resursseista. Tälle virtuaalikoneelle voidaan asentaa isäntäkoneesta poikkeava käyttöjärjestelmä ilman, että isäntäkoneen kiintolevyä tulisi alustaa. Virtuaalikone käyttäytyy lähes täysin normaalisti muutamaa poikkeusta lukuunottamatta, ja sitä voidaankin käyttää tehokkaasti esimerkiksi erilaisten käyttöjärjestelmien testaukseen tai vaihtoehtoisesti sovellusten kehittämiseen.

3.7 Twitter Bootstrap

Twitter Bootstrap on käyttöliittymäkehys, jonka avulla voidaan kehittää yhtaikaisesti perinteisille työpöytä- ja mobiiliympäristöille sopivia sovelluksia. Twitter Bootstrap on JavaScript-kirjasto, joka muokkaa sivun ulkoasua dynaamisesti selainikkunan koon mukaan. Ulkoasun elementtien koon määrittely perustuu kuvitteelliseen ruuduk-

koon, jonka leveys on 12 yksikköä. Sovelluksen kehittäjä voi jakaa HTML-elementeilleen leveysarvoja jakamalla nuo 12 yksikköä haluamiinsa osiin. Twitter Bootstrap sisältää useita erilaisia ruudukkomääritelmiä erilaisille laitteille, joten sovelluskehittäjä voi myös määritellä elementille eri leveyden eri laitteille. (Bootstrap grid examples n.d.)

4. Teknologiavalintojen perusteluja

4.1 LAMP

LAMP (Linux, Apache, MySQL ja PHP) valikoitui käyttöön useista syistä: alusta on jo ennalta tuttu, joten kehitystyön aloittaminen on nopeaa, sillä tärkeimmät asetukset löytyvät totutuista paikoista. Asetuksia on myös helppo muokata ja varsinkin Apache-palvelimen yleisimmät käyttötapaukset on dokumentoitu melko hyvin.

LAMP on myös yhteensopiva useiden eri ORM-kirjastojen kanssa. Verratessa esimerkiksi MEAN (MongoDB Express AngularJs Node.js) alustaan ORM-kirjastoja on huomattavasti enemmän. Sovelluksen suunnitteluvaiheessa päätettiin käyttää ORM-kirjastoa, joten valittiin alusta, jolle kyseisiä kirjastoja on kehitetty useita. Näin valinnanvaraa oli enemmän.

Nopea ja vaivaton asentaminen oli myös syy LAMP-alustan valintaan: Ubuntu Linuxissa LAMPin asennus tapahtuu yksinkertaisella komennossa ”sudo tasksel”, joka avaa listan erinäisiä sovelluksia, joista käyttäjä voi valita tarvitsemansa ohjelmat. Tämän jälkeen tasksel asentaa sovellukset järjestelmään ilman sen suurempaa puuttumista käyttäjältä. Mikäli valitaan Ubuntun palvelinversio, käynnistyy tasksel käyttöjärjestelmän asennuksen yhteydessä, mikä tekee LAMPin asentamisesta vielä hieman vaivattomampaa.

Lisäksi laaja dokumentaatio on eräs tärkeä etu LAMP-alustan valinnan kannalta: PHP ja MySQL ovat olleet jo pitkään kehityksessä ja niihin on kehitetty laaja, hyvä ja helppolukuinen dokumentaatio. Hyvä dokumentaatio on tärkeää ohjelmoinnin vaivattomuuden ja laadun takaamiseksi, minkä vuoksi dokumentaation laatu on iso kriteeri alustan valinnassa.

Osaltaan alustan valintaan vaikutti myös Linuxin kuuluisa vakaus, joskaan kriteerinä se ei ollut kovin suuri: Varsinainen Apache, MySQL ja PHP osio oli jo valikoitunut

tässä vaiheessa, mutta vaihtoehtona olisi ollut myös WAMP (Windows Apache MySQL PHP). Vertailtaessa Linuxia ja WAMPia, Linux-versio valikoitui juuri edellä mainitun vakauden lisäksi myös ilmaisuuden ja sen vuoksi, että käyttöjärjestelmä on myös ylläpitomielessä tuttu.

4.2 Doctrine 2

Doctrine 2:n valintaan vaikutti pääosin alustan valinta. LAMP-arkkitehtuurissa PHP tarjoaa hyvän valikoiman erilaisia ORM-kirjastoja (Object-relational mapping), joista Doctrine 2 korostui kattavan dokumentaationsa ja helpohkon asentamisprosessinsa vuoksi edukseen. Doctrine 2 on myös yksi suurimpia ORM-kirjastoja PHP:lle, mikä tarkoittaa, että ylläpito on yleensä toteutettu hyvin.

Itse ORM-teknologian valinta taas johtui siitä, että ORM-kirjasto helpottaa tietokannan ylläpitoa. Kehittäjän ei tarvitse kirjoittaa itse tietokantakyselyjä, vaan kirjasto on vastuussa kyselyiden muodostamisesta. Kehittäjän vastuulla on vain ohjelmoida olioihin tieto siitä, miten olion eri jäsenmuuttujat tulee kirjoittaa tietokantaan. Tällä tavoin kehittäjä voi kirjoittaa olioita suoraan tietokantaan tai vastaanottaa niitä suoraan tietokannasta. Tämä vähentää käyttäjän tekemiä inhimillisiä virheitä kyselyjen kirjoittamisessa. Lisäksi tietokannan turvallisuus kasvaa, sillä esimerkiksi Doctrine2 sisältää MySQL-kyselyjen esitarkistuksen ennen niiden välittämistä tietokannalle, jolloin tietokantainjektion riski vähenee. Lisäksi kehittäjän työmäärä sovelluksen tietoturvan varmistamisesta helpottuu, koska kehittäjän ei tarvitse itse tehdä esitarkistuksia.

4.3 Zend Framework 2

Zend Framework 2 valikoitui suurelta osin Doctrine 2 -yhteensopivuuden vuoksi. Varsinaisesti Doctrine 2 toimii muidenkin kehysmallien kanssa, mutta käyttö Zend Frameworkin kanssa on hyvin dokumentoitua, mikä helpottaa sovelluksen kehittämistä hyvin paljon.

Zend Framework 2 sisältää myös Twitter Bootstrapin valmiina, jolloin sovelluksen kehittäminen mobiiliystävälliseksi voidaan aloittaa heti. Samalla vältytään mahdollisesti aikaa vievältä ja työläältä asennukselta.

Zend on lisäksi helposti asennettavissa omiin projekteihin. Kehittäjä voi ladata valmiin pohjan, jota aletaan muokkaamaan tai vaihtoehtoisesti asentaa tarvittavat osat esimerkiksi Composerilla.

Yleisesti MVC-rakenne valikoitui, koska kyseinen malli mahdollistaa nopean kehittämisen pitäen samalla projektin rakenteen selkeänä. Selvä jako erilaisiin komponentteihin auttaa hahmottamaan sovelluksen eri osia selkeämmin sekä vaikuttaa osaltaan koodin laatuun positiivisesti pienentämällä eri osien rivimäärää.

4.4 Twitter bootstrap

Twitter Bootstrapin valintaan vaikutti asiakkaan vaatimus mobiiliystävällisyydestä. Tähän tarkoitukseen Twitter Bootstap on erinomainen, sillä se on hyvin dokumentoitu ja sen käyttöönotto on helppoa. Esimerkiksi Zend Frameworkin mukana se tulee esiasennettuna eikä ylimääräistä työtä käyttöönoton osalta tarvitse tehdä, jolloin sovelluksen kehittäjä voi suoraan keskittyä tuottavaan työhön. Lisäksi kehittäjällä on käytettävissä suuri määrä valmiita ikoneita omilla sivustoillaan. Näitä on helppo lisätä haluamiinsa elementteihin, esimerkiksi verkkosivun painikkeisiin.

4.5 Composer

Composer valittiin projektiin paketinhallintasovellukseksi helpottamaan sovelluksen eri komponenttien päivittämistä ja asentamista. Sen sijaan, että jokainen kolmannen osapuolen kirjasto tulisi ladata ylläpitäjän kotisivuilta ja korvata sillä vanha kirjasto, on helpompaa antaa siihen tarkoitukseen kirjoitetun sovelluksen hoitaa raskas työ ja keskittyä varsinaiseen kehitystyöhön. Lisäksi Composer tarjoaa kehittäjälle autoloader-kirjaston, jonka avulla kehittäjä voi ladata kaikki Composerin kautta ladatut kolmannen osapuolen kirjastot lisäämällä kyseinen autoloader mukaan sovelluksen latausskriptiin. Esimerkiksi Zend Framework 2:ssa tämä autoloader-kirjasto lisätään osaksi init_autoloader.php-tiedoston suoritusta, jolloin kaikki kirjastot ladataan sovellukseen aina, kun käyttäjä lataa uuden sivun.

4.6 Oracle VirtualBox

Oraclen VirtualBox-virtualisointiympäristö valittiin, koska tarvittiin nopeasti ja helposti käyttöönotettava kehitysympäristö. Koska Serecom ei voinut tarjota kehityspal-

velinta, tultiin siihen tulokseen, että varsinainen kehitystyö toteutetaan virtualisoidussa ympäristössä, josta lähdekoodi siirretään palvelimelle työn valmistuttua. Muista virtualisointiympäristöistä VirtualBox valikoitui hintansa, helppokäyttöisyytensä ja kehittäjän aiempien käyttökokemusten perusteella. Kehittäminen on paljon helpompaa aloittaa työkalulla, jota on helppo käyttää ja sen käytöstä on jo aiempaa kokemusta. Lisäksi kehityksen varhaisessa vaiheessa ei ole aiheellista vuokrata tai ostaa palvelinta. On kustannustehokkaampaa kehittää sovellus käyttökelpoiseen kuntoon virtuaalisessa ympäristössä.

5. Toteutus

5.1 Työn aloitus

Opinnäytetyö aloitettiin pitämällä asiakkaan kanssa palaveri, jossa Serecom esitteli sovellus- ja liikeideansa, kohdeyleisön sekä toiveita ja vaatimuksia sovellukseen liittyen. Näistä toiveista ja vaatimuksista muodostettiin lista ominaisuuksista, joita lähdettiin sovellukseen kehittämään.

Palaverin jälkeen valittiin teknologiat, joiden päälle sovellusta lähdettiin kehittämään. Ensimmäisenä valittiin palvelinalusta ja ohjelmointikielet, joilla sovellus kehitettäisiin. Tämän jälkeen valikoitiin muita työkaluja asiakkaan vaatimuksia täyttämään. Mobiiliystävällisen käyttöliittymän kehittämiseen valittiin käyttöliittymäkehys, joka mahdollistaa helposti mobiililaitteille sopivan käyttöliittymäkehityksen. Samalla valittiin myös sovelluksen rakenteeksi MVC-arkkitehtuuri. Varsinainen toteutus tapahtui virtuaalikoneen sisällä, tarkoituksena kuitenkin siirtää lähdekoodi työn valmistuttua Serecomin hankkimalle palvelimelle.

Työssä käytettävien kolmannen osapuolen kirjastojen ylläpitämisen helpottamiseksi otettiin käyttöön Composer-työkalu. Kyseinen työkalu sisältää json-tiedoston, johon voi lisätä haluamiaan kirjastoja tiedostosta löytyvään listaan. Tämän jälkeen komennolla php composer.phar install voidaan asentaa listalla olevat kirjastot ja niiden riippuvuudet kehitysympäristöön ilman kehittäjän suurempia toimenpiteitä. Ensimmäisen asennuksen jälkeen kirjastot voidaan tarvittaessa päivittää komennolla php composer.phar update, jolloin Composer etsii listalla oleviin kirjastoihin päivityksiä ja tarvittaessa asentaa ne.

PHP-olioiden ohjelmoinnissa käytettiin pääosin PSR-2 -ohjelmointikäytäntöä. Olion muuttujat pidettiin pääosin yksityisinä, ellei ollut aihetta määritellä muuttujaa julkiseksi. Tavoitteena on säilyttää mahdollisimman suuri osa vastuusta omiin muuttujiinsa oliolla itsellään. Tämä lähestymistapa helpottaa myös sovelluksen jäsentämistä, kun tiedetään, mistä asioista kukin komponentti vastaa.

Ensin pystytettiin virtuaalikone, jonka sisällä kaikki kehitystyö tapahtui. VirtualBoxin New-painikkeella saatiin virtuaalikoneen opastettu luomistoiminto käyttöön. Vaihtoehdoista valittiin 64-bittinen Ubuntu 14.04-käyttöjärjestelmä 20 gigatavun kokoisella virtuaalikiintolevyllä ja 1024 megatavun välimuistilla varustettuna. Tämän jälkeen asennettiin yllä mainittu käyttöjärjestelmä luodulle virtuaalikoneelle.

Seuraavaksi asennettiin LAMP-alusta virtuaalikoneelle. Ensin piti asentaa tasksel-niminen sovellus komennolla **sudo apt-get install tasksel**, jonka jälkeen asennettu sovellus suoritettiin komennolla **sudo tasksel**. Tasksel tarjosi listan sovelluksia, jotka sen kautta voitiin asentaa. Listalta valittiin LAMP server painamalla välilyöntiä. Valinnan jälkeen enter-painalluksella tasksel aloitti LAMP:n asennuksen. Asennuksen aikana piti syöttää MySQL:n pääkäyttäjän salasana, jonka jälkeen asennus suoriutui loppuun ilman erillistä toimenpiteitä.

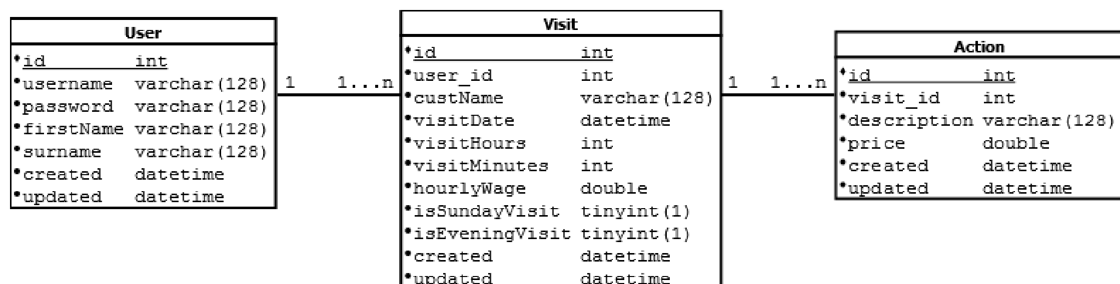
Zend Framework 2:n pohja pystytettiin Composerin avulla ajamalla virtuaalikoneessa komento **php composer.phar create-project –stability=”dev” zendframework/skeleton-application /home/hujis/Oppari/potilaskirjaus**. Composer latasi Zend Framework 2:n luurankosovelluksen, joka sisältää kaiken tarvittavan Zendillä työskentelyyn.

Kun Zend Framework 2 oli pystytetty, ladattiin Doctrine 2:n kirjastot Composeria käyttäen. Ensin lisättiin composer.json-tiedoston require-taulukkoon uusi tieto **"doctrine/orm":** "*", jonka jälkeen suoritettiin komnentorivillä komento **php composer.phar install**, joka aloittaa Composerin asennunprosessin. Tämän jälkeen ei tarvittu jatkotoimenpiteitä, koska Zend Framework 2:n bootstrap-skriptissä oli jo vakiona ladattu mukaan Composerin autoload.php latausskripti, jolla Doctrine saadaan ladattua osaksi sovelluskokonaisuutta.

Seuraavaksi määriteltiin Apache 2 -verkkopalvelimelle virtuaalinen isäntä eli virtual host. Aluksi luotiin konfiguraatiotiedosto potilaskirjaus.com.conf, joka sisälsi LAMPia

käsittelevässä luvussa esitellyn virtual host -määrittelyn. Tämän jälkeen otettiin käyttöön rewrite-moduli, jota Zend Framework 2 tarvitsee komennolla **sudo a2enmod rewrite**, ja ladattiin asetukset uudelleen komennolla **sudo service apache2 reload**. Seuraavaksi otettiin potilaskirjaussivusto käyttöön navigoimalla palvelimen hakemistoon /etc/apache2/sites-available ja suorittamalla komento **sudo a2ensite potilaskirjaus.com.conf**. Seuraavassa vaiheessa käynnistettiin Apache 2 uudelleen komennolla **sudo service apache2 restart**. Lopuksi lisättiin Ubuntun hosts-tiedostoon reititysohje sisäiseen IP-osoitteeseen ja testattiin sivuston toiminta selaamalla potilaskirjaus.com -osoitteeseen.

Tämän jälkeen keskityttiin tietokannan suunnitteluun. Tässä työssä tietokanta koostuu kolmesta taulusta. User-taulu sisältää käyttäjän tiedot, kuten id:n, käyttäjätunnuksen, etunimen, sukunimen, salasanan, taulun luontipäivämäärän ja taulun muokkauspäivämäärän. Visit-taulu sisältää käynnin id:n, asiakkaan nimen, käynnin päivämäärän, käynnin keston tunneissa, käynnin keston minuuteissa, tuntitaksan, tiedon, onko käynti tapahtunut sunnuntaina, ilta- tai yöaikaan, taulun luontipäivämäärän, taulun muokkauspäivämäärän sekä moni-yhteen -yhteyttä kuvastavan user_id-kentän. User_id -kenttä sisältää käynnin omistavan käyttäjän id:n. Action-taulu sisältää id:n, toimeenpiteen kuvauksen, toimenpiteen hinnan, taulun luontipäivämäärän, taulun muokkauspäivämäärän, sekä moni-yhteen -yhteyttä kuvaavan visit_id-kentän, joka sisältää toimenpiteen omistavan käynnin id:n. (Ks. kuvio 2)



Kuvio 2. Tietokannan käsitemalli

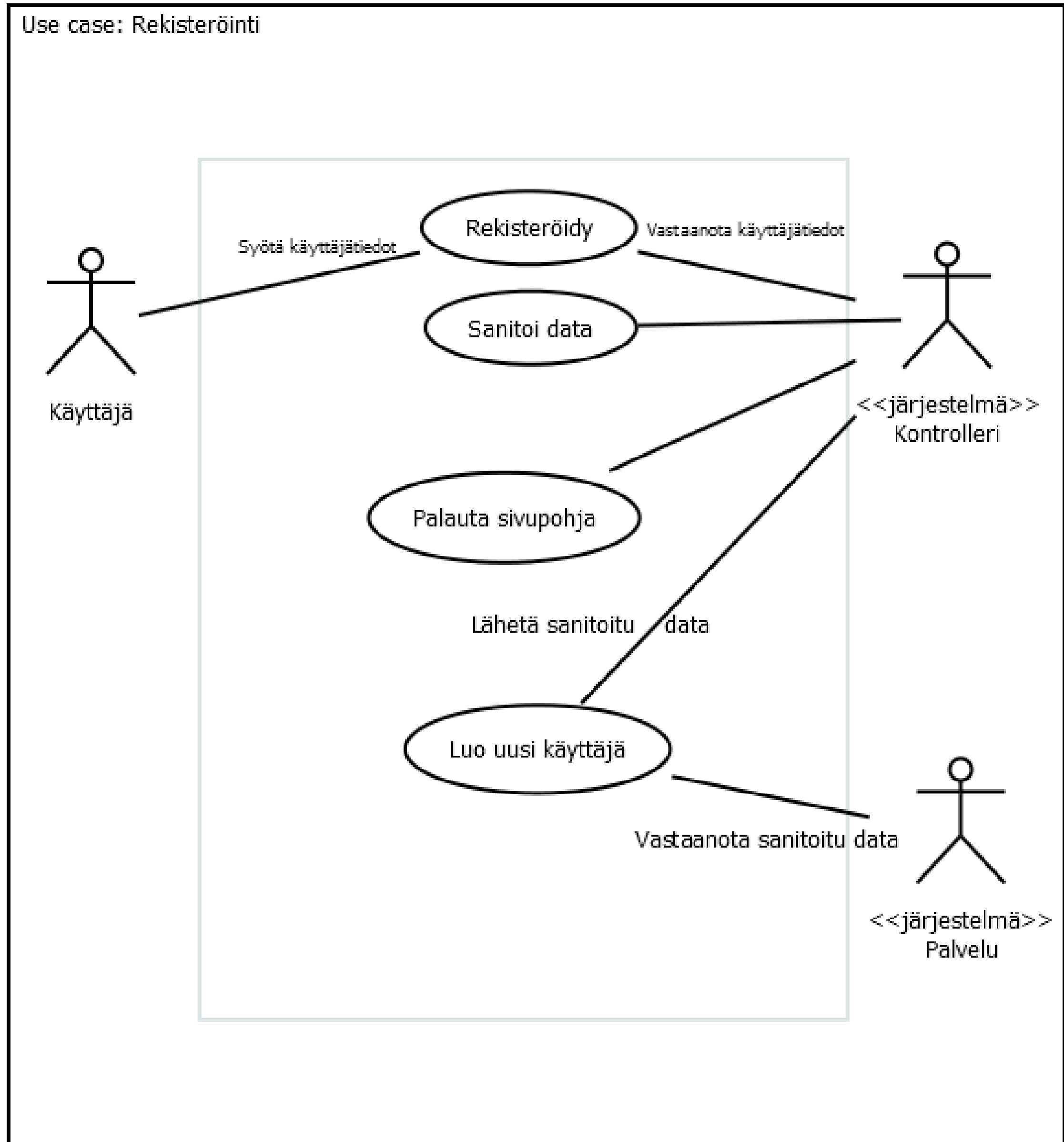
Tietokannan käsitemalli havainnollistaa tietokannan rakennetta. Jokaisella taululla id-kenttä on automaattisesti generoitu integer-tyyppinen numero, joka toimii taulun pääavaimena sekä vaihteleva määrä muita kenttiä. Yksikään kenttä ei voi olla tyhjä yhdessäkään taulussa, vaan kaikkiin on syötettävä jonkinlainen arvo.

Tietokannan rakenne muistuttaa hieman pyramidia: ylimpänä on User-taulu, joka voi omistaa monta käyntiä, joka taas voi omistaa monta toimenpidettä. Moni-moneen -yhteyksiltä vältyttiin, koska käynnit ja toimenpiteet ovat uniikkeja tapahtumia, joita ei voida toistaa esimerkiksi toiselle käyttäjälle tai käynnille.

5.2 Rekisteröityminen

Ensimmäisenä toteutettiin sivulle rekisteröityminen. Kaikki muut sovelluksen toiminnot nojaavat käyttäjän kykyyn rekisteröityä, joten sen toteuttaminen ensimmäisenä oli itsestäänselvää. Rekisteröinnille luotiin oma sivupohja register.phtml, joka sisälsi rekisteröintilomakkeen, sekä kontrolleri RegisterController, joka vastasi käyttäjän lomaketietojen käsittelystä.

Käyttötapauskaaviossa (Ks. kuvio 3) esitellään rekisteröinnin käyttötapausta. Rekisteröinnissä käyttäjä syöttää käyttäjätiedot rekisteröintilomakkeeseen ja lähettää ne kontrollerille. Kontrolleri vastaanottaa lomakedatan ja sanitoi eli poistaa ei-halutun osan kaikista lomakkeessa olevista tiedoista. Yleisin sanitoinnin kohde ovat mm. <script>-tunnisteet, joiden avulla käyttäjä kykenisi hyväksikäyttämään XSS, eli cross-site-scripting-hyökkäystä sivustolla. Kun lomakedata on sanitoitu, lähettää kontrolleri sanitoidun datan listana palveluluokalle. Tämä palveluluokka vastaanottaa lähetetyn listan ja luo sen avulla uuden käyttäjän, joka kirjataan tietokantaan.



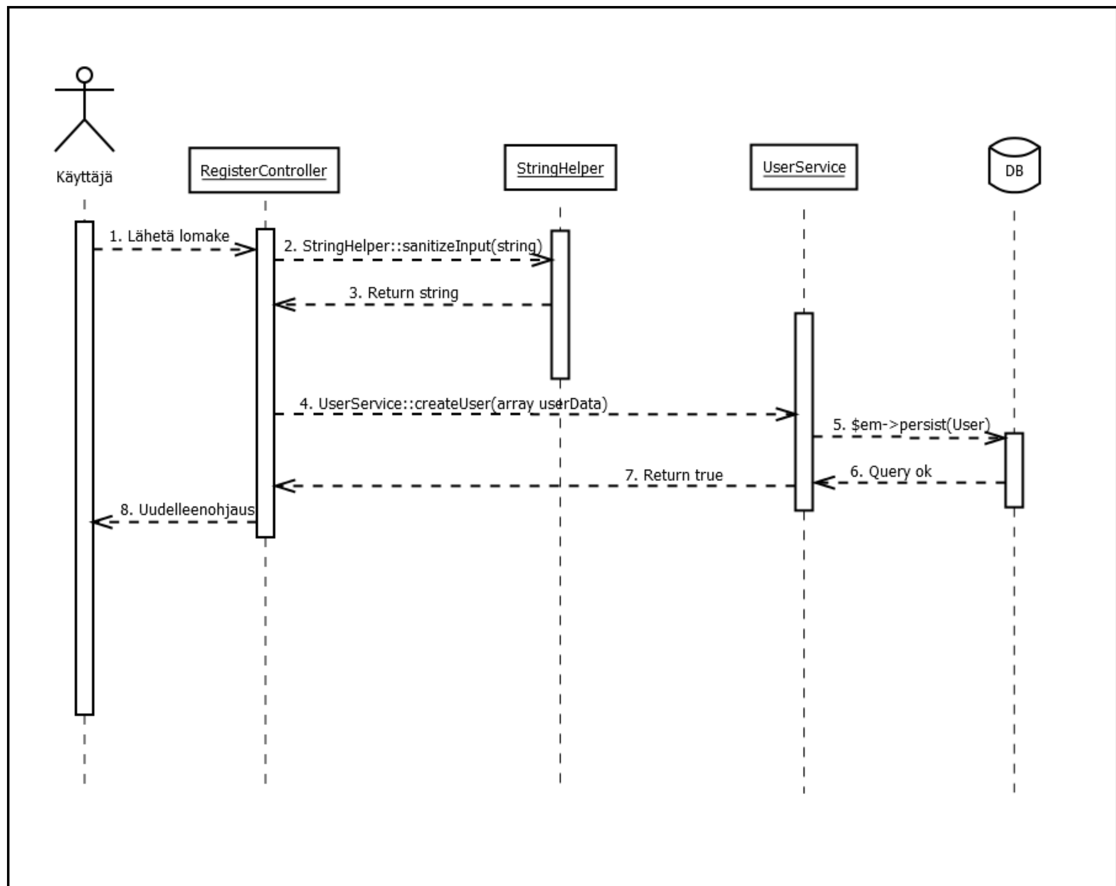
Kuvio 3. Käyttötapauskaavio uuden käyttäjän rekisteröitymisestä

Sekvenssikaavio (Ks. kuvio 4) kuvaa rekisteröintiprosessia yksityiskohtaisemmin. Alussa käyttäjä lähettää lomakedatan POST-menetelmällä RegisterControllerille, jonka registerAction-metodi aktivoituu. RegisterActionissa lomakkeen tiedot ohjataan StringHelperille kutsumalla metodia StringHelper::sanitizeInput($_POST['kenttä']). StringHelper palauttaa sanitoidun version annetusta merkkijonosta, jonka RegisterController lisää $userData-listaan. Kun kaikki lomakkeen tiedot on sanitoitu ja lisätty $userDataan, lähetetään lista UserService-luokalle, jonka staattisella metodilla UserService::createUser($userData) luodaan uusi käyttäjä. Ensiksi metodi luo uuden User-luokan, jonka tiedot täytetään Set-metodeilla. Kun tiedot on

täytetty, kutsutaan Doctrine 2:n entitymanager-luokan persist-metodia, joka kirjaa juuri luodun User-luokan tietokantaan. Onnistuessaan UserService palauttaa boolean-arvon true. Kun RegisterController vastaanottaa true-arvon, se ohjaa käyttäjän etusivulle onnistumisviestin saattelemana. Mikäli uuden käyttäjän luonti syystä tai toisesta epäonnistuu, ohjaa RegisterController käyttäjän takaisin rekisteröintisivulle, jossa käyttäjälle näytetään virheviesti merkiksi epäonnistuneesta rekisteröitymisestä.

Rekisteröintisivulla käyttäjälle tarjotaan lomake, jossa kysytään yleisiä käyttäjätietoja. Käyttäjän tulee syöttää sähköpostiosoite, joka toimii myös käyttäjätunnuksena, etu- ja sukunimi sekä salasana kahdesti. Salasana tulee syöttää kahteen kertaan, jotta vältytään kirjoitusvirheiltä.

Alustavasti lomakkeen lähetyspainike on poissa käytöstä. Tällä tavalla estetään käyttäjää painamasta painiketta vahingossa lomakkeen täytön ollessa kesken. Lisäksi käyttäjän enter-painallukset on lomakkeessa estetty samasta syystä. Kun käyttäjä on syöttänyt oikeantyyppisen sähköpostin ja molemmat salasanakentät täsmäävät, aktivoidaan lähetyspainike, jolloin käyttäjä voi viimeistellä rekisteröinnin.

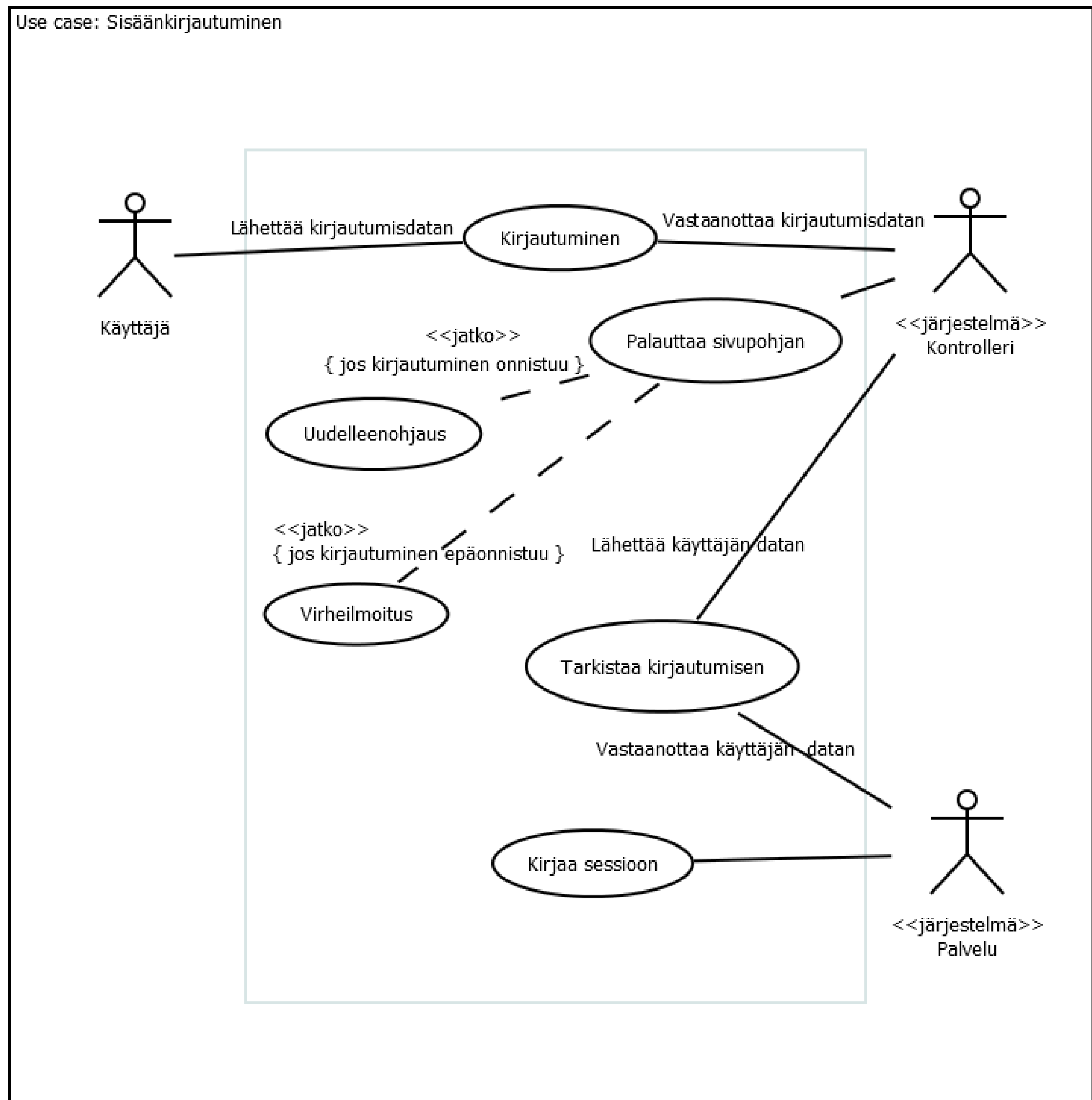


Kuvio 4. Sekvenssikaavio uuden käyttäjän rekisteröitymisestä

5.3 Sisäänkirjautuminen

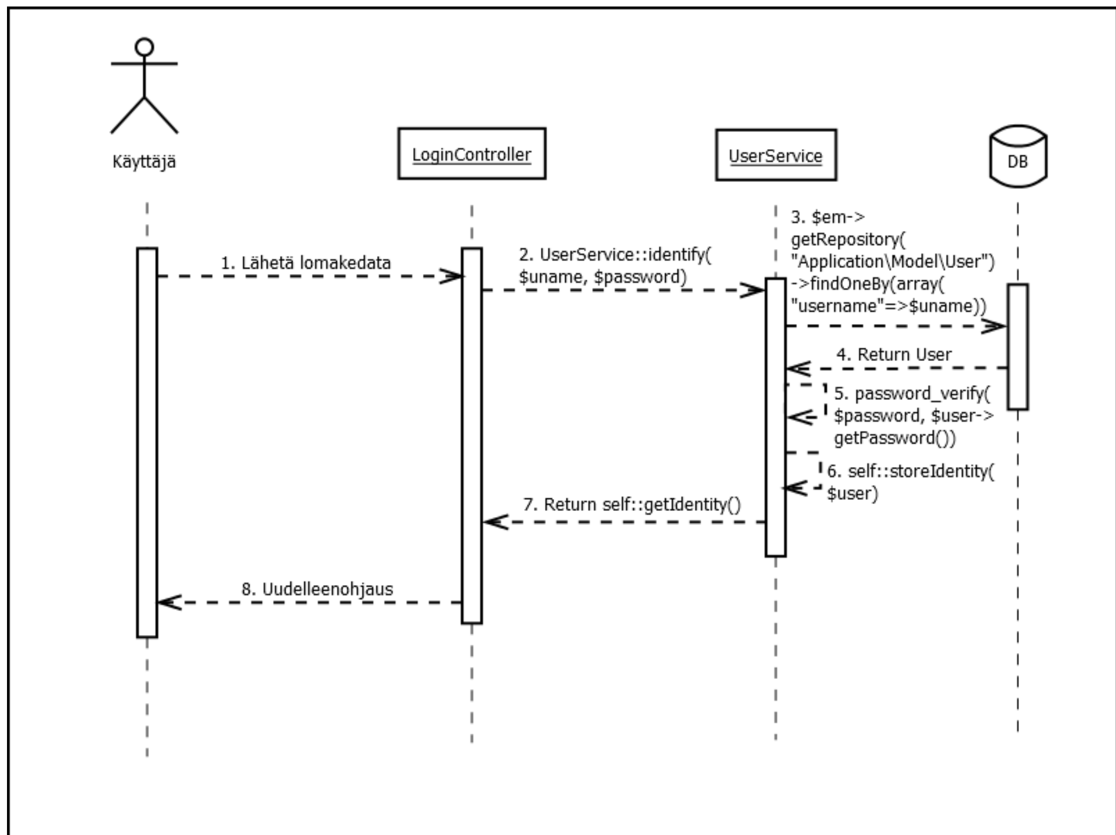
Seuraava looginen toteutuskohta oli sisäänkirjautuminen pääosin samoista syistä kuin rekisteröityminenkin. Lisäksi toimivalla kirjautumistoiminnolla voitiin varmentaa rekisteröinnin toimivuus ja tietojen oikeellisuus tietokannassa. Sisäänkirjautumiselle luotiin oma sivupohja login.phtml ja LoginController, joka vastasi kirjautumistietojen käsittelystä. Lisäksi UserService-luokkaa laajennettiin lisäämällä identify-, storeIdentity-, getIdentity- ja hasIdentity-metodit

Käyttötapauskaaviossa (Ks. kuvio 5) käyttäjä lähettää kirjautumislomakkeen tiedot kontrollerille, joka puolestaan ohjaa ne palveluluokalle. Palveluluokka vastaanottaa tiedot ja tarkistaa käyttäjän kirjautumisen tietokannan tietojen pohjalta. Palveluluokka myös kirjaa käyttäjän tiedot sessioon eli istuntoon. Kontrolleri palauttaa käyttäjälle sivupohjan riippuen kirjautumisen onnistumisesta. Mikäli kirjautuminen onnistuu, ohjataan käyttäjä etusivulle. Jos kirjautuminen epäonnistuu, annetaan käyttäjälle kirjautumissivulla virheilmoitus.



Kuvio 5. Käyttötapauskaavio sisäänkirjautumisesta

Sekvenssikaaviossa (Ks. kuvio 6) käyttäjä lähettää lomaketiedot LoginControllerille, joka kirjaa tiedot muuttujiin $uname ja $password. LoginController kutsuu tämän jälkeen UserServicen identify-metodia. Parametreinä lähetetään muuttujat $uname ja $password. UserService hakee osana identify-metodia tietokannasta käyttäjän, jolla on sama käyttäjänimi, käyttämällä apuna Doctrine 2:n entitymanager-luokkaa. Mikäli käyttäjä löytyy tietokannasta, palautetaan se UserServicelle. Tämän jälkeen UserService kutsuu PHP:n password_verify-metodia, joka tarkistaa, että syötetty salasana on oikein. Mikäli salasana on oikein, kutsuu UserService omaa storeIdentity-metodia, joka kirjaa käyttäjän tiedot sessioon. Kun käyttäjä on kirjattu sessioon, palautetaan LoginControllerille juuri kirjatun käyttäjän tiedot. Tämän jälkeen käyttäjä ohjataan etusivulle.



Kuvio 6. Sekvenssikaavio sisäänkirjautumisesta

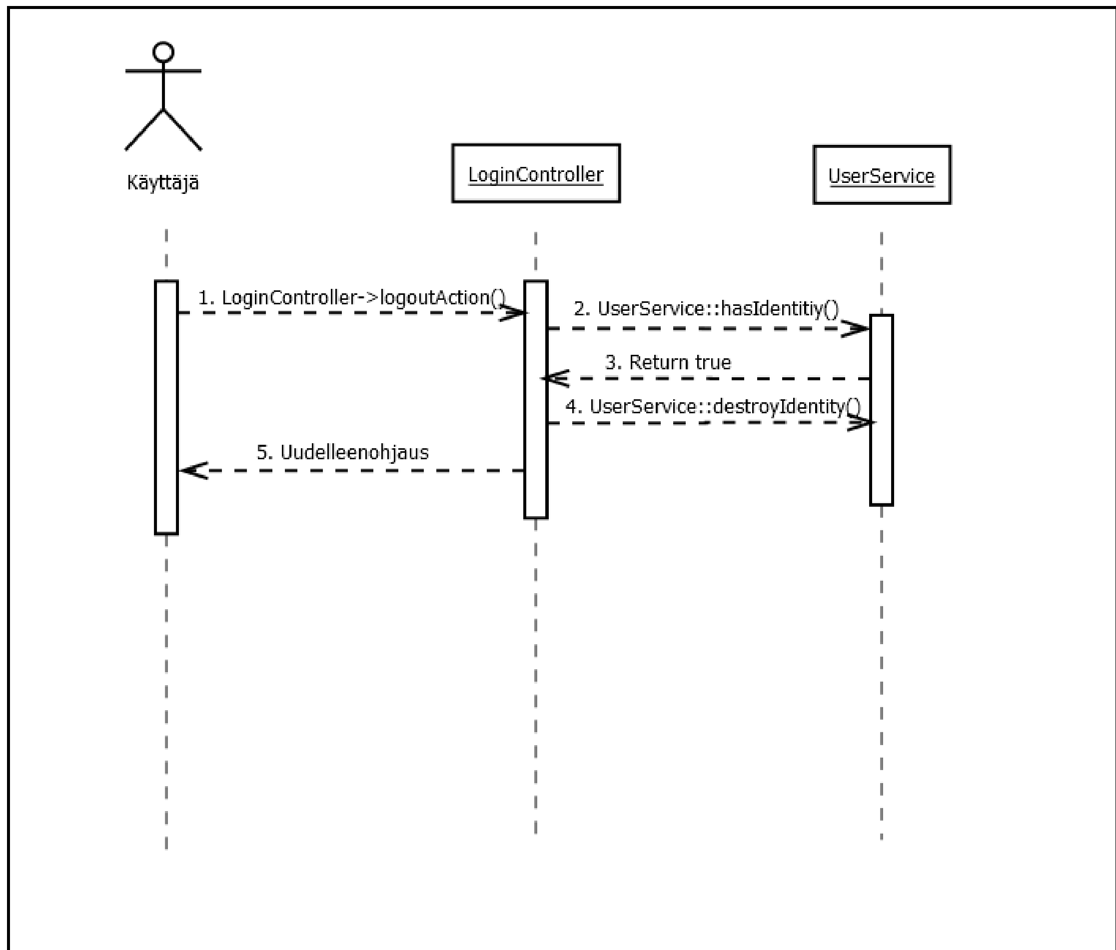
5.4 Uloskirjautuminen

Osana kirjautumiskokonaisuutta toteutettiin myös uloskirjautuminen, jotta eri käyttäjien kirjautumista ja kirjautumista vaativien sivujen tietoturvaa voitiin testata. Uloskirjautumiseen ei tarvittu uutta sivupohjaa, ainoastaan LoginControlleriin ja UserServiceen lisättiin uusia metodeja. LoginControlleriin lisättiin logoutAction-metodi ja UserServiceen destroyIdentity-metodi.

Käyttötapauskaaviossa (Ks. kuvio 7) käyttäjä kirjauttuu ulos. Kontrolleri lähettää uloskirjautumispyynnön palveluluokalle, joka vastaanottaa kontrollerin lähettämän pyynnön. Ennen session tuhoamista palveluluokka tarkistaa, että käyttäjä on kirjautunut sisään. Sessio tuhotaan, mikäli käyttäjä on kirjautunut sisään.

Kuvio 7. Käyttötapauskaavio käyttäjän uloskirjautumisesta

Sekvenssikaaviossa (Ks. kuvio 8) käyttäjä kutsuu LoginControllerin logoutAction-metodia. LoginController tarkistaa sisäänkirjautumisen kutsumalla UserServicen hasIdentity-metodia. Mikäli UserService palauttaa boolean-arvo truen, kutsutaan UserServicen destroyIdentity-metodia, joka tuhoaa voimassaolevan session, jolloin kirjautuminen ei enää ole voimassa. Tämän jälkeen käyttääjä ohjataan etusivulle.

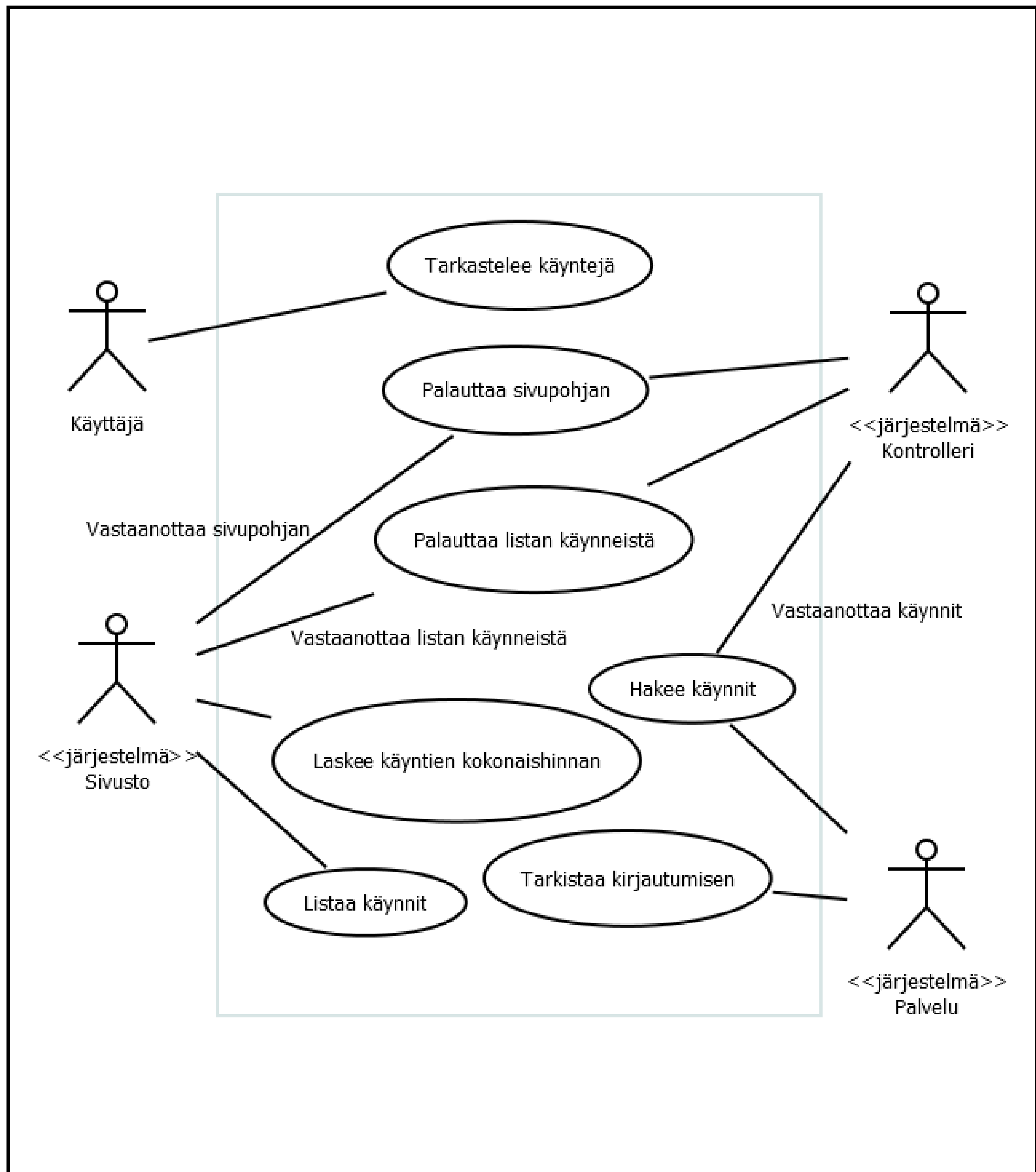


Kuvio 8. Sekvenssikaavio käyttäjän uloskirjautumisesta

5.5 Käyntien tarkastelu

Seuraavana toteutettiin käyntien yleissivu, jossa käyttäjä näkee kaikki kirjaamansa käynnit. Yleissivulla käynneistä näytetään taulukko, johon on listattu käynnin asiakas, käynnin päivämäärä, käynnin kesto tunneissa ja minuuteissa sekä kokonaishinta erilliset toimenpiteen mukaanlukien. Käyntien yleissivulle luotiin oma sivupohja visit.phtml sekä VisitController-luokka vastaamaan käynnin tietojen hallinnasta.

Käyttötapauskaaviossa (Ks. kuvio 9) käyttäjä haluaa tarkastella käyntejä. Kontrolleri kutsuu palveluluokkaa tarkistaen ensin, että käyttäjä on kirjautunut sisään. Tämän jälkeen luokka hakee listan käyttäjän käynneistä ja palauttaa sen kontrollerille. Kontrolleri vastaanottaa listan ja palauttaa sen sivustolle sivupohjan ohessa. Sivusto vastaanottaa sivupohjan sekä käyntilistan ja listaa käynnit HTML-taulukkoon. Jokaiselle taulukon käynnille sivusto laskee kokonaishinnan, joka listataan kyseisen käynnin taulukkokirjaukseen.

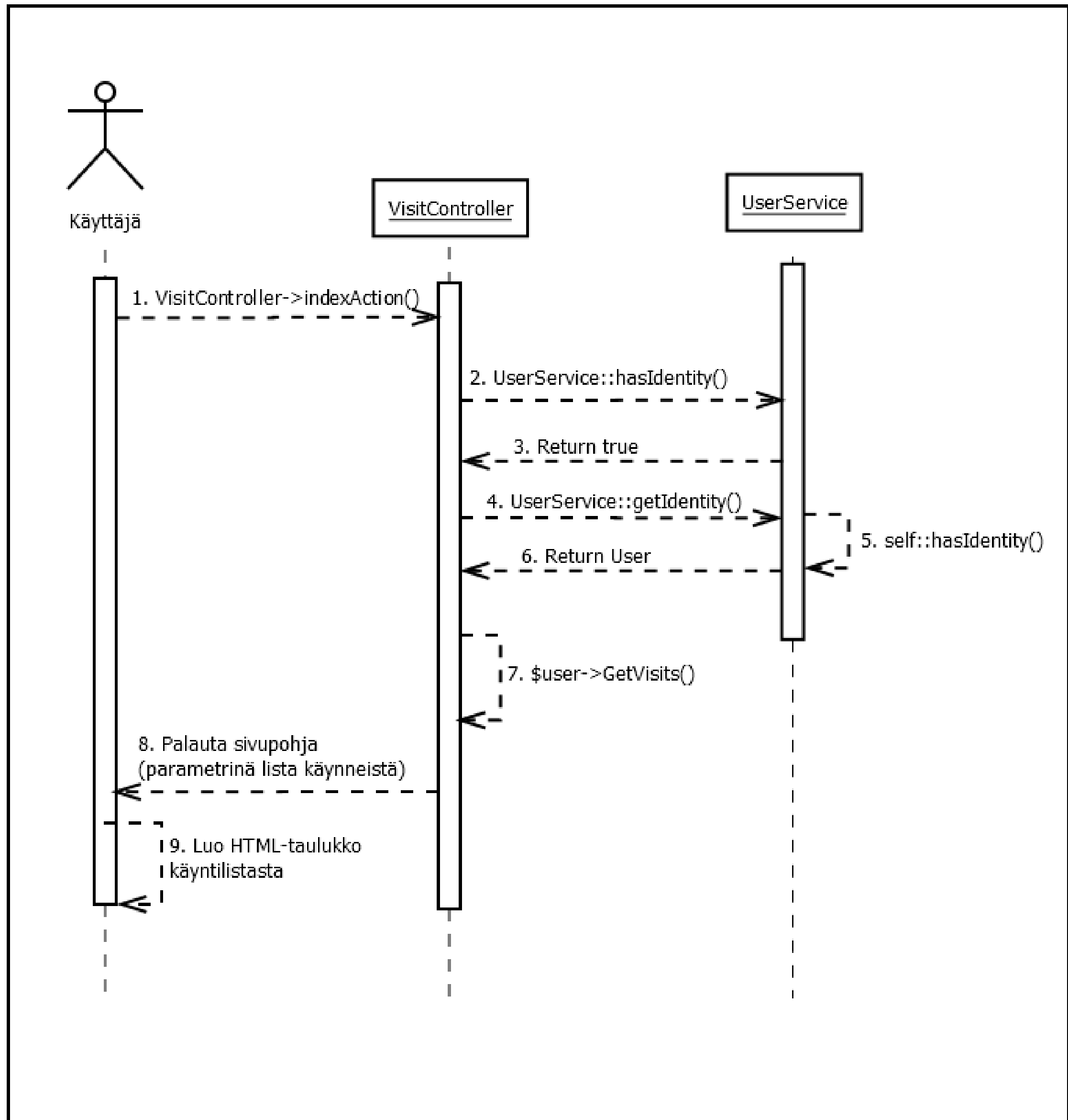


Kuvio 9. Käyttötapauskaavio käyntien tarkastelusta

Sekvenssikaaviossa (Ks. kuvio 10) käyttäjä kutsuu VisitControllerin indexAction-metodia. IndexActionissa VisitController kutsuu UserServicen hasIdentity-metodia tarkistaakseen käyttäjän kirjautumisen. UserServicen palauttaessa boolean-arvon true VisitController kutsuu UserServicen getIdentity-metodia. Ennen käyttäjän tietojen palauttamista UserService tarkistaa kirjautumisen vielä kerran ja mikäli käyttäjä on kirjautunut, palauttaa käyttäjän tiedot VisitControllerille User-olion muodossa. Kun VisitController on vastaanottanut User-olion, hakee se kirjautuneen käyttäjän käynnit palautetun User-olion GetVisits-metodilla. Kun lista käynneistä on haettu User-oliosta, palauttaa VisitController visit.phtml-sivupohjan, jolle on parametrinä lähetetty

käyntilista. Käyttäjän selain luo PHP:n avulla parametrinä annetun listan pohjalta taulukon.

PHP luo taulukon visit.phtml-dokumenttiin käyttämällä foreach-silmukkaa: jokaiselle käynnille luodaan echo-komennolla oma rivi taulukkoon ja lasketaan tuntitaksasta, käynnin kestosta ja toimenpiteiden hinnoista kokonaishinta, joka kirjataan taulukon hinta-sarakkeeseen.

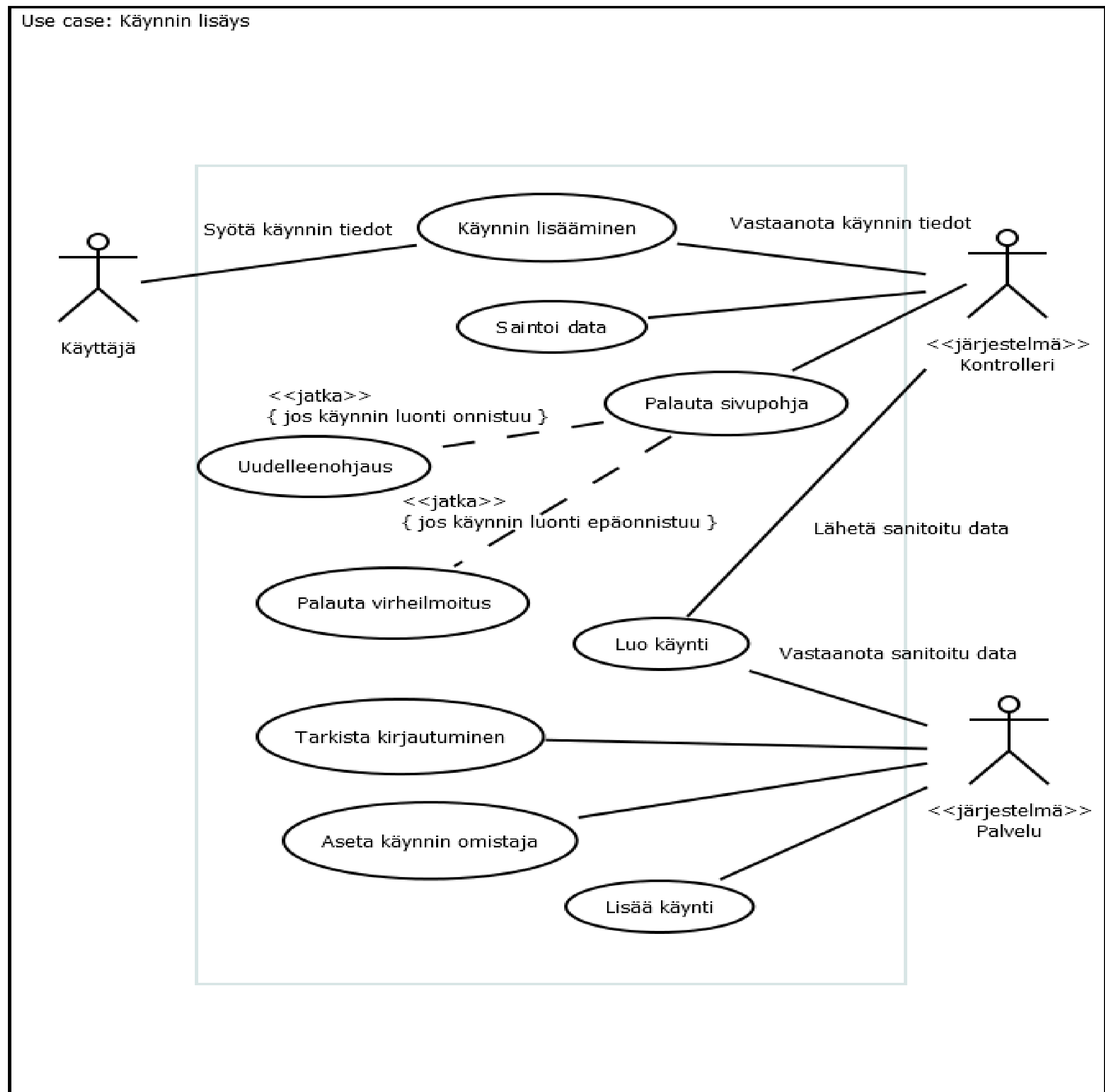


Kuvio 10. Sekvenssikaavio käyntien tarkastelusta

5.6 Käynnin lisäys

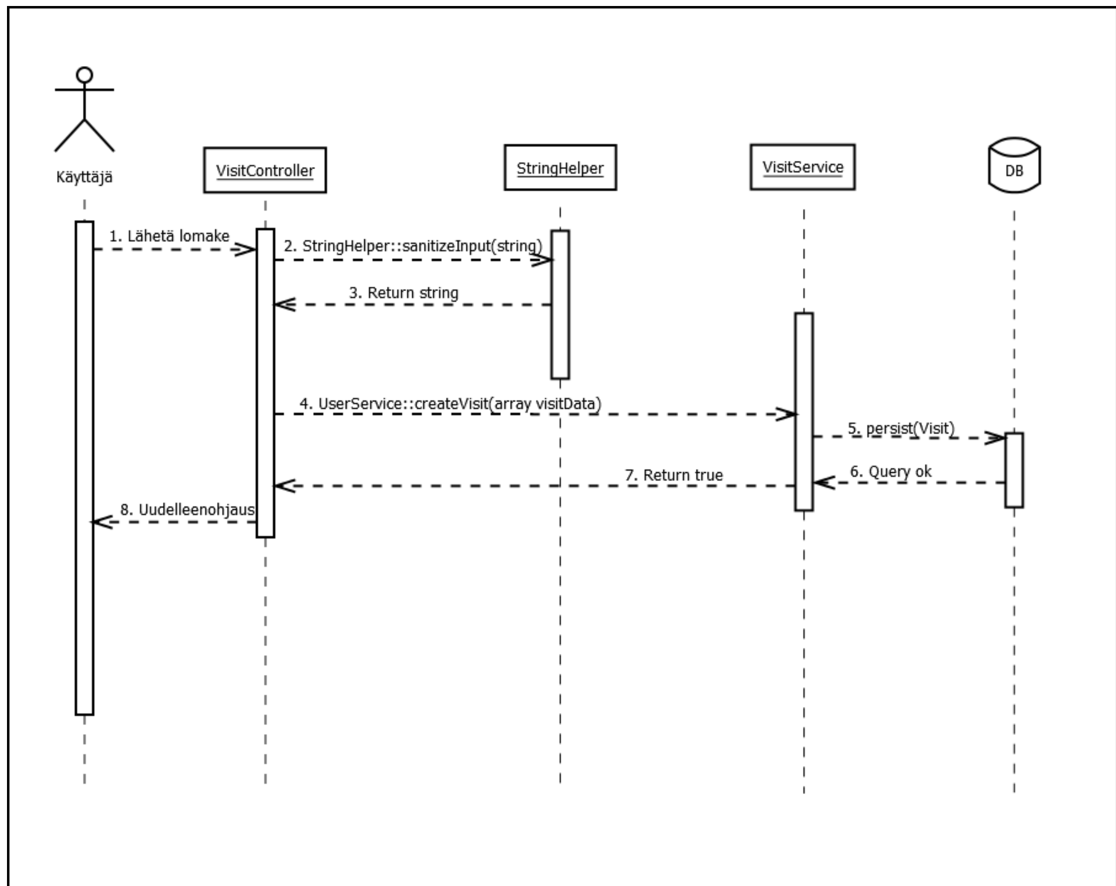
Seuraavaksi toteutettiin uuden käynnin kirjaaminen. Aiemmassa visit.phtml-sivupohjassa on painike, jota painamalla käyttäjä ohjataan uudelle newVisit.phtml-sivulle, joka sisältää lomakkeen, johon käyttäjä kirjaa käynnin tiedot ennen tietokantaan tallettamista. VisitControlleriin lisättiin uudet newAction ja addAction-metodit. NewAction palauttaa newVisit.phtml-sivupohjan käyttäjälle ja addAction vastaa uuden käynnin luomisesta.

Käyttötapauskaaviossa (Ks. kuvio 11) käyttäjä syöttää käynnin tiedon lomakkeeseen, jonka kontrolleri vastaanottaa. Tämän jälkeen kontrolleri sanitoi lomakedatan ja lähettää sanitoidut tiedot palveluluokalle. Palveluluokka tarkistaa käyttäjän kirjautumisen ja, jos käyttäjä on kirjautunut, luo uuden käynnin vastaanotettuaan kontrollerin sanitoiman lomakedatan. Luodulle käynnille palveluolio asettaa omistajan ja lisää luodun käynnin tietokantaan. Tämän jälkeen kontrolleri palauttaa sivupohjan sen perusteella, onnistuiko käynnin lisäys vai ei. Lisäyksen onnistuessa käyttäjä ohjataan uudelleen käyntien yleisnäkymäsivulle ja epäonnistuessa käyttäjälle esitetään virheilmoitus.



Kuvio 11. Käyttötapauskaavio käynnin lisäyksestä

Sekvenssikaaviossa (Ks. kuvio 12) käyttäjä lähettää VisitControllerille lomakkeen. VisitController lähettää lomakedatan StringHelper-luokalle kutsumalla sanitizeInput-metodia. StringHelper palauttaa VisitControllerille sanitoidun lomakedatan, josta VisitController luo taulukon. Näin luotu taulukko lähetetään VisitService-luokalle createVisit-metodia käyttäen. VisitService luo uuden Visit-olion ja entitymanagerin persist-metodia käyttäen lisää luodun olion tietokantaan. Tietokanta palauttaa Query ok-viestin, jonka jälkeen VisitService palauttaa boolean-arvo truen VisitControllerille. Tämän jälkeen VisitController uudelleenohjaa käyttäjän visit.phtml-sivulle.

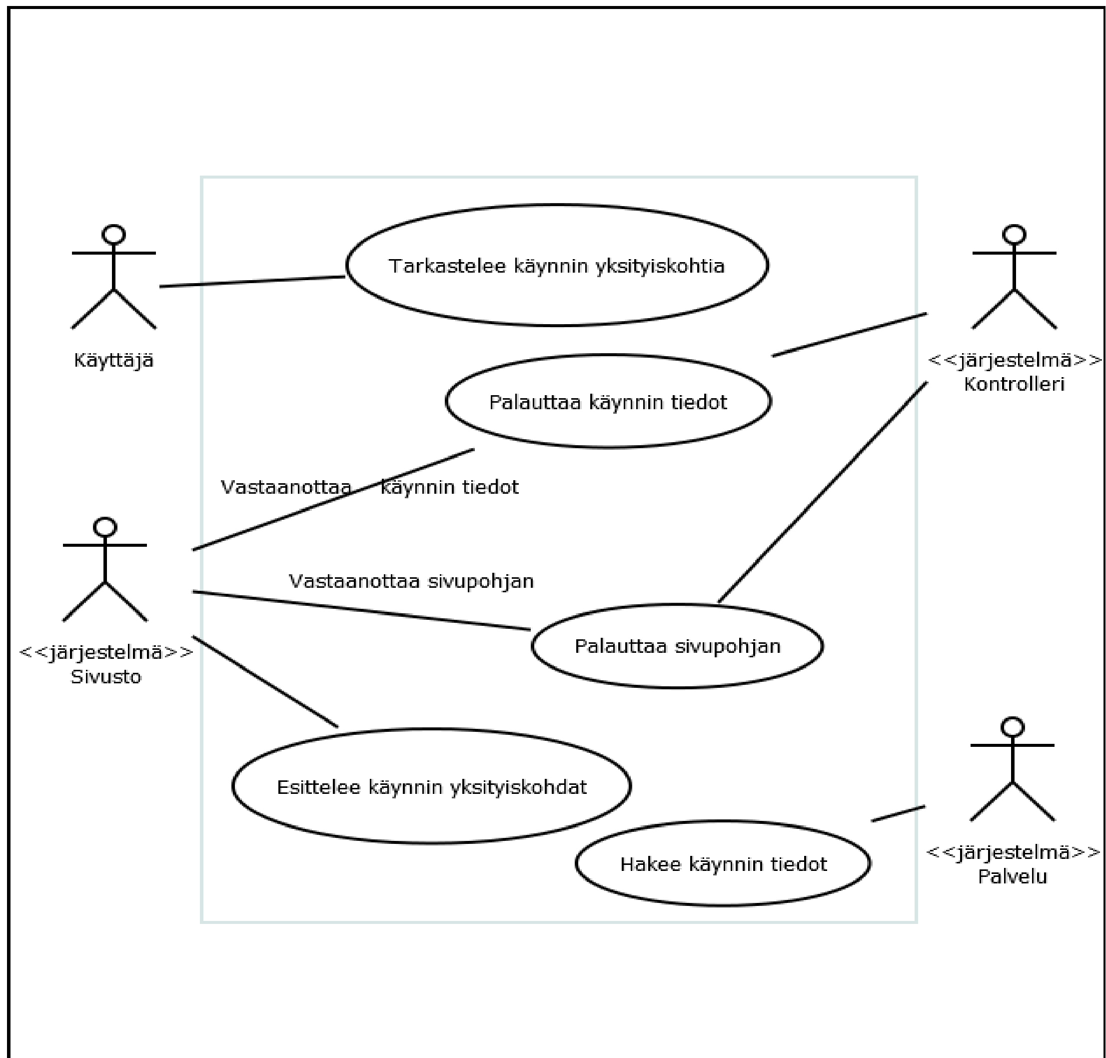


Kuvio 12. Sekvenssikaavio käynnin lisäyksestä

5.7 Käynnin yksityiskohtien tarkastelu

Toteutusjärjestyksessä seuraava oli käynnin yksityiskohtien tarkastelu. Yleisnäkymässä käyttäjälle esitetään käyntien tiedoista vain välttämättömimmät, joten käynnin tarkemman tarkastelun mahdollistamiseksi luotiin uusi visitDetails.phtml-sivupohja ja lisättiin VisitControlleriin uusi detailsAction-metodi, joka vastaa käynnin yksityiskohtaisten tietojen noutamisesta ja esittämisestä käyttäjälle. Lisäksi VisitServiceen lisättiin uusi metodi getVisitById, joka nimensä mukaisesti noutaa käynnin annetun id-numeron perusteella.

Käyttötapauskaaviossa (Ks. kuvio 13) käyttäjä haluaa tarkastella käynnin yksityiskohtia. Palveluluokka hakee käynnin tiedot, jotka kontrolleri palauttaa sivustolle sivupohjan ohessa. Sivusto vastaanottaa käynnin tiedot sekä sivupohjan ja vastaa käynnin yksityiskohtien esittelystä käyttäjälle.

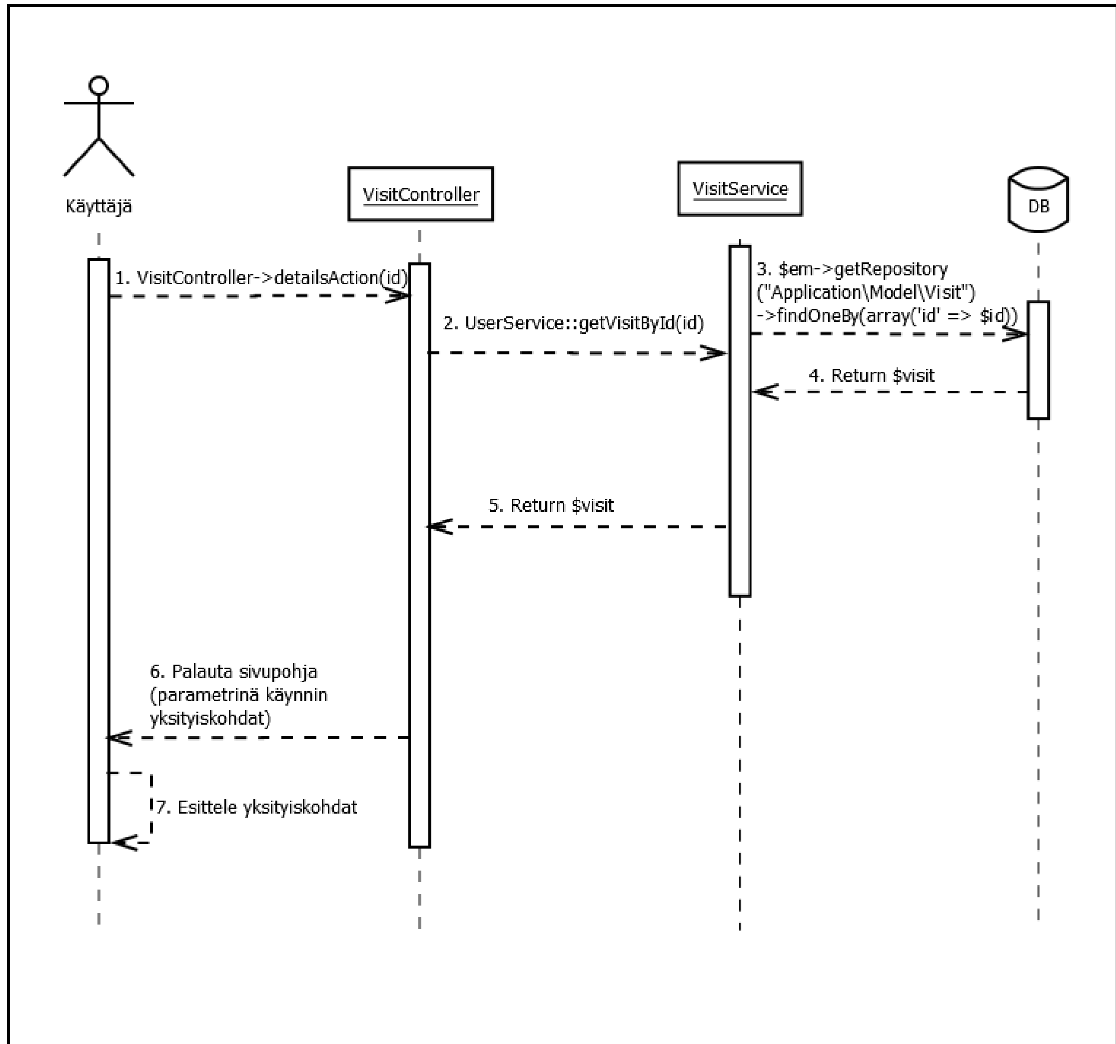


Kuvio 13. Käyttötapauskaavio käynnin yksityiskohtien tarkastelusta

Sekvenssikaaviossa (Ks. kuvio 14) käyttäjä kutsuu VisitControllerin detailsAction-metodia parametrilla id, joka on tarkasteltavan käynnin tunnistenumero. VisitController kutsuu VisitServicen getVisitById-metodia käyttäjältä saadulla id-tunnistenumerolla. VisitService puolestaan kutsuu Doctrinen entitymanagerin getRepository-metodia, jonka palauttama Repository-olio hakee tietokannasta käynnin findOneBy-metodilla annetun tunnistenumeron perusteella. Tietokanta palauttaa haetun käynnin, jonka VisitService palauttaa jälleen VisitControllerille. VisitController palauttaa visitDetails.phtml-sivupohjan, jonka parametrina annetaan haettu käynti. Käyttäjän selain esittelee käynnin yksityiskohtaiset tiedot osana visitDetails.phtml-sivupohjaa.

VisitDetails.phtml-sivulla käynnin tietojen esittely pohjaa samaan tekniikkaan kuin käyntien yleisnäkymäsivulla, sillä erotuksella, että käynnin omat tiedot esitellään taulukon ulkopuolella listaamalla jokainen tieto omalle rivilleen. Käynnin toimenpiteet

taas esitellään taulukkomuodossa. VisitDetails.phtml-sivu myös kutsuu käyttötapaus-kaaviossa mainittua UserServicen hasIdentity-metodia tarkistaakseen, että käyttäjä on kirjautunut sisään ja omistaa käynnin, jonka yksityiskohtia hän haluaa tarkastella.

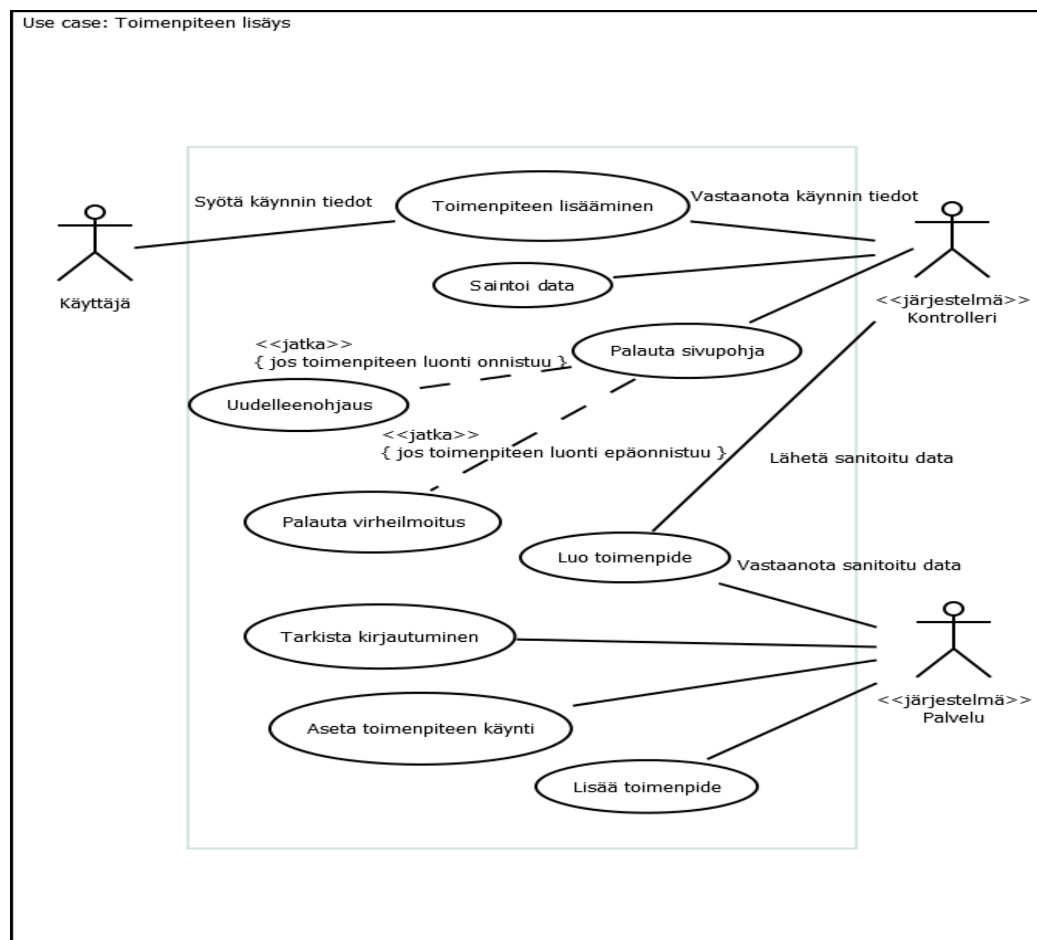


Kuvio 14. Sekvenssikaavio käynnin yksityiskohtien tarkastelusta

5.8 Käynnin toimenpiteen lisäys

Viimeisenä toteutettiin käynnin toimenpiteen lisäys. Toimenpiteet ovat vapaaehtoisia käynnin aikana tehtäviä tapahtumia, joita käyttäjä voi vapaasti lisätä käynnin yksityiskohtanäkymän ”Uusi toimenpide” -painikkeella. Tämä ohjaa käyttäjän uudelle action.phtml-sivulle, jossa käyttäjälle esitetään lomake, jossa on kentät vapaamuotoiselle kuvauksella sekä toimenpiteen hinnalle. Lisäksi luotiin uusi kontrolleri ActionController, joka vastaa action.phtml-sivupohjan tarjoamisesta käyttäjälle sekä uuden toimenpiteen luomisesta.

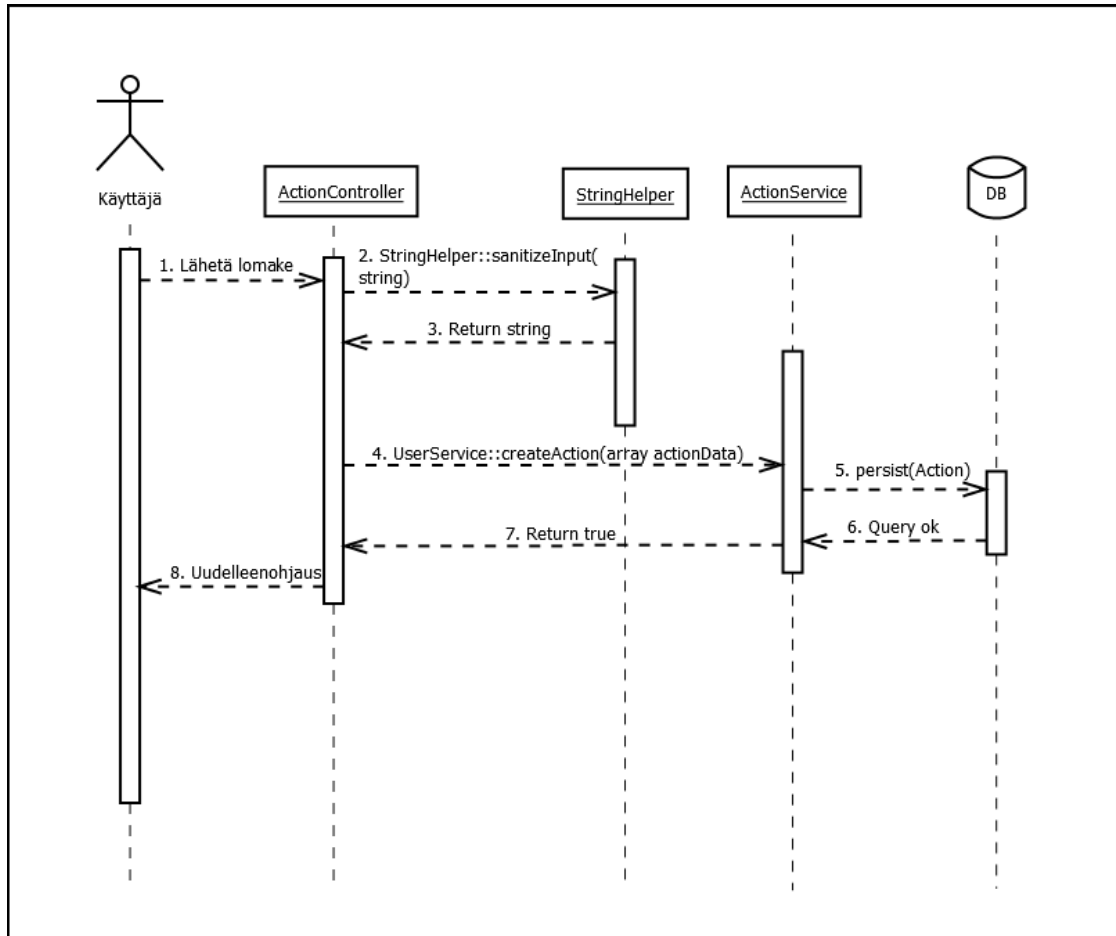
Käyttötapauskaaviossa (Ks. kuvio 15) käyttäjä haluaa lisätä uuden toimenpiteen osaksi valittua käyntiä. Käyttäjä syöttää toimenpiteen tiedot lomakkeeseen ja lähettää sen. Kontrolleri vastaanottaa käyttäjän syöttämät tiedot ja sanitoi ne. Kontrolleri lähettää tämän jälkeen sanitoidut tiedot palveluluokalle, joka ensin tarkistaa käyttäjän kirjautumisen järjestelmään. Vahvistettuaan käyttäjän kirjautumisen ja vastaanotettuaan kontrollerin lähettämät sanitoidut tiedot, luo palveluluokka uuden toimenpide-olion vastaanottamillaan tiedoilla. Tämän jälkeen palveluluokka asettaa toimenpide-oliolle sen omistavan käynnin ja lisää luodun toimenpide-olion tietokantaan.



Kuvio 15. Käyttötapauskaavio toimenpiteen lisäämisestä

Sekvenssikaaviossa (Ks. kuvio 16) käyttäjä lähettää lomakkeen ActionControllerille, joka kutsuu StringHelperin sanitizeInput-metodia. StringHelper palauttaa sanitizeInput-metodin sanitoiman datan takaisin ActionControllerille. ActionController kutsuu ActionServicen createAction-metodia ja lähettää parametrina toimenpiteen datan taulukkomuodossa. ActionController luo uuden Action-olion actionData-taulukon

pohjalta ja kutsuu entitymanagerin persist-metodia, joka lähettää tietokannalle kyselyn, jolla toimenpide kirjataan tietokantaan. Tietokanta palauttaa Query ok-viestin ActionServicelle, joka palauttaa ActionControllerille boolean-arvon true. ActionController ohjaa tämän jälkeen käyttäjän aiemmin valitun käynnin yksityiskohtanäkymään.



Kuvio 16. Sekvenssikaavio toimenpiteen lisäämisestä

5.9 Tietoturva

Jos verkkosovelluksessa aiotaan käsitellä käyttäjän henkilökohtaisia tietoja, on sovelluksen tietoturva tärkeässä osassa. Web-kehityksessä yleisimmät tietoturva-aukot muodostuvat SQL-injektioista ja XSS-hyökkäyksistä eli Cross site scriptingista.

SQL-injektio on hyökkäys sovelluksen tietokantaan, jossa käyttäjä pyrkii upottamaan esimerkiksi lomaketiedon mukana SQL-kyselyitä. Onnistunut SQL-injektio voi mahdol-

listaa hyökkääjälle muun muassa luottamuksellisten tietojen lukemista tai varastamista, tietokannan tietojen muokkaamista joko syöttämällä uutta dataa, muokkaamalla aiemmin lisättyä dataa tai poistamalla sitä. Pahimmassa tapauksessa hyökkääjä voi jopa lähettää komentoja palvelimen käyttöjärjestelmälle. (SQL Injection n.d.).

Potilaskirjausjärjestelmässä Doctrine 2 vastaa SQL-injektioilta suojautumiselta. Kaikki käyttäjän kirjaamat tiedot kirjataan tietokantaan Doctrinen entitymanagerista löytyvän persist-metodilla, joka on suojattu SQL-injektioilta. (Security n.d.).

XSS-hyökkäys on injektiohyökkäys, jossa hyökkääjä pyrkii upottamaan javascript-komentoja hyväntahtoisiin ja luotettaviin sivustoihin. XSS-hyökkäys tapahtuu, kun hyökkääjä käyttää web-sovellusta lähettämään pahantahtoista koodia toiselle loppukäyttäjälle esimerkiksi sivuston lomakkeen avulla. XSS-hyökkäys on poikkeuksellisen ikävä, koska hyökkäyksen uhri ei voi tehdä juuri mitään estääkseen kyseisiä hyökkäyksiä. Käyttäjän selaimella ei ole kykyä erotella hyvän- ja pahantahtoisen javascriptin välillä, vaan se suorittaa kaikki komennot, jotka verkkosivulla ovat. Koska selain luulee, että javascript-komento tuli luotettavasta lähteestä, pääsee skripti käsiksi kaikkiin evästeisiin, sessiotunnisteisiin tai muuhun luottamukselliseen tietoon, jota selain on mahdollisesti säilönyt sivustolta. XSS-hyökkäyksellä voidaan myös ylikirjoittaa HTML-sivun sisältöä. (Cross-site Scripting (XSS) n.d.)

Potilaskirjausjärjestelmä torjuu XSS-hyökkäyksiä sanitoimalla kaikki syötteet, jotka sivusto saa käyttäjältä. Sanitoinnissa syöte ajetaan PHP:n strip_tags-metodin kautta ja palautunut syöte tarkistetaan vielä varmuuden vuoksi preg_replace-metodilla läpi siten, että <script>-tunniste poistetaan syötteestä, jos sellainen löytyy.

Käyttäjän luodessa uutta tunnusta salataan syötetty salasana ennen tietokantaan kirjaamista. Mikäli tietokannan käyttäjätiedot kaapattaisiin, olisivat salasanatiedot paremmassa turvassa kuin selkokielisenä kirjattuna. Salasanojen tiivistämiseen käytetään PHP:n password_hash-metodia, joka vakioarvoisesti tiivistää annetun merkkijonon käyttämällä erillistä salasanatiivistealgoritmia ja satunnaisesti generoitua suola-arvoa. Potilaskirjausjärjestelmässä käytetään tiivisteen laskentaan bcrypt-algortmia, joka on vuonna 1999 julkaistu, Blowfish-salakirjoitusalgoritmiin pohjautuva algoritmi. Password_hash-metodissa bcrypt-algoritmi valitaan antamalla $algo-parametrille arvo PASSWORD_BCRYPT. Lisäksi metodiin voi halutessaan erikseen määritellä myös

algoritmin laskennallisen hinnan, joka määrää tiivisteen laskentaan käytettävän ajan, ja oman suola-arvon. Mikäli näitä arvoja ei aseteta käsin, generoi metodi sattumanvaraisen suolan ja käyttää algoritmin laskennassa vakiohintaa. Suorituksen jälkeen metodi palauttaa tiivistetyn merkkijonon, johon sisäänkirjautumisen aikana syötettyä salasanaa verrataan password_verify-metodilla. (password_hash n.d.).

6. Tulokset

Työn päätteeksi saatiin aikaan verkkosivusto, joka soveltuu asiakkaan omaan käyttöön. Täysin kaupallisesti käyttövalmista tuotetta ei saatu toimitettua, joten tuotetta tullaan jatkokehittämään alustavan käyttäjäkokeilun jälkeen. Asiakas oli toimitettuun sovellukseen tyytyväinen.

Projektissa saavutettiin kaikki asiakkaan vaatimukset sekä osa omista paranteluehdotuksista. Laskujen muodostaminen käynnin tiedosta jäi toteutumatta, mutta se tullaan todennäköisesti lisäämään tulevaisuudessa.

Lopullisessa tuotteessa pystyi rekisteröitymisen jälkeen lisäämään itselleen käyntejä ja toimenpiteitä sekä tarkastelemaan niitä suppeasti ja yksityiskohtaisesti. Käytettävyys saatiin asiakkaan toivomalle tasolle ja kaikki kriittisimmät bugit korjattiin. Käyttöliittymä myös mukautui dynaamisesti selaimen koon mukaan. Yleisimmistä käytössä olevista selaimista sovelluksen toimivuus varmennettiin Mozillan Firefox- ja Google Chrome-selaimilla. Microsoftin Internet Explorer- ja Edge-selaimilla sivustoa ei testattu, joten niiden käyttöä potilaskirjauksen kanssa ei voi tässä vaiheessa suositella. Myös Firefoxin ja Chromen vanhoilla versioilla voidaan havaita ongelmia käyttöliittymän toiminnassa, mutta näiden vanhojen selainversioiden tukemista ei tällä hetkellä koeta aiheelliseksi.

Asiakkaalta ei saatu tarkkoja vaatimuksia käyttöliittymästä, joten tässä vaiheessa päädyttiin hyvin minimalistiseen ulkoasuun. Sivuston ulkoasu tulee suoraan Zend Frameworkin omasta Twitter Bootstrap-teemasta ja omia tyylimääritelmiä ei työssä juurikaan ole. Asiakas oli kuitenkin tyytyväinen luotuun pohjaan.

Zend Framework 2:n käyttö sujui melko vaivattomasti muutamaa poikkeusta lukuun ottamatta. Varsinkin aluksi ongelmia muodostui uusien kontrollerien lisäämisessä ja

tietokannan alustuksessa. Alussa kontrollerien lisäämisessä unohtui niiden lisääminen sivustomodulin konfigurointitiedostossa olevaan invokables-taulukkoon. Tämä johti siihen, että sivua vaihdettaessa ilmestyikin virheilmoitus, jossa ilmoitettiin, että kontrolleria ei löytynyt. Ongelma saatiin korjattua selaamalla Zend Frameworkin dokumentaatiota tarkemmin. Tietokannan alustuksessa ongelmaksi muodostui Zend Frameworkin autoload-skriptin suoritusjärjestys. Alun perin tietokanta oli tarkoitus alustaa muusta sovelluksesta erikseen ajettavalla alustusskriptillä, mutta tämän epäonnistuttua päätettiin suorittaa alustus osana sivuston latausta. Alussa alustusskripti suoritettiin väärässä järjestyksessä, jolloin tärkeät globaalit muuttujat olivat vielä määrittelemättä. Ongelma ratkaistiin suoritusjärjestystä muuttamalla.

Doctrine 2 osoittautui hyväksi valinnaksi tietokannan käsittelyyn. Melkein jokainen sivuston osa käsittelee jotain tietokannan tietoa, jolloin sivukohtainen tietokantakyselyiden muodostaminen ja paluuviestinä saatujen tietojen käsittely olisi käynyt työlääksi. Sisäänkirjautumisen ohessa istuntoon talletettu käyttäjä-olio sisälsi kaikki tarvittavat tiedot, jolloin esimerkiksi käyntien yleisnäkymää muodostettaessa ei tarvittu uutta tietokantakyselyä vaan tietoja voitiin käsitellä suoraan oliotasolla. Näin sivuston suorituskyky ei kärsi toistuvista tietokantakyselyistä.

Doctrinen käytössä ilmeni yksi erikoinen ongelma Action-taulun luomisessa. Kun toimenpiteitä ryhdyttiin esittämään käyttäjälle ensimmäistä kertaa, hajosi potilaskirjauksen käyntien yleisnäkymä sekä yksityiskohtanäkymä. Apachen lokitiedostosta havaittiin, että PHP:lta oli muisti loppunut kesken. Ongelma ei ratkennut kuitenkaan muistia lisäämällä vaan virheilmoitus pysyi samana. Tietokantaa tarkastellessa kaikki tiedot näyttivät nopealla vilkaisulla oikeilta. Vasta tarkemmin katsottaessa huomattiin, että taulusta puuttuu kaksi kenttää, joihin on kuitenkin toimenpiteen lisäyksessä kirjoitettu dataa. Tämä johti Action-luokan lähdekoodin tarkasteluun ja kuumeisen virhejahdin jälkeen virheeksi paljastui kommenttiosion aloitus. Doctrine 2 lukee luokkien metadatan muuttujien ylläolevasta kommentiosiosta, jonka tulee olla tietyntyyppinen, esimerkiksi kommenttiosion tulee alkaa /** -merkinnällä. Action-luokan puuttuvien kenttien kommenttiosioista puuttui yksi *-merkki, jolloin Doctrine ei nähnyt kyseisiä muuttujia taulun kenttinä, vaan jätti ne huomiotta. Koska varsinaista tietokantavirhettä ei tapahtunut, ei myöskään Doctrine jättänyt minkäänlaista virheilmoitusta, joten ongelman ratkaisusta tuli melko haastavaa.

7. Pohdinta

Projektin toteuttamisen kannalta suurin haaste oli koulun ohessa käytävä täysipäiväinen työ. Työssä käynti rajoitti potilaskirjauksen kehittämiseen saatavilla olevaa aikaa ja siten johti useisiin viivästymisiin koko opinnäytetyön aikana. Samalla rajallinen aika ja sen epäonnistunut resursointi johtivat myös toisinaan pitkiin väleihin kommunikaatiossa, jolloin asiakaan kanssa ei keskusteltu juuri ollenkaan sovelluksen tilasta. Kun taas aikaa oli enemmän, päästiin kehityksessä nopeasti eteenpäin ja saatiin toteutettua sellainen versio, joka täytti asiakkaan toiveet ja vaatimukset, ja johon asiakas oli tyytyväinen. Toteutunut sovellus muodostaa hyvän pohjan, jota voidaan lähteä jatkokehittämään asiakkaan toiveiden mukaisesti, luultavasti edellä mainittujen jatkokehitysideoiden mukaisesti. Vaikka sovelluksen testaus jäi taka-alalle, toimivat sovelluksen pääominaisuudet tarkoitetulla tavalla ja suurempia virheitä sovelluksessa ei enää ole.

Vaikka toteutettu sovellus on jo itsessään toimiva kokonaisuus, myös parantamisen varaa on. Tuotteen ensimmäinen versio tulee aluksi Serecomin sisäiseen käyttöön, joten muutamia ominaisuuksia, kuten käyttöliittymän ulkoasua, tulee parannella ennen kuin tuote viedään markkinoille. Kun potilaskirjaus on ollut jonkin aikaa Serecomilla käytössä, voidaan sopia tarkemmin siitä, mihin suuntaan sovellusta lähdetään kehittämään ulkoasun kannalta. Tällaisia voisivat olla esimerkiksi kuvien lisääminen sivustolle ja käyttöliittymän responsiivisuuden parantaminen. Tällä hetkellä sivuston käyttöliittymästä hyvin pieni osa sisältää Twitter Bootstrapin lisäksi minkäänlaista javascriptia, vaan sivut luodaan PHP:lla.

Lisäksi, kun potilaskirjausta lähdetään kaupallistamaan, tarvitaan kyky laskuttaa järjestelmän käyttölisenssistä. Alustavissa keskusteluissa mainittiin kuukausimaksumallia, jossa käyttäjä maksaa Serecomille kuukausittain ja saa vastineeksi käyttöoikeuden sovellukseen. Koska omaa maksujärjestelmää ei ole aiheellista lähteä kehittämään vain yhden sovelluksen osaksi, on järkevää valita jokin aiemmista maksunvälittäjistä yhteistyökumppaniksi. Näistä maksunvälittäjistä Paytrail tarjoaa työkalut maksamiseen suomalaisissa verkkopankeissa sekä PayPal-palvelussa. Kun potilaskirjausta aletaan kaupallistaa, on Paytrail varmasti yksi harkittavista maksunvälittäjistä.

Mahdollinen jatkokehityssuunta on myös sivuston lokalisointi. Zend Framework 2 tarjoaa kehittäjälle valmiina lokalisointityökalut, mutta koska sivusto on ensimmäisessä versiossa vain suomenkielinen, ei lokalisointitiedostoja tässä vaiheessa tehty. Sivusto on kuitenkin ohjelmoitu siten, että lokalisointi voidaan ottaa käyttöön hyvin nopealla aikataululla.

Uutena ominaisuutena sovellukseen voisi lisätä laskutusjärjestelmän, joka muodostaisi tulostettavan laskun käyntiin kirjatuista tiedoista. Käynnin tuntitaksasta, kestosta ja toimenpiteistä voisi vaivattomasti muodostaa laskun erittelyn ja kokonaishinnan, jolloin käyttäjän tulisi vain lisätä laskuun tilinumero, viitenumero, eräpäivä ja asiakkaan osoitetiedot ja tulostaa. Laskuun voisi myös liittää mukaan viivakoodin pankkien itsepalvelupisteitä varten, joskin kyseisen viivakoodin käyttöä sääntelee Finanssialan Keskusliitto, jolloin sen käyttö vaatii tiettyjen reunaehtojen täyttymistä.

Tietoturva taas on jatkuvaa kilpajuoksua uusien haavoittuvuuksien kanssa. Esimerkiksi XSS-hyökkäyksiä koskevat, nyt tehdyt tietoturvaratkaisut voivat riittää vielä tässä vaiheessa, mutta tulevaisuudessa nykyiseen toteutukseen tulee puuttua. Myös SQL-injektioihin tulee kiinnittää enemmän huomiota sovelluksen ikääntyessä. Doctrine 2:n persist-metodi on injektioturvallinen, mutta kun sovelluksen tietokannassa alkaa olla satojatuhansia merkintöjä, johtavat Doctrinen tietoturvaoptimoinnit vähemmän injektioturvallisiin toteutustapoihin. Tällöin SQL-injektion riski kasvaa merkittävästi ja sen torjumiseksi tulee käyttäjän syöttämät tiedot tarkastaa entistä tarkemmin ja hylätä kaikki virheellinen syöte.

Projekti onnistui pääosin hyvin. Lukuun ottamatta yllä mainittuja ajankäytön ongelmia, saatiin toimitettua asiakkaalle käyttökelpoinen sovellus, joka ainakin alussa riittää asiakkaan tarpeisiin. Jotta sovellus saataisiin kaupalliseksi, tarvitaan vielä hieman parantelua etenkin käyttöliittymän puolella. Tällä hetkellä käyttöliittymä on melko minimalistinen ja se sisältää vain kaiken tarvittavan eikä yhtään enempää. Mobiililaitteilla sivusto toimii melko hyvin, mutta parantamisen varaa on. Sivuston mobiilitoteutus vain sovittaa työpöytäversion kentät mobiililaitteen ruudulle sopivampaan muotoon, mutta ei varsinaisesti helpota käyttöä esimerkiksi puhelimella tai tabletilla.

Projektin aikana varsinkin Zend Frameworkin ja Doctrinen osaaminen vahvistui. Aiempi kokemus varsinkin Zend Frameworkista oli hyvin pinnallista, käytännössä tiedossa oli vain perusteet, joilla voitiin tehdä yksinkertaisia sivustoja. Nyt opituilla taidoilla voidaan luoda merkittävästi laajempia kokonaisuuksia. Doctrinessa suurin oppiminen tapahtui varsinkin alkukonfiguraation puolella. Aiemmin Doctrinen käyttöönotto oli ollut täysin pimennossa, mutta projektin kautta varsinkin tietokannan alustamisprosessi tuli tutuksi.

Lähteet

Apache Core Features, AllowOverride Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/core.html#allowoverride

Apache Core Features, <Directory> Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/core.html#directory

Apache Core Features, DocumentRoot Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/core.html#documentroot

Apache Core Features, Order Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/mod_access_compat.html#order

Apache Core Features, ServerName Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/core.html#servername

Apache Core Features, <VirtualHost> Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/core.html#virtualhost

Apache Module mod_access_compat, Allow Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/mod_access_compat.html#allow

Apache Module mod_authz_core, Require Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. https://httpd.apache.org/docs/current/mod/mod_authz_core.html#require

Apache Module mod_env, SetEnv Directive. N.d. Apache HTTP Server Version 2.4. Viitattu 14.4.2016. http://httpd.apache.org/docs/2.0/mod/mod_env.html#setenv

Association mapping, One-To-Many, Bidirectional. N.d. Doctrine 2 ORM 2 documentation. Viitattu 14.4.2016. http://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/association-mapping.html#one-to-many-bidirectional

Basic Mapping, Identifiers / Primary Keys. N.d. Doctrine 2 ORM 2 documentation. Viitattu 14.4.2016. http://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/basic-mapping.html#identifiers-primary-keys

Basic MVC Architecture. N.d. Tutorialspoint. Viitattu 8.4.2016. http://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm

Bootstrap grid examples. N.d. Grid template for Bootstrap. Viitattu 14.4.2016. http://getbootstrap.com/examples/grid/

Creating and selecting a database. N.d. MySQL 5.7 Reference Manual. Viitattu 14.4.2016. http://dev.mysql.com/doc/refman/5.7/en/creating-database.html

Cross-site Scripting (XSS). N.d. OWASP. Viitattu 8.4.2016. https://www.owasp.org/index.php/SQL_Injection

Installation and Configuration, Obtaining an EntityManager. Doctrine 2 ORM 2 documentation. Viitattu 14.4.2016. http://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/configuration.html#obtaining-an-entity-manager

Object-relational mapping. N.d. Technopedia. Viitattu 8.4.2016. https://www.techopedia.com/definition/24200/object-relational-mapping—orm

password_hash. N.d. PHP manual. Viitattu 8.4.2016. http://php.net/manual/en/function.password-hash.php

Preface. N.d. PHP manual. Viitattu 14.4.2016. http://php.net/manual/en/preface.php

Routing and controllers. N.d. Zend Framework 2 2.4.8 documentation. Viitattu 14.4.2016. http://framework.zend.com/manual/current/en/user-guide/routing-and-controllers.html

Security. N.d. Doctrine 2 ORM 2 documentation. Viitattu 8.4.2016. http://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/security.html

SQL Injection. N.d. OWASP. Viitattu 8.4.2016. https://www.owasp.org/index.php/SQL_Injection

Tools, Database Schema Generation. N.d. Doctrine 2 ORM 2 documentation. Viitattu 14.4.2016. http://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/tools.html#database-schema-generation

Usage statistics and market share of Apache for websites. N.d. W3techs. Viitattu 14.4.2016. http://w3techs.com/technologies/details/ws-apache/all/all

Usage statistics and market share of Linux for websites. N.d. W3techs. Viitattu 14.4.2016. http://w3techs.com/technologies/details/os-linux/all/all