LAPIN AMK
Lapland University of Applied Sciences

LINUX-PALVELIMEN HALLINTAOHJELMA

Aarne Kyppö

Opinnäytetyö
Tietotekniikan koulutusohjelma
Insinööri (AMK)

2014

LAPIN AMMATTIKORKEAKOULU

TEOLLISUUDEN JA LUONNONVAROJEN OSAAMISALA

Tietotekniikan koulutusohjelma

LINUX-PALVELIMEN HALLINTAOHJELMA

2014

Toimeksiantaja Lapin Ammattikorkeakoulu, Ohjelmistotekniikan laboratorio

Tekijä Aarne Kyppö

Hyväksytty

Työ on luettavissa Theseus-verkkokirjastossa.

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön
tiivistelmä

Tekniikka ja liikenne
Tietotekniikka

Tekijä	Aarne Kyppö	Vuosi	2014
Toimeksiantaja	Ohjelmistotekniikan laboratorio pLAB		
Työn nimi	Linux-palvelimen hallintaohjelma		
Sivu- ja liitemäärä	36		

Opinnäytetyönä on toteutettu Linux-palvelimen hallintaohjelma. Hallintaohjelma mahdollistaa vain yksinkertaista palveluiden hallintaa, kuten tietokannan ja versioarkiston luontia. Työn tavoitteena ja haasteena on ollut hallintaohjelman muuttaminen modulaariseksi. Hallinnoitaviksi palveluiksi valikoituivat MySQL, PostgreSQL, Subversion, SSH, SFTP ja Disk Quota. Tavoitteena oli saada nämä palvelut hallintaohjelmalla hallinnoitavaksi.

Hallintaohjelma tehtiin Python-ohjelmointikielellä käyttäen Django framework:a. Lisäksi hallintaohjelma käytti Linux-palvelimelle asennettuja komentorivikomentoja palveluiden hallitsemista varten.

Hallintaohjelma modulaariseksi tekeminen onnistui hyvin. Palveluiden lisääminen hallintaohjelmaan on tehty helpoksi. Käyttöliittymästä tuli selkeä. Kaikkia hallintaohjelmalle asetettuja vaatimuksia ei kuitenkaan kyetty saavuttamaan, sillä kaikkia hallintaohjelman palveluiden toiminnallisuuksia ei ehditty tarpeeksi viimeistelemään. Testaaminen jäi kireän aikataulun vuoksi tekemättä. Testaaminen olisi ollut todella tarpeellista.

Avainsanat Modulaarisuus, Linux, Python, Django

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

School of Technology, Communication and Transport
Information Technology Programme

Author	Aarne Kyppö	Year	2014
Commissioned by	Software Engineering Laboratory pLAB		
Subject of thesis	Control program for Linux server		
Number of pages	36		

The subject of this thesis is control program for Linux-server. Control program included only the most important functionalities. It was challenging to make this program to be modular. This goal has been reached quite successfully.

Control program administers MySQL, PosgreSQL, Subversion, SSH, SFTP and Disk Quota services. With control program administrator can create databases for users. Control program also allows making Subversion repositories. Disk space of user can be restricted via control program. Also locking and disabling service is possible to do with control program.

I didn’t reach every goal, because I didn’t have enough time to finish every functionality. Exception handling was unfinished in some services. Control program wasn’t tested. Testing would have been important to do in service updates.

Keywords Modularity, Linux, Python, Django

SISÄLTÖ

TAULUKKO- JA KUVIOLUETTELO

TERMIT JA LYHENTEET

Framework	Kokoelma työkaluja web-sovelluksen rakentamiseen. (Rouse, M 2014)
SWAP-operaatio	Objektien arvojen vaihtaminen keskenään.
Validointi	Tiedon oikeellisuuden tarkistaminen
Renderöinti	Muuttujan arvon syöttäminen HTML-sivulle.
Subversion	Versionhallintaohjelma (Apache Software Foundation 2014)
Widget	Graafinen elementti (Higgins, D 2014)

1 JOHDANTO

Opinnäytetyöksi tehdään selainpohjainen sovellus, jonka avulla voidaan hallinnoida Linux-palvelinta. Ohjelman ei ole tarkoitus mahdollistaa kaiken kattavaa Linux-palvelimen hallintaa, vaan tarjota keskeisimpiä toiminnallisuuksia loppukäyttäjälle. Tällä hallintaohjelmalla lisätään käyttäjiä Linux-palvelimelle. Näille käyttäjille sitten myönnetään palveluita. Palveluina ovat SSH, SFTP, MySQL, Subversion ja PostgreSQL.

Sovelluksen on määrä olla modulaarinen, jotta siihen voidaan helpolla tavalla lisätä uusia palveluita. Hallintaohjelman laajentamista varten ei tehdä erillistä kirjastoa, vaan palvelut lisätään palveluluokan tekemisellä. Samankaltaisia palveluita on tarkoitus ryhmittää toimintoja yhdistelemällä.

Opinnäytetyössä ehkä haastavinta poikkeuksen käsittelyn tekeminen palveluille. Pitää tehdä paljon tarkastuksia, että tietokantaan ei tallennu väärä palvelun tila. Jos väärä tila kuitenkin tallentuu tai palvelun tilaa muutetaan hallintaohjelman ulkopuolelta, on hallintaohjelman kyettävä tarkistamaan tilan oikeellisuuden, ettei tule virheilmoitusta.

Sovelluksen yksi tarkoitus on tarkoitus nopeuttaa Linux-palvelimen hallinnointia. Hallintaohjelman päänäkymästä voi nopeasti ja helposti nähdä käyttäjien palveluiden tilat. Tällöin pääkäyttäjä näkee heti käyttäjien palveluiden tilan. Palveluiden tilan vaihtumisesta informoidaan käyttäjää. Tämä tulee myös säästämään aikaa, sillä tällöin pääkäyttäjän ei tarvitse erikseen lähettää sähköposteja ja kopioida tunnistautumistietoja viesteihin.

Hallintaohjelmaan olisi voinut liittää paljon enemmänkin toiminnallisuuksia. Hallintaohjelmalla voidaan luoda käyttäjälle versioarkisto tai tietokanta ja hallinnoida sitä. Hallintaohjelma ei mahdollista laajempaa versioarkistojen ja tietokantojen hallintaa. Tärkeintä ei ole toiminnallisuuksien määrä, vaan laatu. Palveluiden hallinta pitää toimia. Vasta sitten voidaan laajentaa ohjelmaa hallinnoimaan muitakin palveluita.

2 PALVELIMEN WEB-HALLINTAOHJELMAT

Monet Linux-palvelimet asennetaan ilman graafista käyttöliittymää (Kuutti 2011,19). Palvelimeen voidaan hallinnoida komentorivin kautta, mutta se voi olla aikaa vievää työtä. Palvelimen hallinnointi web-hallintaohjelman kautta tarjoaa hyvän vaihtoehdon. Tällöin myös kokemattomampi henkilö voi hallinnoida palvelinta. Web-hallintaohjelma asennetaan palvelimelle ja siihen ollaan yhteydessä internet-selaimen kautta. Kun palvelimen hallinnointi on näin toteutettu, ei tarvitse niin paljon käydä palvelinhuoneessa. Työaikaa säästyy kun palvelimen hallinnoinnin voi tehdä omalla työpisteellään. Myös ergonomia on parempi, sillä palvelinhuone ei yleensä ole työpaikan ergonomisin huone.

Opinnäytetyöni ei ole ainoa web-hallintaohjelma Linux-käyttöjärjestelmälle. Web-hallintaohjelmia on kaupallisia sekä vapaita. Web-hallintaohjelmat tarjoavat usein suuren määrän erilaisia toiminnallisuuksia, joista vain pieni osa tulee olemaan osana tekemääni opinnäytetyötä. Usein web-hallintaohjelma on tehty niin kattavaksi ettei tarvita muita ohjelmia palveluiden hallinnoimiseen. Nämä ohjelmat tarjoavat myös usein tietoa palvelimen ja sen palveluiden tilasta. Tietoa annetaan muiden muassa levytilan käytöstä ja IP-osoitteesta, mitä palvelin käyttää. (Web Hosting Hub 2014.)

Web-hallintaohjelmien tärkeänä ominaisuutena on palvelimen sähköpostipalvelun hallinnointi. Jos palvelin toimii myös sähköpostipalvelimena, niin tarvitaan toiminnallisuus sähköpostitilien luomiseen. Muussa tapauksessa kuitenkin usein halutaan sisällyttää palveluihin sähköpostin lähetystoiminto. Esimerkiksi käyttäjän rekisteröitymisen yhteydessä lähetetään sähköpostina vahvistusviesti käyttäjälle. Opinnäytetyönä tekemäni ohjelma lähettää luotavalle käyttäjälle sähköpostiviestin. Tämä viesti kertoo käyttäjälle käyttäjätilin luomisesta sekä sisältää informaatiomateriaalia hänelle suoduista palveluista. (Web Hosting Hub 2014.)

Laajennettavuus on tärkeä asia hallintaohjelmassa, sillä kaikkiin palveluihin ei ole tukea missään hallintaohjelmassa. Jos hallintaohjelma ei mahdollista palvelun hallinnointia, pitää tehdä liitännäinen, jolla hallinnointi mahdollistetaan. Havaitsin, että useisiin hallintaohjelmiin on mahdollista tehdä liitännäi-

siä. Ajentissa on mielestäni dokumentoitu hyvin liitännäisten tekeminen. Liitännäisen toiminnallisuus on yhdessä Python-luokassa, johon sidotaan XML-tiedosto määrittelemään liitännäisen ulkoasu. Opinnäytetyössäni liitännäisten tekeminen tapahtuu saman periaatteen mukaisesti. Ulkoasu tehdään tosin HTML-kielellä. (Pankov, E 2014a; Pankov, E 2014; Raive, S 2013.)

Tuote	**vapaa/kaupallinen**	**PostgreSQL**
cpanel	kaupallinen	ei
plesk	kaupallinen	on
ispconfig	vapaa	ei
ajenti	vapaa	on
kloxo	vapaa	ei
openpanel	vapaa	ei vielä
zpanel	vapaa	ei
ehcp	vapaa	ei
ispcp	vapaa	ei
virtualmin	kaupallinen	on
webmin	vapaa	ei
dtc	kaupallinen	ei
directadmin	vapaa	ei
interworx	kaupallinen	ei
blueonyx	vapaa	ei

Taulukko 1. Hallintaohjelmien vertailu.

Web-hallintaohjelmat yleensä mahdollistavat tietokantajärjestelmien hallinnoimisen. Niillä siis voidaan luoda tietokantoja, asettaa oikeudet tietokantoihin ja suorittaa muita tietokantajärjestelmille ominaisia tehtäviä. Valtaosa hallintaohjelmista mahdollistaa MySQL-tietokantajärjestelmän hallinnoimisen. Taulukosta 1 käy ilmi, että vapaista ohjelmista kuitenkin vain muutama mahdollistaa PostgreSQL:n hallinnoimisen. Taulukko 1 on tehty omien havaintojeni pohjalta. Tämä toiminnallisuus voidaan lisätä liitännäisillä. (Web Hosting Hub 2014.)

3 LINUX

Linuxissa on komentoja, joita voi ajaa vain pääkäyttäjänä. Tähän usein käytetään sudo-komentoa. Sudo-komento antaa mahdollisuuden ajaa komennon pääkäyttäjänä. Monet pitävät sudo-komentoa erinomaisena turvallisuuskäytäntönä. Pääkäyttäjän oikeudet voidaan saada myös kirjautumalla pääkäyttäjäksi. Tätä ei kuitenkaan suositella käytettäväksi. On myös mahdollista suoda käyttäjälle oikeus ajaa joitakin ohjelmia pääkäyttäjänä. Ei siis ole välttämätöntä antaa kaikkia pääkäyttäjän oikeuksia käyttäjälle joka tarvitsee suoritusoikeuden vain joihinkin pääkäyttäjän omistamiin ohjelmiin. On myös mahdollista myöntää suoritusoikeus niin, että käyttäjä ei joudu syöttämään salasanaansa ohjelmaa suorittaessaan. Näitä oikeuksia päästään muokkaamaan visudo-komennolla, jolla pääsee muokkaamaan /etc/sudoers-tiedostoa. Tällaista ”heikennettyä” pääkäyttäjä käytetään hallintaohjelman ajamiseen palvelimella. Jos joku saa kaapattua tämän käyttäjätunnuksen, niin tällä tavalla voidaan rajoittaa hänen mahdollisuuksiaan tehdä tihutöitä palvelimella. (Wallen, J 2010.)

Tiedostojen tai hakemistojen oikeuksia voidaan muuttaa chmod-komennolla. Oikeuksien asettamiseen on kaksi eri esitystapaa: symbolinen ja numeerinen. Numeerisessa esitystavassa käytetään 3-4 numeroa oikeuksien kertomiseen. Kolminumeroisessa esitystavassa ensimmäisellä numerolla asetetaan omistajan oikeudet. Toisella numerolla asetetaan omistajaryhmän oikeudet. Kolmannella numerolla asetetaan oikeudet muille käyttäjille. Numero on kokonaisluku, jonka vaihteluväli on 0-7. Numero muodostuu valittuja oikeuksia vastaavien numeroiden summasta. Suoritusoikeuttavastaa numero 1. Kirjoitusoikeutta vastaa numero 2. Ja lukuoikeutta vastaa numero 4. Jos siis tahdomme antaa tiedostoon tai kansioon omistajalle ja omistajaryhmälle kaikki oikeudet ja muille käyttäjille vain lukuoikeuden, on syötettävä numerosarja silloin ”774”. Opinnäytetyössäni luotavan käyttäjän kotihakemiston oikeudet asetetaan numerosarjalla ”770”. Tällöin muilla käyttäjillä ei ainakaan aluksi ole oikeutta lukea luotavan käyttäjän tiedostoja. (Computer Hope 2014.)

Yksi Linuxin tärkeistä konfiguraatiotiedostoista on fstab, jolla määritellään miten levyosiot liittyvät järjestelmään. Tämä on tekstitiedosto, jossa tiedot on laitettu taulukonomaiseen järjestykseen. Yksi rivi vastaa yhden levyosion tai tiedontallennus-laitteen tietoja. Rivin ensimmäinen sarake kertoo levyosion tai laitteen. Tämä tieto voidaan antaa kolmella eri tavalla. Se voidaan antaa polun muodossa. Se voidaan myös antaa otsakkeen muodossa, joita vastaavat polut ovat listattuna /dev/disk/by-label-hakemistossa. Ja se voidaan antaa myös UUID-tunnisteen muodossa. Toinen sarake kertoo hakemiston, johon tämä osio tai laite liitetään. Kolmas sarake kertoo osion tai laitteen tiedostojärjestelmän. Neljäs sarake sisältää optiot, jota liitettävälle osiolle tai laitteelle asetetaan. Näillä optioilla voidaan asettaa osiolle kirjoituskielto, ohjelmien suorituskielto sekä muita erilaisia estoja. Tämä voi olla hyödyllinen tietoturvan kannalta. Esimerkiksi ohjelmien suorituskielto voi estää haitallisten ohjelmien suorittamisen osiolla tai laitteella. Optioilla voidaan asettaa myös tiedostojärjestelmät käyttämään disk quota:a. Opinnäytetyöni muokkaa fstab-tiedostoa disk quotan käyttöön ottamiseksi. Viidennellä sarakkeella voidaan kertoa halutaanko osio tai laite varmuuskopioitavaksi, kun tehdään dump-työkalulla varmuuskopio. Kuudennessa sarakkeessa kerrotaan osion tai laitteen prioriteetti tiedostojärjestelmien tarkistusta varten. (Arch Linux 2014a; Arch Linux 2014b.)

Käyttäjän kotihakemiston kokoa voidaan rajoittaa Quota:a käyttämällä. Tällä tekniikalla voidaan rajoittaa käyttäjien sekä lohkojen että inoodien lukumäärää. Lohkot ovat 1 kilotavun kokoisia muistialueita ja inoodit ovat tiedostojen, hakemistojen ja objektien metatietorakenteita. Inoodit kertovat muun muassa tiedostojen oikeuksista, tiedostotyypistä ja muokkausajasta. Lohkojen ja inoodien rajoittamista varten on kaksi arvoa: softlimit ja hardlimit. Hardlimit on raja, jota käyttäjä ei voi ylittää. Kun käyttäjä ylittää softlimit-rajan, on hänellä grace period-ajan verran aikaa vähentää lohkojen tai inoodien lukumäärää. Jos grace period-aikaa ei ole asetettu, on softlimit käytännössä hardlimit. En ole nähnyt tarpeelliseksi rajoittaa inoodien lukumäärää opinnäytetyössäni, vaan tarkoituksenani on rajoittaa ainoastaan lohkojen lukumäärää. (Buse, J 2013; van Dooren, R 2003; Ippolito, G 2014.)

4 PYTHON

Python on ohjelmointikieli, jonka Guido Van Rossum loi vuonna 1992 ohjelmointikielen tekemistä käsittelevillä tunneilla ollessaan. Nykyään Python:n kirjastosta löytyy valtava määrä erilaisia moduuleja. Moduuleja on muiden muassa sähköpostien lähettämiseen ja tiedostojen pakkaamiseen. (Python Software Foundation 2014a; Python Software Foundation 2014b.)

C-kieli

```c
#include <stdio.h>

void swap(int *i, int *j) {
   int t = *i;
   *i = *j;
   *j = t;
}

void main() {
   int a = 10, b = 20;
   printf("Before. a: %d, b: %d\n", a, b);
   swap(&a, &b);
   printf("After. a: %d, b: %d\n", a, b);
}
```

Python-kieli

```python
def swap(a,b):
    b,a = a,b
    return a,b

if __name__ == '__main__':
    a = 10
    b = 20
    print "Before. a: %d, b: %d" % (a,b)
    a, b = swap(a,b)
    print "After. a: %d, b: %d" % (a,b)
```

Kuvio 1. Swap-operaation suorittaminen C-kielellä ja Python-kielellä.

Python:n syntaksi on yksinkertainen. Python on tarkoitettu näyttämään ikäänkuin luonnolliselta kieleltä. Kun verrataan Python-kielisen ohjelman ja C-kielisen ohjelman swap-operaatiota, nähdään kuinka Python-kielellä on selkeästi C-kieltä yksinkertaisempi syntaksi. Pythonissa funktiot määritellään niin, että funktion nimen eteen kirjoitetaan avainsana: ”def”. Sana osoittaa Python tulkille, että kyseessä on funktio. Funktion nimen jälkeen kirjoitetaan funktion parametrit kuten C-funktiossakin. Funktion määrittely päättyy Python-kielessä kaksoispisteeseen. Kuvio 1:n C-kielisessä funktiossa käytetään aaltosulkuja rajaamaan ne komennot jotka kuuluvat funktioon. Python käyttää sisennyksiä tähän tarkoitukseen. (Python Software Foundation 2014c; Python Software Foundation 2014d.)

Python-kielessä kaikki sellaiset lausekkeet, jotka sisältävät toisia lausekkeita (combound statement), käyttävät sisennystä erottaakseen otsikko-osan toteutusosasta. Tällaisia lausekkeita ovat if-,while- ja for-lausekkeet. Myös funktiot ja luokat kuuluvat tähän joukkoon sekä try-lauseke. Kuvio 1 näyttää, että myös C-koodissa käytetään sisennyksiä. Tämä tekee koodista hieman

luettavampaa kuin, että ne olisi kirjoitettu kaikki samalle sisennystasolle. Olen havainnut, että sisennyksien käyttö ohjelmakoodia tehtäessä on hyvin yleistä myös muissa ohjelmointikielissä. Koska sisennyksiä yleisesti käytetään ohjelmakoodin jäsentämiseen, miksi tarvitaan aaltosulkuja lähdekoodin jäsentämiseen. Kun sisennyksiä käytetään ohjelmakoodin jäsentämiseen, tarvitaan kunnon editori. Muistelen että joku editori ei automaattisesti sisentänyt koodia niin, että Python-tulkki olisi sen hyväksynyt. (Python Software Foundation 2014c; Python Software Foundation 2014d.)

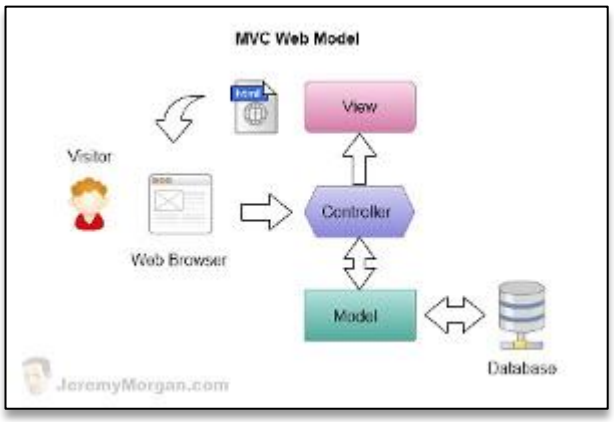
```
def funktio(number):
    if number%2 == 0:
        return None
    return number/0
if __name__ == "__main__":
    print funktio(2)
```

Kuvio 2. Esimerkkikoodi.

Python on tulkattava kieli. C-kielinen ohjelma käännetään kääntäjällä ensin konekieliseen muotoon ja sitten vasta se suoritetaan. Python sen sijaan tulkitaan ja ajetaan suoraan Python-tulkilla. Tuo tulkki on tehty C-kielellä. Ohjelma tosin on nopeampi C-kielellä tehtynä kuin Python-kielellä tehtynä. Jos siis ohjelman on oltava todella nopea, on syytä käyttää C-kieltä Python-kielen sijaan. Koska Python on tulkattava, ei se anna virheilmoitusta suorittaessaan kuviossa 2 olevaa funktiota. Tein C-kielellä melko samanlaisen ohjelman. Tätä käännettäessä kääntäjä antoi varoituksen nollalla jakamisesta. Kun Python-tulkki ei varoita niistä ohjelmointivirheistä, jotka ovat koodissa, jota ei ajeta, voivat nämä ohjelmointivirheet tulla yllätyksenä. Tämä voi olla varsin ikävää siinä tapauksessa, jos ohjelma on jo tuotantokäytössä. Pikaisen tarkastuksen jälkeen näytti siltä, että Python-tulkki ei tarjoa mahdollisuutta suoraan kääntää Python-koodia. (Lott, S 2014; Kokkarinen, I 2004. 254.)

5 DJANGO

5.1 MVC-arkkitehtuuri Djangossa



Kuvio 3. MVC. (ks. Morgan, J 2013)

Django on MVC-framework. MVC-arkkitehtuuri jakaa sovelluksen kolmeen osaan kuten kuvio 3 pelkistetysti osoittaa. Aluksi käyttäjä syöttää internetselaimeensa osoitteen, jota vastaavan sivun hän haluaa nähdä. Pyyntö ohjautuu sitten ohjain-osalle(Controller). Ohjain-osa pyytää sitten mallio-salta(Model) tarvitsemansa tiedon. Tämän jälkeen saatu tieto ohjataan näkymä-osalle, joka renderöi tiedon HTML-sivuksi ja palauttaa sen käyttäjälle. (Holovaty, A – Kaplan-Moss, J 2014a.)

Kun sovellus on jaettu kolmeen eri kokonaisuuteen, sovelluksen yhtä osaa voidaan kehittää vaikuttamatta toisien osien toimintaan. Tämä helpottaa työtä varsinkin silloin, kun on monta työntekijää tekemässä samaa sovellusta. Tämä sovelluksen jakaminen erillisiin kokonaisuuksiin mahdollistaa myös sen, että kaikkien ei tarvitse välttämättä osata kaikkia sovelluksessa käytettyjä kieliä. Esimerkiksi Django-projektissa ei ole välttämätöntä kaikkien osata Pythonia. Jos on taitava web-sivujen tekijä ja hän vastaa ainoastaan näkymäpuolesta, voi olla tarpeetonta käyttää kallista työaikaa Pythonin opetteluun. Djangon arkkitehtuuri kuitenkin jonkin verran eroaa MVC-arkkitehtuurista, ja siksi siitä käytetään myös nimitystä MTV-framework. (Holovaty, A – Kaplan-Moss, J 2014a.)

5.2 Django ORM

ORM tarkoittaa tiedon mäppäämistä objektien ja tietolähteiden välillä. Se on toisin sanoen tietolähteiden muokkaamista objektien kautta. Nämä objektit ovat kytköksissä tietolähteisiin ja niitä muokkaamalla voidaan saada haluttu muutos aikaan tietolähteessä. Olen havainnut, että usein tietolähteenä on tietokanta. Tämän vuoksi en käsittele muun tyyppisiä tietolähteitä. Kun tietokantaoperaatiot tehdään objektien kautta, tarjoaa se yhtenäisen rajapinnan tietokantajärjestelmille. Syntaksi kyselylle voi vaihdella tietokantajärjestelmien välillä. Mutta ORM:ssa objektien ei välttämättä tarvitse tietää tietokannan tietokantajärjestelmää. Tämä toki edellyttää sitä, että käytetty ORM on kunnolla tehty. (Rouse, M 2008; Arvin T 2014; LearnNowOnline 2012.)

```
from django.db import models

class Publisher(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()

class Author(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()

class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()
```

Kuvio 4. Tietokannan rakenne malliluokkina. (ks. Django Software Foundation 2014f)

Djangossa ORM-objektit ovat Model-luokan aliluokista luotavat objektit. Kuviossa 4 näkyy selkeästi tietokannan rakenne. Model-luokka sisältää kaiken

toiminnallisuuden mitä tarvitaan. Yksi malliluokka, joka perii Model-luokan, vastaa yleensä yhtä tietokantataulua. Näin ollen kuvio 4:ssa olevat luokat vastaisivat kolmea tietokantataulua. Tauluja tuleekin neljä, koska Book-luokassa on annettu attribuutin tyypiksi ManyToManyField. Tämä tarkoittaa sitä, että halutaan luoda monta moneen-suhde luokkien Book ja Author välille. Yhdellä kirjalla voi olla monta kirjoittajaa ja yhdellä kirjoittajalla voi olla monta kirjaa. Koska monta moneen-suhde on valittu luokkien Book ja Author välille, luo Django uuden taulun joka yhdistää nämä luokat. Kun kuviossa 4 määritellyt luokat halutaan tehdä tietokantatauluiksi, voidaan se tehdä ajamalla komento ”*python manage.py syncdb*”. Kyseisellä komennolla ei kuitenkaan voida päivittää olemassa olevan tietokantataulun rakennetta. Jos siis halutaan lisätä kuviossa 4 määriteltyyn luokkaan uusi kenttä ja saada uusi kenttä näkymään tietokannassa, sitä ei voi tehdä tällä komennolla. (Holovaty, A – Kaplan-Moss, J 2014a.)

```
p = Publisher.objects.create(name="Julkaisija Oy",address="Koivuranta 10",
    city="Rovaniemi", state_province="Lappi", country="Suomi",
    website="www.julkaisija.fi")
p.save()
b = Book.objects.create(title="Oppikirja",publisher=p,
    publication_date=datetime.date(2014,3,15))
b.save()
```

Kuvio 5. Tiedon lisääminen tietokantaan objektien välityksellä.

```
mysql> insert into publisher (name,address,city,state_province,country,website) values
    -> ("Julkaisija Oy","Koivuranta 10","Rovaniemi","Lappi","Suomi","www.julkaisija.fi");
mysql> insert into book (title,publisher_id,publication_date) values
    -> ("Oppikirja",LAST_INSERT_ID(),"2014-03-15");
```

Kuvio 6. Tiedon lisääminen tietokantaan SQL-kielellä.

Kun tietoa aletaan lisäämään tietokantaan, on havaittavissa kuinka Djangon ORM helpottaa huomattavasti tietojen lisäämistä. Kun ollaan yhteydessä tietokantaan näiden malliluokkien välityksellä, luodaan QuerySet-objekti malliluokan Manager:n kautta. Jokaisella malliluokalla on ainakin yksi Manager. Tämä Manager, joka oletuksena luodaan malliluokalle, on nimeltään 'objects'. Kuvioiden 5 ja 6 esimerkkitapauksessa lisätään ensin julkaisija ja sitten lisätään kirja julkaisijalle. Kuviossa 5 on kerrottuna tiedon lisääminen Djangon ORM:a käyttäen. Kuviossa 6 tieto lisätään SQL-kielellä. Tapa liittää julkaisija

kirjaan on näissä erilainen. SQL-kielisessä kyselyssä käytetään LAST_INSERT_ID()-komentoa. Tämä komento antaa viimeisimmän generoidun id:n(Oracle 2014). Tämä on vasta luomamme julkaisijan id. Ja tämä id sitten annetaan luotavan kirjan publisher_id:n arvoksi. ORM tekee tiedon tallennuksen aivan erillä tavalla. ORM:ssa ei etsitä viimeisen insert:n id-numeroa, vaan annetaan julkaisija-objekti suoraan parametrina luotavalle kirjalle. Tämä on mielestäni luontevampi tapa sitoa julkaisija luotavaan kirja-tietorakenteeseen. Jos pitäisi liittää useiden tietokantataulujen rivejä tähän luotavaan kirja-tietorakenteeseen, pitäisi id-numeroita tallentaa muuttujiin. ORM:ssa käytetään save-funktiota. Jos save-funktiota ei kutsuta, tiedot eivät tallennu tietokantaan. (Rouse M 2008; Holovaty, A – Kaplan-Moss, J 2007a; Django Software Foundation 2014b.)

5.3 Djangon lomakkeet

Djangossa on kirjasto lomakkeiden luomiseen ja käsittelemiseen. Tällä kirjastolla voidaan generoida lomakkeen kentät HTML-sivun elementeiksi. Kirjasto tarjoaa myös mahdollisuuden lomakkeen validointiin. Luotava lomake on forms.Form-luokan aliluokka. Tähän luokkaan sitten lisätään halutut kenttien nimet ja asetetaan niille tyypit. Djangossa on useita valmiita tyyppejä. Tyypin mukaan määräytyvät validointisäännöt. Validointi on monessa suhteessa tarpeellinen. Ensiksi sillä voidaan rajoittaa kentän kokoa niin, ettei käyttäjä anna liian pitkää merkkijonoa. Jos kentässä on 100 merkkiä ja tietokannan kentän maksimikoko on 60, tulee ongelmia kun yritetään siirtää tuota merkkijonoa tietokantaan.

Lomakkeen kentälle voidaan asettaa muotovaatimukset. Esimerkiksi EmailField-kentän arvon on oltava sähköpostiosoitteen muodossa. Jos tätä validointia ei suoritettaisi, voisi vääränmuotoinen sähköpostiosoite mennä tietokantaan, sillä monissakaan tietokantajärjestelmissä ei ole tietotyyppiä sähköpostiosoitteelle (W3schools 2014). Yksi kirjaston tärkeä ominaisuus on myös se, että sillä voidaan saada lähetetyn lomakkeen kenttien arvot Pythonin tietotyyppeinä. Djangossa on myös mahdollista tehdä omia lomakkeen kenttätyyppejä. Itse tein käyttäjänimelle kenttätyypin, koska annetun käyttäjänimen pitää olla tietyn muotoinen. (Django Software Foundation 2014d; Django Software Foundation 2014e.)

6 TIETOTURVA

6.1 Tietoturvauhkat Linux-palvelimelle

Koska Linux on avoin järjestelmä, on se suljettua Windows-järjestelmää turvallisempi. Tämä mahdollistaa haavoittuvuksien paremman havaitsemisen, koska useimmilla henkilöillä on mahdollisuus tarkastella lähdekoodia. Tämän haavoittuvuuden voi sitten jopa itse korjata jos vain tahtoo. Tietysti voi myös vinkata ohjelman kehittäjille haavoittuvuudesta. Jos taas suljetussa järjestelmässä on vakava haavoittuvuus, niin ohjelman käyttäjien ei ole ehkä edes mahdollisuutta saada tietää siitä. Silloin ei myöskään voida varautua tuota uhkaa varten. (Kuutti 2011. 271.)

Oikeuksien myöntämisessä tulee noudattaa vähimmän oikeuden periaatetta. Käyttäjille ei siis tule antaa enemmän oikeuksia kuin he tarvitsevat. Tällä voidaan minimoida riskit sen skenaarion suhteen, että joku saa kaapattua toisen käyttäjätilin. Kun luodaan tiedosto tai hakemisto, niin sen oikeudet määräytyvät umask-arvon mukaan. Tarkistin, että oman koneeni käyttöjärjestelmän umask-arvo oli 022. Tämä tarkoittaa sitä, että käyttäjä saa kirjoitusoikeudet ja kaikki saavat lukuoikeudet oletuksena tiedostoa luotaessa. Kun umask on asetettu arvoon 022, kaikki käyttäjät silloin siis pääsevät lukemaan toisen käyttäjän tiedostoja. Tämä on mielestäni huono käytäntö. Parempi umask:n arvo voisi olla 077, jolloin muilla käyttäjillä ei ole lukuoikeutta. (nixCraft 2012.)

6.2 Tietoturvauhkat hallintaohjelmalle

Internet-palveluita kohtaan kohdistuu useita tietoturvauhkia. Kun käyttäjä lähettää selaimella internet-palveluun tietoa, ei voida lähtökohtaisesti olettaa käyttäjän olevan rehellisissä aikeissa. Internetpalveluita kohtaan on monenlaisia erilaisia hyökkäyksiä, joista osaa käsitellään tässä kappaleessa. Internetpalvelun ylläpitäjän suojattava palvelu kaikilta hyökkäyksiltä, mutta hyökkääjä tarvitsee vain yhden haavoittuvuuden voidakseen hyökätä. Tietysti internetpalvelun ylläpitäjä voi jättää suojaamatta palvelun joitain hyökkäyksiä kohtaan. Näitä turvatoimia kannattaa mitoittaa sen mukaan kuin sovelluksen turvallinen käyttö vaatii. Jos on kyseessä internetpalvelu, joka ei käytä tietokantaa, ei silloin luonnollisestikaan tarvitse suojautua SQL-injektiota vastaan.

Olen kuitenkin sen havainnut, että internet-palvelu melko todennäköisesti käyttää tietokantaa.

```
select * from books where name = ""; delete from publisher where "a" = "a";
```

Kuvio 7. SQL-injektio.

```
>>> i = ServiceType.objects.filter(
...     name="'' OR 'a'='a'").values("name")
>>> len(i)
0
>>> s = str(i.query)
>>> cursor.execute(s)
3L
>>> len(cursor.fetchall())
3
```

Kuvio 8. SQL-injektion tekeminen Djangon ORM:a käyttäen.

SQL-injektio on yleinen hyökkäys. Jos selaimelta saadut arvot liitetään tietokantakyselyyn käsittelemättä, voi niiden vaikutukset olla tuhoisia tietokannan tiedoille. Kuvio 7 osoittaa sellaisen kyselyn, johon on kirjoitettu kirjan nimen sijaan merkkijono ”"; delete from publisher where "a" = "a”. Kysely siis hakee kirjaa, jonka nimeksi on annettu tyhjä merkkijono. Puolipiste päättää edellisen kyselyn ja sitten lisätään toinen kysely, jonka on tarkoitus poistaa kaikki rivit publishers-tietokantataulusta. Taulu tietysti täytyy olla olemassa, että sieltä voidaan poistaa kaikki rivit. Django onneksi tarjoaa tähän hyökkäykseen ratkaisun. Kun käytetään Djangon QuerySet-objekteja, niin Djangon tietokanta-ajuri siistii tuotettavan SQL-koodin, jotta injektiot eivät pääsisi tekemään tuhojaan. Kuviosta 8 nähdään, kuinka Django siistii kyselyn ennen kuin se antaa sen tietokantajärjestelmään. Nimen sijaan on kirjoitettu merkkijono: ”'' OR 'a'='a'”. Tämän merkkijonon tarkoitus on saada kysely palauttamaan kaikki rivit taulusta. Tämän kyselyn tarkoitus on kuitenkin hakea ne rivit, joissa nimi on sama kuin käyttäjän antama nimi. Koska injektio on antanut kyselyyn nimelle arvoksi tyhjän merkkijonon, tietokannasta ei löydy nimeä. Kysely ei palauta yhtään riviä. Tämä johtuu siitä, että Django on siistinyt kyselyn. Kun QuerySet:stä otetaan kysely ja suoritetaan se suoraan tietokantajärjestel-

mään, niin kysely meneekin läpi. Injektion vaikutuksesta kysely palauttaakin rivejä. Jos siis ei ole pakko suoraan syöttää kyselyitä tietokantajärjestelmään, kannattaa suorittaa kyselyt malliluokkien välityksellä. (Holovaty – Kaplan-Moss 2014b; Django Software Foundation 2014a; Django Software Foundation 2014b.)

Toinen käsiteltävä hyökkäys on Cross site Scripting(XSS). Tässä hyökkäyksessä sisällytetään asiakaspuolen skriptejä suoritettavaksi muiden käyttäjien selaimissa. Nämä skriptit ovat tehdyt esimerkiksi Javascript-kielellä. Yleensä se tapahtuu niin, että käyttäjä syöttää skriptit tietokantaan. Kun sitten tietokannasta haetaan tietoa näytettäväksi internet-sivulla, voidaan ladata myös käyttäjän skripti, joka sitten suoritetaan muiden käyttäjien selaimissa. Kun käytetään Djangon template:ja, saadaan suoja valtaosaan XSS hyökkäyksistä. Kun Djangossa lisäsin HTML-sivulla näytettäväksi merkkijonon: ”<script>alert('asdasd');</script>”, se näkyi sivulla ”<script>alert('asdasd');</script>”-merkkijonona. Skriptiä ei siis ajettu. Kun syötin tuon saman merkkijonon suoraan HTML-sivulle, skripti ajettiin. Tämä johtuu siitä, että Django siistii HTML-sivulle lisättävät muuttujat oletuksena. Muuttujat voidaan kyllä asettaa turvallisiksi safe-filter:llä. Eli voidaan kertoa HTML-sivun renderöijälle sisällytettävän muuttujan olevan turvallinen. Tällöin muuttujan arvoa ei siistitä Tätä mahdollisuutta tulee kuitenkin käyttää varovaisesti. Edellä kerrottu skripti, jossa on vain alert-komento ei ole kovinkaan vahingollinen. Javascriptillä voi kuitenkin tehdä paljon ilkeämpää. Esimerkiksi komento ”window.location.href=’http://www.lapinamk.fi’” ohjaa käyttäjän Lapin ammattikorkeakoulun etusivulle. Ohjaus voidaan tehdä jollekin vahingolliselle sivulle, joka voi saattaa käyttäjän tietokoneen hyökkäysten kohteeksi. (Django Software Foundation 2014a; Django Software Foundation 2014c.)

Cross-Site Request Forgery(CSRF) on hyökkäys, jossa käytetään hyväksi käyttäjän istuntoa palvelussa. Kun käyttäjä on kirjautunut palveluun ja hän käyttää toista verkkosivua. Tämä toinen sivu sitten käyttäjän tietämättä tekee pyynnön siihen palveluun, jossa hän on kirjautuneena käyttäen hänen oikeuksiaan. Tämä on erityisesti silloin vahingollista, jos palvelun hyökkäyksen kohteena on vaikkapa verkkopankki. Silloinhan hyökkääjä voisi vaikka siirtää

rahaa toisen tililtä omalle tililleen. Tosin näin kriittisissä palveluissa on todennäköisesti kunnon suojaukset tämän tyyppisiä hyökkäyksiä vastaan. Django tarjoaa suojan useimmille CSRF-tyyppisille hyökkäyksille. (Holovaty, A – Kaplan-Moss, J 2014b.)

7 VAATIMUSMÄÄRITTELY JA SUUNNITTELU

Hallintaohjelma on yhteydessä moniin järjestelmiin. Nämä järjestelmät ovat MySQL, PostgreSQL, Subversion, SSH, SFTP, Disk Quota sekä palvelimen käyttöjärjestelmä. Näitä järjestelmiä ajetaan samalla palvelimella kuin hallintaohjelmaa. Hallintaohjelmassa näitä järjestelmiä kutsutaan palveluiksi. Hallintaohjelma suorittaa komentoja palvelimelle ja siinä oleville palveluille, joita halutaan hallinnoidan tämän ohjelman kautta. Hallinnoitavia tietokantajärjestelmiä ovat PostgreSQL sekä MySQL. Näihin järjestelmiin hallintaohjelma luo tietokantoja sekä käyttäjätunnuksia. Ohjelmalla voidaan myös poistaa tai lukita tietokantoja sekä käyttäjätunnuksia. Käyttäjät ovat etäyhteydessä omaan kotihakemistoonsa SSH:lla . Tiedostojen siirtäminen palvelimen ja käyttäjän koneen välillä tapahtuu SFTP-protokollaa käyttäen. Disk Quota:lla rajoitetaan käyttäjän kotihakemiston kokoa. Hallintaohjelmalla hallinnoidaan myös Subversion-versionhallintajärjestelmää. Tähän järjestelmään hallintaohjelma luo versioarkistoja sekä antaa käyttäjille oikeuksia versioarkistoon. Pääkäyttäjä valitsee, että mihin palveluihin käyttäjälle suodaan oikeus. Kaikkien tuskin täytyy voida käyttää kaikkia palveluita. Muilla käyttäjillä on ainoastaan oikeus resetoida salasana omiin palveluihinsa. Resetoinnin jälkeen käyttäjälle lähetetään sähköpostilla uusi salasana.

Jokaiselle luotavalle käyttäjälle tehdään käyttäjätunnus palvelimen käyttöjärjestelmään. Käyttäjälle generoidaan salasana, jonka hän voi vaihtaa passwd-komennolla. Tämä komento ei kuitenkaan muuta hallintaohjelman salasanaa käyttäjälle. Käyttäjälle luodaan myös kotihakemisto, jonka koko rajoitetaan käyttäjän tarpeiden mukaan. Tätä rajoitusta voidaan muuttaa myöhemmin. Käyttäjätunnus voidaan lukita ja poistaa hallintaohjelmaa käyttäen. Kun käyttäjätunnus poistetaan, arkistoidaan sekä poistetaan hänen kotihakemistonsa. Tietokantajärjestelmiin luodaan käyttäjälle oma tietokanta sekä käyttäjätunnus. Kirjautumistiedot ovat samat kuin käyttöjärjestelmän käyttäjätunnuksella. Käyttäjälle annetaan kaikki oikeudet hänelle luotuun tietokantaan. Käyttäjän tietokannan nimeksi annetaan hänen käyttäjätunnuksensa. Kun käyttäjältä poistetaan tietokantapalvelu, otetaan hänen tietokannastaan SQL-skripti ja tallennetaan se hänen kotihakemistoonsa. Sitten käyttäjän tietokanta ja käyt-

täjätunnus poistetaan. Subversion:n luodaan käyttäjälle oma versioarkisto. Kun käyttäjältä poistetaan tämä palvelu, poistetaan hänen versioarkistonsa. Versioarkistosta ei tehdä varmuuskopiota. SSH- ja SFTP-palveluissa käyttäjä lisätään käyttöjärjestelmän käyttäjäryhmään, jolla on oikeus käyttää näitä palveluita.Tämä palvelu on käyttäjällä aina päällä. Jokaisen palvelun kohdalla lukitseminen merkitsee vain käyttöoikeuden eväämistä. Mitään tietoja ei poisteta. Käyttäjän sähköpostiin lähetetään informaatiomateriaali hänelle suotujen palvelujen mukaan. Informaatiomateriaali kertoo muun muassa kuinka näihin palveluihin ollaan yhteydessä. Myös kirjautumistiedot lähetetään käyttäjälle.

Hallintaohjelman käyttöliittymä on melko yksinkertainen. Aloitusnäkymässä on käyttäjätaulukko. Käyttäjätaulukon jokainen rivi vastaa yhtä käyttäjää ja sen sarakkeissa on palveluiden hallintaan liittyvät toiminnallisuudet. Toisella sivulla on lomake käyttäjän luomiseen. Myös siinä voidaan lisätä palveluita luotavalle käyttäjälle. Nämä palvelut kuitenkin voidaan lisätä myös jälkikäteen. Hallintaohjelmassa on myös sivu, jossa voidaan aktivoida ja poistaa palveluita käytöstä. Lähtökohtainen periaate palveluiden suhteen on se, että ne täytyy vain asentaa. Hallintaohjelma hoitaa muut tehtävät palveluiden käyttöönottoon liittyen. Monien palveluiden kohdalla tämä merkitsee vain palvelutyypin lisäämistä tietokantaan. Disk Quota-palvelun käyttöönotto vaatii kuitenkin asennuksen ja palvelutyypin lisäämisen lisäksi muitakin toiminpiteitä. Merkittävä sivu on myöskin kirjautumissivu. Jos käyttäjä ei ole kirjautunut hallintaohjelmaan, ohjataan hänet tälle sivulle.

```
if postgresql:
    doSomething()
    if mysql:
        doSomething()
```

Kuvio 9. Käyttäjän luominen hallintaohjelmassa.

Hallintaohjelman modulaarisuus ei aluksi ollut vaatimusmäärittelyssä. Tehdessäni ohjelmakoodia ymmärsin kuitenkin tarpeen kapseloida palvelut omiksi luokiksi. Muistan käyttäneeni aluksi sisäkkäisiä if-lauseita käyttäjän luomiseksi ja palveluiden lisäämiseksi käyttäjän luonnin yhteydessä. Tämä on ongelmallinen tapa toimia. Tällöin palveluiden lisääminen on kytköksissä toisiinsa. Aiemmin palveluiden lisääminen tapahtui kuvion 9 mukaisesti. Jos siis käyttäjälle ei haluta PostgreSQL-palvelua, jää myös MySQL-palvelu lisäämättä. Kuvio 9:a ei ole suoraan otettu lähdekoodista. Jokainen palvelu on hallintaohjelmassa oma luokkansa, joka perii ServiceBase-luokan. ServiceBase-luokassa on määritelty funktiot, jotka palveluiden luokkien täytyy toteuttaa. Näillä funktioilla muun muassa lisätään palvelu käyttäjälle. Myös lukituksen poistamiseen sekä palvelun salasanan

resetoimiseen on omat funktionsa. Toteutus voi olla tyhjä, kuten SSH-palveluluokan salasanan resetointi, sillä SSH käyttää käyttöjärjestelmän käyttäjätunnusta. ServiceBase-luokassa on myös funktiot, joilla voidaan kysyä palvelun tilasta käyttäjään kohden. Näitä funktioita ei ole toteutettu palveluiden omissa luokissa. Jos funktion toteutus on sama useissa luokissa, ei ole järkevää kirjoittaa samaa funktiota jokaiseen luokkaan. Tämä vähentää tarvetta ohjelmakoodin siirtämiseen copy-paste-toimintoa käyttäen. Copypaste-toimintoa ei tulisi koskaan käyttää ohjelmakoodia tehtäessä, sillä jos ohjelmakoodissa on ohjelmointivirhe, pitää sama virhe korjata useassa kohdassa. Näihin luokkiin on tarkoitus sitoa mahdollisimman paljon palveluihin liittyviä toiminnallisuuksia, jotta palvelun lisäämiseksi pitäisi mahdollisesti luoda vain palveluluokka. Jos jokin palveluun liittyvä asia voidaan tehdä palveluluokan ulkopuolella kohtuullisen pienellä vaivalla, ei ole välttämätöntä sisällyttää sitä palveluluokkaan. Kuitenkin on pidettävä huoli, että tällaisia tapauksia ei ole monta.

Koska ohjelma ei aluksi ollut laajennettava, laitoin palveluiden lomakekentät forms.py-tiedostoon. Kenttien lisääminen tähän tiedostoon tuskin on kovinkaan suuri työ, mutta modularisuuden kannalta se on järkevä vaihtoehto. On yksi tiedosto vähemmän muokattavana. Sitten päätin laittaa kentät hajautus-

tauluun, johon tuli avaimeksi kentän tunniste ja avaimen arvoksi Djangon lomakekenttä-objekti. Kun päätin laittaa myös päivitysfunktiot tähän rakenteeseen, tein luokan sisältämään nämä ominaisuudet. Luokassa on attribuuttina myös palveluluokka, johon se on kytketty. Tämä on tehty sen vuoksi, että päivitysfunktiosta voidaan kutsua päivitettävän palveluluokan funktioita. Se on tarpeen palvelun päivittämiseksi. Päivitysfunktio oikeastaan valitsee ainoastaan palveluluokan funktion, jota kutsutaan. Tämä valinta perustuu useiden palveluiden kohdalla niiden edelliseen ja nykyiseen tilaan. Disk Quotan päivitysfunktiossa kutsutaan aina samaa metodia, koska niiden kenttien arvot ovat numeeriset.

Kun lomakkeen kentät on laitettu palveluluokan attribuuteiksi, pitää lomake koota näistä kentistä. Tulin huomaamaan, että lomakkeen kenttien dynaaminen lisääminen lomakkeeseen ei ollut helppoa. Lopulta löysin ratkaisun tähän ongelmaan. Poistin lomakkeesta kaikki palvelukohtaiset kentät ja ainoastaan käyttäjänimi-kenttä jäi lomakkeeseen. Kentät annetaan konstruktorille parametrinä. Nämä kentät sitten poimitaan konstruktorissa pois ja kutsutaan yliluokan konstruktoria. Näin saadaan käyttäjän arvot lisätyiksi lomakkeen kenttien arvoiksi. Sen jälkeen iteroidaan läpi kentät ja asetetaan lomakkeeseen palveluluokista saadut kentät. Tämä kenttien lisääminen täytyy toteuttaa joka kerta kun lomake alustetaan. Eli myös lomakkeen käsittelijä-funktiolle täytyy tuoda nämä kentät. Tämän vuoksi muodostui järkeväksi ratkaisuksi laittaa kenttien hakeminen omaan funktioonsa. Lomakkeen käsittelijäfunktio käy jokaisen lomakkeen kentän läpi ja yrittää päivittää kentän arvoa jokaiseen palveluun. Jos kenttä on palvelun oma kenttä, niin palveluluokan päivitysfunktio kutsuu kentän päivitysfunktiota kentän päivittämiseksi. Tämän jälkeen palveluluokan päivitysfunktio palauttaa True-arvon, jotta lomakkeen käsittelijäfunktio ei enää yrittäisi kokeilla kentän arvon päivittämistä muihin palveluihin. Jos palvelussa ei ole kenttää, päivitysfunktio palauttaa False-arvon.

Toteutin ensin palvelun päivittämisen lisäämällä jokaisen palvelun lomakkeen käsittelijäfunktioon. Tämä on modulaarisuuden kannalta myöskin ongelmallinen. Aina palvelun lisäämisen yhteydessä täytyisi päivittää myös tätä osaa

ohjelmasta. Edellä mainittu ratkaisu tarjoaa dynaamisen tavan päivittää palvelut käyttäjälle. Palveluissa voi olla erityyppisiä arvoja, joita päivitetään käyttäjälle. Tämä toki riippuu siitä, mitä sisällytetään palveluluokkaan. Niimpä päätin liittää jokaiselle kentälle palveluluokkaan oman päivitysfunktionsa. Kun kentän arvoa päivitetään, niin palveluluokka kutsuu kentän omaa päivitysfuntiota antaen sille parametrinä palveluluokan objektin. Koska palveluluokka annetaan objektina, ei tarvitse luoda päivitysfunktiota jokaiselle palvelulle, vaan voidaan kutsua saadun palveluobjektin metodeja. Palveluluokilla on kuitenkin oltava yhtenäinen rajapinta, jotta niillä voisi olla sama päivitysfunktio. Eli niillä pitää olla samat metodit. Myös metodien toteutus on oltava yhtenäinen. Esimerkiksi palvelun poistamisfunktion on poistettava palvelu käytöstä, muutoin tietokantaan voi tallentua väärä tila palvelulle.

Palveluluokkien suunnittelussa on periaatteenani ollut laittaa mahdollisimman paljon palveluihin liittyvää toteusta ServiceBase-luokkaan. Jos jollain metodilla on sama toteutus monessa palveluluokassa, on tarpeetonta laittaa sama metodi jokaiseen palveluluokkaan. Tämä metodi kannattaa laittaa ServiceBase-luokan metodiksi. Sitten laittaa tälle metodille oman toteutuksen niihin palveluluokkiin, jossa ei voida käyttää tätä yliluokan toteutusta. Tästä hyvä esimerkki on palveluluokan päivitysmetodi. Tämä metodi on määritelty ServiceBase-luokassa. Jokaisella palveluluokalla on sama toteutus tälle metodille. Toinen esimerkki on getServiceOfUser-metodi. Tämä metodi palauttaa palvelun kenttää vastaavan arvon annetulta käyttäjältä. Useat palveluluokat perivät tämän toteutuksen. Näitä palveluita ovat MySQL, Subversion, SSH/SHTP ja PostgreSQL. Näillä palveluilla on vain yksi kenttä. Tämän kentän arvo kertoo palvelun tilan käyttäjää kohden. Hallintaohjelmaan voi tulla useita uusia palveluita, joiden toteutuksia voidaan yhdistää. Tällöin kannattaa luoda uusi luokka, joka on ServiceBase-luokan aliluokka. Tämä luokka tulee laittaa näiden uusien palveluiden palveluluokkien yliluokaksi ja sisällyttää siihen palveluluokkien metodit, jossa toteutus on yhteneväinen.

Opinnäytetyö vaati melko paljon poikkeuksen käsittelyä. Tietokantaan on tallennettu palveluiden tila käyttäjiä kohden. Näiden tietojen pitää kertoa palve-

luiden todellinen tila käyttäjää kohden. Jos vaikka MySQL-palvelun tila on käyttäjälle merkitty aktiiviseksi, pitää käyttäjällä olla oma tietokanta, tietokantakäyttäjä sekä oikeudet hänen tietokantaansa. Olen kuitenkin sisällyttänyt tarkistuksia siltä varalta, että palvelun tila olisikin väärä tietokannassa. Esimerkiksi MySQL-palvelun lukitsemismetodi tarkistaa ensin onko käyttäjällä oikeudet tietokantaansa. Jos oikeuksia ei ole, ei yritetä poistaa oikeuksia. Tämä voi nimittäin palauttaa virheilmoituksen. Sen sijaan kerrotaan hallintaohjelman käyttäjälle, että käyttäjän MySQL-palvelu on jo lukittu. Myös oikea tila tallennetaan tietokantaan.

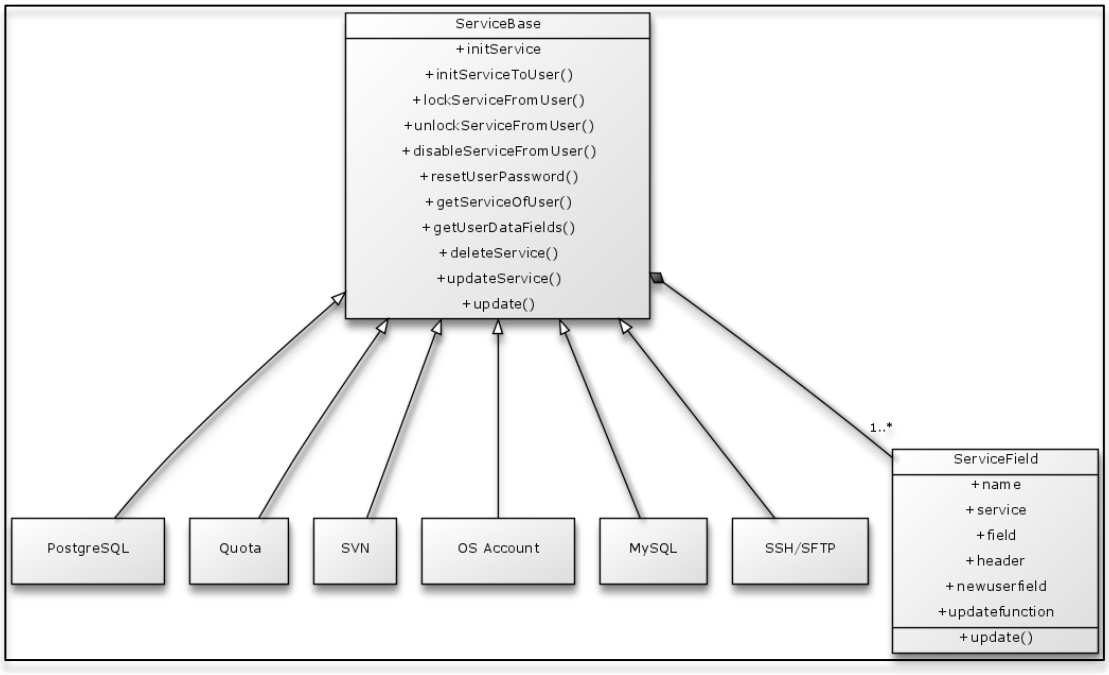
Jotta hallintaohjelma todella olisi modulaarinen, myös käyttöliittymän on oltava dynaaminen. Dynaamisen käyttöliittymän tekeminen ei tuntunut millään onnistuvan. Meinasin jopa luovuttaa sen tekemisen. Ongelmana oli saada otsikot vastaamaan kenttiä. Eli sarakkeen otsikot saattoivat johtaa harhaan. Tämä johtui varmaan siitä, että olin laittanut otsikot kenttien arvoja vastaavaan järjestykseen. Näiden järjestys oli kuitenkin eri kuin kenttien järjestys. En tosin ymmärrä miksi näiden järjestys on eri, sillä ne ovat samalla tavalla haettu. Laitoin otsikot kenttiä vastaavaan järjestykseen ja nyt kentät ja otsikot vastaavat toisiaan. Otsikkoja varten lisäsin otsikko-attribuutin kenttäluokalle. Tämä otsikot kerätään ja renderöidään HTML-sivulle.

Aluksi tarkoituksenani oli sisällyttää kaikki palvelut samaan luokkaan, mutta tiedostosta alkoi tulla aivan liian iso. Rivejä on tullut jo yli 600 tuohon tiedostoon. Tuon tiedoston muokkaaminen palveluiden lisäämistä varten ei todellakaan ole hauskaa. Niinpä päätin tehdä hakemiston, johon kokoan nämä palveluluokat. Nämä palveluluokat ovat kukin omana tiedostonaan hakemistossa. Myös ServiceBase-luokka on laitettu omaan tiedostoonsa. Tämän hakemiston __init__.py-tiedostossa on metodi, joka hakee kaikki palveluluokat. Palveluluokat kuitenkin sisällytettiin aluksi import-komennolla tähän tiedostoon, kunnes löysin dynaamisen tavan palveluluokkien sisällyttämiseen. Näin myös tarpeelliseksi laittaa kaikki kenttiin liitettävät päivitysfuntiot omaan tiedostoonsa. Aiemmin kaikki olivat samassa services.py-tiedostossa. Tämä

hajauttaminen ainakin omasta mielestäni selkeyttää ja helpottaa palveluiden muokkaamista ja lisäämistä.

8 TOTEUTUS

8.1 Palveluluokat



Kuvio 10. Palveluluokkien rakenne.

Kuvio 10 havainnollistaa palveluluokkien rakennetta. Hallintaohjelman modulaariseksi tekeminen vaati paljon suunnittelua. Tämä suunnittelu kohdistui erityisesti palveluluokkiin. Tavoitteenani oli, että uuden palvelun luomisessa täytyisi tehdä ainoastaan palveluluokka. Tämä päämäärä toteutui melko hyvin. Ei kuitenkaan riitä, että palveluiden toiminnallisuus ainoastaan sijaitsee palveluluokassa. Palveluluokan rakenteen selkeys on myös tärkeä, jotta palveluiden lisääminen olisi mielekästä. Koska useimmissa palveluissa niiden hallinnointi tarkoittaa palvelun lisäämistä, lukitsemista sekä poistamista käyttäjää kohden, on ServiceBase-luokkaan laitettu abstraktit metodit näille operaatioille. Näille metodeille siis täytyy olla palveluluokissa oma toteutus. Myös salasanan vaihtamiseen on laitettu abstrakti metodi, koska sen toteutus vaihtelee eri palveluissa. Olen laittanut nämä metodit abstrakteiksi, koska useisiin palveluihin tarvitaan toteutus näille metodeille. Olen käyttänyt abstrakteja metodeja myös yhtenäisen rajapinnan saamiseksi palveluluokille. Saman niminen metodi tekee saman asian joka palvelulle. Tämä helpottaa palvelu-

luokkien ja muun ohjelman yhteydenpitoa. Jos jokaisella palveluluokalla olisi erinimiset metodit vaikka salasanan nollaamiseen, täytyisi salasanan vaihtamisen yhteydessä ehtolauseilla etsiä oikea metodi.

Koska rajapinta on yhteneväinen, voidaan lomakkeen käsittelijäfunktiossa salasanan vaihtaminen tehdä kutsumalla jokaisen palveluluokan salasanan vaihto-metodia. Vaikka toteutus palveluluokassa olisi tyhjä, tämä toimintatapa minimoi tarpeen muokata lomakkeen käsittelijäfunktiota. Tässä opinnäytetyössä olen toiminut sillä periaatteella, että lomakkeiden käsittelijäfunktioissa ei olisi tarkistuksia, vaan ne tehtäisiin palveluluokissa. Hyvä esimerkki tästä on palveluluokan update-metodi. Update-metodi ottaa vastaan parametrinä käyttäjän, päivitettävän kentän nimen sekä sitä vastaavan arvon. Tähän metodiin on mahdollista myös antaa avainsana-parametri ilmaisemaan, että tarkoituksena on luoda uusi käyttäjä. Jos palveluluokassa ei ole parametrinä saadun kentän nimeä vastaavaa kenttää, lopetetaan metodin suorittaminen.

Palvelun tilan hakemisen sujuvoittamiseksi tein Djangon malliluokilla tietokannan kertomaan palvelun tilasta käyttäjää kohden. Aiemmin hain käyttäjät etusivun käyttäjätaulukkoon suoraan /etc/passwd-tiedostosta awk-komentoa käyttäen. Jos pelkästään käyttäjiä pitäisi listata, niin silloin tämä ehkä voisi olla sopiva ratkaisu. Hallintaohjelman kuitenkin pitää kertoa palveluiden tilasta käyttäjää kohden. Tämä kyllä voitaisiin tehdä palveluilta kyselemällä. Tämä kuitenkin todennäköisesti hidastaisi hallintaohjelman käyttöä. Hallintaohjelmassa voi olla käyttäjiä, jonka palveluiden tilaa muutetaan todella harvoin. Miksi sitten tällaisen käyttäjän palveluiden tilaa pitäisi aina kysellä, kun avataan hallintaohjelman etusivu? Se tuo tarpeetonta työtä. Kuitenkin tämän toimintatavan etuna on tarkan tilan saaminen palveluiden tilasta. Palveluiden tarkan tilan saaminen on mahdollista myös silloin, kun käytetään tietokantaa palveluiden tilan tallentamiseen. Tällöin kuitenkin täytyy tehdä paljon tarkistuksia, jotta tallentuva tila on varmasti oikea tila. Esimerkiksi tietokantapalvelun lisäämisessä täytyy tarkastaa, että tietokannan sekä tietokantajärjestelmän käyttäjätunnuksen luominen onnistuu. Muutoin tila ei saa vaihtua aktiiviseksi. Tietokantaa luodessa täytyy varmistua siitä, että samalla nimellä ole-

vaa tietokantaa ei ole. Jos sellainen löytyy, täytyy hallintaohjelman käyttäjää informoida tästä. Virheilmoituksen jälkeen palvelun päivittäminen lopetetaan ja tila jätetään ennalleen. Tällä tavalla olen päättänyt toimia myös muissa vastaavanlaisissa poikkeustilanteissa. Olen lisännyt hallintaohjelmaan myös tarkistuksia siltä varalta, että tietokantaan olisi tallentunut väärää tietoa. Nämä tarkistukset kuitenkin tapahtuvat vain palveluiden tilaa päivitettäessä.

8.2 Hallintaohjelman ja palvelimen tietoturva

Hallintaohjelman tietoturvan suunnittelussa täytyi perehtyä myös Linux-käyttöjärjestelmän tietoturvaan. Perehtyminen ei kuitenkaan ollut niin syvällistä, että voisi itseään tietoturva-asiantuntijaksi nimittää. Tällä tietoturvan osa-alueella keskeisimpänä suunnittelun kohteena oli hallintaohjelman ajajakäyttäjän oikeuksien rajoittaminen. Hallintaohjelmaa on määrä ajaa jatkuvasti. Koska hallintaohjelma tarvitsee poikkeuksellisen paljon oikeuksia, on hallintaohjelman käyttäjällä oltava mahdollisimman pienet oikeudet.

Hallintaohjelmaan ei tarvinut tehdä paljon koodia tietoturvan lisäämiseksi. Djangon tietoturvaominaisuudet olivat helposti käyttöön otettavissa. Luotavan käyttäjän käyttäjätunnukselle tein oman validoinnin. Hallintaohjelma ei salaa tietoliikennettä, koska sitä ei nähty tarpeelliseksi.

8.3 Moduulin tekeminen hallintaohjelmaan

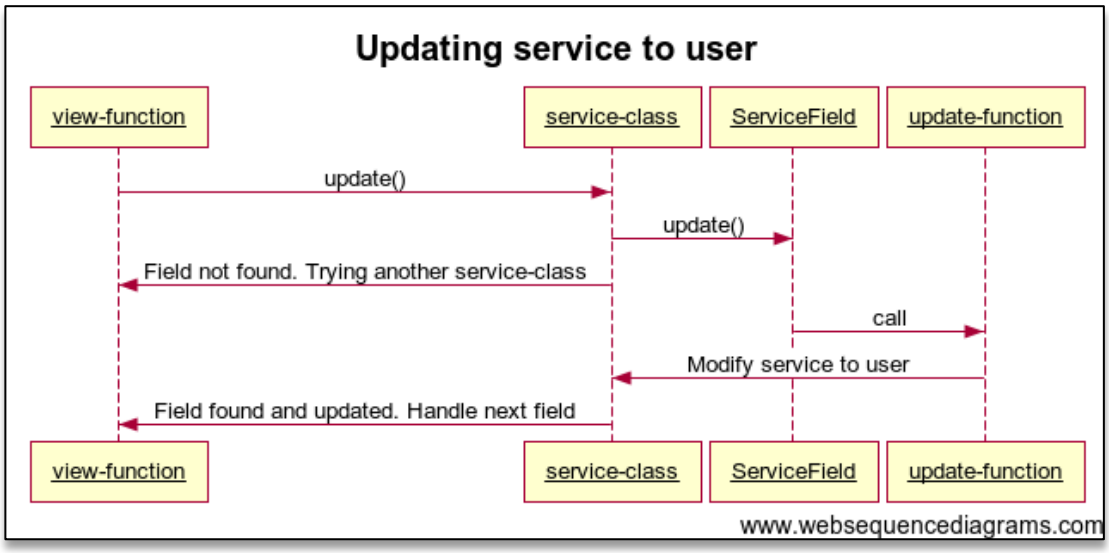
Hallintaohjelman moduulien tekeminen on tehty mahdollisimman yksinkertaiseksi. Jätin tarkoituksellisesti PostgreSQL-palvelun moduulin tekemisen myöhemmäksi, jotta voin kokeilla moduulien tekemistä hallintaohjelmaan. Palveluluokkien lopullinen rakenne ei nimittäin vielä ollut tiedossa. Paljon vaivannäön jälkeen palvelun luominen voi onnistua ainoastaan palveluluokan tekemisellä. Aiemmin palveluluokkahakemiston __init__.py-tiedostoon täytyi lisätä import-komennolla palveluluokat. Sain muokattua palveluiden hakufunktiota niin, ettei import-komentoja tarvitse käyttää. Jälleen on yksi tiedosto vähemmän muokattavana.

PostgreSQL-palvelun lisääminen alkoi asennuksien tekemisellä, sillä hallintaohjelma ei asenna ohjelmia. Asennuksen jälkeen hallintaohjelma huolehtii käyttöönottoon liittyvistä tehtävistä. Harvojen palveluiden kohdalla on tarvetta

tällaiseen. Aluksi kopioin MySQL-palvelun luokan ja liitin sen postgresql-moduuliin. Tämä ei ole suositeltava tapa ohjelmakoodin tekemiseen, sillä tällöin mahdolliset ohjelmointivirheet täytyy korjata useaan kohtaan. Koska MySQL:n ja PostgreSQL:n toteutus on melko samanlainen päädyin tähän ratkaisuun. Kopioitua koodia ei tarvitse kovinkaan muokata. MySQL:n päivitysfunkio käy myös PostgreSQL-palvelulle. Myös malliluokka tietokantaa varten on sama kuin MySQL-palvelulla. Tietokantajärjestelmille olisi voinut tehdä yhteisen yliluokan, joka perisi ServiceBase-luokan. Tietokantajärjestelmiä tuskin kuitenkaan tulee niin paljon hallintaohjelmaan.

Tekemäni palveluluokat ovat olleet samanlaisia rakenteeltaan. Niillä on kenttiä, joiden avulla muutetaan palvelun tilaa käyttäjää kohden. Jos halutaan lisätä mahdollisuus luoda tietokantoja sekä asettaa käyttäjille eri oikeudet eri tietokantoihin, täytyy palveluluokat tehdä erillä tavalla. On toki mahdollista laittaa jokainen tietokanta omaan palveluluokkaansa, mutta mieluummin sen tekee dynaamisesti. MySQL-palveluluokassa voisi olla kenttänä painike, jota painamalla avautuu sivu tietokantojen oikeuksien asettamiseen. Tällöin kannattaa lisätä kenttäluokkaan template-sivu, jota käytetään tietokannan oikeuksien muuttamiseksi. Tietokantojen oikeuksien asettamista varten täytyy tehdä oma päivitysfunktionsa. ServiceBase-luokan update-metodia ei tarvitse muuttaa. Malliluokka täytyy kuitenkin tehdä. Palveluluokkaan täytyy tehdä metodi tietokantojen hakemiseen ja tietokannan lisäämiseen.

8.4 Palveluiden päivittäminen



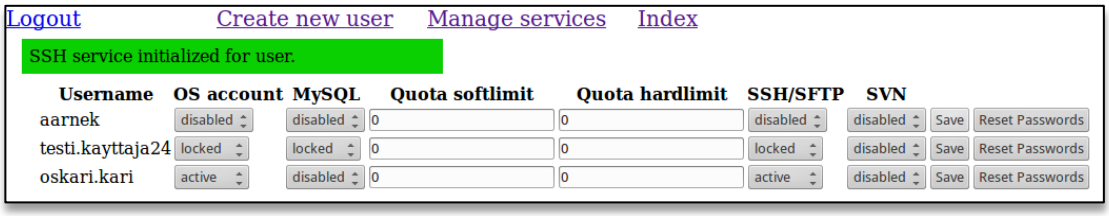
Kuvio 11. Palveluiden päivittäminen.

Palveluiden päivittäminen ei ole yksinkertainen prosessi. Koska hallintaohjelma on modulaarinen, niin palveluiden päivittäminen on myös tehtävä modulaarisesti. Päivittämisen helpottamiseksi olen tehnyt ServiceField-luokan. Tämä sisältää palveluluokan yhden kentän tiedot ja päivitysfunktion. Päivitysfunktio on luokan attribuutti, jota luokan update-metodi kutsuu. Update-metodi kutsuu päivitysfunktiota sekä välittää sille tiedon, onko kyseessä uusi käyttäjä vai ei. ServiceField luokan objekteista käytetään nimitystä palveluluokan kenttä.

Päivitysprosessi on kuvattu kuviossa 11. Päivitys alkaa siitä, kun lomakkeenkäsittelijäfunktio saa lomakkeen. Tässä lomakkeessa on palveluiden tilat. Funktio hakee sitten palveluluokat. Tämän jälkeen jokaista kenttää yritetään päivittää jokaiseen palveluluokkaan. Tämä päivitys tehdään kutsumalla palveluluokan update-metodia. Jos vastaavaa kenttää ei löydy palveluluokasta, palveluluokan update-metodi palauttaa False-arvon. Tällöin kenttää yritetään päivittää toiseen palveluluokkaan. Jos palveluluokasta löytyy vastaava kenttä, jatketaan päivitettävän kentän reititystä. Päivityksen jälkeen palautetaan True-arvo lomakkeen käsittelijäfunktiolle, jotta se tietää siirtyä seuraavan kentän päivittämiseen. Kun oikea palveluluokka on löytynyt, siirretään lomakkeelta saatu kentän arvo sitä vastaavan kentän päivitysfunktiolle sekä mahdollisesti kentän päivitystä edeltävä arvo. Kun palvelun päivittäminen on siir-

tynyt palveluluokan kentälle, kutsutaan kentän päivitysfunktiota. Päivitysfunktio kutsuu palveluluokan funktioita palvelun päivittämiseksi. Nämä funktiot tiedottavat käyttäjää päivityksen onnistumisesta sekä epäonnistumisesta.

8.5 Käyttöliittymä



Kuvio 12. Hallintaohjelman päänäkymä.

Hallintaohjelman käyttöliittymä on melko yksinkertainen. Kuvion 12 kuva on otettu hallintaohjelman päänäkymästä. Näkymiä on tosin enemmänkin, mutta niiden rakenteessa ei ole niin paljon kerrottavaa. Pääsivun kautta hallinnoidaan jo luotujen käyttäjien palveluita. Kun käyttäjän palvelua on muokattu painetaan save-painiketta käyttäjää vastaavalta riviltä. Ohjelma mahdollistaa myös salasanan vaihtamisen kaikille käyttäjä palveluille. Uudeksi salasanaksi tulee generoitu salasana joka lähetetään sähköpostilla käyttäjälle.

Palveluiden päivittämisen onnistumisesta kerrotaan pääkäyttäjälle. Kuviossa 12 on esillä esimerkkitapaus. Pääkäyttäjälle tiedotetaan onnistuneesta SSH/SFTP-palvelun käyttöönotosta käyttäjälle. Myös epäonnistumisesta kerrotaan pääkäyttäjälle. On tärkeä, että pääkäyttäjä saa tiedon päivityksen onnistumisesta. Silloin ei tarvitse komentorivillä tarkistaa päivityksen onnistumista. Viestin taustaväri kertoo päivityksen onnistumisesta. Pääkäyttäjä heti näkee onko päivitys epäonnistunut. Väreillä viestiminen on mielestäni hyvä asia. Silloin tulee selkeästi esille päivityksen tila.

9 POHDINTA

Hallintaohjelman tekeminen modulaariseksi onnistui yllättävän hyvin, sillä palvelu voidaan lisätä palveluluokan tekemisellä. Palvelun tyyppi kuitenkin ratkaisee, että joudutaanko tekemään palveluluokan luomisen lisäksi muuta työtä. Jos luotava palvelu kovin paljon eroaa jo luoduista palveluista, voi hallintaohjelmaa joutua paljonkin muokkaamaan. Tarkoituksenani on kuitenkin ollut tehdä helpoksi myös tällaisten palveluiden lisääminen. Hallintaohjelma on toiminnallisuuksiltaan melko suppea. Hallintaohjelmassa on paljon vielä kehittämisen varaa. Tietokantajärjestelmille olisi hyvä olla omat näkymät, joissa voidaan luoda tietokantoja sekä käyttäjätunnuksia tietokantajärjestelmiin. Eri tietokantajärjestelmille ei välttämättä tarvitsisi olla erilliset näkymät, vaan niiden hallintatoiminnot voisivat olla samassa näkymässä. Näkymiä voisi olla kaksi. Toisessa näkymässä täytyisi valita käyttäjä ja toisessa käyttäjä olisi jo valittu. Tämä olisi hyödyllinen toimintatapa, koska silloin voidaan valita päänäkymästä käyttäjä ja päästä suoraan hallinnoimaan tietokantajärjestelmiä käyttäjää kohden.

Hallintaohjelmassa olisi hyvä olla pakettimanageri, jonka avulla voitaisiin asentaa sekä päivittää ohjelmia. Myös terminaalin käyttö hallintaohjelman kautta olisi hyvä vaihtoehto. Tällöin ei tarvitsisi erikseen avata komentoriviä. Komentorivin lisääminen hallintaohjelmaan voisi kuitenkin olla niin riskialtista, että tietoliikenne pitäiisi salata. Komentojen lähettäminen selkokielisenä palvelimelle ei ole järkevä ratkaisu.

Hallintaohjelman tietoliikenteen salaaminen voisi olla järkevää. Hallintaohjelman tietoliikenteessä lähetetään käyttäjätunnuksia. Jos käyttäjä on laittanut helpon salasanan, voi käyttäjän käyttäjätili joutua kaapatuksi. Pahoissa aikeissa oleva henkilö voi tarkkailla palvelimen tietoliikennettä ja saada selville käyttäjätunnukset. Tämän jälkeen jokaisen käyttäjän salasana yritetään saada selville. Ei tarvita muuta kuin yksi laiska käyttäjä, niin murtautujalla on pääsy palvelimen käyttöjärjestelmään. Yksi vaihtoehto tämän välttämiseksi olisi salasanan vaihtamisen evääminen tavallisilta käyttäjiltä. Tällöin käyttäjä joutuu kuitenkin pyytämään salasanan vaihtamista pääkäyttäjältä. Tämä ei ole välttämättä pääkäyttäjän kannalta hyvä asia. Vaihtoehtona olisi myös kir-

jautuminen pankkitunnuksilla hallintaohjelmaan. Hallintaohjelmassa käyttäjä sitten vaihtaisi salasanansa.

Hallintaohjelman päänäkymästä olisi hyvä nähdä käyttäjien levytilan käyttö. Jos käyttäjä ylittää softlimit rajan, voisi hallintaohjelma varoittaa käyttäjää sähköpostilla. Levytilan käyttö voitaisiin ilmoittaa paneelin avulla. Paneeli olisi yhtä pitkä jokaisella käyttäjällä. Sitten piirretään paneeliin käytetty ja käyttämätön tila siinä suhteessa, jossa ne ovat. Jos levytilaa on käytetty puolet käyttäjälle suodusta tilasta, piirrettäisiin vasemmalta oikealle puolet paneelin pituudesta tietyllä värillä ja loput toisella värillä. Tämänkaltaisia widget:ja ovat akun varaustason ilmaisevat widgetit.

LÄHTEET

Kuutti, W 2011. Linux käsikirja. Saarijärvi: Saarijärven Offset Oy

Wallen, J 2010. Linux 101: Introduction to sudo. Osoitteessa https://www.linux.com/learn/tutorials/306766:linux-101-introduction-to-sudo. 12.5.2010

Computer Hope 2014. Linux and Unix chmod command. Osoitteessa http://www.computerhope.com/unix/uchmod.htm 22.2.2014

Holovaty, A – Kaplan-Moss, J 2014a. Chapter 5: Models. Osoitteessa http://www.djangobook.com/en/2.0/chapter05.html. 15.3.2014

Lott, S 2014. Why Python is So Cool. Osoitteessa http://www.itmaybeahack.com/homepage/books/nonprog/html/p01_getting_started/p01_c05_cool.html. 6.3.2014

Python Software Foundation 2014a. Python Software Foundation. Osoitteessa http://www.python.org/psf/. 6.3.2014

Python Software Foundation 2014b. Python Standard Library. Osoitteessa http://docs.python.org/2/library/. 20.3.2014

Python Software Foundation 2014c. 4.More Control Flow Tools. Osoitteessa http://docs.python.org/2/tutorial/controlflow.html#defining-functions. 12.3.2014

Python Software Foundation 2014d. 7.Combound statements. Osoitteessa https://docs.python.org/2/reference/compound_stmts.html. 14.5.2014

Raive, S 2013. 18 Open Source/Commercial Control Panels to Manage Linux Servers. Osoitteessa http://www.tecmint.com/web-control-panels-to-managelinux-servers/. 27.9.2013

Django Software Foundation 2014a. Security in Django. Osoitteessa https://docs.djangoproject.com/en/dev/topics/security/. 20.3.2014

Holovaty, A – Kaplan-Moss, J 2014b. django.contrib. Osoitteessa http://www.djangobook.com/en/2.0/chapter16.html. 20.3.2014

Django Software Foundation 2014b Built-in template tags and filters. Osoitteessa https://docs.djangoproject.com/en/dev/ref/templates/builtins/. 20.3.2014

Django Software Foundation 2014c Making queries. Osoitteessa https://docs.djangoproject.com/en/dev/topics/db/queries/. 20.3.2014

NixCraft 2012. What is Umask and How To Setup Default umask Under Linux? Osoitteessa http://www.cyberciti.biz/tips/understanding-linux-unixumask-value-usage.html. 25.1.2012

Arch Linux 2014a. fstab. Osoitteessa https://wiki.archlinux.org/index.php/fstab. 6.2.2014

Arch Linux 2014b. Persistent block device naming. Osoitteessa https://wiki.archlinux.org/index.php/Persistent_block_device_naming. 22.2.2014

Web Hosting Hub 2014. Introduction to Control Panels. Osoitteessa http://www.webhostinghub.com/web-hosting-guide/introduction-to-controlpanels/. 1.4.2014

Ippolito, G 2014. Linux File System Quotas. Osoitteeessa http://www.yolinux.com/TUTORIALS/LinuxTutorialQuotas.html 3.4.2014

Django Software Foundation 2014d. Working with forms. Osoitteessa https://docs.djangoproject.com/en/1.5/topics/forms/. 8.4.2014

W3schools 2014. SQL Data Types for MS Access, MySQL, and SQL Server. Osoitteessa http://www.w3schools.com/sql/sql_datatypes.asp. 8.4.2014

Django Software Foundation 2014e. Form fields. Osoitteessa https://docs.djangoproject.com/en/1.5/ref/forms/fields/. 8.4.2014

van Dooren, R 2003. What is quota? Osoitteessa http://www.tldp.org/HOWTO/Quota-1.html. 9.8.2003

Buse, J 2013. Intro to Inodes. Osoitteessa http://www.linux.org/threads/introto-inodes.4130/. 9.7.2013

Morgan, J 2013. What Is MVC? Osoitteessa http://www.jeremymorgan.com/blog/programming/what-is-mvc/. 10.4.2013

Django Software Foundation 2014f. Class-based generic views. Osoitteessa https://docs.djangoproject.com/en/1.5/topics/class-based-views/generic-display/. 12.5.2014

Django Software Foundation 2014f. Class-based generic views. Osoitteessa https://docs.djangoproject.com/en/1.5/topics/class-based-views/generic-display/. 12.5.2014

Rouse, M 2014. web development framework (WDF). Osoitteessa http://searchcontentmanagement.techtarget.com/definition/web-development-framework-WDF. 14.5.2014

Apache Software Foundation 2014. Apache Subversion. Osoitteessa http://subversion.apache.org/. 14.5.2014

Higgins, D 2014. What is widget ? Osoitteessa http://whatis.techtarget.com/definition/widget. 14.5.2014