KARELIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Sami Oinonen
Mauri Mustonen

PUUTAVARAYRITYKSEN TUOTANNONOHJAUSJÄRJESTELMÄN SUUNNITTELU, VALMISTUS JA TESTAAMINEN

Opinnäytetyö
Toukokuu 2016

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Toukokuu 2016** **tietotekniikan koulutusohjelma** Karjalankatu 3 80200 JOENSUU (013) 260 600

Tekijät
Sami Oinonen, Mauri Mustonen

Nimeke
Puutavarayrityksen tuotannonohjausjärjestelmän suunnittelu, valmistus ja testaaminen

Toimeksiantaja
Asmeco

Tiivistelmä

Opinnäytetyön tarkoitus oli suunnitella ja toteuttaa web-pohjainen ohjelmisto Puupietari Oy nimiselle yritykselle. Puupietari Oy valmistaa mittatilaustyönä lauta- ja puutavaraa. Yritys tarvitsi räätälöidyn ohjelmiston korvaamaan ja helpottamaan heidän entistä toimintatapaansa. Lisäksi ohjelman tuli sisältää integraatio yrityksellä jo käytössä olevan Lemonsoft-toiminnanohjausjärjestelmän kanssa.

Työn tärkeitä aiheita oli valita kehityksessä käytettävät tekniikat ja työkalut, toteuttaa Lemonsoft ERP -integraatio, miettiä ohjelmiston turvallisuus- ja testaamisosiot, sekä suunnitella ohjelman yleinen rakenne. Työssä perehdytään myös paljon ohjelmallisten rajapintojen toteutukseen ja niiden turvallisuusratkaisuihin. Opinnäytetyöraportti keskittyy vahvasti serveripuolen toteutukseen mutta koskettaa myös vähän asiakaspuolen toteutusta. Syynä tähän oli ohjelmistoprojektin vastuualueen jaottelu kehittäjien kesken.

Ohjelma toteutettiin onnistuneesti sen rajaamisen mukaisesti ja on toimitettu nyt asiakkaalle. Asmeco tulee jatkamaan ohjelman kehitystä tämän jälkeen. Myös toteutukseen valitut tekniikat ja työkalut todettiin hyviksi ja Asmeco tulee käyttämään niitä myös yrityksen tulevissa asiakasprojekteissa.

Kieli Suomi	Sivuja 91 Liitteet 1

Asiasanat

Ohjelmistokehitys, Django, ERP, toiminnanohjausjärjestelmä, Lemonsoft, tuotannonohjaus

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2016
Degree Programme in Information Technology
Karjalankatu 3
80200 JOENSUU
FINLAND
+358 13 260 600

Authors
Sami Oinonen, Mauri Mustonen

Title
Planning, Developing and Testing of a Production Control System – Software for Timber Manufacturing Company

Commissioned by
Asmeco

Abstract

The object of this thesis was to plan and implement a web-based software for a company called Puupietari Ltd. Puupietari Ltd manufactures customized board and wood products. The company needed a custom-made software to replace and ease their old method of operations. In addition, the software had to include integration with Lemonsoft ERP which the company had already in use.

Important parts of the work were to choose the techniques and tools to be used in the development, to implement integration to Lemonsoft ERP, to consider the software's safety and testing aspects, and also to plan the software's main structure. The work also introduces a lot of the implementation of programmatic interfaces and their security solutions. The thesis focuses strongly on the server-side implementation but touches also the client-side. The reason for this was the allocation of responsibilities between developers.

The software was made successfully to the point it was intended to and it is now supplied to the customer. Asmeco will continue to further develop the program after this. Also, the chosen techniques and tools that were in use, were found to be useful and Asmeco will use them in the company's future customer projects.

Language	Pages 91
Finnish	Appendices 1

Keywords

software development, Django, ERP, enterprise resource planning, Lemonsoft, production management

Sisältö

Liitteet

Liite 1 Uhkamalli (Application Threat Modeling)

Lyhenteet

CI Continuous Integration on jatkuvan integraation ohjelmisto, mikä rakentaa ohjelmistosta aina uusimman version julkaistavaksi. Ohjelma voi samalla suorittaa paljon muita haluttuja toiminnallisuuksia rakennusprosessin yhteydessä, kuten testien suorittamista.

CSV Comma-separated values on tiedonsiirtoon tarkoitettu tiedostomuoto ja tiedon formatointiin käytetty menetelmä. Tiedosto on ihmisluettava ja tiedot erotetaan pilkulla tai muulla vastaavalla merkillä.

DRF Django Rest Framework on Django kehyksen lisäosa REST-rajapintojen nopeaan luomiseen. Suuren lisäosien tarjonnan takia, DRF on todella suosittu vaihtoehto rajapinnan toteuttamiseen Djangon kanssa.

ERP Enterprise Resource Planning ohjelmistot on tehty yritysten resurssien hallintaan. ERP-ohjelmistojen on usein tarkoitus sopia kaikenlaisten yritysten käyttöön ja sen takia ne ovat usein todella laajoja kokonaisuuksia.

Evästeet Evästeet/cookiet luodaan ensimmäisen kerran sivustolle mentäessä käyttäjän selaimeen ja niihin voidaan tallentaa muun muassa onko käyttäjä kirjautunut, mikä käyttäjä on kirjautuneena, verkkokaupoissa ostoskorin sisältö, sivuston selailun historiadata. yms.

HTTP Hypertext Transfer Protocol on tiedonsiirtoprotokolla jota selaimet ja web-palvelimet käyttävät tiedonsiirtoon. Esimerkiksi selain avaa TCP-yhteyden sen ja palvelimen välille. Tämän jälkeen selain lähettää pyynnön tiettyyn osoitteeseen, mihin palvelin vastaa sivuston sisällöllä kuten tarjoamalla HTML sivun käyttäjälle.

JSON JavaScript Object Notation on kevyt tiedonsiirtoformaatti. Formaatti on helposti ihmisen luettavissa ja muokattavissa. Tiedonsiirtoformaattia voidaan esimerkiksi käyttää ohjelmallisissa rajapinnoissa.

JWT JSON Web Token on avoimen lähdekoodin standardi ja se pohjautuu JSON-tiedonsiirtoformaattiin. JWT:tä käytetään web-sovelluksissa tiedon turvalliseen välittämiseen osapuolten välillä tokeneihin perustuvassa autentikoinnissa.

ORM Object-relational mapping on tekniikka jolla saadaan sopimaton tieto muutettua olio-pohjaiseksi tiedoksi. Tällainen tilanne on esimerkiksi tietokannan ja oliopohjaisen kielen välillä. ORM tekniikan avulla tietokantataulu saadaan muistuttamaan olio-pohjaista luokkaa, jonka jälkeen kielen omia ominaisuuksia voidaan käyttää tiedon käsittelyssä.

SPA Single-page application on web-sivu tai web-applikaatio, jossa kaikki sivun toiminta tapahtuu yhdellä web-sivulla ilman että selain lataa sivua uudestaan ohjelman ajon aikana. Sivulle lataa kaikki tarvittavat tiedostot kerralla tai tarvittavat tiedostot ladataan dynaamisesti ohjelman tarpeen mukaan. Tarkoituksena on antaa käyttäjälle tunne paikallisesta applikaation käytöstä, jos verrataan perinteiseen web-applikaatioon, jossa siirryttäessä sivu ladataan aina uudelleen. Tekniikka mahdollistaa paljon sulavamman käyttäjäkokemuksen sivujen siirtymisen välillä. Yleensä tässä toteutuksessa ohjelma kommunikoi serveripuolen kanssa koko ajan tiedon siirtämisen takia.

SQL Structured Query Language on standardoitu kieli, joka on suunniteltu relaatiotietokantojen hallintaan.

URI Uniform Resource Identifier on teksti joka osoittaa tiettyyn osoitteeseen. Esimerkiksi “http://esimerkki.fi/packages/”.

WSDL Web Services Description Language on XML-pohjainen tiedosto, jonka tarkoitus on kuvata saatavilla olevat rajapinnan tietotyypit ja funktiot toiselle ohjelmointikielelle. Yleensä toinen ohjelmointikieli generoi käytettävät ohjelmakomponentit WSDL-kuvaustiedoston pohjalta ohjelmistokehittäjille. Käytetään esimerkiksi SOAP-rajapintojen toteutuksessa.

1 Johdanto

Tämä dokumentti on tarkoitettu kuvaamaan opinnäytetyön prosessi ja toteutus, joka toteutettiin Karelia-Ammattikorkeakoulussa vuonna 2016. Tekijöinä työssä ovat Mauri Mustonen ja Sami Oinonen, ja työ on toteutettu osana Karelia Ammattikorkeakoulun tietotekniikan Insinööri -koulutusohjelmaa (ohjelmointitekniikka). Opinnäytetyö on toteutettu parityönä ja toimeksiantajana työlle toimii joensuulainen ohjelmistoalan yritys Asmeco. Toimeksiantaja on yksityinen toiminimi ja opinnäytetyön toinen tekijöistä, Mauri Mustonen, on yrityksen omistaja ja perustaja. Mauri Mustonen on siis opinnäytetyössä samaan aikaan sekä työntekijä että myös toimeksiantaja itselleen ja Sami Oinoselle. Projektin toteutuksessa on myös mukana toimeksiantajan yhteistyökumppani Elektroniikkapalvelu Parikka, josta yrityksen perustaja ja työntekijä Arto Parikka oli yksi ohjelmiston kolmesta tekijästä. Opinnäytetyön ohjaajana toimi Petri Laitinen Karelia Ammattikorkeakoulusta.

Opinnäytetyön aiheena oli toteuttaa räätälöity web-pohjainen ohjelmisto joensuulaiselle puu- ja lautatavaraa valmistavalle ja myyvälle Puupietari Oy:lle. Puupietari on Asmeco:n ja Elektroniikkapalvelu Parikan asiakas ja samalla ohjelmiston tilaaja. Työn tarkoituksena oli suunnitella ja toteuttaa ohjelmisto yrityksen puutavaran tuotannon tueksi. Ohjelmisto tilattiin, kun tarvittiin tehostaa työn toimintaa ja siirtää yrityksen tuotantoprosessi nykyaikaan. Puupietari Oy:llä oli ennestään vähäisessä käytössä toiminnanohjausjärjestelmä Lemonsoft-nimiseltä toimijalta Joensuusta. Osana työtä oli myös toteuttaa integraatio tähän toiminnanohjausjärjestelmään.

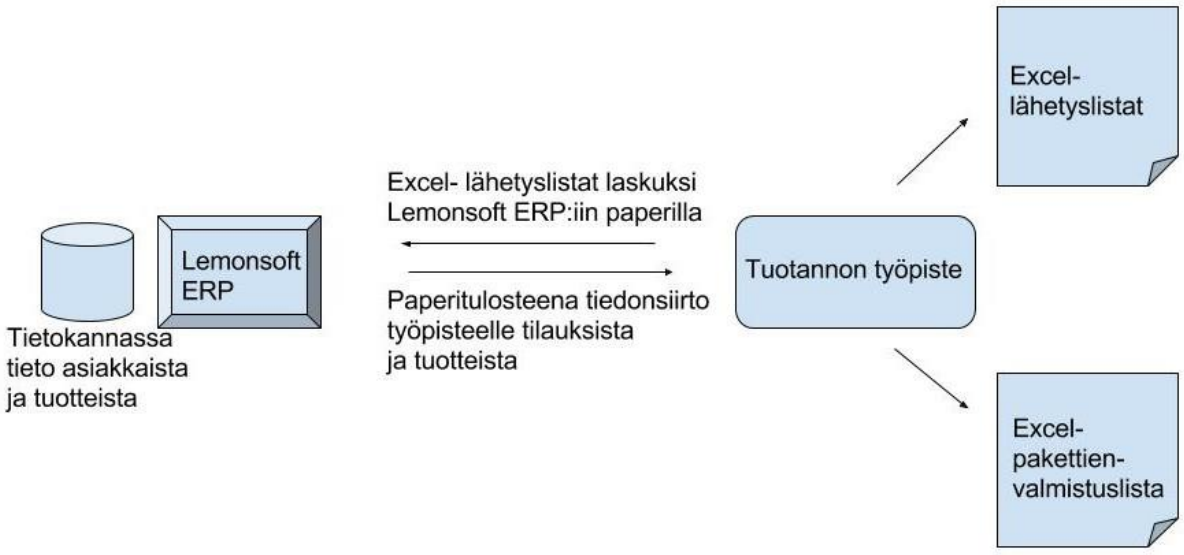
Arto Parikka oli vastuussa asiakaspuolesta, eli ohjelmiston käyttöliittymän suunnittelusta asiakkaan kanssa ja sen toteuttamisesta. Opinnäytetyön tekijöillä Oinosella ja Mustosella vastuualueenaan oli serveripuolen ohjelmointi ja suunnittelu. Tämän takia opinnäytetyöraportti keskittyy todella vahvasti pelkkään serveripuolen toteutukseen, eikä niinkään paljoa asiakaspuoleen. Työn laajuuden takia opinnäytetyö rajataan ensimmäiseen julkaistavaan versioon asiakkaalle ja ohjelmiston kehitystyötä jatkaa toimeksiantaja opinnäytetyön jälkeen.

Raportissa käydään ensin läpi tarvittava tietoperusta. Tietoperustan jälkeen lukijalla pitäisi olla perusymmärrys työssä käytettävistä termeistä ja vaadittavista asioista. Tämän jälkeen käydään läpi työn toteutukseen liittyvät vaiheet ja itse toteutettu ohjelma. Lopussa on pohdintaosio, missä mietitään ammatillista hyötyä ja kasvua työn aikana, ja sitä miten paljon uutta opinnäytetyö sen tekijöille opetti.

2 Projektin lähtökohdat

2.1.1 Tausta

Lähtötilanteessa Puupietari Oy:n tuotantoprosessi tukeutui Excel-taulukkolaskentaan ja Lemonsoft ERP -ohjelmistoon. ERPillä yritys hallitsi asiakkaitaan, tuotteitaan ja laskujaan, mutta sitä ei käytetty varastonhallintaan. Tuotannon eri vaiheet syötettiin excel-taulukkoon, joka sisälsi puupakettien tiedot, kuten niiden mitat ja niistä mitatut kosteusarvot, sekä puupakettien perusteella luodut lähetyslistat. Tuotantohallissa oli käytössä vain yksi tuotantopiste, jolle tieto tilauksista kuljetettiin toimistosta paperilla. Valmiiden tilausten lähetyslistoista tulostettiin ylimääräiset kappaleet, ne toimitettiin toimistoon ja syötettiin sitten ERP:iin. Alla on esitetty Puupietari Oy:n toimintatapa ennen opinnäytetyön lopettamista (kuva 1).



Kuva 1. Toimintatavan alkutilanne

Asiakas halusi laajentaa toimintaansa ja saada käyttöön lisää työpisteitä, mikä ei ollut vanhan toimintatavan kanssa mahdollista. Tämän esti Excelin sopimattomuus jaettuihin resursseihin, eli samaa taulukkoa ei voida käyttää kahdelta eri työpisteeltä. Tätä kuitenkin tarvittaisiin, koska paketit yksilöivä tunniste on numero, joka kasvaa aina yhdellä kun uusi paketti luodaan.

2.2 Tavoitteet

Ohjelmiston toteutus tapahtui vahvasti asiakkaan toiveiden mukaan, ja neuvottelussa sovittiin, että sovellus tehtäisiin web-pohjaisilla tekniikoilla. Ohjelmiston tuli olla yhteydessä jo olemassa olevaan Lemonsoft ERP:iin ja Excelin käyttö tuli korvata kokonaan. Koska projektin laajuus työn tiimin kokoon verrattaen oli suuri, jaoteltiin sen työvaiheet pääpiirteittään seuraaviin osioihin:

- tekniikoiden ja työkalujen valinta työn toteutukseen
- ohjelmiston suunnittelu ja toteutus asiakkaan tarpeiden mukaan
- Lemonsoft ERP -integraation suunnittelu ja toteutus
- PDF-tulosteiden generointi Excel-tulosteiden tilalle
- rajapintojen suunnittelu ja toteutus
- ohjelman turvallisuus ja testaaminen
- käyttöliittymän suunnittelu ja toteutus.

2.3 Vastuualueet

Parikka oli vastuussa asiakaspuolesta, eli ohjelmiston käyttöliittymän suunnittelusta asiakkaan kanssa ja sen toteuttamisesta. Mustonen ja Oinonen olivat molemmat vastuussa serveripuolen ohjelmiston suunnittelusta ja toteutuksesta, minkä takia raportissa keskitytään pääasiassa vain siihen osa-alueeseen.

Mustosen vastuualueeseen sisältyi käytettävien tekniikoiden ja työkalujen valinta sekä asiakas- että myös serveripuolella ja Lemonsoft ERP-integraation suunnittelu ja toteutus sekä REST-rajapinnan suunnittelu asiakasohjelmille. Oinosen vastuualueeseen kuului aiemmin mainitun lisänä, PDF-tulosteiden suunnittelu ja toteutus, rajapintojen ja kirjautumisen turvallisuusasiat ja ohjelmiston testaus. Vaikka vastuualueet oli jaettu tasavertaisesti henkilöiden kesken, tiivis

yhteistyö auttoi tekemään työnteosta sujuvaa ja saamaan parhaan lopputuloksen.

3 Vaatimusmäärittely

3.1 Käyttäjävaatimukset

Ohjelmistoon tarvittiin kirjautuminen kaikille ohjelman käyttäjille ja että sen mitkään resurssit eivät olisi saatavilla ulospäin. Käyttäjille tarvittiin myös käyttäjätilien hallinta ja hallinnan kautta oikeutettujen käyttäjien tuli pystyä tekemään uusia käyttäjätilejä.

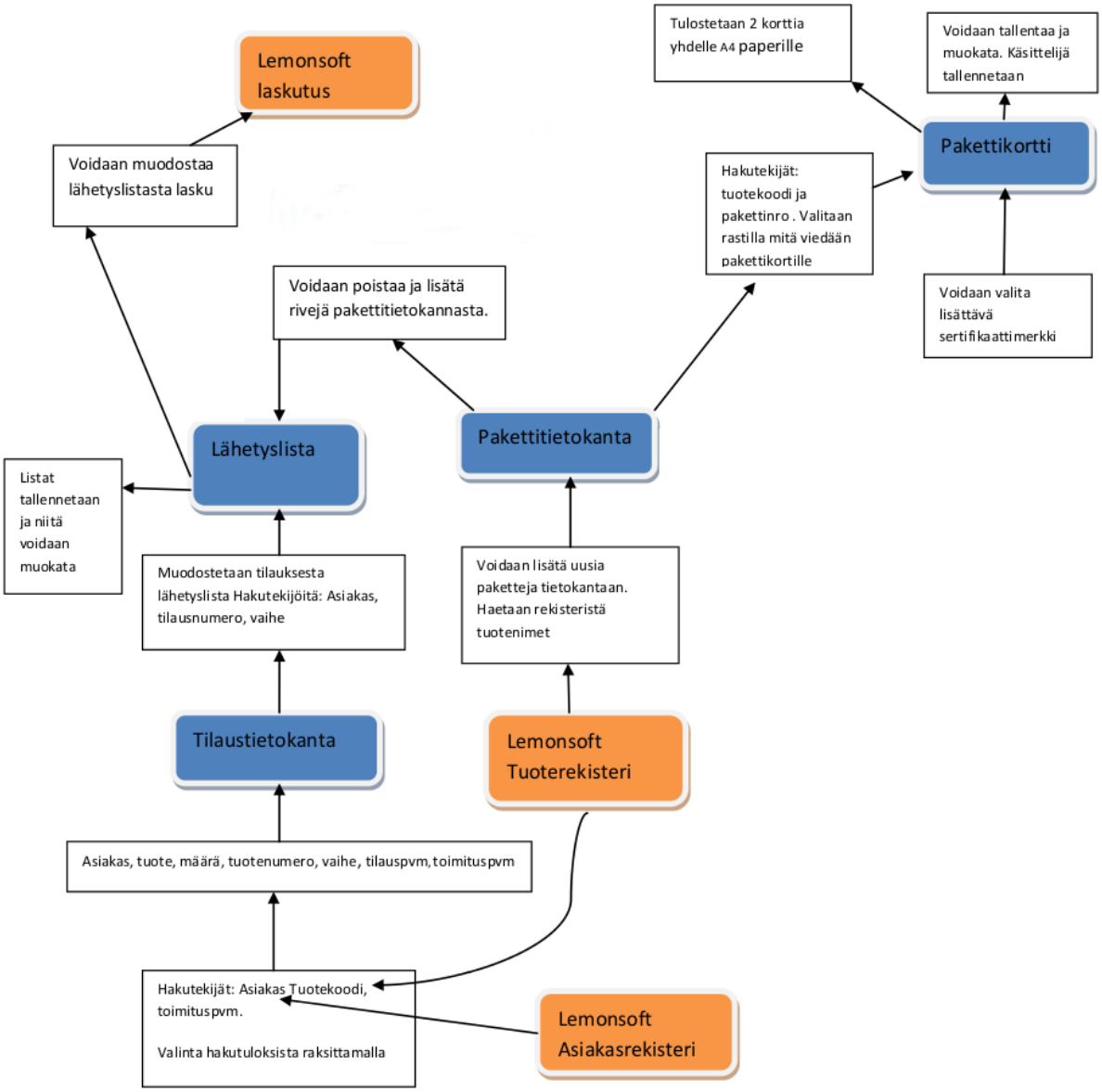
Ohjelmassa täytyi käyttäjätileihin liittyä käyttäjätasot. Tasoilla varmistetaan se että käyttäjä voi tehdä vain hänelle oikeutettuja asioita. Esimerkiksi uusien käyttäjätilien luominen onnistuu vain järjestelmän admin-henkilöiltä ja käyttäjäoikeuksien muuttaminen vain järjestelmän ylläpidolta. Ohjelmistoon tulisi siis yksi käyttäjätili jokaista työntekijää kohden ja ylläpidolle omat käyttäjätilit.

Ohjelmiston täytyi korvata Puupietarin entinen Excel-toimintamalli kokonaan ja tieto täytyi pystyä siirtämään kokonaan ohjelmalliseen muotoon. Tähän tietoon lukeutuvat mm. tilaukset, asiakkaat, tuotteet, valmistetut tuotteet ja niiden lukumäärät. Tiedon täytyi olla saatavilla myös monelta työpisteeltä jaetusti ja helposti. Tietoa piti pystyä myös siirtämään Lemonsoftin ERPin ja toteutettavan ohjelmiston välillä esimerkiksi laskujen muodostukseen Lemonsoftiin toimitettujen tilausten perusteella.

Lautapakettien tuotannossa valmistushistorian pitämisen ja selaamisen tuli olla mahdollista ja ohjelmaan piti myös olla mahdollista syöttää paketeille mitatut mittausarvot, kuten kosteudet. Lisäksi ohjelmistoon piti pystyä lisäämään asiakkaille tilauksia, tilauksiin tilattavat tuotteet ja määrät, ja tämän jälkeen tilauksista piti pystyä muodostamaan lähetyslistat.

Järjestelmän piti korvata Excel-tulosteet Puupietarin entisestä toimintamallista. Tulosteisiin lukeutuivat lähetyslistat ja puupaketteihin liimattavat tietolaput. Puupakettien tietolaput pitivät sisällään tietoa valmistetusta paketista ja sen mittatiedoista ja lähetyslistat näyttivät lähetysten sisällön. Tulosteet piti saada PDF-

muodossa selaimeen, jotta niitä olisi mahdollista tallentaa myöhempää käyttöä varten. Kuva 2 määrittää ohjelman pääominaisuuksia ja niiden relaatioita.



Kuva 2. Alkuvaiheen suunnitelma

3.2 Järjestelmävaatimukset ja tekniset rajoitteet

Järjestelmävaatimukset ohjelmistolle olivat serveri, mille ohjelmisto asennettiin ja asiakaskoneet työpisteille ohjelmiston käyttöä varten. Serverille tarvittiin tietokanta ohjelmaa varten. Tuotannossa työkoneita täytyi pystyä olemaan useita ja niillä piti pystyä työskentelemään yhtä aikaa. Serverin piti pystyä ottamaan yhteyttä Lemonsoft ERPin rajapintaan tuotteiden ja asiakkaiden noutamiseksi pal-

velusta. Näitä päivitysprosesseja ajetaan yöllä automaattisesti ajoitetusti, kun muu liikenne verkossa on vähäistä.

Teknisiä rajoitteita järjestelmälle ei ole, koska järjestelmään ei asenneta ulkopuolisia antureita tai lisälaitteita. Ainoa vaatimus on että serveri on tarpeeksi tehokas ohjelman ajamiseen ja sen asiakaskoneiden palveluun.

3.3 Uhkamalli

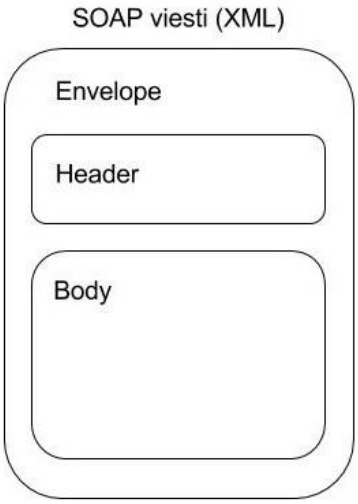
Ohjelmistolle täytyi toteuttaa uhkamalli, jossa käytiin läpi ohjelmistoon liittyviä turvallisuusasioita sekä analysoitiin niitä. Uhkamallissa uhat luokiteltiin kahdeksaan eri ryhmään uhkatyyppien perusteella ja myös niille toteutetut vastatoimet käytiin läpi, eli kuinka uhkat ohjelmistossa estettiin.

Uhkamallin määrittelyt vaikuttivat ohjelmiston muiden vaatimusten suunnitteluun. Esimerkkinä tästä on, että rajapintojen kaikki liikenne tuli tapahtua ainoastaan suojatun yhteyden avulla ja rajapinnat vaativat aina kirjautumisen.

4 Ohjelmistojen rajapinnat

4.1 SOAP

SOAP (Simple Object Access Protocol) on XML-kieleen pohjautuva tietoliikenneprotokolla, joka on luotu mahdollistamaan eri sovellusten välinen kommunikointi internetissä HTTP-protokollan avulla. Sen käyttö on riippumatonta alustasta, käytettävistä tekniikoista tai ohjelmointikielestä.



Kuva 3. SOAP-viestin rakenne

SOAP viestin rakenne koostuu neljästä eri osasta jotka ovat envelope-, header-, body-, ja fault-osat (kuva 3):

- Envelope (suomeksi: kirjekuori, kuori) sisältää tunnisteen XML-dokumentin muodosta, joka tässä tapauksessa on SOAP-viesti.
- Header (suomeksi: otsikko, ylätunnus) sisältää viestin otsikon.
- Body (suomeksi: runko) sisältää viestin kutsu- ja vastaustiedon.
- Fault (suomeksi: häiriö, vika) sisältää tila- ja virhetiedon. (W3C, 2007.)

Seuraavassa esimerkkitapauksessa serverille lähetetään onnistuva “GetTemperature” pyyntö lämpötilatiedon hakemiseen Joensuun kaupungista. Se sisältää lähetettäessä City-parametrin, joka on kaupungin nimi josta lämpötilatietoa pyydetään, minkä jälkeen vastauksena saadaan “Temperature”-parametri. Parametri pitää sisällään pyydetyn kaupungin lämpötilan. Alla on esimerkki SOAP-kutsun rakenteesta HTTP-pakettina. (W3Schools 2016.)

```
POST /CityTemperature HTTP/1.1
Host: www.forexample.fi
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body xmlns:m="http://www.forexample.fi/temperature">
 <m:GetTemperature>
```

```
   <m:City>Joensuu</m:City>
 </m:GetTemperature>
</soap:Body>
</soap:Envelope>
```

Yllä olevassa pyynnössä viestin sisältö on määritelty olevan “application/soap+xml”. Tästä viestivät osapuolet tietävät, mikä on viestin sisällön tyyppi ja kuinka sitä käsitellään. Viestin runko otsikoiden jälkeen on XML-muotoa. Alla on esimerkki miltä vastaus näyttää serveriltä asiakasohjelmaan.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body xmlns:m="http://www.forexample.fi/temperature">
 <m:GetTemperatureResponse>
   <m:Temperature>15.5</m:Temperature>
 </m:GetTemperatureResponse>
</soap:Body>
</soap:Envelope>
```

SOAP-rajapintaan liittyy yleensä myös WSDL-tiedosto (Web Services Description Language), jonka tarkoitus on kuvata rajapinnan tietotyypit ja kaikki saatavilla olevat funktiot. Tämä tiedosto on saatavilla joissakin tapauksissa rajapinnan osoitteesta laittamalla GET-parametrina koko osoitteen perään “?wsdl” parametri. Tämän jälkeen rajapinta palauttaa WSDL-kuvaustiedoston asiakkaalle.

Ohjelmointikielissä SOAP-rajapinnan kommunikointiin asiakasohjelmaan on käytössä yleensä valmis komponentti, jolle annetaan vain rajapinnan kuvaustiedosto. Yleensä apuna oleva kirjasto generoi rajapinnan perusteella ohjelmointikielen mukaisia luokkia määrittelyjen perusteella. Luokkia voidaan käyttää ohjelmistokehityksen apuna kielen omilla ominaisuuksilla ja kirjasto näiden perusteella muodostaa lähtevät XML-viestit. Myös vastauksen lukeminen takaisin ohjelmaan hoidetaan automaattisesti.

4.2 REST

REST (REpresentational State Transfer) on ohjelmointirajapintojen toteuttamiseen luotu arkkitehtuurimalli. REST-malli ei suoraan määritä, minkä protokollan päällä kommunikoinnin tulisi tapahtua. Kuitenkin web-maailmassa REST toteutetaan HTTP-protokollan päälle käyttäen HTTP-protokollan metodeita (GET, POST, PUT, DELETE). REST-rajapinnan tietojen formaatti voi vaihdella rajapinnasta riippuen, yleisin käytäntö on kuitenkin käyttää JSON-formaattia web-rajapinnoissa. Myös XML- ja CSV-formaatit ovat mahdollisia, mutta harvemmin käytettyjä. REST-rajapinta on nykyaikaisempi toteutustapa SOAP-rajapintaan verrattuna, koska pienen tiedon siirtämiseen SOAP-toteutuksessa tarvitaan paljon enemmän tietoa viestin perille saamiseen, kun taas REST-toteutuksessa viestin koko on paljon kompaktimpi. SOAP toimii myös operaatioperiaatteella, jossa kutsutaan rajapinnan funktioita, kun taas RESTissä jokainen resurssi on saatavilla omasta osoitteestaan. RESTin etuna on myös välimuistin käyttö, ja näin ollen samaa resurssia ei tarvitse välttämättä noutaa palvelimelta aina uudestaan.

REST-rajapinnassa jokainen resurssi on saatavilla omasta osoitteestaan. Lista resursseista voi olla esimerkiksi osoitteessa “www.esimerkki.fi/packages/” ja pelkän tietyn resurssin noutaminen sen yksilöivällä numerolla löytyy osoitteesta “www.esimerkki.fi/packages/123”. Tämä palauttaisi yhden package-resurssin, jonka numero on 123. Rajapinnassa on myös mahdollista toteuttaa sisäkkäisiä osoitteita, eli osoite voisi olla muotoa “www.esimerkki.fi/packages/123/values/”, mikä palauttaisi vain resurssiin relaatiossa olevat arvot. REST-rajapinnassa resursseja muokataan tekemällä kutsuja resurssien osoitteisiin HTTP-protokollan metodeilla GET, PUT, POST ja DELETE. Jokainen metodi vastaa tiettyä toimintoa mitä resurssille ollaan tekemässä. Taulukossa 1 on esitetty resurssin osoitteeseen tehdyt HTTP-metodit ja metodien tehtävät. (RestApiTutorial.com 2016.)

Taulukko 1. HTTP-metodin merkitys RESTissä (RestApiTutorial.com 2016.)

URI(Uniform Resource Identifier)	**GET** (luku)	**PUT** (päivitys)	**POST** (uuden luominen)	**DELETE** (poistaminen)

Kokoelma, esimerkiksi: http://esimerkki.fi/packages/	Listaa kaikki elementit kokoelmasta	Korvaa kokoelman annetulla kokoelmalla	Luo uuden elementin kokoelmaan	Poistaa kokoelman
Yksi osa, esimerkiksi: http://esimerkki.fi/packages/123/	Hakee elementin numero 123	Korvaa elementin numero 123 tai luo uuden jos sitä ei ole	Ei yleisesti käytössä	Poistaa elementin numero 123

5 Web-ohjelmistojen yleisimmät haavoittuvuudet

5.1 XSS

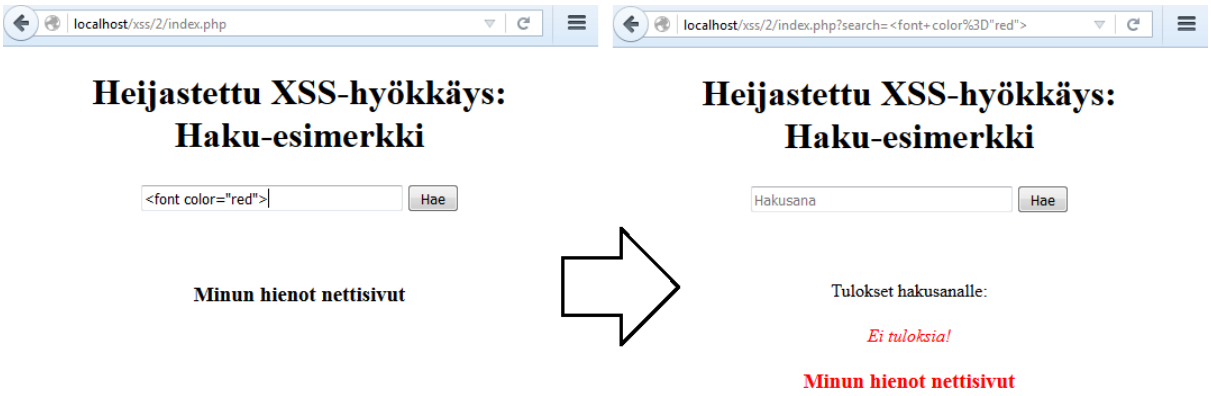
XSS eli Cross-Site Scripting, on tunkeutumistyyppinen hyökkäys, ja se on tällä hetkellä yksi yleisimmistä web ohjelmistoihin kohdistuvista hyökkäystyypeistä. Tarkoituksena XSS-hyökkäyksessä on syöttää haitallista koodia hyökkäyksen mahdollistavalle sivustolle ja tätä kautta päästä käsiksi muun muassa sivun käyttäjien evästeisiin - jolloin hyökkääjä voi esiintyä hyökkäyksen uhrin käyttäjänä tai hyväksikäyttää uhrin palvelusta löytyviä henkilökohtaisia tietoja. Hyökkäyksen mahdollistaa useimpien sivustojen vaatima JavaScript tuki, jota hyödynnetään muutamin eri tavoin. Jotta voi ymmärtää mitä XSS-hyökkäys on ja mitä siinä tapahtuu, henkilöltä vaaditaan perustietämys HTML- sekä JavaScript-kielistä, sekä perustieto siitä, mikä tehtävä serveripuolella web-ohjelmistoissa on. (OWASP 2016.)

5.1.1 Pysymättömät tai heijastetut XSS-hyökkäykset

Pysymättömät tai heijastetut XSS-hyökkäykset (englanniksi: Non-Persistent, Reflected tai Type-II XSS) koskevat vain sitä henkilöä joka suorittaa haitallisen skriptin omassa selaimessaan. Tässä tapauksessa haitallinen skripti on kiinni esimerkiksi linkin tai urlin lopussa, jota kokematon käyttäjä ei välttämättä pysty huomaamaan, varsinkin kun keinoja sen naamioimiseksi on olemassa. Tällä hyökkäystyypillä voidaan esimerkiksi kalastaa uhrin yksityisiä tietoja tai varastaa

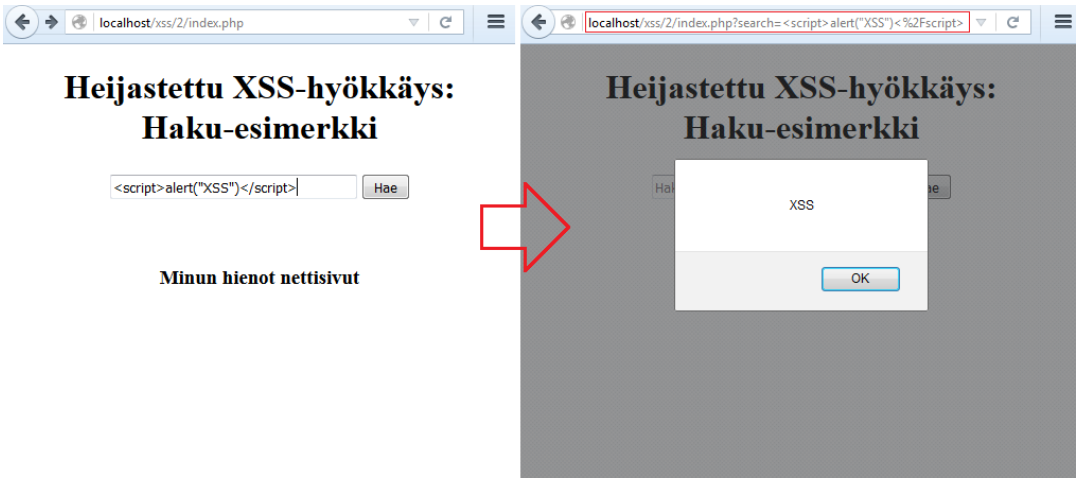
uhrin evästeet ja päästä sen kautta käsiksi kaikkiin uhrin palvelusta löytyviin tietoihin. (OWASP 2016.)

Käydään nyt läpi yksinkertainen esimerkki heijastetun XSS-hyökkäyksen perusperiaatteista. Esimerkkiä varten on pystytetty HTTP-palvelin, ja sinne luotu www-sivu joka sisältää otsikon, yhden syötekentän ja hakunapin, joka haettaessa käyttää GET-metodia tulosten näyttämiseen. Kun alkutilanteessa sivuston osoite on http://localhost/xss/2/index.php, muuttuu se hakutoiminnon jälkeen “asia”-hakusanalla muotoon: http://localhost/xss/2/index.php?search=asia. Kun web sivuston syötekenttää ei tässä tapauksessa ole suojattu XSS-hyökkäykseltä, se ottaa sisäänsä kaikki html-tagit, jotka tämän jälkeen suoriutuvat muun sivun koodin mukana. Esimerkissä hakukenttään syötetään html font-tagi jonka väri-attribuutin arvoksi annetaan “red” eli punainen. Font-tagi jätetään sulkematta ja tällöin se määrittelee lopun sivuston fonttien värin punaiseksi (kuva 4).



Kuva 4. Heijastettu XSS-hyökkäys ensimmäinen esimerkki

Tällä tavalla heijastettu XSS-hyökkäys perustasolla toimii. Erona on vain se, että font-tagin sijasta käytetään script-tagia. Tagi myös suljetaan, ettei sivuston muu toiminta häiriintyisi. Haittakoodi syötetään script-tagien väliin, suoritetaan hakutoiminto, jonka tuloksena saadaan haitallinen linkki. Enää tarvitsee vain saada henkilö avaamaan saastunut linkki (esimerkiksi sähköpostista) ja hyökkäys on onnistunut (kuva 5).



Kuva 5. Heijastettu XSS-hyökkäys toinen esimerkki

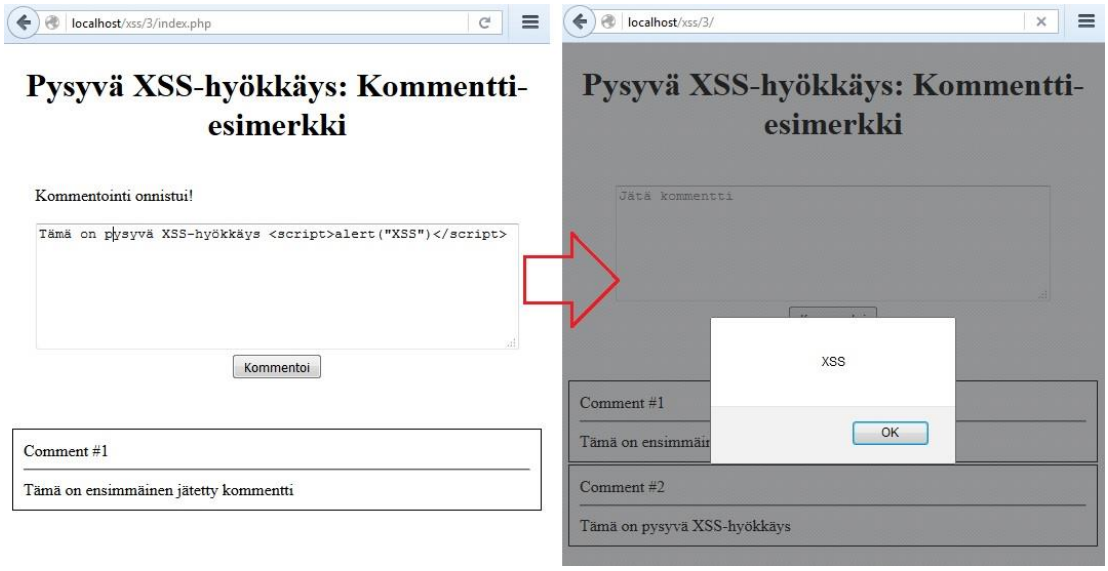
5.1.2 Pysyvät tai säilötyt XSS-hyökkäykset

Pysyvät tai säilötyt XSS-hyökkäykset (englanniksi: Persistent, Stored tai Type-I XSS), ovat niitä joissa haitallinen skripti saadaan tallennettua suoraan kohdeservereille, pääasiassa niiden tietokantoihin. Kohteita ovat esimerkiksi erilaiset keskustelufoorumit tai kommentointikentät. (OWASP 2016.)

Tehdään tästäkin XSS-hyökkäyksestä esimerkkitapaus. Esimerkkiä varten on luotu www-sivu, joka sisältää otsikon, yhden syötekentän sivulle jätettävälle kommentille ja napin kommentin jättämistä varten. Myös jätetyt kommentit näytetään sivulla ja jos kommentteja ei ole, sivu näyttää ilmoituksen siitä. Koska pysyvässä XSS-hyökkäyksessä tavoitteena on saada ujutettua haitallinen skripti palvelimelle, esimerkkiä varten täytyi käynnistää HTTP-palvelin ja myös luoda sivulla käytettävä tietokanta. Se sisältää yhden taulun (comments) ja taululla on kaksi attribuuttia id ja comment.

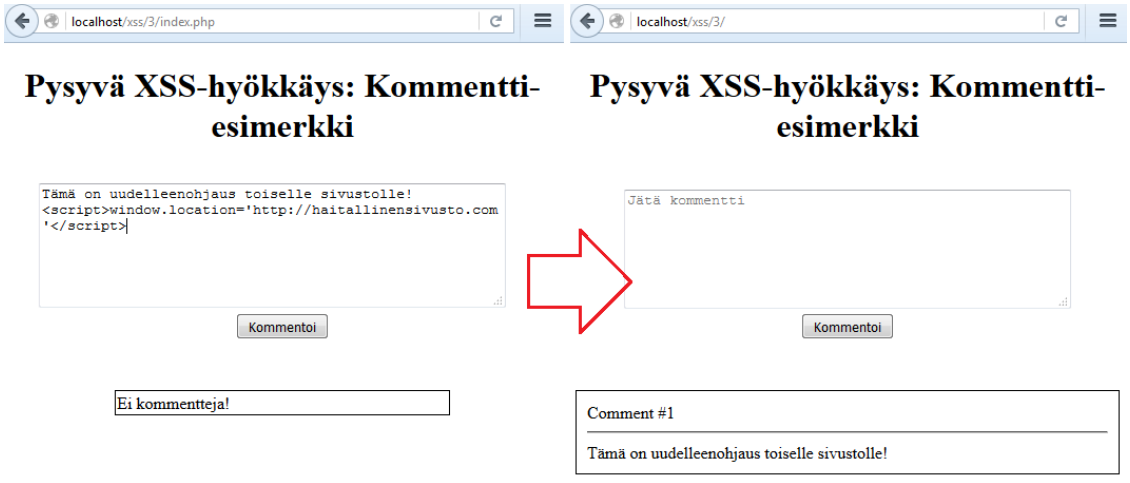
Kuvitellaan että esimerkin nettisivu on foorumi, jonne voi jättää viestejä muiden luettavaksi ja tälläkään kertaa XSS-hyökkäyksiä kohtaan ei ole osattu varautua millään tavalla. Hyökkääjä syöttää kenttään script-tagien väliin halutun JavaScript-koodin, mikä esimerkissä on pelkkä “alert” kutsu. Toinen mahdollisuus on suoran JavaScript koodin sijaan käyttää script-tagin src-attribuuttia ja syöttää siinä linkki haitalliseen JavaScript tiedostoon, joka sijaitsee toisessa osoitteessa (esim. <script src”http://toinensivu.com/XSS_scripti.js.”></script>). Kommentin jättämisen jälkeen, jokainen sivulla vieraileva suorittaa automaattisesti kyseessä

olevan JavaScript kutsun ja ruudulle ilmestyy popup-ikkuna missä viesti näkyy (kuva 6).



Kuva 6. Pysyvä XSS-hyökkäys ensimmäinen esimerkki

Yksi mahdollisuus on myös uudelleenohjata sivustolle saapuva suoraan haitalliselle sivulle. Tällöin script-tagien väliin kirjoitetaan esimerkiksi “window.location=’http://haitallinensivusto.com’”. Kun käyttäjä avaa sivun, vain teksti ennen script-tagia jää näytettäväksi, ja selain hakeutuu heti skriptin ajettuaan haitalliselle sivustolle (Kuva 7).



Kuva 7. Pysyvä XSS-hyökkäys 2. esimerkki

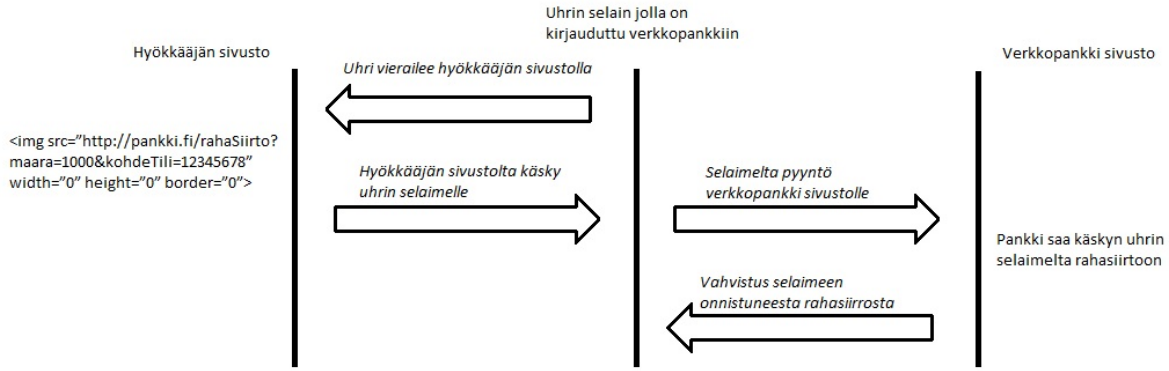
Haitallinen sivusto voisi esimerkiksi olla ulkoisesti sekä mahdollisesti myös osoitteeltaan alkuperäistä sivua muistuttava sivusto. Uudelleenohjattu käyttäjä

luottaa sivustoon, eikä huomaa eroavaisuuksia alkuperäiseen. Uhri voisi esimerkiksi ladata ohjelman sivulta ja asentaa haittaohjelman omalle laitteelleen, ja näin hyökkäyksen tarkoitusperä on saavutettu.

5.2 CSRF

CSRF eli Cross-Site Request Forgery hyökkäyksessä tarkoituksena on saada uhrin selain lähettämään väärennetty HTTP-pyyntö haluttuun palveluun. Tämä pyyntö voi sisältää uhrin ja hänen käyttämänsä palvelun välistä tilatietoa, esimerkiksi evästeen sisältämän istuntotunnisteen muodossa. Hyökkääjän saatua haltuunsa istuntotunnisteen, olettaen että uhrin suorittama autentikaatio on vielä voimassa, voi hän esiintyä uhrina ja lähettää pyyntöjä hänen tunnistetiedoillaan. Näitä pyyntöjä voivat olla esimerkiksi varojen siirtotapahtumat, email-osoitteen vaihtaminen tai vaikka tilin poistaminen. Jos hyökkäyksen uhri on palvelun admin-henkilö, koko palvelun toimivuus voi olla vaarassa. (OWASP 2016.)

Esimerkkitapauksessa hyökkääjä lähettää uhrille sähköpostia, jossa on linkki hyökkääjän haittasivustolle (kuva 8).



Kuva 8. CSRF esimerkki

Haittasivustolta löytyy HTTP-koodista näkymätön kuva jonka selain normaalisti kääntää ja joka on muotoa “<img src=”http://pankki.fi/rahaSiirto?maara=1000&kohdeTili=12345678” width=”0” height=”0” border=”0”>”. Samaan aikaan uhri on kirjautuneena verkkopankkiin, jolloin haittasivuston suorittama pyyntö kulkee uhrin selaimen kautta pankille asti, eli uhrin tililtä siirtyy rahaa hyökkääjän tilille.

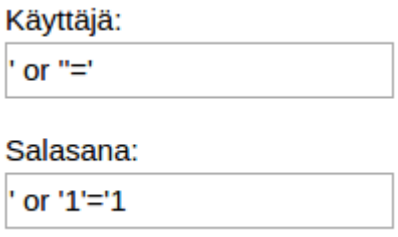
5.3 SQL-injektio

SQL-injektio on web-pohjaisten ohjelmistojen tietokantoihin kohdistuva hyökkäystyyppi. Injektio suoritetaan ohjelman käyttöliittymän tai muun sen ulkoisen tietolähteen kautta. Lähteestä tieto menee järjestelmän kautta tietokanta-tasolle asti. Tietokanta suorittaa hyökkääjän kyselyn ilman ongelmia, jos tietokantaa käyttävä ohjelma ei ole osannut sitä vastaan puolustautua. Yleisemmin SQL-injektiolla hyökkääjän tarkoituksena on saada arkaluontoista tietoa tietokannasta haltuunsa, mitä ohjelmasta ei muuten saa ulos. Tällaisia ovat voivat olla esimerkiksi salasanat, sähköpostiosoitteet, luottokorttien tiedot, jne. Tai tavoite voi olla muu tahallisen pahan aiheuttaminen, tietojen poistaminen tai muuttaminen. (W3Schools 2016.)

Esimerkkitapauksessa kuvitellaan kirjautuminen web-sovellukseen. Lomakkeessa on kentät käyttäjänimelle ja salasanalle, joiden saamat syötteet siirretään suoraan tietokannalle suoritettavaan SQL-lauseeseen ilman minkäänlaista suodatusta tai suojausta. Tällöin SQL-kysely lomakkeen kentille voisi olla esimerkiksi PHP-koodilla seuraava.

```
$sql = “SELECT * from Users WHERE Username = ‘“ + $käyttäjä + “‘ AND
Password = ‘“ + $salasana + “‘;”;
```

Hyökkääjä normaalin käytön sijaan yrittää syöttää kenttiin käyttäjänimeksi ' or ''=' ja salasanaksi ' or '1'='1 (kuva 9).



Kuva 9. SQL-injektio 1. esimerkki

Yllämainittujen syötteiden suora liitos SQL-kyselyyn ilmaan minkäänlaista validointia tai suojausta muodostaisi seuraavanlaisen SQL-lauseen PHP-koodissa.

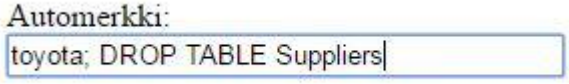
```
$sql = “SELECT * from User WHERE Username = '' or ''='' AND Password = ''
or '1'='1';”;
```

Yllämainittu kysely palauttaa jokaisen rivin Users-taulusta ja näin hyökkääjä saa tietoonsa kaikkien käyttäjien tiedot ja salatut salasanat. Syynä tälle on, että käyttäjänimen kohdalla kysely katsoo onko käyttäjänimi tyhjä teksti tai onko tyhjä teksti sama kuin tyhjä teksti. Tämä aiheuttaa sen että kaikki kyselyn käyttäjänimen vertailut menevät läpi. Salasana-kentän kohdalla on sama tilanne. Ensin kysely vertaa onko salasana tyhjä tai sitten yksi on yhtäkuin yksi. Tässäkin tapauksessa viimeinen vaihtoehto on aina tosi kaikille taulun riveille. Näin ollen kyselyn kummatkin ehdot täyttyvät kaikille tietokannan riveille ja kaikki rivit palautetaan. Miten hyökkäykselle altis ohjelmisto tällaisen tuloksen käsittelee, on aina kuitenkin ohjelmistokohtaista. Pahimmassa tapauksessa ohjelma tulostaa kaikki asiakkaat esimerkiksi virheen sattuessa, jos kehittäjä ei ole ottanut sitä huomioon.

Useimmista tietokannoista löytyy myös tuki sarjassa oleville SQL-kyselyille. Kuvitellaan tapaus missä autokauppa-sivustolta löytyy hakukenttä, jolla haetaan automerkkejä. SQL-lause muodostuisi seuraavasti.

SELECT * FROM Cars WHERE Name = automerkki

Koska SQL-lause päättyy hakukentän syötteeseen, hyökkääjä voi jatkaa lisäämällä alkuperäisen kyselyn perään puolipisteen ja kirjoittaa tämän jälkeen toisen SQL-lauseen. Jos hyökkääjä tietää tietokannasta löytyvien taulujen nimiä, tai onnistuu vaikkapa arvaamaan sellaisen, voi hyökkääjä esimerkiksi poistaa niitä. Esimerkkinä hyökkääjä kirjoittaa hakukenttään “toyota; DROP TABLE Suppliers” (kuva 10). Ensimmäiseksi kysely noutaa kaikki Toyota-merkkiset autot, minkä jälkeen koko Suppliers-taulu tuhotaan tietokannasta.



Kuva 10. SQL-injektio 2. esimerkki

6 Web-ohjelmoinnin turvallisuusratkaisut

6.1 CORS

Cross-origin resource sharing on mekanismi ja standardi, mikä on luotu tarpeesta hakea resursseja toisesta domainista kuin mistä haettavan resurssin pyyntö suoritetaan. Tämä onkin ollut jo mahdollista internet sivujen toteutuksessa muun muassa kuvien, CSS-tiedostojen ja videoiden muodossa. Kuitenkin esimerkiksi AJAX-pyynnöissä eri domainien väliset pyynnöt ovat olleet oletuksena estettyjä, niiden kyetessä suorittamaan muun muassa sellaisia HTTP-pyyntöjä, kuten POST, PUT ja DELETE. Tällöin sivusto voi altistua erilaisille XSS-hyökkäystyypeille. CORS:n ollessa käytössä, tämäkin on mahdollista tehdä turvallisesti. (Hossain 2011.)

Esimerkissä sivusto osoitteessa www.ekasivu.fi tekee pyynnön www.tokasivu.fi osoitteeseen noutaakseen sieltä tietoa www.ekasivu.fi:hin kirjautuneesta käyttäjästä. CORS määrittelee AJAX- ja HTTP-dataa muokkaavien pyyntöjen suorittamisen selaimella seuraavasti. Selain tekee esipyynnön osoitteeseen www.tokasivu.fi käyttämällä HTTP OPTIONS-metodia, joka sisältää lähdedomainin HTTP-otsikoissa “Origin: http://www.ekasivu.fi”. Palvelin vastaa joko lähettämällä ”Access-Control-Allow-Origin”-otsikon, joka sisältää pyyntöihin sallitut sivustot. Esimerkiksi “Access-Control-Allow-Origin: http://www.ekasivu.fi”. Tai otsikko voi sisältää tähti-merkinnän (*), mikä tarkoittaa että mistä vain lähetetyt pyynnöt ovat sallittuja. Jos pyyntö ei ole sallittu, niin palvelimelta saadaan virheviesti. (Hossain 2011.)

6.2 HTTPS

HTTPS eli Hyper Text Transfer Protocol Secure on turvallisempi versio HTTP-protokollasta. Sana secure HTTPS:ssä tarkoittaa että siinä käytetään joko SSL (Secure Sockets Layer), tai TLS (Transport Layer Security) turvallisuusprotokollaa muodostamaan salattu HTTP-yhteys palvelimen ja selaimen välille. SSL-sertifikaatteja, joita käytetään HTTPS-yhteyden muodostamiseen, on kahta eri tyyppiä ja näitä ovat itse allekirjoitetut ja ns. viralliset sertifikaatit. Itse allekirjoitetussa sertifikaatissa vierailijan tullessa sivustolle, selain ensin varoittaa sertifi-

kaatista ja kertoo käyttäjälle että sertifikaatin myöntänyt taho ei ole luotettava ja tämän jälkeen pyytää vierailijalta vahvistuksen sivustolle siirtymiseen. Virallisen sertifikaatin voi anoa erinäisiltä luotetuilta tahoilta (mm. Comodo, DigiCert, Symantec) muutaman eri vaiheen kautta. Prosessin onnistuttua sivustoa pidetään luotettavana ja selain ei enää sivustolle siirtyessä erikseen kysy vierailijan vahvistusta asialle. HTTPS:ää käytetään muun muassa sivustoilla joissa käytetään verkkomaksuja, kuten erilaisissa verkkokaupoissa tai suoraan verkkopankeissa. Nykyään HTTPS:n käyttö on kuitenkin yleistynyt myös muilla sivustolla sen taatessa luotettavuutta käyttäjille, pitämällä heidän toiminnan palveluissa yksityisenä. (Rescorla 2000, 1-5.)

HTTP-yhteydessä siirretty data kulkee pelkistettynä tekstinä. Eli jos hyökkääjä saa vakoiltua siirretyn datan sisällön, on se silloin myös helposti luettavissa. HTTPS-yhteydessä selain ja palvelin suorittavat SSL-kättelyn käyttäen asymmetristä julkisen avaimen infrastruktuuria (Public Key Infrastructure eli PKI) seuraavasti:

1. Selain ottaa yhteyden SSL-suojattuun serveriin ja lähettää pyynnön tunnistautumisesta.
2. Serveri lähettää kopion sen SSL -sertifikaatista, joka sisältää myös serverin julkisen avaimen.
3. Selain tarkistaa kyseisen sertifikaatin valtuutettujen myöntäjien listasta, onko sertifikaatti voimassa, koskematon ja oikeellinen kyseiselle sivustolle. Tarkistuksen onnistuessa selain ensin luo symmetrisen istuntoavaimen, sitten salaa ja tämän jälkeen lähettää symmetrisen istuntoavaimen käyttämällä serverin antamaa julkista avainta.
4. Serveri purkaa symmetrisen istuntoavaimen käyttämällä serverin yksityistä avainta ja tämän jälkeen lähettää kuittauksen, mikä on salattu istuntoavaimella ja salattu istunto voidaan aloittaa.
5. Serveri ja selain salaavat kaiken siirrettävän datan luodulla istuntoavaimella. (Heaton 2014.)

Näin siirretty data selaimen ja palvelimen välillä salataan epämääräiseksi merkkijonoksi ja vaikka hyökkääjä saisi kaapattua dataa sen siirron aikana, hyökkääjä ei voi purkaa sitä ilman salausavainta, joka on vain käyttäjän selaimen ja palvelimen tiedossa. (Heaton 2014.)

6.3 Kirjautuminen

Perinteisesti web-ohjelmistoissa käyttäjän autentikoinnissa on käytetty evästeisiin pohjautuvaa ratkaisumallia, missä käyttäjän oikeudet tarkistetaan jokaisella palvelimelle suoritetulla pyynnöllä käyttäjälle luodusta evästeestä. Tällöin palvelimen ei tarvitse joka kerta käyttäjän sivustolla siirtyessä pyytää häntä kirjoittamaan käyttäjätunnusta ja salasanaa, että saadaan selville käyttäjän oikeudet sivustolla, tai mitä hän on siellä aiemmin tehnyt, vaan nämä kaikki tiedot palvelin löytää evästeistä. Evästeiden käytöllä on myös haittapuolensa ja se voi altistaa palvelun käyttäjän hakkereiden suorittamille hyökkäyksille (esim. XSS- tai CSRF-hyökkäystyypit). Evästeiden käyttämiselle käyttäjän autentikoinnissa löytyy myös vaihtoehtoinen tapa, tokeneihin perustuva autentikointi.

Tokeneihin perustuvassa autentikoinnissa on paljon eroavaisuuksia ja etuja eväste-pohjaiseen ratkaisutapaan verrattuna. Kirjautumisen onnistuessa serveri antaa käyttäjälle tokenin, jota sen jälkeen käytetään todentamaan käyttäjän oikeellisuus palvelimen ja käyttäjän välisessä kommunikoinnissa. Tokenit tallennetaan käyttäjän laitteelle ja niiden data kulkee salattuna HTTP headerissa eikä evästeissä, mikä jo itsessään sulkee pois hyökkääjältä mahdollisuuden CSRF hyökkäykseen. Tokenit on tehty itsenäisiksi kokonaisuuksiksi varastoimaan käyttäjän tietoja, jolloin palvelimen ei tarvitse käyttää resurssejaan istunnon säilyttämiseen, vaan kaikki tarvittava tieto kulkee tokenien mukana. Myös CORS toimii tokeneiden kanssa, mikä ei usein evästeiden tapauksessa ole mahdollista. Koska tokeneihin on jo luotu standardeja kuten JWT, niihin löytyy kattavat kirjastot ja dokumentaatiot useille eri ohjelmointikielille (mm. Python, Ruby, .NET, Java, PHP), ja täten tokeneiden käyttöönotto on sovelluskehittäjille vaivatonta.

7 Ohjelmistokehitystyön hallinnointi

Ohjelmistokehityksen tueksi on tarjolla paljon erilaisia ohjelmia ja työkaluja. Näistä kaikista saatavilla olevista työkaluista voidaan toteuttaa jokaiselle ohjelmistokehittäjälle sopiva ohjelmistokehitysympäristö. Työkaluissa on yleensä hyvä tarjonta ohjelmallisille rajapinnoille, joilla ohjelmistokehittäjä voi toteuttaa kokonaan oman haluamansa uniikin kehitysympäristön. Tulevissa kappaleissa on käyty läpi mitä erilaisia työkaluja eri asioiden hallintaan ohjelmistokehityksen

tueksi on saatavilla. Kappaleet eivät käsittele mitään tiettyjä työkaluja tai työmenetelmiä, vaan niissä käydään läpi asioita yleisellä tasolla.

7.1 Versionhallinta

Versionhallinta mahdollistaa ohjelmistokehityksen monelta kehittäjältä samanaikaisesti lähdekoodin versioinnin ansiosta. Tunnetuimpia versionhallintatyökaluja ovat Git ja Mercurial, joita on kuitenkin vaikea vertailla toiminnallisuuden ollessa pääpiirteittäin samanlainen ja valinta perustuukin usein käyttäjien omiin tottumuksiin ja mielipiteisiin. Ne ovat pääosin komentorivityökaluja, mutta molempiin on saatavilla myös lisäosia graafiselle käyttöliittymälle, jotka voivat osaltaan vähentää työn ja opettelun määrää.

7.1.1 Ohjelman säilö (repository)

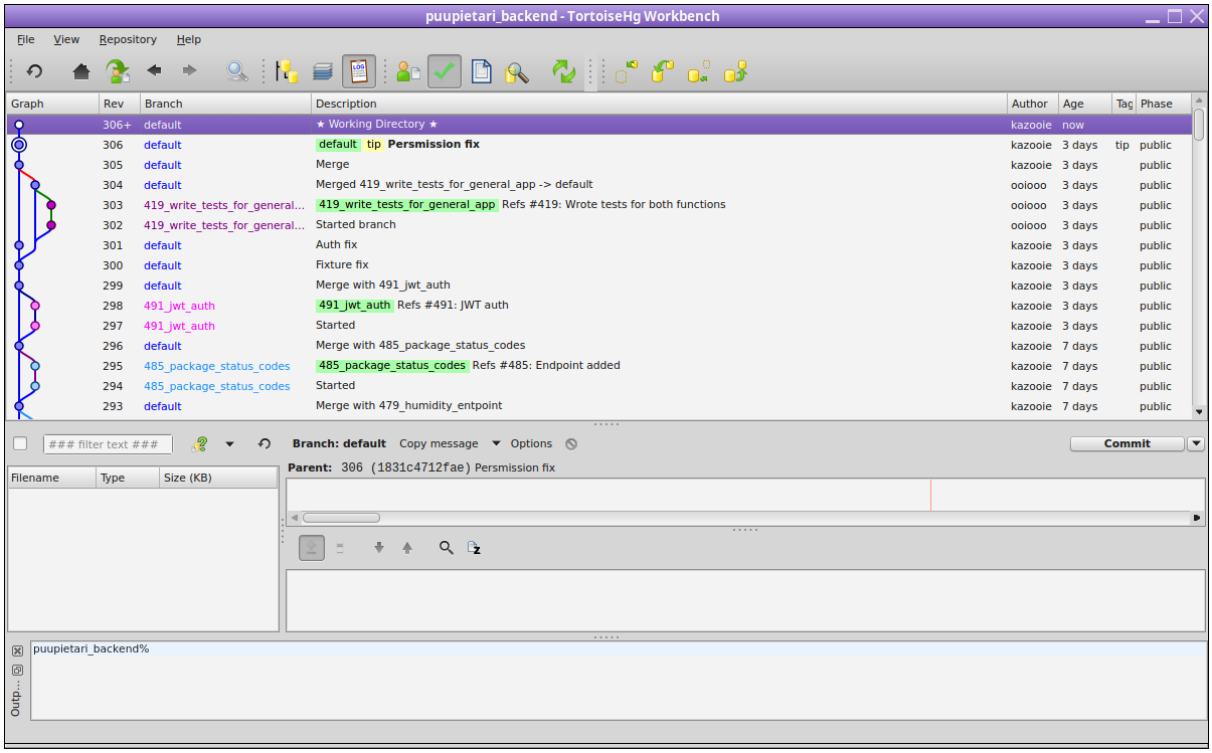
Versionhallinnan yksi tärkein osa on ns. ohjelmiston säilö, eli repository. Repository ylläpitää ja säilöö koodin muutoksia. Yleensä ohjelmistokehitykseen sisältyvät päärepository ja jokaisen ohjelmistokehittäjän henkilökohtainen oma repository. Henkilökohtaiset repositoryt sijaitsevat ohjelmistokehittäjien omilla tietokoneilla, joissa tehdyt muutokset haluttaessa lähetetään päärepositoryyn. Päärepositoryn tehtävänä on ylläpitää ja tarjota ohjelmistokehityksen tuloksena saadut lopulliset muutokset ohjelmistokehittäjille. Tällöin ohjelmiston kehittämistyötä voidaan tehdä usealta tietokoneelta samanaikaisesti. (git-scm 2016).

7.1.2 Kehittämistyön prosessi

Ohjelmistokehitystyön ensimmäisessä vaiheessa ohjelmistokehittäjä lataa viimeisimmän version ohjelmistosta omalle tietokoneelleen, avatakseen sen henkilökohtaisessa ohjelmointiympäristössään. Kun kehittäjä tekee lisäyksiä koodiin, hän varmistaa ensin päärepositorysta ovatko muut kehittäjät tehneet muutoksia ohjelmistoon ja pitää huolen siitä, että käytössä on aina viimeisin versio. Tämän jälkeen tulee tarkastaa että omat muutokset ovat yhteensopivia viimeisimmän version kanssa. Varmistuksen saatua voidaan omat muutokset päivittää päärepositoryn hallinnoimaan lopulliseen versioon muiden saataville. Yleensä kehittä-

jä ottaakin tehtäväkseen toteuttaa yhden pienen toiminnallisuuden tai vian korjauksen ohjelmasta kerrallaan.

Versionhallinnassa käytetään niin sanottua haaroitusta kehitystyön tukena, mistä nähdään kuvaava esimerkki TortoiseHg ohjelmiston graafisesta käyttöliittymästä (kuva 11).



Kuva 11. TortoiseHG-ohjelmiston graafinen käyttöliittymä

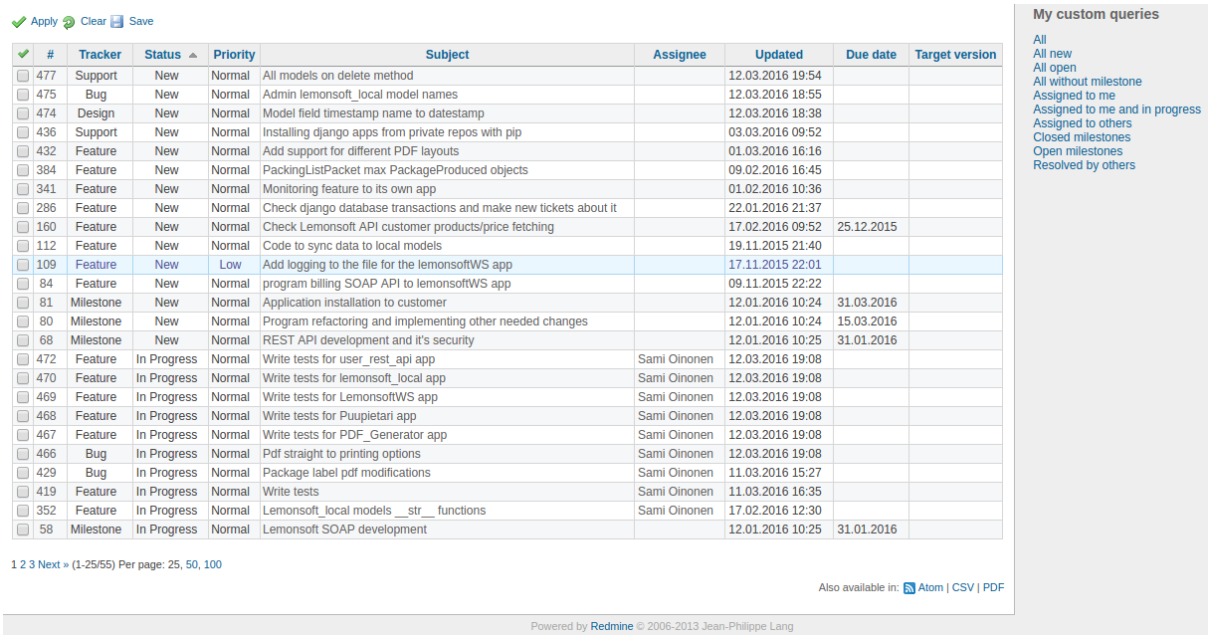
Kuvasta on nähtävillä Graph-sarake joka esittää kehitystyön haaroitusta ja liitoksia eri kehitysvaiheiden välillä. Pallot sarakkeessa kuvaavat pisteitä joihin voidaan palata ja joista voidaan aloittaa uusia haaroja. Viivat pallojen välillä kuvaavat haarojen yhteyttä eli eri pisteisiin tehtyjä muutoksia. Haarapuussa on aina olemassa niin sanottu päähaara, mitä pidetäänkin aina päärepositoryn viimeisimpänä versiona, mistä uudet haarat aloitetaan ja mihin sivuhaarat lopulta loppuvat. On myös mahdollista että sivuhaaraa ei liitetä päähaaraan, jos kehittäjän tekemiä ominaisuuksia ei haluta käyttöön tai ne jätetään odottamaan mahdollista myöhempää jatkokehitystä varten. Näin turvataan päähaaran toimivuus eivätkä muutokset näy muille ohjelmistokehittäjille.

7.2 Projektinhallinta

Projektinhallinta toimii apuvälineenä ison ohjelmistoprojektin hallintaan. Projektinhallintaohjelmisto voi ylläpitää mm. projektin tehtäviä, aikataulua, julkaistuja versioita, tekijöiden työmäärää ja työtehtävien seurantaa. Projektityökaluja on tarjolla niin ilmaisia kuin maksullisiakin, ja ominaisuudet vaihtelevat paljon jokaisen työkalun välillä.

7.2.1 Issue-tracking

Yksi projektinhallintatyökalun tärkeimmistä osioista on issue-tracking eli niin sanottu tikettien tai tehtävien seuranta. Tiketit ovat usein projektille kirjoitettuja pieniä tehtäviä ja yksi tiketti voi olla esimerkiksi uuden pienen toiminnallisuuden tekeminen tai vian korjaaminen ohjelmassa. Kuvasta nähdään Redmine ohjelmiston tikettien listaaminen sivulla (kuva 13).



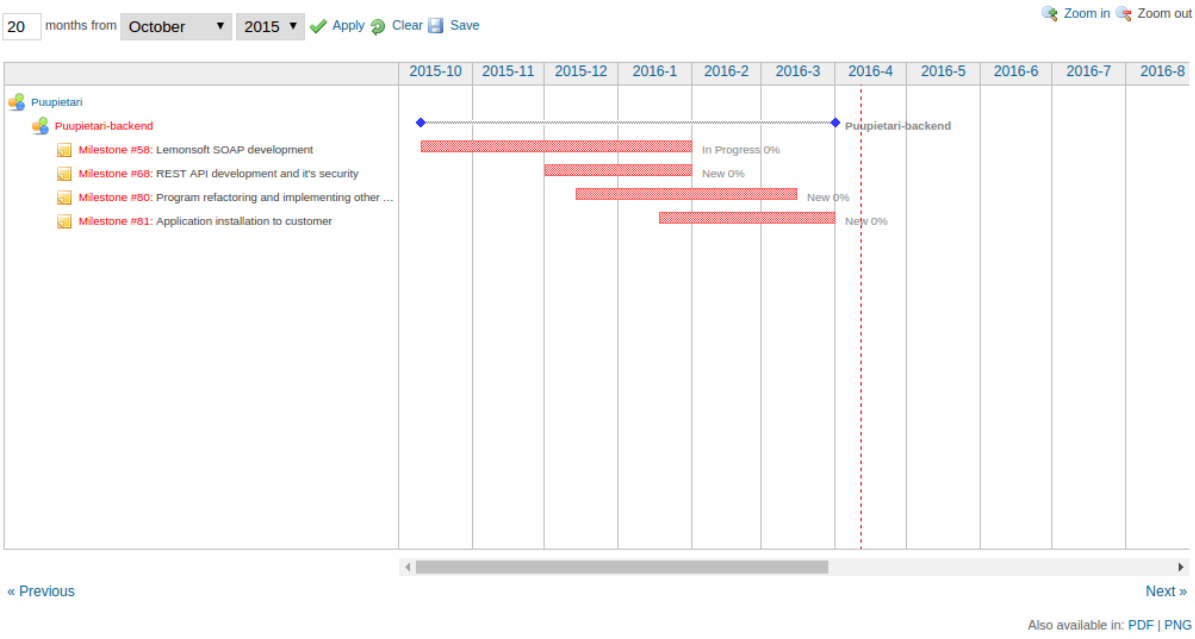
#	Tracker	Status	Priority	Subject	Assignee	Updated	Due date	Target version
477	Support	New	Normal	All models on delete method		12.03.2016 19:54		
475	Bug	New	Normal	Admin lemonsoft_local model names		12.03.2016 18:55		
474	Design	New	Normal	Model field timestamp name to datestamp		12.03.2016 18:38		
436	Support	New	Normal	Installing django apps from private repos with pip		03.03.2016 09:52		
432	Feature	New	Normal	Add support for different PDF layouts		01.03.2016 16:16		
384	Feature	New	Normal	PackingListPacket max PackageProduced objects		09.02.2016 16:45		
341	Feature	New	Normal	Monitoring feature to its own app		01.02.2016 10:36		
286	Feature	New	Normal	Check django database transactions and make new tickets about it		22.01.2016 21:37		
160	Feature	New	Normal	Check Lemonsoft API customer products/price fetching		17.02.2016 09:52	25.12.2015	
112	Feature	New	Normal	Code to sync data to local models		19.11.2015 21:40		
109	Feature	New	Low	Add logging to the file for the lemonsoftWS app		17.11.2015 22:01		
84	Feature	New	Normal	program billing SOAP API to lemonsoftWS app		09.11.2015 22:22		
81	Milestone	New	Normal	Application installation to customer		12.01.2016 10:24	31.03.2016	
80	Milestone	New	Normal	Program refactoring and implementing other needed changes		12.01.2016 10:24	15.03.2016	
68	Milestone	New	Normal	REST API development and it's security		12.01.2016 10:25	31.01.2016	
472	Feature	In Progress	Normal	Write tests for user_rest_api app	Sami Oinonen	12.03.2016 19:08		
470	Feature	In Progress	Normal	Write tests for lemonsoft_local app	Sami Oinonen	12.03.2016 19:08		
469	Feature	In Progress	Normal	Write tests for LemonsoftWS app	Sami Oinonen	12.03.2016 19:08		
468	Feature	In Progress	Normal	Write tests for Puupietari app	Sami Oinonen	12.03.2016 19:08		
467	Feature	In Progress	Normal	Write tests for PDF_Generator app	Sami Oinonen	12.03.2016 19:08		
466	Bug	In Progress	Normal	Pdf straight to printing options	Sami Oinonen	12.03.2016 19:08		
429	Bug	In Progress	Normal	Package label pdf modifications	Sami Oinonen	11.03.2016 15:27		
419	Feature	In Progress	Normal	Write tests	Sami Oinonen	11.03.2016 16:35		
352	Feature	In Progress	Normal	Lemonsoft_local models __str__ functions	Sami Oinonen	17.02.2016 12:30		
58	Milestone	In Progress	Normal	Lemonsoft SOAP development		12.01.2016 10:25	31.01.2016	

Kuva 12. Redmine-ohjelmiston tikettien listaaminen

Tiketit voivat olla vapaita poimittaviksi, tai ne voidaan merkitä tietylle ohjelmistokehittäjälle. Kun tiketti otetaan tehtäväksi, sen valinnut ohjelmistokehittäjä aloittaa versionhallintaan oman haaran, ja tehtävän valmistuttua haara liitetään takaisin päähaaraan. Tiketti merkitään tikettijärjestelmään ratkaistuksi mahdollisten lisätietojen kanssa, jonka jälkeen esimerkiksi projektin vetäjä voi arvioida tehtävän ja tehdä siihen myös muita merkintöjä.

7.2.2 Muuta toiminnallisuutta

Hyvä projektinhallintaohjelmisto on todella monipuolinen ja tarjoaa paljon muutakin toiminnallisuutta tikettijärjestelmän lisäksi. Tällaisia ovat esimerkiksi gantt-taulu, wiki-sivut, työtuntien ajan seuranta ja julkaistujen versioiden hallinta. Kuvassa 13 nähdään, miltä gantt-taulu näyttää Redmine-projektinhallintaohjelmistossa.



Kuva 13. Redmine gantt-taulu

Gantt-taulussa on esitetty tikettijärjestelmän tikettien aloitus- ja lopetuspäivämäärät. Näin projektipäälliköt pystyvät hallitsemaan projektin kulkua ja katsomaan että pysytään suunnitellussa aikataulussa.

7.3 Jatkuva integraatio ja sen ominaisuudet

Jatkuva integraatio englanniksi on Continuous Integration, lyhyesti CI. CI-ohjelmistojen tehtävä on suorittaa ohjelman jatkuva rakentaminen julkaisuun ja muiden työtehtävien ajaminen siihen liittyen, kuten kirjoitettujen testien ajaminen. Tarjolla on paljon erilaisia CI-ohjelmistoja ja niitä tarjotaan myös valmiiksi asennettuina maksullisina palveluina. On olemassa myös avoimen lähdekoodin palveluita, jotka voi asentaa omalle palvelimelle. Joitain tunnettuja palveluja ovat esimerkiksi Jenkins, BuildBot ja Travis CI.

CI-ohjelmisto voi rakentaa aina lähdekoodin muutoksen jälkeen uuden julkaisun ohjelmistosta, tai hoitaa sen ajoitetusti tietyin väliajoin. Jos julkaisussa missään vaiheessa tulee ongelma, julkaisu epäonnistuu ja mahdollinen sähköpostiviesti asiasta lähetetään projektipäällikölle. Etuna CI-ohjelmistossa on yleisen käsityön vähentäminen ja automaattinen julkaisu aina uusimmasta versiosta. Uusin versio on helposti saatavilla testaajille, asiakkaille tai muille ohjelmistokehittäjille kokeiltavaksi. Ohjelmiston tarkoitus on säästää rahaa ja työaikaa, ja myös helpottaa julkaisuprosessin käsityötä tekemällä kaiken automaattisesti. (Fowler 2006.)

CI-ohjelmisto voidaan laittaa suorittamaan komentorivi-skripti julkaisun yhteydessä. Yleensä tämä ajaa testit ja muut tarvittavat käskyt mitä rakennusvaiheessa tarvitaan. Alla on esitetty esimerkki suoritettavasta bash-skriptistä Django projektin rakentamisen yhteydessä Jenkins-ohjelmalla. Skripti on lainattu lopullisesta toteutuksesta.

```
#!/bin/bash
source /home/puupietari/python-venv/puupietari/bin/activate
cd /var/lib/jenkins/workspace/Puupietari-backend/
pip install -r requirements_build.txt --upgrade
# Export settings file to be used when build is executed.
# Overrides manage.py settings file to use this setting file.
export DJAN-
GO_SETTINGS_MODULE="PuupietariBackend.settings.build_settings"
python manage.py migrate
python manage.py compilemessages
python manage.py collectstatic --noinput
python manage.py jenkins --enable-coverage --pep8-max-line-length 120 --
pep8-exclude migrations
```

Skripti aktivoi Python-virtuaaliympäristön, asentaa tarvittavat paketit ohjelmistolle, ja asettaa ympäristömuuttujia ohjelmalle. Lopuksi se suorittaa Django-ohjelmistolle testit, tekee viimeisimmät tietokantamuutokset ja tämän jälkeen tekee käännökset käyttöliittymän teksteille.

Jos uusi julkaisuprosessi CI-ohjelmistossa epäonnistuu, otetaan viimeisin onnistunut julkaisu käyttöön. Tällöin virhe tapahtuu julkaisuvaiheessa esimerkiksi testien aikana ja se ei riko julkaistua versiota mitenkään. Tämä on erityisen tärkeää esimerkiksi silloin, jos ohjelmistosta laitetaan julkaisun yhteydessä uusin versio

suoraan asiakkaalle. CI-ohjelmisto pitää myös kirjaa ohjelmiston julkaisuhistoriasta.

7.4 Ohjelmiston testaus ja testien kirjoittaminen

Ohjelmiston testauksella tarkoitetaan ohjelmiston kehitystyön vaiheessa käsin kirjoitettujen ohjelmallisten testien suorittamista. Ohjelmoidut testit testaavat kehitteillä olevan ohjelman eri ominaisuuksia ja rajatapauksia että ohjelma toimii halutulla tavalla. Testejä voidaan kirjoittaa mille ohjelmiston osalle tahansa ja testit voi kehittäjä suorittaa omalla koneellaan tai CI-ohjelmisto suorittaa testit julkaisuvaiheessa.

Testien tärkein tehtävä on estää ohjelmiston virheitä mahdollisimman aikaisin ohjelmistokehitysprosessin aikana. Näin viat saadaan aikaisemmin korjattua ja ohjelmisto toimii niin kuin kehittäjät olettavat. Myös kun ohjelma ollaan julkaisemassa asiakkaalle, niin voidaan olla varmempia siitä, että ohjelma toimii asiakkaalla ilman ongelmia ja testaus ei jää asiakkaan harteille. Testien kirjoittaminen myös vähentää kehittäjien käsin tekemää testaamista kehitystyön aikana. Mitä normaalisti testattaisiin ohjelmistossa käsin, voidaan suorittaa ohjelmallisten testien avulla. Koodi ilman testejä voidaan helposti katsoa huonosti ylläpidetyksi. Testien määrä ylläpitää myös kirjoitetun koodin laatua. (Martin Fowler 2006.)

Testien kirjoittaminen vie paljon työaikaa muulta ohjelmistokehitykseltä, mutta työaikaa saadaan säästettyä hyvin paljon kun julkaistun ohjelmiston ylläpito ja päivitys alkaa. Jos kirjoitetut testit ohjelmiston kaikille eri ominaisuuksille ovat kattavat, voidaan paljon vähemmällä työmäärällä päätellä, että ohjelma toimii tarkoitetulla tavalla. Myös käsityön määrä vähenee huomattavasti ja työaikaa säästyy. (Martin Fowler 2006.)

Ohjelmistokehityksessä on tunnetusti eritasoisia testejä. On olemassa järjestelmän pienten osien, esimerkiksi luokkien ja pelkästään niiden osien toimivuuden testaukseen liittyvät testit. Voidaan myös kirjoittaa järjestelmän testejä, joiden tarkoitus on testata jotain järjestelmän toiminnallisuutta kokonaisuudessaan puuttumatta alla toimiviin pienempiin osiin. Testauksessa voidaan myös puhua turvallisuus testauksesta. Näiden testien tarkoitus on toteuttaa tunnettuja hyökkäys tilanteita kuten XSS ja muita vastaavia. Testit kertovat onko ohjelmisto

suojattu näitä vastaan ja toimiiko se halutulla tavalla. Testejä voi myös kirjoittaa ohjelman virheelliseen käyttöön, eli ohjelman sisääntulokenttiin annetaan tarkoituksella väärää tietoa, millä voidaan todeta, että ohjelma toimii halutulla tavalla myös virheellistä tietoa syötettäessä. Alla on esimerkki Django-kehykseen kirjoitetusta testistä.

```
class UrlStates(TestCase):
  fixtures = ["LemonsoftWS_testdata.json"]

  def setUp(self):
    # Set up data for the whole TestCase
    self.ser = Service()
    self.user_based_url = self.ser.settings.user_based_url
    self.data_based_url = self.ser.settings.data_based_url
    self.user_based_url_in_use = self.ser.user_based_url_in_use
    # Wrong Credentials
    self.lemonsoft_settings = LemonsoftSettings()
    self.wrong_username = "wrongusername"
    self.wrong_password = "wrongpassword"

  def test_if_data_based_url_is_working(self):
    """
    Response should be equal to 200
    if the Lemonsoft WSDL url is working
    """
    response = urllib.request.urlopen(self.data_based_url).getcode()
    self.assertEqual(response, 200)
  def test_if_user_based_url_is_working(self):
    """
    Response should be equal to 200
    if the Lemonsoft WSDL-Userservices url is working
    """
    response =      urllib.request.urlopen(self.user_based_url).getcode()
    self.assertEqual(response, 200)
```

Testeissä on yleensä esiasetus-funktio, joka asettaa tarvittavat muuttujat ja muut asetukset testiluokalle. Ylläolevassa tapauksessa funktion nimi on “setUp()”. Tämän jälkeen testiluokan funktiot suoritetaan järjestyksessä, missä jokaisen funktion on tarkoitus testata yksi toiminnallisuus testattavasta osasta. Jokaisen funktion lopussa suoritetaan kysely onko annettu vastaus sitä mitä sen pitäisi olla, jos vastaus ei ole oletettu arvo, testi epäonnistuu. Testien kirjoittamistyyli vaihtelee käytössä olevasta testit ajavasta ohjelmasta ja käytössä olevasta ohjelmointikielestä. Idea on kuitenkin sama.

7.5 Pakettienhallinta ja riippuvuuksien määritys

Ohjelmistokehityksessä kolmannen osapuolen ohjelmistokomponenttien asentamiseen on mahdollisesti käytössä melkeinpä aina jonkinlainen pakettienhallinta ohjelmisto ohjelmointikielestä riippuen. Pakettihallintaohjelmiston tarkoitus on helpottaa ohjelmistokehittäjiä hallinnoimaan ohjelmiston eri riippuvuuksia ja asentamaan niistä halutut versiot. Tarkoitus on vähentää käsin tiedostojen siirtelyä ja helpottaa lisäosien asentamista eri kehittäjien kesken. Usein ohjelmistoprojektin mukana on jonkinlainen riippuvuudet määrittävä tiedosto. Tiedosto kulkee mukana päärepositoryssä muille ohjelmistokehittäjille ja he voivat helposti tämän tiedoston avulla asentaa ohjelmiston riippuvuudet itselleen. Muut asennetut paketit eivät näin kulje päärepositoryyn ja veisi sieltä turhaa tilaa.

Pakettinhallinta ohjelmistolle kerrotaan mikä versio saatavilla olevasta paketista tai komponentista halutaan asentaa. Pakettienhallinta ohjelmistolle kerrotaan haluttu versionumero ja näin ohjelmiston riippuvuudet pysyvät vakioina muillekin kehittäjille ja yhteensopivuus ongelmilta vältytään.

Eri ohjelmointikielillä on olemassa eri pakettien hallintaan tarkoitettuja ohjelmistoja. Alla on esitetty, miltä Python kielen pip-paketinhallintaohjelman riippuvuuksien määrittävän tiedoston sisältö näyttää.

```
Django==1.9.4
django-cors-headers==1.1.0
django-crispy-forms==1.6.0
django-filter==0.12.0
djangorestframework==3.3.2
djangorestframework-jwt==1.8.0
drf-nested-routers==0.11.1
mysqlclient==1.3.6
Pillow==3.0.0
PyJWT==1.4.0
reportlab==3.2.0
suds-jurko==0.6
```

Riippuvuuksien määrittävän tiedoston rakenne vaihtelee käytössä olevan pakettienhallinta-ohjelmiston mukaan.

8 Käytetyt tekniikat ja ohjelmiston rakenne

Nykyisin on ohjelmointikielestä riippumatta tarjolla useita erilaisia sekä eri asioihin keskittyviä kehyksiä ja kirjastoja. Ei ole olemassa vain yhtä ratkaisua tiettyyn ongelmaan, vaan ohjelmistokehittäjä valitsee itse työkalunsa ja voi toteuttaa itse omat ratkaisunsa. Kehyksien komponentit keskittyvät yleensä tietyn ongelman ratkaisemiseen tai tietyn tyyppisten ohjelmien kehittämiseen, kuten esimerkiksi web-ohjelmiston kehittäminen, graafisen käyttöliittymän komponentit tai tietokannan käsittelyyn liittyvät työkalut kuten ORM. Ohjelmistokehitystä voisi toteuttaa ilman kehyksiä tai lisäosia, mutta se lisäisi valtavasti ohjelmistokehitykseen kulutettua aikaa. Yleensä ammattimaisessa ohjelmistokehityksessä käytetään aina apuna kolmannen osapuolen ohjelmistokomponentteja vähentämään toistuvaa työtä ja helpottamaan kehitysprosessia.

Tulevissa kappaleissa käydään läpi ohjelman rakenteen suunnittelua, kielien ja tekniikoiden valintaa sekä erilaisia kehyksiä asiakas- ja serveripuolen ohjelmointiin. Vertaillaan niiden tarjoamia eri ominaisuuksia ja miksi päädyttiin niihin tekniikoihin, joilla ohjelma toteutettiin.

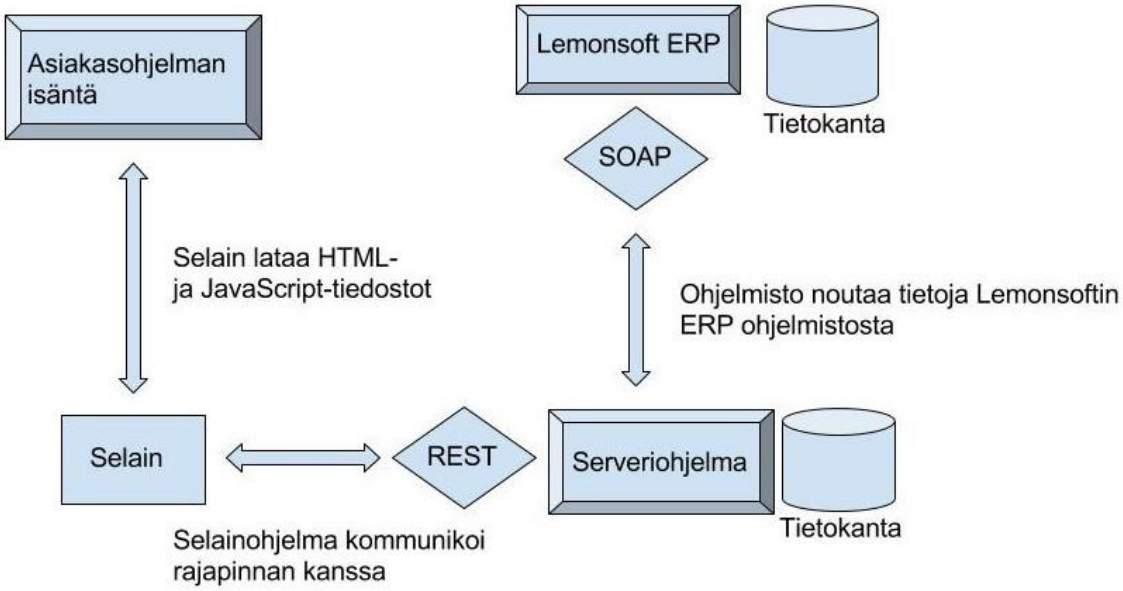
Ajan rajallisuuden sekä lisäosien valtavan tarjonnan vuoksi kaikkia kehyksiä ei pystytä kokeilemaan käytännössä. Tutkimustyötä onkin tehty tekijöiden omien kokemuksien, testaamisen ja muiden käyttäjien kokemusten pohjalta.

8.1 Ohjelmiston pääpiirteinen rakenne

Puupietari Oy:n kanssa tultiin päätökseen, että web-pohjainen ratkaisu sopisi parhaiten yrityksen vaatimuksiin. Ohjelman piti pystyä liittymään SOAP-rajapintaan ja tietokantaa tarvittiin jaettujen resurssien vuoksi. Näiden kriteerien lisäksi, heräsi kysymyksiä siitä, kuinka web-pohjainen ohjelmisto toteutettaisiin. Toimiiko serveripuolen ohjelma myös HTML-sivujen ja JavaScript-tiedostojen isäntänä? Tehdäänkö ohjelmaan ohjelmallista rajapintaa tai onko sille edes tarvetta, vai toteutetaanko asiakaspuolen ohjelma SPA menetelmällä?

Ohjelman rakenteessa päädyimme valitsemaan SPA-mallin asiakaspuolelle ja serveripuoli toteutettaisiin tarjoamaan resursseja REST-rajapinnan yli asiakasohjelmille. Näin molemmat osat ovat kokonaan toisistaan riippumattomia ja tie-

tämättömiä. Serveripuoleen liittyminen mahdollisissa myöhemmin toteutettavissa järjestelmissä voidaan tehdä helposti REST-rajapintaa käyttäen. Rakenne antoi myös mahdollisuuden tehdä kaksi toisistaan riippumatonta ohjelmistokehitysprojektia, joilla molemmilla olisi omat repositorynsa ja projektinhallintasivunsa. Näin asiakas- ja serveripuolen kehittäjät pystyvät helposti erottamaan omat toimialueensa toisistaan. Lisäksi tämä erottelu mahdollistaa omien uudelleen käytettävien pakettien paremman luokittelun. Esimerkiksi asiakaspuolelle toteutettua autentikointipakettia voidaan hyödyntää tulevissa asiakaspuolen projekteissa helposti paketinhallinnalla. Sama pätee myös serveripuoleen. Kuvassa on esitetty suunnitelma järjestelmän toteutuksesta (kuva 14).



Kuva 14. Järjestelmän toteutuskaavio

8.2 Ohjelmointikielen valinta

Mustonen on aiemmin käyttänyt PHP-ohjelmointikieltä muutaman yrityksensä asiakasprojektin toteutuksessa ja niiden ylläpidossa. Sillä on toteutettu integraatio toiseen web-pohjaiseen järjestelmään käyttämällä Curl-kirjastoa kommunikointiin. Kyseisen integraation aikana ongelmia tuli esimerkiksi muistivuotojen kanssa, mitkä eivät johtuneet omasta lähdekoodista (CentOS 2015.). Tämä johti vianetsinnän kannalta suureen määrälliseen etsimiseen ennen kuin syy selvisi. Vika on kuitenkin myöhemmin saatu korjattua ja integraatio on edelleen käytössä. PHP-ohjelmointikieli on tarkoitettu enemmän web-sivujen kuin integraatioi-

den toteutuksiin. Mustonen halusikin kokeilla yrityksensä tulevissa projekteissa uusia tekniikoita ja toteutustapoja.

Molemmat opinnäytetyön tekijöistä suorittivat valinnaisena Python ohjelmointikieli -kesäopintokurssin, josta saatiin peruslähtökohdat Pythoniin ja tässä projektissa käytettyyn Django-kehykseen. Kurssin jälkeen Mustonen käytti myös Pythonia osassa koulutehtävistä, joissa ohjelmointikielen sai päättää itse, sekä myös Django-kehystä joidenkin web-ohjelmien toteuttamisessa. Uuden halu ja innostus, kokemukset sekä hyvä palaute muilta ohjelmointikieleen tutustuneilta kollegoilta vaikuttivatkin päätökseen ottaa Python serveripuolen ohjelmointikieleksi. Python tunnetaan myös monikäyttöisenä ohjelmointikielenä, eli sillä voidaan tehdä ohjelmia melkein mihin tarkoitukseen tahansa. Tällöin sitä voidaan käyttää myös muissa Asmeco:n asiakasprojekteissa, joihin ei tule web-toteutusta.

PHP- ja Python-ohjelmointikielien lisäksi, projektissa käytettäväksi harkittiin myös niin kutsuttua Ruby on Railsiä. Termi sisältää Ruby-ohjelmointikielen, jonka kanssa käytetään Rails-kehystä. Näihin opinnäytetyön tekijöillä ei ollut kuitenkaan mitään kosketuspintaa, mikä helpotti Pythonin ja Djangon valitsemisessa. On tuotava esille, että ohjelmointikielen valitsemiseen ei ole olemassa vain yhtä ja oikeaa tai ylitse muiden olevaa ratkaisua. Ohjelmisto olisi ollut toteutettavissa monella eri ohjelmointikielellä, useille niistä löytyvän kirjasto- ja kehystarjonnan ollessa erittäin kattavaa. Pythonin valintaan vaikuttikin suuresti yrityksen tulevat asiakasprojektit, kielen pieni opiskelukynnys ja sen monikäyttöisyys. Selaimen ohjelmointikieleksi valittiin suoraan JavaScript. JavaScriptiä generoivaa kieltä, kuten esimerkiksi Dart:ia ei kuitenkaan haluttu käyttää. Pelkkä JavaScript-ohjelmointikieli on todettu hyväksi tavaksi pitää lähdekoodi siistinä ja tällöin myös tarjolla oleva kehysten määrä on kattavampi.

8.3 Serveripuolen tekniikat

8.3.1 Ohjelmisto

Serveripuolen toteutuksen ohjelmointikieleksi valittiin siis Python ja ohjelmaan haluttiin tehdä REST-rajapinta asiakasohjelmille. Ohjelmiston valmistamiseen

haluttiin käyttää tarjolla olevia valmiita komponentteja, joiksi pääasiassa valikoituivatkin Django ja siihen saatavilla oleva Django Rest Framework (DRF).

Django-kehys on noussut suureen suosioon web-kehityksen saralla Python-ohjelmointikieltä käytettäessä. Se on tarkoitettu web-applikaatioiden nopeaan kehitykseen selkeällä ohjelmiston rakenteella, mikä pyrkii noudattamaan DRY-tapaa (Don’t Repeat Yourself). Django itsessään ei sisällä funktionaalisuutta ohjelmallisen rajapinnan tekemiseen, mutta siihen on saatavilla asiaan tarkoitettu DRF. DRF:ää käytetään erittäin paljon ohjelmallisten rajapintojen toteutuksessa Djangon kanssa, minkä vuoksi siitä löytyikin tutkimusta tehdessä paljon positiivia kommentteja ja kokemuksia. Tämä vaikuttikin päätökseen valita DRF.

DRF-lisäosa voi käyttää Djangon ORM-malleja rajapintakohtien luomiseksi ja näin ollen nopeuttaa kehitystä huomattavasti. Django ja DRF tekevät suuren määrän automaattista ohjelmointityötä verrattuna perinteisiin toteutusmalleihin. Ne on myös suunniteltu niin että ohjelmistokehittäjä voi helposti itse lisätä omaa toiminnallisuutta käytettäväksi kehyksen tukena. Kaiken lisäksi molempiin on tarjolla kattavat dokumentaatiot. Ne sisältävät koodi-esimerkkejä, jotka löytyvät helposti kehysten omilta internetsivuilta. Djangoa ja DRF:ää käyttää myös suuri määrä ihmisiä, jolloin useimpiin ongelmatapauksiin apu on helposti saatavilla esimerkkien kanssa.

Djangon ORM-mallit toimivat monen tietokantaohjelman kanssa, joihin lukeutuvat esimerkiksi MySQL, Oracle, SQLite ja PostgreSQL. Django hoitaa tämän erillisillä lisäosilla, joista osa tulee Djangon mukana ja joista toiset täytyy asentaa erikseen. (Django 2016).

8.3.2 Tietokanta

Tietokannan valitsemisessa ohjelmaan ei käytetty paljoa aikaa. MySQL tietokanta on ollut käytössä pitkään Asmecon muissa projekteissa, joten siihen on kertynyt paljon kokemusta. Myös koulun projekteissa on usein käytetty MySQL-tietokantaa PHPMyAdmin ohjelmiston kanssa, mikä todettiin hyväksi ratkaisuksi tämänkin ohjelman tapauksessa.

8.4 Asiakaspuolen tekniikat

Asiakaspuolen ohjelmointiin pelkällä JavaScriptillä on tarjolla monia erilaisia kehyksiä ja kirjastoja paljon enemmän verrattuna esimerkiksi serveripuolella käytetylle Pythonille. Asiakaspuolelle haluttiin kehittää SPA-ohjelmisto, joten kehyksen ominaisuuksiin tuli kuulua osoitteiden reititys yhdellä sivulla, sekä sisällön dynaaminen vaihtaminen ja lataaminen. Lisäksi tuki REST-rajapinnan käyttämiseen tulisi myös olla saatavilla. Käyttötapaukseen sopivina isoina kehyksinä tulivat heti esiin Angular, Ember, Backbone ja Knockout. Näistä Backbone on enemmänkin pieni kirjasto, joka antaa perusmäärittelyjä ORM:n hoitamiseen ja osoitteiden reititykseen. Angular, Ember ja Knockout ovat enemmän kokonaisvaltaisia kehyksiä ja ne tarjoavat paljon enemmän funktionaalisuutta, kuten kaksisuuntaisen tiedon sitomisen muodossa(two-way data binding).

8.4.1 Kehyksen valitseminen

JavaScript projektiin kehyksen valitsemista voidaan helpottaa TodoMVC-sivustolla (TodoMVC 2016). Sivustolle on toteutettu kaikilla tunnetuilla kehyksillä sama pieni ohjelmisto. Sivustolla voi helposti verrata, mitä saman asian tekeminen vaatii eri kehyksiltä ja miltä lähdekoodi näyttää eri tapauksissa. Lähdettä on käytetty valinnan tukena aiempien kokemuksien lisäksi.

Mustosella oli jo paljon aikaisempaa kokemusta Backbone- ja Knockout-kehyksistä aikaisempien projektien pohjalta. Knockout ei tue reititystä valmiina, eli se olisi pitänyt toteuttaa siihen itse erilaisilla komponenteilla (Weinstock-Herman 2013). Backbone taas tarjoaa valmiit työkalut reititykseen ja SPA sivun tekemiseen ja tällä saralla Backbone tarjosi enemmän kuin ohjelmistossa oli tarvetta. Sen sijaan Knockout tarjosi kaksisuuntaisen datan sitomisen käyttöliittymässä, mikä taas Backbone:sta puuttuu kokonaan. Knockout ja Backbone ovat molemmat suosittuja kehyksiä tiettyihin tarkoituksiin, ja toinen hoitaa joitakin asioita toista paremmin. On kuitenkin olemassa kehys nimeltään Knockback, joka pyrkii ottamaan kummastakin, niiden parhaat ominaisuudet ja yhdistämään ne (Malakoff 2016). Tästä kehyksestä ei kuitenkaan ollut aikaisempaa kokemusta ja projektin suuruuden takia tarve monipuolisemmalle kehykselle oli tarpeellinen. Ember ja Angular ovat kokonaisvaltaisia kehyksiä, mitkä tarjoavat

paljon enemmän funktionaalisuutta kuin Backbone tai Knockout. Tämän takia valinta kallistui enemmän näiden kahden kehyksen välille.

Angularin ja Emberin väliltä valitsimme ensin Ember-kehyksen sen ORM-mallien takia ja sille löytyi myös hyvä tuki REST-rajapinnan käsittelyyn. Angular ei sisällä ORM-malleja ollenkaan, kun taas projektin tarkoituksiin Ember-kehyksessä voitiin käyttää Ember Data-kirjastoa. Ember Data on selaimessa sijaitseva säilö objekteille, mitä kysellään serveripuolelta. Saman muuttumattoman tiedon moneen kertaan lukemista voidaan nopeuttaa, tekemällä se suoraan selaimen säilöstä eikä serveriltä. Myös TodoMVC-sivustolla Emberistä sai lähdekoodin perusteella paremman kuvan kuin Angularista. Asiakaspuolen ohjelmointia aloitettiinkin tekemään Emberillä.

8.4.2 Emberin ongelmat

Alunperin Ember eli pelkkä JavaScript kirjasto ja nyt uusien päivitysten myötä, sen mukana tulee myös komentorivityökalu projektin hallitsemiseksi (Ember-CLI). Työkalulla pystytään hallitsemaan Ember-projektin luominen, sen rakentaminen ja testaaminen. Emberiä ehdittiin käyttää projektin alussa jonkin jo jonkin aikaa, kunnes ohjelmalla aloitettiin työstämään vähän vaativampaa logiikkaa, kuten rajapinnan JWT-kirjautumista. Aikaa kului paljon oman funktionaalisuuden tekemiseen ja sen selvittämiseen, miksi jokin ei toiminut kuten oletettiin. Syyksi selvisi Emberin monet muutokset eri päivitysten välillä sekä näiden huono dokumentointi. Muiden tekemät kirjastot ja esimerkit eivät olleet enää yhteensopivia eri Ember versioiden kanssa. Hyvä esimerkki tästä oli JWT lisäosa, joka ei silloin tukenut uusinta Ember-CLI versiota.

Lisäksi ongelmaksi tuli Ember-CLI -kokonaisuuden raskas käytettävyys. Komentorivityökalussa tuli mukana projektinajotyökalut. Tämä sisälsi myös automaattisen lataamisen jos lähdekoodin tiedostoihin tuli muutoksia. Ensimmäinen projektin ajon alustaminen kesti todella kauan ja sama toistui myös automaattisessa latauksessa. Automaattisen latauksen ongelma oli, että jos projektin alustus-ohjelmaa tai sen asetuksia muutettiin, niin automaattinen lataaminen kaatui. Ongelma oli korjattavissa käynnistämällä automaattinen lataaminen uudestaan ja odottaa koko prosessi alusta loppuun, mikä vei paljon työaikaa.

Ember-kehyksen JavaScript-koodi kirjoitetaan noudattamalla ES6-syntaksia, joka oli työn tekemisen aikaan osa juuri tullutta JavaScript päivitystä. Yleisen tuen takia Ember-CLI käänsi tällä kirjoitetun lähdekoodin vanhempaan ES5-muotoon, minkä pystyi suorittamaan helposti kaikilla selaimilla. Prosessi teki kuitenkin selaimella lähdekoodin vianetsinnästä hankalaa, koska koodi ei joistakin kohdista enää muistuttanut alkuperäistä. Ember-CLI sievensi aina koodin yhdeksi tiedostoksi ajoa varten, milloin oman koodin löytäminen tiedostosta oli vaikeaa. Selaimeen oli saatavilla ohjelman ongelmien etsintään tarkoitettu lisäosa. Lisäosassa ongelmana kuitenkin oli, että Ember-ohjelman tuli onnistua käynnistymisessä kokonaan, että sitä pystyttiin käyttämään. Tämä teki joidenkin alustuskoodien vian etsimisestä vaikeaa. Lisäksi selaimen ajovirheiden viestit eivät kertoneet tarkasti, missä vika lähdekoodin sisällä sijaitsi. Virheviestien linkit osoittivat suoraan Emberin lähdekoodiin, eivätkä siihen osioon, mihin oma koodi oli kirjoitettu.

Ember sisälsi kehyksenä paljon lupaavia komponentteja, ja hitausongelmat olisivat olleet todennäköisesti ratkaistavissa nopeammalla koneella. Käytimme Emberin kanssa projektiin noin 40 tuntia ja työmäärään nähden saimme liian vähän aikaiseksi. Tämän jälkeen päätimme vaihtaa Angulariin, projektin ollessa kuitenkin sen verran alussa että vaihto oli vielä mahdollista. Emberistä oli tulossa uudempi versio 2.x ja jo kehitys hetkellä huomattiin, että vanhempiin versioihin nähden dokumentaatiot olivat paljon kattavammat. Emberiä voisi olla parempi kokeilla myöhemmin uudestaan, kunnes se on päässyt eteenpäin kehityksessä ja on vakaampi lisäosien suhteen.

8.4.3 Angular

Angularissa ei ollut komentorivityökaluja kuten Emberissä ja se oli pelkkä JavaScript-kirjasto. Tämän avuksi projektiin päätettiin ottaa Grunt-, Bower- ja Yeoman-ohjelmistot. Työkalujen avulla projekti saatiin nopeasti käyntiin ja tarvittu funktionaalisuus toteutettiin. Grunt tarjosi myös samaa toiminnallisuutta kuin Ember-CLI, esimerkiksi projektin ajamiseen ja rakentamiseen liittyen. Isona erona Gruntissa oli prosessien valtava nopeusero Emberiin nähden. Myös kirjastojen saatavuus oli paljon parempi Angularille sekä muiden ihmisten tekemiä esimerkkejä löytyi reilusti enemmän. Angularilla oli paljon suositumpi kehittäjä-

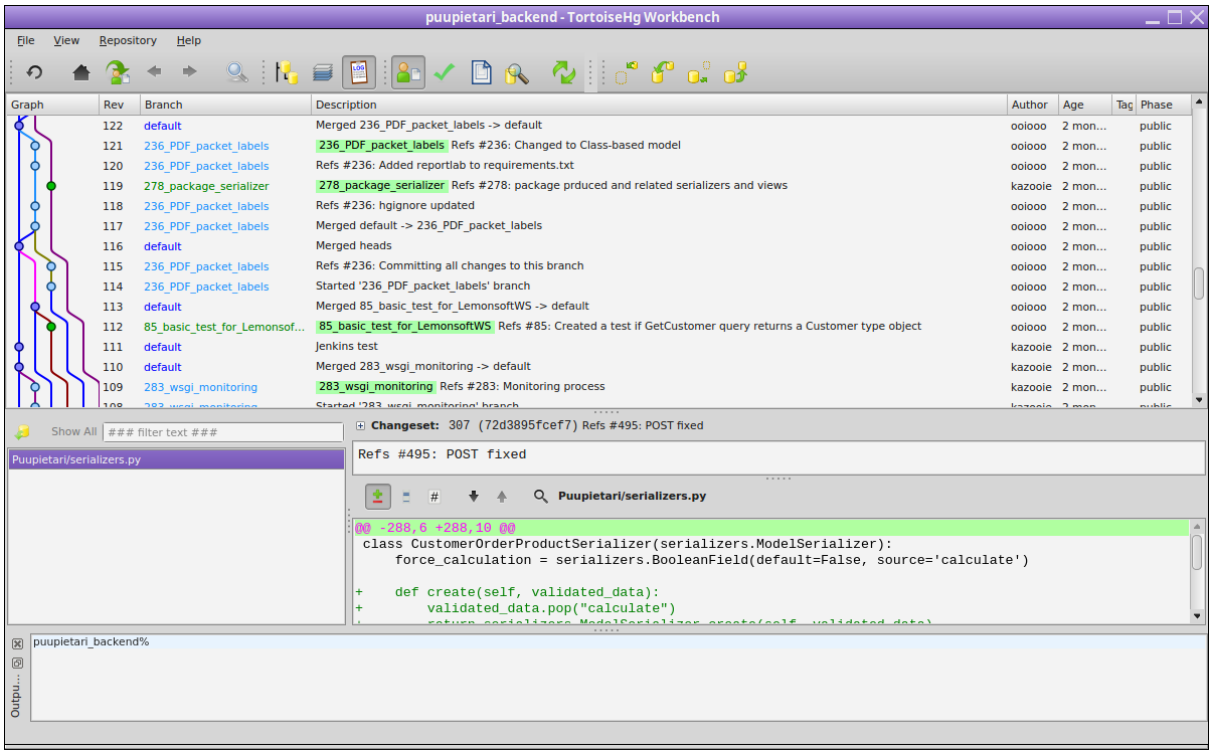
kunta kuin Emberillä ja siitä oli iso apu kehitystyön tueksi. Myöskin Angular-projektin lähdekoodin vianetsintä oli paljon helpompaa, koska kaikkea lähdekoodia ei pakattu vain yhteen tiedoston. Lisäksi JavaScript kirjoitettiin Angularia käyttäessä perinteisellä ES5-syntaksilla ilman ulkopuolista käännösprosessia. Näiden piirteiden vuoksi, Angular todettiin paremmaksi vaihtoehdoksi projektin kehitystyön kannalta.

9 Käytetyt ohjelmistokehitystyökalut

Tämän osion kappaleissa käydään läpi mitä työkaluja ja ohjelmia projektin kehitystyössä on käytetty ja kuinka niitä on hyödynnetty kokonaisuudessaan. Osioissa kerrotaan myös miten eri ohjelmat ovat helpottaneet kehitystyötä ja kuinka se poistaa toistuvia rutiineja.

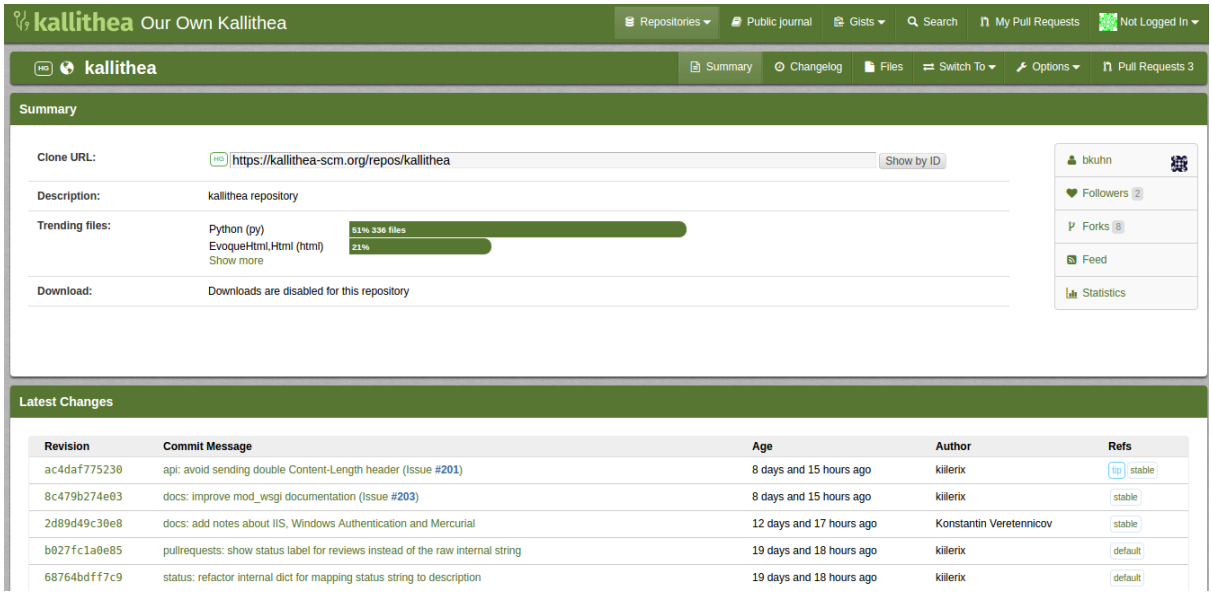
9.1 Versionhallinta

Kehitystyötä eri tekijöiden välillä hallittiin Mercurial versionhallinnan komentorivityökalulla, lyhennettynä Hg. Lyhenne tulee kemiasta elohopean merkinnästä. Lisäksi työtä helpottamaan asennettiin TortoiseHg, mikä on graafinen käyttöliittymä-ohjelmisto. TortoiseHg:n ollessa käytössä, komentorivityökalua ei tarvinnut käyttää ollenkaan. Kuvasta nähdään TortoiseHg-ohjelman graafinen käyttöliittymä Linux-alustalla (kuva 15).



Kuva 15. TortoiseHg graafinen käyttöliittymä linuxilla

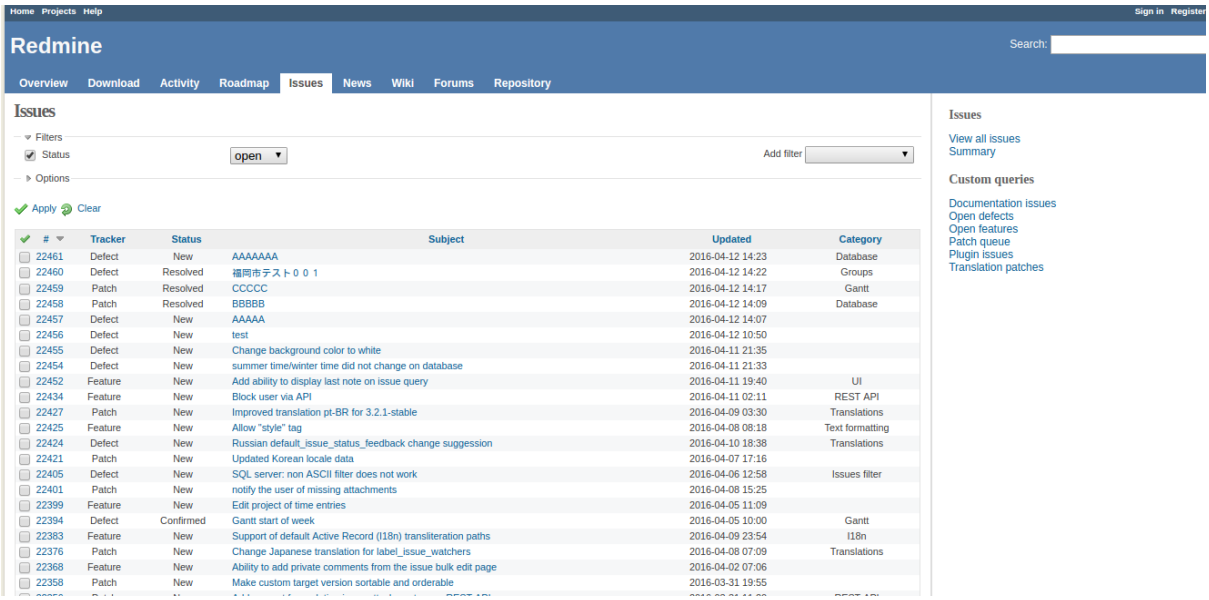
Lähdekoodin sijoituspaikkana käytettiin serverille asennettua Kallithea-versionhallintaohjelmistoa. Kallithea tukee Git- ja Mercurial-versionhallintatyökaluja ja se on monipuolinen ja ilmainen avoimen lähdekoodin ohjelmisto, käyttäjien ja repositoryjen hallintaan. Kuvakaappauksessa on esitetty miltä Kallithean käyttöliittymä selaimessa näyttää (kuva 16).



Kuva 16. Kallithean käyttöliittymä (Kallithea, 2016)

9.2 Projektinhallinta

Projektinhallintaan käytimme serverille asennettua Redmine-ohjelmistoa. Redmine on ilmainen avoimen lähdekoodin projektinhallintaohjelmisto, jonka voi asentaa omalle palvelimelle. Redmine sisältää käyttäjienhallinnan ylläpitäjille ja sillä voidaan hallita montaa erilaista projektia. Kuvakaappaus esittää miltä Redminen käyttöliittymä näyttää selaimessa (kuva 17).



Kuva 17. Redminen käyttöliittymä selaimessa (Redmine, 2016)

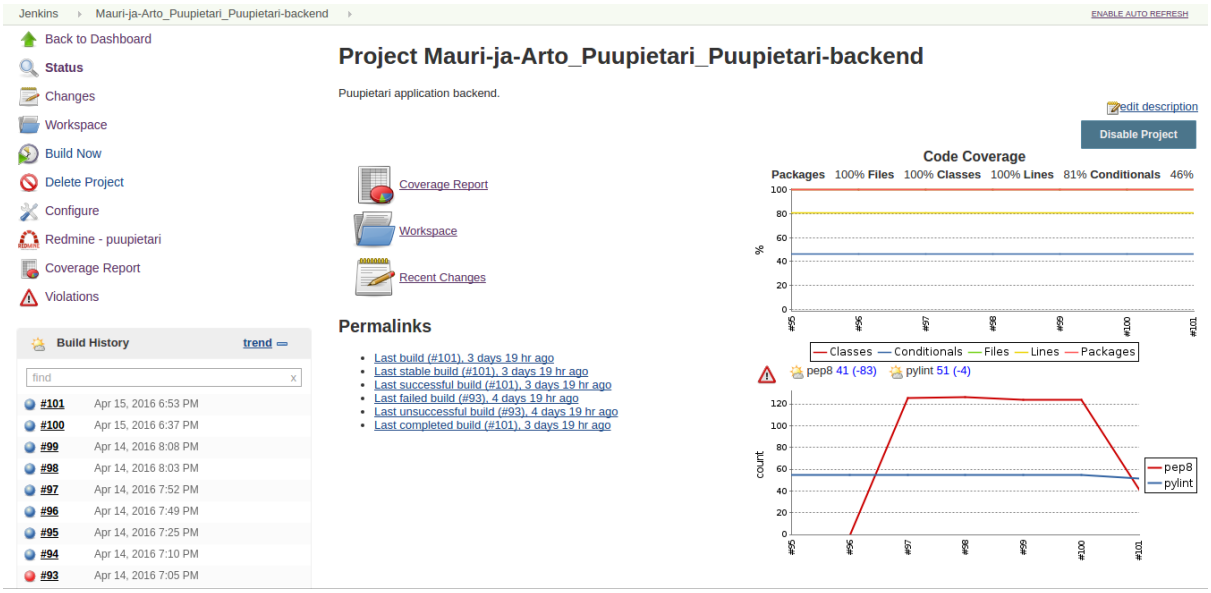
Redmine-ohjelmistoa käytettiin projektissa työajan mittaamisessa, aikataulun suunnittelussa, sekä tikettien tekemisessä ja niiden seurannassa. Tärkein työkalu Redminessä oli sen tikettijärjestelmä. Sillä pystyttiin suunnittelemaan työhön liittyvät tehtävät etukäteen ja vältettiin, ettei samaa työtä tehnyt kaksi eri henkilöä. Menetelmä noudattaa agile-kehitysmenetelmiä.

9.3 Jatkuva integraatio

9.3.1 Jenkins

Automaattisten julkaisujen rakentamiseen käytettiin palvelimelle asennettua Jenkins ohjelmistoa. Jenkins on Java-pohjainen avoimen lähdekoodin CI-ohjelmisto. Jenkinsiin on saatavilla useita erilaisia lisäosia helpottamaan työtä ja

lisäämään Jenkinsin toiminnallisuutta. Kuvakaappauksessa nähdään projektissa käytetyn Jenkins-ohjelman käyttöliittymä (kuva 18).



Kuva 18. Jenkinsin käyttöliittymä

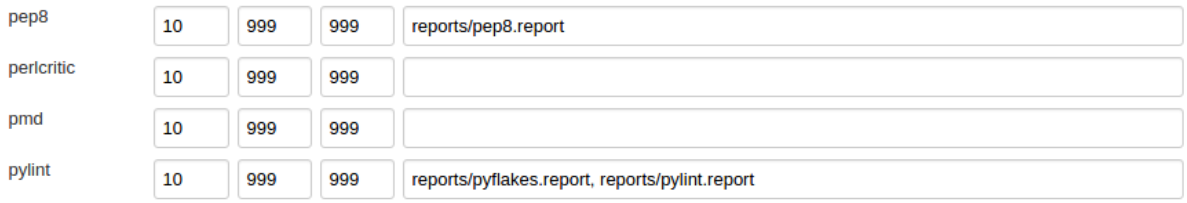
Tärkeä osa ohjelmiston kehitystyötä oli rakentaa aina uusin julkaisu tiettyyn osoitteeseen palvelimelle, että se olisi saatavilla kaikille ohjelmistokehittäjille. Rakennusprosessi suoritettiin, kun päärepositoryn päähaaraan tuli uusi lähdekoodimuutos. Kallithea antoi merkin Jenkinsille, joka rupesi heti suorittamaan julkaisuprosessia. Jenkins suoritti julkaisut molemmista ohjelmiston osista, joita olivat siis serveri- ja asiakaspuoli. Näin Parikalla oli aina saatavilla uusin ja viimeisin toimiva versio REST-rajapinnan käyttämiseen ja asiakaspään ohjelman kehittämiseen. Hänen ei tarvinnut suorittaa serveripuolen ohjelmaa omalla työkoneellaan, vaan hän pystyi käyttämään Jenkinsin rakentamaa julkaisua.

Jenkins-ohjelmisto suoritti ohjelmalle kirjoitetut ohjelmalliset testit automaattisesti jokaisen julkaisun yhteydessä, mikä osaltaan takasi koodin laatua ja toimivuutta. Jos julkaisu epäonnistui testien takia, viat saatiin selville jo aikaisessa vaiheessa kehitystyötä ja ne voitiin korjata nopealla aikataululla.

9.3.2 PEP8 ja PyFlake

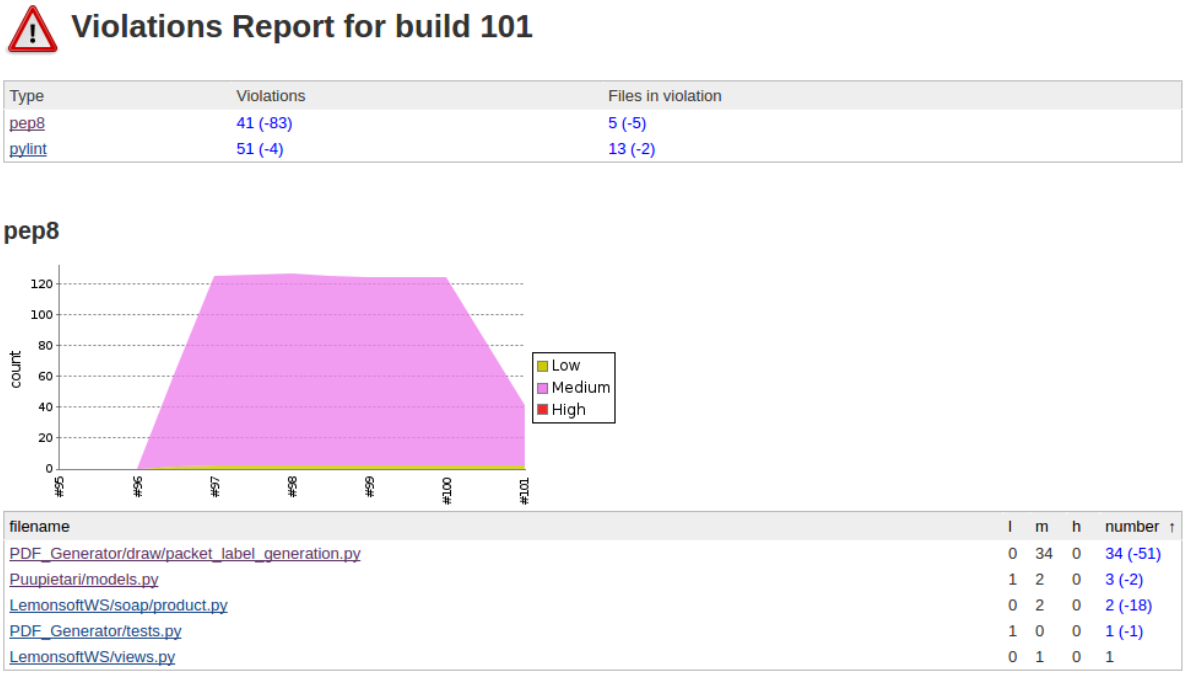
Django-kehykseen on saatavilla “django-jenkins”-lisäosa, joka on paketti helppoon ja suoraan integraatioon Jenkins-ohjelmiston kanssa. Lisäosa ajoi testit

sekä PEP8- ja PyFlake-tarkastustoiminnoille. PEP8 on Pythonille suunniteltu tarkka formatoinnin ohjenuora. PyFlake on taas koodivirheiden tarkistukseen tarkoitettu ohjelma. Molemmat parsivat lähdekoodin ja tekivät tulosten pohjalta raporttitiedostot. Nämä raporttitiedostot voidaan antaa Jenkinsiin löytyvälle “Violations plugin”-lisäosalle, joka tekee niiden pohjalta näyttävät ja monipuoliset graafit sekä lähdekoodin virheiden merkinnät. Kuvassa 19 nähdään, miltä lisäosan asetukset projektissa näyttävät.



Kuva 19. Lisäosien asetukset

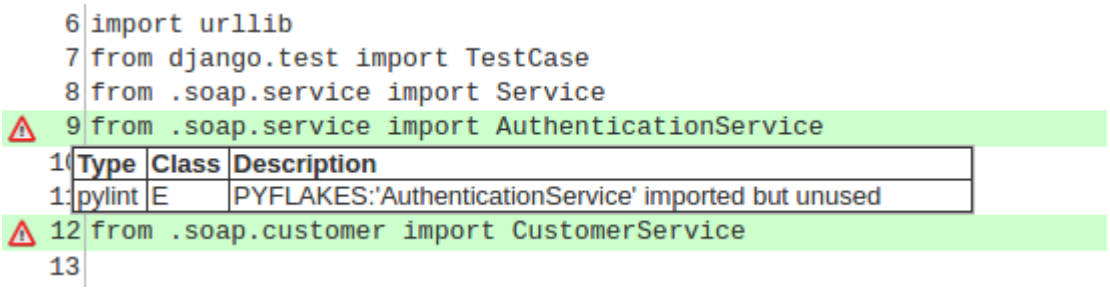
Kuvassa 20 esitetään miltä graafi ja listaus PEP8-virheitä sisältävistä tiedostoista näyttää. Samanlainen näkymä on saatavilla myös PyFlake-ohjelman virheille.



Kuva 20. PEP8-raportti

Jenkins pystyi myös näyttämään rivit missä osassa lähdekoodia ongelma oli ja minkä tyyppinen ongelma oli kyseessä. Kuvassa on esitetty “imported but unused”-virheviesti PyFlake ohjelman raportista (kuva 21). Näiden virheentar-

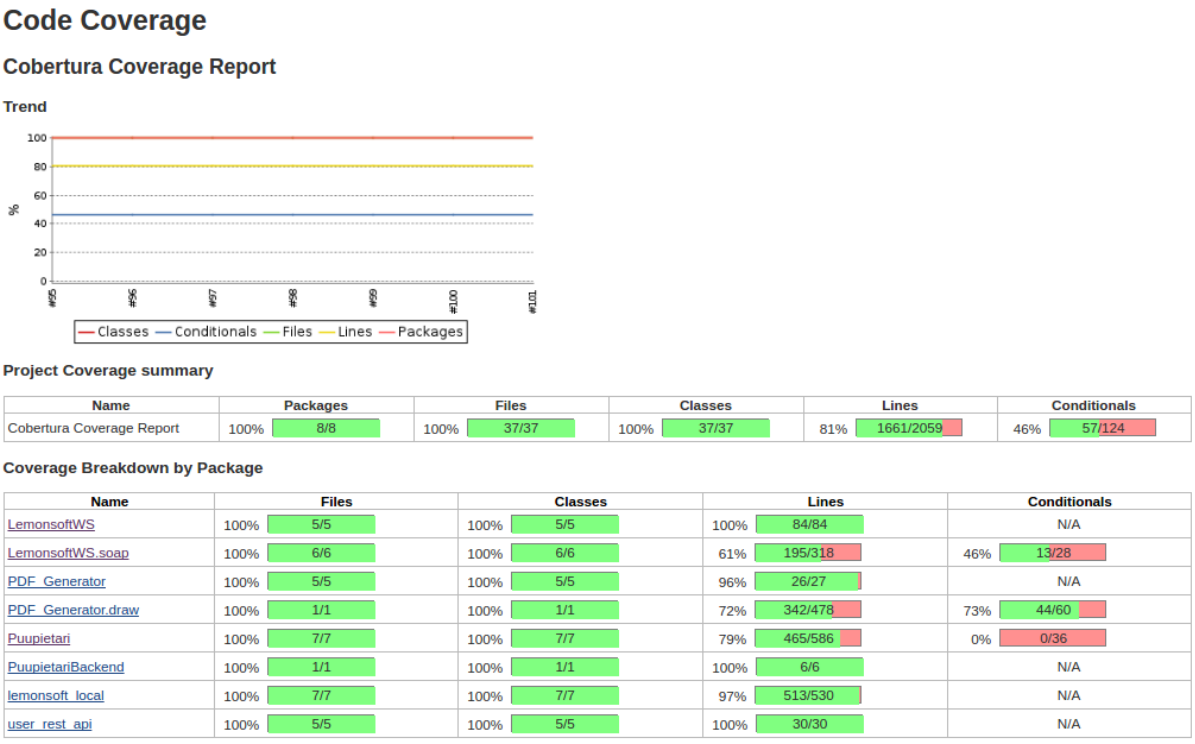
kistustyökalujen avulla lähdekoodin laatua saatiin paljon nostettua ja pienet, kehittäjiltä huomaamatta jääneet virheet tulivat myös esille.



Kuva 21. PyFlake-raportti

9.3.3 Testien kattavuus -raportti

Jenkinsin “Cobertura Plugin”-lisäosa suorittaa testien kattavuus -raportin lähdekoodille. Django-projektiin asennetulle “django-jenkins”-lisäosalle tulee antaa komentorivillä “--enable-coverage”-optio, jolloin projektin kattavuusraportit luodaan. Kuvassa 22 nähdään miltä raportin graafi näyttää ja mitä testien prosentuaalinen kattavuus on. Ohjelma erittelee myös prosentuaalisen kattavuuden niin tiedostojen, luokkien, rivien kuin myös ehtolauseiden mukaan.



Kuva 22. Koodin kattavuus -raportti

Tiedostoja lähemmin tarkastelemalla nähdään, mitkä rivit koodista on testattu. Kuvassa on esitetty SOAP rajapinnan oman “Service”-luokan rakentajan testien kattavuus (kuva 23). Punaisella taustalla olevia rivejä ei testata, kun taas vihreätaustaiset rivit ovat katettuja testeillä. Näin testien kirjoittajan on helppo nähdä mitä osia ohjelmasta ei ole vielä testattu ja mille ominaisuuksille testejä voisi vielä kirjoittaa.

```
15 1 from LemonsoftWS.models import LemonsoftSettings, LemonsoftService
16 1 from LemonsoftWS import exceptions
17 1 from .customer import CustomerService
18 1 from .authentication import AuthenticationService
19 1 from .product import ProductService
20 1 from .factory import Factory
21
22
23 1 class Service(object):
24
25 1     def __init__(self, **kwargs):
26 1         self.settings = LemonsoftSettings.objects.first()
27 1         self.services = LemonsoftService.objects.first()
28 1         self.user_based_url_in_use = kwargs.get('user_based', self.settings.user_based_url_in_use)
29 1         if not self.settings or not self.services:
30 0             raise exceptions.DatabaseValueError(
31                     _("Lemonsoft database models are missing settings or services data")
32                 )
33 1         if settings.SOAP_API_DEBUG:
34 0             logging.basicConfig(level=logging.INFO)
35 0             logging.getLogger('suds.client').setLevel(logging.DEBUG)
36 1         cache = ObjectCache(days=settings.CACHE_INTERVAL_DAYS)
37 1         if settings.CACHE_LOCATION is not None:
38 0             cache.setlocation(settings.CACHE_LOCATION)
39 1         if self.user_based_url_in_use:
40 1             self.client = Client(self.settings.user_based_url, cache=cache)
41          else:
42 0             self.client = Client(self.settings.data_based_url, cache=cache)
43 1         self._collect_services(**kwargs)
44
45 1     def __str__(self):
46 0         return self.client.__str__()
47
48 1     def _collect_services(self, **kwargs):
49 1         self.authentication = AuthenticationService(self, **kwargs)
50 1         self.customer = CustomerService(self, **kwargs)
51 1         self.product = ProductService(self, **kwargs)
52 1         self.factory = Factory(self, **kwargs)
```

Kuva 23. "Service"-luokan kattavuus-raportti

9.4 Pakettienhallinta

9.4.1 Python

Python-ohjelman pakettienhallintaan käytimme Pythonin mukana tulevaa pip-komentoriviohjelmaa. Pip-ohjelman lisäksi projektissa käytettiin “requirements.txt”-tiedostoa, joka määritti serveripuolen projektin riippuvuudet. Tiedosto kulki aina projektin viimeisimmän version mukana muille ohjelmistokehittäjille, mistä he pystyivät helposti tarkastamaan ja tämän jälkeen asentamaan ohjelmiston tarvitsemat riippuvuudet omille laitteilleen. Myös Jenkins-ohjelmisto käytti tiedostoa julkaisuohjelmassa riippuvuuksien asentamisessa. Kun projektiin lisättiin uusia riippuvuuksia, ne tuli päivittää myös “requirements.txt”-tiedostoon. Näin muut kehittäjät ja Jenkins-julkaisuun saatiin helposti asennettua uudet tar-

vitut paketit. Riippuvuudet määrittävän tiedoston saa ulos pip-ohjelmasta komennolla “pip freeze > requirements.txt” ja tiedoston määrittävien riippuvuuksien asentaminen tapahtui komennolla “pip install -r requirements.txt”.

Pythonin virtuaaliympäristöt mahdollistavat erilliset toisistaan riippumattomat Python-asennukset. Toisten virtuaaliympäristöjen paketit eivät liity mitenkään toisiinsa ja eri virtuaaliympäristöissä voi olla käytössä myös omat Python-versiot. Samalla työkoneella on esimerkiksi mahdollista ajaa vanhempaa ja uudempaa Python-kääntäjää toisistaan riippumatta.

9.4.2 NPM

Node Package Manager on NodeJS-alustan pakettienhallintaan tarkoitettu komentoriviohjelma. NPM:ää käytettiin toteutettaessa Angular-kehystä käyttävää asiakaspuolen ohjelmistoa. NPM sisältää myös oman riippuvuuksien määrittävän “package.json”-tiedoston. Teksti alla esittää miltä osa “package.json” tiedostosta näyttää.

```
{
 "name": "puupietarifrontend",
 "private": true,
 "devDependencies": {
   "autoprefixer-core": "^5.2.1",
   "grunt": "^0.4.5",
   "grunt-angular-templates": "^0.5.7",
   "grunt-concurrent": "^1.0.0",
   "grunt-contrib-clean": "^0.6.0",
   "grunt-contrib-concat": "^0.5.0",
   "grunt-contrib-connect": "^0.9.0",
   "grunt-contrib-copy": "^0.7.0",
   "grunt-contrib-cssmin": "^0.12.0",
   "grunt-contrib-htmlmin": "^0.4.0",
   "grunt-contrib-imagemin": "^1.0.0",
   "grunt-contrib-jshint": "^0.11.0",
   "grunt-contrib-uglify": "^0.7.0",
   "grunt-contrib-watch": "^0.6.1",
   "grunt-filerev": "^2.1.2",
   "grunt-google-cdn": "^0.4.3",
   "grunt-jscs": "^1.8.0",
   "grunt-karma": "*",
   "grunt-newer": "^1.1.0",
   "grunt-ng-annotate": "^0.9.2",
   "grunt-postcss": "^0.5.5",
   "grunt-svgmin": "^2.0.0",
```

```
    "grunt-usemin": "^3.0.0",
    "grunt-wiredep": "^2.0.0",
    "jit-grunt": "^0.9.1",
    "jshint-stylish": "^1.0.0",
    "karma-jasmine": "*",
    "karma-phantomjs-launcher": "*",
    "time-grunt": "^1.0.0"
  }
}
```

NPM ei asenna niinkään kehitteillä olevan ohjelman kirjastoja vaan NodeJS-serverin lisäosa-paketteja, mitä käytetään asiakaspuolen ohjelmiston kehityksen apuna, eikä niitä myöskään asenneta ohjelmiston julkaisuserverille. Paketit ovat yleensä vain kehitystyötä varten. Jos NodeJS-alustalle kehitettäisiin serveripuolen ohjelmaa, paketteja tarvittaisiin myös julkaisuvaiheessa. Tämän projektin tapauksessa, NodeJS-alustaa käytettiin vain apuna kehitystyössä ja sillä ajettiin ohjelmia Grunt- ja Yeoman-ohjelmia.

9.4.3 Bower

Asiaskaspuolen ohjelmistokehityksen pakettienhallintaan käytettiin Bower-komentoriviohjelmistoa. Bower ylläpiti asiakaspuolen ohjelmiston riippuvuuksia ”bower.json”-tiedostolla. Teksti alla antaa esimerkin, miltä osa “bower.json” tiedostosta näyttää asiakaspään projektissa.

```
{
  "name": "puupietari-frontend",
  "version": "1.0.0",
  "dependencies": {
    "angular": "^1.4.0",
    "bootstrap": "^3.2.0",
    "angular-animate": "^1.4.0",
    "angular-aria": "^1.4.0",
    "angular-cookies": "~1.5.3",
    "angular-messages": "^1.4.0",
    "angular-resource": "^1.4.0",
    "angular-route": "^1.4.0",
    "angular-sanitize": "^1.4.0",
    "angular-touch": "^1.4.0",
    "angular-bootstrap": "~0.14.3",
    "js-data-angular": "~3.1.0",
    "js-data": "~2.9.0",
    "restangular": "~1.5.2"
  },
  "devDependencies": {
```

```
  "angular-mocks": "^1.4.0",
  "angular-jwt": "^0.0.9"
 }
}
```

Tiedosto on hyvin samanlainen sisällöltään kuten NPM:n ”package.json” tiedosto. Sitä voi muokata käsin jos tulee tarve tehdä tarkkoja muutoksia ilman Bower-komentorivityökalua, joka voi erotella riippuvuudet kehitys- ja julkaisuversion mukaan. Kehitystyön tukena käytettyjä paketteja ei tarvita asiakkaalle toimitettavassa julkaisuversiossa, vaan siihen kuuluvat ainoastaan julkaisuun ja ohjelman toimivuuteen tarvittavat paketit.

9.5 Asiakaspuolen kehityksen muut työkalut

9.5.1 Grunt

Grunt on JavaScript-tehtävien suorittaja. Gruntin tarkoitus lyhyesti on projektissa toistuvien tehtävien automatisointi ja työn helpottaminen. Grunt on komentorivityökalu, joka asennetaan NPM-paketinhallinnan kautta ja se konfiguroidaan projektin tarpeisiin sopivaksi. Gruntin vahvuus on sen helppo muokattavuus kaikenlaisten projektien tarpeisiin sekä sen laaja lisäosavalikoima. Grunt voidaan asettaa tekemään toistuvia työvaiheita kehitystyön aikana, kuten ajamaan testejä, minimoimaan JavaScript- ja CSS-tiedostot julkaisukansiossa, suorittamaan lähdekoodin tyyli-tarkastukset ja käynnistämään testiserveri projektille.

Grunt konfiguroidaan “Gruntfile.js”-tiedostossa. Se määrittää komentorivityökaluun komennot, mitä komentorivillä voidaan suorittaa ja näiden komentojen alle voidaan toteuttaa tehtäviä, mitä kehittäjät tarvitsevat. Kerralla hyvin konfiguroitu ”Gruntfile.js”-tiedosto edesauttaa paljon ohjelmistokehitystä ja säästää työaikaa. Alla olevassa esimerkissä nähdään miltä osa ”Gruntfile.json”-tiedostosta näyttää kyseessä olevassa projektissa.

```
grunt.initConfig({
  yeoman: appConfig,
  watch: {
    bower: {
      files: ['bower.json'],
      tasks: ['wiredep']
    },
```

```
    js: {
      files: ['<%= yeoman.app %>/scripts/{,*/}*.js'],
      tasks: ['newer:jshint:all', 'newer:jscs:all'],
      options: {
        livereload: '<%= connect.options.livereload %>'
      }
    },
    jsTest: {
      files: ['test/spec/{,*/}*.js'],
      tasks: ['newer:jshint:test', 'newer:jscs:test', 'karma']
    },
    styles: {
      files: ['<%= yeoman.app %>/styles/{,*/}*.css'],
      tasks: ['newer:copy:styles', 'postcss']
    }
}
```

Konfigurointi-tiedoston syntaksin ja sen määrittelyn, voi lukea kokonaisuudessaan Gruntin omasta dokumentaatiosta. Tärkein funktio tiedostossa kuitenkin on sen “initConfig()”-funktio, joka määrittää sille parametrina annetussa JavaScript-objektissa komennot, joita komentoriviltä voidaan suorittaa. Tämän alle kerrotaan objekti-notaatiolla tai listana, mitä tiedostoja komento käyttää ja mitä niille tehdään.

Grunt voidaan esimerkiksi määrittää sijoittamaan Bowerilla asennettujen JavaScript-tiedostojen nimet automaattisesti index.html tiedostoon, mikä normaalisti kirjaston asennuksen yhteydessä pitäisi tehdä tiedostoon manuaalisesti. Alla olevassa esimerkissä esitetään, miltä osio näyttää “index.html”-tiedostossa, johon Gruntin wiredep-lisäosa linkittää asennetut JavaScript-tiedostojen nimet.

```
<!-- bower:js -->
<script src="bower_components/jquery/dist/jquery.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/bootstrap/dist/js/bootstrap.js"></script>
<script src="bower_components/angular-animate/angular-animate.js"></script>
<script src="bower_components/angular-aria/angular-aria.js"></script>
<script src="bower_components/angular-cookies/angular-cookies.js"></script>
<script src="bower_components/angular-messages/angular-
messages.js"></script>
<script src="bower_components/angular-resource/angular-
resource.js"></script>
<script src="bower_components/angular-route/angular-route.js"></script>
<script src="bower_components/angular-sanitize/angular-sanitize.js"></script>
<script src="bower_components/angular-touch/angular-touch.js"></script>
<script src="bower_components/angular-bootstrap/ui-bootstrap-
tpls.js"></script>
```

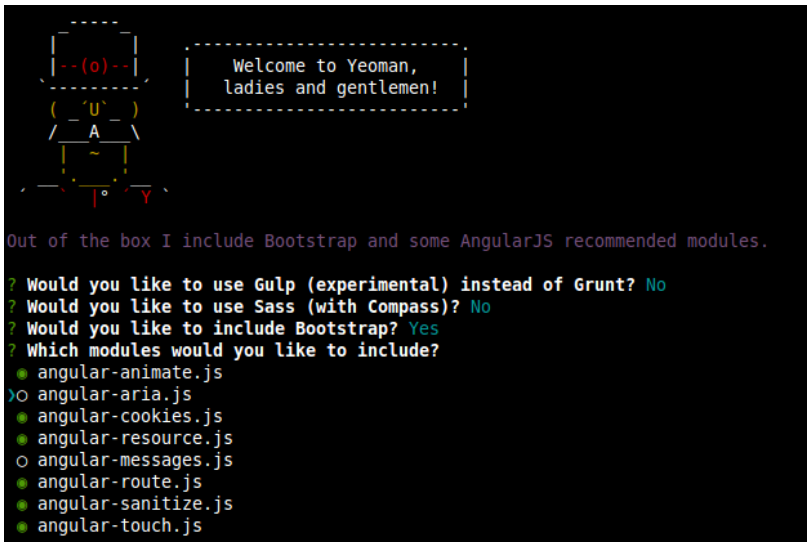
```
<script src="bower_components/js-data/dist/js-data.js"></script>
<script src="bower_components/js-data-http/dist/js-data-http.js"></script>
<script src="bower_components/js-data-angular/dist/js-data-
angular.js"></script>
<script src="bower_components/lodash/lodash.js"></script>
<script src="bower_components/restangular/dist/restangular.js"></script>
<script src="bower_components/angular-jwt/dist/angular-jwt.js"></script>
<!-- endbower -->
```

Esimerkistä on nähtävissä, kuinka Wiredep-lisäosa sijoittaa script-merkinnät “<!-- bower:js -->” ja “<!-- endbower -->” kommenttien väliin.

9.5.2 Yeoman

Yeoman on NodeJS-alustan päällä toimiva työkalu, mikä on tehty edesauttamaan projektien nopeaa aloittamista ja lähdekoodissa toistuvien osien generointia. Kuten Grunt ja Bower, myös Yeoman on komentorivityökalu. Yeoman-ohjelmistoon on saatavilla myös laaja kattaus lisäosia ja ne asennetaan NPM-paketinhallinta-ohjelmalla.

Yeoman-ohjelmaa käytettiin generoimaan pohja ja kansiorakenne Angular-projektille heti projektin alkuvaiheessa. Yeoman generoi projektille paljon valmista työn nopeaan aloittamiseen, mikä piti sisällään myös sen, että projektin rakenne noudattaisi suositeltuja käytäntöjä. Yeoman generoi projektiin valmiiksi myös ”Gruntfile.json”-tiedoston, jolloin myös Gruntin konfigurointi projektille tapahtuu automaattisesti. Yeoman asentaa riippuvuudet käyttäen Bower-paketinhallintaa. Angular-projekti aloitetaan asentamalla generator-angular -lisäosa ja suorittamalla komento “yo angular [ohjelman_nimi]”. Kuvassa 24 on esitetty, miltä projektin ensiasetusten valinta näyttää komentorivillä, jos yllämainittu komento suoritetaan ja uusi projekti luodaan.



Kuva 24. Projektin ensiasetusten valinta

Projektin edetessä Yeomaniä voidaan käyttää generoimaan ohjelmaan toistuvia osia ilman niiden uudelleen kirjoittamista. Angular-projektiin voidaan esimerkiksi luoda uusi kontrolleri komennolla “yo angular:controller [kontrollerin_nimi]”. Alla olevassa esimerkissä nähdään miltä generoitu kontrolleri-tiedosto näyttää.

```
'use strict';
/**
* @ngdoc function
* @name puupietariFrontendApp.controller:TestCtrl
* @description
* # TestCtrl
* Controller of the puupietariFrontendApp
*/
angular.module('puupietariFrontendApp')
 .controller('TestCtrl', function () {
   this.awesomeThings = [
     'HTML5 Boilerplate',
     'AngularJS',
     'Karma'
   ];
 });
```

10 Lemonsoft ERP -integraatio

Projektissa tehtiin järjestelmäintegraatio ohjelmallisella rajapinnalla Lemonsoft ERP-toiminnanohjausjärjestelmään. Puupietari Oy käyttää Lemonsoft-ohjelmistoa asiakkaiden ja tuotteiden hallinnointitehtävissä sekä lähetettyjen

tilausten laskujen muodostamisessa. Puupietari Oy ei käyttänyt ERPiä varaston tai tuotannon hallinnoimisessa. Integraation tehtävänä oli mahdollistaa asiakkaiden ja tuotteiden tuominen kehitteillä olevaan ohjelmistoon. Ohjelmiston tuli myös luoda laskut ERPiin luotujen lähetysten perusteella.

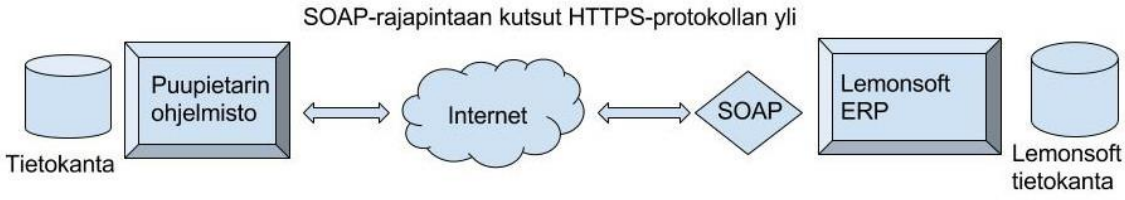
Tyypillinen ERP-ohjelmiston integraatio on verkkokauppa, jossa toteutettu verkkokauppa kirjoittaa tietoa ja lukee tietoa ERPistä. Integraation toteutuksessa roolit projektin sisällä jaetaan integroitavaan sovellukseen, toiminnanohjausjärjestelmään johon integraatio tehdään ja integraattoriin. Integraattori on taho, joka suunnittelee ja toteuttaa integroitavan ohjelmiston asiakkaan tarpeiden mukaan.

10.1 Ohjelmallinen rajapinta

Lemonsoft ERP-ohjelmisto käyttää tiedonvälitysmuotona SOAP-protokollaa. Ohjelmisto tarjoaa rajapinnan tietokannan tai yrityksen perusteella ja jokaista yritystä varten ERPiin tulee asentaa oma rajapinta. Rajapinta tarjoaa pääsyn seuraaviin resursseihin:

- Lemonsoft ERPin hallintaan
- CRM-asiakkuuden hallintaan
- varastotapahtumiin
- osto- ja myyntilaskuihin
- laskutukseen
- tuotantoon.

Kuvassa 25 on esitetty integraation kokonaiskuva, joka esittää mitä osia integraatioon liittyy.



Kuva 25. Integraation kuvaus

10.1.1 Rajapinnan variaatiot

ERP tarjoaa kaksi variaatiota rajapintaan, mitä ovat data- ja käyttäjälähtöiset rajapinnat. Rajapinnat asentuvat oletuksena ja kummatkin rajapinnat ovat omissa osoitteissaan. Näin ollen rajapinnoilla on omat WSDL-tiedostot, jotka kuvaavat rajapinnan toiminnallisuutta. Datalähtöisessä rajapinnassa rajapinnan omistaja päättää mistä yrityksestä tiedot tulevat. Se ei vaadi autentikointia, eli käyttäjätiedoilla ei ole merkitystä rajapinnan käyttämisessä. Yhdellä rajapinnalla voidaan käyttää kerrallaan vain yhtä yritystä.

Käyttäjälähtöisessä rajapinnassa käyttäjän täytyy kirjautua rajapintaan ja valita tietokanta mihin hän kirjautuu. Jotta tietokantaan saadaan yhteys, käyttäjällä täytyy olla hyväksyttävät tunnukset ja sen myötä tietokannan käyttämiseen vaaditut oikeudet. Toteutettavan ohjelman kannalta tämä todettiin paremmaksi vaihtoehdoksi. Malli mahdollisti myös sen, että voisimme helposti muuttaa mihin tietokantaan liitos tehtäisiin. Asiakkaan ERPissä on olemassa myös testitietokanta, jota pystyttiin käyttämään integraation testaamisessa asiakkaan palvelimella ennen lopullisen tuotantoversion asentamista.

10.1.2 Rajapintaan kirjautuminen

Ohjelmallinen rajapinta web-ympäristössä voidaan suojata monella eri tavalla. Jotkin suojausmenetelmistä ovat helpompia toteuttaa tietyllä käyttöjärjestelmällä, mikä riippuu käyttöjärjestelmän sisältämästä tuesta. Lemonsoft ERPissä tämä puoli on kuitenkin hoidettu hyvin ja rajapintaan kirjautuminen on helppoa, riippumatta ohjelmointikielestä tai alustasta.

Lemonsoftin käyttäjälähtöisessä rajapinnassa kirjautuminen tapahtuu ensin kutsumalla login-funktiota käyttäjänimellä, salasanalla ja tietokannalla mihin halutaan kirjautua. Onnistunut kirjautuminen palauttaa istunnon tunnistavan avaimen. Avain on tarkoitus lisätä jokaiseen rajapintaan tehtävän kutsun mukaan ja se on voimassa 30 minuuttia kutsujen välissä. Tämän jälkeen käyttäjän täytyy kirjautua rajapintaan uudelleen, jotta hän saa uuden avaimen tai se voidaan uusia ohjelmallisesti kutsujen välissä renew-funktiolla. Kirjautumiseen liittyvissä

kutsuissa liikkuu arkaluonteista tietoa ohjelmien välillä ja siksi pyynnöt pitää suorittaa suojatun, esimerkiksi HTTPS-yhteyden yli. Myös yleisesti rajapintaa suositellaan käytettäväksi vain suojatulla yhteydellä.

Käyttäjälähtöistä rajapintaa käytettäessä, istunnon avain täytyy antaa jokaiselle rajapintaan tehtävälle kutsulle. Avain lisätään aina ensimmäisenä parametrina Rajapinta-funktioiden kutsuihin ja perään tulevat muut tarvittavat parametrit. Esimerkkinä tästä on asiakaslistan noutaminen ryhmän mukaan. Funktiossa “GetCustomerListByGroup(strSessionID, iGroup)” ensimmäinen parametri on istunnon avain ja toinen on ryhmän numero. Datalähtöisessä rajapinnassa funktiokutsu on muuten sama, mutta ensimmäinen “strSessionID”-parametri jätetään pois. Huomaa että rajapinta käyttää parametrien nimien määrittelyssä Unkarilaista notaatiota, missä muuttujan tyyppi lyhennetään muuttujan nimen eteen.

10.2 Toteutus

10.2.1 Ohjelmointikieli

Python-kieleen löytyy jonkinlainen tarjonta SOAP-rajapinnan käyttöön pystyviä kirjastoja. Saatavilla olevat kirjastot voidaan monessa ohjelmointikielessä laittaa kahteen ryhmään. Kirjastoihin jotka generoivat pysyvät käytettävät luokat ohjelmallisesti SOAP-rajapinnan kuvaavasta tiedostosta. Se tapahtuu ohjelmistokehitysprojektin alussa ja näitä pysyviä luokkia on tarkoitus käyttää ohjelmistokehityksen aikana. Toiset kirjastot generoivat käytettäviä funktioita ja luokkia dynaamisesti ohjelman ajon aikana. Luokat ja funktiot ovat olemassa vain ohjelman ajon aikana ja ne tulee luoda aina uudestaan ohjelman käynnistymisen yhteydessä. Pythonilla käytettiin jälkimmäistä vaihtoehtoa ja SUDS-Jurko nimistä kirjastoa. Kirjasto on vain haaroitus alkuperäisestä SUDS-kirjastosta, koska sen ylläpito on loppunut. Kirjasto toimii kuitenkin täysin samalla tavalla kuin edeltäjänsä ja sitä voidaan käyttää myös uusilla Python-versioilla.

SUDS-kirjasto parsii WSDL-tiedoston aina kun ohjelma ottaa ensimmäisen kerran yhteyttä rajapintaan ja tästä syystä ensimmäinen kutsu ohjelman käynnistymisen jälkeen kestää kauemmin kuin tämän jälkeiset kutsut. WSDL-tiedosto

voidaan antaa kirjastolle myös paikallisena tiedostona, jos se ei ole saatavissa rajapinnasta. ERPin rajapinnasta on mahdollista saada yhdellä kertaa kuvaustiedosto, joka kuvaa kaikki käytettävät rajapinta-funktiot samanaikaisesti.

10.2.2 Rajapinnan käyttö

SUDS-kirjasto generoi käytettävät rajapinta-funktiot ohjelman ajon aikana, kun rajapintaan otetaan ensimmäisen kerran yhteyttä ja sieltä noudetaan tarvittava WSDL-tiedosto. Kirjastossa on “client”-objekti, joka toimii päätasona kirjastolle ja tämän alla on saatavilla kaksi muuta palvelua: “factory”- ja “service”-objektit. “Service” tarjoaa funktiot, jotka on generoitu rajapinnan pohjalta ja “factory” tarjoaa funktioiden määrittämien objektien luontiin tarkoitetun rajapinnan. Alla nähdään SOAP-rajapinnan kirjautumisen suorittava funktio toteutettuna SUDS-kirjastolla.

```
def _authenticate(self):
    login_info = self.client.factory.create('ns1:LogInInfo')
    login_info.CompanyDatabase = self.settings.database_name
    login_info.UserName = self.settings.username
    login_info.Password = self.settings.password
    login_result = self.auth_service.LogIn(login_info)
    # Update session_id in database
    if login_result.ResultCode == "OK":
        self.settings.session_id = login_result.SessioID
        self.settings.save()
    else:
        self.settings.session_id = ""
        self.settings.save()
        raise exceptions.LemonsoftLoginError(_("Cannot login to Lemonsoft with
message: {error}".format(error=login_result.ResultText)))
    return login_result
```

Funktion rivi “self.client.factory.create('ns1:LogInInfo')” tekee “factory”-palvelulla “LogInInfo”-objektin, jonka rajapinta on määrittänyt WSDL-tiedostosta. Muuttujien asetusten jälkeen “self.auth_service.LogIn(login_info)”-rivi kutsuu rajapinnan “LogIn”-funktiota ja tälle annetaan parametrina “factory”-palvelulla tehty objekti, joka sisältää kirjautumistiedot palveluun.

Huomaa “self.auth_service.LogIn(login_info)”-rivin sisältämä “auth_service”-palvelu. Projektiin tehdyllä rajapintapaketilla on tehty Lemonsoftin palvelut erotteleva taso, jonka tehtävänä on rajata funktiot palveluiden mukaan. Esimerkissä

palveluun liittyvät kaikki kirjautumiseen sisältyvät funktiot ja sama on toistettu myös asiakkaille, tuotteille ja muille Lemonsoft-ohjelmiston resursseille.

10.2.3 Tiedon käyttö

Joissakin tapauksissa integroitavan ohjelman täytyy noutaa suuria määriä tai riittävän usein dataa rajapinnan yli. Tällaisissa tapauksissa on mietittävä, halutaanko rajapinnan tiedoista pitää kopiotietoa ohjelman omassa tietokannassa. Yleensä kun tietoja pidetään kahdessa eri järjestelmässä, nousee ilmi erilaisia kysymyksiä, jotka tulee ottaa huomioon integraation suunnittelussa. Kumpi järjestelmä on master-järjestelmä, eli kumman tiedot ovat oikein jos tietoja on muutettu molemmissa järjestelmissä? Milloin tietoja päivitetään? Mitä tehdään jos tietoja ei saada päivitettyä toiseen järjestelmään?

Tässä projektissa tietojen lukemismäärän suuruuden ja päivitysmäärän vähäisyyden takia, ohjelma on suunniteltu pitämään Lemonsoftin tietoja ohjelmiston omassa tietokannassa ja näitä tietueita päivitetään tietyin väliajoin. Käyttäjä voi myös itse tarvittaessa pakottaa päivitysprosessin päälle ohjelmasta. Lemonsoft tarjoaa rajapinta-funktiot muuttuneiden tietojen noutamiseen päivämäärän perusteella. Tätä toiminnallisuutta integroitavan ohjelmiston on tarkoitus hyödyntää tietojen päivittämisessä sen omaan tietokantaan. Tällöin käyttölukumäärät rajapinnasta saadaan vähemmäksi ja ohjelmasta tulee paljon sujuvampi toimivuudeltaan.

Suurten tietomäärien tuominen rajapinnan yli ei ole järkevää ja joskus se voi olla jopa mahdotonta. Esimerkiksi ohjelmiston asennuksen yhteydessä tehty ensimmäinen tiedon lataaminen omaan tietokantaan voi olla aikaa vievä prosessi, jos kyseessä on esimerkiksi 60 000 asiakkaan tai tuotteen tiedot. Näissä tapauksissa on suositeltavaa toteuttaa jonkinlainen ensimmäinen ajo tietojen noutamisessa järjestelmästä toiseen. Lemonsoft-ohjelmiston kanssa tämä voidaan toteuttaa esimerkiksi CSV-muotoisella tuonti-tiedostolla. Tämä tiedosto on mahdollista parsia erillisellä Python-ohjelmalla integroitavan ohjelman omaan tietokantaan. Ensimmäisen tietojen päivittämisen jälkeen, voidaan käyttää tietyin väliajoin ajettavaa tietojen päivitystyötä Lemonsoft-järjestelmästä.

11 Toteutettu ohjelmisto

Seuraavassa osiossa käsitellään itse ohjelmiston toteutusta asiakkaalle. Kappaleissa käsitellään ohjelmiston yleistä rakennetta, sen muita tärkeitä toiminnallisia piirteitä sekä selvitetään, kuinka projektiin valitut työkalut toimivat ominaisuuksiltaan. Tärkeimmät ohjelmistossa olleet kehykset olivat asiakaspuolella AngularJS ja serveripuolella Django. Molempia kehyksiä käydään läpi, mutta opinnäytetyön tekijöiden painon ollessa serveripuolella, asiakaspuolta käsitellään vähemmän.

Asiakas- ja serveripuolen toteutukset ovat täysin toisistaan riippumattomia kokonaisuuksia. Tästä johtuen kumpikin palvelu voi käyttää joko samaa tai eri isäntäohjelmistoa (Apache, IIS). Lopullisen asennuksen yhteydessä molemmat toteutetut osat sijoitettiin kuitenkin asiakkaan omalle palvelimelle, Lemonsoft ERP:n sijainnin jäädessä kuitenkin erilliselle kolmannen osapuolen tarjoamalle palvelimelle. Palveluiden keskeinen tiedonsiirto tapahtuu JSON-formaatissa käyttämällä REST-rajapintaa. REST-rajapinnan käyttö tarjoaa kehitysmahdollisuudet muihin asiakaspäihin, kuten älypuhelinohjelmistoihin tai kämmenlaitteisiin. Tämä tuli ottaa suunnitteluvaiheessa huomioon ja yksi jatkokehitysidea onkin saada asiakkaalle kämmenlaitteet tukemaan tuotannon eri prosessien vaiheita.

11.1 Angular

11.1.1 Kehyksen rakenne

AngularJS-kehyksen ohjelmarakenne koostuu kontrollereista, näkymistä ja palveluista. AngularJS ei omaa yhtä vahvaa tietomallia, jota esimerkiksi Ember käyttää. Emberissä tieto mallinnetaan erikseen, kun taas AngularJS:n tietomallit ovat normaaleja JavaScript-objekteja kontrollereiden seassa. AngularJS-ohjelmistot koostuvat uudelleenkäytettävistä moduuleista, joita ohjelmistokehittäjä voi toteuttaa itse lisää. Näihin lukeutuvat myös Bowerilla asennetut lisäosat. Angularprojektit sisältävät aina yhden päämoduulin, johon määritellään riippuvuudet muista moduuleista. Yksi moduuli rakentuu alikomponenteista, kuten kontrollereista, direktiiveistä tai palveluista, joiden toiminnallisuuksia ohjelman

päämoduuli käyttää. Alla nähdään esimerkki moduulin riippuvuuksien määrittelystä. Funktiolle ”.module()” annetaan riippuvuudet määrittävä lista. Tämän jälkeen ajetaan ”.config()”-funktio, missä asetukset ennen ohjelman käynnistämistä määritellään.

```
angular.module('puupietariFrontendApp', [
  'ngAnimate',
  'ngAria',
  'ngCookies',
  'ngMessages',
  'ngResource',
  'ngRoute',
  'ngSanitize',
  'ngTouch',
  'ui.bootstrap',
  'js-data'
]).config(function ($routeProvider, $httpProvider) {
  // Configuration here
}));
```

Projektissa pyrittiin luomaan yleiskäyttöisiä moduuleita, joita pystyttäisiin myöhemmin hyödyntämään samankaltaisissa projekteissa. Esimerkiksi REST-rajapintaan toteutettiin JWT-autentikointi, joka on jo käytössä toisessa asiakasprojektissa.

11.1.2 Riippuvuudet

Bower-paketinhallinnalla voidaan AngularJS-projektin riippuvuudet määritellä erilliseen ”bower.json”-tiedostoon. Bowerista löytyy valmis tuki eri kehitysympäristöjen tarvitsemille riippuvuuksille. Tämä näkyy riippuvuustiedoston rakenteessa, sen jakaessa kehitystyössä käytettävät riippuvuudet omaan lohkoonsa. Alla on koodiesimerkki miltä tiedoston kyseinen osa näyttää.

```
 "dependencies": {
   "angular": "^1.4.0",
   "bootstrap": "^3.2.0",
   "angular-animate": "^1.4.0",
   "angular-aria": "^1.4.0",
   "angular-cookies": "~1.5.3",
   "angular-messages": "^1.4.0",
   "angular-resource": "^1.4.0",
   "angular-route": "^1.4.0",
   "angular-sanitize": "^1.4.0",
   "angular-touch": "^1.4.0",
```

```
  "angular-bootstrap": "~0.14.3",
  "js-data-angular": "~3.1.0",
  "js-data": "~2.9.0",
  "restangular": "~1.5.2"
},
"devDependencies": {
  "angular-mocks": "^1.4.0"
}
```

Esimerkissä ”devDepencies”-osio määrittää vain kehitystyössä käytettävät moduulit. Kun paketti asennetaan Bowerilla, komennolle annetaan parametri määrittelemään, onko paketti käytössä tuotanto- vai kehitysympäristössä. Paketin asennuskomennon ”bower install angular-mocks --save-dev” ”—save dev”-osa määrittelee paketin olevan tarkoitettu vain kehitystyössä käytettäväksi.

11.2 Django

Serveripuolen tekniikaksi valittiin Django-kehys ja REST-rajapinnan toteuttamisen tueksi asennettiin myös Django Rest Framework(DRF). Lisäksi käytössä oli useita muita pieniä lisäosia helpottamaan työn määrää. Osion tarkoituksena ei ole käydä läpi kaikkea Djangon toiminnallisuutta, vaan käsitellä sen tärkeimpiä ominaisuuksia ohjelmiston kehityksen kannalta. Djangon kaikki toiminnallisuus löytyy kehyksen omasta dokumentaatiosta. Kappaleissa pyritään käsittelemään kehystä niin, että Djangosta tietämätönkin pystyisi asian ymmärtämään.

11.2.1 Django-paketit

Django noudattaa projektin tiedostojen rakenteessa pakettikohtaista lähestymistapaa. Jokainen paketti on oma ohjelmistokokonaisuus ja jokainen paketti pitää sisällään oman funktionalisuuden. Paketit on tarkoitus suunnitella uudelleenkäytettäviksi kokonaisuuksiksi ja niistä muodostuvat Django-projekteihin asennettavat lisäosat. Pakettien funktionaalisuus koostuu sen alla olevista Python-tiedostoista. Paketin alla voi olla myös muita paketteja, mutta ohjelman selkeän jaottelun vuoksi ohjenuorana voidaan pitää, että yksi paketti sisältää vain siihen pakettiin liittyvän toiminnallisuuden. Django-projekti sisältää yhden pääpaketin.

Sen tarkoituksena on ajaa ohjelman tuotantotilan käynnistykseen kuuluva tiedosto(wsgi.py), sekä pitää sisällään projektin asetuksiin liittyvä tiedosto.

Ohjelmaan toteutettiin uudelleenkäytettäviä paketteja, joiden ansiosta tulevien samankaltaisten projektien kehitystyötä saadaan vähemmäksi. Tärkeimmät uudelleen käytettävät paketit ohjelmassa ovat Lemonsoftin ERP-integraatioon liittyvät kaksi pakettia. Yksi näistä toteuttaa kokonaan rajapintaan liittyvän toiminnallisuuden ja tarjoaa tästä helposti ymmärrettävän ja käytettävän rajapinnan ohjelmoijalle. Toinen paketti hoitaa paikallista kopiota Lemonsoftin ERPin resursseista, kuten sen sisältämistä asiakkaista ja tuotteista. Paketista löytyy myös logiikka resurssien päivittämistyöhön rajapinnan yli. Ohjelma sisältää myös oma paketin pdf-raporttien generoimiseen, mitä on mahdollista käyttää pienellä muokkaamisella muidenkin raporttien luomisessa. Tästä paketista ja sen toteutuksesta kerrotaan myöhemmin.

11.2.2 Projektin managerointi

Django-projektin hallintaan ja erilaisten komentojen suorittamiseen on tarkoitettu projektin juuresta löytyvä “manage.py”-tiedosto. Sen voi suorittaa Python-kääntäjällä komentoriviltä ja sillä ohjelmalle voidaan antaa erilaisia suoritettavia komentoja. Kehitystyössä yleisimpiä näistä ovat tietokantamigraation määrittävien tiedostojen tekeminen ja suorittaminen. Migraatio-tiedostot voidaan esimerkiksi luoda komennolla “python manage.py makemigrations” ja tämän jälkeen ajaa tietokantaan komennolla “python manage.py migrate”.

Projektissa työkalulle voidaan määritellä myös omia komentoja. Jos tarkoitus on toteuttaa väliajoin ajettavia muusta ohjelmasta riippumattomia ajoja, tulee ne laittaa ”manage.py”-tiedostoon. Tapaa hyödynnettiin esimerkiksi Lemonsoft ERP:stä haettujen tietojen päivityksessä paikalliseen tietokantaan. Komento voidaan suorittaa ulkopuolisesta ohjelmasta ajastetusti, kuten Linuxilla käyttäen Crontab-ajastuskomentoa.

11.2.3 Asetukset eri ympäristöille

Django-projektin asetukset määritetään projektin pääpaketissa “settings.py”-tiedostoon. Asetukset ovat vakioita ja niitä ei ole tarkoitus muuttaa ohjelman

ajon aikana. Asetustiedostossa määritellään esimerkiksi tietokantayhteyteen liittyvät tiedot ja Djangon muiden lisäosien asetukset. Asetusten määrittämiseen käytetään Python-listoja, hajautustauluja (dictionary) tai tavallisia primitiivi tyyppejä. Alla nähdään esimerkki asetustiedoston sisällöstä.

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
ALLOWED_HOSTS = []
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
MIDDLEWARE_CLASSES = (
   'django.contrib.sessions.middleware.SessionMiddleware',
   'django.middleware.locale.LocaleMiddleware',
   'corsheaders.middleware.CorsMiddleware',
   'django.middleware.common.CommonMiddleware',
   'django.middleware.csrf.CsrfViewMiddleware',
   'django.contrib.auth.middleware.AuthenticationMiddleware',
   'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
   'django.contrib.messages.middleware.MessageMiddleware',
   'django.middleware.clickjacking.XFrameOptionsMiddleware',
   'django.middleware.security.SecurityMiddleware',
)
```

Työn alkuvaiheessa Django-projektiin oletuksena sisältyvä yksi asennustiedosto nostatti ongelman. Ongelman muodosti eri ympäristöjen ja kehittäjien tarvitsema mahdollisuus määrittää eri asetuksia, mihin yksi asetustiedosto ei riittänyt. Ratkaisutavaksi päätyi asetusten perimistä käyttävä rakennemalli. Projektin pääpaketin alle määriteltiin uusi “settings”-paketti ja sen alle määriteltiin “base_settings.py”-tiedosto. Kyseiseen tiedostoon määriteltiin kaikki yleiset muuttumattomat asetukset, jotka ovat voimassa jokaisessa kehitysympäristössä. Tämän jälkeen määriteltiin kehittäjille tarkoitettu “dev_settings.py”-tiedosto ja ohjelman koontiversiolle oma “build_settings.py”-tiedosto, jotka molemmat perivät pohjan “base_settings.py”-tiedostolta. Tällä menetelmällä Jenkinsin julkaisuversiolle sekä jokaisen kehittäjän omalle versiolle pystyttiin määrittämään omat asetukset. Kumpikaan asetustiedostoista ei sijaitse pääpositoryssa ja molemmat on lisätty “.hgignore”-tiedostoon, jolloin toisten käyttämät asetustiedostot eivät mene muille kehittäjille, eivätkä asetustiedostot ole riippuvaisia Mercurialin päivitys-operaatioista.

Alla on esitetty esimerkki miten Jenkinsin julkaisuympäristössä määritetty asetustiedosto perii “base_settings.py”-tiedostoa ja ylikirjoittaa sen määrittämiä asetuksia.

```
from .base_settings import *

DEBUG = False
DATABASES = {
  'default': {
     'ENGINE': 'django.db.backends.mysql',
     'NAME': 'database_name',
     'USER': database_user,
     'PASSWORD': password,
     'HOST': '127.0.0.1',
     'PORT': '3306',
  }
}
```

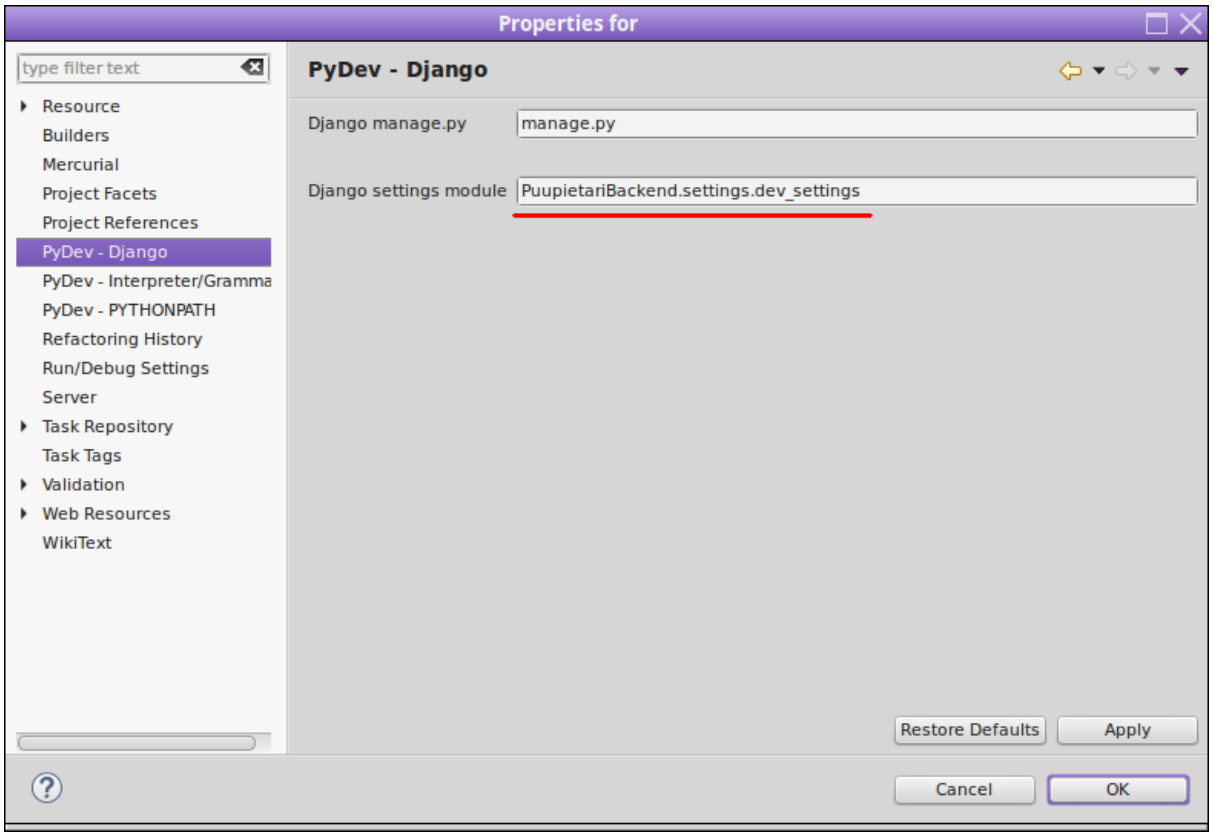
Lisäksi pitää myös muistaa muokata Djangon automaattisesti tekemiä tiedostoja, jotka määrittävät ympäristömuuttujan sille missä Djangon asetustiedosto sijaitsee. Oletuksena nämä osoittavat Djangon normaaliin asetustiedostoon, jota ei muutosten jälkeen ole enää olemassa. Ongelma korjataan muokkaamalla “wsgi.py” tiedostoa ja muuttamalla seuraavaa riviä.

```
os.environ.setdefault(
  "DJANGO_SETTINGS_MODULE",
  "PuupietariBackend.settings.build_settings"
)
```

Ympäristömuuttuja “DJANGO_SETTINGS_MODULE” on laitettu osoittamaan “PuupietariBackend.settings.build_settings"-polkuun, mikä oli aiemmin muotoa “PuupietariBackend.settings". Sama ympäristömuuttujan asetus tulee tehdä Jenkinsin suorittamassa bash-skriptissä.

```
export DJAN-
GO_SETTINGS_MODULE="PuupietariBackend.settings.build_settings"
```

Yleensä jos IDE tukee Django projektia, sen asetuksiin voidaan suoraan määrittää käytettävä asetustiedosto kun ohjelma suoritetaan paikallisesti. Kuvassa on esitetty miltä asetuksen määrittäminen näyttää Eclipse-ohjelmointiympäristössä (kuva 26).



Kuva 26. Asetusten määrittäminen Eclipse-ohjelmalle

11.2.4 Ohjelmiston riippuvuudet

Pythonin paketinhallintaohjelmalle (pip) voidaan antaa ulkopuolinen “requirements.txt”-tiedosto, joka määrittää ohjelman tarvitsemat riippuvuudet ja asentaa ne.

Ongelmaksi syntyi jälleen toimintamallin tukemattomuus eri ympäristöihin asetetuille riippuvuuksille, mikä tulee esille esimerkiksi kehitys- ja tuotantoympäristössä. Kehitysympäristössä voi esiintyä riippuvuuksia, joita voidaan käyttää kehitystyön tukena ja joille ei ole tarvetta asiakkaan tuotantoympäristössä. Myös Jenkins:in automaattinen ympäristö tarvitsee omat paketit, mitä kehitysympäristössä ei tarvita.

Koska Djangon asetukset noudattavat perintämallia, haluttiin sama toiminnallisuus myös riippuvuuksille. Ohjelmaan tehtiin “requirements_base.txt”-tiedosto, joka määritti kaikkien ympäristöjen käytössä olevat riippuvuudet. Tämän jälkeen määritettiin “requirements_build.txt”-, “requirements_dev.txt”- ja “requirements_prod.txt”-tiedostot. Tiedostot perivät “requirements_base.txt”-tiedostoa ja

niihin on lisätty riippuvuuksia, mitä kukin ympäristö tarvitsi. Alla sijaitsee esimerkki Jenkinsin julkaisuympäristön käyttämästä “requirements_build.txt” tiedostosta.

```
-r ./requirements_base.txt
pyflakes==1.1.0
pep8==1.7.0
django-jenkins==0.18.1
coverage==4.0.3
```

Tiedosto sisältää Jenkinsin lähdekoodin testaamiseen ja raportointiin liittyvät lisäosat. Ensimmäisellä rivillä on teksti joka sisältää polun “requirements_base.txt”-tiedostoon. Rivin “-r”-osa on Python-paketinhallinnan parametri, millä määritetään ulkopuolinen riippuvuus-tiedosto. Tämän vuoksi myös Jenkinsin suorittamaan bash-skriptiin pitää määrittää rivi, mistä tiedostosta riippuvuudet Python virtuaaliympäristöön asennetaan.

```
pip install -r requirements_build.txt –upgrade
```

Tämän asettelun avulla projektiin saatiin siistimpi riippuvuuksien määrittävä rakenne, eikä turhia asennuksia tehdä ympäristöihin, joissa niitä ei tarvita.

11.2.5 Django ORM

Djangon yksi vahvoista puolista on sen ORM-toiminnallisuus. Djangoon määritetään mallit tietokantataulun mukaan ja mallehin kentät, jotka vastaavat tietokantataulun sarakkeita. Djangossa tietokannan tietoja voidaan käsitellä kokonaan Pythonin omilla objekteilla ja funktioilla ilman SQL-kielten käyttämistä. Objektien käyttämisen perusteella Djangon tietokanta-lisäosa hoitaa kyselyiden muodostamisen tietokanta-tasolle. Djangon tietokanta-lisäosien takia sen ORM mallit pystyvät tukemaan monia eri tietokantoja. Tämä mahdollistaa ohjelman tietokantamoottorin vaihtamisen kokonaan toiseen, ilman minkäänlaisia koodimuutoksia ohjelmassa.

Djangossa mallit määritellään niille sopivan paketin alle “models.py” tiedostoon. Django etsii näihin tiedostoihin kirjoitetut mallit, joita voi sijaita useassa eri paketissa. Edellä mainittu sijoittelu onkin suotavaa, koska tällöin mallit voidaan lajitella pakettien toimivuuksien mukaan. Alla on esimerkki siitä, miltä mallin määritys näyttää.

```
class CompanySettings(models.Model):
  company_name = models.CharField(max_length=200)
  address1 = models.CharField(max_length=200, blank=True)
  address2 = models.CharField(max_length=200, blank=True)
  phone_number1 = models.CharField(max_length=100, blank=True)
  phone_number2 = models.CharField(max_length=100, blank=True)
  fax = models.CharField(max_length=100, blank=True)
  business_id = models.CharField(max_length=100)

  class Meta:
    verbose_name = _("company settings")
    verbose_name_plural = _("company settings")

  def __str__(self):
    args = {
      "name": self.company_name,
      "phone": self.phone_number1,
    }
    return _("Name:{name}, puh:{phone}").format(**args)
```

Mallissa määritellään objektin kentät eli samalla myös tietokantataulun sarakkeet. Luokan sisäisessä Meta-luokassa määritellään muuta objektiin liittyvää tietoa, kuten käännökset objektin nimelle yksikkö- ja monikkomuodossa. Luokan “__str__”-funktiota käytetään objektin tulostamisessa ja se määrittelee mitkä kentät luokasta menevät ulostuloon esimerkiksi käyttöliittymässä. Mallin kenttien määritys tapahtuu Djangon sisäisillä kentillä (CharField, BooleanField, jne.), mitkä vastaavat tietokannan eri tietotyyppejä. Näitä käytetään myös muodostamaan kentät ja JSON formaatin rakenne DRF:llä tehtyyn REST-rajapintaan, minkä Django toteuttaa automaattisesti mallien perusteella.

Mallien pohjalta Django-projektissa tehdään migraatiot ja migraatiot ajetaan tietokantaan. Aina kun malleihin tehdään muutoksia, tulee migraatiot tehdä uudestaan ja sen jälkeen ajaa ne jälleen tietokantaan. Migraatiot helpottavat tietokannan päivitystä valtavasti, kun kehittäjän ei tarvitse itse kirjoittaa SQL-skriptejä kun tietokokantaa halutaan päivittää. Migraatioiden hyöty tulee esille varsinkin silloin, kun ollaan päivittämässä jo asiakkaalla olevaa vanhaa ohjelmistoa uuteen versioon ja tietokantaan täytyy lisätä uusia tauluja tai sarakkeita. Seuraavassa koodiesimerkissä tuodaan esille, miltä Djangon tekemä migraatio näyttää.

```
from __future__ import unicode_literals
from django.db import migrations

class Migration(migrations.Migration):
```

```
dependencies = [
    ('Puupietari', '0002_auto_20160314_1218'),
]
operations = [
    migrations.RenameModel(
        old_name='StatusCode',
        new_name='OrderStatusCode',
    ),
]
```

Djangon malleilla voidaan kyselyjä muodostaa itse objektien funktioiden pohjalta ja ne joko palauttavat listan objekteja tai yksittäisen objektin, mikä riippuu kyselyn rakenteesta. Alla nähdään esimerkkikoodi, millä saadaan haettua tietyn id:n omaava tuote. Djangossa id kenttä tunnetaan myös nimellä pk.

```
Product.objects.get(pk=self.product_id_id)
```

Esimerkkikoodi palauttaa yhden Product-objektin, jos sellainen on olemassa. Kyselyissä on myös mahdollista käyttää muitakin funktioita ".get()"-funktion sijaan, kuten ".filter()"- tai ".all()"-funktioita, mitkä kummatkin palauttavat listan objekteja. Funktioille tarvitsee vain syöttää suodatukseen tarvittavat parametrit.

Toteutetussa ohjelmistossa malleja eriteltiin eri paketteihin niiden toiminnallisuuden mukaan. Hyvänä esimerkkinä tästä on Lemonsoft ERPistä tulevien tietojen paikallinen kopiotieto. SOAP-rajapinnan kautta tuleva tieto mallinnettiin Djangon malleiksi omaan pakettiin ja siihen kirjoitettiin myös mallien tietojen päivityskoodi rajapinnan yli.

11.2.6 Tietokannan lähtötiedot

Django tukee tietokantaan syötettävää lähtötietoa ja lähtötiedot voidaan tallentaa erilliseen "fixture.json"-tiedostoon, malleihin kuuluvan paketin alle. Näitä tiedostoja voidaan kirjoittaa käsin niiden ollessa JSON-formaatissa ja täten ne ovat myös helposti ihmisen luettavissa. Lähtötieto-tiedostoja voidaan luoda tietokannassa olevien tietojen perusteella ja ladata sitten tietokantaan. Tiedostojen luontia ja lataamista hallitaan "manage.py" työkalulla. Lähtötietoja voidaan myös käyttää Djangon suorittamien testien tueksi. Alla on esimerkki miltä lähtötieto-tiedoston sisältö näyttää.

```
{
```

```
  "pk": 1,
  "model": "Puupietari.orderstatuscode",
  "fields": {
     "status_code": "Tilattu",
     "description": "Toimitus on j\u00e4tetty tilaukseen"
  }
},
{
  "pk": 2,
  "model": "Puupietari.orderstatuscode",
  "fields": {
     "status_code": "Tuotannossa",
     "description": "Tilaus on tuotannossa"
  }
},
{
  "pk": 3,
  "model": "Puupietari.orderstatuscode",
  "fields": {
     "status_code": "Osa toimitettu",
     "description": "Osa tilauksesta on viel\u00e4 toimittamatta"
  }
}
```


Tätä toiminnallisuutta käytettiin lopullisessa toteutuksessa lataamaan yleistä kerran asetettavaa tietoa tietokantaan, joihin lukeutuvat mm. peruskäyttäjät, tarvittavia tilakoodeja lähetyksille, osoitetietoja jne.

11.2.7 Käyttäjien hallinta

Djangon mukana tulee valmiiksi asennettuna käyttäjähallinta, joka tukee käyttöoikeuksien eri tasoja ja käyttäjäryhmiä. Djangossa on superkäyttäjä ja admin-käyttäjä, jonka malliin on määritetty “is_staff”-kenttä. Superkäyttäjä on käyttäjä, jolla on oikeudet kaikkeen ohjelmassa ilman minkäänlaisia rajoitteita. “is_staff”-käyttäjällä on rajoituksia, mutta käyttäjätaso sallii pääsyn Djangon admin-sivuille. Muut näistä poikkeavat käyttäjät ovat normaaleja käyttäjiä ilman yllä mainittuja oikeuksia. Django tarjoaa paljon valmista funktionaalisuutta käyttäjien suhteen, mitä tulisi toteuttaa jokaiseen ohjelmaan, jossa käsitellään käyttäjiä.

Djangon valmiita käyttäjiä käytettiin ohjelman työntekijöiden sisäänkirjautumisessa REST-rajapintaan ja rajapinnan käyttöä rajoitettiin käyttäjäryhmillä tietyiltä osin, esimerkiksi laskujen luomisessa Lemonsoft ERPiin tehdyistä lähetyksistä. Djangossa ei-kirjautuneet -käyttäjät tunnistetaan “AnonymousUser”-

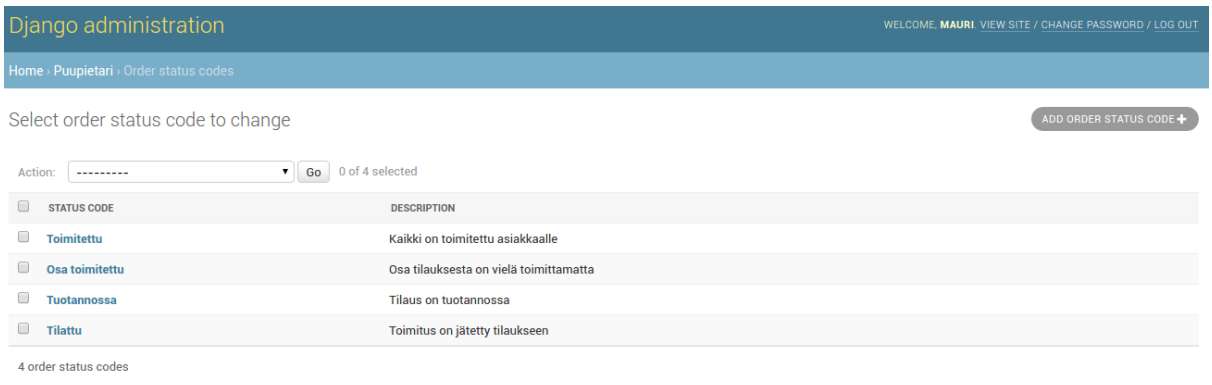
käyttäjinä. Näillä käyttäjillä toteutetussa ohjelmassa ei ole mitään oikeuksia mihinkään. Kaikki ohjelman toiminnallisuudet vaativat jonkinlaisen tunnistautumisen.

Käyttäjien lisääminen onnistuu Djangon admin-sivuilta tai komentoriviltä käyttäen “manage.py”-työkalua. Ensimmäinen superkäyttäjä on myös tehtävä komentorivin kautta, jotta ohjelmaan pääsee kirjautumaan sisään.

11.2.8 Django admin-sivut

Django mukana tulee lisäosa joka tekee ohjelmaan valmiit hallintasivut, joita voidaan muokata ohjelman tarpeiden mukaan. Sivuja ei ole tarkoitettu jokapäiväiseen käyttöön, vaan ainoastaan ohjelman ylläpitotehtäviin, kuten käyttäjien lisäämiseen tai joidenkin tietojen muokkaamiseen. Admin-sivut on tarkoitettu vain ja ainoastaan ylläpitäjille. Jos ohjelman malleista ei ole suoraa tietoa, voidaan admin-sivuilta saada aikaan suurtakin tuhoa, esimerkiksi poistamalla objekteja mitkä ovat riippuvaisia toisista objekteista.

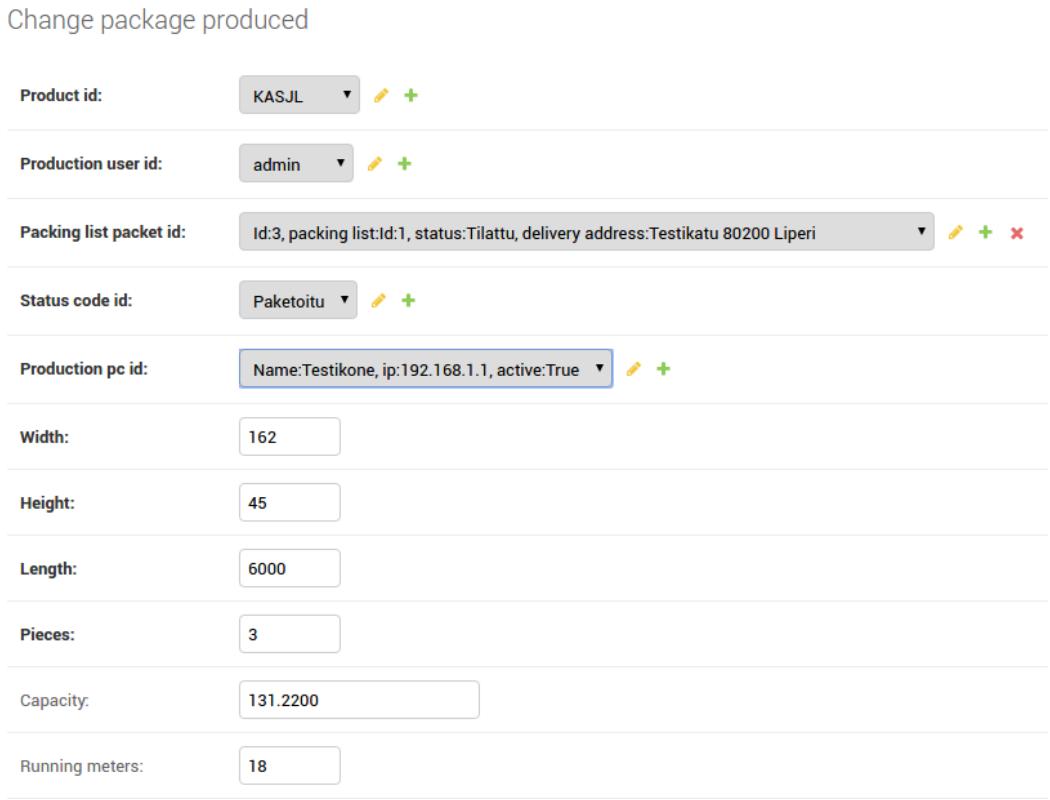
Django luo admin-sivut siihen määritettyjen ORM-mallien pohjalta. Kuvasta nähdään miltä objekteja listaava sivu ohjelmassa näyttää (kuva 27). Ohjelmallisesti on mahdollista määrittää, mitä malleja admin-sivuilta voidaan muokata.



Kuva 27. Objektien listaaminen admin-sivuilla

Admin-sivut tukevat automaattisia käännöksiä selaimen kielen pohjalta, joihin lukeutuvat myös käännökset suomen kielelle. Admin-sivut tukevat kehittäjän tekemiä suodatuksia, jolla listaan voidaan hakea vain tietyllä perusteella objekteja, esimerkiksi pelkkiä lähetettyjä lähetyslistoja. Objektin muokkaukseen tar-

koitetun lomakkeen admin-sivut luovat Djangon ORM-mallin pohjalta ja malliin määritetyt kentät pyrkivät vastaavat HTML-kenttiä. Esimerkiksi “CharField” vastaa “<input type=”text”>” kenttää ja “TextField” vastaa “<textarea>” kenttää. Kuvassa on esitetty miltä automaattisesti luotu lomake admin-sivuilla näyttää (kuva 28).



Kuva 28. Objektin muokkaaminen admin-sivuilla

Admin-sivuja voidaan muokata halutunlaisiksi samassa paketissa, missä ORM-mallit ovat. Paketissa täytyy olla “admin.py”-tiedosto, jota muokkaamalla admin-sivujen toimintaa ja ulkoasua voidaan muokata. Alla on koodiesimerkki sille, miltä admin-määritys kuvassa 28 olevan lomakkeen objektille näyttää.

```
class PackageProducedAdmin(admin.ModelAdmin):
  list_display = (
      'product_id',
      'width',
      'height',
      'length',
      'pieces',
      'capacity',
      'running_meters',
      'production_timestamp',
```

```
    'production_user_id',
    'packing_list_packet_id',
    'production_pc_id',
    'packing_list_packet_bit',
    'status_code_id',
    'packet_barcode',
)
list_filter = [
    'production_timestamp',
    'production_user_id',
    'production_pc_id__name',
    'packing_list_packet_id__packing_list_id__status_code_id__description',
]
search_fields = [
    'product_id__Product_description',
    'product_id__Product_description2',
    'production_pc_id__name'
]
```

Tiedostoon määritellään jokaiselle admin-sivulle haluttuun malliin admin-luokka ja luokalle tietyt muuttujat, joita sen halutaan muokkaavan. Yllä esimerkiksi “list_display” määrittää mitkä kentät näytetään sarakkeina listanäkymässä. Luokalle määritetään myös suodatusta ja etsintää varten kentät omiin listoihin. Tämän jälkeen luokka rekisteröidään käyttöön Djangolle.

```
admin.site.register(PackageProducedStatusCode, PackageProducedStatusCodeAdmin)
```

Lopullisessa ohjelmassa oikeudet admin-sivuille annettiin asiakkaalle itselleen ja toimeksiantajalle, joka jälkikäteen ylläpitää ja päivittää ohjelmistoa asiakkaalle. Asiakasta ohjeistettiin sivun käyttämisessä, esimerkiksi kuinka uusien käyttäjien lisääminen tapahtuu ja kuinka tilakoodeja voidaan muokata.

11.3 REST-rajapinta

11.3.1 Rakenne

DRF on Django-kehykseen saatavilla oleva lisäosa, joka on luotu REST-rajapintojen toteuttamiseksi. DRF:llä rajapinnan toteutus Djangon ORM-mallien pohjalta onnistuu vähäisellä työllä yleisimmille käyttötapauksille. DRF:n hyvä puoli on sen lisäosien loistava liitettävyys vähäisillä muutoksilla. Näihin lukeutuvat mm. rajapinnan autentikointityypin vaihtaminen kokonaan toiseen menetel-

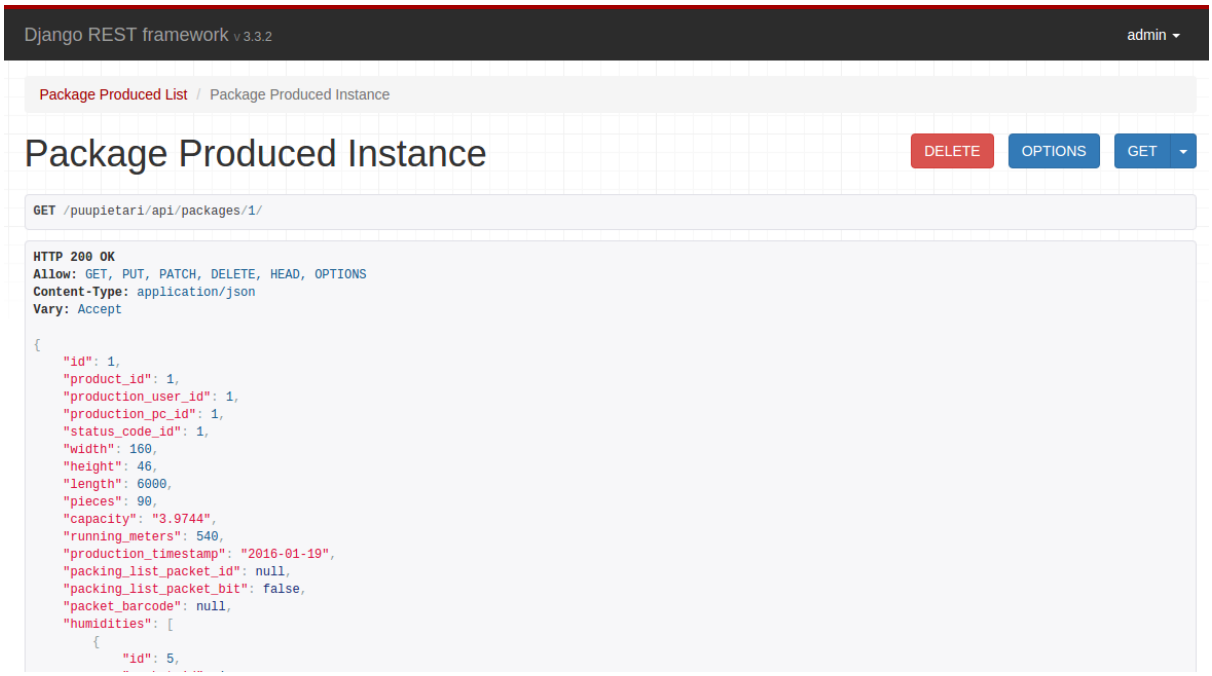
mään, rajapinnan rajoitusten määrittäminen, suodatusten monipuolinen muokattavuus ja automaattinen dokumentointi, mikä on mahdollista myös kolmannen osapuolen lisäosille.

DRF ei suoraan tue sisäkkäisiä resursseja, mutta siihen löytyi kolmannen osapuolen lisäosa, jolla tämä oli mahdollista toteuttaa. Tämä haluttiin toteuttaa sekä rajapinnan eri käyttötapausten, että myös ohjelmiston resurssien määrän vuoksi. Esimerkkinä sisäkkäisistä resursseista toimivat mm. puupakettien kosteusarvot. Lista puupaketeista on saatavilla osoitteesta “/packages/” ja yksittäinen puupaketti osoitteesta “/packages/1/”. Tämän jälkeen paketti 1:n kosteusarvot saadaan osoitteesta “/packages/1/humidities/”, joka listaa vain paketti 1:lle annetut kosteusarvot. Samaa toiminnallisuutta noudatettiin muidenkin sitä tarvitsevien resurssien kanssa, kuten lähetyslistojen tai tilauksien tuotteiden yhteydessä. Joissakin tapauksissa resurssit tehtiin saataville kahteen eri osoitteeseen. Esimerkiksi mainittakoon suora osoite listaamaan puupaketteja, ja toinen lähetyslistan sisäkkäisenä listauksena, jolloin esitetään vain ne paketit, mitkä kuuluvat tiettyyn lähetyslistaan.

Kaikkia ohjelman tietokantaresursseja ei tarjota rajapinnan yli muokattavaksi ja joitakin resursseja rajapinnasta on mahdollista vain lukea. Tällaisia resursseja ovat esimerkiksi lähetysten tilakoodit, yrityksen yhteystiedot ja käyttäjien työkoneet. Nämä resurssit eivät ole sellaisia, mitä muutetaan usein ja yleensä niiden muokkaukset tekee ylläpitäjä. Kyseisiä resursseja muokataan vain Djangon admin-sivuilta.

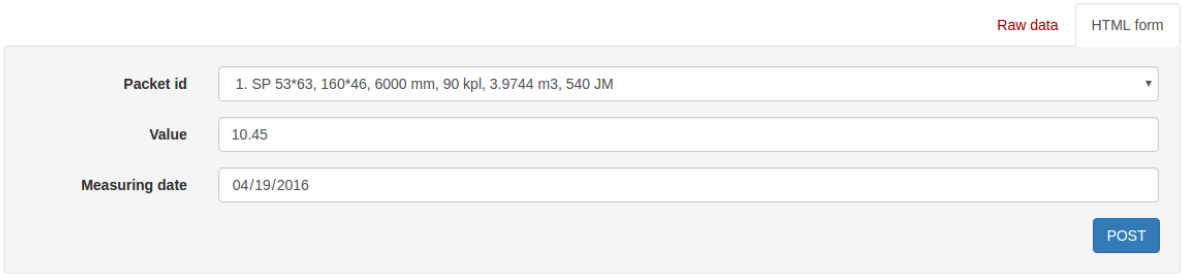
11.3.2 Dokumentointi

DRF muodostaa automaattisesti selaimella selattavat HTML-sivut, jotka kuvaavat rajapinnan rakennetta todella kattavasti. Kuvassa on esitetty, miltä REST rajapinnan yhteen resurssiin tekemä sivu näyttää selaimessa (kuva 29).



Kuva 29. REST-rajapinnan resurssit selaimessa

Rajapinnan selattavat sivut tarjoavat käyttäjälle automaattisesti generoidut lomakkeet, joilla asiakasohjelman kehittäjä voi lisätä resursseja rajapintaan ja testailla kuinka rajapintaa voidaan käyttää (kuva 30).



Kuva 30. Resurssien lisääminen rajapintaan selaimessa

Tämän lisäksi jos rajapinnan lähdekoodia on dokumentoitu kommenteilla, DRF purkaa kommentit tietyistä kohdista tarjotuille sivulle. Näillä valmiiksi tulevilla ominaisuuksilla asiakasohjelman liittäminen rajapintaan on paljon helpompaa ohjelmistokehittäjälle. Djangoon on myös saatavilla kolmannen osapuolen dokumentointivälineitä, jotka generoivat automaattisesti rajapinnasta erillisen web-pohjaisen dokumentin. Ohjelmistossa päädyttiin kuitenkin DRF:n omien dokumentointityökalujen lisäksi kirjoittamaan erillinen rajapintadokumentti, joka ku-

vaa sen resurssit ja niiden käytön. Dokumentin tarkoituksena on helpottaa uusien asiakasohjelmien kehitystä järjestelmään.

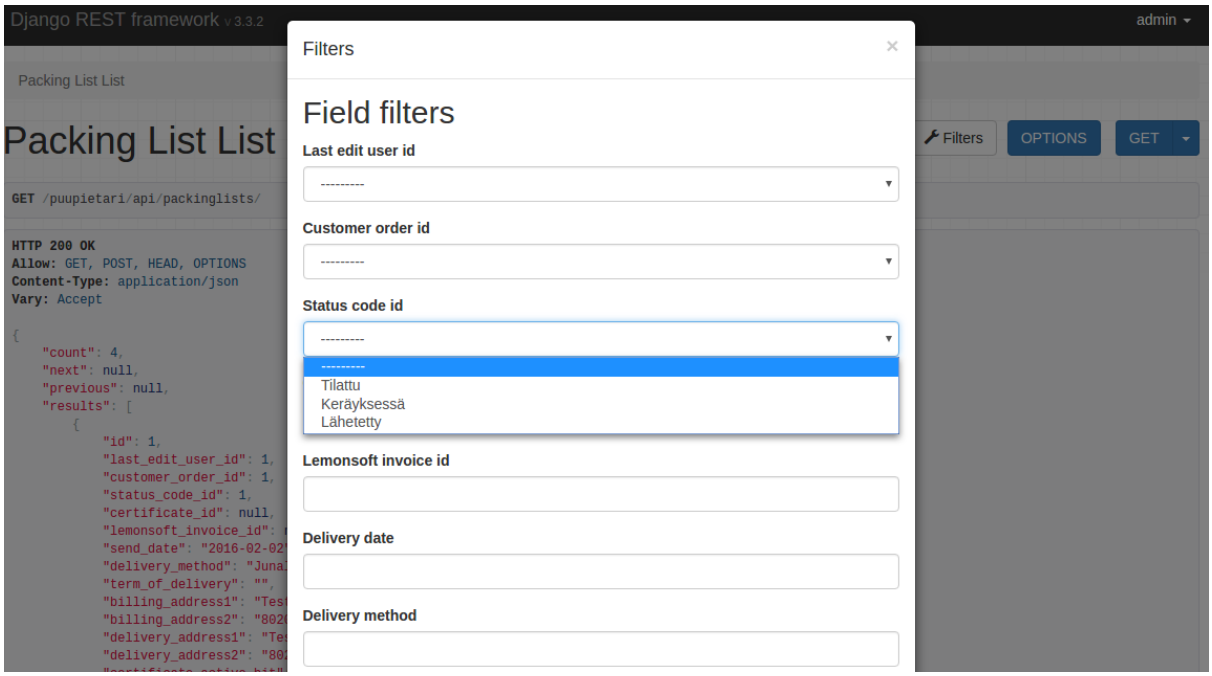
11.3.3 Suodatus ja sivutus

DRF tarjoaa rajapintaan todella hyvät ominaisuudet rajapintakyselyiden muokkaamiseen, mikä riippuu käyttötapauksesta ja siitä mitä tietoa sieltä halutaan. DRF hoitaa suodatuksen osoitteen GET-parametreilla ja parametrit ovat kokonaan rajapinta-kehittäjän päätettävissä. Tuki löytyy teksti-, numero- ja päivämäärä-hauille ja näihin on vielä saatavilla erilliset rajapinnan suodatukset. Näitä käyttämällä voidaan esimerkiksi noutaa kaikki lähetyslistat tietyltä aikaväliltä, tai ne puupaketit, joiden kosteusarvot sijoittuvat jollekin tietylle välille. Koodiesimerkki alla näyttää miltä DRF:n suodatuksen määrittävä luokka voi näyttää. Luokalle on määritetty minimi- ja maksimi-suodatuskentät ja ORM-mallin muut kentät löytyvät sen sisäisestä Meta-luokasta.

```
class HumidityFilter(rest_filters.FilterSet):
  min_measure_date = filters.DateFilter(name='measure_timestamp',
lookup_type='gte')
  max_measure_date = filters.DateFilter(name='measure_timestamp',
lookup_type='lte')

  class Meta:
    model = models.Humidity
    fields = [
      'packet_id',
      'value',
      'measure_timestamp',
      'min_measure_date',
      'max_measure_date',
    ]
```

DRF:n suodatusominaisuuksia rajapinta-kehityksessä toteutettiin monipuolisesti, jotta ne kattaisivat mahdollisimman paljon erilaisia käyttötapauksia. Kehitystyön aikana ilmaantui kuitenkin tapauksia, milloin suodatuskenttiä piti lisätä. Kuvassa on esitetty miltä DRF:n selaimella avattavassa rajapinnassa suodatusten selaaminen näyttää (kuva 31).



Kuva 31. Selaimessa rajapinnan suodatuskenttien tarkastelu

DRF tukee myös kyselyiden automaattista sivutusta jos saatujen tulosten määrä on tarpeeksi suuri. Sivutus on kokonaan muokattavissa DRF:n asetusten perusteella. Sieltä asetetaan miten monta resurssia listasta näytetään yhdellä sivulla. DRF antaa myös mahdollisuuden kyselyn tekevälle osapuolelle vaikuttaa sivutuksen suuruuteen “pagesize” GET-parametrilla.

11.3.4 Suojaus

Rajapinta olisi internetissä kaikkien saatavilla, jos palvelu julkaistaisiin sisäverkosta ulkoverkkoon. Kuitenkin sisäverkossakin rajapinnan suojaus tulee ottaa huomioon, että vain sallitut ja sisäänkirjautuneet käyttäjät pääsevät rajapintaan kiinni. Koska rajapintaa käytetään tulevaisuudessa mahdollisesti muillakin asiakasohjelmilla, päätettiin autentikointi toteuttaa tokeneilla, eikä sessiokirjautumista olisi edes ollut kaikissa käyttötapauksissa mahdollista käyttää.

Rajapintaan päätimme toteuttaa JWT-autentikoinnin. DRF:n mukana tulee valmius tokeni autentikoinnille, minkä huonoja puolia ovat, että tokenit eivät vanhene ja ne tallennetaan tietokantaan. JWT-tokenit asetetaan vanhentumaan, mutta se voidaan myös ottaa pois päältä tarpeen vaatiessa. Tokenit jotka vanhentuvat voidaan päivittää uuteen tokeniin ja vanhenemisaika voidaan nollata

takaisin alkutilanteeseen. JWT-tokeneita ei myöskään tallenneta tietokantaan, vaan aina kun uutta tokenia pyydetään, sellainen tehdään ohjelmistoon määritellyllä salausavaimella ja tokeni voidaan tallentaa esimerkiksi käyttäjän laitteelle paikallisesti. JWT-tokeni onkin paljon monipuolisempi vaihtoehto kuin DRF:n tarjoama vaihtoehto. Rajapinta on tarjolla kirjautumisen takana ja antamalla oikeat tunnukset, käyttäjä saa JWT tokenin käyttöönsä minkä perusteella käyttäjä pääsee käsiksi hänelle tarkoitettuihin resursseihin. JWT:n asetuksiin on määriteltävissä, kuinka pitkään tokeni on voimassa, ennen kuin se tulee päivittää tai kuinka pitkän ajan sisällä tokenia voi päivittää, ennen kuin käyttäjältä vaaditaan uusi sisäänkirjautuminen. Rajapintaan kirjautuessa osapuolten välillä vaihdetaan arkaa tietoa, kuten käyttäjänimi ja salasana. Tämä on myös otettu huomioon, tarjoamalla rajapinta ainoastaan HTTPS-protokollan ollessa käytössä.

11.3.5 Käyttöluvat ja rajoitukset

DRF tarjoaa valmiudet eri rajapintakutsujen rajoittamiseen käyttölupien mukaisesti. DRF voi hyödyntää Djangon tarjoamia käyttäjätasoja rajapinnassaan. Käyttäjäryhmiin voidaan määritellä, millä käyttäjätasolla on muokkausoikeudet resurssiin, kun muille käyttäjätasoille tarjotaan vain lukuoikeudet. Ohjelmaan toteutetut Lemonsoft laskutukseen liittyvät rajapintaliitokset, sallivat vain laskutukseen kuuluvan käyttäjäryhmän luoda laskuja lähetyslistoista Lemonsoft ERPiin.

Rajapinnan toteutuksessa hyödynsimme myös DRF:n rajapintakutsujen rajoittavia ominaisuuksia, niin käyttäjän kuin myös IP-osoitteiden perusteella. DRF:n mukana tämä toiminnallisuus tulee jo valmiiksi asennettuna, mutta se ei ole oletuksena käytössä. Ohjelmassa rajoitettiin kutsuja tietyltä käyttäjältä ja tietystä IP-osoitteesta määrään, mikä sopi ohjelman käytön rajoihin. Tällä varmistetaan palvelun toimivuutta serverin kapasiteetin suhteen ja myös estetään palvelunestohyökkäyksiä.

11.4 PDF-tulosteet

Ohjelmassa käytettiin PDF-tulosteita lähetyslistojen ja puupakettien tarralappujen tulostamiseen. PDF-tulosteet generoidaan serveripuolella aina kutsun tul-

lessa palvelimelle. Tiedostoja ei tallenneta niiden helpon ja nopean piirtämisen takia, milloin säästetään työvaiva tallenteiden manageroinnista. Lisäksi jos alkuperäistä tietoa muuttaa, täytyy tuloste joka tapauksessa piirtää uudestaan.

Pythonilla PDF-dokumenttien piirtämiseen käytettiin Pythoniin löytyvää Report-Lab-lisäosaa. Serverillä PDF-dokumentti palautetaan Djangon näkymästä, kun palvelimelle tulee kysely tiettyyn osoitteeseen. Osoitteeseen täytyi määrittää parametrina esimerkiksi puupaketille sen yksilöivä numero. Näin saatiin tulostettua juuri tälle paketille kuuluva pakettikortti. Osoite on muotoa “/pdf/package/2/”, jossa numero 2 tarkoittaa halutun paketin yksilöivää numeroa. Jos numeroksi laitetaan paketti mitä ei ole olemassa, saadaan palvelimelta vastaukseksi 404 status-koodi. Tulevaisuutta ajatellen tulosteisiin suunniteltiin myös tuki erilaisille valittaville raporttityypeille, mikä ei kuitenkaan valmistunut asiakkaalle asennettavaan ensimmäiseen versioon. Tämä on kuitenkin tarkoitus toteuttaa ohjelmiston tuleviin versioihin.

11.5 Turvallisuus

Koska opinnäytetyön tekijöistä kumpikaan ei ole lähtökohtaisesti web-ohjelmistojen turvallisuuden ammattilaisia, tarvittiin asian tarkasteluun tueksi ulkopuolista ja luotettavaa lähdetietoa. Pääasialliseksi turvallisuus-osion lähteeksi työhön valikoituikin OWASP.

OWASP eli The Open Web Application Security Project, on avoin ja voittoa tavoittelematon kansainvälinen järjestö. Sen tehtävänä on auttaa kehittämään, hankkimaan, käyttämään ja ylläpitämään sovelluksia joihin käyttäjä voi luottaa. Kaikki materiaali jota OWASP tarjoaa, on ilmaista ja avointa kaikille sovellusten turvallisuusasioista kiinnostuneille. OWASP:n materiaalista olikin suuri apu opinnäytetyöhön liittyvissä turvallisuusasioissa.

Turvallisuuden ollessa yksi tärkeimpiä osia sovelluksissa, joissa käsitellään yrityksen sisäistä herkkää tietoa, lähdimme tarkastelemaan sitä tekemällä sovellukselle uhkamallin (liite 1). Koska sovellus toteutettiin web-pohjaisena, uhkamallin toteutuksen apuna käytettiin OWASP:n tarjoamaa uhkamalli-pohjaa.

11.6 Uhkamallin analyysi

Toteutetun uhkamallin perusteella uhkatyypit ja niiden vastatoimet jaoteltiin kahdeksaan eri alakohtaan. Nämä alakohdat olivat: autentikointi, oikeudet, konfiguraatioiden hallinta, datan suojaaminen tietokannoissa ja datan siirron aikana, datan tai muuttujien validointi, virheenkäsittely ja poikkeuksien hallinta, käyttäjien ja istuntojen hallinta sekä viimeisenä, tiedon tarkastaminen ja sen tallentaminen.

Autentikointi eli käyttäjän todennus, tarkoittaa operaatiota missä käyttäjän identiteetti varmennetaan. Ohjelmistossa autentikointi on käytössä jokaisessa käyttäjän tekemässä toiminnossa ja saatavilla olevien resurssien käytössä, eikä ohjelmistoa täten voi käyttää ilman sisäänkirjautumista. Sisäänkirjautumisen yhteydessä sovelluksen käyttäjälle luodaan JWT, jota käytetään applikaation sisäisessä siirtymisessä tai resursseja pyytäessä käyttäjän identiteetin sekä hänen oikeuksien varmistamiseksi. Kirjautumistiedot, mukaanluettuna tokenit joita käytetään autentikointiprosessissa, ovat suolan kanssa salattuina tietokannassa, ja järjestelmä käyttää jokaisella sivulla datan siirtämisen suojana HTTPS protokollaa. Django käyttää salasanojen tallentamisessa tietokantaan PBKDF2 avaimen derivaatiofunktiota. Tallennettava salasana muodostetaan hashaus algoritmista, algoritmin iteraatioiden määrästä, satunnaisesta suolasta sekä salasana hashin tuloksesta. Käyttäjän unohtaessa salasanan, sen uusimista pyydetään admin-henkilöiltä tai ylläpidolta, ja kaikkien salasanojen tulee noudattaa vahvan salasanan käytäntöä.

Oikeuksilla tarkoitetaan käyttäjien oikeuksia tehdä eri toimintoja ohjelmistossa. Ohjelmistossa on kolme eri käyttäjätasoa, joita ovat työntekijä, admin ja ylläpito. Jokaiselle käyttäjätasolle on määritelty oikeudet suorittaa niitä prosesseja, joita käyttäjän on tarkoitus tehdä. Työntekijän oikeudet ovat pienimmät ja ne oikeuttavat vain normaaliin työntekoon liittyviin asioihin järjestelmässä. Admin-henkilöillä, eli tässä tapauksessa työpaikan esimiehillä, on oikeudet kaikkiin muihin resursseihin paitsi käyttäjien lisäämiseen, joihin oikeudet riittävät vain ylläpidolla.

Konfiguraatioiden eli asetusten hallintaan käytetään admin-sivuja. Niiden kautta voidaan hallita kaikkea ohjelmistosta löytyvää tietoa, kuten esimerkiksi luoda

ohjelmistoon uusia tilauksien tila-määrityksiä. Asetusten hallintaan oikeudet löytyvät vain admin-henkilöiltä ja ylläpidolta, ja näiden roolien ero on jo aiemmin mainittu käyttäjien lisääminen. Lisäksi kaikki admin-sivujen tapahtumat tallennetaan lokitiedostoihin. Virhetilanteissa ylläpidon on helppo löytää virheeseen johtanut tapahtuma.

Datan suojaamiseen tietokannoissa ja datan siirron aikana käytetään siis standardoituja salausmetodeja ja kaikki tietoliikenne salataan HTTPS yhteydellä. Salausalgoritmien käyttämät secretit pidetään suojattuina siirron aikana ja niiden varastointi tehdään ohjelmiston omaan turvalliseen tietokantaan johon hyökkääjät eivät voi päästä käsiksi.

Datan tai muuttujien validointi eli niiden vahvistaminen tapahtuu aluksi järjestelmän “model.py” tiedostoissa django-kehyksen käyttäessä MVC(Model View Controller) arkkitehtuuria. Niissä käytettäville malleille saadaan määriteltyä muun muassa datan tyyppi, formaatti, maksimipituus ja alue. Näin saadaan varmistettua ohjelmiston käyttäjältä saadun syötteen oikea muoto ja samalla käyttäjän virheet minimoituvat. URL:iin syötettävät muuttujien arvot, esimerkiksi REST-rajapinnan yhteydessä sekä käyttöliittymään siirtyvä tieto HTML-enkoodataan, eli erikoismerkit muutetaan vastaamaan HTML-merkistöä, millä eliminoidaan mm. XSS-hyökkäystyyppi.

Virheenkäsittely ja poikkeuksien hallinta käsitellään jäsennetysti jolloin ongelmat ovat helpointa jäljittää. Kun ohjelma on asennettu asiakkaan työympäristöön, yksityiskohtaiset kehitysvaiheen virheviestit otetaan pois käytöstä ja ne korvataan geneerisillä HTML virheviesteillä. Tällöin ohjelma ei anna ulos mitään ylimääräistä tai arkaa tietoa käyttäjälle. Tarkasteltua tietoa ei myöskään tallenneta lokitiedostoihin.

Käyttäjien ja istuntojen hallinnassa on otettu huomioon evästeet ja tokenit. Arkaa tietoa ei tallenneta evästeisiin ja tieto pidetään aina salattuna. Evästeet ja tokenit vanhentuvat tietyn aikamääreen kuluessa sekä uloskirjautumisen yhteydessä. Myös yhtenä suojauskeinona käyttäjien ja istuntojen hallinnassa käytetään jo aiemmin puhuttua HTTPS-protokollaa.

11.7 Testaus

11.7.1 Testien kirjoittaminen

Ohjelmistossa käytettiin testaamiseen ohjelmallisia testejä, joista Jenkins myös loi testien kattavuusraportit. Testejä käytettiin myös ohjelmiston kehittäjien omissa kehitysympäristöissä tarkistamaan ja takaamaan uusien ohjelmiston osien toimivuus, ennen muutetuiden osien päärepositoryyn lähettämistä. REST-rajapintaa testattiin paljon kun asiakaspuolen AngularJS ohjelmiston kehitystyö oli käynnissä, minkä johdosta syntyi paljon uusia vaatimuksia rajapinnan käyttötapauksiin. Lisäksi yhdeksi testaamisen osioksi muodostui kommunikointi asiakkaan kanssa. Ohjelmiston käyttöliittymää näytettiin asiakkaalle lyhyin väliajoin, jolloin saatiin välitöntä palautetta ohjelmiston käytettävyydestä, sen toiminnallisuudesta ja mahdollisista parannusehdotuksista.

Django sisältää todella kattavat testit jo itsessään, mitkä kykenevät kattamaan hyvin ohjelmoijan tekemää koodia yleisellä tasolla. Ei tietenkään itse kirjoitettua logiikkaa, vaan esimerkiksi että osoite vastaa ja mitään ei ole konfiguroitu väärin. Django tarjoaa valmiit komponentit testien kirjoittamiseen ja hoitaa myös itse niiden ajamisen “manage.py”-työkalun kautta. Djangossa testit sijaitsevat sen paketin alla jonka toiminnallisuutta testit testaavat. Konkreettisemmin niiden sijainti on “tests.py” tiedostoissa, jotka sisältävät testaukseen määritetyn luokan, jolle määritetään erilaiset testi-funktiot. Alla nähdään esimerkki siitä miltä yksi testi-funktio Django projektissa näyttää.

```
def test_if_user_based_url_is_working(self):
    """
    Response should be equal to 200
    if the Lemonsoft WSDL-Userservices url is working
    """
    response = urllib.request.urlopen(self.user_based_url).getcode()
    self.assertEqual(response, 200)
```

Esimerkin funktio testaa onko Lemonsoft ERP:n tarjoama SOAP-rajapinta saatavissa. Jos HTML status-koodi ei ole yhtäkuin 200, eli pyyntö ei ole onnistunut, tällöin testi ei mene läpi. Testejä kirjoitettiin ohjelmistokehityksen aikana kaikkien tärkeimpien toiminnallisuuksien tarkastamiseen niin AngularJS-, kuin myös

Django-projektissa. Ajan puutteen vuoksi kaikille ohjelmiston osioille ei voitu kirjoittaa testejä, ohjelmistokehityksen prioriteettien painottuessa muualle.

Jenkins-julkaisuprosessin käynnistyessä kaikki testit suoritetaan automaattisesti. Jos yksikin testi epäonnistuu, prosessi pysäytetään ja julkaisua uusimpaan versioon ei tapahdu. Jenkinsin konsolista on mahdollista tarkistaa, mikä prosessin aikana meni vikaan ja mitkä testit eivät menneet läpi onnistuneesti. Alla nähdään esimerkki miltä konsolitulostus Jenkinsissä näyttää kun testit onnistuvat.

```
OK
Creating test database for alias 'default'...
<HttpResponse status_code=200, "application/pdf">
Destroying test database for alias 'default'...
Storing coverage info...
Executing django_jenkins.tasks.run_pyflakes...
Executing django_jenkins.tasks.run_pep8...
Done
[Cobertura] Publishing Cobertura coverage report...
Publishing Cobertura coverage results...
Cobertura coverage report found.
Finished: SUCCESS
```

11.7.2 Tietokanta ja testien lähtötiedot

Django luo testejä varten oman tietokannan ja tuhoaa sen kun testit on suoritettu. Tietokanta on mahdollista myös säilyttää, mikä nopeuttaa osaltaan testien suorittamista kun tietokantaa ei aina luoda uudelleen. Testausprosessin luoman oman testitietokannan ansiosta, ohjelman käyttämän tietokannan oikeat tiedot eivät tule muuttumaan testien aikana ja testeille voidaan antaa oikean ohjelman tietokannasta riippumatonta tietoa. Testausprosessi palauttaa jokaisen testin välillä testitietokannan takaisin lähtötilanteeseen. Näin jokainen testitapaus on yksilöllinen ja riippumaton toisista testeistä.

Djangon testeille voidaan asettaa lähtötieto-dataa, mikä ladataan testitietokantaan ennen testien suorittamista. Aikaisemmissa osioissa käytiin läpi kuinka lähtötietojen luominen tapahtuu Djangolla. Alla nähdään esimerkki testiluokan alustus-funktiosta, joka määrittää joitain muuttujia ennen testien suorittamista.

```
class UrlStates(TestCase):
  fixtures = [‘LemonsoftWS_testdata.json’]
  def setUp(self):
```

```
# Set up data for the whole TestCase
self.ser = Service()
self.user_based_url = self.ser.settings.user_based_url
self.data_based_url = self.ser.settings.data_based_url
self.user_based_url_in_use = self.ser.user_based_url_in_use
# Wrong Credentials
self.lemonsoft_settings = LemonsoftSettings()
self.wrong_username = 'wrongusername'
self.wrong_password = 'wrongpassword'
```

Huomaa luokan määrittelyn jälkeen "fixtures = ['LemonsoftWS_testdata.json']"-rivi. Se määrittelee Djangon testiluokalle, mitä lähtötieto-tiedostoa se käyttää tiedon lataamisessa testausprosessin luomaan testitietokantaan. Tämän jälkeen Django suorittaa luokan testit yksi kerrallaan.

12 Pohdinta

12.1 Asiakkaan hyödyt

Ohjelmiston tuloksena Puupietari Oy:n tuotanto siirtyi kokonaan ohjelmalliseen toimintamalliin. Näin yritys pääsi eroon excel-taulukon sekä paperisten asiakirjojen epäkäytännöllisestä toimintamallista. Työntekijöiden ei enää tarvitse kuljettaa saapuneiden tai valmistuneiden tilausten tietoa käsin toimiston ja tuotantopisteen välillä, vaan tiedot ovat saatavilla työntekijöille riippumatta työpisteestä. Tilausten ja lähetysten hallinta on selkeämpää ja täten työntekijät löytävät niihin liittyvät tiedot helpommin. Työpisteitä pystytään lisäämään, jolloin työntekoa saadaan tehostettua. Kaikki aikaisemmat tuotantoon tarvitut resurssit löytyvät nyt sähköisessä muodossa jokaiselta käyttöönotetulta työpisteeltä, kun lähtötilanteessa käytössä oli vain yksi tietokone ja sen yksi excel-tiedosto.

Asiakkaiden tilaukset ovat suoraan nähtävissä ohjelmasta entisten paperitulosteiden sijaan. Ohjelmasta näkee suoraan mitä lautatavaraa tilauksia varten tulee tuottaa. Työpisteillä tuotetut lautapaketit kirjataan ohjelmistoon, mikä samalla pitää historiadataa tuotannon tapahtumista sitä tarvitseville. Tuotettuja lautapaketteja voidaan lisätä tilauksiin ja ne merkitään toimitetuiksi, kun tilaukset on tehty valmiiksi. Toimitetuista tilauksista voidaan muodostaa lasku suoraan Lemonsoft ERPiin yhdellä napin painalluksella. Tällöin laskujakaan ei tarvitse teh-

dä enää käsin, vaan ne muodostuvat automaattisesti suoraan ohjelmasta, jonka jälkeen muodostetut laskut vain hyväksytään Lemonsoft ERPissä.

Ohjelmisto siirtää työntekijöiden aiemmin tarvitsemien prosessien viemää aikaa, käytettäväksi muuhun tarpeellisempaan työhön. Työpisteiden lisääminen on tehty helpoksi, mikä myös mahdollistaa asiakkaan tuotannon laajentamisen ja ylläpidon tulevaisuudessa.

12.2 Kohdatut vaikeudet

Projektin aikana vaikeudeksi muodostui työn sisältämä suuri tietoperusta ja sen opetteleminen. Tähän kuuluivat käytetyt työkalut, tekniikat ja myös erilaiset projektin kehityskaaren aikana nousseet ohjelmalliset ongelmat. Monet työkalut ja tekniikat olivat tekijöille uusia, eikä niitä oltu käsitelty koulun aikana. Tämä osaltaan vei opinnäytetyöntekijöiltä suuren osan työn kokonaistuntimäärästä.

Oinoselle alkuvaiheen ongelmiksi muodostuivat täysin uudenlaisten työkalujen käyttäminen. Näitä olivat esimerkiksi versionhallinta sekä projektinhallinta, ja niiden sisältämät työvaiheet ja hallintamenetelmät isomman työryhmän sisällä. Ajan myötä eri prosessit ja vaiheet kuitenkin ns. rutinoituivat, minkä jälkeen työkalujen käyttäminen kehittyi luontevaksi. Python-ohjelmointikielen opettelu sekä Django-kehykseen perehtyminen veivät oman aikansa. Myös ohjelmistokehityksen aikana syntyneet ongelmat ja niiden ratkominen, vaativat paljon tutkimustyötä ja asiaan perehtymistä.

Mustosella oli jo aiempaa kokemusta useimmista käytössä olleista työkaluista, Asmecon muiden asiakasprojektien kautta. Esimerkiksi versionhallinnan pystyttäminen yrityksen omalle serverille oli toteutettu jo ennen opinnäytetyön aloittamista ja sen käyttäminen oli tullut jo tutuksi. Suuren työn projektin alkuvaiheessa vaati työkalujen yhteentoimivuuden saavuttaminen. Tämä sisälsi Kallithean toimimisen Redmine-projektinhallintaohjelmiston kanssa, minkä lisäksi Kallithea tuli saada toimimaan myös Jenkinsin kanssa. Suurin osa opiskelusta koostui kuitenkin ohjelmistoon liittyvästä suunnittelusta, projektin ollessa Mustoselle tähän mennessä yrityksen suurin toteutettava asiakasprojekti. Eniten vaikeuksia tuottivat kehyksen toiminnan opiskelu sekä joidenkin suunnitteluun liittyvien päätösten vertailu ja niiden tekeminen.

Kaikin puolin projekti vaati kummaltakin opinnäytetyön tekijältä erittäin paljon uuden tiedon opiskelua. Se oli haastava tekijöiden lähtötasoon suhteutettuna, mutta myös opettava ja paljon antanut kokemus molemmille tekijöistä. Toinen tekijöistä oli parempi jossain asiassa kuin toinen, mutta molemmat myös oppivat toisiltaan paljon.

12.3 Työn ja kirjoittamisen jaottelu

Työn jaottelu opinnäytetyössä onnistui suunnitellusti. Mustoselle tuli työn aikana lisävastuuta työn muusta organisoinnista, hänen toimiessa työssä myös toimeksiantajana. Tämä sisälsi esimerkiksi tapaamisten järjestämiset Parikan sekä Puupietari Oy:n edustajan kanssa.

Opinnäytetyöraportin kirjoittaminen jakautui jotakuinkin tekijöiden vastuualueiden mukaan. Eriteltynä Mustonen kirjoitti seuraavat osat:

- ohjelmistokehitystyön hallinnointi
- käytetyt tekniikat ja ohjelmiston rakenne
- käytetyt ohjelmistokehitystyökalut
- Lemonsoft integraatio
- tietoperusta
- toteutettu ohjelmisto.
- pohdinta.

Oinonen kirjoitti:

- tietoperusta
- toteutettu ohjelmisto
- pohdinta
- liite.

Jos listoissa mainitaan samoja otsikoita, niin molemmat olivat mukana osioiden kirjoittamisessa. Raportin viimeistelyn ja lopullisen oikolukemisen teki suurelta osin Oinonen.

Ohjelmistokehitys pysyi suurin piirtein omilla vahvuusalueillaan. Kehitystä tehtiin myös paljon omien vastuualueiden ulkopuolella, sen mukaan mikä sillä hetkellä

oli tärkeää ohjelmistokehityksen kannalta. Kaikissa työvaiheissa kummatkin tekijät pyrkivät tekemään tiivistä yhteistyötä päästäkseen parempaan lopputulokseen.

12.4 Tulevaisuuden näkymät

Asmeco jatkaa ohjelmiston kehittämistä ja tulee toimimaan myös ohjelmiston ylläpitäjänä asiakkaalle. Opinnäytetyöprojekti auttoi ja kannusti työnteossa sekä tietoperustan ja työkalujen opettelussa, mistä Asmeco sai paljon hyötyä yrityksen tulevaisuutta ajatellen. Lisäksi ohjelmistoon toteutettiin uudelleenkäytettäviä komponentteja, joita yritys tulee hyödyntämään myös tulevissa asiakasprojekteissa. Tulevaisuuden näkymät toteutetulle ohjelmistolle ovat hyvät. Työhön valitut tekniikat ja työkalut todettiin työn aikana niin hyviksi ja työtä helpottaviksi, että Asmeco tulee käyttämään niitä myös tulevaisuudessa.

12.5 Mitä työstä opimme?

Yhteenvetona työ antoi kummallekin opinnäytetyön tekijälle suuren määrän kokemusta, mitä pelkkä koulun käynti ei pysty tarjoamaan. Molemmat oppivat mitä ison projektin toteuttaminen vaatii ja mitä kaikkea sen eri työvaiheet sisältävät. Tekijät oppivat myös paljon ohjelmistokehityksessä käytetyistä työkaluista, tekniikoista, ohjelmointikielistä, kehyksistä ja siitä, mitä vaaditaan, kun työskennellään projektissa missä on monta tekijää.

Lähteet

Ambysoft Inc. 2013. Mapping Objects to Relational Databases: O/R Mapping In Detail. 3.5.2016.

Alchin, M. 2013. Pro Django, 2nd Edition. New York: Apress.

Auth0. 2016. JWT Debugger. https://jwt.io/. 1.5.2016.

Barrow, B. 2013. Kickstart Your AngularJS Development with Yeoman, Grunt and Bower. https://www.sitepoint.com/kickstart-your-angularjs-development-with-yeoman-grunt-and-bower/. 10.5.2016.

CentOS. 2016. CentOS Bug Tracker. https://bugs.centos.org/view.php?id=9391. 4.5.2016.

Christensen, E. Curbera, F. Meredith, G. Weerawarana, S. 2000. Web Services Description Language (WSDL) 1.0. http://xml.coverpages.org/wsdl20000929.html. 5.5.2016.

Django. 2016. Django Documentation – Models. https://docs.djangoproject.com/en/1.9/topics/db/models/. 4.5.2016.

Django. 2016. Django Documentation – Databases. https://docs.djangoproject.com/en/1.9/ref/databases/. 4.4.2016.

DrapsTV. 2015. What is Cross Site Scripting?. https://www.youtube.com/watch?v=M_nIIcKTxGk&list=PL1A2CSdiySGIRec2pvDMkYNi3iRO89Zot&nohtml5=False. 4.4.2016.

Django Rest Framework. 2016. Django Rest Framework Documentation. http://www.django-rest-framework.org/. 1.5.2016.

Fowler, M. 2006. Continuous Integration. http://www.martinfowler.com/articles/continuousIntegration.html. 1.5.2016.

GRUNT. 2016. Grunt Documentation. http://gruntjs.com/getting-started. 1.5.2016.

Greenfield, D. & Greenfield, A. 2015. Two-scoops-of-django-1-8. Los Angeles: Two Scoops Press.

Gudgin, M. 2007. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). https://www.w3.org/TR/soap12/. 5.4.2016.

Heaton, R. 2014. How does HTTPS actually work?. http://robertheaton.com/2014/03/27/how-does-https-actually-work/. 25.4.2016.

Jorgensen, P. 1995. Software Testing - A Craftsman’s Approach. Boca Raton: Auerbach Publications.

Jauker, S. 2014. 10 Best Practices for Better RESTful API. http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/. 1.5.2016.

JSON. 2016. Introducing JSON. http://www.json.org/. 8.5.2016.

Kanakiya, J. 2016. Grunt-wiredep Readme. https://github.com/stephenplusplus/grunt-wiredep/blob/master/readme.md. 9.5.2016.

Kesteren, A. 2014. Cross-Origin Resource Sharing. W3C. https://www.w3.org/TR/cors/. 16.4.2016.

Kmmbvnr. 2016. django-jenkins Tutorial. https://sites.google.com/site/kmmbvnr/home/django-jenkins-tutorial. 2.5.2016.

Kaner, C. 2006. Exploratory Testing. http://www.kaner.com/pdfs/ETatQAI.pdf. 6.5.2016.
Lutz, M. 2014. Python Pocket Reference, 5th Edition. Sebastopol: O'Reily Media.
LeBlanc, D. & Howard, M. 2001. Writing Secure Code. Redmond: Microsoft Press.
Malakoff, K. 2016. Knockback.js. http://kmalakoff.github.io/knockback/. 4.5.2016.
Mustonen, M. & Parikka, A. 2015. Tuntiarvio ja vaatimukset. Asmeco. 12.6.2016.
Microsoft. 2016. What's New in Windows Communication Foundation 4.5. Microsoft. https://msdn.microsoft.com/en-us/library/dd456789%28v=vs.110%29.aspx. 12.7.2016.
NPM. 2016. NPM Documentation. https://docs.npmjs.com/. 4.5.2016.
OWASP. 2016. Cross-Site Request Forgery (CSRF). OWASP. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29. 13.4.2016.
OWASP. 2016. Cross-site Scripting (XSS). OWASP. https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29. 4.4.2016.
Percival, H. 2014. Test-Driven Development with Python. Sebastopol: O'Reily Media.
Pose, A. 2014. Cookies vs Tokens. Getting auth right with Angular.JS. https://auth0.com/blog/2014/01/07/angularjs-authentication-with-cookies-vs-token/. 28.5.2016.
PyPA. 2014. PIP Documentation. https://pip.pypa.io/en/stable/. 01.2016.
Python Software Foundation. 2016. Installing Python Modules¶. https://docs.python.org/3.6/installing/index.html. 7.5.2016.
Pan, J. 1999. Software Testing. https://users.ece.cmu.edu/~koopman/des_s99/sw_testing/. 4.5.2016.
Rescorla, E. 2000. HTTP Over TLS. https://tools.ietf.org/html/rfc2818. 25.4.2016.
RestApiTutorial.com. 2016. Using HTTP Methods for RESTful Services. http://www.restapitutorial.com/lessons/httpmethods.html. 12.5.2016.
Sahni, V. 2016. Best Practices for Designing a Pragmatic RESTful API. http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api. 1.5.2016.
Spolsky, J. 2015. Making Wrong Code Look Wrong. http://www.joelonsoftware.com/articles/Wrong.html. 10.5.2016.
SUDS. 2016. SUDS Documentation. https://fedorahosted.org/suds/wiki/Documentation. 1.5.2016.
Santamaria, J. 2015. The Single Page Interface Manifesto. http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php. 2.5.1026.
Turner, K. 2014. How Django REST Framework Changed My Life. http://ngenworks.com/technology/how-django-rest-framework-changed-my-life/. 2.5.2016.
TodoMVC. 2016. Helping you select an MV* framework. http://todomvc.com/. 2.5.2016.
Weinstock-Herman, E. 2013. AngularJS vs Knockout – SPA Routing/History (8 of 9). LessThanDot. http://blogs.lessthandot.com/index.php/webdev/uidevelopment/angularjs-vs-knockout-spa-routing-history-8/. 6.5.2016.

Wes. 2014. What are best practices for REST nested resources. StackOver-Flow. http://stackoverflow.com/questions/20951419/what-are-best-practices-for-rest-nested-resources. 3.5.2016.

W3Schools. 2016. XML Soap. W3Schools. http://www.w3schools.com/xml/xml_soap.asp. 5.4.2016.

W3C. 2016. Token Based Authentication -- Implementation Demonstration. W3C. https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/. 4.5.2016.

W3Schoolcs. 2016. SQL Injection. http://www.w3schools.com/sql/sql_injection.asp. 4.4.2016.

Yeoman.io. 2016. Getting started with Yeoman. http://yeoman.io/learning/index.html. 3.5.2016.

Uhkamalli (Application Threat Modeling)

Uhkamallin tiedot(Threat Model Information)	
Ohjelman versio:	1.0
Kuvaus:	Sovellus on toiminnanohjausjärjestelmä puutavaran tuottajalle, minkä avulla: • luodaan lähetykset • tulostetaan kolleihin pakettikortit ja tilauksiin lähetyslistat • hallinnoidaan tilauksia, paketteja ja laskuja. Ohjelmistolla on kolme eri tasoa käyttäjille. 1. Työntekijä 2. Admin a. Esimiehet (muokkaus- ja lisäysoikeudet kaikkiin paitsi käyttäjätietoihin) 3. Superuser/ylläpito . Palvelun ylläpitäjät (muokkaus- ja lisäysoikeudet kaikkiin resursseihin) Ohjelma on rakennettu siten, että sitä voidaan käyttää turvallisesti myös julkisessa verkossa, vaikkakin tässä tapauksessa niin ei tapahdu sovelluksen toimiessa asiakkaan omassa sisäisessä verkossa.
Dokumentin omistaja:	Sami Oinonen
Osallistujat:	Mauri Mustonen

Ulkoiset riippuvuudet (External Depencies)	
ID	**Kuvaus**
1	Ohjelmisto toimii Windows-palvelimella joka sijaitsee asiakkaan lähiverkossa(ei ole yhteydessä ulkoverkkoon)
2	Tietokanta on MYSQL-muotoa ja sijaitsee samalla palvelimella
3	Lemonsoft ERP ja SOAP-palvelut sijaitsevat pilvipalvelimella joihin päästään käsiksi HTTPS-yhteydellä (TLS/SSL)

Tulokohdat (Entry points)			
ID	**Nimi**	**Kuvaus**	**Käyttäjätasot**
1	HTTPS Portti	Toiminnanohjausjärjestelmä toimii aina HTTPS-protokollan kautta. Kaikki sovelluksen sivut	• Käyttäjä väärillä kirjautumistiedoilla (1) • Käyttäjä oikeilla kirjau-

Uhkamalli (Application Threat Modeling)

		käyttävät tätä tulokohtaa.	tumistiedoilla (2) • Työntekijä (3) • Admin (4) • Superuser/ylläpito (5)
1.1	Kirjautumis-sivut	Toimii tulokohtana kaikille sovellusta käyttäville henkilöille.	• Käyttäjä väärillä kirjautumistiedoilla (1) • Käyttäjä oikeilla kirjautumistiedoilla (2) • Työntekijä (3) • Admin (4) • Superuser/ylläpito (5)
1.1.1	Kirjautumis-funktio	Vastaanottaa käyttäjän syötteen kirjautumistietoina ja vertaa niitä tietokannasta löytyviin käyttäjiin.	• Käyttäjä väärillä kirjautumistiedoilla (1) • Käyttäjä oikeilla kirjautumistiedoilla (2) • Työntekijä (3) • Admin (4) • Superuser/ylläpito (5)
1.2	Paketit-sivu	Näyttää valmistetut paketit. Voidaan luoda uusia paketteja.	• Työntekijä (3) • Admin (4) • Superuser/ylläpito (5)
1.3	Tilaukset-sivu	Näyttää kaikki työn alla olevat ja valmiit tilaukset. Voidaan luoda uusia tilauksia.	• Työntekijä (3) • Admin (4) • Superuser/ylläpito (5)
1.4	Lähetyslistat-sivu	Näyttää kaikki lähetyslistat. Voidaan luoda uusia lähetyslistoja.	• Työntekijä (3) • Admin (4) • Superuser/ylläpito (5)
1.5	Admin-sivut	Sivustolta voidaan muokata, lisätä tai poistaa kaikkia sovelluksen sisältämiä resursseja.	• Käyttäjä väärillä kirjautumistiedoilla (1) • Käyttäjä oikeilla kirjautumistiedoilla (2) • Admin (4) • Superuser/ylläpito (5)
1.5.6	Kirjautumis-funktio	Vastaanottaa syötteenä käyttäjätunnuksen ja salasanan, minkä jälkeen vertaa niitä tietokannasta löytyviin käyttäjä ja salasana pareihin.	• Admin (4) • Superuser/ylläpito (5) • Käyttäjä väärillä kirjautumistiedoilla (1) • Käyttäjä oikeilla kirjautumistiedoilla (2)

Osat(Assets)			
ID	**Nimi**	**Kuvaus**	**Käyttäjätasot**
1	Sovelluksen käyt-	Työntekijöiden käyttämät	

Uhkamalli (Application Threat Modeling)

	täjät eli työntekijät	ominaisuudet	
1.1	Käyttäjän kirjautumistiedot	Tunnukset joita työntekijä käyttää sovellukseen kirjautumiseen.	• Työntekijä (3) • Superuser/ylläpito (5) • Palvelimen työntekijän prosessit(6) • Tietokanta: Käyttäjän lukutoiminnot(7) • Tietokanta: Käyttäjän luku- ja kirjoitustoiminnot(8)
1.2	Käyttäjän yksityinen tieto	Toiminnanohjausjärjestelmäässä käyttäjätunnuksen luomiseen vaaditaan nimi, salasana ja sähköposti.	• Työntekijä (3) • Superuser/ylläpito(5) • Palvelimen työntekijän prosessit(6) • Tietokanta: Käyttäjän lukutoiminnot(7) • Tietokanta: Käyttäjän luku- ja kirjoitustoiminnot(8)
2	Admin-sivujen käyttäjät	Admin henkilöiden ja Superuser/ylläpidon ominaisuudet.	
2.1	Käyttäjän kirjautumistiedot	Tunnukset joita admin tai superuser/ylläpito henkilöt käyttävät admin-sivuille kirjautumiseen.	• Admin (4) • Superuser/ylläpito (5) • Tietokanta: Käyttäjän lukutoiminnot(7) • Tietokanta: Käyttäjän luku- ja kirjoitustoiminnot(8)
2.2	Käyttäjän yksityinen tieto	Käyttäjätunnuksen luomiseen vaaditaan nimi, salasana ja sähköposti.	• Admin (4) • Superuser/ylläpito (5) • Tietokanta: Käyttäjän lukutoiminnot(7) • Tietokanta: Käyttäjän luku- ja kirjoitustoiminnot(8)
3	Järjestelmä	Järjestelmän ominaisuudet	
3.1	Sovelluksen saatavuus	Sovelluksen tulee olla aina työntekijän käytettävissä.	• Superuser/ylläpito (5)
3.2	Työntekijän prosessit	Työntekijä voi suorittaa työn eri vaiheita sovelluksessa	• Superuser/ylläpito (5) • Palvelimen työntekijän prosessit(6)
3.3	Kyky suorittaa SQL-lauseita tieto-	SQL-komennot kuten SELECT, INSERT ja UPDATE	• Superuser/ylläpito (5)

Uhkamalli (Application Threat Modeling)

	kannassa		• Palvelimen työntekijän prosessit(6) • Tietokanta: Käyttäjän lukutoiminnot(7) • Tietokanta: Käyttäjän luku- ja kirjoitustoiminnot(8)
4	Sovellus	Sovellukseen liittyvät ominaisuudet	
4.1	Kirjautumis-istunto	Käyttäjän kirjautuminen sovellukseen ja istunnon ylläpitäminen.	• Työntekijä (3) • Admin (4) • Superuser/ylläpito (5) • Käyttäjä oikeilla kirjautumistiedoilla (2)
4.2	Suora pääsy tietokantapalvelimelle	Hallintaoikeudet tietokannan kaikkeen muuhun dataan paitsi käyttäjätietoihin	• Admin (4) • Superuser/ylläpito (5)
4.3	Kyky luoda uusia käyttäjiä	Kyky lisätä uusia käyttäjiä tietokantaan	• Superuser/ylläpito (5)

Luottamustasot (Trust Levels)		
ID	**Nimi**	**Kuvaus**
1	Käyttäjä väärillä kirjautumistiedoilla	Palvelun käyttäjä joka yrittää kirjautua järjestelmään väärillä tunnuksilla.
2	Käyttäjä oikeilla kirjautumistiedoilla	Palvelun käyttäjä joka kirjautuu järjestelmään olemassaolevilla tunnuksilla.
3	Työntekijä	Sovellukseen kirjautunut työntekijä voi luoda ja hakea paketteja, lähetyslistoja sekä tilauksia.
4	Admin	Voi hallinnoida kaikkea palvelussa, paitsi lisätä uusia käyttäjiä.
5	Superuser/ylläpito	Korkeimmat oikeudet. Hallinnointioikeudet myös käyttäjien lisäämiseen. Omaa myös oikeudet kaikkiin palvelintoimintoihin.
6	Palvelimen työntekijän prosessit	Prosessit joita sovellukseen kirjautunut käyttäjä voi sovelluksen logiikan mukaan suorittaa.
7	Tietokanta: Käyttäjän lukutoiminnot	Käyttäjä joka pääsee lukemaan tietokannan tietoja.
8	Tietokanta: Käyttäjän luku- ja kirjoitustoiminnot	Käyttäjä joka pääsee lukemaan ja kirjoittamaan tietokannasta/tietokantaan tietoja.

Uhkamalli (Application Threat Modeling)

Uhka kategorisointi (STRIDE Threat List)		
Tyyppi	**Esimerkki**	**Turvallisuuden hoitaminen**
Huijaaminen	Uhka jossa aikomuksena on päästä luvatta käsiksi käyttäjän tietoihin kuten käyttäjänimeen ja salasanaan.	Autentikointi
Peukalointi	Uhka jossa aikomuksena on päästä muokkaamaan pysyvää dataa tietokannoissa tai siirtyvää dataa ohjelman verkon välillä.	Koskemattomuus
Kieltäminen	Uhka joka on kohdistettu systeemiin joka ei pysty jäljittämään estettyjä operaatioita.	Ei kieltäminen
Tiedon julkituominen	Uhka jossa yritetään päästä käsiksi tiedostoihin järjestelmässä joihin henkilöllä ei ole lupaa. Myös datan lukeminen sen kulkemisen aikana.	Salassapito
Palvelun estäminen (DOS)	Lyhenne sanoista Denial of service. Eli uhka jossa aikomuksena on estää käyttäjiä käyttämästä palvelua. Esimerkiksi ruuhkauttamalla palvelin saaden sen tilapäisesti pois käytöstä.	Saatavuus
Oikeuksien muuttaminen	Uhka jossa aikomuksena on saada etuoikeutettu asema resursseihin, kuten luvattomaan tiedon hankintaan.	Oikeuttaminen

Sovelluksen uhkat ja niiden vastatoimet (Application Security Frame Threat & Countermeasures List)	
Uhkatyyppi	**Vastatoimet**
Autentikointi	1. Kirjautumistiedot ja autentikointiotokenit ovat salattuina, niin varastoinnin kuten myös tiedonsiirron aikana 2. Protokollat joita käytetään ottavat huomioon “brute force”-, “dictionary”- ja “replay”-hyökkäykset 3. Salasanojen luonnissa käytetään vahvan salasanan käytäntöjä 4. Salasanat tallennetaan salattuina suolan kanssa 5. Salasanojen uusiminen tapahtuu ylläpidon kautta 6. Käyttäjätunnuksia ei lukita sisäänkirjautumisen epäonnistuessa liian monta kertaa.
Oikeudet	1. Käyttäjille luodaan eri käyttötasot joilla mää-

Uhkamalli (Application Threat Modeling)

	ritellään käyttäjien oikeudet eri resursseihin sekä operaatioihin mitä sovelluksessa voidaan tehdä 2. Käyttäjille annetaan minimaaliset oikeudet, eli oikeudet vain niihin tehtäviin, mitä minkäkin roolin on tarkoitus pystyä tekemään.
Konfiguraatioiden/asetusten hallinta	1. Käyttäjien oikeuksissa käytetään pienimmän oikeuden periaatetta 2. Kaikki admin-tason tapahtumat tallennetaan lokeihin 3. Asetustiedostojen ja admin-sivujen käyttö on rajoitettu vain admin-tason käyttäjille ja palvelun ylläpidolle
Datan suojaaminen tietokannoissa ja datan siirron aikana	1. Käytetään standardoituja salausmetodeja. 2. Secretit joita käytetään salauksissa suojataan sekä siirron sekä varastoinnin aikana 3. Sisäänrakennettu turvallinen varasto pitää huolen salausavaimien tallessapidosta 4. Mikään kulkeva tieto ei siirry pelkkänä tekstinä.
Datan / Muuttujien validointi	1. Datan tyyppi, formaatti, pituus ja alue määritellään 2. Kaikki data joka lähetetään asiakaspäästä serverille, vahvistetaan 3. Muuttujia ei syötetä URL:in tavalla joka voisi haavoittaa ohjelmaa 4. Käyttäjän syötteet vahvistetaan että ne ovat halutunlaista muotoa 5. Ohjelman ulospäin lähtevä tieto(UI, REST) enkoodataan
Virheenkäsittely ja poikkeuksien hallinta	1. Kaikki poikkeustilanteet käsitellään jäsennetysti 2. Virheviesteissä ei näytetä ohjelman käyttäjälle mitään ylimääräistä tai arkaa tietoa
Käyttäjien ja istuntojen hallinta	1. Arkaa tietoa ei tallenneta evästeisiin 2. Autentikointi-evästeiden sisältö on salattua 3. Evästeistä ei tehdä pysyviä vaan ne asetetaan vanhentuviksi 4. Istunnot luodaan vastustamaan replay attack-hyökkäysmuotoa 5. HTTPS-protokolla on käytössä myös suojaamaan autentikointi-evästeitä 6. Istunnot vanhentuvat uloskirjautumisen jälkeen
Tiedon tarkastaminen ja tallentaminen	1. Arkaa tietoa ei tallenneta lokitiedostoihin