VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Petri Mäki-Jaakkola

WORDPRESS-LISÄOSAN KEHITYS

Prinetti-integraatio WooCommerce-verkkokauppaan

Liiketalous
2015

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

TIIVISTELMÄ

Tekijä	Petri Mäki-Jaakkola
Opinnäytetyön nimi	WordPress-lisäosan kehitys: Prinetti-integraatio WooCommerce-verkkokauppaan
Vuosi	2015
Kieli	suomi
Sivumäärä	58 + 2 liitettä
Ohjaaja	Raija Tuomaala

Tämä opinnäytetyö käsittelee WordPress-lisäosan kehittämistä ja tarkoituksena oli kehittää lisäosa WooCommerce-verkkokauppaan osoitekorttien tulostamista varten. WooCommerce on lisäosa, joka laajentaa WordPress-sisällönhallintajärjestelmää verkkokauppaominaisuuksilla. Osoitekorttien tulostus tapahtuu Postin Prinetti-järjestelmän integrointirajapintaa käyttäen. Työn toimeksiantaja oli Kalustemarkkinointi Oy.

Teoriaosuudessa käsitellään WordPress-sisällönhallintajärjestelmää, WooCommerce-verkkokauppaa sekä Prinetti-järjestelmää. Päätavoitteena oli tutkia toiminnallisuuden eli lisäosien kehittämisen tekniikoita ja parhaita käytänteitä sekä käytettävissä olevia rajapintoja. Teoriaosuudessa luodaan myös katsaus tekniikoihin, joita käytettiin kehitystyössä. Raporttiosuudessa kuvataan Kalustemarkkinointi Oy:n verkkokauppaan toteutetun Prinetti-integraation työvaiheet ja menetelmät käytännön tasolla.

Opinnäytetyön tuloksena syntyi WooCommerce Prinetti -niminen WordPress-lisäosa, joka on päivittäisessä käytössä toimeksiantajalla. Lisäosa nopeuttaa päivittäistä rutiinia verkkokauppatilausten käsittelyssä ja on välttämätön osa toimeksiantajan verkkokaupparatkaisua. Toimeksiantaja oli työhön erittäin tyytyväinen.

Avainsanat WordPress, lisäosa, ohjelmistokehitys, tietoturva

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittelyn koulutusohjelma

ABSTRACT

Author	Petri Mäki-Jaakkola
Title	WordPress Plugin Development: Prinetti Integration for WooCommerce
Year	2015
Language	Finnish
Pages	58 + 2 Appendices
Name of Supervisor	Raija Tuomaala

This thesis researched WordPress plugin development and the aim was to create a plugin for Kalustemarkkinointi Oy's WooCommerce web shop for printing shipment labels. WooCommerce is a plugin that adds ecommerce functionality to WordPress content management system. Printing shipment labels happens through Posti's Prinetti system. The assignment for this thesis was given by Kalustemarkkinointi Oy.

The theoretical background of this thesis consists of an overview of WordPress content management system, WooCommerce web shop and Prinetti system. The main objective of the thesis was to study WordPress plugin development: its best practices and available APIs. The framework of the thesis also builds up a brief overall of the techniques used in the practical section. The practical section goes through the development process of the plugin for Kalustemarkkinointi Oy's web shop, the steps and the methods used at a practical level.

As a result of this thesis, a plugin called WooCommerce Prinetti was developed and it is in the client's daily use. The plugin speeds up the daily routines of handling orders and is a mandatory part of the client's ecommerce system. The client was very satisfied with the results.

Keywords WordPress, plugin, development, security

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

LIITELUETTELO

KESKEISET KÄSITTEET JA LYHENTEET

AJAX Asynchronous JavaScript and XML, tekniikka, jonka avulla verkkosivun sisältöä voidaan muuttaa palvelimelta haetulla sisällöllä ilman sivun uudelleenlatausta.

CSRF Cross-site Request Forgery, hyökkäystapa, jossa tietoa yritetään välittää palvelimelle väärennetystä lähteestä.

FTP File Transfer Protocol, tiedonsiirtoprotokolla tiedoston siirtämiseen verkon välityksellä.

GPLv2 General Public License version 2, avoimen lähdekoodin lisenssi.

i18n Lyhenne sanasta internationalization, kansainvälistäminen.

Koukku Englannin kielen sanasta hook. Tapa, jolla liitetään funktioita osaksi ydinkoodia.

l10n Lyhenne sanasta localization, lokalisointi.

Lisäosa Englannin kielen sanasta plugin. Ohjelmiston toimintaa laajentava ohjelma tai kokoelma funktioita.

MySQL Avoimen lähdekoodin relaatiotietokanta.

Nonce Number Used Once, lomakkeiden, linkkien ja AJAX-pyyntöjen suojaamiseen käytettävä avain.

PHP Ohjelmointikieli, jota suoritetaan palvelimella. Palvelin palauttaa vastauksena PHP-skriptistä muodostetun verkkosivun.

REST Representational State Transfer, ohjelmistoarkkitehtuuri, jonka avulla verkkopalvelu voi välittää tietoa.

XML Rakenteisen tiedon esittämiseen tarkoitettu merkintäkieli.

1 JOHDANTO

WordPress on ilmainen, avoimeen lähdekoodiin perustuva, verkkosivujen, blogien ja verkkosovellusten julkaisuun tarkoitettu sovellus. WordPress alustaa käytetään yli 20 prosentissa suosituimmista internetsivuista, mikä tekee siitä maailman käytetyimmän sisällönhallintajärjestelmän. WordPressillä on laaja kehittäjäyhteisö ja suuri valikoima erilaisia lisäosia sekä teemoja, joiden avulla web-kehitystä osaamatonkin käyttäjä voi rakentaa pienellä vaivalla toimivan ja tyylikkään sivuston. WordPress.org-sivuston lisäosahakemisto sisältää yli 35 000 lisäosaa, jotka kaikki ovat käytettävissä ilmaiseksi saman GPLv2-lisenssin alaisesti, kuten WordPress itsekin.

WordPress-alustalla voidaan ylläpitää monenlaisia sivustoja ja verkkosovelluksia ja monesti sivuston elinkaari kulkeekin pienestä yhden sivun esittelysivustosta verkkokaupaksi tai laajaksi yrityssivustoksi, mikä tekeekin WordPressistä mielenkiintoisen sovelluksen. WordPressin valtava suosion vuoksi se on myös tärkeä osaamisalue web-kehittäjälle. WordPress noudattaa täysin omaa kehitysmallia niin verkkosivun ulkoasun luomisen kuin toiminnallisuuden laajentamisenkin suhteen, minkä takia voidaankin puhua WordPress-kehityksestä ja aihetta voidaan pitää omana osaamisalueenaan, vaikka WordPress onkin pohjimmiltaan rakennettu PHP-kielellä.

Tämän opinnäytetyön tarkoituksena on tutustua pintaa syvemmältä WordPress-kehitykseen, sen parhaisiin käytäntöihin ja tekniikkoihin. Toiminnallisessa osuudessa integroidaan Postin Prinetti-toiminnallisuus toimimaan WordPressille tarkoitetun WooCommerce-verkkokauppalisäosan kanssa. Ohjelmointityö suoritetaan siten, että se huomioi tietoturvallisuuden ja hyvän käytettävyyden. Työn toimeksiantaja on Kalustemarkkinointi Oy.

Teoriaosuudessa käsitellään WordPressin ja WooCommercen ominaisuuksia, historiaa ja niiden tarjoamia ohjelmointirajapintoja sekä koodausstandardeja ja WordPress-kehityksen parhaita käytänteitä. Lisäksi luodaan katsaus Prinetti-järjestelmän integraatiorajapintaan ja sen kanssa tarvittavaan XML-tekniikkaan.

WordPress lisäosa voidaan jakaa joko ilmaiseksi WordPress-lisäosakirjastossa tai myydä maksullisena lisäosana erilaisissa tähän erikoistuneissa verkkokaupoissa. Vaikka Prinetti-integraatio toteutetaankin toimeksiantajalle, tutustutaan teoriaosuudessa myös lisäosan jakeluun ja ylläpitoon.

Opinnäytetyön käytännön osuus käsittelee yksityiskohtaisesti ratkaisuja, joita käytettiin Prinetti-integraatio -lisäosan ohjelmoinnissa ja pohditaan, millainen toteutus lisäosalta vaaditaan, jotta se tehostaa verkkokaupan tilauksen käsittelyn prosessia mahdollisimman paljon.

WordPress kehityksellä tarkoitetaan tavallisesti joko teeman tai lisäosan kehitystyötä. Teeman kehittäminen keskittyy siihen, miltä WordPress-sisällönhallintajärjestelmälle luotu verkkosivusto tai sovellus näyttää, kun taas lisäosakehityksen ja tämän opinnäytetyön tavoitteena on luoda toiminnallisuus, joka laajentaa järjestelmän toimintaa.

Näistä lähtökohdista johdettu tutkimusongelma voidaan jakaa kolmeksi tutkimuskysymykseksi:

- Miten WordPressin toiminnallisuutta laajennetaan ilman, että muokataan ytimen lähdekoodia?
- Mitä menetelmiä on olemassa WooCommerce-verkkokaupan laajentamiseksi ja miten se poikkeaa tavallisesta WordPress-lisäosakehityksestä?
- Millä tavalla Postin Prinetti-järjestelmä voidaan saada toimimaan verkkokauppaohjelman sisällä?

Vaikka toteutettavassa lisäosassa ei tarvitakaan kaikkia WordPress-kehityksen osa-alueita, pyritään teoriaosuudessa tarkastelemaan useimpia tärkeimmistä osa-alueista. Tämän opinnäytetyön teoreettinen pohja perustuu alan kirjallisuuteen sekä ohjelmistojen ja tekniikoiden dokumentaatioihin.

2 WORDPRESS

WordPress on maailman suosituin julkaisujärjestelmä, sen päällä toimii yli 20 prosenttia internetin suosituimmista sivuista (Built With 2015a). Alun perin WordPress kehittyi erityisesti blogien ylläpitoa varten, mutta nykyisellään se on kehittynyt täysimääräiseksi sisällönhallintajärjestelmäksi. WordPress on avoimen lähdekoodin sovellus, jota käyttäjä ylläpitää omalla palvelimellaan. Sovelluksen käyttö on täysin ilmaista, ja käytettävissä on myös kattava valikoima erilaisia lisäosia ja teemoja.

Avoimen lähdekoodin sovelluksen lisäksi on myös olemassa WordPress.com-palvelu, jonka avulla kuka tahansa voi tehdä ilmaisen WordPress-sivuston, jota ylläpitää Automattic Inc. WordPress.com-palvelulla tehty sivusto on kuitenkin huomattavasti rajoitetumpi ympäristö, kuin itse ylläpidetty WordPress-asennus. WordPress.com-palveluun tehdylle sivustolle käyttäjä ei voi asentaa lisäosia tai muita teemoja, kuin mitä palvelussa on jo valmiina tarjolla. Tässä opinnäytetyössä WordPressillä tarkoitetaan aina itse ylläpidettyä WordPress-asennusta, ellei erikseen mainita kyseessä olevan WordPress.com-palvelu.

WordPress-sovellus on rakennettu PHP-ohjelmointikielellä ja se käyttää MySQL-tietokantaa. WordPress on laajennettavissa erilaisten lisäosien avulla ja se sisältää ohjelmointirajapinnan, joka mahdollistaa lisäosien liittämisen osaksi sovelluksen lähdekoodia. Käytännössä lähes kaikki, mitä voidaan saada aikaan PHP-ohjelmointikielellä ja MySQL-tietokannalla, voidaan liittää myös osaksi WordPress-toteutusta. (Messenlehner & Coleman 2014, 5.)

WordPress on julkaistu GPLv2 (General Public License version 2) mukaisesti. Tämä lisenssi antaa kenelle tahansa oikeuden käyttää, kopioida, muuttaa ja edelleen jakaa ohjelmaa ja sen lähdekoodia sillä edellytyksellä, että alkuperäisestä muutettu ohjelma julkaistaan myös samalla lisenssillä ja samojen ehtojen alaisena. (GNU 2014; WordPress Codex 2015a.)

WordPress sisältää suuren määrän erilaisia ominaisuuksia, jotka voidaan ryhmitellä ensisijaisen käyttäjän mukaan. Tässä opinnäytetyössä WordPress-käyttäjät on jaettu ryhmiin seuraavasti:

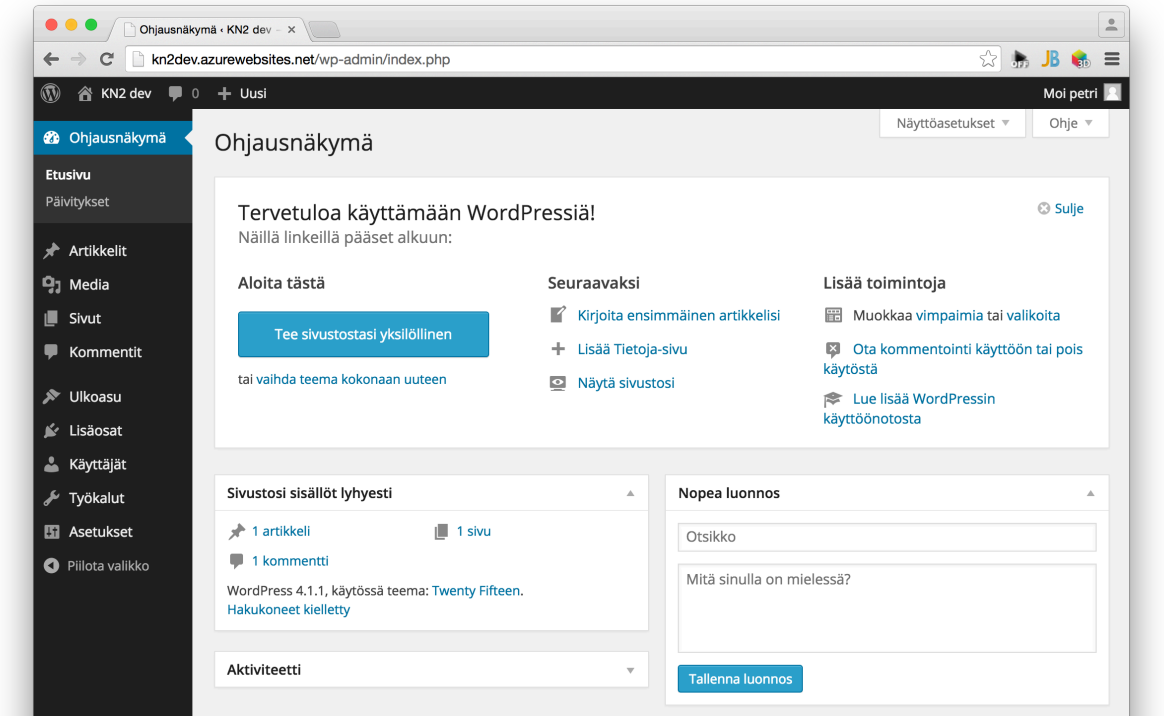
- Kehittäjä on ohjelmistokehittäjä, joka luo sivustoja, teemoja ja kehittää toiminnallisuutta.
- Ylläpitäjä on henkilö, joka vastaa sivuston materiaalin tuotannosta ja julkaisemisesta, käyttää hallintapaneelia ja sen ominaisuuksia, mutta ei koske lähdekoodiin.
- Loppukäyttäjä on sisällön kuluttaja, kuka tahansa, joka päätyy selaamaan sivustoa. Loppukäyttäjä ei yleensä ole edes tietoinen käyttävänsä WordPressiä.

2.1 Ominaisuudet ylläpitäjän näkökulmasta

WordPress oli alun perin artikkelien julkaisuun eli bloggaamiseen tarkoitettu ohjelmisto, mikä näkyy nykyäänkin laajoissa blogin kirjoitus- ja ylläpitoominaisuuksissa. WordPress koostuu ylläpitäjälle tarkoitetusta hallintapuolesta (Kuvio 1.) ja julkisesta, verkkosivustolla vierailijoille avoimesta puolesta. WordPress-sivusto tai ohjelma rakentuu teemasta ja sitä tukevista lisäosista. Tämän luvun tarkoitus on luoda katsaus niihin ominaisuuksiin, joita perusasennuksessa on valmiina ja joita ylläpitäjä voi käyttää julkaistessaan sisältöä sivustollaan. WordPressin perusasennus koostuu neljästä erilaisesta tyypistä sisältöä. Nämä sisältötyypit ovat: artikkelit, sivut, media-tiedostot ja kommentit.

2.1.1 Artikkelit

Artikkelit on tarkoitettu julkaisuiksi, jotka etenevät kronologisessa järjestyksessä ja joita voidaan luokitella kategorioittain tai avainsanojen mukaan. Artikkelit juontavat juurensa aikaan, jolloin WordPress oli luonteeltaan blogien julkaisualusta. Artikkelit-ominaisuutta voidaan käyttää blogin kirjoittamisen lisäksi esimerkiksi erilaisten ajankohtaisten uutisten julkaisuun.



Kuvio 1. Kuvakaappaus WordPressin hallintapaneelista.

2.1.2 Sivut

Sivuilla tarkoitetaan staattisia internetsivuja, joilla julkaistaan suhteellisen pysyvää tietoa. Sivut voidaan järjestää hierarkkisesti kuvaamaan eri sivujen sisällön suhdetta toisiinsa. Sivuilla voi olla erilaisia pohjatiedostoja (template), joiden avulla sivun asettelua tai ominaisuuksia voidaan vaihtaa. Nämä pohjatiedostot luodaan teeman sisälle.

2.1.3 Media

Mediatiedostot ovat yleensä kuvia ja ääni- tai videotiedostoja, mutta ne voivat olla mitä tahansa tiedostoja. Nämä tiedostot voidaan ladata palvelimelle artikkelien tai sivujen muokkaus-sivulta tai mediaselaimen kautta. Mediaselaimessa tiedostoja voidaan selata ja kuvatiedostoista nähdä esikatselukuvat. Kuvia voidaan jopa pienimuotoisesti muokata mediaselaimen avulla. Mediatiedostoista tallennetaan metatiedot tietokantaan, joten jos tiedostot halutaan näkyväksi mediaselaimessa, ei

riitä, että ne siirretään FTP:llä palvelimelle. WordPressin perusasennuksessa tiedostot sijaitsevat palvelimella `/wp-content/uploads/` -kansiossa, mutta tätä sijaintia voidaan muuttaa tarvittaessa, esimerkiksi käytettäessä pilvipalveluja, joissa mediatiedostot säilytetään eri paikassa kuin suoritettava lähdekoodi.

2.1.4 Kommentit

WordPress sisältää ominaisuudet, joiden avulla loppukäyttäjät voivat jättää kommentteja sisällöstä. Ylläpitäjä voi halutessaan määritellä, miten nämä kommentit käsitellään: julkaistaanko ne automaattisesti, vai vaaditaanko ylläpitäjän tarkistus ennen kommentin julkaisua.

2.2 Ominaisuudet kehittäjän näkökulmasta

WordPress kehitystyö ei pohjaudu mihinkään yleiseen ohjelmistokehityksen malliin, vaan se tapahtuu noudattaen sen omia käytänteitä. Tämän luvun tarkoitus on tarkastella WordPressin ominaisuuksia ohjelmistokehittäjän näkökulmasta yleisellä tasolla.

2.2.1 Teemat

WordPress-teema on tiedostokokonaisuus, joka sisältää kaiken sen, miltä WordPress-sivusto näyttää ulospäin. WordPress-kehityksessä pyritään erottamaan ulkoasu sisällöstä, jotta säilytetään joustava muokattavuus. Teeman vaihto voidaan tehdä suoraan ylläpitäjän hallintapaneelista, joten ylläpitäjältä ei edellytetä taitoja ohjelmistokehityksestä.

Teema muodostuu CSS-, PHP- ja JavaScript-tiedostoista sekä kuvista ja muista mediatiedostoista. Lisäksi teema saattaa sisältää myös erityisiä verkkosivuilla käytettäviä kirjasintyyppejä. Teeman muodostaman verkkosivuston HTML-koodi kirjoitetaan PHP-kielisiin sivupohjatiedostoihin. Pohjatiedostoja voidaan luoda useita erilaisia ja WordPress valitsee käytettävän pohjatiedoston erityisen logiikan tai sivukohtaisen määrityksen mukaan. Esimerkiksi front-page.php -tiedostoa käytetään muodostettaessa etusivua. Pohjatiedostot muodostavat hierarkian. Jos sisältötyyppiä varten ei ole erikseen määritettyä pohjatiedostoa, käytetään hierarkiassa

geneerisempää pohjaa. Yleispohja on index.php ja sitä käytetään aina, kun spesifimpää pohjatiedostoa ei ole käytettävissä.

2.2.2 Lisäosat

WordPressin ytimen muokkaamisen sijasta, sen toimintaa voidaan laajentaa lisäosan avulla. WordPress-lisäosa on sarja funktioita tai ohjelma kirjoitettuna PHP-ohjelmointikielellä, joka lisää ominaisuuksia tai palveluja ja voidaan liittää osaksi WordPress-toteutusta käyttäen WordPress-lisäosarajapinnan tarjoamia palveluita. Kun lisäosa pidetään erikseen WordPressin ytimestä, mahdollistetaan sekä WordPressin että lisäosan hyvä päivitettävyys. (WordPress Codex 2015a.)

2.2.3 Käyttäjien hallinta

WordPress sisältää valmiina toimintoja käyttäjien hallintaan. WordPressiin voidaan lisätä sekä ylläpitäjiä että loppukäyttäjiä tai voidaan luoda mitä tahansa muita käyttäjäryhmiä. Käyttäjien pääsyä voidaan rajoittaa sisällön mukaan. Käyttäjien hallinta on myös helposti laajennettavissa ja se antaa mahdollisuuden määrittää, mitkä toiminnot ovat mahdollisia tietylle käyttäjäryhmälle. (Messenlehner & Coleman 2014, 4–5.)

2.2.4 Tietokanta

WordPress tietokanta koostuu 11 taulusta, jotka on kuvattu liitteessä 1. Taulut sisältävät rivejä artikkeleista, sivuista, käyttäjistä ja kommenteista. Lisäksi on omat taulut asetuksille ja metatiedoille. Tietokannassa olevaan `wp_posts`-tauluun tallennetaan kaikki erilainen sisältö. Sisältö luokitellaan esimerkiksi sivuksi tai artikkeliksi `post_type`-ominaisuudella. Tällä tavoin voidaan luoda rajaton määrä erilaista luokiteltua tietoa olevaa sisältöä. Esimerkiksi tässä opinnäytetyössä käsiteltävän WooCommerce-verkkokauppalisäosan tuotteet tallennetaan `wp_posts`-tauluun `post_type`-ominaisuudella `wc_product`.

2.3 Tietoturva

WordPress on käytössä miljoonilla sivustoilla, minkä vuoksi se on jatkuvien tietoturvaloukkauksien kohde. Tämä on kuitenkin myös etu WordPress-yhteisölle, koska WordPressin tietoturva on näinollen koeteltu ja jatkuvan testauksen alainen. Kehittäjät ovat nopeita reagoimaan haavoittuvuuksiin ja julkaisevat päivityksiä niihin. Päivitysten asennus tapahtuu automaattisesti hallintapaneelin kautta, eikä käyttäjältä edellytetä erikoisosaamista, joten tietoturvapäivitysten asennus tapahtuu vaivatta. (Messenlehner & Coleman 2014, 6.)

Tietoturvapäivitysten luomaan turvallisuuden tunteeseen ei kuitenkaan tule tuudittautua, vaan tulee olettaa, että ennemmin tai myöhemmin sivusto joutuu tietoturvaloukkauksen kohteeksi. Messenlehnerin ja Colemanin mukaan yksi yleisimmistä hyökkäystyypeistä on ”brute force attack” eli hyökkäys, jonka teho perustuu raakaan voimaan. Hyökkääjä yrittää kokeilla valtavan määrän satunnaisia käyttäjätunnus-salasanapareja tekemällä jopa tuhansia kyselyjä sekunnissa. Vaikka hyökkääjä ei arvaisikaan oikeaa tunnusyhdistelmää, voi raskas kuorma aiheuttaa palvelimen kaatumisen, tätä kutsutaan palvelunestohyökkäykseksi. (Messenlehner & Coleman 2014, 215.)

Tietoturva ei ole pelkästään yhden kerran tehtävä projekti, vaan se vaati jatkuvaa ylläpitoa ja seurantaa. Niin kuin jo todettu, päivitysten asennus sekä itse WordPressiin, että lisäosiin ja teemoihin parantaa tietoturvaa. Kriittiset tietoturvapäivitykset asentuvat nykyään jopa automaattisesti, ilman että ylläpitäjän tarvitsee kirjautua hallintapaneeliin. Sivuston ylläpidossa tulee kiinnittää huomio myös yksinkertaisilta tuntuviin asioihin: käyttäjätunnuksena ei tule missään nimessä käyttää admin- tai muuta ylläpitäjään liittyvää yleistä sanaa, koska haitalliset botit, eli robottiohjelmat, yrittävät yleensä juuri näiden käyttäjätunnusten kautta tunkeutua järjestelmään sisälle. Samalla tavalla salasanan valintaan tulee kiinnittää huomiota luomalla uniikki merkkijono, joka ei ole millään tavalla pääteltävissä tai arvattavissa. Luvussa 3.11 esitellään konkreettisia tapoja tietoturvan parantamiseksi. (Messenlehner & Coleman 2014, 215–217.)

3 WORDPRESS-LISÄOSAN KEHITYS

WordPress kehitystyöllä voidaan tarkoittaa joko teeman tai lisäosan kehittämistä. Tämä opinnäytetyö käsittelee erityisesti WordPressin toiminnallisuuden laajentamista eli lisäosan kehittämistä. Lisäosien kehitykseen WordPress tarjoaa ohjelmointirajapinnan, jonka avulla lisäosa liitetään toimimaan WordPress-ytimen rinnalla. Tämä rajapinta on varsin laaja, minkä vuoksi se on jaettu useisiin pienempiin rajapintoihin. Tässä luvussa tutustutaan rajapintoihin ja WordPress-kehitystyön tärkeimpiin tekniikoihin.

3.1 Plugin API

Plugin API on rajapinta, jonka avulla uusi toiminnallisuus liitetään toimimaan WordPress-ytimen rinnalle. Toimintojen liittäminen perustuu koukkuihin, joita on kahdensuuntaisia: koukut joita on sijoitettu WordPress ytimeen, ja koukut, joilla näihin tartutaan lisäosasta tai teemasta. Koukkuja on sijoitettu suuri määrä WordPress-ytimen lähdekoodin ja suoritettaessa WordPressiä, tarkistetaan onko koukkuihin rekisteröity funktioita. Jos koukkuun on rekisteröity funktio, suoritetaan se ennen kuin pääohjelman suoritusta jatketaan eteenpäin. Tämä on yksi tärkeimmistä tavoista, joilla WordPressin toimintaa laajennetaan. Koukkuihin voidaan kiinnittyä joko toiminta- (actions) tai suodatinkoukuilla (filters). (WordPress Codex 2015b.)

3.1.1 Actions eli toimintakoukut

Toimintakoukut suoritetaan yleensä jonkun tapahtuman seurauksena, kuten artikkelin julkaisun, teeman vaihtamisen tai ennen tai jälkeen ylläpitäjän hallintapaneelin näyttämistä. Toimintakoukku määritellään lisäosassa tai teemassa ja kiinnitetään tiettyyn koukkuun WordPressin ytimen koodissa, toisessa lisäosassa tai teemassa. Toimintakoukuilla suoritetaan yleensä muutoksia tietokantaan, lähetetään sähköpostia tai muulla tavalla muokataan käyttäjälle näytettävää näkymää.

Toimintakoukku muodostetaan luomalla ensin funktio lisäosatiedostoon tai teeman functions.php-tiedostoon, joka on tarkoitus suorittaa, kun määritelty WordP-

ress tapahtuma (event) tapahtuu. Tämän jälkeen vielä kiinnitetään luotu funktio haluttuun tapahtumaan käyttäen `add_action()` -funktiota, jonka ensimmäiseksi parametriksi annetaan koukku, ja toiseksi suoritettava funktio. (WordPress Codex 2015b.)

Kolmanneksi parametriksi `add_action()` -funktiolle voidaan antaa suoritusjärjestystä kuvaava luku ja neljänteen parametriin voidaan määritellä montako parametria suoritettavalle funktiolle voidaan antaa. Kaikki toimintakoukun funktion muodostama toiminta suoritetaan tai sen tuottama syöte ilmestyy näkymään siinä kohtaa, jossa toimintakoukkua kutsutaan. (WordPress Codex 2015c; WordPress Codex 2015b.)

3.1.2 Filters eli suodatinkoukut

Nimensä mukaisesti suodatinkoukulla liitetään WordPress-ytimeen funktio, jonka tarkoitus on ottaa parametrina muokkaamatonta dataa, ja suodattaa tai muulla tavoin muokata dataa ja palauttaa se. Suodatinkoukku muodostetaan toimintokoukuista tutulla tavalla. Ensin luodaan funktio, joka on tarkoitus suorittaa määriteltävässä koukussa, tämän jälkeen funktio liitetään koukkuun `add_filter('koukun_nimi', 'suodatinfunktio')` -funktiota käyttäen. Myös `add_filter()` -funktio hyväksyy valinnaisina kolmanneksi parametriksi tärkeysjärjestystä kuvaavan kokonaisluvun ja neljänneksi parametriksi suoritettavan funktion parametrien määrää kuvaavan luvun, mikä oletusarvoisesti on yksi. Suodatinkoukku voidaan halutessa myös poistaa `remove_filter()` -funktiolla, jos esimerkiksi halutaan muokata lisäosan toimintaa teeman functions.php-tiedostosta käsin. (WordPress Codex 2015d.)

3.2 Options API

Options API, eli ohjelmointirajapinta, on yksinkertainen ja standardisoitu tapa tallettaa asetuksiin liittyvää tietoa tietokantaan. Rajapinnan avulla voidaan luoda, lukea, muokata ja poistaa asetustietoja, joita säilytetään wp_options-taulussa. Asetustietue voidaan luoda `add_option()` tai `update_option()` -funktioiden avulla. Ero näissä funktioissa on siinä, että `add_option()` -funktio lisää tietueen vain, jos

sitä ei vielä ole olemassa, kun taas `update_option()` -funktio joko lisää tietueen, jos sitä ei ole olemassa, tai päivittää sen arvon, jos tietue on jo olemassa. Näin ollen voidaan yleensä aina käyttää `update_option()` -funktiota, ellei erityisesti haluta, että olemassa olevaa tietuetta ei ylikirjoiteta. (WordPress Codex 2015e.)

Muut tärkeät funktiot Options-rajapinnassa ovat `get_option()`, jonka avulla luetaan tietue, sekä `delete_option()`, jonka avulla tietue poistetaan `wp_options()` -taulusta esimerkiksi lisäosan päivityksen tai poistamisen yhteydessä. (WordPress Codex 2015e.)

3.3 Settings API

Kun tehdään toiminnallisuutta, jonka loppukäyttäjällä ei ole ohjelmistokehitysosaamista, tarvitaan keinoja, joiden avulla käyttäjä voi muokata ohjelman toimintaa omiin tarpeisiinsa tai edellytyksiinsä sopivaksi. WordPress tarjoaa tähän Settings-rajapinnan, jonka avulla voidaan luoda standardin mukainen asetukset-sivu osaksi ylläpitäjän hallintapaneelia. (Messenlehner & Coleman 2014, 193–194.)

Messenlehner & Coleman toteavat, että on turhaa tehdä asetukset-sivua, jos lisäosalla ei ole kehittäjän lisäksi muita käyttäjiä, tai muut käyttäjät ovat myös todennäköisesti kehittäjiä. Tässä tapauksessa paras tapa lisätä asetuksia on asettaa globaali muuttuja, joka sisältää asetustiedoista muodostetun taulukon, lisäosatiedoston alkuun. Globaalia muuttujaa voidaan käyttää myös kehitystyön aikaiseen testaukseen, jolloin vältytään turhilta muutoksilta tietokantaan. Kuviossa 2 esitetään esimerkki globaalin muuttujan käytöstä. (Messenlehner & Coleman 2014, 193–194.)

```php
<?php

global $woocommerce_prinetti_settings;
$woocommerce_prinetti_settings = array(
    'sender' => 'Kalustemarkkinointi Oy',
    'account' => '123456'
);

```

Kuvio 2. Esimerkki globaalin muuttujan luonnista.

Globaali muuttuja luodaan lisäosan päätiedostoon, jolloin se on aina käytettävissä. Globaali muuttuja määritetään avainsanalla global ja sitä voidaan käyttää joko tavallisena muuttujana tai taulukkona avain-arvopareja, niin kuin kuvion 2 esimerkissä esitetty.

3.4 Asennus ja poisto

Lisäosa tulee näkyviin ylläpitäjän hallintapaneeliin, kun lisäosan päätiedosto sisältää otsaketiedot, jotka määrittävät mm. lisäosan nimen. Tämän jälkeen lisäosa tulee vielä asentaa eli ottaa käyttöön, ennen kuin sitä voidaan käyttää. Tämä tapahtuu hallintapaneelista. Käyttöönoton yhteydessä lisäosa rekisteröidään tietokantaan käyttöönotetuksi. Jos lisäosan toiminnassa ilmenee ongelma, joka estää WordPressin käytön, helpoin tapa korjata ongelma, on poistaa tai siirtää lisäosan tiedostot toiseen sijaintiin hetkellisesti.

Lisäosan käyttöönoton yhteydessä voidaan käyttää `register_activation_hook()` -funktiota suorittaakseen käyttöönottoon liittyviä funktioita, kuten tietokantaan tehtävät muutokset. Funktiolle annetaan kaksi parametria, joista ensimmäinen on polku lisäosan päätiedostoon. Tässä kohtaa voidaan käyttää PHP:n ns. magic vakiota (magic constant) ”`__FILE__`”, joka palauttaa täydellisen polun ja tiedostonimen siihen tiedostoon, josta sitä kutsutaan (PHP Group 2015d). Toiseksi parametriksi annetaan suoritettava funktio, jonka sisällön kehittäjä siis määrittää itse.

Vastaavasti kun tehdään muutoksia tietokantaan, on parhaiden käytänteiden mukaista tehdä mahdolliseksi muutosten poisto. Tähän voitaisiin käyttää `register_activation_hook()`-funktion käänteistä vastaparia, `register_deactivation_hook()`-funktiota, jonka määrittämä funktio suoritetaan kun lisäosa poistetaan käytöstä. Parempi tapa kuitenkin on käyttää `register_uninstall_hook()`-funktiota, joka suoritetaan, kun käyttäjä painaa ”Poista asennus” -näppäintä hallintapaneelissa. Tällöin mahdollistetaan lisäosan väliaikainen käytöstä poisto ilman, että sen tallentamat tiedot menetetään. (WordPress Codex 2015g.)

3.5 Tietokannan rakenne ja käsittely

Tietokannan käsittelyä varten WordPress sisältää erityisen Wpdb-nimisen luokan. Tätä luokkaa käytetään suoraan tietokannan käsittelyyn. WordPress sisältää myös erityisiä tietokannan käsittelyä helpottavia funktioita, mutta toisinaan on tarpeellista käyttää tehokkuuden vuoksi $wpdb-luokan tarjoamia metodeja tiedon hakemiseen, päivittämiseen, tallentamiseen ja poistamiseen. (Messenlehner & Coleman 2014, 68.)

WordPress sisältää myös dbDelta-nimisen funktion, joka helpottaa tietokannan käsittelyä. dbDelta-funktio tutkii ensin olemassa olevan taulun rakenteen, vertaa sitä uuteen rakenteeseen ja joko muokkaa olemassa olevaa taulua tai luo uuden taulun. Tämä helpottaa lisäosakehitystä, koska uusien rivien lisääminen tauluun tarpeen niin vaatiessa on helppoa eikä hävitä jo kertynyttä dataa. WordPress Codexissa kuitenkin todetaan dbDelta-funktion olevan melko tarkka syntaksistaan, joten seuraavia toimintatapoja tulee noudattaa:

- Jokainen kenttä tulee laittaa omalle rivilleen SQL-komennossa.
- Sanojen PRIMARY KEY ja sen määrityksen väliin tulee laittaa kaksi välilyöntiä.
- Sanaa KEY tulee käyttää ennemmin kuin sen synonyymia INDEX ja vähintään yksi KEY tulee esiintyä komennossa.
- Heitto- tai lainausmerkkejä ei tule käyttää kenttien nimien ympärillä.

- Kenttien nimet tulee kirjoittaa kokonaan pienillä kirjaimilla.
- SQL komennot tulee kirjoittaa kokonaan isoilla kirjaimilla. (WordPress Codex 2015f.)

Tämä dbDelta-funktio sijaitsee wp-admin/includes/upgrade.php-tiedostossa, joka tulee sisällyttää mukaan, kuvion 3 mukaisesti, luotaessa tai päivitettäessä omaa taulua WordPress-tietokantaan, koska sitä ei oletusarvoisesti ladata. (WordPress Codex f.)

```php
<?php

global $wpdb;

$charset_collate = $wpdb->get_charset_collate();
$table_name = $wpdb->prefix . 'woocommerce_prinetti';

$sql = "CREATE TABLE $table_name (
  id mediumint(9) NOT NULL AUTO_INCREMENT,
  order_id mediumint(9),
  created datetime DEFAULT NOW() NOT NULL,
  trackingcode varchar(50) NOT NULL,
  sent datetime,
  UNIQUE KEY id (id)
) $charset_collate;";

require_once(ABSPATH . 'wp-admin/includes/upgrade.php');
dbDelta($sql);

```

Kuvio 3. Esimerkki dbDelta-funktion käytöstä taulun luomiseksi WordPress-tietokantaan.

3.6 WordPress silmukka

”The WordPress Loop” eli WordPress silmukka, on tapa, jolla WordPress esittää datan. Silmukka sijoitetaan teeman sivupohjiin. Sivupohjajärjestelmä (templating engine) valitsee käytettävän pohjatiedoston sen mukaan, minkä sivun kävijä pyytää. Kehittäjän tehtäväksi jää sijoittaa silmukka ja sen sisältämä logiikka pohjatiedostoihin haluamallaan tavalla. WordPress tekee kyselyt tietokantaan ja hakee da-

tan, mikä näytetään sivulla vierailijalla, ja iteroi kaikkien tietueiden läpi kuvion 4 osoittamalla tavalla.

```php
<?php
if ( have_posts() ) {
    while ( have_posts() ) {
        the_post();

        the_title( '<h2>', '</h2>' );
        the_content();
    }
} else {
    // Täällä voidaan näyttää viesti, esimerkiksi
    // virhetilanteesta kun sisältöä ei löydy
}
```

Kuvio 4. Peruskäytäntö WordPress-sisällön esittämiseksi.

3.7 WordPressin omat globaalit muuttujat

Luvussa 3.3. tutustuttiin omien globaalien muuttujien luontiin kehitettäessä lisäosaa. WordPress sisältää myös sisäänrakennettuna muutamia globaaleja muuttujia, jotka helpottavat kehitystyötä. Käytetyimpiä näistä ovat $post- ja $authordata-muuttujat. $post-muuttuja sisältää kaiken datan, jota voi olla esimerkiksi blogikirjoitus, yksittäinen sivu tai mm. WooCommerce-tuote, jotka tallennetaan juuri tällaisena erityisenä WordPress-tietotyyppinä. $authordata-muuttuja taas sisältää käsiteltävän $post-tietotyypin luoneen käyttäjän tiedot. (Messenlehner & Coleman 2014, 67-68.)

3.8 Kansainvälistäminen ja lokalisointi

Kansainvälistäminen sana tulee englannin kielen sanasta internationalization ja sen lyhenne i18n muodostuu sanan alku- ja loppukirjaimista sekä näiden välissä olevasta luvusta 18, joka kuvaa väliin jäävien kirjaimien lukumäärää. Lokalisointi sanasta on tehty samalla periaatteella myös oma l10n lyhenne. Näitä termejä ja lyhenteitä käytetään yleisesti WordPress-kehityksessä, joten on hyvä tietää, mistä

lyhenteet muodostuvat. Suomen kieli on suhteellisesta pienuudestaan johtuen harvoin valmiiksi käännettynä WordPress teemoissa tai lisäosissa, joten kehittäjä joutuu usein tekemisiin käännöstyön kanssa. Tämä luku käsittelee yksityiskohtaisesti käännösprosessia.

Kansainvälistäminen ja lokalisointi -termejä käytetään helposti ristiin. Ero termien välillä tulee lähinnä siitä, kuka suorittaa termin tarkoittamaa tehtävää. Kansainvälistämisestä puhutaan, kun ohjelmiston kehittäjä tekee toimia, jotka mahdollistavat ohjelmiston kääntämisen muille kielille, kun taas lokalisoinnilla tarkoitetaan esimerkiksi lisäosaa tai itse WordPressiä omaan tarkoitukseensa muokkaavan ylläpitäjän tekemiä käännös- tai muita kulttuurillisiin eroihin liittyviä muutostöitä. (WordPress.org 2015a.)

Kansainvälistäminen ja lokalisointi ovat sanoja, joita käytetään kuvaamaan niitä tekoja, joita vaaditaan, jotta ohjelmistoja voidaan käyttää eri kielillä ja paikallisilla tavoilla. Kulttuurillisia eroja ovat esimerkiksi: käytetäänkö pilkkua vai pistettä tuhat- vai desimaalierottimena ja kummalle puolelle lukua valuuttamerkki merkitään. Myös yksikön ja monikon muodostaminen poikkeaa kielten välillä toisistaan, jolloin tarvitaan tapa muodostaa taivutus eri tavoin. (WordPress.org 2015a.)

Kääntäminen on kaksivaiheinen prosessi. Ensimmäisessä vaiheessa ohjelmistokehittäjä tarjoaa kehittämässään ohjelmassa mekanismin, jonka avulla tekstin vaihtaminen on mahdollista. Toinen vaihe sisältää varsinaisen käännöstyön, jossa käytetään kehittäjän tarjoamaa menetelmää ja käännetään tekstit halutulle kielelle ja paikalliseen ympäristöön sopivaksi. (WordPress.org 2015a.)

WordPress on käännetty useille kielille ja käännökset ovat ilmaiseksi käytettävissä. Versiosta 3.7 lähtien halutun kielen on voinut valita jo asennusvaiheessa ja kielitiedostojen päivitys tapahtuu automaattisesti (Mullenweg 2013). Käännöstyökaluja tarvitaan, kun halutaan käyttää kolmannen osapuolen lisäosaa tai teemaa, josta ei vielä ole olemassa käännöstä. Kääntämiseen liittyvät asiat tulee myös huomioida lisäosia tai teemoja kehitettäessä, koska jos kehittäjä ei tarjoa mekanismia kääntämisen suorittamiseksi, ei kääntäminen onnistu ilman lähdekoodin muokkaamista.

3.9 Kieliversioiden luominen gettextin avulla

WordPress käännös perustuu GNU:n gettext-kirjastoon. Käännösprosessissa määritellään ensin PHP-lähdekoodissa merkkijonot `_("String")` -merkintätapaa käyttäen. Vaikka tämä on PHP:ssa yleinen tapa, on WordPressissä määritelty oma tapa merkitä käännettävät merkkijonot `__()` -funktiolla (ylimääräinen alaviiva). Tämän jälkeen lähdekoodille ajetaan ohjelma, joka etsii käännettävät merkkijonot ja muodostaa niistä .pot-tiedoston. Pot-tiedosto sisältää kaikki alkuperäisen kielen merkkijonot. Tämän jälkeen pot-tiedostosta muodostetaan po-tiedosto, johon tehdään käännökset halutulle kielelle. Tämän jälkeen po-tiedosto käännetään vielä mo-tiedostoksi, jonka jälkeen käännös on valmis käytettäväksi. Tämän melko monimutkaisen prosessin suorittamiseen on olemassa eri työkaluja, esimerkiksi PoEdit, joka helpottaa käännöstyötä visuaalisen käyttöliittymän ansiosta. (WordPress.org 2015b.)

WordPress-lisäosan käännöstä varten merkitään ns. text domain lisäosan otsaketietoihin. Tämän nimeämiseksi käytetään käytäntöä, jossa lisäosan nimi kirjoitetaan pienillä kirjaimilla ja välilyöntien tilalle merkitään väliviiva kuvion 5 mukaisesti. ”Domain Path” on polku mo-tiedostojen sijaintiin ja se määritellään, mikäli kielitiedostot sijoitetaan muualle kuin juurihakemistoon. (WordPress.org 2015b.)

```
1  /*
2   * Plugin Name: WooCommerce Prinetti
3   * Author:      Petri Mäki-Jaakkola
4   * Text Domain: woocommerce-prinetti
5   * Domain Path: /languages
6   */
7
```

Kuvio 5. Esimerkki lisäosan otsaketiedoista, jotka tarvitaan käännöstä varten.

Kun käännettäviä merkkijonoja muodostetaan lisäosan lähdekoodin, on hyvä huomata, että `__('String', 'my-plugin')` -metodi ei tulosta mitään, vaan toimii muuttujan tapaan. Tulostus voidaan tehdä käyttäen PHP:n echo-funktiota

”`echo __('This is just an example', 'my-plugin')`” tai lyhennettyä versiota `_e('This is just an example', 'my-plugin')`, jossa jälkimmäinen alaviivoista korvataan e-kirjaimella. HTML-koodin sekaan kirjoitettaessa funktiot kääritään vielä PHP-tagin sisään seuraavasti: `<?php _e('This is just an example', 'my-plugin'); ?>`. (WordPress.org 2015b.)

Monikkojen käsittelyyn on myös oma tapansa. Tätä tarvitaan esimerkiksi tilanteessa, jossa halutaan kirjoittaa ostoskorissa olevan joko ”1 tuote” tai ”2 tuotetta”. Tämän ongelman ratkaisemiseksi käytetään `_n()` -funktiota, joka hyväksyy neljä parametria:

- yksikkömuoto
- monikkomuoto
- määrä, eli numero joka ilmaisee käytetäänkö yksikköä vai monikkoa
- text domain, eli lisäosan yksilöllinen tunniste.

Paluuarvona `_n()` -funktio palauttaa oikean käännetyn muodon. (WordPress.org 2015b.)

Sanat, joiden kirjoitusasu on sama, mutta tarkoitus on eri, tulee huomioida käännösvaiheessa. Esimerkiksi ”Click here to post your comment” ja ”Edit this post” -lauseissa post-sanan merkitys on eri. Ensimmäisessä tapauksessa post-sanan käännös olisi ”lähettää” ja jälkimmäisessä tapauksessa ”artikkeli”. Tällaisessa tapauksessa käytetään `_x()` -funktiota, joka ottaa parametriksi myös kontekstin. Parametrit ovat:

- käännettävä sana
- konteksti, esim. verbi tai substantiivi
- text-domain, eli lisäosan yksilöllinen tunniste.

Myös `_x()` -funktio palauttaa pelkän arvon eikä tulosta mitään, ja myös tämän funktion paluuarvon tulostamiseksi on olemassa `_ex()` -funktio, joka muuten toimii täysin samalla tavalla. (WordPress.org 2015b.)

3.10 WP Mail

Vaikka PHP-sisältääkin valmiit funktiot sähköpostin lähettämiseen, on WordPressiin toteutettu omat funktiot, jotka tekevät sähköpostin lähettämisestä ja muotoilusta helpompaa. WordPressin sähköpostinlähetysominaisuuksia käytetään esimerkiksi: kun halutaan toteuttaa yhteydenottolomake, jonka loppukäyttäjä voi täyttää sivustolla, ja joka lähetetään ylläpitäjälle. Myös WooCommerce-verkkokauppalisäosa lähettää paljon sähköpostia, kuten tilaustiedot ylläpitäjälle ja tilausvahvistukset ja tilauksen etenemistiedot asiakkaalle.

WordPress lähettää sähköpostia `wp_mail()` -funktion avulla, mikä korvaa PHP-kieleen sisäänrakennetun `mail()` -funktion. Tätä funktiota käytetään seuraavasti: `wp_mail($kenelle, $aihe, $viesti, $otsake, $liitteet)`. (Messenlehner & Coleman 2014, 207.)

3.11 WP Cron

Cron-tehtävä on skripti, joka suoritetaan palvelimella asetettujen ajanjaksojen välein tai ennalta asetettuna aikana. WP-Cron toteuttaa tämän toiminnallisuuden WordPress-sivustolle. Cron-tehtävä voidaan suorittaa esimerkiksi joka minuutti, joka tunti, joka päivä tai vaikka tiettyinä viikonpäivinä. Tavallisia tehtäviä ovat esimerkiksi sähköpostin lähettäminen tai tietojen synkronointi kolmannen osapuolen rajapinnan kanssa. (Messenlehner & Coleman 2014, 202.)

WordPressin WP-Cron toteutus ei kuitenkaan täysin vastaa Unix-järjestelmistä tuttua Cron-toteutusta. WordPressin Cron-tehtävät laukaistaan sivulatauksien yhteydessä, mikä johtaa siihen, että jos sivustolla ei ole säännöllistä liikennettä, tai sivut tarjotaan staattisesta välimuistista, ei Cron-tehtävät laukea ajallaan, vaan mahdollisesti huomattavasti myöhässä. Vaikka tämä onkin yleensä täysin siedettävää, joissain tapauksissa tarvitaan kuitenkin luotettavuutta. Tällöin on määriteltävä config.php-tiedostoon `define('DISABLE_WP_CRON', true);` ja manuaalisesti tehtävä kysely wp-cron.php-tiedostoon. Tämä vaatii kuitenkin pääsyn tekemään muutoksia palvelimelle, joten yleiseen käyttöön tulevalle lisäosalle tämä ei ole mahdollinen toteutus. (Messenlehner & Coleman 2014, 205.)

3.12 Julkaisu ja ylläpito

Lisäosa voidaan julkaista ja ylläpitää WordPress lisäosakirjastossa. Tällöin on käytettävä GPLv2 lisenssiä, jonka alaisena lisäosa on maksuton ja kenen tahansa on mahdollista tehdä lisäosasta oma versionsa ja jakaa sitä saman lisenssin alaisena. WordPress lisäosakirjastossa ylläpidettäviin lisäosiin on liitettävä readme.txt-tiedosto, johon kirjoitetaan erityisen mallin mukaisesti tietoja lisäosasta, kuten asennusohjeet, päivitykseen liittyvät huomautukset sekä muutosloki. (WordPress Codex 2015a.)

Jotta lisäosa voidaan jakaa WordPress lisäosakirjastossa, tulee se lähettää ensin tarkastettavaksi. Tarkastuksen jälkeen kehittäjä saa käyttöönsä subversion repositoryn, joka on eräänlainen versionhallintajärjestelmä, jonka avulla kehittäjä voi tehdä muutoksia lisäosaan ja julkaista uudet versiot, jotka tulevat lisäosan käyttäjälle näkyviin ylläpitäjän hallintapaneelin päivitykset-osioon.

3.13 Tietoturvan vahvistaminen

Tietoturvan vahvistamiseksi on olemassa muutamia tekniikoita, joista kehittäjän on hyvä olla tietoinen. WordPress sallii oletusarvoisesti ylläpitäjien muokata minkä tahansa lisäosan tai teeman lähdekoodia suoraan hallintapaneelin kautta. Jos hakkeri onnistuu kirjautumaan sisään, saa hän siis suoran pääsyn kirjoittamaan vahingollista koodia, jolla voi jopa vapaasti käsitellä tietokantaa. Tämän ominaisuuden voi kuitenkin estää määrittelemällä wp-config.php-tiedostoon seuraavasti: `<?php define( 'DISALLOW_FILE_EDIT', true ); ?>`. (Messenlehner & Coleman 2014, 218.)

WordPressin perusasennuksessa tietokannan taulujen etuliitteeksi on valmiiksi asetettu ”wp_”. Tämä voidaan kuitenkin asennusvaiheessa määritellä miksi tahansa ja muuttaminen on vaivaansa nähden erittäin merkittävä tietoturvan parannus. Tämän jälkeen luvattoman tunkeutujan on paljon vaikeampi keksiä tietokannan taulujen nimiä. (Messenlehner & Coleman 2014, 218-219.)

WordPressin konfiguraatiotiedosto wp-config.php sisältää useita arkaluonteisia tietoja sivustosta, kuten tietokannan sijainnin, käyttäjänimen ja salasanan, sekä sivuston autentikointiavaimet. Näitä säilytetään PHP-muuttujissa, joten periaatteessa niihin ei pitäisi päästä käsiksi selaimen kautta. Tietoturva-asioissa kuitenkaan mikään ei ole täysin varmaa, joten wp-config.php-tiedoston voi salata paremmin siirtämällä sen yhden askeleen ylemmäs tiedostorakenteessa, mikä useimmissa tapauksissa on ei-julkinen kansio palvelimella. WordPress etsii automaattisesti tätä tiedostoa yhden askeleen ylempää, ellei se löydy asennuksen juurihakemistosta. (Messenlehner & Coleman 2014, 219-220.)

Muita tapoja tietoturvan tiukentamiseksi on piilottaa WordPress versio, estää kirjautuminen wp-login.php-sivun kautta ja sallia kirjautuminen /wp-admin -sivun kautta pelkästään tietyistä IP-osoitteista .htaccess-tiedoston avulla. (Messenlehner & Coleman 2014, 220-222.)

CSRF on hyökkäystapa, jossa lomakkeelle yritetään välittää dataa väärästä lähteestä. Jos palvelinpuolen koodi käsittelee yksinkertaisesti `$_POST`-muuttujan sisältämän datan ilman sen todentamista, data saattaa olla peräisin miltä tahansa lomakkeelta, ei pelkästään omalta sivulta. Tämän ongelman ratkaisemiseksi käytetään noncea, joka on lyhenne sanoista ”number used once”. Nonce-merkkijono generoidaan joka kerta sivunlatauksen yhteydessä ja lisätään lomakkeelle piilotettuna kenttänä. Kun lomake lähetetään takaisin palvelimelle, tutkitaan nonce ja varmistetaan sen voimassaolo.

4 WOOCOMMERCE

WooCommerce on suosittu lisäosa, joka lisää verkkokauppatoiminnot WordPress-sisällönhallintajärjestelmään. WooCommerce on avoimen lähdekoodin sovellus, jota on kehittänyt WooThemes. WooCommerce syntyi, kun WooThemes palkkasi alun perin Jigoshop-verkkokauppaa kehittäneet Mike Jolleyn ja James Kosterin ja se on kehitetty eriyttämällä uusi kehityslinja avoimen lähdekoodin Jigoshop-verkkokaupasta, joten ne muistuttavat jonkin verran toisiaan. Ensimmäinen WooCommerce-versio julkaistiin vuonna 2011.

WooCommerce on muutamassa vuodessa kasvanut suosituimmaksi WordPressin verkkokaupparatkaisuksi lähes 700 000 käyttäjällään (Built With 2015b). Syitä suosioon lienevät helppo käyttöönotto, moderni ja selkeä käyttöliittymä sekä WooThemesin itse kehittämä laaja valikoima erityisesti WooCommerce-verkkokauppaan tarkoitettuja teemoja ja lisäosia.

WooCommerce sisältää monia varsin edistyksellisiäkin ominaisuuksia. WooCommercen ylläpitämä tietosisältö koostuu asiakkaista, tuotteista ja tilauksista. Tuotteet määritellään joko perus-, muunnelma-, joukko- tai yhteistyötuotteeksi. Eniten käytettyjä tuotetyyppejä ovat perustuote, joka on yksittäinen tuote sekä muunnelmatuote, joka mahdollistaa monipuolisten valintojen tekemisen esimerkiksi värin, koon tai muun ominaisuuden perusteella ja näiden ominaisuuksien eri yhdistelmistä.

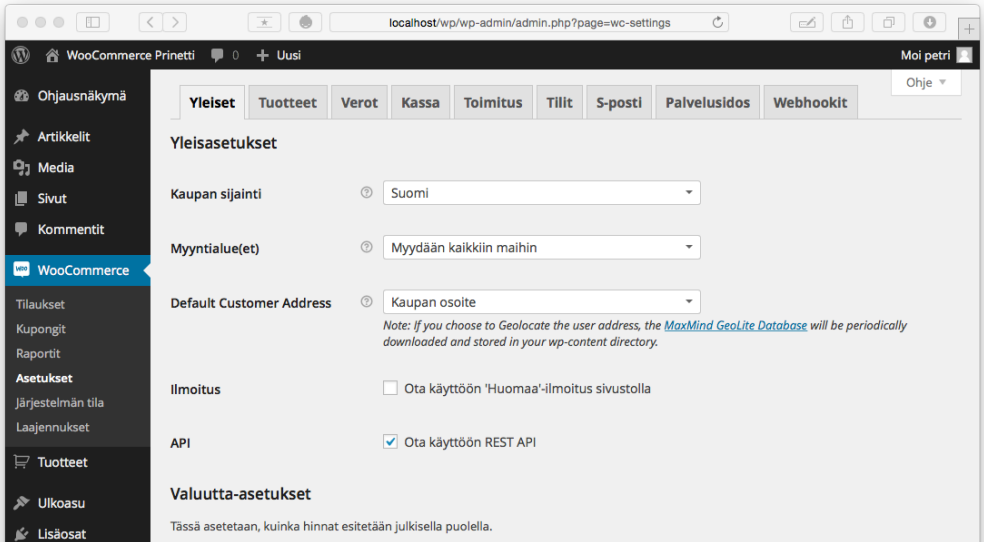
Asiakkaat käsitellään WordPressin käyttäjätilitoiminnallisuuden kautta ja rooliksi asetetaan asiakas. Asiakkaan ei tarvitse olla kirjautunut tililleen tehdäkseen tilauksen, mutta kirjautunut asiakas pystyy seuraamaan tilin kauttaan tekemiään tilauksia myöhemmin ja myös ylläpitäjä voi selailla asiakkaan ostohistoriaa. Tilaukset yhdistetään asiakkaaseen, jos hän on kirjautunut. Tehty tilaus vähentää tuotteille määriteltyä varastosaldoa. Varastosaldoa varten ylläpitäjä voi määrittää, ettei tuotteita myydä yli rajojen, tai että asiakkaalle ilmoitetaan, mikäli tuotetta ei ole heti varastossa, mutta sitä voidaan jälkitoimittaa. Ylläpitäjä voi myös määrittää, ettei

varastosaldoja seurata erikseen, jolloin ostaminen on aina mahdollista eikä asiakkaalle kerrota varastotilannetta.

4.1 WooCommercen toiminnallisuuden laajentaminen

WooCommercen toiminnallisuutta voidaan laajentaa. Syitä laajentamiselle voi olla esimerkiksi paikallisen maksunvälittäjän integrointi tai täysin uudenlainen ominaisuus, jota ei vielä ole lisäosakirjastossa. Erityisesti WooCommercea varten kehitettävien lisäosien kehitys tapahtuu samaa rakennetta noudattaen kuin muidenkin WordPress-lisäosien kehittäminen, mutta erojakin löytyy. Jos lisäosa ei ole ns. ”standalone”-lisäosa, joka on riippumaton muista lisäosista, vaan toimii WooCommerce-lisäosan kanssa yhdessä, on WooCommercen dokumentaatiossa kuvattu tavat, joilla integrointi on mahdollista suorittaa.

Luvussa 3.3 tutustuttiin siihen, miten WordPressissä voidaan tehdä sivu lisäosan asetusten tallentamista varten. Vaikka WordPressissä tällainen mahdollisuus onkin, on parempi keskittää kaikki WooCommercea varten tarvittavat asetukset saman valikon alle, jotta ne olisivat helposti käyttäjän löydettävissä (Kuvio 6). WooCommerce tarjoaa tähän WC_Integration-luokan, jota periyttämällä voidaan luoda asetukset-sivu lisäosille, joiden toiminta on riippuvaista juuri WooCommercesta. Tämän tavan käyttäminen on hyvän tavan mukaista, jotta WooCommerce-ylläpitäjä oppii tietämään kaikkien WooCommerceen-liittyvien asetustietojen löytyvän samasta paikasta. (WooThemes 2015.)



Kuvio 6. WooCommerceen liittyvät asetukset on kerätty yhteen paikkaan.

WooCommerce tukee kahdeksaa eri tietotyyppiä asetusten tallentamisessa:

- text
- price
- decimal
- password
- textarea
- checkbox
- select
- multiselect.

Näille tietotyypeille voidaan määritellä erilaisia argumentteja, jotka muuttavat tapaa, jolla yksittäinen asetus esitetään asetukset-sivulla, kuten placeholder, joka lisää paikanvaraajatekstin tekstikenttiin. Kuviossa 7 havainnollistetaan miten WC_Integration luokasta perittyyn luokkaan tehdään init_form_fields()-funktio, jonka sisälle halutut asetustietueet luodaan. (WooThemes 2015)

```php
<?php

public function init_form_fields() {
        $this->form_fields = array(
            'api_key' => array(
                'title'             => __( 'Prinetti Routing Account', 'woocommerce-prinetti' ),
                'type'              => 'text',
                'description'       => __( 'Enter your account number.', 'woocommerce-prinetti' ),
                'desc_tip'          => true,
                'default'           => ''
            ),
            'debug' => array(
                'title'             => __( 'Secret Key', 'woocommerce-prinetti' ),
                'type'              => 'password',
                'default'           => 'no',
                'description'       => __( 'Enter your secret key given by Posti', 'woocommerce-prinetti' ),
            ),
        );
    }

```

Kuvio 7. Esimerkki init_form_fields()-funktiosta.

4.2 Lokalisointi

WooCommerce on käännetty useille eri kielille, myös suomeksi. Käännös on saatavilla täysin ilmaiseksi ja sen muokkaaminen omiin tarkoituksiin on suoraviivaista, koska kääntäminen tapahtuu samoilla työkaluilla kuin muidenkin WordPress-lisäosien ja -teemojen kääntäminen.

WooCommerce sisältää valmiit toiminnot verojen käsittelyyn ja kulttuurisidonnaisten eroavaisuuksien muokkaamisen. WooCommercen asetuksista määritellään mm. käytetäänkö pilkkua vai pistettä tuhat- vai desimaalierottimena, mitä mittayksiköitä käytetään, sisältääkö tuotteiden hinnat veron ja lasketaanko vero rivikohtaisesti vai loppusummasta. Näiden asetusten lisäksi kauppiaan tehtäväksi jää määritellä eri veroluokat ja se, lasketaanko verot kauppiaan vai asiakkaan osoitteen perusteella.

5 PRINETTI

Prinetti on Postin sopimusasiakkaille tarkoitettu ohjelma, jonka avulla asiakas voi tulostaa lähes kaikki Postin kotimaan ja kansainvälisten lähetysten osoitekortit. Prinetin avulla onnistuvat lähetysten käsittely, seuranta ja lähetystietojen hallinta. Asiakkaalle Prinetin käyttö on täysin ilmaista. Prinetin ylläpidosta vastaa Posti. (Posti 2015.)

Prinetti toimii internetissä. Ohjelmaan kirjaudutaan asiakkaan käyttöönsä saamilla tunnuksilla. Osoitekortin tulostaakseen asiakas valitsee haluamansa palvelun, kirjoittaa vastaanottajan tiedot, lähetyksen tiedot ja tämän jälkeen tulostaa osoitekortin. Posti laskuttaa asiakasta toteutuneiden toimitusten perusteella.

Verkkokaupan kannalta Prinetti on kätevä työkalu, mutta suurten tilausmäärien käsittely, asiakkaan toimitustietojen siirto käsin verkkokauppaohjelmasta Prinettiin, on hidasta. Posti tarjoaa Prinetin internet-versiosta integrointirajapinnan, minkä avulla Prinetti-järjestelmä saadaan kommunikoimaan muiden järjestelmien kanssa.

5.1 Prinetti internet-version integrointirajapinta

Ulkoinen järjestelmä, eli tässä tapauksessa verkkokauppa, saadaan yhdistettyä Prinetti-järjestelmään integrointirajapinnan kautta. Ulkopuolinen järjestelmä toimittaa Prinetin tarvitsemat lähetystiedot sähköisesti. Prinetti tallentaa tiedot lähetysrekisteriin ja palauttaa viitetiedon tallennetuista lähetyksistä. Lähetystietojen tallentaminen ja osoitekortin muodostaminen tapahtuu erillisinä pyyntöinä. (Posti 2014, 4.)

5.2 Tietojen siirto

Tietojen siirto tapahtuu http-pyyntöinä XML-muodossa. Vastaus pyyntöihin välitetään http-vasteessa. Tätä tapaa kutsutaan RESTful-tekniikaksi, jota käsitellään luvussa 6.3. Erillisiä tapahtumia on kaksi. Ensin välitetään osoitekortin muodostukseen vaadittavat tiedot Prinetti-järjestelmään. Prinetti-järjestelmä välittää viitetiedon vastauksena. Toisessa pyynnössä muodostetaan PDF-osoitekortti, jonka

Prinetti-järjestelmä palauttaa pyynnön mukaisesti joko linkkinä Prinetti-palvelimen levylle, osana XML-muotoista paluutiedostoa base64-koodattuna tai suoraan http-vasteeseen ilman kehyksiä, jolloin PDF-tiedosto aukeaa suoraan Adobe Readeriin tai vastaavaan. (Posti 2014, 4–5.)

5.3 Autentikointi

Integraatiorajapinnassa siirrettävät tiedostot ovat XML-muotoisia. Jokaisen pyyntötiedoston tulee sisältää autentikointi- ja reititysssegmentti. Sanoma autentikoidaan reititys-segmentin Key-kentässä, johon muodostetaan arvo konkatenoimalla Routing.Account- ja Rounting.Id-kentät sekä asiakkaalle käyttöönottovaiheessa annettu salausavain ja muodostamalla näin saadusta merkkijonosta md5-tiiviste. (Posti 2014, 5.)

MD5 on salausalgoritmi, joka toimii siten että algoritmille annetaan syötteenä merkkijono, josta algoritmi muodostaa 128-bittisen tiivisteen, joka voidaan esittää 32-merkkiä pitkänä merkkijonona heksakoodatussa muodossa. Algoritmi toimii pelkästään yhteen suuntaan, eli samasta merkkijonosta saadaan aina sama tiiviste, mutta tiivistettä ei voida muuttaa takaisin alkuperäiseksi merkkijonoksi. PHP-kieli sisältää valmiin funktion MD5-tiivisteen muodostamiseksi, md5()-funktiolle annetaan parametrina merkkijono, ja paluuarvona saadaan md5-tiiviste (PHP Group 2015a).

5.4 Lähetystietojen tallennus

Osoitekortin muodostus aloitetaan lähetystietojen tallentamisella. Tätä varten siirtotiedostoon, eli http-pyyntönä välitettävään XML-tiedostoon, kirjoitetaan Shipment-segmentti. Segmenttiin kirjoitettavat tiedot ovat kuvattu liitteessä 2. Osa elementeistä on pakollisia ja muiden pakollisuus riippuu valitusta palvelusta. PHP-kieltä käyttäen lähetystietojen tallennus voidaan tehdä käyttäen CURL-kirjastoa kuvio 8:n esimerkin mukaisesti. (Posti 2014, 6.)

```php
<?php

$file_cont = file_get_contents('test.xml'); $xml_data = $file_cont;
$curl_url = 'https://echannel.prinetti.net/import.php';

$ch = curl_init($curl_url);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: text/xml'));
curl_setopt($ch, CURLOPT_POSTFIELDS, "$xml_data");
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
$output = curl_exec($ch);
curl_close($ch);

```

Kuvio 8. CURL-kirjaston käyttö tietojen lähettämisessä.

5.5 Osoitekortin muodostus

Niin kuin aiemmin mainittu, lähetystietojen tallennus ja osoitekortin muodostus tapahtuvat erillisinä pyyntöinä. Kun lähetystiedot on tallennettu, saadaan vastauksena seurantatunnus, jota käyttämällä voidaan pyytää osoitekortin muodostusta. Usean eri lähetyksen osoitekortit on mahdollista pyytää samaan PDF-tiedostoon lähettämällä osoitekortin muodostuspyynnössä kaikki halutut seurantatunnukset. Mikäli seurantatunnus on osa lähetyserää, kuten monipaketti-palvelun tilanteessa, palauttaa Prinetti-järjestelmä kaikki kyseisen erän osoitekortit. Osoitekortin muodostus tapahtuu XML-tiedoston PrintLabel-segmentillä. Kuviossa 9 on esitetty lähetettävän XML-tiedoston rakenne. Sanoma alkaa routing-segmentillä, jonka avulla tunnistetaan lähettäjä, ja jatkuu PrintLabel-segmentillä, jossa määritetään pdf-tiedoston palautusmuoto ja haettavat lähetykset. (Posti 2014, 10–11.)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<eChannel>
   <ROUTING>
      <Routing.Target>Kohdejärjestelmän tunnus</Routing.Target>
      <Routing.Source>Lähdejärjestelmän tunnus</Routing.Source>
      <Routing.Account>eChannel-tunnus</Routing.Account>
      <Routing.Key>Tarkiste</Routing.Key>
      <Routing.Id>Sanoman yksilöivä id</Routing.Id>
      <Routing.Name>Sanoman nimi</Routing.Name>
      <Routing.Time>Sanoman luontiaika</Routing.Time>
      <Routing.Version>Sanoman versio</Routing.Version>
      <Routing.Mode>Testi-indikaattori</Routing.Mode>
      <Routing.Comment>Kommentti</Routing.Comment>
    </ROUTING>
    <PrintLabel responseFormat='pdf-tiedoston palautusmuoto'>
      <Reference>Linkkitieto lähettävään järjestelmään ( Esim. tilausnumero)</Reference>
      <TrackingCode>Tulostettava seurantakoodi</TrackingCode>
    </PrintLabel>
</eChannel>
```

Kuvio 9. Esimerkki osoitekortin pyynnössä käytetyn XML-tiedoston sisällöstä.

6 TYÖSSÄ KÄYTETTYJÄ TEKNIIKOITA

Tässä luvussa käyn läpi muutamia WooCommerce-Prinetti-lisäosan kehityksessä tarvittuja tekniikoita ja niiden käyttötapoja. Web-kehitys on aina yhdistelmä suuresta määrästä erilaisia tekniikoita, ja tämän opinnäytetyönkin toteutuksessa käytettiin mm. HTML-, CSS-, Javascript-, jQuery-, PHP-, AJAX ja XML-tekniikoita ja kieliä.

6.1 PHP

PHP on ohjelmointikieli, jonka syntaksi on lainattu suurimmaksi osaksi C-kielestä. PHP:n peruskäytössä koodi upotetaan osaksi HTML-dokumenttia. PHP-kieltä ei käännetä konekieliseksi, vaan se tulkataan ajonaikaisesti web-palvelimella. Rantala kirjoitti jo vuonna 2005 PHP:n olio-ominaisuuksien ja poikkeustenkäsittelymekanismien saavuttaneen liki parhaiden olio-ohjelmointikielten tason. Uusia versioita julkaistaan edelleen säännöllisesti, joten nykyään voitaneen nimittää PHP:tä jo puhtaaksi olio-ohjelmointikieleksi. Kieli ei kuitenkaan pakota oliomallin hyödyntämiseen, joten se on aloittelijallekin helppo kieli aloittaa. (Rantala 2005, 24.)

6.2 XML

XML on HTML:ää muistuttava merkkauskieli rakenteellisen tiedon kuvaamiseksi tekstitiedostoon. XML ei kuitenkaan varsinaisesti ole HTML:ää, vaan pikemminkin toisinpäin. HTML-versiot aina 4.01 versioon asti olivat SGML-sovelluksia (Standard Generalized Markup Language). XML taas on muodostettu SGML:stä. Mutta jottei asia olisi liian yksinkertaista, muotoiltiin HTML-uudestaan XML-metakielen avulla, ja tuloksena syntyi XHTML 1.0, jonka voidaan siis sanoa olevan XML-sovellus. HTML5 taas ei ole varsinaisesti XML:ää, mutta siitäkin on olemassa oma XML-laajennoksensa: XHTML5. (Nykänen 2001, 5–6; W3C 2008.)

XML-dokumentin tulee olla hyvin muodostettu. Käytännössä tämä tarkoittaa, että merkkauksen tulee olla virheetöntä, dokumentissa tulee olla vähintään yksi ele-

mentti, dokumentin tulee sisältää yksi juurielementti sekä elementtien alku- ja lopputagien tulee noudattaa tarkasti hierarkiaa menemällä aidosti sisäkkäin. Kuvio 4 havainnollistaa XML-dokumentin rakennetta. (Nykänen 2001, 16–17.)

```
<?xml version="1.0"  ?>
<esimerkki tyyppi="pakollinen">
    <tervehdys>Hei maailma!</tervehdys>
</esimerkki>

```

Kuvio 10. XML rakentuu hierarkkisesti vapaasti määriteltävistä elementeistä.

Yksinkertaistamalla voidaan todeta että XML-dokumentti sisältää esittelyosan, joka sisältää XML-julistuksen, jossa määritellään käytettävän spesifikaation versio sekä esiintymäosan, joka sisältää elementtejä ja elementeille määritettyjä attribuutteja. Nämä elementit, attribuutit ja niiden arvot ovat täysin käyttäjän päätettävissä.

Toisin kuin HTML, jossa selain pyrkii korjaamaan virheet olettamalla tai arvaamalla, XML on tarkka syntaksinsa suhteen. Lopputagin unohtaminen tai huolimattomasti koodattu erikoismerkki ovat virheitä, joita sovelluksen ei tule yrittää arvata. Kun dokumentti on hyvin muodostettu, se voidaan jäsentää puumaiseksi rakenteeksi ja sen sisältö voidaan lukea ohjelmallisesti. (Nykänen 2001, 17–20.)

XML-dokumentti ei itsessään vielä tee mitään tai näytä miltään. Dokumentti on siis vain puumainen rakenne, joka sisältää tietoa ja metatietoa. XML:n hyöty tuleekin sen luomasta mahdollisuudesta välittää tietoa, joka on yksiselitteisesti jäsennelty. Tämä mahdollistaa esimerkiksi kahden järjestelmän keskinäisen tiedonvaihdon verkon välityksellä. Kummankaan järjestelmän ei tarvitse tietää toisistaan mitään, kun tieto välitetään XML-muodossa ja muodostettuna sovittujen sääntöjen mukaisesti.

Jotta XML:n avulla voidaan välittää dynaamista tietoa, on XML-dokumentti pystyttävä muodostamaan ohjelmallisesti. Tuki XML:n käsittelyyn lienee sisäänra-

kennettuna useimpiin eri ohjelmointikieliin. Tässä luvussa tutustutaan PHP-kielen XML:n käsittelyyn tarkoitettuihin ominaisuuksiin.

PHP sisältää XMLWriter nimisen laajennoksen, jonka avulla pystytään muodostamaan XML-dokumentti PHP-kielen avulla. Laajennos on sisällytetty PHP:hen oletuksena versiosta 5.1.2 lähtien, joka julkaistiin jo vuonna 2006, joten laajennoksen voidaan olettaa olevan käytettävissä useimmilla palvelimilla (PHP Group 2015b). XMLWriter-laajennosta voidaan käyttää joko proseduraalisesti tai olio-ohjelmointia noudattaen. XMLWriter-luokka sisältää useita funktioita XML-dokumentin muodostamiseksi. Kuviossa 11 luodaan kuvion 10 esimerkki XMLWriter-luokan avulla.

```
<pre> <!-- HTML elementti pelkästään esimerkin muotoiluun -->
    <?php

    $xml = new XMLWriter(); // Luodaan instanssi XMLWriter-luokasta
    $xml->openMemory(); // Käytetään muistia merkkijonon säilyttämiseksi
    $xml->setIndent(true); // Asetetaan automaattinen sisennys
    $xml->startDocument('1.0" encoding="UTF-8'); // Kirjoittaa otsakkeen
        $xml->startElement('esimerkki'); // Aloittaa elementin
            $xml->startElement('tervehdys');
                $xml->writeAttribute('tyyppi', 'pakollinen'); // Kirjoittaa kokonaisen attribuutin
            $xml->endElement(); // Päättää 'tervehdys'-elementin
        $xml->endElement(); // Päättää 'esimerkki'-elementin

    echo htmlentities($xml->outputMemory());
    ?>
</pre>

```

Kuvio 11. Yksinkertainen esimerkki XMLWriter-luokan käytöstä.

XMLWriter luokkaa käytetään oliomallin mukaisesti ensin luomalla luokasta ilmentymä. Tämän jälkeen valitaan muodostettavan dokumentin käsittelytapa, `openMemory()` -metodilla aloitetaan XML-dokumentin muodostus välimuistiin. Kuviossa 11 esitelty `setIndent()` -metodi pelkästään muotoilee tuotettavan dokumentin asettamalla sisennykset automaattisesti XML-puun mukaisesti. Metodi `startDocument()` luo otsakkeen dokumenttiin, parametreina annetaan XML-versionumero sekä merkistö. XMLWriter-luokan metodit on nimetty siten, että start-alkuiset metodit, kuten `startElement()` ja `startAttribute()`, aloittavat

elementin kirjoittamisen. Nämä metodit tulee sulkea oikeassa kohdassa vastaavilla metodeilla kuten `endElement()` ja `endAttribute()`, jotta XML-dokumentista tulisi hyvin muodostettu. Metodit, jotka alkavat write-sanalla, kuten `writeAttribute()` tai `writeElement()` puolestaan kirjoittavat koko elementin tai attribuutin, eikä näitä elementtejä tai attribuutteja tarvitse erikseen sulkea. Näille metodeille annetaan parametreina elementin tai attribuutin nimi ja arvo. (PHP Group 2015c.)

XML-tiedoston lukeminen PHP-kielellä tapahtuu käyttäen DOMDocument-luokkaa. Luokkaa käytetään oliomallin mukaisesti luomalla instanssi. Kuviossa 12 olevan esimerkin preserveWhiteSpace-propertyn arvoksi asetetaan false, jotta parseri jättää huomiotta epäoleelliset välilyönnit ja tabuloinnit. Seuraavaksi ladataan XML-sisältö luokan käyttöön käyttäen `load()` -metodia. Esimerkissä käytetty `getElementsByTagName()` -metodi palauttaa tulosjoukon taulukkomuodossa (array), koska samannimisiä elementtejä voi olla XML-dokumentissa useita. Lisäämällä metodin perään vielä ”`item(0)->nodeValue`”, palautetaan ensimmäisen parametrina olleen elementin esiintymän arvo. (Abeysinghe 2008, 36–37.)

```
1  <?php
2
3      $doc = new DOMDocument;
4      $doc->preserveWhiteSpace = false;
5      $doc->load('example.xml');
6
7      $example = $doc->getElementsByTagName('example');
8
```

Kuvio 12. XML-tiedoston lukeminen voidaan tehdä DOMDocument-luokan avulla.

6.3 REST

REST eli Representational State Transfer on suosittu web-palveluiden toteutuksessa käytetty ohjelmistoarkkitehtuuri, jonka kuvasi ensimmäisenä Roy Fielding väitöskirjassaan. REST pohjautuu resursseihin (resource) ja se käyttää tiedonvälitykseen http-protokollaa. Resurssit yksilöidään URI:n (Uniform Resource Identifier) avulla. URI:n alaluokka on yleisesti tutumpi URL (Uniform Resource Locator). REST hyödyntää HTTP-verbejä taulukon 1 mukaisesti. (Abeysinghe 2008, 13–15.)

Taulukko 1. HTTP-verbit REST-arkkitehtuurissa.

Verbi	**Kuvaus**
GET	Pyytää URI:n mukaisen resurssin
POST	Lähettää resurssin palvelimella, päivittää resurssin URI:n määrittämässä kohteessa
PUT	Lähettää resurssin palvelimelle tallennettavaksi URI:n määrittelemään sijaintiin
DELETE	Poistaa URI:n määrittämän resurssin
HEAD	Pyytää URI:n määrittämän resurssin metadatan

REST-arkkitehtuuria voidaan käyttää PHP-ohjelmoinnissa CURL-kirjaston avulla. CURL on PHP-laajennos, joka on yleensä valmiiksi asennettuna palvelimelle. CURL:in käytössä on neljä päävaihetta:

1. Alustetaan CURL.
2. Asetetaan lähtöarvot.
3. Suoritetaan CURL.
4. Suljetaan CURL.

Kuviossa 13 havainnollistetaan CURLI:n käyttöä esimerkin avulla. Ensimmäisenä alustetaan muuttuja `curl_init()`-funktion avulla, seuraavaksi asetetaan lähtöarvot `curl_setopt()`-funktion avulla, jonka parametreina annetaan alustettu muut-

tuja, lähtöarvon nimi ja arvo. CURLOPT_URL määrittää käytettävän resurssin ja CURLOPT_GET määrittää käytettävän HTTP-verbin olevan GET. Lopuksi vielä suoritetaan ja suljetaan CURL-skripti. (Abeysinghe 2008, 26.)

```
1  <?php
2
3      $url = "http://etsi.esimerkki.fi/pmj?appName=WooCommercePrinetti";
4      $ch = curl_init();
5      curl_setopt($ch, CURLOPT_URL, $url);
6      curl_setopt($ch, CURLOPT_GET, true);
7      curl_exec($ch);
8      curl_close($ch);
9
```

Kuvio 13. Yksinkertainen esimerkki CURLin käytöstä PHP:llä.

6.4 AJAX

AJAX on akronyymi sanoista ”Asynchronous Javascript and XML”, mikä valottaakin jo sen luonnetta kokoelmana eri tekniikoita. AJAX on tekniikka, jolla verkkosivun sisältö saadaan päivittymään ilman sivun uudelleenlatausta. AJAX:illa on monia erilaisia käyttötapoja, jotka parantavat verkkosivujen käyttökokemusta, sivulle voidaan toteuttaa ”raahaa ja pudota”-toiminnallisuus tai syötteen validointi voidaan suorittaa palvelinta hyväksikäyttäen. AJAX mahdollistaa myös erilaisten lisätietojen hakemisen palvelimelta, esimerkiksi kun käyttäjä vie hiiren osoittimen tiettyyn kohtaan sivulla. Tekniikka ei kuitenkaan ole täysin aukoton, se ei toimi ilman JavaScriptiä ja selainten välisiä eroja toiminnallisuudessa joudutaan ratkomaan. JavaScriptin pakollisuuden vuoksi verkkosivu tulisikin toteuttaa aina ensisijaisesti siten, että se toimii myös ilman AJAX-toiminnallisuutta, ja sen jälkeen lisätään käyttökokemusta parantavia ominaisuuksia AJAX:in avulla.

AJAX:in toiminnallisuus toteutetaan luomalla kutsut palvelimelle taustalla JavaScriptin avulla, näin AJAX ratkaisee ongelman asiakkaan ja palvelimen välisessä viestinnässä mahdollistamalla niiden kommunikoinnin samaan aikaan kun

käyttäjä tekee toimia avoinna olevan sivun kanssa. AJAX:in nimen alkuperässä olevasta XML-sanasta huolimatta XML ei ole välttämätön AJAX:in toiminnan kannalta. Tietoa voidaan välittää XML:n lisäksi myös tekstimuodossa tai JSON-formaatissa. (Brinzarea-Iamandi 2009, 14–16.)

```
<!doctype html>
<html>
<head>

    <script type="text/javascript">
function loadXMLDoc()
    {
    var xmlhttp;
    if (window.XMLHttpRequest)
    {// Luodaan XMLHttpRequest-objekti, modernit selaimet, IE7+
    xmlhttp=new XMLHttpRequest();
    }

    xmlhttp.onreadystatechange=function()
    {
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("esimerkkiDiv").innerHTML=xmlhttp.responseText;
    }
    }
    xmlhttp.open("GET","esimerkki.txt",true);
    xmlhttp.send();
    }
    </script>

    <title></title>
</head>

<body>
    <div id="esimerkkiDiv">
        <h2>Tämä teksti vaihtuu</h2>
    </div><button type="button" onclick="loadXMLDoc()">Vaihda teksti</button>
</body>
</html>

```

Kuvio 14. Yksinkertainen esimerkki AJAX-toiminnallisuuden toteuttamisesta.

Kuviossa 14 esitetään yksinkertainen AJAX-pyyntö, jossa nappia painamalla haetaan esimerkki.txt-tiedoston sisältö asynkronisesti palvelimelta ja vaihdetaan esimerkkiDiv-elementin sisällöksi tekstitiedoston sisältö. Tavallisesti JavaScript-koodi kirjoitettaisiin erillisiin tiedostoihin mutta yksinkertaistamisen vuoksi se on sijoitettu suoraan script-elementin sisällöksi.

7 WOOCOMMERCE PRINETTI -LISÄOSAN KEHITYS

Tämän opinnäytetyön toiminnallisen osuuden, Postin Prinetti-järjestelmän integrointi WooCommerce verkkokauppaan, toimeksiantaja on Kalustemarkkinointi Oy. Tässä luvussa kuvailen käytännön tasolla lisäosan kehitykseen käyttämiäni menetelmiä, pyrin avaamaan valitsemiani ratkaisuja ja niiden syitä sekä pohtimaan vaihtoehtoisia ratkaisumalleja ja vertailemaan niiden eroja. Tein lisäosasta kaksi eri versiota, ensin lähestyin kehitystyötä helpommalla, proseduraalisella tavalla. Loin kaikki välttämättömät toiminnallisuudet siis siten, että ne kulkevat tiedoston suoritusjärjestyksen mukaisesti. Vaikka toteutustapa oli suoraviivainen, totesin sen hankalaksi ylläpitää, kun koodimäärä kasvaa. Päädyin ohjelmoimaan lisäosan uudelleen noudattaen olio-ohjelmointia. Kuvailen tässä opinnäytetyössä vain oliomallin mukaisen prosessin.

7.1 Taustatietoa toimeksiantajasta

Kalustemarkkinointi Oy on huonekalujen ja sisustustuotteiden vähittäiskauppaa harjoittava yritys, jonka kotipaikka on Kurikka. Yrityksen vähittäismyymälä sijaitsee Vaasan keskustassa ja sen markkinointinimi on Stemma Vaasa, Kaluste Niemelä. Kalustemarkkinointi Oy harjoittaa myös verkkokauppaa markkinointinimellä KN2.fi.

Kalustemarkkinointi Oy on perheyritys, jonka toimitusjohtaja Yrjö Niemelä, on aloittanut yritystoiminnan jo 70-luvulla. Yritys on toiminut Vaasassa jo yli 40 vuoden ajan, joten se on hyvin tunnettu alueella. Vuonna 2010 yrityksen toimintaa päätettiin laajentaa ja lähteä mukaan verkkokauppaan. Tämän opinnäytetyön tekijä on ollut perustamassa verkkokauppaa, suunnittelemassa konseptia ja on toiminut perustamisesta asti mm. graafisen suunnittelun ja verkkokaupan kehittämisen sekä toteutuksen parissa. Alun perin verkkokauppa perustettiin Kotisivukonepalveluun, mutta toiminnan laajetessa haluttiin lisää ominaisuuksia ja mahdollisuutta muokata kaupan ulkoasua vapaammin, joten loppuvuodesta 2014 siirryttiin WooCommerce-verkkokaupparatkaisuun ja verkkokauppaa alettiin ylläpitää itse. Verkkokaupparatkaisua valitessa tutustuttiin useisiin markkinoilla oleviin avoi-

men lähdekoodin sovelluksiin ja tehtiin demototeutukset OpenCart ja Magento -alustoille. WooCommerce-alustaan päädyttiin kuitenkin sen laajojen ominaisuuksien ja miellyttävän käyttöliittymän sekä suuren WordPress-yhteisön vuoksi. WooCommerce verkkokaupassa voidaan hyödyntää mitä tahansa WordPress-ominaisuutta ja tästä syystä sen sisällönhallintaominaisuudet ovat omaa luokkaansa verrattuna pelkkiin verkkokauppasovelluksiin. Sisällöntuotanto koettiin tärkeäksi erottautumistavaksi internetissä.

7.2 Vaatimusmäärittely

Tämän lisäosan toiminnallisuus rakennettiin vastaamaan Kalustemarkkinointi Oy:n verkkokaupan tarpeita. Kalustemarkkinointi Oy myy KN2.fi-verkkokaupassaan huonekaluja sekä sisutustuotteita kuluttajille ja yrityksille. Verkkokaupan tuotteet toimitetaan Postin kuljetuspalveluja käyttäen. Lisäosaa varten tarvittavia palveluja ovat nouto postista, kotiinkuljetus tai yrityksiä varten kuljetus perille työpäivän aikana. Näiden palvelujen nimet Postilla ovat Economy 16, Express Flex 21 sekä Express Business Day 14.

Postin pakettipalveluiden lisäpalveluista vaadittiin tässä vaiheessa toteutettavaksi monipakettilähetys sekä lisäksi tulevaisuuden varalta postiennakko, vaikkei se tällä hetkellä olekaan mahdollinen maksutapa KN2.fi-verkkokaupassa.

Lisäosaa varten haluttiin toteutettavaksi myös asetussivu verkkokaupan hallintapuolelle, jossa verkkokaupan ylläpitäjä voi muuttaa oleellisia asetuksia, kuten lähettäjän nimi, osoite ja puhelinnumero sekä tilinumero postiennakkotilityksiä varten. Lisäksi tavoitteeksi asetettiin tehdä lisäosasta mahdollisimman nopea käyttää ja pyrkiä mahdollisimman vähäiseen määrään käyttäjältä vaadittavia toimia.

Lisäosan käyttäjiä ovat verkkokaupan ylläpitäjät, joten todettiin, että käyttäjällä voidaan olettaa olevan käytössä mikä tahansa moderni internet-selain sekä JavaScriptin suoritus päällä.

7.3 Lisäosan toteutus

Toteutin lisäosan jakamalla ohjelman ensin pienimpiin mahdollisiin osa-alueisiin ja lähdin työstämään sitä osa-alue kerrallaan aloittaen siitä, mikä ei ollut riippuvainen mistään toisesta osa-alueesta. Pyrin tekemään kaikki nimeämiset englanniksi, jotta lisäosan jatkokehittäminen olisi mahdollista myös suomea osaamattomalle kehittäjälle. Suomenkielisen käännöksen toteutin WordPressin kääntämiseen tarkoitetulla PoEdit-työkalulla. Lisäosan julkiseksi nimeksi valitsin ”WooCommerce Prinetti” ja sen sisäisesti käytin nimeä woocommerce-prinetti, pyrkien siihen, että nimi olisi mahdollisimman lyhyt mutta kuvaava.

7.3.1 Ohjelmointiympäristön asennus

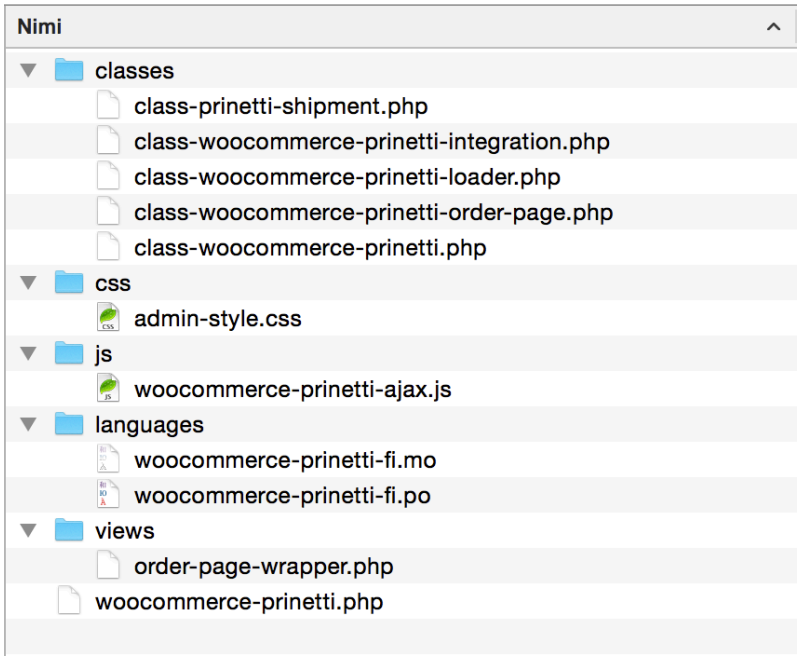
Aloitin lisäosan kehityksen ohjelmointiympäristön asennuksella. Käyttöjärjestelmänä toimi Mac OS 10.10, joka sisältää valmiina Apache-palvelimen sekä PHP-tulkin. Päivitin PHP-tulkin ja asensin MySQL-palvelimen uusimman vakaan version. Web-palvelimen käyttöönotto vaati runsaasti konfigurointia, mutta lopulta sain asetettua Apachen kuuntelemaan http://localhost/ -polkua juurihakemiston ollessa käyttäjäkansiossani sijaitseva Sites-kansio. Tämän jälkeen asensin WordPressin ~/Sites/wp/ -kansioon, loin uuden tietokannan ja ajoin asennusohjelman tavalliseen tapaan. Paikallisen asennuksen ansiosta säästyin jatkuvalta tiedostonsiirrolta oman koneen ja palvelimen välillä.

Web-palvelinympäristön asentaminen pöytäkoneeseen Mac OS 10.10-käyttöjärjestelmään suoraan oli melko työläs prosessi ja asennus vaatii myös ylläpitoa. Joissain tapauksissa käyttöjärjestelmän päivittäminen ylikirjoittaa luodut konfiguraatiot, joten totesin että järkevintä lienee luoda kehitysympäristö joko virtuaalikoneeseen tai käyttää MAMP-ohjelmaa, joka sisältää Apache ja MySQL-palvelimet sekä PHP-tulkin valmiissa paketissa. MAMP on tarkoitettu Mac-ympäristöön, mutta Windows-ympäristölle on oma vastaava toteutuksensa nimeltään WAMP.

7.3.2 Tiedostorakenteen luonti

WordPress-lisäosa voi koostua pienimmillään vain yhdestä tiedostosta. Tämä on lisäosan päätiedosto, joka sijoitetaan `/wp-content/plugins` -kansioon. Jos lisäosa koostuu vain yhdestä tiedostosta, se voidaan sijoittaa suoraan kyseiseen kansioon, muussa tapauksessa luodaan oma kansio lisäosaa varten käyttäen samaa nimeä kuin päätiedostolla, joka sijoitetaan tähän kansioon.

Kehitetyn lisäosan tiedostot sijaitsevat `/wp-content/plugins` -kansiossa ja tiedostorakenne on kuvattu kuviossa 15.



Kuvio 15. Lisäosan tiedostorakenne.

WordPress Codex huomauttaa että lisäosien paikkaa voidaan vaihtaa, joten lisäosan polku tai URL-osoite pitää muodostaa aina konkatenoimalla se `plugin_dir_path()` tai `plugins_url()` -funktioiden palauttaman arvon perään. (WordPress Codex 2015a.)

7.3.3 Lisäosan päätiedosto

Lisäosan tärkein tiedosto (main plugin file) on tiedosto, joka sijaitsee lisäosan juurihakemistossa, tai jos lisäosa koostuu vain yhdestä tiedostosta, sijaitsee se `/wp-content/plugins/` -hakemistossa. Lisäosan päätiedoston tulee sisältää WordPress-dokumentaation mukainen header-osio, joka sisältää tietoja mm. lisäosan kehittäjästä, versionumerosta sekä lisenssistä kuvion 16 mukaisesti. Vähimmäisvaatimus otsaketiedoille on ”Plugin Name” eli lisäosan nimi. Muita rivejä, jos annettu, käytetään ylläpitäjän hallintapuolella olevan lisäosat-sivun tietojen muodostamiseen sekä käännösten suorittamiseen. (WordPress Codex 2015a.)

```
/**
 * Plugin Name:       WooCommerce Prinetti-integraatio
 * Plugin URI:        http://www.makijaakkola.fi/wc-prinetti
 * Description:       Lisää Prinetti-toiminnallisuuden WooCommerce-verkkokauppaan
 * Version:           0.1.0
 * Author:            Petri Mäki-Jaakkola
 * Author URI:        http://www.makijaakkola.fi
 * Text Domain:       woocommerce-prinetti
 * Domain Path:       /languages
 */

```

Kuvio 16. Päätiedoston otsaketiedot.

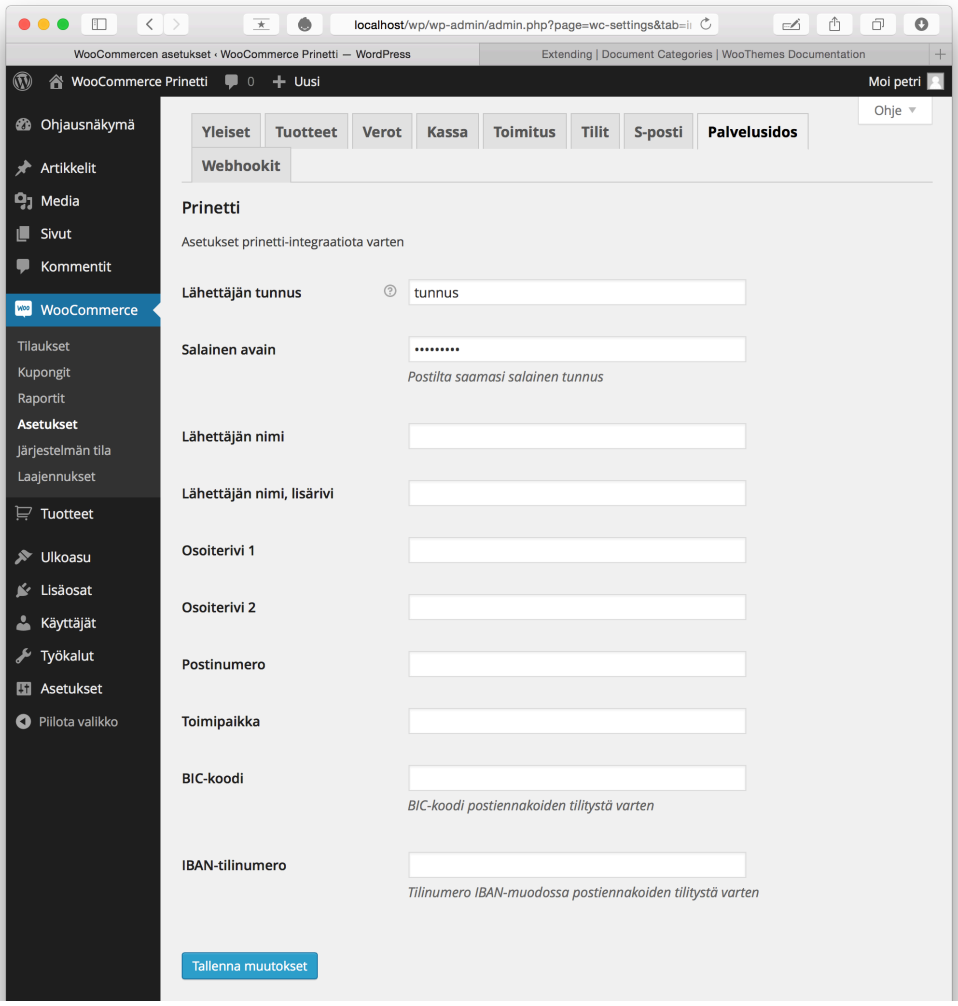
Periaatteessa lisäosan päätiedostoon voidaan kirjoittaa vaikka koko lisäosan toiminnallisuus, mikä onkin hyvin yleistä, jos lisäosa on laajuudeltaan hyvin pieni. WooCommerce Prinetti-lisäosan tapauksessa käytin kuitenkin lisäosan päätiedostoa pelkästään eräänlaisena ”bootstrap”-tiedostona, jonka ainoa tehtävä otsaketietojen esittelyn lisäksi on tehdä instanssi varsinaisesta lisäosan `WooCommerce_Prinetti`-luokasta ja aloittaa sen suoritus.

7.3.4 Asetukset-sivu

Tein lisäosan asetukset-sivun (Kuvio 16) noudattaen WooCommerce-dokumentaatiota, tekemällä `WooCommerce_Prinetti_Integration`-luokan, joka perii `WC_Integration`-luokan ja tekemällä siitä instanssin, joka luodaan lisäosan

päätiedostossa. Asetukset-sivu sisältää kaikki muuttuvat tiedot, jotka ovat riippuvaisia verkkokaupan omistajasta, näin ollen lisäosa voidaan ottaa helposti käyttöön myös muissa kuin toimeksiantajan verkkokaupassa.

`WC_Integration`-luokasta periytetty `WooCommerce_Prinetti_Integration`-luokka sisältää konstruktorin, jossa asetetaan pakolliset propertyt, kuten integraation yksilöivä merkkijono, nimi sekä kuvaus. Konstruktorissa kutsutaan `init_form_fields()`-metodia. Metodin käyttöä on käsitelty luvussa 4.1.

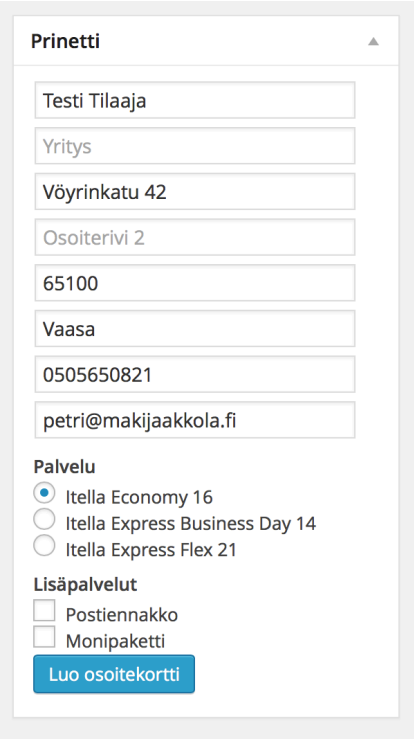


Kuvio 17. WooCommerce asetukset -sivulla verkkokaupan ylläpitäjä voi antaa kauppakohtaiset tietonsa.

7.3.5 Tilaukset-sivun lomake

Tilaukset sivua varten loin oman `WooCommerce_Prinetti_Order_Page`-luokan. Luokan vastuulla on sisällyttää tarvittavat tyylitiedostot sekä lisätä paneeli (meta-box) WooCommercen tilauksenkäsittelysivulle. Luokan instanssi luodaan `WooCommerce_Prinetti`-luokan `define_admin_hooks()`-metodissa ja instanssi annetaan osana `add_action()`-metodin parametreja, jotka sitovat sen `add_meta_boxes`-koukkuun. Tämän koukun kohdalla suoritetaan `WooCommerce_Prinetti_Order_Page` -luokan metodi `add_meta_box()`, joka lisää paneelin `shop_order`-parametrin ansiosta vain, kun kyseessä on tilaus. `add_meta_box()`-metodi puolestaan kutsuu `render_meta_box()`-metodia, joka luo `WC_Order`-olion `$_GET['post']`-muuttujasta saatavalla `post_id`-arvolla ja tämän jälkeen renderöi `woocommerce/prinetti/views/order-page-wrapper.php` -tiedoston sisällön luotuun paneeliin.

Tilaukset-sivun paneelista halusin erittäin minimalistisen, koska tilauksenkäsittelysivu on ennestään jo melko täynnä erilaisia tietoja ja toimintoja. Paneelin sisältö on kuvattu kuviossa 18. Input-kentille on jätetty laittamatta label-kentät tarkoituksellisesti minimalistisen ulkoasun saavuttamiseksi. Kenttien tietosisältö on itseään selittävä ja placeholder-attribuuttiin on annettu kentän kuvaus, joka näkyy vaaleammalla kuin kenttä, johon on annettu tietoa. Kenttien tiedot täytetään `WC_Order`-luokasta luodun instanssin sisältämillä tiedoilla. Tilauksen käsittelijän tehtäväksi jää valita palvelu ja mahdollisesti lisäpalvelu. Lisäpalvelujen vaihtoehdot on piilotettu näkyvistä ja tulevat valittaviksi ainoastaan kun valintaruutu valitaan. Tällöin postiennakon alle aukeaa kenttä, joka sisältää valmiina tilauksen loppusumman ja jos valitaan monipaketti, aukeaa monivalinta, josta voidaan valita arvo kahdesta viiteen. Painamalla ”Luo osoitekortti” -näppäintä, osoitekortin muodostus alkaa ja keskeneräisen käsittelyä kuvataan pienellä animaatiolla. Kun muodostus on valmis, animaatio loppuu ja osoitekortti aukeaa uuteen ikkunaan, josta se voidaan tulostaa. Vastauksena saatu seurantatunnus tallennetaan tietokantaan myöhempää käyttöä varten.



Kuvio 18. Prinetti-toiminnallisuuden hallintapaneeli tilauksenkäsittelysivulla.

Kuviossa 18 nähdään lisäosan tärkein osa, eli paneeli, jossa osoitekortin muodostus tapahtuu tilauksenkäsittelysivulla.

7.3.6 WooCommerce Prinetti -luokka

Tämän luokan tehtävä on ladata muut tarvittavat luokat, alustaa tarvittavat oliot ja määrittää tarvittavat koukut ja niihin liitettävät oliot metodeineen. Luokka hoitaa myös AJAX-pyynnön käsittelyn. Riippuvuudet ladataan `load_dependencies()`-metodilla, jossa luokkien lataamisen lisäksi luodaan instanssi `WooCommerce_Prinetti_Loader`-luokasta, joka on eräänlainen apuluokka joka toteuttaa oliomallin mukaisesti WordPressin koukkujen määrittämisen. Koukut määritetään `define_admin_hooks()`-metodin avulla. Metodi luo instanssin `WooCommerce_Prinetti_Order_Page`-luokasta, joka huolehtii tilauksenkäsittelysivulle tulevan Prinetti-paneelin luonnista.

7.3.7 Prinetti Shipment -luokka

Tässä luokassa tapahtuu lähetystietojen tallentaminen Prinetti-järjestelmään sekä osoitekortin pyyntö. Konstruktori ottaa kaksi parametria. Ensimmäisenä annetaan array, joka sisältää tiedot tilauksesta. Toisena parametrina annetaan lähettäjän tiedot sisältävä array. Konstruktori asettaa tarvittavat tiedot näistä luokan propertyihin. Luokka sisältää `viitenumero()` -funktion, jonka avulla lähtöarvona annettavasta numerosta luodaan viitenumero, joka käy pankkien tilisiirtoihin. Käytännössä funktio muodostaa annetusta luvusta tarkisteen ja palauttaa annetun luvun ja siihen liitetyn tarkisteen. Luokan tärkeimpiä metodeja ovat `createXML()`, joka luo lähetettävän XML-tiedoston sisältöineen sekä `sendXML()`, joka hoitaa tietojen lähetyksen ja palauttaa saadun vastauksen. Luokkaa käytetään AJAX-kutsun käsittelevässä metodissa `WooCommerce_Prinetti` -luokassa. Luokasta luodaan ensin instanssi kuvailluilla lähtöarvoilla, ja sen jälkeen instanssin avulla kutsutaan `createXML()` ja `sendXML()` -metodeja.

7.3.8 Tietoturva

Oleellinen osa kaikkea kehitystyötä on tietoturvanäkökulmien pohtiminen. Tämä lisäosa käsittelee asiakas- ja lähetystietoja sekä myös pankkiyhteystietoja, joita käytetään postiennakkomaksujen tilityksessä, joten tietoturvan huomioiminen on erityisen tärkeää. Tietoturvan toteutus tapahtuu tässäkin tapauksessa monella tasolla. WordPress-sovellus itsessään on valtavan käyttäjäpohjan vuoksi hyvin testattu ja automaattisesti asentuvat tietoturvapäivitykset pitävät sovelluksen ajan tasalla. Lisäksi lisäosakehityksessä on menetelmiä, joita tulee noudattaa tietoturvan edistämiseksi.

Kehitetyn lisäosan kaikki PHP-tiedostot aloitetaan estämällä suora pääsy tiedostoon kuviossa 19 esitellyn skriptin avulla. Tällä tavalla ohjelmaa ei voida suorittaa osoittamalla siihen suoraan selaimen osoiterivillä, vaan sitä voi käyttää ainoastaan palvelin toisaalta lähdekoodista. (WordPress Codex 2015a.)

```
defined( 'ABSPATH' ) or die( 'No script kiddies please!' );
```

Kuvio 19. Määritetään ohjelman suoritus loppumaan, jos siihen pääsyä yritetään suoraan.

Lisäosan Prinetti-järjestelmältä saatavia PDF-tiedostoja, joiden avulla paketit lähetään ja joiden perusteella lähettäjää laskutetaan, ei tallenneta minnekään. Prinetti-järjestelmä välittää vastauksessaan linkin PDF-tiedostoon, joka sijaitsee Prinetti-järjestelmän palvelimella. Tämä linkki on kertakäyttöinen, joten sitä ei voida avata toista kertaa. Ensimmäinen avaus tapahtuu automaattisesti, kun paluutieto saapuu. Toteutin tämän toiminnallisuuden ensin siten, että pyysin paluutietona itse PDF-tiedoston ja tallensin sen omalle palvelimelle, mutta muutin toiminnallisuuden juuri mahdollisten tietoturvaloukkausten välttämiseksi. Tämä heikentää hieman käytettävyyttä, koska käyttäjä ei voi avata tiedostoa uudestaan, jos vaikka tulostuksessa tulee ongelma, muste tai paperi loppuu. Tällöin ainoa tapa on muodostaa kokonaan uusi osoitekortti. Tämä käytettävyyden heikennys todettiin kuitenkin siedettäväksi tietoturvan paranemisen vuoksi.

AJAX-kutsujen suojaksi toteutin NONCE-tarkistuksen. NONCE-merkkijonon vastaavuus tarkistetaan aina ennen AJAX-kutsun prosessointia. NONCEn luomiseen käytin `wp_create_nonce()`-funktiota, jonka luoma merkkijono otetaan osaksi AJAX-kutsun tietosisältöä ja verrataan `WooCommerce_Prinetti`-luokan metodissa, joka käsittelee AJAX-kutsun. NONCEn vahvistaminen tapahtuu `wp_verify_nonce()`-funktion avulla, jos funktio palauttaa arvon false, lopetetaan AJAX-kutsun suoritus.

8 YHTEENVETO

8.1 Oppimisprosessi

Lähtiessäni työstämään tätä opinnäytetyötä, en ollut vielä tutustunut kovin syvällisesti WordPress-lisäosakehitykseen. Teeman kehitys ja WordPress-toteutusten tekeminen oli minulle kuitenkin tuttua, mikä helpotti kehitystyöhön lähtöä. WordPress-kehitys on aivan oma maailmansa, ja mielestäni sain tämän opinnäytetyön aikana vahvan ymmärryksen erilaisista rajapinnoista ja tavoista, joita WordPress-kehityksessä tarvitaan. Lisäosan kehityksessä käytettiin useita eri tekniikoita, mikä vaati jatkuvaa uuden tiedon sisäistämistä. XML oli minulle entuudestaan tuttua, mutta käytännön kokemus sen soveltamisesta puuttui. Tämän opinnäytetyön aikana opin, kuinka XML-dokumentti muodostetaan PHP-ohjelmointikielellä, ja miten se välitetään vastaanottavalle järjestelmälle käyttäen REST-ohjelmistoarkkitehtuuria ja CURL-kirjastoa.

Suurin haaste tässä opinnäytetyössä oli oliomallin hyödyntäminen WordPress-lisäosakehityksessä. WordPress on hyvä ympäristö opiskella web-kehitystä, koska kynnys aloittaa ohjelmointityö on hyvin matala ja toimivan ohjelman saa aikaan vähäiselläkin osaamisella, mutta niin kuin tavataan sanoa, että ohjelmoinnissa monimutkaisuus kasvaa eksponentiaalisesti, niin käy myös WordPress-kehityksessä. Tämän ongelman ratkaisemiseksi on kehittäjän siirryttävä esimerkiksi oliomallin hyödyntämiseen, mikä avaa aivan uusia mahdollisuuksia jatkokehityksen ja ylläpidon kannalta. Sanotaan, että olio-ohjelmointi luo yksinkertaiseen projektiin tarpeetonta monimutkaisuutta ja monimutkaiseen projektiin tarpeellista yksinkertaisuutta, mikä pitää paikkaansa tässäkin tapauksessa. Aiheen tutkimista voisi jatkaa esimerkiksi selvittämällä oliopohjaisten suunnittelumallien (design patterns) soveltamista WordPress-ympäristössä.

8.2 Tulokset

Tutkimuksen aikana vahvistui kuva WordPress-kehityksestä omana osaamisalueenaan. Kehitystyö vaatii useiden erilaisten tekniikoiden tuntemista ja monessa tapauksessa WordPress-sisältää vakiintuneita käytäntöjä, millä tavalla tiettyä tekniikkaa tulee hyödyntää. Vakiintuneiden käytäntöjen tunteminen on osa hyvää ohjelmointitapaa, koska se helpottaa työskentelyä ryhmässä tai toisen koodin ymmärtämisessä ja virheiden jäljityksessä.

Tutkimuksen lähtökohdiksi asetettiin kolme tutkimuskysymystä. Ensimmäisenä haluttiin tutkia, miten WordPress toiminnallisuutta laajennetaan ilman, että muokataan ytimen lähdekoodia. Luvussa 3.1 on käsitelty yksityiskohtaisesti, miten ongelma ratkaistaan. Tärkeimmäksi menetelmäksi paljastui ns. koukkujen käyttö, jonka avulla lisäosa liitetään osaksi WordPress-sovelluksen lähdekoodia.

Toinen tutkimuskysymys koski WooCommerce-verkkokaupan laajentamista, ja miten se eroaa tavallisesta WordPress-lisäosakehityksestä. Tähän kysymykseen vastattiin luvussa 4.1, jossa todettiin, että WooCommerce-kehitys pohjautuu lähes täysin WordPress-lisäosakehityksen malliin ja käytäntöihin, joten WooCommerce-kehityksen osuus tutkimustyössä jäi oletettua vähäisemmäksi ja pääpaino pidettiin WordPress-kehityksessä.

Kolmas tutkimuskysymys käsitteli Prinetti-järjestelmää ja sen toiminnallisuuden toteuttamista verkkokauppaohjelman sisällä. Selvisi, että Prinetti-järjestelmässä on valmis integrointirajapinta ja Posti on julkaissut kattavan oppaan rajapinnan käytöstä. Prinettiä ja sen integrointirajapintaa käsiteltiin luvussa 5. Tutkimuskysymyksen selvittäminen ja siitä johdettu käytännön toteutus onnistuivat oletetusti, vaikka työmäärä olikin arvioitua suurempi.

Opinnäytetyön tuloksena syntyi WooCommerce Prinetti -niminen, toimiva lisäosa, joka on toimeksiantajan päivittäisessä käytössä. Lisäosa noudattaa olio-ohjelmoinnin käytäntöjä, mikä antaa hyvän lähtökohdan sen jatkokehittämiselle ja kaupallistamiselle. Lisäosan jatkokehittämiseksi on suunniteltu tämän opinnäytetyön ulkopuolelle rajattujen Postin palveluiden lisääminen osaksi lisäosaa sekä

lähetyksen seurannan toteutus siten, että asiakas voi nähdä lähetyksen kulun suoraan kirjautumalla asiakastililleen verkkokaupassa.

Tutkimustuloksia voidaan pitää luotettavina, koska ne perustuvat alan kirjallisuuteen sekä ohjelmistojen ja rajapintojen alkuperäisiin dokumentaatioihin. Projektityössä käytetyt menetelmät perustuvat ohjelmistokehityksen parhaisiin käytäntöihin ja tutkimustyössä keskityttiin pohtimaan niitä erityisesti WordPress-kehityksen kannalta.

WordPress-alustan asema maailman suosituimpana sisällönhallintajärjestelmänä teki työstä mielenkiintoisen ja erityisen ajankohtaisen. Myös WooCommercen suosio maailmalla ja Suomessa tekee opinnäytetyön tuloksena syntyneestä lisäosasta tarpeellisen myös muille kuin toimeksiantajalle. Toivon että tämä opinnäytetyö on luonut lukijalle hyvän yleiskuvan WordPress-kehityksestä ja siinä tarvittavista tekniikoista. Koska jokaisesta käsitellystä tekniikasta olisi jo itsessään voinut tehdä opinnäytetyön, jouduttiin sisältö rajaamaan pelkkään esittelyyn, mutta tältä pohjalta on hyvä lähteä tutustumaan tähän erittäin ajankohtaiseen, jatkuvasti kasvavaan, WordPress-kehityksen mielenkiintoiseen maailmaan.

LÄHTEET

Abeysinghe, S. 2008. RESTful PHP Web Services. Birmingham. Packt Publishing.

Brinzarea-Iamandi, B. 2009. AJAX and PHP: Building Modern Web Applications. Birmingham. Packt Publishing.

Built With. 2015a. WordPress Usage Statistics. Viitattu 6.4.2015. http://trends.builtwith.com/cms/WordPress

Built With. 2015b. Websites using WooCommerce. Viitattu 6.4.2015. http://trends.builtwith.com/websitelist/WooCommerce

GNU. 2014. GNU General Public Licence, version 2. Viitattu 15.3.2015. http://www.gnu.org/licenses/gpl-2.0.html

Messenlehner B., Coleman, J. 2014. Building Web Apps with WordPress. Sebastopol. O'Reilly.

Mullenweg, M. 2013. WordPress 3.7 "Basie". WordPress.org. Viitattu 27.3.2015. https://wordpress.org/news/2013/10/basie/

Nykänen, O. 2001. XML. 1. painos. Jyväskylä. Docendo.

PHP Group. 2015a. MD5. Viitattu 6.3.2015. http://it2.php.net/md5

PHP Group. 2015b. XMLWriter, Installations. Viitattu 6.4.2015. http://php.net/manual/en/xmlwriter.installation.php

PHP Group. 2015c. XMLWriter. Viitattu 17.3.2015. http://php.net/manual/en/book.xmlwriter.php

PHP Group. 2015d. Magic Constants. Viitattu 13.4.2015. http://php.net/manual/en/language.constants.predefined.php

Posti. 2014. Prinetin internet-version integrointirajapinta. Viitattu 12.4.2015. http://www.posti.fi/liitteet-yrityksille/muut/ prinetin-integraatiorajapintakuvaus.pdf

Posti. 2015. Prinetti-tulostusohjelma yritysasiakkaille. Viitattu 13.3.2015. http://www.posti.prinetti.fi/main.php

Rantala, A. 2005. Web-ohjelmointi. 1. painos. Jyväskylä. Docendo

W3C. 2008. HTML5, One vocabulary, two serializations. Viitattu 6.4.2015. http://www.w3.org/blog/2008/01/html5-is-html-and-xml/

WooThemes. 2015. Implementing the WC Integration Class. Viitattu 21.3.2015.
http://docs.woothemes.com/document/implementing-wc-integration/

WordPress Codex. 2015a. Writing a Plugin. Viitattu 23.3.2015.
https://codex.wordpress.org/Writing_a_Plugin

WordPress Codex. 2015b. Plugin API. Viitattu 16.3.2015.
http://codex.wordpress.org/Plugin_API

WordPress Codex. 2015c. Function Reference / add action. Viitattu 15.3.2015.
http://codex.wordpress.org/Function_Reference/add_action

WordPress Codex. 2015d. Function Reference / add filter. Viitattu 15.3.2015.
https://codex.wordpress.org/Function_Reference/add_filter

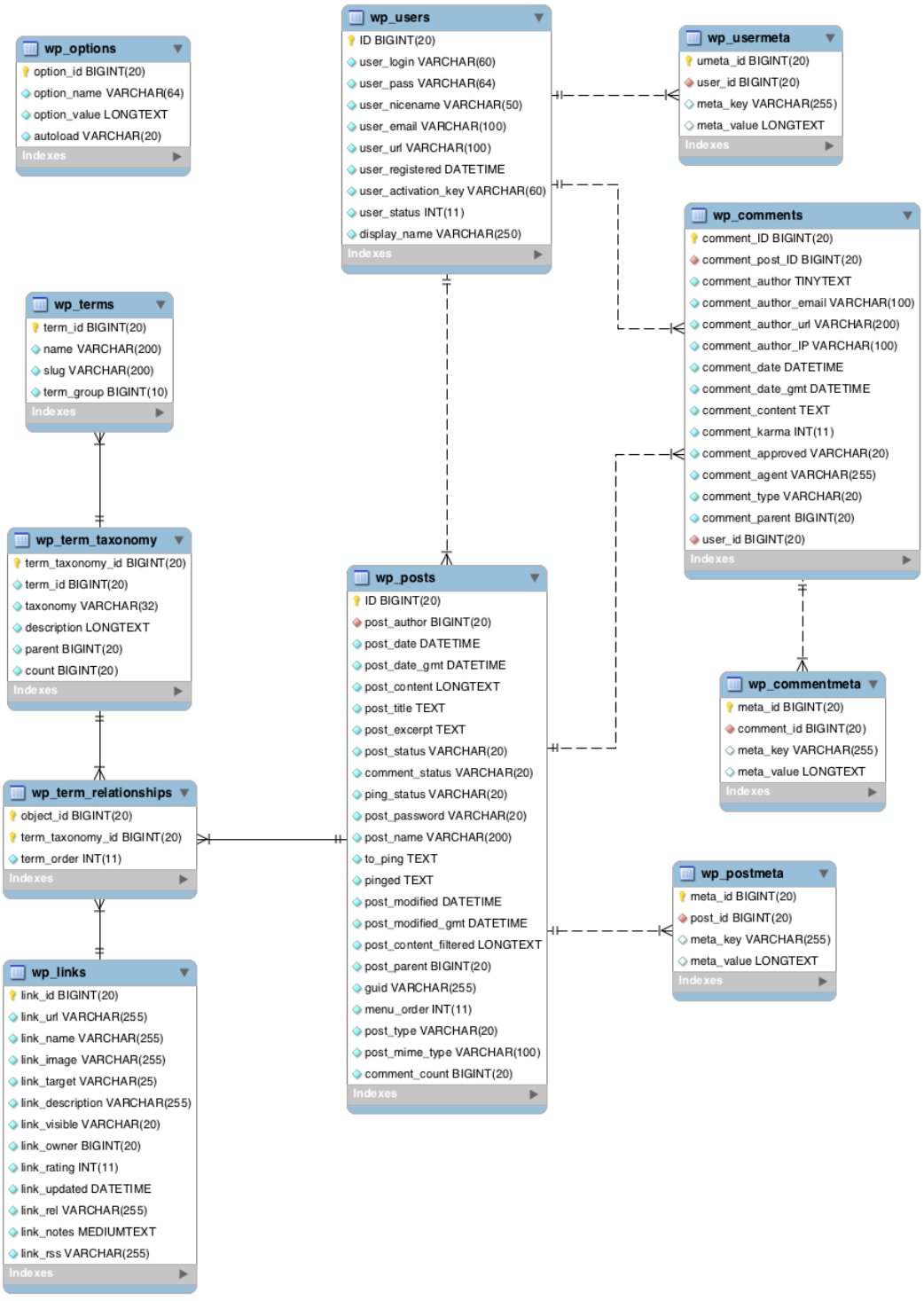
WordPress Codex. 2015e. Options API. Viitattu 17.3.2015.
http://codex.wordpress.org/Options_API

WordPress Codex. 2015f. Creating Tables with Plugins. Viitattu 29.3.2015.
https://codex.wordpress.org/Creating_Tables_with_Plugins

WordPress.org. 2015a. Translator Handbook. Viitattu 3.3.2015.
https://make.wordpress.org/polyglots/handbook/

WordPress.org. 2015b. How to Internationalize Your Plugin. Viitattu 27.3.2015.
https://developer.wordpress.org/plugins/internationalization/
how-to-internationalize-your-plugin/

WordPress-tietokannan rakenne



Shipment-segmentin tietosisältö

Elementti	**Tietosisältö**	**Muoto**
Sender	Lähettäjän tiedot	
Recipient	Vastaanottajan tiedot	
Contractid	Itellan sopimustunnus	AN(10)
Name1, Name2	Osoitekortille tulostuvat nimirivit	AN(35)
Addr1, Addr2, Addr3	Osoitekortille tulostuvan osoitteen rivit	AN(35)
Postcode	Postinumero. Suomalaisilla voimassa oleva viisinumeroinen postinumero	AN(10)
City	Toimipaikan / paikan nimi	AN(35)
Country	ISO-standardin mukainen kaksimerkkinen maakoodi esim. FI, SE, DE, US	AN(2)
Phone	Puhelinnumero	AN(20)
Vatcode	Y-tunnus	AN(20)
Email	Sähköpostiosoite	AN(50)
Consign-ment	Lähetyserä	
Reference	Viitetieto lähettävästä järjestelmästä. Paluutiedostossa luodut lähetys- tunnisteet kohdistetaan tähän arvoon	AN(50)
Product	Osoitekortille valittavan palvelun tuotekoodi. Katso taulukko 3 ”Itellan tuotekoodit”	
Additional-service	Lisäpalvelu-segmentti. Lähetyseräkohtainen lisäpalvelu, joka koskee kaikkia package tasolla määriteltyjä kolleja (esim. MPS ja PE).	
ServiceCode	Lisäpalvelun koodi. Katso taulukko ”Itellan lisäpalvelukoodit ja tarkkennekenttien nimet”	
Specifier	Lisäpalvelun tarkenteen arvo. Atribuutissa ’specifier name’ kerrotaan tarkennekentän nimi. Katso taulukko 4 ”Itellan lisäpalvelut ja tarkennekenttien nimet”	
Contentcode	Kansainvälisten lähetysten lähetyksen luonne ja sisältö. Katso taulukko 6 ”Kansainvälisen lähetyksen sisältökoodit”	
Returninst-	Kansainvälisten lähetysten palautusohjekoodi. Katso taulukko 7	

ruction	”Kansainvälisen lähetyksen palautusohjekoodit”	
Additiona-linfo	Osoitekortille tulostuvat lisätiedot	
Text	Lisätietotekstirivi. Enintään kaksi riviä.	AN(35)
Invoi-cenumber	Kansainvälisen lähetyksen liitelaskun tai liiteproforman numero. Pakollinen lähetettäessä EU:n ulkopuolelle kauppatavaraa, näytettä tai lahjaa	AN(20)
Merchan-disevalue	Kansainvälisen lähetyksen kauppatavaran arvo kun lähetys on kauppatavara, näyte tai lahja.	N
Currency	Kansainvälisen lähetyksen kauppatavara-arvon valuutta. Oletusarvona euro (EUR).	
Parcel	Lähetyksen kollit	
Packagetype	Kollilaji. Kollilajin koodi. Katso taulukko 5 ”Kollilajikoodit”. (Kirjeillä aina ’PC’)	
Contents	Lähetyksen sisällön kuvaus. Pakollinen kansainvälisillä lähetyksillä lähetettäessä EU:hun. Voidaan kertoa myös kollikohtaisesti.	AN(20)
Weight	Kollin paino kilogrammoina	N
Volume	Kollin tilavuus kuutiometreinä	N
Delivery-Time	T oimitusajankohta	
Infocode	Infokoodi. Voidaan tuoda joko lähetyseräkohtaisesti tai kollikohtaisesti	N
Informa-tionservice-code	Tiedotuspalvelutunnus pienlähetykselle	AN(4)
ParcelSer-vice	Kollikohtaiset, erilliset lisäpalvelut monipakettien yhteydessä. Sallittuja kollikohtaisia lisäpalveluja ovat erilliskäsiteltävä. Kollien yhteiset lisäpalvelut määritellään Consignment tasolla.	
ServiceCode	Lisäpalvelun koodi. Katso taulukko 4 ”Itellan lisäpalvelukoodit ja tarkkennekenttien nimet”	
Specifier	Lisäpalvelun tarkenteen arvo. Atribuutissa ’specifier name’ kerrotaan tarkennekentän nimi. Katso taulukko 4 ”Itellan lisäpalvelut ja tarkennekenttien nimet”	
ReturnSer-vice	Palautuskortin tuotekoodi Katso taulukko 3 ”Itellan tuotekoodit”	
Contentline	Sisältötietorivi ulkomaan lähetyksissä (medici-maat)	

Description	Sisältöerittelyrivin kuvaus. Pakollinen rivitieto.	
Quantity	Kpl-määrä, oletuksena 1 kpl	
Currency	Valuutta, joka tulee olla sama eri sisältöerittelyriveillä. Oletuksena EUR.	
Netweight	Sisältöerittelyrivin paino grammoissa (g). Pakollinen rivitieto.	
Value	Sisältöerittelyrivin hinta-arvo. Pakollinen rivitieto jos lähetyksen luonne on näyte, lahja tai kauppatavara.	
Country-OfOrigin	Sisällön alkuperämaa, oletuksena FI	
TariffCode	Tullinimike	