# HÄRMÄLÄN ALUEEN KESKITETTY TAPAHTUMIENHALLINTA

## WordPress-lisäosien toteutus tapahtumienhallintaan

Janne Kähkönen

Opinnäytetyö
Joulukuu 2014
Tietojenkäsittelyn
koulutusohjelma
Ohjelmistotuotanto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotanto

KÄHKÖNEN, JANNE:
Härmälän alueen keskitetty tapahtumienhallinta
WordPress-lisäosien toteutus tapahtumienhallintaan

Opinnäytetyö 61 sivua, joista liitteitä 8 sivua
Joulukuu 2014

---

Moni yhteisö järjestää erilaisia tapahtumia, joista tiedotetaan tällä hetkellä usein ainoastaan toimijan sivuilla ja ehkä lisäksi myös muissa medioissa. Jotta tieto tapahtumista leviäisi tehokkaammin, tapahtumailmoituksien tietosisällöt pitäisi standardoida koneluettaviksi.

Opinnäytetyön tarkoituksena oli luoda kustannustehokas keskitetty tapahtumatietojen hallintajärjestelmä pienille toimijoille, niin ettei synny riippuvuutta yhdestä suljetusta tekniikasta tai palveluntuottajasta. Järjestelmä luotiin avointa lähdekoodia ja -dataa, semanttista verkkoa sekä linkitetyn datan -ratkaisuja käyttäen. Tavoitteena oli tutkia käytettäviä tekniikoita, palvelun rakennetta ja sen toimintaa sekä luoda selkeyttävä dokumentaatio palvelun jatkokehitystä ja käyttämistä varten. Opinnäytetyöni toimeksiantajana oli Tampereen kaupunkilähetys ry:n ja Tampereen Vanhuspalvelu ry:n yhteinen Likioma-projekti.

Opinnäytetyön tuloksena syntynyt keskitetty hallintajärjestelmä luotiin WordPress-alustan päälle toteutettuja lisäosia hyödyntäen sekä WWW-tekniikoita käyttäen. Toimeksiantajan palautteen perusteella toteutettu palvelu oli onnistunut, ja se otetaan kehitys- ja tuotantokäyttöön vuoden 2014 lopussa. Yhdessä raportin kanssa palvelu mahdollistaa ja ohjeistaa jatkokehityksen, keskitetyn tapahtumienhallinnan käyttöönoton sekä sen käyttämisen myös pienille toimijoille, sillä erikoisosaamista ja suurempia resursseja ei vaadita.

---

Asiasanat: wordpress, lisäosa, tapahtumienhallinta, json-ld, semanttinen verkko.

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

KÄHKÖNEN, JANNE:
The Centralized Event Management of Härmälä Area
Programming the WordPress Plugins for Event Management

Bachelor's thesis 61 pages, appendices 8 pages
December 2014

---

Various organizations coordinate, manage and organize events. The information about these events needs to be provided and presented somehow to the targeted audience. Often it is only through the website of the provider and maybe via some other media like a newspaper that the information is passed on. Standardizing the information flow will make it easy for interpretation by client applications and developers.

The purpose of this thesis was to create a centralized cost-effective event management system for small organizations which is easy to use, maintain and develop in the future without dependency on specific provider or technique. The objective of this thesis was to research and experiment with the possibilities that open source event management software has for small organization purposes. The new system was built on top of the content management system WordPress by programming the needed WordPress plugins. All the techniques used in the implementation were carefully considered and properly demonstrated.

The product of the thesis was a prototype of the event management system, which consists of two WordPress plugins. The host plugin allows adding and managing of the events, while the client plugin allows query commands to fetch them from the host and to display them. The information is passed on by an application programming interface, which is embedded in the host plugin. Together with the created documentantion, the plugins provide an easy event management without special programming knowledge or expensive resources from the end user, compared to commercial solutions. According to the client, the product was successful, and will be developed further in the future.

---

Keywords: wordpress, plugin, event management, json-ld, semantic web.

**SISÄLLYS**

**LYHENTEET JA TERMIT**

| | |
|---|---|
| AJAX | Asynchronous JavaScript And XML, joukko verkkosovelluskehityksen tekniikoita |
| Apache | Avoimeen lähdekoodiin perustuva HTTP-palvelinohjelma |
| API | Application Programming Interface, ohjelmointirajapinta |
| Branch | Oma kehityshaaransa Git-versionhallinnassa |
| CMS | Content Management System, sisällönhallintajärjestelmä |
| CSS | Cascading Style Sheets, WWW-dokumenttien tyylimäärittelykieli |
| Git | Versionhallintaohjelmisto |
| GPL | GNU General Public License, GNU-hankkeen yleinen lisenssi (lyhennettynä GNU GPL tai pelkkä GPL) |
| HTML | Hypertext Markup Language, hypertekstin merkintäkieli |
| HTTP | Hypertext Transfer Protocol, hypertekstin siirtoprotokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon |
| i18n | Internationalization, sovellusten kansainvälistäminen |
| IRC | Internet Relay Chat, pikaviestintäpalvelu |
| JavaScript | Yleiskäyttöinen, kevyt ja löyhästi tyypitetty ohjelmointikieli, jota käytetään pääasiassa internet-ohjelmoinnissa |
| jQuery | Avoimen lähdekoodin JavaScript-kirjasto |
| JSON | JavaScript Object Notation, yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen |
| JSON-LD | Tarjoaa JSON-pohjaisen linkitetyn datan standardin, joka parantaa nykyisten JSON-sovellusten semanttista yhteensopivuutta |
| l10n | Localization, kansainvälistetyn sovelluksen lokalisointi |
| LAMP | Kokoelma avoimen lähdekoodin ohjelmia, jotka yhdessä muodostavat WWW-palvelimen |
| MariaDB | Yhteisön kehittämä MySQL:ään pohjautuva vapaan lisenssin relaatiotietokantajärjestelmä |
| Merge | Kehityshaaran liittäminen Git-versionhallinnassa |
| MySQL | Relaatiotietokantaohjelmisto |
| MVC | Model-View-Controller, ohjelmistoarkkitehtuurityyli jonka tarkoituksena on käyttöliittymän erottaminen sovellutiedosta |

| | |
|---|---|
| Nginx | Avoimeen lähdekoodiin perustuva HTTP-palvelinohjelma |
| PDF | Portable Document Format, PostScript-kieleen pohjautuva tiedostomuoto |
| Permalink | Selkolinkki, lyhytlinkki |
| PERL | Practical Extraction and Report Language, tulkattava proseduraalinen ja skriptimäinen ohjelmointikieli |
| PHP | Ohjelmointikieli, joka soveltuu erityisesti verkkosovellusten ohjelmointiin |
| Python | Monipuolinen ja tulkattava ohjelmointikieli |
| RDF | Resource Description Framework, standardoitu malli datan välisten yhteyksien kuvaamiseen |
| REST | Representational State Transfer, arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen |
| UML | Unified Modeling Language, standardoitu graafinen mallinnuskieli |
| Unit testing | Yksikkötestauksella tarkoitetaan lähdekoodiin kuuluvien yksittäisten osien, kuten olion funktioden, testaamista |
| WWW | World Wide Web, Internet-verkossa toimiva hajautettu hypertekstijärjestelmä. |
| XAMPP | Avoimen lähdekoodin ohjelma, joka koostuu pääasiassa HTTP-palvelimesta, MySQL-tietokannasta sekä PHP-, PERL- ja Python-skripteistä. |
| XML | Extensible Markup Language, merkintäkieli jolla tiedon merkitys on kuvattavissa tiedon sekaan |

# 1 JOHDANTO

Tällä hetkellä moni yhteisö järjestää erilaisia tapahtumia. Näistä tapahtumista tiedotetaan usein ainoastaan toimijan sivuilla ja ehkä lisäksi myös lehdessä, radiossa ja sosiaalisessa mediassa. Jotta tieto tapahtumista leviäisi vieläkin tehokkaammin, tapahtumailmoituksien tietosisällöt pitäisi standardoida koneluettaviksi. Tapahtumien hallintaan on jo valmiiksi olemassa useita sovelluksia ja ratkaisuja. Oman tuotteen toteuttaminen voi kuitenkin olla hyödyllistä ylläpidettävyyden ja tuotteen elinkaaren kannalta, varsinkin, jos resurssit ovat pienet.

Tätä tarkoitusta varten toteutettiin toimeksiantajan pyynnöstä prototyyppinä Härmälän alueen tapahtumatietojen hallintaa varten palvelu, johon alueen eri toimijat syöttävät tapahtumatietonsa. Palvelu myös mahdollistaa tapahtumien esittämisen ja jakamisen avoimen rajapinnan kautta. Opinnäytetyöni toimeksiantajana toimi Tampereen kaupunkilähetys ry:n ja Tampereen Vanhuspalvelu ry:n yhteinen Likioma-projekti.

Opinnäytetyön tarkoituksena oli luoda kustannustehokas tapahtumatietojen hallintajärjestelmä pienille toimijoille, niin ettei synny riippuvuutta yhdestä suljetusta tekniikasta tai palveluntuottajasta. Järjestelmä luotiin avointa lähdekoodia ja -dataa, semanttista verkkoa sekä linkitetyn datan -ratkaisuja käyttäen. Tavoitteena oli tutkia käytettäviä tekniikoita, palvelun rakennetta ja sen toimintaa, sekä luoda selkeyttävä dokumentaatio palvelun jatkokehitystä ja käyttämistä varten.

Raportista ja toteutetusta palvelusta on hyötyä kaikille tapahtumatietojen hallinnan parissa toimiville, ja myös pienten toimijoiden on helppo ottaa kyseinen tekniikka käyttöön, koska se ei vaadi juurikaan erikoisosaamista tai suuria resursseja.

Opinnäytetyöni alkaa palveluun liittyvällä teorialla. Käytetyt tekniikat esitellään yleisesti. Lisäksi myös perustellaan, miksi kyseisiin tekniikoihin on päädytty. Toteutettu palvelu on varustettu teknisellä rajapinnalla, jolloin kuka tahansa voi hakea palvelusta tiedot omiin julkaisukanaviinsa. Myös rajapinnan toiminnallisuus esitellään lyhyesti. Toteutuksessa käytetään osittain hyväksi myös Helsingin kaupungille tehtyjä määrityksiä, jotka liittyvät avoimen tiedon Linked Events -toiminnallisuuteen. Tapahtumatietojen formaattina käytetään Schema.org:in määrityksiin pohjautuvaa JSON-LD-formaattia, joka myös esitellään teoriaosuudessa.

Teoriaosuuden jälkeen keskitytään opinnäytetyöni pääasialliseen tarkoitukseen, eli lisäosien suunnitteluun ja toteutukseen. Lisäosien suunnitteluvaiheen tärkeimmät ja oleellisimmat seikat esitellään yhdessä tärkeimpien toiminnallisuuksien kanssa.

Työssä on käytetty painettuja kirjallisia lähteitä siltä osin kuin oli mahdollista ja järkevää. Tämä toteutui varsinkin tiedon kohdalla, joka ei ole niin usein päivittyvää kuin verkossa käytettävät tekniikat yleensä ovat. Tästä syystä on päädytty pääasiassa käyttämään ajankohtaisia ja oleellisia verkkolähteitä ja -keskusteluja sekä ns. hiljaista tietoa. Työtä aloittaessa aiempaa kokemusta WordPress-lisäosien toteuttamisesta ei ollut, joten varsinkin WordPress-yhteisön tarjoama tuki oli todella tärkeää.

# 2 TAUSTAA

## 2.1 Ongelmat tapahtumienhallinnassa

Tapahtumien järjestäminen ja näistä tiedottaminen on yleensä oleellinen osa yhteisöiden sekä muiden organisaatioiden toimintaa. Tapahtumia, tietoa tapahtumista sekä näiden metatietoja tulee kuitenkin voida hallita mahdollisimman helposti ja edullisesti.

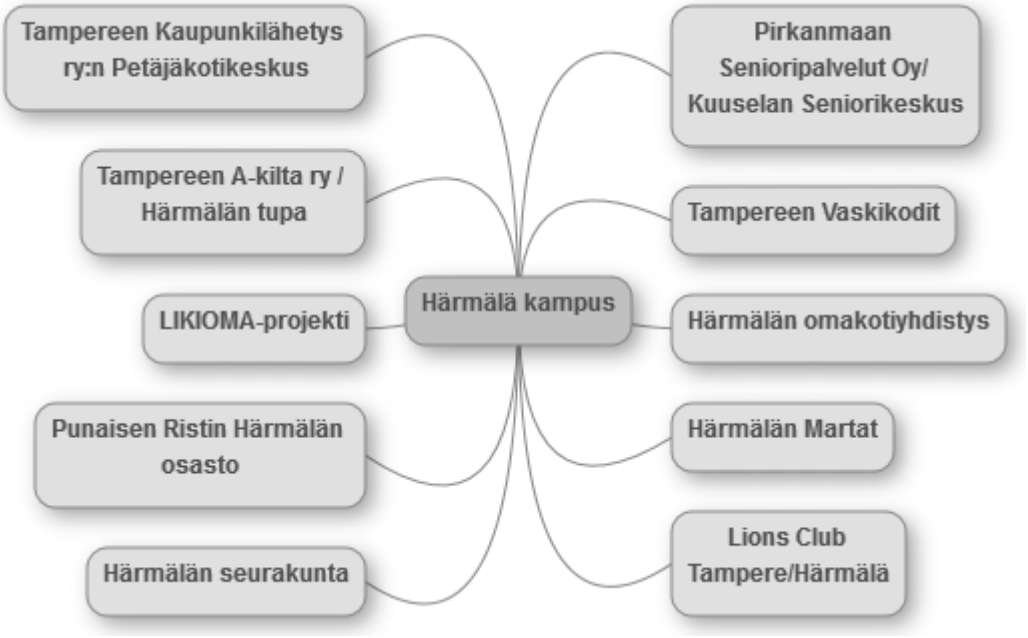
Esimerkkinä hallinnan vaikeuksista voidaan esittää ongelmat pääkaupunkiseudun julkisten toimijoiden parissa, joista useat ovat kehittäneet oman tapahtumakalenterisovelluksensa. Tapahtumatietojen ylläpitoa pidettiin aikanaan sen verran yksikertaisena tiedonhallintaan liittyvänä haasteena, ettei yhteistyötä katsottu tarpeelliseksi. Sama tapahtuma saattoi esiintyä useassa eri tietokannassa, usein vieläpä toisistaan poikkeavin tiedoin. Tämä on johtanut päällekkäiseen työhön sekä korkeisiin ylläpitokustannuksiin. Tieto ei ole myöskään kulkenut hyvin eri järjestelmien välillä, vaikka rajapintoja onkin tehty. Tapahtuman järjestäjälle on myös ollut erittäin työlästä lähettää tiedot useaan eri kohteeseen, esimerkiksi eri toimijoiden kalentereihin, sillä jakelukanavia on nykyään useita ja niiden määrä kasvaa jatkuvasti.

Ratkaisuksi on ehdotettu, että luodaan yksi hyvin määritelty ja rakenteeltaan joustava tapahtumatietokanta, johon tapahtumatiedot syötetään, ja joka eriytetään eri toimijoiden tapahtumasivustoista ja kalentereista. Toimijoiden asiakassovellukset hakevat tiedot yhteisestä tietokannasta täysin avoimen ja ilmaisen rajapinnan kautta haluamallaan tavalla. (Koponen 2012.)

## 2.2 Toimeksiantajan vaatimukset

Härmälän alueen tapahtumatietojen hallintaa varten halutaan yhteinen sivusto tai palvelu, johon alueen eri toimijat syöttävät (tai siirtävät) tapahtumatiedot. Alueen toimijat esitellään kuviossa 1. Sivusto on varustettu teknisellä rajapinnalla, jolloin kuka tahansa voi hakea sivustolta tiedot omiin julkaisukanaviin. Toteutuksessa käytetään hyväksi Helsingin kaupungille tehtyjä määrityksiä liittyen avoimen tiedon Linked Events -toiminnallisuuteen. Tapahtumatietojen formaattina käytetään schema.org:in

määrittelyihin pohjautuvaa JSON-LD-formaattia. Palvelusta haluttiin avoimen lähdekoodin ratkaisuilla toteutettu, ja myös rajapinnat noudattavat avoimia standardeja.



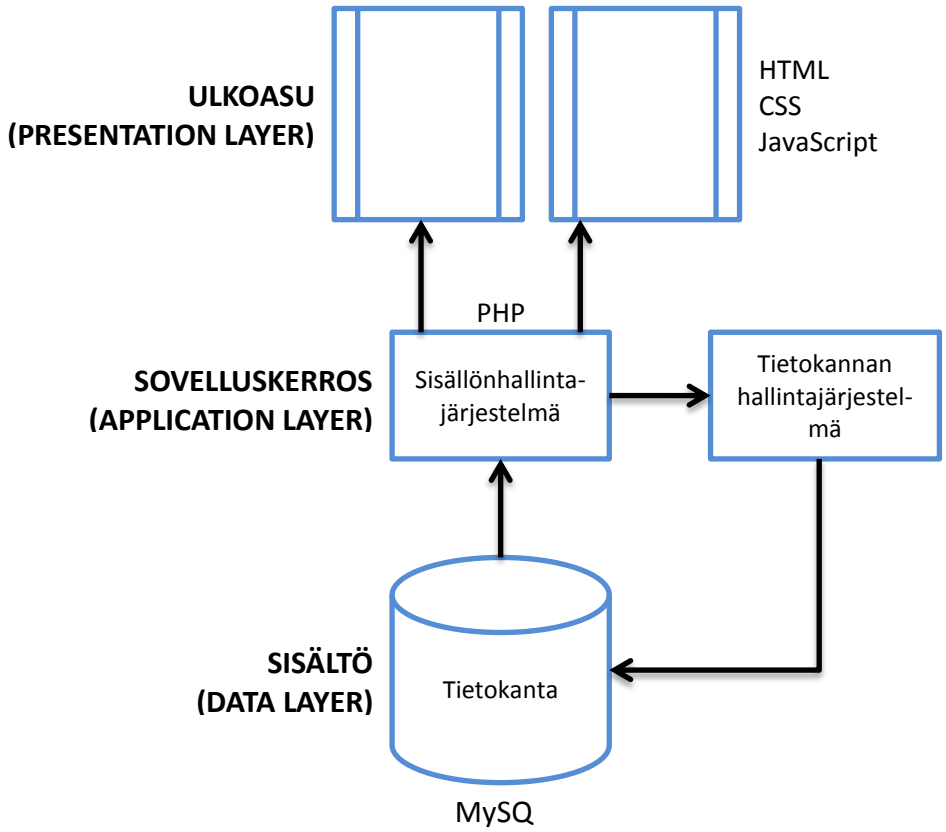

KUVIO 1. Härmälän alueen toimijat

# 3 SISÄLLÖNHALLINTAJÄRJESTELMÄT

## 3.1 Mitä ovat sisällönhallintajärjestelmät

Sisällönhallintajärjestelmä (CMS, Content Management System) on yleisnimitys sisällön hallintaan ja julkaisuun erikoistuneelle tietojärjestelmälle. Sisällönhallintajärjestelmälle ei kuitenkaan ole olemassa mitään varsinaista vakiintunutta määritelmää, vaan toimialasta riippuen sillä saatetaan viitata eri tavoin painottuneeseen tietojärjestelmään käyttäjien tarpeiden mukaan. Sisällönhallintajärjestelmiä on siis useita erilaisia ja niitä käytetään erilaisiin käyttötarkoituksiin. Yksi näistä käyttötarkoituksista on WWW-sisällönhallinta.

WWW-sisällönhallintajärjestelmälle tunnusomaisia piirteitä ovat älykkäät ja dynaamiset sivupohjat, metatietojen hallinnan korostunut rooli, sisältöjen personointi käyttäjälle, sekä sisällön ja ulkoasun erottaminen toisistaan (Boiko 2005). Kuviossa 2 esitellään yksinkertainen esimerkki sisällönhallintajärjestelmän toimintamallista.




KUVIO 2. Esimerkki sisällönhallintajärjestelmän toiminnasta

## 3.2 WordPressin valinta sisällönhallintajärjestelmäksi

Sisällönhallintajärjestelmän valintaan vaikutti eniten sen tunnettavuus, avoimuus sekä helppokäyttöisyys. Yhtenä vaatimuksena oli myös se, että itse sivusto lisäosien lisäksi tulee olla helposti asennettavissa myös mahdollisesti kokemattomalle ylläpitäjälle.

WordPressillä on erittäin suuri kehittäjäyhteisö ja käyttäjäkunta, joten valmiiden ulkopuolisten komponenttien (esim. lisäosat) löydettävyys ja käyttömahdollisuus ovat suuri etu jatkokehityksen kannalta. Tämä mahdollistaa nykymuodossaan monipuolisten sivustojen rakentamisen suhteellisen helposti. WordPressin eduiksi voidaan lukea helppo asennus, päivitettävyys sekä käyttäjäystävällisyys.

W3Techin tilastojen mukaan WordPressin osuus sisällönhallintajärjestelmää käyttävistä sivustoista on tällä hetkellä 61,1 %. Toisena on Joomla 7,8 %, ja kolmantena Drupal 5,1 % osuudella. WordPressin osuus kaikista aktiivisista internetsivuista on 23,2 %. (W3Techs 2014.)

WordPress on sisällönhallintajärjestelmistä selvästi tunnetuin, suosituin ja käytetyin. Lisäksi WordPress oli itselleni ja toimeksiantajalleni jo ennestään jonkin verran tuttu, joten aikaa ei haluttu käyttää uusien järjestelmien opettelemiseen. Vertailuja eri sisällönhallintajärjestelmien välillä tehtiin vain pintapuolisesti, sillä niiden vertaileminen ei ollut opinnäytetyön varsinainen tarkoitus.

## 3.3 WordPressin historia ja toimintaperiaate

WordPress on avoimeen lähdekoodiin perustuva julkaisujärjestelmä, joka on julkaistu GPLv2-lisenssillä tai myöhemmällä versiolla GPL-lisenssistä. Avoin lähdekoodi tarkoittaa sitä, että toimittajalla ei ole omistusoikeutta toimitettuun tuotteeseen. Näin ollen avoimen lähdekoodin järjestelmistä, kuten WordPress, ei peritä lisenssi- tai käyttömaksuja. Lisäksi tuotteen tilaaja ei ole riippuvainen yhdestä toimittajasta sivujen päivityksen ja jatkokehityksen suhteen. Avoimen lähdekoodin periaatteisiin kuuluu myös vapaus käyttää ohjelmaa mihin tarkoitukseen tahansa, sekä oikeus kopioida ja levittää sen alkuperäistä tai muokattua versiota vapaasti. (WordPress 2014a.)

Ensimmäinen versio (versio 0.7) WordPressistä julkaistiin 27.5.2003. Se tunnettiin aluksi pitkään pelkkänä blogialustana, ja vasta myöhemmässä vaiheessa sitä alettiin käyttämään laajemmin myös muihin tarkoituksiin. (WordPress 2003; WordPress 2014b.) WordPressin edeltäjä oli b2/cafelog-nimellä tunnettu järjestelmä, josta haarautui toinenkin projekti, joka tunnetaan nimellä b2evolution.

WordPress tarvitsee toimiakseen WWW-palvelimen, johon se asennetaan ja missä sitä tullaan käyttämään. Nykyinen versio 4.0 tarvitsee toimiakseen PHP-version 5.2.4 tai uudemman, sekä MySQL-version 5.0 tai uudemman. Palvelimen suositellaan olevan joko Apache tai Nginx, mutta muutkin ratkaisut ovat mahdollisia. Ongelmia toki voi esiintyä, jos suositeltuja ratkaisuja ei noudateta. (WordPress 2014c.)

WordPress on suunniteltu tapahtumaohjattua suunnittelumallia (event-driven architecture) käyttäen (McFarlin 2013a.). Se ei ole siis esimerkiksi MVC-mallin mukainen, vaikka WordPress-aiheisissa verkkokeskusteluissa usein törmääkin epäselvyyksiin asiasta (WordPress Support 2014). Tapahtumaohjattuja suunnittelumalleja on useita erilaisia ja WordPress-kehityksessä usein käytettyjä ovat esimerkiksi *Observer Pattern*, *Singleton Pattern* ja *Simple Factory Pattern*. On pitkälti kehittäjästä ja tapauksesta kiinni, minkä lähestymistavan hän valitsee, kunhan vain noudattaa oikeaoppisia koodaustandardeja. Suunnittelumallit helpottavat kuitenkin hyvin paljon, varsinkin suurempien kokonaisuuksien toteutuksessa. (McFarlin 2013b.) Tapahtumaohjatussa ohjelmointimallissa yleisin tapa saada ohjelma reagoimaan tapahtumiin on laatia sille päättymätön silmukka, jossa tapahtumat käsitellään ja joka toistaa tapahtumien vastaanottoa sekä käsittelyä.

# 4 KÄYTETYT TEKNIIKAT WORDPRESS-YMPÄRISTÖSSÄ

Tässä luvussa kuvaillaan käytettäviä tekniikoita yleisesti ja selvitetään, miksi näiden käyttämiseen on päädytty. Alustana toimi sisällönhallintajärjestelmä WordPress, joten tämä pitkälti määritteli käytettävät tekniikat. Itse sisällönhallintajärjestelmään ei paneuduta syvällisemmin, mutta järjestelmän tarjoamaan ohjelmointirajapintaan ja sen alirajapintoihin tutustutaan tarkemmin luvussa 6.

## 4.1 PHP

Lisäosien ohjelmoinnissa pääasiallinen ohjelmointikieli on PHP, joka on suosittu varsinkin WWW-sivujen toteutuksessa. Koodi suoritetaan palvelimella ennen WWW-sivun lähettämistä selaimelle, minkä ansiosta PHP ei vaadi tukea selaimelta, ja sillä voidaan käsitellä esimerkiksi palvelimen tiedostoja ja tietokantoja. (PHP Manual 2014.)

WordPress-sisällönhallintajärjestelmä on rakennettu pääasiallisesti PHP-kielellä. PHP-ohjelmointikielen käyttäminen on tästä syystä pakollista WordPress-lisäosakehityksessä.

## 4.2 MySQL

WordPressissä yleisesti käytettävä tietokantajärjestelmä on MySQL. Muiden tietokantajärjestelmien käyttäminen ei tällä hetkellä ole suositeltavaa (WordPress Codex 2014a). Esimerkiksi MySQL:ään pohjautuvaa MariaDB-tietokantajärjestelmää on kuitenkin mahdollista käyttää pienin rajoituksin IRC-verkosta keskusteluissa saadun tiedon, sekä muiden verkkokeskustelujen perusteella.

MySQL on relaatiotietokanta, joka on erittäin suosittu verkkopalveluiden tietokantana. MySQL-tietokannan päälle rakennettava verkko-ohjelmisto tehdään usein PHP-, Perl- tai Python-ohjelmointikielillä ja julkaistaan Apache-palvelimella, joka toimii Linux-käyttöjärjestelmän päällä. Tällöin järjestelmää voidaan kutsua LAMP-alustaksi. WAMP-alustaksi kutsutaan muuten samanlaista yhdistelmää, mutta Windows-

käyttöjärjestelmällä varustettuna. Lisäksi myös muidenkin ohjelmointikielien käyttö on mahdollista.

### 4.3 HTML & CSS

HTML (Hypertext Markup Language), eli hypertekstin merkintäkieli, ja CSS (Cascading Style Sheets), eli kaskadiset tyyliohjeet, ovat internetsivujen luomiseen käytetyt pääteknologiat. HTML:n avulla sivuille luodaan rakenne ja tuotetaan sisältö, CSS:llä puolestaan määritellään sivujen ulkoasu. (W3C 2013.)

### 4.4 JavaScript

JavaScript on Brendan Eichin vuonna 1995 kehittämä dynaamisesti tyypitetty tulkattava oliopohjainen komentosarjakieli. Standardoitua JavaScriptia kutsutaan nimellä ECMAScript. Ensimmäinen standardin mukainen versio hyväksyttiin vuonna 1997, minkä jälkeen standardia on kehitetty suhteellisen aktiivisesti. (W3Schools 2014.)

JavaScriptilla saadaan lisättyä dynaamisia toimintoja WWW-sivuille. Esimerkiksi lomakkeen tietojen esitarkastaminen ja erilaiset ponnahdusikkunat voidaan toteuttaa JavaScriptin avulla. Kielelle on tarkoituksella asetettu tietoturvaan liittyviä rajoituksia yksityisyyden suojaamiseksi. Selaimen ominaisuuksien asettaminen ja noutaminen, tulostaminen, tietokoneelle asennetun sovelluksen käynnistäminen, tiedostojen ja kansioiden lukeminen ja kirjoittaminen (poikkeuksena evästeet), palvelimelle kirjoittaminen ja sähköpostien lähettäminen taustaprosessina ovat yleensä rajoitettuna, tai estettyinä kokonaan. (Goodman, Morrison, Novitski & Rayl 2010.)

#### 4.4.1 jQuery

jQuery on todella suosittu, kaikille selaimille tarkoitettu ilmainen MIT-lisensoitu avoimen lähdekoodin JavaScript-kirjasto (The jQuery Foundation 2014). WordPress tarjoaa jQuery-kirjaston sulautettuna osaksi järjestelmää, yhdessä useiden muiden skriptien ja kirjastojen kanssa.

### 4.4.2 AJAX

AJAX on akronyymi sanoista Asynchronous JavaScript And XML. Se on joukko verkkosovelluskehityksen tekniikoita, joiden avulla verkkosovelluksista voidaan tehdä vuorovaikutteisempia.

Alkuperäiseltä merkitykseltään AJAX on tekniikkaa, jossa verkkosivulla asynkronisesti tehtävistä HTTP-pyynnöistä palautetaan XML-merkkausta. Nykyään käytetään kuitenkin erittäin laajasti selkeämpää JSON-merkkausta. Käytännössä AJAX on siis usean vanhemman tekniikan yhteiskäyttöä, ja sitä voidaan käyttää monella tapaa.

AJAX eroaa perinteisistä WWW-sovelluksista siten, että tekniikkaa käyttävä sovellus lähettää ja vastaanottaa tietoa osissa, jolloin koko sivustoa ei ladata uudelleen. AJAX:ia käyttämättömissä sovelluksissa on totuttu siihen, että sivuston sisällön muuttaminen vaatii sen uudelleen lataamisen palvelimelta. Tämä ei ole useinkaan toivottua, varsinkin, jos muutokset koskevat vain pientä osaa sivusta. AJAX-tekniikka mahdollistaa tiedon siirtämisen selaimen ja palvelimen välillä ilman, että käyttäjä huomaa koko tapahtumaa (ellei käyttäjän haluta näkevän tapahtumaa). Tieto siirtyy siis taustalla asynkronisesti häiritsemättä muuta sivuston toimintaa.

## 4.5 WordPress API

WordPressin rajapinta (The WordPress Application Programming Interface) on jaettu osioihin (ns. alirajapintoihin), joita voidaan tarkastella myös omina rajapintoinaan sen toimintojen tai aiheen mukaan. Nämä pienemmät alirajapinnat mahdollistavat muiden ulkopuolisten ohjelmien keskustelun WordPress-ytimen kanssa. Rajapinnan toimintoja selvennettään tarkemmin omassa luvussaan (luku 6).

# 5 JSON-LD, LINKITETTY DATA JA SEMANTTINEN VERKKO

## 5.1 Mikä on JSON-LD?

JSON-LD (JavaScript Object Notation for Linked Data) on vaihtoehtoinen standardi RDF-tiedon esittämiseen. JSON-LD rakentuu suositun tiedonesitysmuodon, JSON:in päälle. Nimistöjen sijasta JSON-LD:ssä URI:en lyhentämiseen käytetään avainsanaa *@contex*, määriteltävä ja identifoitava kohde merkitään avainsanalla *@id*, ja avainsana *@type* määrittelee tiedon tyypin. (W3C 2014.) Muitakin avainsanoja (keywords) on käytössä, mutta esittelen opinnäytetyössäni vain lisäosien toteutuksen kannalta oleellisemmat, sillä JSON-LD olisi liian laaja osa-alue käsitellä kattavasti tämän opinnäytetyön yhteydessä.

JSON-LD tarjoaa JSON-pohjaisen linkitetyn datan standardin. Tämä tarjoaa helpon tavan parantaa nykyisten JSON-sovellusten semanttista yhteensopivuutta. Tarkemmin selitettynä JSON-LD laajentaa JSON:ia Linked Data -tekniikoihin, missä JSON-tietorakenteet voivat sisältää kontekstin (context), joka yksikäsitteisesti tunnistetaan IRI-tunnisteilla (Internationalized Resource Identifier). Tämä sallii JSON-LD:een tehdä kuvauksia yksikäsitteisistä objekteista ja myös viitata muihin objekteihin, mikä ei pelkällä JSON:lla ollut mahdollista. (Profium 2014.) IRI-tunnisteet ovat URI-tunnisteiden (Uniform Resource Identifier) laajennoksia, jolla kerrotaan tietyn tiedon paikka (URL, Uniform Resource Locator) tai yksikäsitteinen nimi (URN, Uniform Resource Name).

Hyvänä esimerkkinä voidaan pitää toteutettujen lisäosien rajapinnan syötteen määrittelyjä, sekä näiden määrittelyjen osittaista käyttöönottoa. Esimerkkejä on karsittu, sillä varsinainen prototyypeissä käytetty JSON-LD-syöte on todella laaja. Syöte perustuu pitkälti Schema.org:in määrittelyihin, mutta ei noudata sitä vielä täydellisesti. Esimerkkikoodissa 1 esitellään näyte tapahtumatyyppisestä toteutuksessa käytetystä JSON-LD-syötteestä. Esimerkkikoodissa 2 esitellään Schema.org:in määrittelyihin perustuva yksinkertainen tapahtumatyyppinen JSON-LD-syöte.

```
{
  "@context": "http://schema.org",
  "@type": "Event",
  "description": "Futismatsi kaikille.",
  "endDate": "2014-11-30 23:59:00",
  "eventStatus": "http://schema.org/EventScheduled",
  "location": {
    "@type": "Place",
    "address": "Ratinan stadion",
    "geo": {
      "@type": "GeoCoordinates",
      "latitude": "61.49320789999999",
      "longitude": "23.763920999999982"
    },
    "name": "Ratinan futiskenttä"
  },
  "name": "Jalkapalloa",
  "offers": {
    "@type": "Offer",
    "price": "20"
  },
  "organizer": {
    "@type": "Organization",
    "address": "Kuntokatu 3, Tampere",
    "email": "esimerkki@maili.fi",
    "name": "TAMKin opiskelijat",
    "telephone": "040 1111 222",
    "url": "http://esimerkki.fi"
  },
  "sameAs": "http://example.com",
  "startDate": "2014-11-30 22:00:00",
  "url": "http://localhost/event-worker-host/events/263/",
  "version": "event_worker_1417346896",
  "workPerformed": {
    "@type": "CreativeWork",
    "dateModified": "2014-11-30 14:42:03",
    "keywords": "Urheilu"
  }
}
```

ESIMERKKIKOODI 1. JSON-LD esimerkki isäntälisäosan prototyypin rajapinnan syötteestä

```
{
  "@context" : "http://schema.org",
  "@type" : "Event",
  "name" : "Autechre",
  "startDate" : "2014-07-06",
  "location" : {
    "@type" : "Place",
    "name" : "Ratinan stadion"
  }
}
```

ESIMERKKIKOODI 2. Yleinen JSON-LD esimerkki

## 5.2 Linkitetty data ja semanttinen verkko

Linkitetty data (Linked Data) on nykyään tavallinen käytäntö eri aiheita käsittelevien tietojen esittämiseen. Linkitetyn datan julkaisemisen ansiosta sovelluskehittäjät voivat helpommin yhdistää eri lähteistä saatuja tietoja. (Euroopan unionin avoimen datan portaali 2014.)

Linkitetty data muodostaa semanttisen verkon, joka käsitemallien avulla yhdistää tietoa koneluettavaan muotoon (Ruotsalainen 2010, 26-27). Esimerkiksi internetin kehittäjä Tim Berners-Lee kertoi vuonna 2004 puheessaan Otaniemen Dipolissa tulevaisuuden visiostaan, miten semanttisen verkon avulla tietojenkäsittely helpottuu tulevaisuudessa, koska semanttinen verkko mahdollistaa eri lähteiden sisältämän tiedon keräämisen ja siirtämisen automaattisesti laitteesta tai sovelluksesta toiseen (Karvonen 2004). Semanttinen verkko ei siis ole mikään uusi ajatus.

## 5.3 Schema.org

Schema.org on vuonna 2011 kolmen suurimman hakukoneoperaattorin, Googlen, Yahoon ja Bingin julkaisema aloite yhdenmukaisen jäsennellyn merkintätiedon esittämiseen verkossa (Google Official Blog 2011; Bing Blogs 2011; Yahoo! Search

Blog 2011). Myös Venäjän suurin hakukoneoperaattori Yandex liittyi aloitteeseen vielä samana vuonna (Schema blog 2011).

Prototyyppien toteutuksessa käytettiin Schema.org:in JSON-LD määrittelyjä siltä osin kuin niiden kohdalla oli mahdollista. Yhteinen merkintätapa helpottaa tiedon rakenteen esittämisen valitaan liittyviä ongelmia, ja lisäksi suurimmat hakukoneet ymmärtävät standardin mukaista yhtenäistä merkintätapaa (Schema FAQ 2014).

# 6 WORDPRESS-LISÄOSIEN SUUNNITTELU JA TOTEUTUS: CASE LIKIOMA

## 6.1 Yleistä tietoa lisäosista

WordPressin suhteellisen helppo laajennettavuus sekä monipuolisuus perustuu pitkälti lisäosien käyttöön (WordPress Codex 2014b). Julkaistuja lisäosia löytyy virallisen sivuston lisäosavarastosta tällä hetkellä 34,785 (WordPress 2014d). Lisäosia löytyy myös epävirallisia kanavia pitkin, kuten esimerkiksi GitHubista. Aina tarvittavaa lisäosaa ei kuitenkaan ole valmiina tarjolla niiden suuresta määrästä huolimatta, ja tällöin se onkin ohjelmoitava itse tai tilattava ulkopuoliselta taholta.

## 6.2 Lisäosien suunnittelu ja toimintatapa

Lisäosaa suunnitellessa tulee ottaa huomioon sen käyttötarkoitus, elinkaari sekä alustan vaatimukset ja rajoitukset. Ennen varsinaista suunnitelua on hyvä tarkistaa ainakin seuraavat asiat:

- Onko vastaava lisäosa jo kehitetty?
- Onko lisäosa tarkoitettu laajempaan käyttöön, vai esimerkiksi pelkästään testikäyttöön?
- Onko lisäosalla tarkoitus ansaita tuloja, onko se puhtaasti markkinointitarkoituksessa tehty, vai onko lisäosan tarkoitus edistää ja kehittää avointa WordPress-yhteisöä?
- Lisäosan ylläpito, tuki sekä jatkokehitys vie aikaa. Kun lisäosa kehitetään, on sitä myös ylläpidettävä, tai sen elinkaari jää lyhyeksi. On hyvä miettiä tarkoin, onko ajan käyttäminen tuen ja ylläpidon tarjoamiseen jatkossa mahdollista ja kannattavaa?

Ylläoleva lista auttaa hahmottamaan lisäosan kehityksen vaativuuden. Voidaan sanoa, että lista määrittää, kuinka vakavasti ja huolellisesti kehittäjän on otettava huomioon koodausstandardit, turvallisuus, ylläpito, tuki ja lisäosan dokumentaatio. (Elliot 2014.)

### 6.2.1 Koodausstandardit

Selkeät ja johdonmukaiset koodausstandardit on syytä ottaa huomioon lisäosien toteutuksessa. Pääasiassa koodausstandardit noudattavat normaaleja hyväksi todettuja yleisiä ohjelmointikäytäntöjä. WordPress tarjoaa kuitenkin oman ohjeistuksensa PHP-, HTML-, CSS- ja JavaScript-standardeihin. Koodausstandardien noudattaminen tekee lähdekoodista ymmärrettävämmän ja selkeämmän.

Lisäosa kannattaa rakentaa ja jakaa luokkiin (jos lisäosa on pieni, niin usein yksi luokka riittää). Luokkia käytettäessä funktiot sijaitsevat luokan omassa nimiavaruudessa. Tämä on suositeltavaa, sillä muuten funktiot olisivat globaalissa nimiavaruudessa. Voidaan siis todeta, että on kaksi pääasiallista tapaa rakentaa lisäosa, jos monimutkaisempia suunnittelu- tai ohjelmointimalleja ei käytetä. (WordPress Codex 2014c.)

Koska lisäosat toteutettiin suurelta osin oliomallin mukaisesti eikä proseduaalisesti (esimerkkikoodi 3), esittelen jatkossa koodiesimerkit kuten niitä luokkien sisällä käytetään (esimerkkikoodi 4). Pseudomuuttujalla *$this* viitataan luokan tai olion sisäiseen kontekstiin, esimerkiksi funktioon tai muuttujaan. Oliomallin käyttäminen vaikuttaa myös WordPressin rajapinnan koukkujen (Hooks) käyttötapaan.

```
add_action('init', 'worker_plugin_init');
```

ESIMERKKIKOODI 3. Lisäosan käyttöönotto toiminnallisella koukulla proseduaalisesti

```
add_action('init', array($this, 'worker_plugin_init'));
```

ESIMERKKIKOODI 4. Lisäosan käyttöönotto toiminnallisella koukulla oliomallin mukaisesti

Lisäosan käyttöönoton yhteydessä tarvittavaa *init*-koukkua kutsutaan luokan rakentajan sisällä, ja myös muuta tarvittavaa sisältöä voidaan ladata samalla tavalla. Esimerkkikoodissa 5 rakentajassa ladataan tarvittava PHP-tiedosto, joka sisältää luokan lyhytkoodien (shortcode) suorittamiseen.

```
class WorkerHostCore {

    function __construct() {
        require_once('shortcodes/submit-event.php');
        add_action('init', array($this, 'worker_plugin_init'));
    }

    function 'worker_plugin_init() {
        ...
    }
}

new WorkerHostCore();
```

ESIMERKKIKOODI 5. Luokan rakenne ja rakentajan käyttö

### 6.2.2 Sisältötyypit ja taksonomiat

WordPress tarjoaa oletuksena viisi erilaista sisältötyyppiä (Default Post Types). Näitä ovat artikkelit, sivut, liitteet, versiot (esim. luonnokset) ja valikot. Näiden lisäksi sisältötyyppejä on mahdollista luoda itse. *Action*, *order* ja *theme* ovat kuitenkin yllämainittujen perussisältötyyppien lisäksi varattuja sanoja sisältötyyppien nimiksi, sillä ne ovat jo käytössä muualla WordPressissä ja voivat aiheuttaa sekaannuksia.

Räätälöidyt sisältötyypit (Custom Post Types) mahdollistavat WordPressin perussisältötyyppien kaltaisten uusien sisältötyyppien määrittelyn. Räätälöidyt sisältötyypit käyttäytyvät kuten sivut, eli ne voivat muodostaa hierarkioita, tai kuten artikkelit, eli ne voivat muodostaa sisältövirran.

Omat taksonomiat ovat itse luotuja luokitusjärjestelmiä, jotka toimivat joko vapaamuotoisesti avainsanojen, tai hierarkisesti kategorioiden mukaisesti. Niiden luominen ja käyttäminen on suhteellisen yksinkertaista. Taksonomia voi liittyä yhteen tai useampaan eri sisältötyyppiin.

### 6.2.3 Koukut (actions & filters)

Melkein mihin tahansa tarvittavaan toiminnallisuuteen pääsee käsiksi WordPressin koukkujen kautta. Koukkuja on valtavasti ja niiden määrä kasvaa jatkuvasti. Omia funktioita voidaan liittää tiettyyn koukkuun, jolloin se ajetaan aina kun WordPress suorittaa kyseisen koukun. Koukkuja on kahta eri tyyppiä: toiminnalliset koukut (action) ja suodattimet (filter). Usein haluttu toiminnallisuus voidaan saavuttaa kumpaa tahansa käyttämällä.

Toiminnallisia koukkuja käytetään lisäosissa ja teemojen funktioissa, ja WordPress osaa käynnistää niihin määritellyt funktiot oikean tapahtuman aikana. Toiminnallisilla koukuilla voidaan muokata esimerkiksi tietokantaan siirtyvää tietoa. Koukku (esimerkkikoodi 6) ei tarvitse pakollisina tietoina sisälleen kuin tiedot siitä, missä tapahtumassa funktio suoritetaan, ja mikä funktio suoritetaan (WordPress Codex 2014d).

```
add_action('publish_post', array($this, 'my_custom_function'));
```

ESIMERKKIKOODI 6. Toiminnallisen koukun käyttö sisällön muokkaamiseen

Suodattimet (filters) ovat funktioita, jotka kaappaavat tietokannoista siirtyvän datan ja manipuloivat sitä määrätyllä tavalla ja hetkellä. Suodattimia käytetään samaan tapaan kuin toiminnallisia koukkujakin (esimerkkikoodi 7).

```
add_filter('the_content', array($this, 'my_custom_function'));
```

ESIMERKKIKOODI 7. Suodattimen käyttö sisällön muokkaamiseen

### 6.2.4 Tietoturva

Turvallisuus on yksi tärkeimmistä huomioon otettavista osa-alueista lisäosakehityksessä. WordPress tarjoaa laajan varaston funktioita tietoturvaan liittyen. Lisäosien tietoturva voidaan jakaa karkeasti viiteen osaan:

- syötteen tarkistaminen (Input)
- tulosteen tarkistaminen (Output)
- tietojen kelpoisuuden tarkistaminen (Data Validation)
- käyttäjän oikeuksien tarkistaminen (User Capabilities)
- kertakäyttöavaimen tarkistaminen (Nonce)

Yhtenä perussääntönä voidaan pitää seuraavaa: *Sanitize inputs, espace outputs*. Eli kaikki käyttäjältä tullut tieto puhdistetaan tietoturvaa heikentävistä merkeistä, ja vastaavasti käyttäjälle näkyvä tuloste siivotaan riskitekijöistä. (WordPress Codex 2014e.)

Kriittinen seikka turvallisuudessa on, kun käyttäjä syöttää tietoa esimerkiksi lomakkeen tai sivun osoitteen kautta. Käyttäjän antama tieto voi olla käytännössä mitä tahansa, minkä vuoksi siihen tulee suhtautua hyvin epäluuloisesti. Tietoturvassa oletuksena on, että käyttäjä ei ole ystävä. Tästä syystä tietojen kelpoisuus on tarkistettava. Erittäin yksinkertaisena esimerkkinä voidaan pitää tarkistusta, jossa katsotaan, onko tapahtuman hinnaksi syötetty merkki numero, tavuviiva tai piste (esimerkkikoodi 8). Muita merkkejä ei hyväksytä.

```
function isNumberKey(evt) {
    var charCode = (evt.which) ? evt.which : event.keyCode
    if (charCode > 31 && (charCode < 45 || charCode > 57)) {
        return false;
    }
    return true;
}
```

ESIMERKKIKOODI 8. JavaScript-funktio, joka tarkistaa, onko syötetty merkki numero, tavuviiva vai piste.

Lisäksi PHP-koodissa tarkistetaan että valmiin syötteen ensimmäinen merkki ei ole tavuviiva tai piste, sillä hinta voidaan esittää kahden eri arvon väliltä tai desimaalina (esimerkkikoodi 9).

```
if ($event_price[0] != '-' || $event_price[0] != '.') {
    update_post_meta($post_id,
                     'event_price',
                     sanitize_text_field($event_price));
}
```

ESIMERKKIKOODI 9. Tarkistetaan, onko syötteen ensimmäinen merkki sallittu

*Esc_attr*-funktiota käytetään, kun tieto lopulta tulostetaan käyttäjälle (esimerkkikoodi 10). Lisäksi WordPress tarjoaa myös useita muita funktioita *escape*-tarkistusta varten.

```
echo '<input type="text" class="eventprice" id="AdminEventPrice"
       name="AdminEventPrice" onkeypress="return isNumberKey(event)"
       value="' . esc_attr($price) . '"/><br/>';
```

ESIMERKKIKOODI 10. Tulostetaan hinta käyttäjälle tarkistaen

Yllä olevien esimerkkien lisäksi on tiedettävä, että käyttäjä on luotettava, ja että tieto todella tulee käyttäjältä. Tätä tarkoitusta varten WordPress tarjoaa käyttäjän oikeuksien tarkistamisen (User Cababilities) sekä kertakäyttöavaimet (Nonce). Näillä ehkäistään esimerkiksi CSRF-haavoittuvuutta (Cross-Site Request Forgery), eli *Cross-Site*-pyynnön väärentämistä. CSRF:ssä hyökkääjä hyödyntää sivuston luottamusta sivua selaavan käyttäjän selainohjelmaan. Selain lähettää palvelimelle pyynnön, jonka palvelin luulee tulleen omilta autentikoiduilta käyttäjiltään. (WordPress Codex 2014f.)

*Nonce (number used once)*, on kertakäyttöavain, jota käytetään nimensä mukaisesti kerran lähetetyn tiedon varmistamiseen. Kertakäyttöavain varmistaa aina koodia suoritettaessa, että tieto todella tulee tietyltä käyttäjältä. Esimerkkikoodissa 11 esitellään avaimen käyttö lisäosan ylläpitopaneelissa. Käyttötapoja on kuitenkin useita muitakin.

```
$action = 'event_worker_action_xyz_' . get_the_ID();
$name = 'event_worker_price_nonce';
wp_nonce_field($action, $name);
```

ESIMERKKIKOODI 11. Lisätään kertakäyttöavain tapahtuman hintakenttään (metabox)

Koska hintakenttä sijaitsee ylläpitopaneelissa, käytetään funktiota *check_ajax_referer()* avaimen tarkistamiseen (esimerkkikoodi 12). Lisättäessä tapahtumia julkisen, mutta vain sisäänkirjautuneille näkyvän lisäyssivun kautta, tarkistaminen tapahtuu hieman eri tavalla. Tarkastaminen tapahtuu lisäosissa tallentamisen yhteydessä.

```
check_ajax_referer('event_worker_action_xyz_' . $post_id,
                   'event_worker_price_nonce');
```

ESIMERKKIKOODI 12. Tarkistetaan kertakäyttöavain ylläpitopaneelissa tietoja tallentaessa

Avain voi näkyä lähdekoodissa esimerkiksi seuraavan esimerkkikoodin mukaisesti (esimerkkikoodi 13).

```
<input type="hidden" id="event_worker_price_nonce"
 name="event_worker_price_nonce" value="789ed2a192">

<input type="hidden" name="_wp_http_referer"
 value="/event-worker-host/wp-admin/post-new.php?post_type=events">
```

ESIMERKKIKOODI 13. Kertakäyttöavaimen näkyminen ylläpitopaneelin lähdekoodissa

Lisäksi kannattaa ottaa huomioon normaalit käytännöt verkko-ohjelmoinnissa. Esimerkiksi tietokantojen tietoturva kannattaa huomioida erittäin huolellisesti, vaikka WordPress tarjoaakin omat suojansa tietokantojen käyttöön.

### 6.2.5 Asetukset

WordPressin *Settings API* mahdollistaa asetussivujen luomisen ylläpitopaneeliin (WordPress Codex 2014g). Asetussivujen rakenne voi olla esimerkiksi seuraavaksi esiteltävien yleisten esimerkkien kaltainen. Rakentajassa kutsutaan tarvittavia koukkuja, joista ensimmäinen lisää asetussivun ylläpitopaneeliin. Toinen koukuista rekisteröi asetukset ylläpitopaneelin aktivoinnin yhteydessä (esimerkkikoodi 14).

```
add_action('admin_menu', array($this, 'my_admin_menu'));
add_action('admin_init', array($this, 'my_admin_init'));
```

ESIMERKKIKOODI 14. Asetussivun rakentaja

Funktio *my_admin_menu()*, jota kutsutaan luokan rakentajassa *admin_menu*-koukun avulla, lisää asetussivun ylläpitopaneelin valikkoon *add_options_page()*-funktion avulla (esimerkkikoodi 15). Ensimmäinen parametri funktiossa on asetussivun nimi, joka näytetään selaimen otsikkopalkissa. Toinen parametri on sivun otsikko ylläpitovalikossa. Kolmas parametri huolehtii siitä, kenellä on käyttöoikeus asetuksiin. Neljäs parametri on asetussivun *slug*, eli lyhytlinkki. Viides parametri on varsinaisen funktion nimi, joka näyttää asetussivun.

```
function my_admin_menu() {
    add_options_page('My Plugin', 'My Plugin', 'manage_options',
                     'my-plugin', 'my_options_page');
}
```

ESIMERKKIKOODI 15. Asetussivun linkin luonti ylläpitopaneeliin

Funktio *my_admin_init()*, jota kutsutaan rakentajassa *admin_init*-koukun avulla, rekisteröi asetukset, lisää erilliset osiot asetussivulla sekä kentät sivun osioihin (esimerkkikoodi 16).

```
function my_admin_init() {
    register_setting('my-settings-group', 'my-setting');
    add_settings_section('section-one', 'Section One',
                         'section_one_callback', 'my-plugin');
    add_settings_field('field-one', 'Field One', 'field_one_callback',
                       'my-plugin', 'section-one');
}
```

ESIMERKKIKOODI 16. Asetusten rekisteröinti

*Callback*-funktiot (takaisinkutsut) tulostavat tiedot asetussivun näkymään (esimerkkikoodi 17).

```
function section_one_callback() {
    echo 'Some help text goes here.';
}

function field_one_callback() {
    $setting = esc_attr(get_option('my-setting'));
    echo "<input type='text' name='my-setting' value='$setting' />";
}
```

ESIMERKKIKOODI 17. Tulostusfunktiot

Lopuksi luodaan funktio, joka yhdistää ja esittää yllä olevat asetukset käyttäjille (esimerkkikoodi 18).

```
function my_options_page() {
    ?>
    <div class="wrap">
        <h2>My Plugin Options</h2>
        <form action="options.php" method="POST">
            <?php settings_fields('my-settings-group'); ?>
            <?php do_settings_sections('my-plugin'); ?>
            <?php submit_button(); ?>
        </form>
    </div>
    <?php
}
```

ESIMERKKIKOODI 18. Asetussivun näyttäminen käyttäjälle

### 6.2.6 HTTP-pyynnöt

HTTP-pyynnöt kannattaa toteuttaa WordPressin tarjoamaa *HTTP API* -alirajapintaa käyttäen. Rajapinta valitsee automaattisesti tehokkaimman ja luotettavimman PHP:n tarjoamista vaihtoehdoista tietojen hakemiseen. Asiakaslisäosan tapahtumien hakemiseen käytetään kyseistä rajapintaa. Tapahtumat haetaan *wp_remote_get()*-funktiota käyttäen (esimerkkikoodi 19). Rajapinta tarjoaa myös muita funktioita (esimerkiksi *POST* ja *HEAD*) erilaisiin kutsuihin. (WordPress Codex 2014h).

```
$output = wp_remote_get($url);
```

ESIMERKKIKOODI 19. Tapahtumat haetaan asiakaslisäosan *$output*-muuttujaan palvelinlisäosasta

### 6.2.7 Skripti- ja tyylimäärittelytiedostojen lataaminen (JavaScript ja CSS)

JavaScript- ja CSS-tiedostoja ei koskaan tule ladata suoraan, ns. normaalilla PHP-ohjelmointitavalla, lisäosia kehitettäessä. Sen sijaan WordPress käyttää omia funktioitaan, jotka ovat *wp_enque_script()* ja *wp_enque_style()*. Funktiot voidaan ottaa

käyttöön toiminnallista koukkua *wp_enque_scripts* käyttäen, ja ne esitellään esimerkkikoodissa 20.

WordPress-rajapinnan tarjoamien funktioiden ja koukkujen käyttäminen ehkäisee mahdolliset duplikaattiongelmat, ja lisäksi ne antavat paremman hallinnan ladatun sisällön suhteen. Skripti- ja tyylitiedostoja ladatessa kannattaa kiinnittää huomiota siihen, missä osioissa niitä lisäosassa ja sivustolla käytetään, ja ladata ne käyttöön vain tarvittaessa. (WordPress Codex 2014i; WordPress Codex 2014j.)

```
function plugin_scripts() {
    wp_enqueue_style('style-name',
                     plugins_url('../css/style.css', __FILE__));

    wp_enqueue_script('maphandler',
                      plugins_url('../js/maphandler.js', __FILE__),
                                  array(), '1', false);
}
add_action('wp_enqueue_scripts', array($this, 'plugin_scripts'));
```

ESIMERKKIKOODI 20. Tyyli- ja skriptitiedoston lataus

Ylläpitopaneeliin skriptit on ladattava omaa erityistä (toiminnallista) *admin*-koukkua käyttäen (esimerkkikoodi 21).

```
add_action('admin_enqueue_scripts',
           array($this, 'load_custom_wp_admin_style'));
```

ESIMERKKIKOODI 21. Skriptin lataus ylläpitopaneeliissa

Jos halutaan, että tieto kulkee skripteihin palvelinpuolelta ja päinvastoin, ja halutaan kääntää myös skritpeissä käytettävät tekstit muille kielille, on hyvä käyttää *wp_localize_script()*-funktiota joka esitellään esimerkkikoodeissa 22 ja 23.

```
$vars = "value";
wp_enqueue_script('maphandler', plugins_url('../js/maphandler.js',
                  __FILE__), array(), '1', false);

wp_localize_script('maphandler', 'php_vars', $vars);
```

ESIMERKKIKOODI 22. Skriptin lokalisointi

```
console.log(php_vars.variable);
```

ESIMERKKIKOODI 23. Kutsutaan muuttujaa JavaScript-puolella

### 6.2.8 AJAX lisäosissa

AJAX:in käyttötapa lisäosissa on riippuvainen siitä, käytetäänkö sitä ylläpitopuolella (backend) vai itse sivustolla (frontend). On myös hyvä huomioida, milloin AJAX:ia käytetään, ettei sitä ladata turhaan jokaiselle sivuston osiolle. Yleiset esimerkit molemmista esitellään liitteissä 1 ja 2. *Frontend*-osioon pitää määrittää, halutaanko toiminnot vain sisäänkirjautuneelle käyttäjälle (wp_ajax_my_action), vai myös muille (wp_ajax_nopriv_my_action). Käyttötarkoitukseen sopivat toiminnalliset koukut esitellään esimerkkikoodissa 24.

```
add_action('wp_ajax_my_action',
           array($this, 'my_action_callback'));
add_action('wp_ajax_nopriv_my_action',
           array($this, 'my_action_callback'));
```

ESIMERKKIKOODI 24. Tyyli- ja skriptitiedoston lataus

Lisäksi sivustolle on lisättävä mahdollisuus käyttää AJAX:ia, sillä oletuksena se on sisällettynä vain *backend*-puolelle (esimerkkikoodi 25).

```
wp_localize_script('script_handle',
                   'MyAjax',
                   array('ajaxurl' => admin_url('admin-ajax.php')));
```

ESIMERKKIKOODI 25. Sallitaan AJAX *frontend*-puolella

### 6.2.9 Kansainvälistäminen ja lokalisointi

Lisäosat on hyvä tehdä kääntövalmiiksi heti alusta asti, sillä näin säästytään suurelta työltä myöhemmin. Usein prosessi jaetaan kahteen osaan, jotka ovat kansainvälistäminen (i18n) ja lokalisointi (l10n). Ensimmäinen termi tarkoittaa lisäosan tekemistä käännösvalmiiksi, kun taas seuraava itse kääntöprosessia sovelluksiin. Sovelluksen kehittäjä merkitsee käännettävät sanat ja lauseet lähdekoodiin (i18n), ja tämän jälkeen ne voidaan kääntää sovellus- ja kielikohtaiseen sanastotiedostoon (l10n).

WordPress käyttää sisällön lokalisointiin omia funktiotaan *__()* ja *_e()*, jotka pohjautuvat GNU gettext -kirjastoon (esimerkkikoodi 26). Nämä funktiot toimivat siten, että kun jokin teksti halutaan valmistella myöhemmin käännettäväksi, kirjoitetaan se funktion sisään. Funkioiden ero on se, että *__()*-funktion hakee tekstit PHP:n *return*-lauseella ja sitä käytetään, kun palautetaan muuttuja toiseen funktioon. *_e()*-funktiolla puolestaan tulostetaan *echo*-lauseella tietoa suoraan näkymään. Lisäksi funktioon määritetään käännösten *text domain*, joka määrittää mistä ja mitkä käännökset haetaan käyttöön. Tämä tehdään *load_plugin_textdomain()*-funktiota käyttäen (esimerkkikoodi 27). (WordPress Codex 2014k.)

```
submit_button(__('Save Changes', 'event-worker-translations'));
```

ESIMERKKIKOODI 26. Lähetä-painikkeen teksti luodaan kääntövalmiiksi

```
load_plugin_textdomain('event-worker-translations', FALSE,
                       dirname(plugin_basename(__FILE__)) .
                       '/lib/languages/');
```

ESIMERKKIKOODI 27. Ladataan *text domain* lisäosan käytettäväksi

### 6.2.10 Muisti- ja tietojälki

Lisäosat lisäävät usein omia kenttiä, *Rewrite*-sääntöjä sekä asetuksia tietokantaan. Tämä voi tapahtua ilman, että käyttäjää informoidaan millään tavalla. Lisäosan suunnitellussa ja toteutuksessa on otettava huomioon mitä näille tiedoille tehdään, jos lisäosa esimerkiksi poistetaan kokonaan tai otetaan väliaikaisesti pois käytöstä.

WordPress tarjoaa aktivointi- ja deaktivointikoukun lisäosan käyttöön ottamista sekä käytöstä poistamista varten. Aktivoinnin yhteydessä voidaan esimerkiksi luoda lisäosan tarvimat *Rewrite*-säännöt, ja deaktivoinnin yhteydessä nämä säännöt voidaan puhdistaa (Flush). (WordPress Plugin Development Handbook 2014a.)

Lisäksi, jos lisäosa poistetaan kokonaan, on suositeltua käyttää lisäosan poistamiseen tarkoitettuja metodeja, joita ovat *register_uninstall_hook*-koukun tai *unistall.php*-tiedoston käyttäminen, joista jälkimmäinen on suositeltu ja yleensä turvallisempi vaihtoehto. *Register_uninstall_hook*-koukkua käytettäessä lisäosan päätiedoston koodi ajetaan, ja tämä voi aiheuttaa ongelmia, varsinkin jos jotkin lisäosan toiminnot eivät ole funktioiden tai luokkien sisässä. (WordPress Plugin Development Handbook 2014b.)

Lisäosien prototyyppien kohdalla niiden luomia kenttiä tietokannasta, *Rewrite*-sääntöjä, tai asetuksia ei poisteta, sillä niiden tarjoama informaatio voi olla tärkeää jatkokehityksen kannalta.

## 6.3 Lisäosien toteutus

Härmälän alueen tapahtumienhallintaan toteutettiin kaksi erillistä WordPress-lisäosaa, *Host* (isäntä) ja *Client* (asiakas). Lisäosat käyttävät osittain samoja komponentteja.

Hallittavuuden ja jatkokehityksen helpottamisen kannalta erillinen asiakasohjelma tarvitaan tapahtumien esittämiseen. Tätä tarkoitusta varten toteutettiin prototyyppi asiakaslisäosasta. Molemmat lisäosat voidaan kuitenkin asentaa myöskin samaan WordPress-asennukseen.

Lisäosat ovat avointa lähdekoodia, joten kuka tahansa voi hyödyntää niitä ja jatkaa halutessaan kehitystä. Lisäosat ja niiden dokumentaatio löytyvät osoitteesta https://github.com/event-worker/. Verkkoliitteeseen on päädytty, sillä dokumentaatio on jatkuvasti kehittyvä ja myös liian laaja kuvattavaksi tämän opinnäytetyön yhteydessä.

### 6.3.1 Toiminta tiivistetysti

*Host*-lisäosan, eli isäntäosan avulla voidaan luoda ja hallita tapahtumia. *Client*-lisäosa mahdollistaa tapahtumien esittämisen aikajärjestyksessä keskitetystä lähteestä (isäntälisäosan rajapinta) käyttäjän WordPress-asennuksessa. Rajapinnan tieto on avointa, joten uusia asiakasohjelmia voidaan haluttaessa luoda helposti (myös muille alustoille).

Käyttäjällä on mahdollisuus tulostaa tapahtumat PDF-tiedostona, jonka lisäosa luo. Lisäosa luo tapahtumista myös Plain Text -formaatissa olevan tiedoston (tekstitiedoston), joka mahdollistaa tekstin helpon kopioimisen ja liittämisen.

Tapahtumat poistuvat lisäosista, eli rajapinnan syötteestä ja näkymästä (luonnos-tilaan) kun niiden loppumispäivämäärä ohittuu. Luonnos-tilassa olevat tapahtumat näkyvät vain ylläpitopaneelissa. Tapahtumia on mahdollista myös perua, jolloin ne siirtyvät peruttu-tilaan (cancelled).

### 6.3.2 Asennus ja aktivointi

Lisäosat asennetaan WordPress-asennuksen juuressa sijaitsevaan kansioon *wp-content/plugins/*. Jos käyttäjällä on siihen tarvittavat oikeudet, voidaan lisäosat aktivoida Lisäosat-valikosta WordPressin ylläpitopaneelista.

### 6.3.3 Yhteiset komponentit

Molemmissa lisäosissa käytetään osittain samoja käytäntöjä, luokkia sekä ulkopuolisia kirjastoja, sillä lisäosia voidaan käyttää joko yhdessä tai erikseen. Komponentit eivät ole kuitenkaan ristiriidassa keskenään jos lisäosat asennetaan samaan WordPress-asennukseen, sillä voidaan tarkistaa onko kyseinen luokka jo käytössä. On mahdollista myös tarkistaa, onko tietty lisäosa jo käytössä, ja hallita komponenttien lataamista tätä kautta.

**Tapahtumat sisältötyyppinä**

Tapahtumat perustuvat oman räätälöidyn sisältötyypin käyttämiseen. Lisäosien käyttöönoton aikana luodaan sisältötyyppi *events*, sekä asetetaan sisältötyypille sen vaatimat asetukset (esimerkkikoodi 28). Asetuksia on runsaasti, ja omiin tarpeisiin juuri oikeiden asetuksien löytäminen voi olla haastavaa. Räätälöity sisältötyyppi vaikuttaa kuitenkin lähes kaikkiin lisäosan toimintoihin, sillä kaikki tapahtumat luodaan jatkossa tämän tyyppisiksi.

Myös asiakaslisäosa käyttää sisältötyyppiä, sillä sen on pystyttävä käsittelemään haettavia tapahtumia. Sisältötyypille luodaan myös omat kategoriansa, joita voidaan myöhemmin lisätä, muokata sekä poistaa ylläpitopaanelin kautta.

```
function __construct() {
    add_action('init', array($this, 'create_post_type'));
}

function create_post_type() {
    register_post_type( events',
          array(
              'labels' => array(
                  'name' => __( 'Events' ),
                  'singular_name' => __( 'Event' )
              ),
             'public' => true,
             'has_archive' => true,
          )
    );
}
```

ESIMERKKIKOODI 28. Yksinkertainen esimerkki oman sisältötyypin luomisesta

**Tapahtumien kategoriat (custom taxonomies)**

Tapahtumille luotiin oma taksonomia niiden kategorioita varten. Taksonomia on rekisteröity *events*-sisältötyypille. Esimerkkikoodissa 29 esitellään oman taksonomian luominen ja sen liittäminen sisältötyyppiin.

```
function __construct() {
    add_action('init', array($this, 'create_event_tax' ));
}

function create_event_tax() {
    register_taxonomy(
          ' event_category',
          'events',
          array(
              'label' => __(' Event categories'),
              'rewrite' => array( 'slug' => ' event-category' ),
              'hierarchical' => true,
          )
    );
}
```

ESIMERKKIKOODI 29. Yksinkertainen esimerkki oman taksonomian liittämisestä sisältötyyppiin

**Tapahtumien metatiedot**

WordPress-tietokannan oletuskenttien lisäksi tapahtumien tiedot lisätään omiin kenttiinsä tietokantaan. Uusia tauluja ei kuitenkaan luoda. Tapahtumiin lisätyt kentät esitellään liitteessä 3.

**Karttanäkymä ja muut käytetyt skriptit**

Karttanäkymä, päivämäärän ja ajan valinta sekä joidenkin kenttien validointi tapahtuu ulkopuolisia skriptejä käyttäen. Karttanäkymässä käytetään ulkopuolista rajapintaa (Google Maps JavaScript API). Päivämäärän ja ajan valinnassa käytetään ulkopuolista kirjastoa jQuery Timepicker Addon, joka on avoimen lähdekoodin lisäosa WordPress-ytimestä löytyvään jQuery Datepicker -kirjastoon, joka itsessään ei mahdollista kellonajan valintaa. Lisäksi hintakenttien validointi tapahtuu itse luotua JavaScript-funktiota käyttäen. Myös kartan hallintaa varten on luotu oma skriptitiedostonsa, joka sisältää useita funktioita eri käyttötarkoituksiin, kuten käyttäjän sijainnin hakemisen tapahtumaa lisättäessä, sekä tapahtuman sijainnin näyttämisen tapahtumia esitettäessä.

### 6.3.4 Host-lisäosa

*Host*-lisäosa, eli isäntälisäosa, mahdollistaa tapahtumien lisäämisen sekä hallinnan. Isäntälisäosa koostuu kahdesta päätoiminnallisuudesta: tapahtumien hallinnasta ja teknisestä rajapinnasta. Lisäosan tiedostorakenne esitellään liitteessä 4.

**Lisäosan asetukset**

Lisäosan asetuksista voidaan asettaa rajapinnan polku. Perusarvoltaan polku on muotoa *v01/api*. Rajapinnan polku määrittää, mistä osoitteesta tapahtumat haetaan asiakasohjelmiin.

**Tapahtumien lisääminen**

Tapahtumia voidaan lisätä kahdella eri tavalla. Ylläpitopaneelin tapahtumahallinnan kautta, tai lisäosan mahdollistamalta erilliseltä sivulta. Erillistä lisäyssivua varten luodaan uusi tyhjä sivu, johon lisätään lyhytkoodi: *[worker_form]*. Lisäyssivu on julkinen, mutta sen sisältö (tapahtumien lisäyslomake) näytetään vain sisäänkirjautuneille käyttäjille, joilla on tarvittavat oikeudet. Lisäksi sivun ulkoasu (template) ylikuormitetaan, jotta lyhytkoodin määritykset hallitsevat sitä teemojen sijaan. Liitteessä 5 esitellään tapahtumien lisäysnäkymä etusivulla Twenty Twelve -teemaa käyttäen, ja liitteessä 6 tapahtumien lisääminen ylläpitopaanelin kautta.

**Tapahtumien hallinta**

Tapahtumia hallitaan ylläpitopaneelista. Tapahtumia voidaan poistaa roskakoriin, roskakorin kautta kokonaan ja niiden tietoja voidaan muokata. Lisäksi tapahtumat voidaan siirtää luonnos-tilaan (draft), sekä merkitä tarkastusta vaativiksi (pending). Tapahtumia voidaan myös perua asettamalla ne peruttu-tilaan (cancelled). Tapahtuman tila tallennetaan sen metatietoihin.

**Tapahtumien poistuminen**

Tapahtumien päättymispäivä tarkistetaan aina kun rajapintaa kutsutaan, tai kun käyttäjä saapuu sivulle. Päättymispäivämäärää ja aikaa verrataan nykyiseen hetkeen, ja jos ajankohta on jo mennyt, tapahtumat siirretään luonnos-tilaan. Tähän käytetään *pre_get_posts*-koukkua, jonka kutsumassa funktiossa suoritetaan vertailulogiikka (esimerkkikoodi 30). Koukkua kutsutaan *query*-olion luomisen jälkeen, mutta ennen

kuin se ajetaan. Tämä mahdollistaa vaikuttamisen olion sisältöön, ja siihen mitä tietoa käyttäjälle esitetään.

```
$args = array(
     'post_type'   => 'events',
     'post_status' => 'publish',
     'numberposts' => -1
);

$posts = get_posts($args);
foreach ($posts as $post)
{
    $compare = get_post_meta($post->ID, 'event_end_order');

    if ($compare[0] < $this->parse_the_time('YmdHi')) {
        $post = array('ID' => $post->ID, 'post_status' => 'draft');
        wp_update_post($post);
    }
}
```

ESIMERKKIKOODI 30. Tapahtumat siirretään luonnos-tilaan (draft)

**Rajapinta**

Rajapinta mahdollistaa tiedon viemisen eteenpäin JSON-LD-formaattia käyttäen. Rajapinnassa käytetään Slim PHP Framework -nimistä PHP-sovelluskehystä, joka helpottaa rajapinnan hallintaa sekä mahdollistaa helpon laajennettavuuden jatkossa. Rajapinnan kutsut listataan taulukossa 1. Jatkossa *GET*-kutsujen valikoimaa on tarkoitus laajentaa, ja myös *POST*-kutsujen toteuttamista on suunniteltu, jotta rajapinnasta saataisiin jatkossa mahdollisesti kaksisuuntainen. Muutamia suunniteltuja *GET*-kutsuja esitellään taulukossa 2.

TAULUKKO 1. GET-metodiin perustuvat rajapinnan kutsut

| **KUTSU** | **KUTSUN KUVAUS** |
|---|---|
| <url>/<rajapinnan polku>/event | Palauttaa kaikki tapahtumat |
| <url>/<rajapinnan polku>/event/<id> | Palauttaa yksittäisen tapahtuman ID:n perusteella |

TAULUKKO 2. GET-metodiin perustuvat suunnitellut kutsut

| **KUTSU** | **KUTSUN KUVAUS** |
|---|---|
| <url>/<rajapinnan polku>/event/active | Palauttaa vain aktiiviset tapahtumat |
| <url>/<rajapinnan polku>/event/cancelled | Palauttaa vain perutut tapahtumat |
| <url>/<rajapinnan polku>/location | Palauttaa tapahtumien sijainnit |
| <url>/<rajapinnan polku>/organizer | Palauttaa tapahtumien järjestäjät |

### 6.3.5 Client-lisäosa

*Client*-lisäosa, eli asiakaslisäosa, mahdollistaa tapahtumien hakemisen isäntälisäosasta, sekä näiden tapahtumien esittämisen eri muodoissa. Lisäosan tiedostorakenne esitellään liitteessä 7.

**Lisäosan asetukset**

Lisäosan asetuksista täytyy asettaa isäntälisäosan polku, josta tapahtumat haetaan. Ilman tätä toimenpidettä lisäosa ei toimi, sillä se ei vastaanota tapahtumia mistään.

**Tapahtumien hakeminen**

Tapahtumat haetaan lisäosaan asetuksissa määritellyn isäntälisäosan rajapinnan osoitteen perusteella ja ne lisätään tietokantaan. Tapahtumat haetaan HTTP-rajapintaa käyttäen. Lisäys tietokantaan tapahtuu WordPressin tarjoamia *wp_insert_post()*- ja *wp_update_post()*-funktioita käyttäen. Ensimmäistä käytetään, jos tapahtumaa ei löydy tietokannasta. Tarkistus tapahtuu tapahtuman versionumeron avulla (metakenttä). Jälkimmäistä käytetään, jos tapahtumaan on tullut muutoksia. Muutokset tarkistetaan tapahtuman metatiedoista muokkauspäivämäärän mukaan. Päivityslogiikkaa ei siis suoriteta turhaan.

**Tapahtumien esittäminen**

Tapahtumien esittämiseen käytetään normaalisti WordPress-teemakehityksestä tuttuja menetelmiä. Koska erillistä teemaa ei haluttu toteuttaa, ylikuormitetaan käytettävän teeman arkiston (Post Archive) ulkoasu. Uusi arkistopohja ladataan tapahtumasivun lyhytlinkin (Permalink) nimen mukaan ja tästä syystä onkin luotava uusi WordPress-sivu lyhytlinkillä *events*. Jää käyttäjän päätettäväksi, esitetäänkö tapahtumat alkusivulla vai alasivulla. Tapahtumat järjestetään alkupäivämäärän sekä alkamisajankohdan

mukaiseen järjestykseen. Tapahtumien esittäminen Twenty Twelve- ja Kubrick 2014 -teemoja käyttäen esitellään liitteessä 8.

WordPress-silmukka (The Loop) on mekanismi, joka hakee tapahtumat (räätälöidyt sisältötyypit) tietokannasta ja esittää ne. Muitakin tapoja on. Seuraavassa esimerkissä esitellään kuitenkin yleisin tapa hakea kaikki tapahtumat (esimerkkikoodi 31).

```
<?php
if (have_posts()) {
    while (have_posts()) {
        the_post();
        // sisältö luodaan tänne
    }
}
?>
```

ESIMERKKIKOODI 31. Tapahtumien hakeminen *default*-silmukkaa käyttäen

**Tapahtumien poistuminen**

Tapahtumat poistetaan luonnoksiksi käyttäjän saapuessa sivulle samaan tapaan kuin isäntälisäosassa (Esimerkkikoodi 30).

**Tiedostogeneraatio**

PDF- ja tekstitiedoston luomista varten tiedot haetaan isäntälisäosan rajapinnasta. Tiedostogeneraatio on luotu rajapinnan kautta sen jatkokehitystarpeiden vuoksi. Rajapinnan muokattavuus mahdollistaa sen monikäyttöisyyden, ja tiedostot voidaan jatkossa generoida myös muista lähteistä rajapinnan *template*-tiedostoja muokkaamalla, käyttäjän näin halutessa. Jatkossa asetuksiin luodaan valinta tätä tarkoitusta varten.

PDF-generaatioon käytetään FPDF-nimistä PHP-kirjastoa (esimerkkikoodi 32), joka on ilmainen ja julkaistu vapaalla lisenssillä (FPDF FAQ 2014), kuten muutkin lisäosissa käytetyt ulkoiset kirjastot. Tekstitiedosto generoidaan PHP-funktioita *fwrite()* käyttäen (esimerkkikoodi 33).

```
<?php
require('fpdf.php');
$pdf = new FPDF();
$pdf->AddPage();
$pdf->SetFont('Arial', 'B', 16);
$pdf->Cell(40, 10, 'Hello World!');
$pdf->Output('example.pdf');
?>
```

ESIMERKKIKOODI 32. Yleinen esimerkki FPDF-kirjaston käytöstä

```
<?php
$file = fopen("example.txt", "w");
echo fwrite($file, "Hello World!");
fclose($file);
?>
```

ESIMERKKIKOODI 33. Yleinen esimerkki tekstitiedostoon kirjoittamisesta PHP:een *fwrite()*-funktiota käyttäen

Tiedostojen luominen tapahtuu taustalla AJAX-toimintoja käyttäen aina sivuston aktivoituessa (esim. käyttäjä saapuu sivustolle), joten se ei häiritse tai hidasta lisäosien ja itse sivuston toimintaa.

### 6.4 Lisäosien julkaiseminen, ylläpito ja päivitys

Lisäosa voidaan haluttaessa julkaista virallisessa WordPress-lisäosakirjastossa, jos se läpäisee kaikki julkaisulle määritetyt säännöt. Ylläpitoa ja päivitystä hallitaan prototyyppien kohdalla kuitenkin GitHub-versionhallinan kautta, sillä lisäosia ei ole vielä virallisesti julkaistu tai otettu käyttöön.

Päivittämistä varten lisäosan versionumero kannattaa sisällyttää tietokantaan, sillä sen avulla voidaan helposti myöhemmin tarkistaa, onko esimerkiksi lisäosan tietokannan kenttien päivittämiselle, poistamiselle tai lisäämiselle tarvetta. Jos tietokannan

versionumero on vanhempi kuin lisäosan ilmoittama versionumero, suoritetaan päivityslogiikka ja lisäksi päivitetään uusi versionumero tietokantaan.

## 6.5 Työskentelytavat ja työnkulku

Työskentely- sekä testiympäristö rakentui kotonani sijaitsevasta pöytäkoneesta, toimeksiantajan käyttööni antamasta kannettavasta, sekä Kapsi Internet -käyttäjät ry:n tarjoamasta palvenlintilasta Unix-palvelimella Apache-palvelinohjelmalla varustettuna. Pöytäkoneella alustana toimi Windows 8.1 ja palvelinympäristönä XAMPP. Kannettavalla alustana toimi Linux Mint 16 Petra ja palvelinympäristönä LAMP. Tiedostot synkronisoitiin testiympäristöjen välillä GitHub-versionhallintaa käyttäen.

### 6.5.1 Palvelinympäristöt

XAMPP on alustariippumatton ilmainen avoimen lähdekoodin ohjelma, joka koostuu pääasiassa Apache-säätiön HTTP-palvelimesta, MySQL-tietokannasta sekä PHP- ja Perl-skripteistä. XAMPP on alunperin suunniteltu käytettäväksi vain kehittämisen välineenä lokaalisti (mahdollisesti ilman internetiä). XAMPP-ohjelmistoa on kuitenkin mahdollista käyttää normaalin palvelimen tapaan verkkosivujen esittämiseen omalta koneelta. (Apache Friends 2014.)

LAMP on kokoelma avoimen lähdekoodin ohjelmistoja, jotka yhdessä muodostavat WWW-palvelimen. Lyhennettä käytetään kuvaamaan erittäin yleistä web-palvelinteknologian kokoonpanoa, jossa palvelinkoneen käyttöjärjestelmänä on Linux, palvelinohjelmistona Apache, tietokantaohjelmistona MySQL ja dynaamisuuden ja vuorovaikutteisuuden mahdollistavana palvelinpuolen skriptikielenä PHP. Myös Python- ja Perl-skriptikielet ovat mahdollisia. Lyhenne LAMP on selventävä termi, eikä esimerkiksi ohjelmien välinen vakiintunut yhteyskäytäntö. (Brown 2005.)

### 6.5.2 Versionhallinta

Prototyyppien toteutuksessa versionhallintatyökaluksi valittiin Git ja GitHub. GitHub on erittäin suosittu web-pohjainen palvelu, joka käyttää perustanaan Git-versionhallintajärjestelmää. Vuonna 2014 GitHub:illa arvioitiin olevan yli 3,4 miljoonaa käyttäjää (Whitaker 2014).

Git on hajautettu versionhallintajärjestelmä, jonka kehityksen aloitti Linus Torvalds Linuxin ytimen kehitystä varten. Se korvasi suljetun lähdekoodin BitKeeperin ja julkaistiin vuonna 2005 (Chacon 2009, 4). GitHub on ilmainen kaikille avoimen lähdekoodin projekteille (julkiset tietovarastot), mutta maksullinen suljetun lähdekoodin projekteille (yksityiset tietovarastot). Monet avoimen lähdekoodin projektit ovatkin siirtyneet käyttämään GitHubia sen sijaan, että itse ylläpitäisivät omaa Git-palvelintaan.

GitHubissa käytettävä hajautettu versionhallintajärjestelmä tarkoittaa sitä, että jokaisella käyttäjällä on käytössä paikallinen kopio versionhallinnasta. Käyttäjät tekevät omaan paikalliseen kopioonsa muutoksia ja lopulta liittävät (merge) oman kehityshaaransa (branch) projektin pääasialliseen versionhallintajärjestelmään.

Lisäosia ja dokumentaatiota varten luotiin GitHubiin oma organisaatio. Organisaatiot helpottavat ja yksinkertaistavat ryhmäomisteisten tietovarastojen kehitystä sekä hallintaa. Organisaatiot myös helpottavat ryhmätyöskentelyä (collaboration) huomattavasti.

### 6.5.3 Dokumentaatio

Dokumentaation alustana toimi GitHub. Dokumentaatio on oleellinen osa ohjelmistokehitystä ja dokumentaation muokattavuus sekä avoimuus ovat erittäin tärkeäitä varsinkin suhteellisen pienissä avoimen lähdekoodin projekteissa. Dokumentaatiota työstettiin läpi työskentelyprosessin ja osana sitä syntyneet lisäosien käyttöohjeet löytyvät verkkoliitteenä osoitteesta https://github.com/event-worker/event-worker-dev/wiki. Verkkoliitteeseen päädyttiin, sillä käyttöohjeet ovat erittäin laajat ja päivittyvät jatkossa suhteellisen useasti.

### 6.5.4 Testaus

Testausta on tehty jatkuvasti ja suhteellisen joustavasti eri palvelinympäristöjen välillä lisäosien kehityksen edistyessä. Kunnollista käytettävyystestausta ei kuitenkaan ole päästy tekemään omien ystävilleni teettämieni testitilaisuuksien, sekä toimeksiantajan demotilaisuuksien tarjoamia pienimuotoisia mahdollisuuksia lukuunottamatta. Myöskin laajamuotoinen yksikkötestaus (Unit testing) on tekemättä, sillä lisäosan tuotantokäyttöön saattaminen tulee vielä viemään aikaa, koska kyseessä ovat lisäosien prototyypit. PHP-luokat joita kuitenkin testattiin kehityksen alkuvaiheessa, läpäisivät testiprosessit, ja täten luokat toteutettiin jatkossa samoin menetelmin. Perusteellinen ja laajamittainen testausprosessi on kuitenkin vielä tekemättä.

### 6.5.5 Haasteet

JSON-LD-määrittelyjen noudattaminen osoittautui haastavimmaksi osa-alueeksi opinnäytetyön toteutusosiota tehdessä. Schema.org:in JSON-LD määrittely tapahtumille on todella uusi, eikä aivan vastannut tarpeita. Mallia otettiin myös jonkin verran Helsingin kaupungin Linked Events -määrittelyistä, jotka osoittautuivat keskeneräisiksi, ja eivät myöskään noudattaneet Schema.org:in standardia.

Lisäksi työmäärän arviointi ohjelmointiin osoittautui haastavaksi, sillä mikään lisäosien kehitykseen liittyvä ei ollut oikeastaan ennestään tuttua, käytettyjä ohjelmointikieliä lukuun ottamatta. Tästä syystä osa toiminnoista on toteutettu hyvin nopeasti, ja täten ne eivät välttämättä ole täydellisesti hyvien ohjelmointikäytäntöjen mukaisia. Komponentteja tullaan luultavasti tästä syystä toteuttamaan osittain uudestaan, tai vähintään niiden toimintaa tullaan tehostamaan. Lisäosien rakenne on kuitenkin toimiva ja tarkoituksenmukainen. Lisäksi ne ovat helposti jatkokehitettävissä.

# 7 YHTEENVETO JA POHDINTA

Opinnäytetyötä kirjoitettaessa lisäosien kehitysprosessi oli vielä kesken, ja muutoksia saattoi tulla jatkuvasti uusia ohjelmointivirheitä löydettäessä. Tämä hankaloitti opinnäytetyöprosessia huomattavasti esimerkiksi lisäosien rakenteen kuvaamisen kohdalla UML-kaavioin, tai muilla tavoin. Myös opinnäytetyötä tehtäessä WordPress, sekä sen dokumentaatio päivittyivät. Tästä ei kuitenkaan seurannut suurempia ongelmia.

Alkuvaiheessa kehitystä suunnitelmana oli toteuttaa vain yksi lisäosa, joka olisi sisältänyt *host*- ja *client*-lisäosien toiminnallisuudet. Kehitysprosessin edetessä huomattiin, että lisäosat kannattaa eriyttää. Kehitysprosessin muutos oli kuitenkin kannatava, varsinkin, jos pohditaan jatkokehitystä tai tulevia suurempia muutoksia nykyisiin toiminnallisuuksiin. Osittain suunnitelmista poikkeaminen aiheutti sen, että esiteltävät koodiesimerkit ovat hyvin yleisiä, sillä lisäosien rakenne tulee vielä muuttumaan. Toteutetut toiminnallisuudet pysyvät kuitenkin myös tulevaisuudessa pääosin samoina.

Toteutukseen otettiin haastava näkökulma. Palvelua pitäisi voida käyttää eri WordPress-teemoilla, eli sen pitäisi olla ulkoasusta riippumaton. Kaikissa tapauksissa tämä ei kuitenkaan alkututkimusten perusteella ollut mahdollista, sillä kaikkia teemoja ei ole toteutettu samojen standardien mukaan. Helpointa olisi ollut toteuttaa tarvittavat toiminnot omaan teemaansa. Ulkoasun valintaa ei kuitenkaan haluttu rajoittaa sitomalla toiminnallisuutta tiettyyn ulkoasuun.

Rajapintojen kohdalla JSON-LD, linkitetty data sekä semanttinen verkko olisivat oma laaja aiheensa opinnäytetyöksi, ja niihin ei sen vuoksi ole paneuduttu kuin niiltä osin, mitkä ovat oleellisia toteutuksen kannalta. Rajapinnan määrittelyt eivät myöskään ole täydelliset tai valmiit, joten niitä esitellään vain pintapuolisesti. Järjestelmän kehityksessä on haluttu kuitenkin katsoa tulevaisuuten, joten määritelmät tehtiin sen mukaisesti JSON-LD-formaattia käyttäen. Apuna käytettiin myös Helsingin kaupungille tehtyjä Linked Events -määrittelyjä sen verran kun oli mahdollista.

Opinnäytetyöprosessissa on käytetty aineistona alan kirjallisuuden lisäksi laajasti keskusteluja IRC:issä (Internet Relay Chat), keskustelupalstoja sekä pienimuotoisia

sähköpostikeskusteluja eri tahojen välillä. Suurimpana apuna toimivat kuitenkin WordPressin dokumentaatio sekä erilaiset verkkoartikkelit ja oppaat. Kirjoitusprosessin edetessä tuli selväksi, että työ olisi pitänyt tehdä alusta asti englanniksi, sillä se olisi ollut itselleni luontevampaa alan sanastoa käytettäessä. Kirjoitusprosessi ei edistynyt toivottuun tapaan, sillä suomenkielisten vastineiden etsiminen ja miettiminen eri termeille, jotka eivät ole vakiintuneita alan sanastoon, veivät todella paljon aikaa. Esimerkiksi teemoista löytyy laajasti aiempaa materiaalia. Lisäosien toteutuksesta luotettavaa ja ajankohtaista materiaalia löytyy pääasiassa vain englanniksi, ja materiaalin taso on todella vaihtelevaa.

Jatkokehityksen kannalta olisi järkevää toteuttaa osa tapahtumienhallinan toiminnoista omiksi taksonomioikseen. Esimerkiksi tapahtuman järjestäjä, sillä oletettavasti samat järjestävät lisäävät tapahtumiaan toistuvasti, ja täten järjestäjät olisivat myös tietokannasta valittavina kuten tapahtumien kategoriat. Jatkossa on muutenkin tarkoitus käyttää WordPressin rajapinnan tarjoamia mahdollisuuksia laajemmin kuin nykyisissä versioissa lisäosista. Asiakaslisäosan toimintaperiaatteeseen on myös tulevaisuudessa tulossa suurempi muutos, sillä asiakaslisäosa luo tapahtumista oman tietokantansa, ja tämä ei ole semanttisen verkon ratkaisujen mukaista. Täydellisessä semanttisessa linkitetyn datan ratkaisussa lisäosa vain esittäisi haetut tapahtumat, eikä loisi niistä kopioita omaan tietokantaansa.

Itse olen lopputuloksiin tyytyväinen, sillä kokemusta vastaavasta WordPress-kehityksestä minulla ei ollut, mutta kaikki onnistui silti tavoitteiden mukaisesti. Tyytyväinen on myös toimeksiantaja, sillä tapahtumienhallintaan toteutetut prototyypit ovat halutun mukaiset, ja ne toimivat niin kuin niiden on tarkoituskin. Parannettavaa on silti paljon, mutta nyt parhaat käytännöt ja tekniikat ovat kuitenkin paremmin selvillä, ja niitä voidaan soveltaa jatkokehityksessä laajasti. Erillisien asiakassovellusten kehittäminen on myöskin jatkossa helppoa, sillä kaikki tuotettu materiaali on julkaistu avoimella lisenssillä ja rajapinnan kautta saadaan kaikki tarvittava tieto tapahtumista. Opinnäytetyön tarkoitus ja tavoite täyttyivät siis hyvin.

**LÄHTEET**

Apache Friends. 2014. XAMPP. Luettu 01.08.2014.
https://www.apachefriends.org/index.html

Bing blogs. 2011. Introducing Schema.org: Bing, Google and Yahoo Unite to Build the Web of Objects. Luettu 20.08.2014.
http://blogs.bing.com/search/2011/06/02/introducing-schema-org-bing-google-and-yahoo-unite-to-build-the-web-of-objects/

Boiko, B. 2005. Content management bible. Indianapolis: Wiley Publishing Inc.

Brown, M. 2005. Understanding LAMP. ServerWatch. Luettu 01.08.2014.
http://www.serverwatch.com/tutorials/article.php/3567741/Understanding-LAMP.htm

Chacon, S. 2009. Pro Git. 1st edition. Yhdysvallat: Apress.

Elliot, R. 2011. 7 Simple Rules: WordPress Plugin Development Best Practices. Luettu 1.09.2014.
http://code.tutsplus.com/articles/7-simple-rules-wordpress-plugin-development-best-practices--wp-22996

Euroopan unionin avoimen datan portaali. 2014. Linkitetty data. Tietoa linkitetystä datasta. Luettu 10.10.2014.
https://open-data.europa.eu/fi/linked-data

FPDF FAQ. 2014. What's exactly the license of FPDF? Are there any usage restrictions?. Luettu 10.09.2014.
http://www.fpdf.org/en/FAQ.php

Goodman, D., Morrison, M., Novitski, P., Rayl, T. 2010. JavaScript Bible. 7th edition. New Jersey: Wiley.

Google Official Blog. 2011. Introducing schema.org: Search engines come together for a richer web. Luettu 20.08.2014.
http://googleblog.blogspot.fi/2011/06/introducing-schemaorg-search-engines.html

Karvonen, T. 2004. Berners-Lee: semanttinen verkko vapauttaa ihmisiä tietojenkäsittelystä. Luettu 09.09.2014.
http://www.digitoday.fi/data/2004/06/16/berners-lee-semanttinen-verkko-vapauttaa-ihmisia-tietojenkasittelysta/200411119/66

Koponen, P. 2012. Linked Events. Tapahtumarajapinta. Esitelmä. Helsingin kaupungin kehittäjätapaaminen 10.5.2012. Digiartikkeli. Luettu 20.09.2014.
http://www.slideshare.net/HelsinkiLovesDevelopers/linked-events-1052012

Marshall B. 2014. WordPress AJAX: Frontend & Backend Implementation. Luettu 22.09.2014
http://www.benmarshall.me/wordpress-ajax-frontend-backend/

McFarlin, T. 2013a. WordPress and MVC (Gloves Aren't Made for Feet). Luettu 5.10.2014.
https://tommcfarlin.com/wordpress-and-mvc/

McFarlin, T. 2013b. Design Patterns in WordPress: We're Just Getting Started. Luettu 5.10.2014.
http://code.tutsplus.com/articles/design-patterns-in-wordpress-were-just-getting-started--wp-31663

PHP Manual. 2014. What can PHP do?. Luettu 09.09.2014.
http://php.net/manual/en/intro-whatcando.php

Profium. 2014. JSON-LD. Luettu 25.09.2014
http://www.profium.com/fi/teknologiat/json-ld

Ruotsalainen, R. 2010. Linkitetty avoin tieto – linked open data. Positio 2, 26-27. Luettu 01.10.2014.
http://www.paikkatietoikkuna.fi/c/document_library/get_file?uuid=2e851a4b-c1cf-460e-8956-0e8a0a0b4109&groupId=108478

Schema blog. 2011. Yandex now supports schema.org markup. Luettu 20.08.2014.
http://blog.schema.org/2011/11/yandex-now-supports-schemaorg-markup.html

Schema FAQ. 2014. Frequently asked questions. Luettu 20.08.2014.
https://schema.org/docs/faq.html

The jQuery Foundation. 2014. jQuery License. Luettu 10.08.2014.
https://jquery.org/license/

W3C. 2013. HTML & CSS. Luettu 10.8.2014.
http://www.w3.org/standards/webdesign/htmlcss

W3C. 2014. JSON-LD 1.0: Syntax Tokens and Keywords. Luettu 10.8.2014.
http://www.w3.org/TR/json-ld/#syntax-tokens-and-keywords

W3Schools. 2014. JavaScript Introduction. Luettu 10.8.2014.
http://www.w3schools.com/js/js_intro.asp

W3Techs. 2014. Usage of content management systems for websites. Luettu 20.10.2014
http://w3techs.com/technologies/overview/content_management/all

Whitaker, M. April. 2014. Former UC student establishes a celebrated website in GitHub that simplifies coding collaboration for millions of users. University of Cincinnati. Luettu 12.08.2014.
http://magazine.uc.edu/favorites/web-only/wanstrath.html

WordPress. 2003. WordPress Now Available. Luettu 02.08.2014.
https://wordpress.org/news/2003/05/wordpress-now-available/

WordPress. 2014a. GNU General Public License. Luettu 01.12.2014.
https://wordpress.org/about/gpl/

WordPress. 2014b. About WordPress. Luettu 02.08.2014.
https://wordpress.org/about/

WordPress. 2014c. Requirements. Luettu 02.08.2014.
https://wordpress.org/about/requirements/

WordPress. 2014d. Plugins. Luettu 02.08.2014.
https://wordpress.org/plugins/

WordPress Codex. 2014a. Using Alternative Databases. Luettu 1.10.2014.
http://codex.wordpress.org/Using_Alternative_Databases

WordPress Codex. 2014b. Plugins. Luettu 02.08.2014.
http://codex.wordpress.org/Plugins

WordPress Codex. 2014c. Writing a Plugin. Luettu 02.08.2014.
http://codex.wordpress.org/Writing_a_Plugin

WordPress Codex. 2014d. Plugin API: Actions. Luettu 13.08.2014.
http://codex.wordpress.org/Plugin_API#Actions

WordPress Codex. 2014e. Validating Sanitizing and Escaping User Data. Luettu 02.11.2014.
http://codex.wordpress.org/Validating_Sanitizing_and_Escaping_User_Data

WordPress Codex. 2014f. WordPress Nonces. Luettu 02.11.2014.
http://codex.wordpress.org/WordPress_Nonces

WordPress Codex. 2014g. Settings API. Luettu 11.08.2014.
http://codex.wordpress.org/Settings_API

WordPress Codex. 2014h. HTTP API. Luettu 12.08.2014.
http://codex.wordpress.org/HTTP_API

WordPress Codex. 2014i. Function Reference: wp_enque_script. Luettu 02.09.2014.
http://codex.wordpress.org/Function_Reference/wp_enqueue_script

WordPress Codex. 2014j. Function Reference: wp_enque_style. Luettu 02.09.2014.
http://codex.wordpress.org/Function_Reference/wp_enqueue_style

WordPress Codex. 2014k. I18n for WordPress Developers. Luettu 02.10.2014.
http://codex.wordpress.org/I18n_for_WordPress_Developers

WordPress Plugin Development Handbook. 2014a. Activation / Deactivation Hooks. Luettu 07.10.2014.
https://developer.wordpress.org/plugins/the-basics/activation-deactivation-hooks/

WordPress Plugin Development Handbook. 2014b. Uninstall Methods. Luettu 07.10.2014.
https://developer.wordpress.org/plugins/the-basics/uninstall-methods/

WordPress Support. 2014. Forums. Luettu 12.11.2014.
https://wordpress.org/support/

Yahoo! Search Blog. 2011. Introducing schema.org: A Collaboration on Structured Data. Luettu 20.08.2014.
http://www.ysearchblog.com/2011/06/02/introducing-schema-org-a-collaboration-on-structured-data/

# LIITTEET

Liite 1. AJAX:in käyttö proseduaalisesti *backend*-puolella

Liitteenä olevassa esimerkkikoodissa esitellään AJAX:in lisääminen ja käyttö *backend*-puolella (Marshall 2014).

```
<?php
add_action( 'admin_footer', 'my_action_javascript' );

function my_action_javascript() {

  //Set Your Nonce
  $ajax_nonce = wp_create_nonce( "my-special-string" );
  ?>
  <script>
  jQuery( document ).ready( function( $ ) {
    var data = {
      action: 'my_action',
      security: '<?php echo $ajax_nonce; ?>',
      whatever: 1234
    };

    // Since 2.8 ajaxurl is always defined in the admin header and
points to admin-ajax.php
    $.post( ajaxurl, data, function( response ) {
      alert( 'Got this from the server: ' + response );
    });
  });
  </script>
<?php } ?>


<?php
add_action( 'wp_ajax_my_action', 'my_action_callback' );

function my_action_callback() {
  global $wpdb; // this is how you get access to the database

  check_ajax_referer( 'my-special-string', 'security' );
  $whatever = intval( $_POST['whatever'] );
  $whatever += 10;
  echo $whatever;

  die(); // this is required to return a proper result
}
?>
```

Liite 2. AJAX:in käyttö proseduaalisesti *frontend*-puolella

Liitteenä olevassa esimerkkikoodissa esitellään AJAX:in lisääminen ja käyttö *frontend*-puolella (Marshall 2014).

```
<?php
// Add the JS
function theme_name_scripts() {
  wp_enqueue_script( 'script-name', get_template_directory_uri() .
'/js/example.js', array('jquery'), '1.0.0', true );
  wp_localize_script( 'script-name', 'MyAjax', array(
    // URL to wp-admin/admin-ajax.php to process the request
    'ajaxurl' => admin_url( 'admin-ajax.php' ),

    // generate a nonce with a unique ID "myajax-post-comment-
nonce"
    // so that you can check it later when an AJAX request is sent
    'security' => wp_create_nonce( 'my-special-string' )
  ));
}
add_action( 'wp_enqueue_scripts', 'theme_name_scripts' );

// The function that handles the AJAX request
function my_action_callback() {
  check_ajax_referer( 'my-special-string', 'security' );

  $whatever = intval( $_POST['whatever'] );
  $whatever += 10;
  echo $whatever;
  die(); // this is required to return a proper result
}
add_action( 'wp_ajax_my_action', 'my_action_callback' );


<script>
jQuery(document).ready(function($) {

  var data = {
    action: 'my_action',
    security : MyAjax.security,
    whatever: 1234
  };

  // since 2.8 ajaxurl is always defined in the admin header and
points to admin-ajax.php
  $.post(MyAjax.ajaxurl, data, function(response) {
    alert('Got this from the server: ' + response);
  });
});
</script>
?>
```

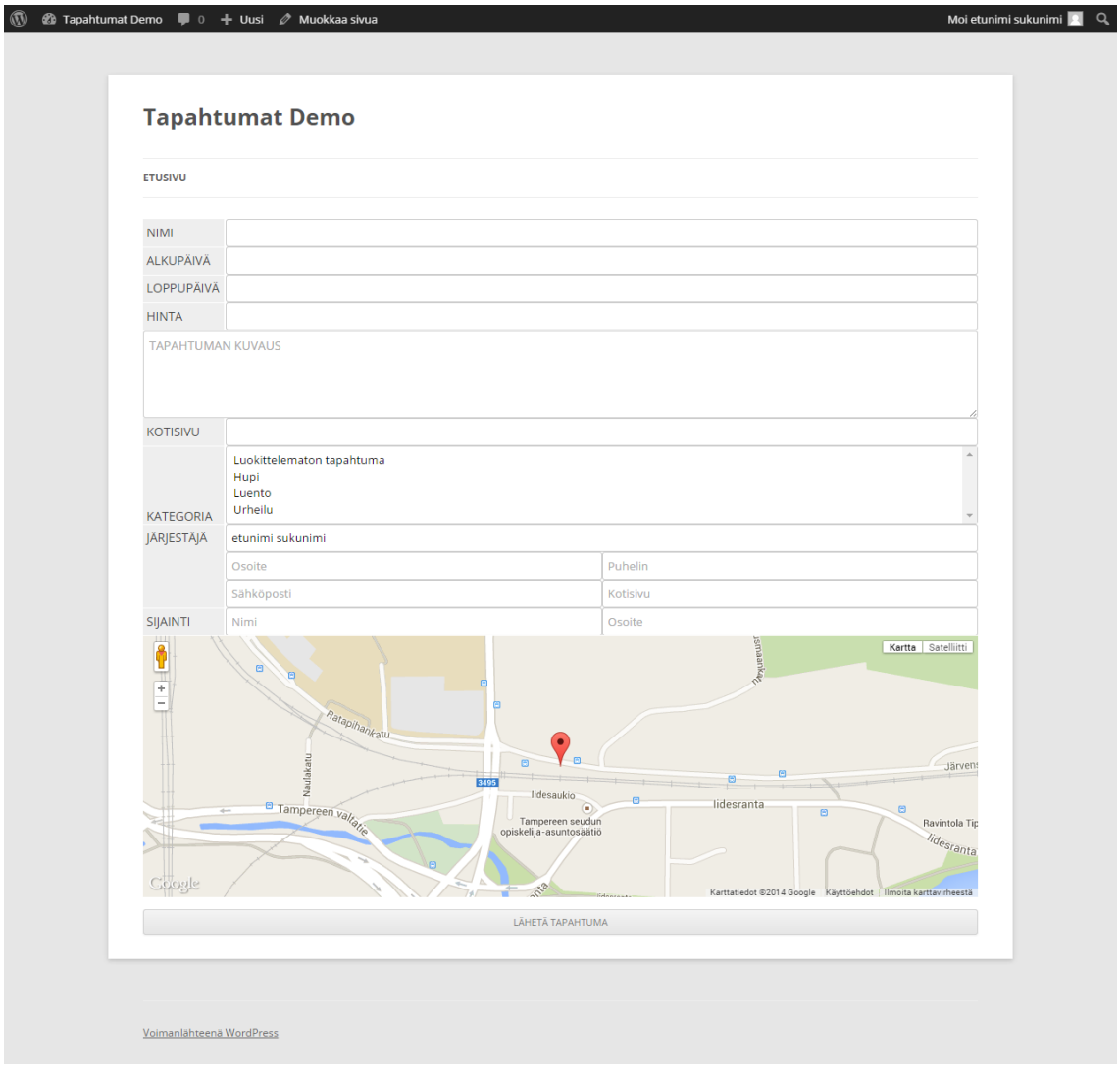
Liite 3. Lisäosien tapahtumien metatietokentät tietokannassa

| KENTTÄ | KUVAUS | ESIMERKKIARVO |
|---|---|---|
| event_start_date | alkuaika | 29.11.2014 14:59 |
| event_end_date | loppuaika | 30.11.2014 15:59 |
| event_location | osoite | Kuntokatu 3, Tampere |
| event_location_name | sijainnin nimi | TAMK |
| event_geolocation | sijainnin koordinaatit | (61.491082, 23.788598999999977) |
| event_price | hinta | 4 |
| event_website | kotisivu | http://www.esimerkki.fi |
| event_organizer | järjestäjän nimi | Esimerkkijärjestäjä |
| event_organizer_data | järjestäjän tiedot:<br>- osoite<br>- puhelin<br>- email<br>- kotisivu | a:4:{s:7:"address";<br>s:20:"Kuntokatu 3, Tampere";<br>s:5:"phone";s:12:"0401111222";<br>s:5:"email";s:18:"esimerkki@maili.fi";<br>s:7:"website";s:19:"http://esimerkki.fi";} |
| event_start_order | apukenttä alkamisajan tarkistamiseen | 201411291459 |
| event_end_order | apukenttä loppumisajan tarkistamiseen | 201411301559 |
| event_modified | tapahtuman viimeinen muokkausaika | 2014-11-30 10:20:09 |
| event_version | versionumero, joka toimii tapahtuman tunnuksena | event_worker_1417265984 |
| event_status | tapahtuman tila | http://schema.org/EventCancelled |

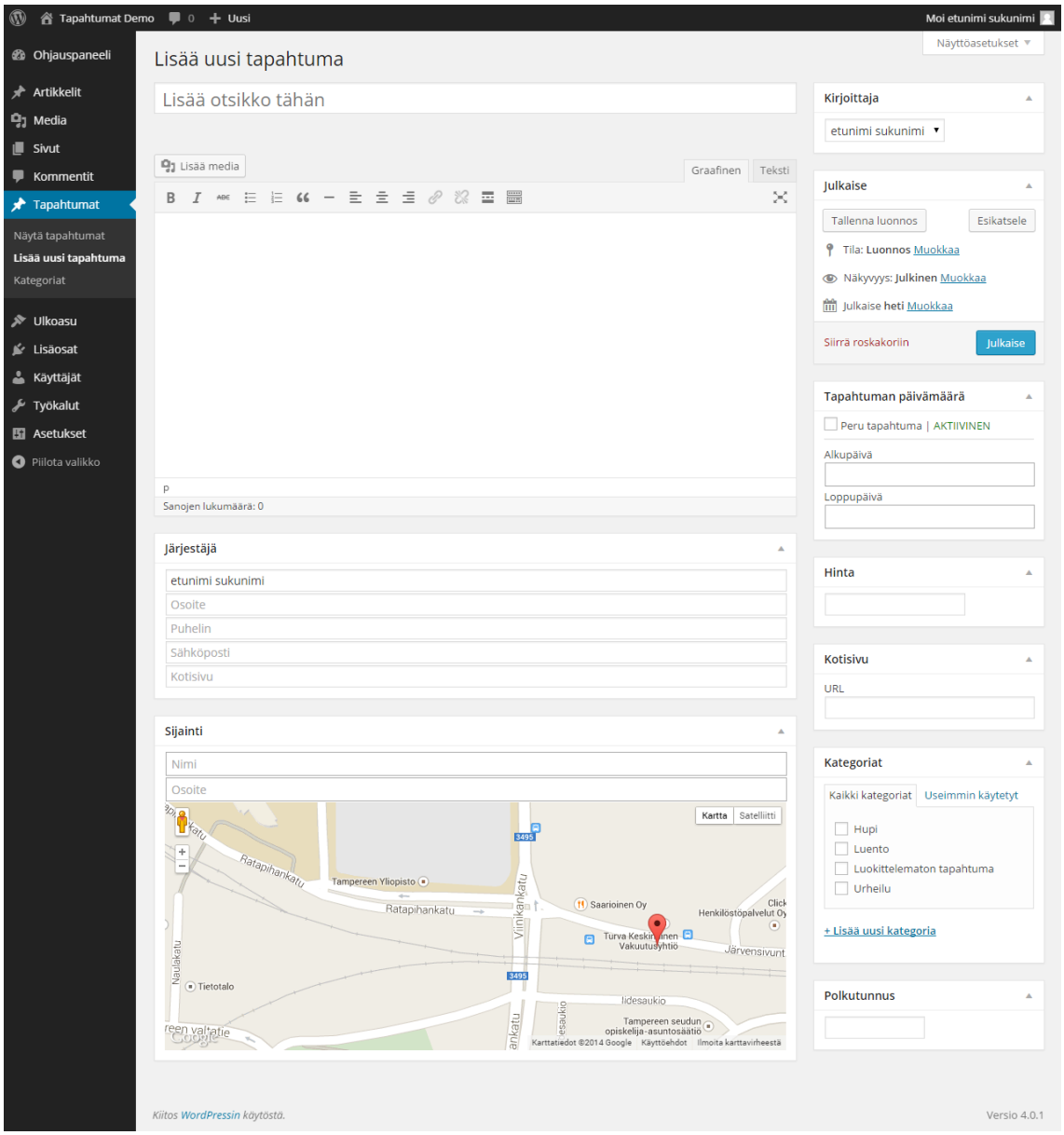
Liite 4. Isäntälisäosan tiedostorakenne

| POLKU | TIEDOSTO | KUVAUS |
|---|---|---|
| ./ | main.php | Lisäosan pääluokka |
| ./lib/ | admin-meta.php | Lisäosan ylläpitopaneelin toiminnot |
| | common-options.php | Lisäosan asetukset |
| | core.php | Lisäosan aktivointi |
| | menu-items.php | Lisää valikkoon linkin tapahtumien lisäyssivulle |
| | pre-order-events.php | Tapahtumien tarkistaminen ja järjestäminen |
| ./lib/api/ | api-routes.php | Rajapinnan kutsut, jotka kutsuvat sen näkymiä |
| | slim-helper.php | Apuluokka rajapinnalle, jossa luodaan sen asetukset |
| ./lib/api/templates/ | events.php | Rajapinnan näkymä yksittäisille ja kaikille tapahtumille |
| ./lib/api/slim/ | <slim framework> | Slim framework liitettynä WordPress-rajapintaan |
| ./lib/css/ | jquery-ui-timepicker-addon.css | Aikavalitsin-lisäosan käyttämät tyylimäärittelyt |
| | style.css | Lisäosan käyttämät tyylimäärittelyt |
| ./lib/js/ | jquery-ui-timepicker-addon.js | Javascript-kirjasto myös ajan valitsemiseen pelkän päivämäärän sijaan |
| | maphandler.js | Apufunktiota sisältävä skriptitiedosto karttatoimintojen hallintaan lisäosassa |
| | validator.js | Tarkistaa kenttien oikeaoppisuuden |
| ./lib/languages/ | event-worker-translations-fi.mo | Käännöstiedoston binääritiedosto |
| | event-worker-translations-fi.po | Käännöstiedoston työtiedosto |
| ./lib/loaders/ | scripts-and-styles-loader.php | Luokka, joka lataa skriptit ja tyylitiedostot |
| | page-template-loader.php | Apuluokka, joka lataa tarvittavat sivupohjat |
| ./lib/shortcodes/ | submit-event.php | Luokka lyhytkoodin lisäykseen |
| ./lib/templates/ | add-events.php | Luodaan lisäyssivun pohjana toimiva ns. tyhjä template-luokka, jonka avulla poistetaan turhat komponentit |

Liite 5. Tapahtumien lisääminen isäntälisäosassa Twenty Twelve -teemaa käyttäen



Liite 6. Tapahtumien lisääminen isäntälisäosan ylläpitopaanelin kautta



Liite 7. Asiakaslisäosan tiedostorakenne

| POLKU | TIEDOSTO | KUVAUS |
|---|---|---|
| ./ | main.php | Lisäosan pääluokka |
| ./lib/ | common-options.php | Lisäosan asetukset |
| | core.php | Aktivoidaan lisäosa |
| | pre-order-events.php | Tapahtumien tarkistaminen ja järjestäminen |
| | redirect.php | Ohjaa tapahtuma-arkistoon |
| .lib/file-generation/ | ajax-helper.php | Apuluokka, joka lataa ja akivoi AJAX:in käytettäväksi, ja jossa luodaan funktio jota AJAX:in kautta kutsutaan |
| | file-generator.php | Luokka jossa tiedostot generoidaan ja jota kutsutaan AJAX:in avulla apuluokan kautta |
| lib/file-generation/js/ | ajax.js | AJAX-toiminnot |
| lib/file-generation/fpdf/ | <fpdf library> | PHP-Kirjasto PDF-generointiin |
| ./lib/js/ | maphandler.js | Apufunktiota sisältävä skriptitiedosto karttatoimintojen hallintaan lisäosassa |
| ./lib/languages/ | event-worker-translations-fi.mo | Käännöstiedoston binääritiedosto |
| | event-worker-translations-fi.po | Käännöstiedoston työtiedosto |
| ./lib/loaders/ | scripts-and-styles-loader.php | Luokka joka lataa skriptit ja tyylitiedostot |
| | page-template-loader.php | Apuluokka, joka lataa tarvittavat sivupohjat |
| ./lib/templates/ | archive-events.php | Luokka tapahtuma-arkiston esittämiseen |
| | single-events.php | Luokka yksittäisen tapahtuman esittämiseen |

Liite 8. Asiakaslisäosan tapahtumanäkymä Twenty Twelve- ja Kubrick 2014 -teemoja käyttäen