

Karar Mehdi Habib

Wordpress- A convenient content-management system (CMS)

Helsinki Metropolia University of Applied Sciences

Degree Bachelor

Degree Programme Information Technology

Thesis WordPress- A convenient conten-management system (CMS)

Date 27.05.2018

Author Title	Karar Mehdi Habib Title of the Thesis
Number of Pages Date	59 pages + 5 appendices 27 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of Department
<p>The objective of this study was to discuss about WordPress in detail and how custom integrations can be implemented with WordPress. Content Management Systems have made it easy-peasy to develop a web-site of any organization within a short period of time without having any knowledge of programming. There are several Content Management Systems which has great popularity. Some of them are WordPress, Drupal, Joomla. In this entire study I have focused on WordPress as it is quite famous among the other Content Management Systems (CMS).</p> <p>The main theme of the study was to explain about WordPress in detail and the security issues or checks needed before launching a WordPress web-site. Additionally, the paper also discuss about how a custom integration can be implemented with WordPress. For instance, a custom integration has been elaborated which was to connect a contact form of a WordPress powered web-site to a database of Customer relationship management system (CRM) and sending the submitted contact form data to that database.</p>	
Keywords	Wordpress, Drupal, Joomla, jQuery, PHP, typeahead.js, Customer relationship management system (CRM)

Contents

1	Introduction	1
2	What is Content management system (CMS)?	1
2.1	Business advantages of implementing CMS	1
2.2	Structure of CMS	2
3	What is Wordpress?	4
3.1	History of WordPress	4
3.2	Comparison of WordPress with Drupal and Joomla	4
3.3	Why WordPress is the best?	4
4	WordPress in Details	11
4.1	Properties of WordPress	11
4.2	WordPress System Architecture	13
4.2.1	User roles in WordPress	13
4.2.2	Use cases	14
4.2.3	Architecture	17
4.2.4	File architecture	19
4.2.5	Work flow of a WordPress request	23
4.2.6	Architecture of a WordPress theme	23
5	Security checks before launching a WordPress web-site	28
5.1	Hosting	
5.1.1	Enable HTTP secure communication protocol for the entire web-site	28
5.2	User management	29
5.2.1	Admin access limitation	29
5.3	Authentication	29
5.3.1	Two-step authentication	29
5.3.2	Protecting administrator and login page with password	29
5.3.3	Secured and strong admin dashboard password	29
5.3.4	Limit login attempts	30

5.4	WordPress core, themes and plug-ins	30
5.4.1	Latest version of WordPress	30
5.4.2	Latest version of themes and plug-ins	30
5.4.3	Staging environment	30
5.4.4	Non-essential plug-ins	30
5.4.5	Use popular themes and plug-ins	31
5.4.6	Unique WordPress database prefix	31
5.4.7	Security questions while logging in	31
5.4.8	Restrict direct access to plug-in and theme PHP files	31
5.4.9	Disabling directory indexing and browsing	32
5.4.10	Disabling php file execution in WordPress directories	32
5.4.11	Disabling file editing through WordPress editor	32
5.4.12	Disabling WordPress XML-RPC	33
5.4.13	Changing url for admin dashboard login	33
5.4.14	Hiding WordPress version number	33
5.4.15	Logging out Idle users	34
5.4.16	Delete unused plug-ins and themes	34
5.4.17	Changing file permissions	34
5.4.18	Keep WordPress updated	34
5.5	Server Administration	35
5.5.1	Connecting the server using VPN (Virtual Private Network)	35
5.5.2	Protecting the wp-config.php	35
5.5.3	Real time backup of WordPress web-site	35
5.5.4	Assigning strong password to the database	36
5.5.5	Using security plug-in	36
6	Connecting and sending web-site contact form data to Customer relationship management system (CRM) on submit	36
6.1	Purpose	36
6.2	Design and phases	36
6.2.1	Phase 1	37
6.2.1.1	Steps for implementing phase 1	41

6.2.1.2 Testing of phase 1	50
6.2.2 Phase 2	52
6.2.2.1 Steps for implementing phase 2	52
6.2.2.2 Testing of phase 2	55
7 Conclusion	58
References	60
Appendices	
Appendix 1. Contact form 7 code for Isännöitsijäpalvelu Oy's web-site contact form	
Appendix 2. Code to view the 'Taloyhtiöt' custom post type to the HTTP endpoint of the WordPress Rest API	
Appendix 3. Function to enqueue css style and javascript files to the WordPress theme of Isännöitsijäpalvelu Oy in the functions.php file	
Appendix 4. Code of typeheadIntegration.js file	
Appendix 5. Code which enables sending the contact form data to the CRM of the customer	

Abbreviations and terms

ACF	Advanced custom field
API	Application Programming Interface
CRM	Customer relationship management system
CMS	Content management system
CSS	Cascading Style Sheet
DHTML	Dynamic Hypertext Markup Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IT	Information Technology
json	JavaScript Object Notation
MVC	Model-view-controller
PHP	Hypertext Preprocessor
SEO	Search engine optimization
Sql	Structured query language
UI	User interface
url	Uniform Resource Locator
WCMS or WCM	Web Content Management System
VPN	Virtual Private Network
XHTML	Extensible Hypertext Markup Language
XML-RPC	Extensible Markup Language-Remote Procedure Call

1 Introduction

Nowadays, a web-site is an essential part of an organization which makes a great portion of contribution in boosting a business. Obviously, developing a web-site means messing around with codes. However, the tradition has changed after the invention of content management systems (CMS). Content management systems (CMS) have made it easy-peasy to create, edit and update the contents of a web-site. Content management systems have made it possible for users to maintain a web-site without having any knowledge of web development.

This entire study focuses on WordPress which has become the leading content management system (CMS) among the others. Additionally, the thesis also elaborates the essential security checks required before launching a WordPress web-site and the mechanism of implementing custom integrations within WordPress such as a custom integration has been discussed in the paper which elaborates the process of connecting a contact form of a WordPress web-site to a customer relationship management system (CRM) and sending the submitted contact form data to the database of the CRM.

2 What is content management system (CMS)?

A content management system (CMS) is regarded as the triumph of almost all web-sites. This perplexing system allows the users to create, manage, distribute and publish information very easily. Moreover, it covers the lifecycle of the pages on a web-site from creating the content with different tools to publishing and archiving. It also gives users full power to control the hierarchy of the web-site and decorate the view of the pages. In the case of managing web content, content management system (CMS) is also known as Web Content Management System (WCMS or WCM). [1]

2.1 Business advantages of implementing CMS

Implementation of CMS aggregate lots of business advantages which includes efficient authorization process, prominent consistency, upgraded site navigation, improved security, decreased content duplication, increased growth capacity, decreased maintenance expense and greater site flexibility. [1]

Beyond all of these, the main advantage that CMS can offer is to assist one's business targets and plans. For instance, CMS can palliate to enlarge sales, user contentment and also help to communicate with customers. [1]

2.2 Structure of CMS

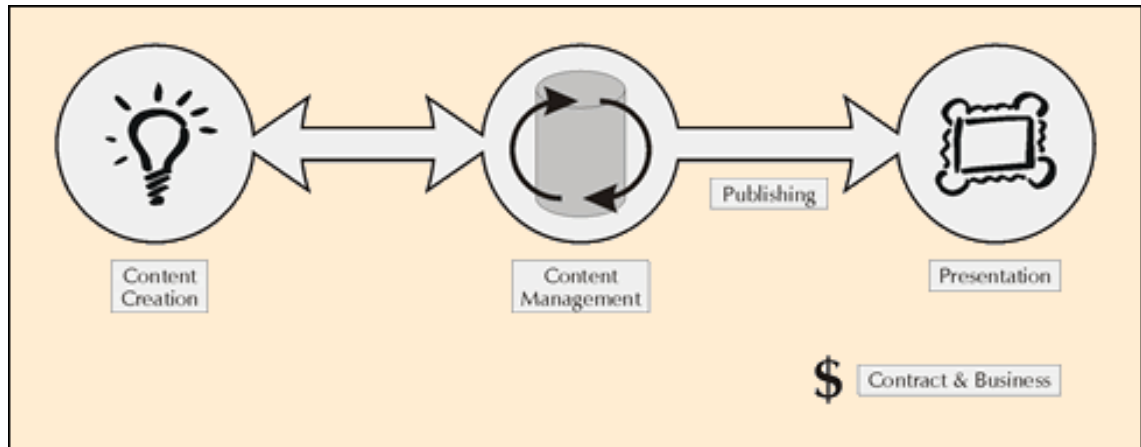


Figure 1: Structure of CMS. [1]

The functions of a content management system (CMS) can be divided into four categories:

1. Content creation
2. Management of content
3. Publishing page
4. Presentation of content

1. Content creation:

This phase of CMS involves creating pages and writing contents with few clicks of your mouse and keyboard which does not require any knowledge of HTML stuffs. As we have already discussed that CMS also offers the ability to manage the structure of a web-site and this is where the pages actually live and linked with each other. Some CMS offers drag and drop functionality to construct the site which decreases the risk of breaking any hyperlinks. [1]

Nowadays, nearly all CMS has its own user interface for creating contents which makes the implementation facile and allows updating contents from anywhere in the world. [1]

2. *Management of content:*

In this phase of CMS, the pages are stored into the main repository which contains the contents of the pages and other vital data. [1]

The main repository has bunch of beneficial attributes that is offered by the CMS:

- Holding data about different revisions of pages and information about the author who made the changes and when. [1]
- Checking that all users are working on their own part of the web-site that they are assigned to make changes. [1]
- Integration with available data sources and IT systems. [1]

All in all, CMS offers step-by-step workflow abilities. For instance, when a page is created by a user, it notifies the person responsible for approval as well as the IT team for final review of the content. Finally, the page is published automatically after the final review and approval. In every step of the workflow, CMS controls the page updates, notifies the responsible person and intensifies any task if needed. Most importantly, the workflow features offered by CMS enable many users to work together in maintaining a web-site which ensures good standard, perfection and constancy of the contents. [1]

3. *Publishing page:*

Finally, the page is ready to get live when the reviewed and approved contents are stored in the main repository. [1]

Content management systems (CMS) possess robust page publishing mechanism which applies the page design and styles by itself when it is published. Obviously, every web-site does not look alike and that is why CMS allows the designers and developers to decide on the page appearance by adding CSS, jQuery, JavaScript etc. which maintains the constancy and high quality visual presentation throughout the web-site. Moreover, this astonishing feature let content writers to focus only on creating unique contents for the web-site. [1]

4. *Presentation of content:*

Content management systems (CMS) offer bunch of functionalities for boosting the standard and efficiency of a web-site. For example, content management system assembles the navigation of the site by reviewing the anatomy directly from the content repo which increases support for range of browsers. To summarize, CMS helps to create dynamic and interactive web-sites which intensifies the impression of the site. [1]

3 **What is WordPress?**

WordPress is a popular open source content management system (CMS) which is written entirely on PHP and uses MySQL to manage the database. WordPress was released in 2003 and since then it has become the leading content management system on the web. Initially, WordPress was known as a blogging tool but now it has emerged as a technology which helps to develop and manage web-sites as well as online stores without having any knowledge of programming and designing expertise. Currently, above 25% of the web-sites are built using WordPress and liked by thousands of people as it is as easy as pie to learn and maintain. [2]

3.1 History of WordPress

The history of WordPress is a great example of dedication and contribution of open source supporters who came up with an astonishing technology. WordPress is managed, developed and improved by a group of programmers and users. WordPress was initiated after the termination of upgrades of a blogging application called b2/cafelog. In the year 2003, couple of end users of b2/cafelog, Mike Little and Matt, settled down on developing a new blogging application besides b2/cafelog. The initial version of WordPress was released on 27th May 2003 which grasped attention of many blogging application freaks. The initial version was mainly an upgraded b2/cafelog which included an admin panel, new templates and engendered XHTML 1.1 templates. Figure 2 is a glance of the initial version of WordPress. [2]

Figure 2: Initial version of WordPress. [2]

In 2004, WordPress 1.2 was released which offered Plug-in Architecture enabling users to build their own plug-ins to increase WordPress functionalities. This version of WordPress enlarged the acceptance ratio of the technology. [2]

Figure 3: WordPress 1.2. [2]

WordPress started to get improved with growing number of users. Afterwards WordPress 1.5 was launched in the year 2005 which provided page creation functionalities, commenting functions and a Theming system. [2]

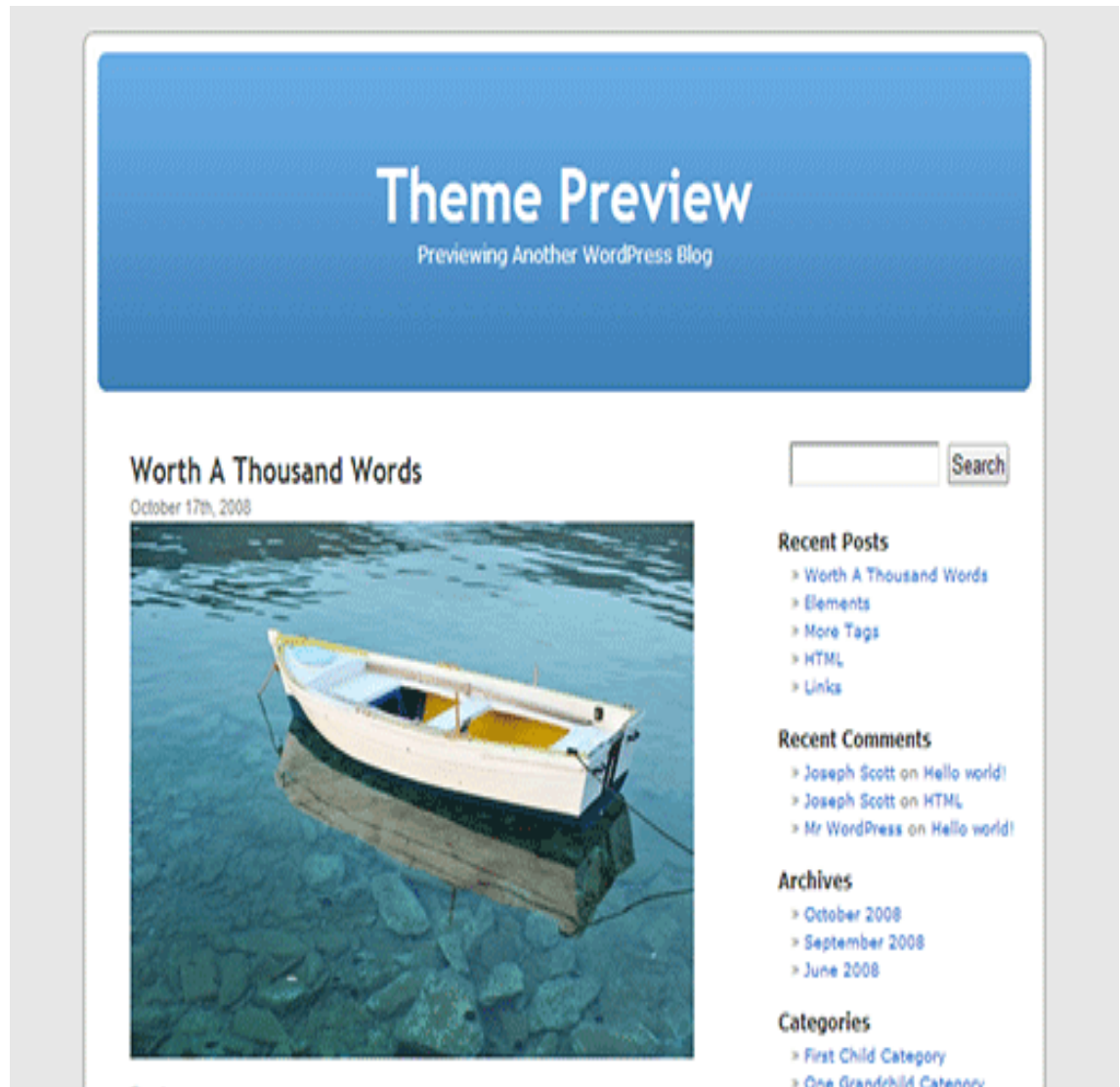


Figure 4: WordPress 1.5 with Theme system. [2]

WordPress 2.0 which was introduced at the end of 2005 came up with improvements of admin dashboard which was developed with JavaScript and DHTML in order to improve user experience. Moreover, this release of WordPress also included an anti-spam plug-in called Akismet and addition of functions.php file in the Theming system. [2]

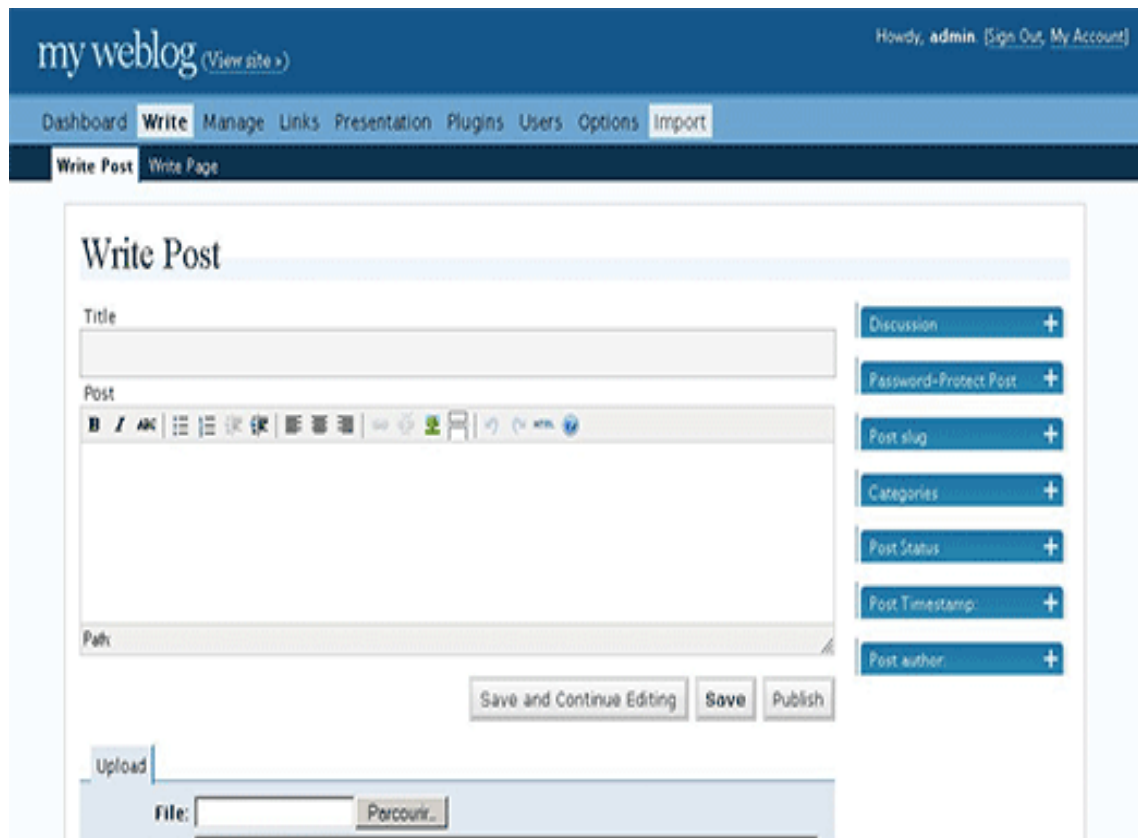


Figure 5: WordPress 2.0. [2]

After the release of WordPress 2.0, there were several releases of WordPress which featured shortcodes, one-click update and capability to install integrated plug-ins. However, the launch of WordPress 3.0 in 2010 was the utmost direction of WordPress towards content management system (CMS). This release came up with bunch of functionalities like custom post types, improved custom taxonomies, custom background, header, menus and so on. Afterwards, there were several releases of WordPress which came up with post formats, admin bar, theme customizing tool, theme previewer, media manager and responsive WordPress admin panel which made it easy for users to manage a web-site on mobile screens and tablets. The latest WordPress version is 3.9 which was introduced on April 2014 with addition of previewing functionality of live widget, audio playlist and few other improvements. [2]



Figure 6: Responsive WordPress admin panel. [2]

3.2 Comparison of WordPress with Drupal and Joomla

There are several content management systems (CMS) in the market but WordPress, Joomla and Drupal are regarded as the notable competitors in the field. These three top notch technologies have many similarities. Most importantly, they are all open source and being developed using PHP and their databases are managed by MySQL. Lastly, they all have theming and templating system to manage the view of the web-site and also offer extending functionalities using plug-ins. As we have seen that there are plenty of similarities between WordPress, Joomla and Drupal, they also differ in many major cases like handling themes and plug-ins, security issues etc. [3]

	WordPress	Joomla	Drupal
Usability	It is super flexible to create a web-site with WordPress without having any web development skills. Moreover, the user interface is also straightforward.	Joomla is a bit difficult in terms of usability. It requires some time to get used to with the technology for a beginner.	Drupal is the most complex among the three. The user interface for creating contents is uncomplicated while it is quite complicated in terms of custom UI development.
Properties	WordPress offers great tools for web developers for developing custom web-sites.	Joomla is preferred for users who require functionalities for social networking.	Drupal offers robust taxonomy function enabling the capabilities to manage complicated contents.
One-click installation	WordPress offers one-click installation.	Joomla offers one-click installation.	Drupal offers one-click installation.
Theme availability	Over two thousand WordPress themes available in the market.	Over one thousand Joomla themes are available in the market.	Over eighteen-hundred Drupal themes are available in the market.
Plug-in availability	Over forty-five thousand WordPress	Over thirty-two thousand mod-	Over seven thousand extensions

	Press plug-ins are available online.	ules for Joomla are available online.	of Drupal are available online.
Functionalities for ecommerce	Ecommerce in WordPress web-sites can be implemented using WooCommerce plug-in.	There is availability of modules to implement ecommerce in Joomla web-sites.	There is availability of plug-ins to implement ecommerce in Drupal web-sites.
Search engine optimization capabilities (SEO)	WordPress is the best for search engine optimization (SEO).	Joomla offers elementary search engine optimization facilities.	Drupal offers average level of search engine optimization capabilities.
Responsiveness	WordPress offers functionalities to create responsive web-sites.	Joomla offers functionalities to create responsive web-sites.	Drupal offers functionalities to create responsive web-sites.
Loading speed	WordPress has good capabilities for tackling web traffic but still depends on the hosting plan of the consumer. The better the hosting service the better the speed.	Joomla requires regular tweaking to increase the speed.	Drupal is the best choice for tackling web traffic and improved loading speed than Joomla and WordPress.

Figure 7: Comparison of WordPress, Joomla and Drupal. [3]

3.3 Why WordPress is the best?

Nowadays, WordPress is used by well-known organizations and business firms. There are sundry of grounds why WordPress should be used for building web-sites. WordPress is open source which means it is free of cost and many developers are always contributing their time on updating the code for WordPress to make it more efficient. Moreover, web-sites developed with WordPress are easy to maintain and update. WordPress is versatile in improving the visual appearance and implementing different features by using WordPress plug-ins and themes. Additionally, WordPress has good support as it is famous among the developers. WordPress is convenient for Search engine optimization (SEO) which makes it easy to enhance search engine results. Lastly, WordPress offers easy to maintain content mechanism and feasibility to integrate contents from different systems. [4]

4 WordPress in Details

4.1 Properties of WordPress

Almost over 25% of the web-sites are powered by WordPress because of its loads of handy and awe-inspiring features as discussed below:

Role-based access control: Different users can be assigned different roles so that each of the users can have access only to their part of responsibility.

Numerous authoring system: WordPress supports numerous users to create posts or contents depending on their role. For instance, if a user who is assigned minimal amount of rights, creates post or content and wants to make it live then a request is sent to the admin for approval and publishing.

Migration: WordPress web-sites can be easily migrated to another server by just copying the files to the target system's public_html folder and importing the sql database to the new server's database. Moreover, WordPress also offers some plug-ins which makes the migrating trouble-free.

Implementation and updating: Installing WordPress on any system is very facile which takes few minutes. Furthermore, upgrading WordPress is automated because of its automatic upgrading feature.

Translation management: WordPress comes with multiple language support. However, there are also plug-ins which can extend the translation management with different functionalities. For instance, developing a web-site with many languages.

RSS feeds: WordPress has the capability to output RSS feeds in several formats.

Search engine support and permanent links: Link of pages, posts, categories etc. are straightforward and excellent for being in the top results of search engines.

Compatibility with different browsers: HTML codes compiled by WordPress satisfy the HTML guidelines of the World Wide Web Consortium (W3C) which assures that web-sites powered by WordPress are supported by all browsers.

WordPress archives and search engine: Posts in WordPress are chronologically listed in the archive from where they can be viewed and also posts can be searched using the search engine inside WordPress.

WordPress post category: WordPress posts can be organized by using the category feature of WordPress which enable users to assign posts to different categories and find it easily.

Content protection with password: WordPress offers great feature for locking down contents with password so that they can be viewed or edited by limited number of users.

Media facilities: WordPress enable users to include images and audio files while creating the post or page making it look attractive and informative.

Customizing features: Plug-in and theming system of WordPress is one of the most astonishing feature of the technology which makes it facile to develop custom web-sites and adding functionalities utilizing the availability of miscellany of themes and

plug-ins online. Moreover, WordPress has its own editor which allows editing the theme files on any browser.

XML-RPC: WordPress has an interface for XML-RPC making it possible for different applications to couple with the web-site.

4.2 WordPress System Architecture

4.2.1 User roles in WordPress

There are seven user roles in WordPress which offers different rights for the users. This allows the users to get access only to their part of responsibility. The seven user roles are visitor, subscriber, contributor, author, editor, administrator, super administrator as in figure 8.

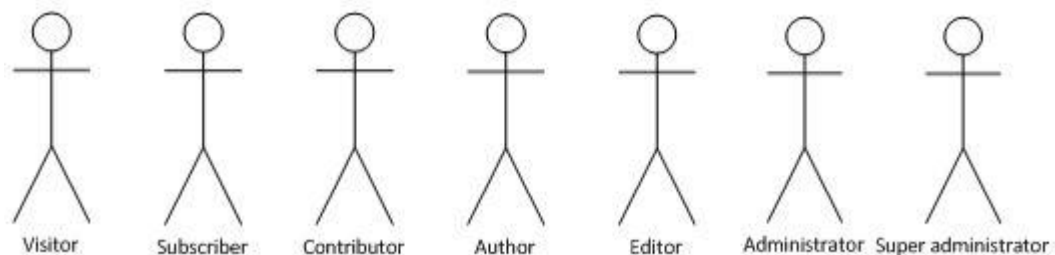


Figure 8: User roles in WordPress.

Visitor: A user who have rights only to view the web-site.

Subscriber: A user who have the rights to control only his profile.

Contributor: A user who have the rights to create posts and handle them. However, contributor does not have any rights to make the posts live.

Author: A user who have the rights to make posts live and handle the posts.

Editor: A user who has the rights to make posts live, handle the posts and pages. Moreover, an editor also has the ability to control posts of different users.

Administrator: The admin functionalities are enabled for administrator role in WordPress who can manage the whole web-site.

Super administrator: A user who have the rights to access the network admin functionalities as well as other admin functionalities.

4.2.2 Use cases

Use case of a visitor:

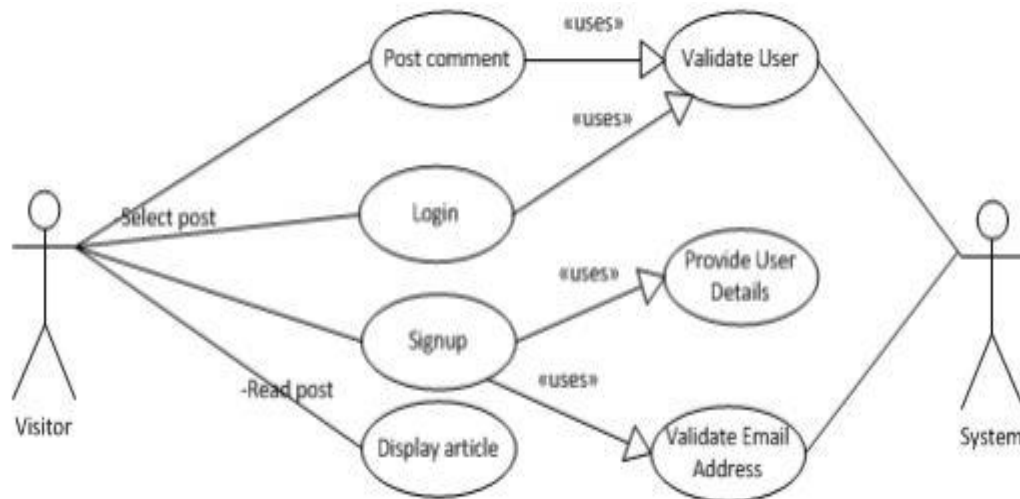


Figure 9: Use case of a visitor.

Use case of a subscriber:

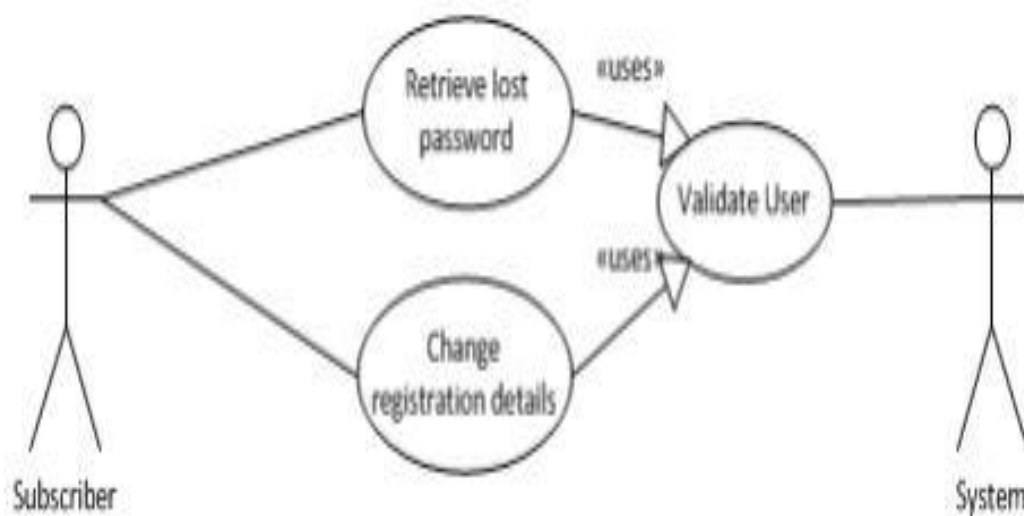


Figure 10: Use case of a subscriber.

Use case of a contributor:

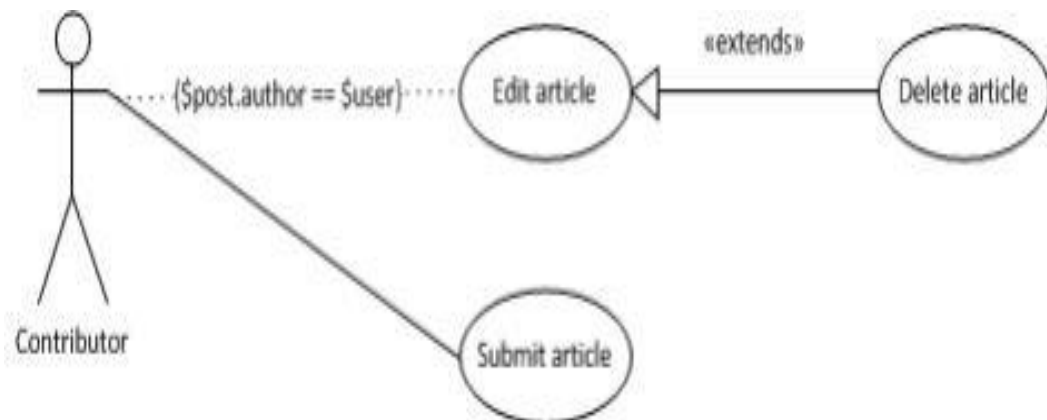


Figure 11: Use case of a contributor.

Use case of an author:

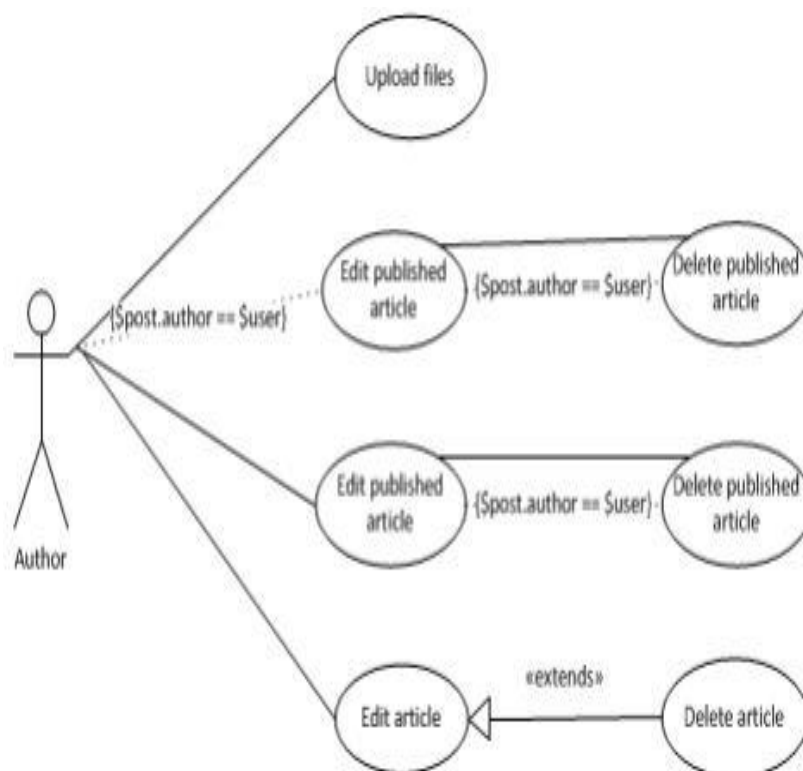


Figure 12: Use case of an author.

Use case of an editor:

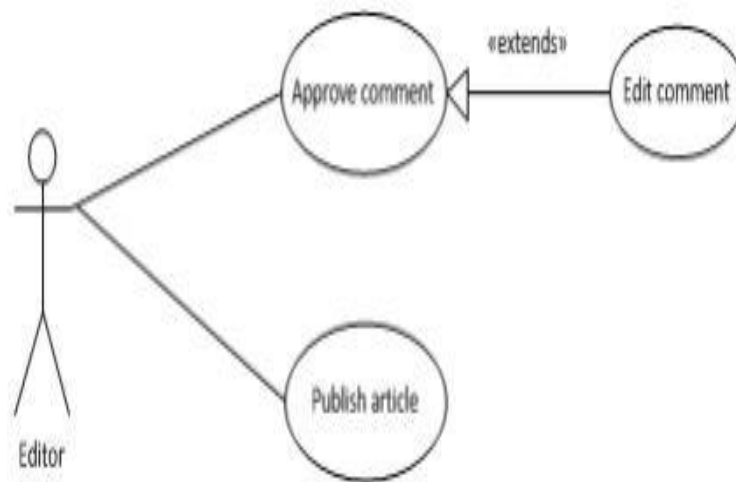


Figure 13: Use case of an editor.

Use case of an administrator:

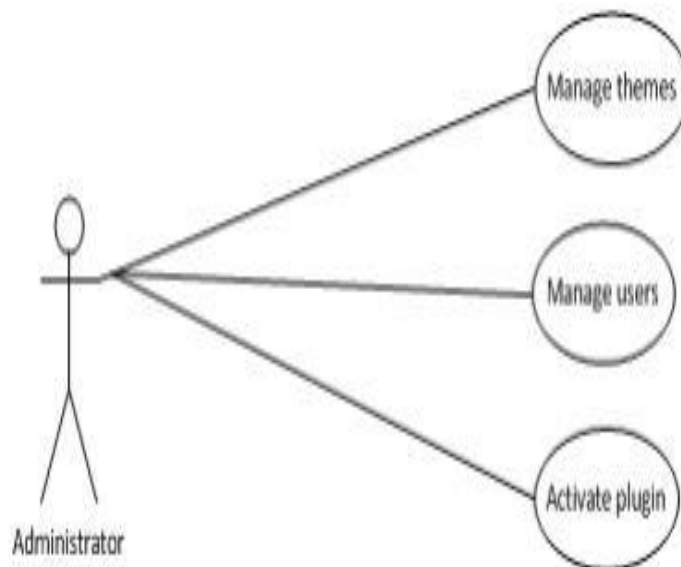


Figure 14: Use case of an administrator.

Use case of a super administrator:

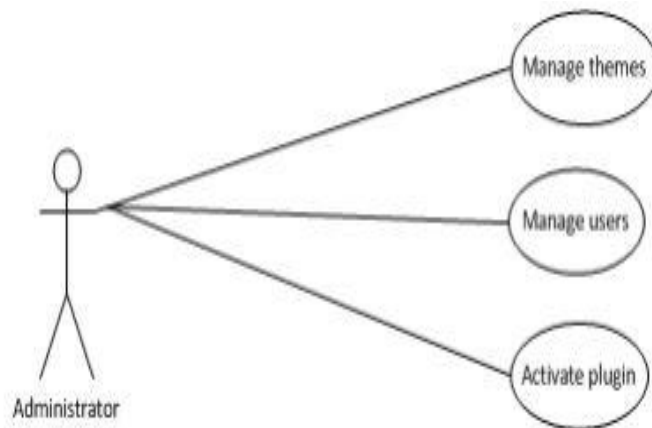


Figure 15: Use case of a super administrator.

4.2.3 Architecture

WordPress follows a systematic and organized mechanism for development. WordPress cannot be regarded as a Model-view-controller (MVC) technology. However, it follows the MVC pattern till certain extent. The model contains the PHP files which are responsible for applying database queries. The central section incorporates the logic and responsible for content management. Lastly, the view part is responsible for the visual appearance of the pages. According to MVC rule, there should be a complete separation between model, view and controller but WordPress does not follow this rule at certain extent because the database queries can be viewed from the view section in WordPress. The structure of WordPress is made up of four parts as follows:

View:

The view contains different parts of a page such as header, footer, sidebar which can be manipulated for advanced personalization.

Templates:

These are PHP files that contain the contents of different parts of a page such as header, footer and sidebar. Each page should derive the contents from the template files for viewing it accurately. As an example, the contents of homepage are derived from index.php, header.php, footer.php and sidebar.php files. All in all,

these templating files with stylesheets and javascript files together constitute a WordPress theme.

The loop:

The loop is a technique in WordPress used for portraying pages which retrieves the posts from the database and views it properly to a certain page.

Backend:

The backend layer controls the transmission with the database server, files, mail systems etc.

Metadata:

Metadata are the data of different pages which are required while classifying the pages. However, these data are received from the database while viewing the pages. For instance, a post's author name, post creation time are regarded as the metadata of the post.

The codes of WordPress follows object oriented pattern up to certain extent. There are sometimes utilization of PHP classes and instances of inheritance, encapsulation and abstraction. As WordPress codes use PHP classes so it is quite reasonable to proof that WordPress follows object oriented pattern. Figure 16 is a class diagram which explains several widgets and their coding style.

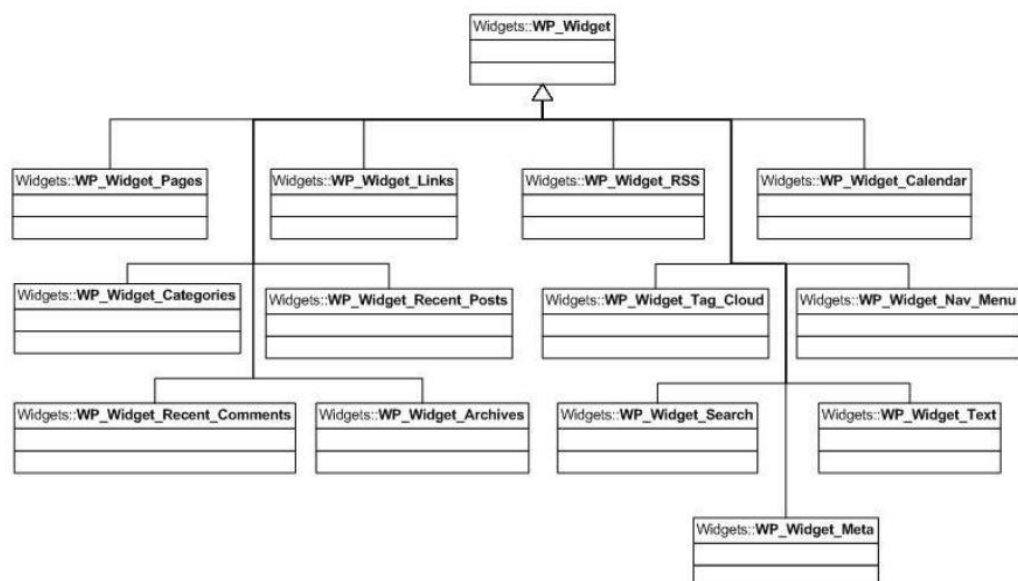


Figure 16: WordPress widget class diagram.

4.2.4 File architecture

The WordPress files and folders are arranged in a systematic manner. The principal WordPress files and folders are:

wp-admin (folder), wp-content (folder), wp-includes (folder), index.php, license.txt, readme.html, wp-activate.php, wp-blog-header.php, wp-comments-post.php, wp-config-sample.php, wp-config.php, wp-cron.php, wp-links-opml.php, wp-load.php, wp-login.php, wp-mail.php, wp-settings.php, wp-signup.php, wp-trackback.php, .htaccess, xmlrpc.php. [5]

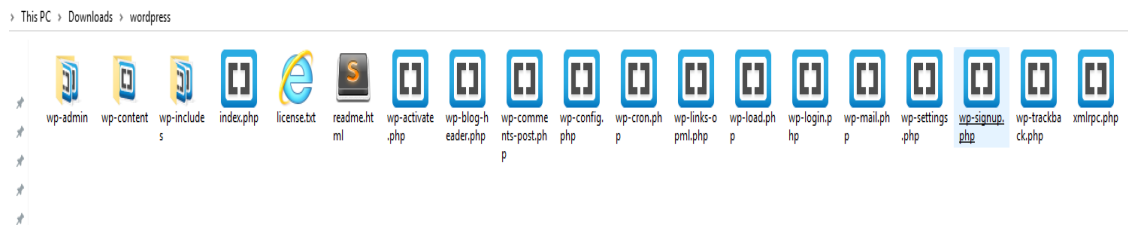


Figure 17: WordPress files and directories.

wp-admin (folder):

The wp-admin is one of the core directories of WordPress which provides all functionalities related to administration. There are many important files and folders inside the wp-admin directory but among all of them admin.php is the major one which manages the communication with the database, views admin dashboard and deals with some essential admin functions like ensuring the user roles. Figure 18 presents the whole idea of the files and folders in wp-admin directory. [5]

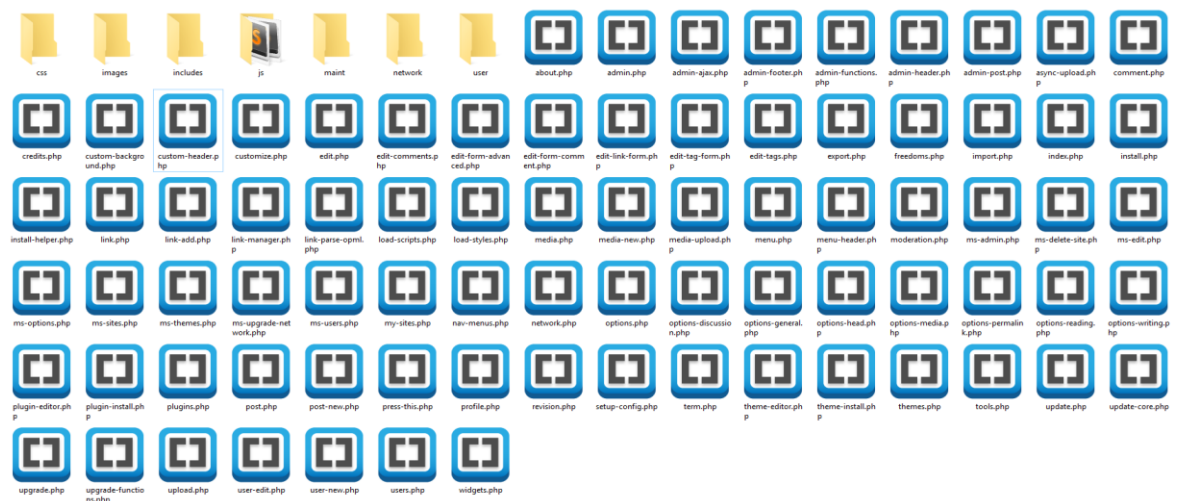


Figure 18: wp-admin directory structure.

wp-content (folder):

The wp-content directory deals with extending functionalities using plug-ins and managing the visual appearance of a WordPress web-site using themes. This directory contains the plug-ins, themes and uploads folders as shown in figure 19. [5]

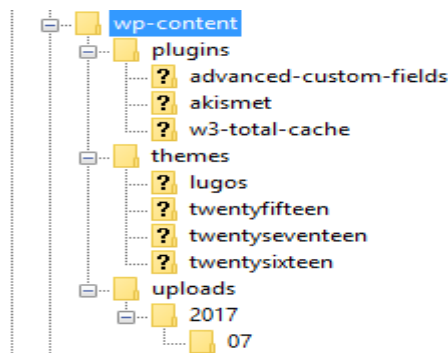


Figure 19: wp-content directory structure.

As shown in figure 19, the WordPress themes are stored in the themes folder. Default installation of WordPress comes with three themes which are twentyfifteen, twentyseventeen and twentysixteen. However, there is no limitation of installing more themes which means WordPress allows users to install as many themes as they require but activating one theme at a time. Moreover, WordPress requires a theme to manage the visual appearance of a web-site and that is why the themes directory should not be blank. [5]

In figure 19, it is also defined that the plug-ins folder contains the WordPress plug-ins used to extend functionalities. However, it is not mandatory to use plug-ins because a WordPress web-site can be powered without any plug-ins but it improves a web-site's performance if only the obligatory plug-ins are used. [5]

Lastly, the uploads folder accommodates the media files that is uploaded using the WordPress media uploading function. As seen in figure 19, the uploaded media files are arranged according to the year and month. Media files that can be uploaded using the WordPress uploading feature are pdf, mp3, video, pictures. [5]

wp-includes (folder):

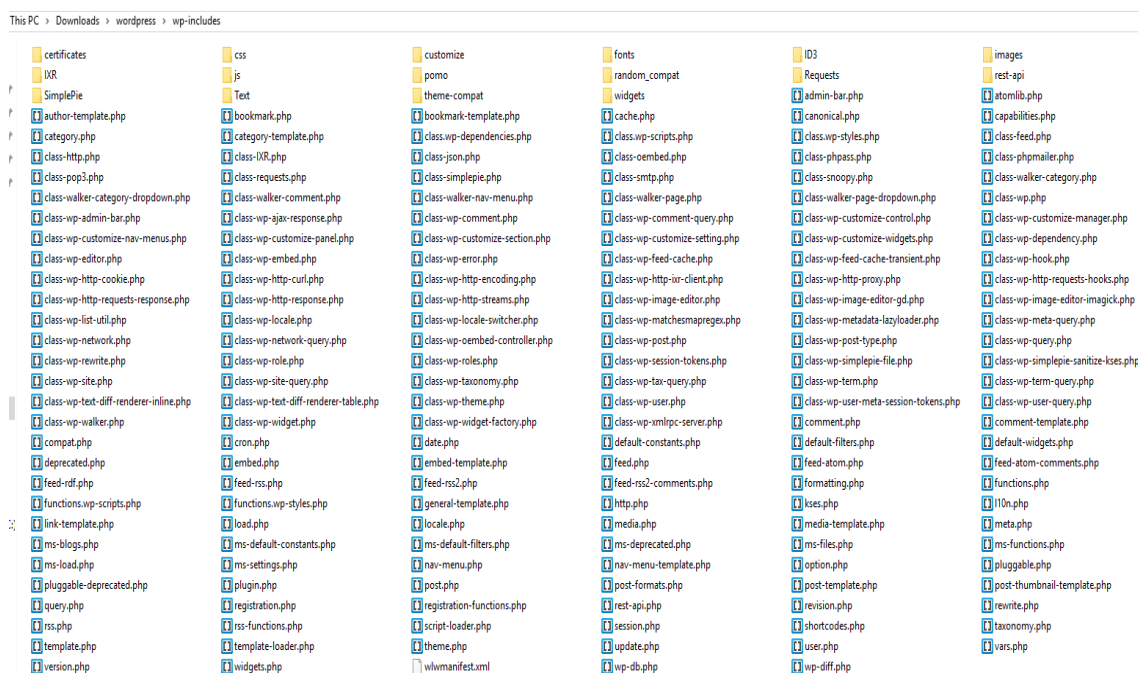


Figure 20: wp-includes directory structure.

The wp-includes is the largest core directory which contains the most important core files that are obligatory for turning on a WordPress website. Default installation of WordPress comes with more than one-hundred files in the wp-includes folder as well as fourteen other sub-directories which contains certificates, javascripts, css, images, widgets etc. as it is shown in figure 20. The sub-directories of wp-includes folder are not as vital as the files in the wp-includes folder. For instance, the functions.php file executes a bunch of different functions that helps to undergo the installation of WordPress properly. [5]

index.php:

When a web-page is requested, the index.php file executes and runs the WordPress files required for loading the page. [5]

license.txt:

As its name suggests, license.txt contains the license information of WordPress. [5]

readme.html:

This file carries some guidelines for installing and using WordPress. [5]

wp-activate.php:

As its name suggests, the file activates and executes some functions. Functions like `do_activate_header`, `activate_wp_head`, `wpmu_activate_stylesheet` and `activate_header` are executed here. [5]

do_activate_header():

This function applies an action that is executed on `wp_head()`. [5]

activate_wp_head:

This function is executed on the action in `wp_head()` before loading the site activation page. [5]

wpmu_activate_stylesheet:

This runs the stylesheets required for a certain page. [5]

activate_header:

This is an action which is executed before loading the site activation page. [5]

wp-blog-header.php:

Http headers are defined in this `wp-blog-header.php` file. [5]

wp-config.php:

The `wp-config.php` file is one of the most important file which holds data about the database name, host, database password and database user which helps to connect to the database. [5]

xmlrpc.php:

WordPress has an XML-RPC user interface. WordPress has an API which is known as WordPress API where particular functionalities are executed. The API is only utilized when required. The default WordPress installation comes with XML-RPC enabled since version 3.5. [5]

4.2.5 Work flow of a WordPress request

Whenever a user pops in on a WordPress web-site, html is generated dynamically applying the stylesheets and javascripts required for the specific page. The work flow of a WordPress request is combined of five steps as follows:

1. The browser sends a request to the page.
2. The core of WordPress then executes the necessary php codes beginning with the index.php.
3. The WordPress core connects with the database and fetches the required data.
4. After receiving the data from the database, they are merged together along with the enabled plug-ins' data and data from the enabled theme which is then converted to html dynamically.
5. Finally, the html code is executed to the viewer's browser. [6]

Likewise, whenever any post is created, saved or searched then the required steps are executed and stored in the database notifying the admin about the event. The steps are more clearly explained in figure 21. [6]

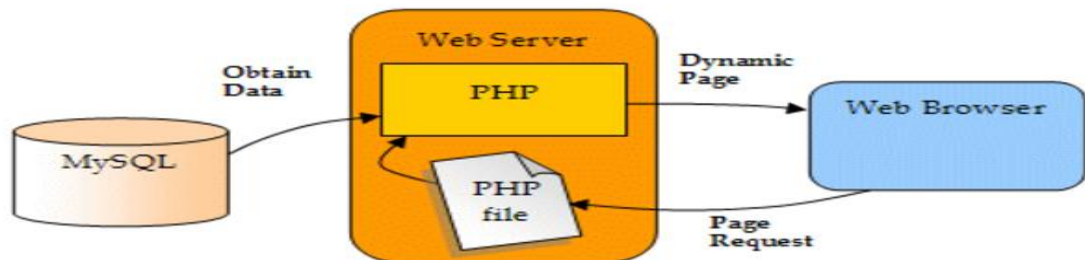


Figure 21: Work flow of a WordPress request. [6]

4.2.6 Architecture of a WordPress theme

WordPress requires a theme to work and that is why it comes with a default theme. However, custom made themes can also be used. A WordPress theme is a combination of several templates, css files, javascripts, pictures and PHP scripts. As discussed in section 4.2.4, the themes are stored in wp-content/themes directory. Figure 22 shows the architecture of a WordPress theme in more details. [7]

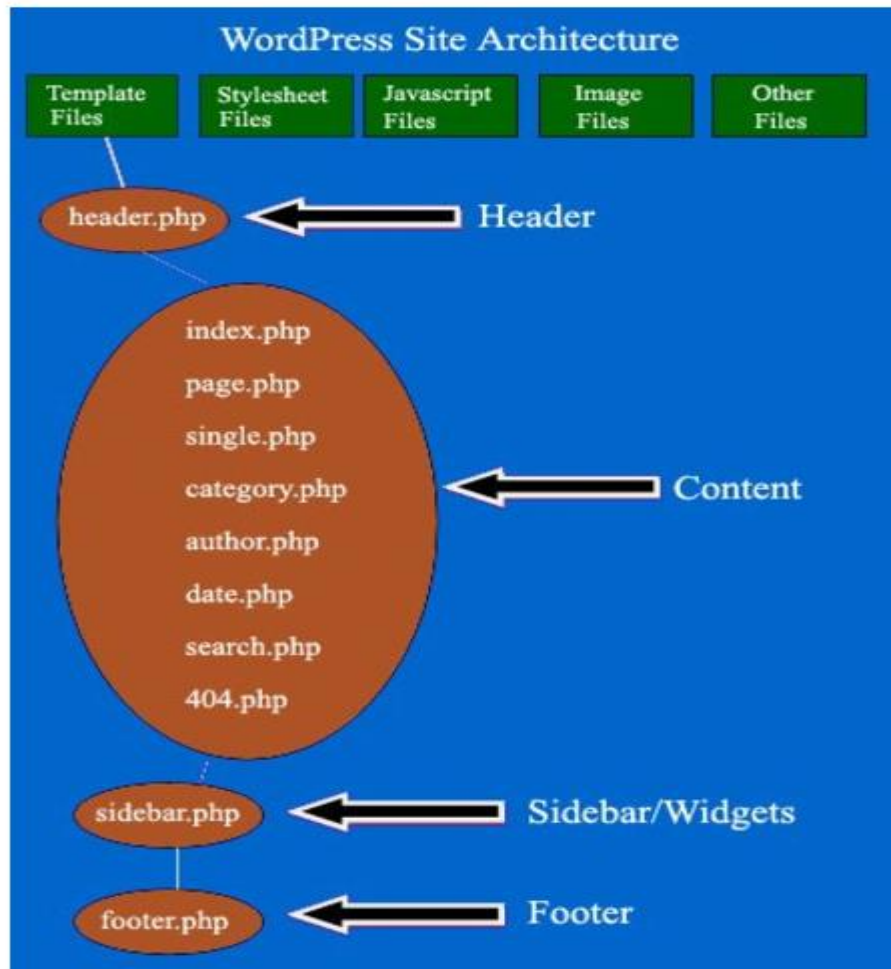


Figure 22: Architecture of WordPress theme. [7]

Templates:

Templates are files with .php extension which are building blocks of a WordPress web-site. Additionally, these templates increase the chance of extending functionalities. A theme with number of template files allows more customizing ability. Template files like header.php and footer.php are mandatory for every web page while the rests are required for certain functionalities. [7]

The main template files that are required in every WordPress theme are described below:

header.php:

As its name suggests, the file contains the template of header and navigation menu. This file can be edited from the WordPress editor inside Appearance menu in admin dashboard as shown in figure 23. [7]

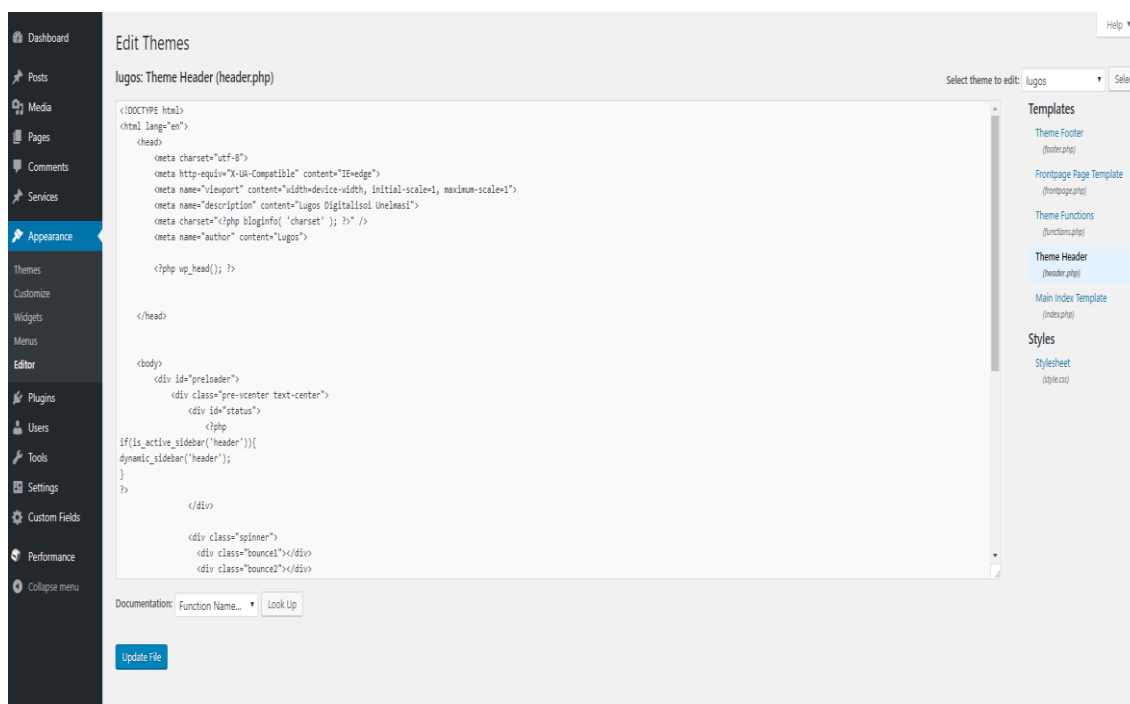


Figure 23: WordPress editor.

index.php:

The index.php executes a loop of PHP for viewing the index of recent blog posts as a list. This post index can also be viewed on the homepage as well as on any other pages. The index.php file can be customized from the WordPress editor as shown in figure 23. [7]

home.php:

This home.php file also executes a loop of PHP for viewing the index of recent blog posts. This post index can also be viewed on the homepage as well as on any other pages. However, if the home.php does not exist in the enabled theme then the core executes the template in index.php file. As usual, this home.php can also be customized using the WordPress editor. [7]

front-page.php:

The front-page.php is the homepage template of a WordPress web-site which can be a fixed web page or the index of blog posts. When a WordPress web-site is loaded, the core looks for the front-page.php file inside the active theme's directory and executes that template. If the theme does not contain the front-page.php then the template of home.php is executed otherwise the template existing in the index.php is used for displaying the front page. This file can be edited using the WordPress editor as well. [7]

page.php:

The title and contents of a page are viewed using this template file. Furthermore, the listicle of comments and form for placing comments are also viewed using the same template. [7]

single.php:

The template executed in single.php is used for viewing the name, contents, name of the writer, creation date, categorization, listicle of comments, comment form of a blog post. This template also includes scripts for viewing a navigation to navigate through all posts. [7]

category.php:

The category.php file is responsible for viewing the pages which include posts classified into different categories. [7]

author.php:

The template in author.php is responsible for managing the view of an author web page. [7]

search.php:

WordPress offers the ability to integrate a search engine in WordPress powered websites and the search.php executes the template which manages the view of the search results. [7]

404.php:

Whenever WordPress core is unable to locate a particular post or a page then it redirects to 404.php which contains a template for 'Not found' searches. [7]

footer.php:

As its name indicates, footer.php is used for templating the footer of pages. It can be edited through WordPress editor for advanced customization. [7]

comments.php:

The visual presentation of the comments is managed by the comment.php file. [7]

sidebar.php:

The sidebar.php is accountable for managing the visual appearance of the sidebar. The constituents of the sidebar are configured through the administrator dashboard. To sum up, the template of this file can be customized using the WordPress editor. [7]

functions.php:

The functions.php is an optional but powerful part of a theme which creates a way to extend functionalities. As it is a theme file so the functionalities added in a certain theme using this file will be disabled after switching to another theme. However, the functions.php file can be created inside the active theme's folder to add more features if the file does not exist. [7]

style.css:

The style.css is the primary css file which manages the visual appearance of the whole theme which can be edited using the WordPress editor for advanced customization of the user interface. [7]

Code reusability:

The template codes in WordPress can be reiterated by loading it to another template. As an example, the single.php can use the template of post-single.php using the code as follows:

```
get_template_part('post', 'single');
```

In contrast, if the post-single.php manages the view of the blog posts constituents then this file should be configured for advanced customization in place of configuring single.php. [7]

5 Security checks before launching a WordPress web-site

Due to emerging hackers, it has become quite essential to integrate great security in applications and web-sites. As a result, it is necessary to buy good hosting plans, secure information with passwords, enable http secure communication protocol in web applications and web-sites etc. The following sub-chapters explain about the security issues in details.

5.1 Hosting

5.1.1 Enable HTTP secure communication protocol for the entire WordPress web-site

When a user sign in to the admin dashboard, the accreditations given while signing in can be blocked if the web-site is not encoded with HTTPS protocol. Additionally, the secret word input field may indicate only circles yet the word can be accessed by everyone over the internet. The hackers are able to acquire the accreditations using several tricks as follows:

1. By tracking wireless connections which are extremely facile to manipulate with the help of some hacking applications.
2. By manipulating the routers over the cyberspace which allows the hackers to view the credentials which are passed through the routers.
3. By acquiring an internal area at a hosting provider. [8]

To sum up, empowering HTTPS protocol in web-sites increments their ranking in google look up because initially google arranges the web-sites with HTTPS in its results. The HTTPS protocol can be enabled by adding in the root directory a .htaccess file containing the piece of code as in figure 24 or by enabling HTTPS from the hosting provider control panel. [8]

```
RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

Figure 24: Code for enabling HTTPS.

5.2 User management

5.2.1 Admin access limitation

Multi-user support is one of the tremendous features of WordPress which allows a team to work together on a web-site. Henceforth, every user is not required to have access to all admin functionalities and that is why it is possible to limit admin access according to the user role. For instance, a developer and a manager of a web-site is required to have accessibility to the entire admin functions whereas a writer or an editor can have less accessibility to admin functions as they will be just writing and updating constituents of pages. [11]

5.3 Authentication

5.3.1 Two-step authentication

It is a good practice to add two-step authentication to WordPress web-sites which reduces the chance of brute force attack. To enumerate, two-step authentication is a way to authenticate users which requires a pass code and sends a secret code as a text message to the respective user's cell phone to make a successful login process. WordPress plug-ins like Clef, Google Authenticator and Duo Two-Factor Authentication makes it facile to integrate this feature in WordPress powered web-sites. [9]

5.3.2 Protecting administrator and login page with password

A cyberpunk can easily access the wp-admin directory and login page which makes it easier for them to hack the entire web-site and retrieve confidential data. As a result, a password authentication can be set up on the back-end or using the .htaccess file which will allow the required users to access the wp-admin directory and the login page. [10]

5.3.3 Secured and strong admin dashboard password

It is a good practice to assign strong password for accessing admin dashboard otherwise hackers may easily break the border and access the admin dashboard. Strong

passwords can be generated using password generators online which is a combination of alphabets, numbers and symbols. [11]

5.3.4 Limit login attempts

The possibility of a web-site getting hacked increases with the opportunity a hacker gets to make a brute force attack. Therefore, it is quite necessary to assign limitation for login attempts from a particular IP address before blocking it which can be implemented by using plugin like Limit Login Attempts. [11]

5.4 WordPress core, themes and plug-ins

5.4.1 Latest version of WordPress

Using the latest version of WordPress ensures that a website is much more secured and updated than previous versions of WordPress.

5.4.2 Latest version of themes and plug-ins

Using the latest version of plug-ins ensures that a website is free from any bugs.

5.4.3 Staging environment

It is important to set up a staging environment so that the updates can be tested before implementing it to the live site which enables to ensure that the updates are working well and will not cause any harm to the live site.

5.4.4 Non-essential plug-ins

Using unnecessary or lots of plugins make it easier for hackers to hack a WordPress website easily. Moreover, plug-ins need to be updated always which may cause incompatibility with other plug-ins if not updated. Lastly, using large amount of plug-ins slowdown the website.

5.4.5 Use popular themes and plugins

It is a great practice to use popular themes and plug-ins because plug-ins and themes from less reputable sites may contain easily exploitable code or just downright malicious content which increase the possibility of a web-site getting hacked. [11]

5.4.6 Unique WordPress database prefix

If a WordPress site is using the default database prefix (wp), then it makes it easier for hackers to guess what your table name is. This is why it should be assigned something unique prefix. For example: wp_a123456_ . [12]

5.4.7 Security questions while logging in

Implementing security questions while logging in to the admin dashboard decreases the chances of uncertified access. However, this security can be enabled by installing WP Security Questions plug-in. After installation, the user has to open Settings » Security Questions page to make the settings as required. [13]

5.4.8 Restrict direct access to plug-in and theme PHP files

There are several grounds for which authorizing undeviating access to PHP files can be harmful. Especially, the files of themes and plug-ins may hold PHP files that are not modelled for direct execution. In contrast, these files may be executing functions that are implemented in another PHP file. As a result, it will engender bugs or warnings which may lead to divulgence of confidential data. Hence, the restriction of access to PHP file can be implemented by placing the code in figure 25 inside the .htaccess file. [14]

```
# Restrict access to PHP files from plugin and theme directories

RewriteCond %{REQUEST_URI} !^/wp-content/plugins/file/to/exclude\.php
RewriteCond %{REQUEST_URI} !^/wp-content/plugins/directory/to/exclude/
RewriteRule wp-content/plugins/(.*)\.php$ - [R=404,L]

RewriteCond %{REQUEST_URI} !^/wp-content/themes/file/to/exclude\.php
RewriteCond %{REQUEST_URI} !^/wp-content/themes/directory/to/exclude/
RewriteRule wp-content/themes/(.*)\.php$ - [R=404,L]
```

Figure 25: Code for restricting direct access to PHP files. [14]

5.4.9 Disabling directory indexing and browsing

When the server is unable to locate index.php or index.html file, the list of directories is viewed which contains list of directory contents. Therefore, the divulgence of the contents is detrimental to the web-site as the data can allow a hacker to make a successful attack on WordPress plug-ins, theme and also the server. The directory indexing and browsing can be disabled by adding a single line of code which is 'Options -Indexes' at the end of .htaccess file. [14]

5.4.10 Disabling php file execution in WordPress directories

One of the great features of WordPress is enabling the users writing or uploading contents. As a result, the upload folder of WordPress should have 'write' permission enabled which arises as a risk when PHP files are uploaded. However, WordPress by default restrict the users from uploading PHP files through the admin dashboard but this can be achieved by the use of a plug-ins or a theme that has the particular capability. As a consequence, uploading spiteful PHP files may lead to destruction of the server. The following piece of code can be added to the .htaccess file in order to get rid of the risk:

```
<Files *.php>
deny from all
</Files>
[14]
```

5.4.11 Disabling file editing through WordPress editor

As it is quite known that WordPress has an integrated editor through which theme and plug-in files can be edited from the admin dashboard. The WordPress editor can offer threats if not used by the right user. Hence, it is required to disable the editing feature by adding the code which is `define('DISALLOW_FILE_EDIT', true);` in the wp-config.php file. [14]

5.4.12 Disabling WordPress XML-PRC

XML-PRC in WordPress allows coupling between a WordPress web-site and different applications. Normal installation of WordPress comes with XML-PRC turned on by default since WordPress 3.5. As XML-PRC is quite potent and that is why it can escalate the probability of brute-force attack. For instance, a cyberpunk inputs password 600 times for logging in to the admin dashboard which is counted as 600 different login attempts. Hence, these attempts can be traced by plug-ins like Limit Login Attempts and block the ip address of the hacker. However the scenario with XML-PRC enabled is different because the use of system.multicall function in XML-PRC can help a cyberpunk to guess hundreds of passwords with just few requests. [10]

As a result, it is recommended to turn off XML-PRC if not required which can be achieved by adding the lines of code below inside the .htaccess file:

```
# Block Wordpressxmlprc.php requests
    <Files xmlprc.php>
        orderdeny,allow
        deny from all
        allow from 123.123.123.123
</Files>
[15]
```

5.4.13 Changing url for admin dashboard login

The default url for login page of admin dashboard of WordPress powered web-sites is www.example.com/wp-admin or www.example.com/wp-login which makes it easy for a hacker to make successful brute-force attacks. Hence, it is a good practice to change the url for admin login by using plug-in like Move Login. [11]

5.4.14 Hiding WordPress version number

Revealing the version number of WordPress makes it easy for a hacker to decide the type of attack they can perform to harm a web-site. Therefore, hiding the version information of WordPress increases the security level which is quite easy to implement by adding few lines of code to the functions.php file as below:

```
functionremove_wordpress_version() {
return "";
}
add_filter('the_generator', 'remove_wordpress_version');
```

[16]

5.4.15 Logging out Idle users

The situation is usual that logged in users may stray away from the computer display which may result as a threat to the security of the web-site. As a consequence, anyone will be able to steal their sessions, manipulate their password and account settings. Hence, using plug-in like Idle User Logout can add a layer of security to a WordPress web-site. [10]

5.4.16 Delete unused plug-ins and themes

Plug-ins and themes that are not being used are not updated as well which makes it facile for hackers to hack web-site information. Therefore, unused plug-ins and themes are recommended to be uninstalled which blocks the hackers to make their attack from this entry point. [12]

5.4.17 Changing file permissions

It is highly recommended to steer clear of assigning 777 permission to files and directories in WordPress which enables the read, write and execute permissions for everyone. In WordPress, it is advised to assign 640 or 644 permission to files and directories and 600 to wp-config.php file. [17]

5.4.18 Keep WordPress updated

Every update of WordPress overhauls any bugs from previous version and comes with additional security features. As a result, keeping WordPress updated increments layer of security of the web-site every time. [14]

5.5 Server Administration

5.5.1 Connecting the server using VPN (Virtual Private Network)

A computer savvy is expected to have the knowledge of using VPN in order to maintain seclusion. To summarize, VPN is a solution which makes a user's browsing information secured when a computer is connected to an open Wi-Fi network. However, there are some web-sites which belong to different parts of the world as well as some government web-sites which restricts requests from VPNs. A VPN can be regarded as a protected passage connecting a computer to the destination which a user requests through the browser. When a user uses a VPN, the computer first establishes a connection with the VPN server situated in a region other than the user's actual location. For instance, the location of the user is Finland and the location of the VPN server is Bangladesh. As a result, the user's location will be visible as Bangladesh which makes it hard for a hacker to exploit the server and the web-site. [18]

5.5.2 Protecting the wp-config.php

The wp-config.php file of WordPress carries lots of highly confidential data like security keys and database credentials which can arise as a threat if obtained by a cyberpunk. Therefore, adding more protection to the wp-config.php file can boost the security level of a WordPress web-site which can be achieved by placing the following code to the .htaccess file:

```
# protect wpconfig.php
<fileswp-config.php>
orderallow,deny
deny from all
</files>
```

[19]

5.5.3 Real time backup of WordPress web-site

Backing up a WordPress web-site every day or periodically is essential which prevents from any damage happening to the data or the web-site because of any faults in the server. As a result, it is recommended to use plug-ins like Vaultpress or Backupbuddy which makes a backup of the entire web-site every day or periodically. [20]

5.5.4 Assigning strong password to the database

As all information is stored in the database so it is highly recommended to set a strong password for the database which is not that easy-peasy to guess. There are several password generators online which can be used to generate strong passwords. Ensuring that the database password is strong also ensures that a WordPress web-site is highly secured.

5.5.5 Using security plug-in

Use of security plug-in like Wordfence or All in one WP security and firewall is regarded as one of the most important security checks as it keeps an eye on the number of exertions made to harm a website. For instance, if there are lots of blocked exertions in the security plug-in dashboard which indicates that someone is trying to hack the website continuously. As a result, it becomes easier to decide on the protection needed to increase the security level of the website. [11]

6 Connecting and sending web-site contact form data to Customer relationship management system (CRM) on submit

6.1 Purpose

Contact form is one of the major parts of a web-site which enables the users to get in contact with the web-site owner. This chapter is based on a task that was assigned while working in Bildy Oy. The goal of this task was to implement a contact form for “Isännöitsijäpalvelu Oy” web-site which connects and sends the submitted contact form data to the company’s CRM database.

6.2 Design and phases

In section 6.1 the main purpose of the task has been discussed. Figure 26 is the overall visual design of the contact form.

Figure 26: Contact form design.

The work consists of two phases which are discussed in the following sections of chapter 6.2.

6.2.1 Phase 1

Isännöitsijäpalvelu Oy is a real estate company which have many properties around Helsinki, Finland. Logically, it is quite easier to remember a name compared to an address. As a result, the first phase of the work was to implement an auto search field in the contact form which gives auto suggestions on typing the name of a particular property and automatically fills up some mandatory fields on select. The web-site of

Isännöitsijäpalvelu Oy is developed with WordPress. The implementation of this phase was succeeded using a combination of plug-ins and jQuery libraries as follows:

Contact form 7:

Contact form 7 is one of the popular plug-ins for WordPress powered web-sites for implementing contact forms offering all types of input fields. The contact form required for this task has been implemented with Contact form 7. Adding input fields to the contact form with contact form 7 is quite straightforward because it supports shortcodes. For instance, adding a text field to a contact form requires the following shortcode to be placed in the contact form editor:

```
[text* Puhelinnumeroid:puh]
```

According to the above shortcode, the “text*” indicates that it is a required text field because of the presence of “*”, the “Puhelinnumero” is the name of the field and the “id:puh” indicates that the id of the field is “puh”. Similarly, a textarea field can be assigned with the shortcode: [textarea* KuvausMuutostyoid:kuvausMuu]. All in all, other types of fields also follow the similar kind of pattern. Moreover, the contact form 7 editor also supports html codes but it is always better to use shortcodes while adding input fields.

Typeahead.js:

Typeahead.js is a jQuery plug-in which helps to implement an auto suggestion input field with advanced customization as required. The features that make typeahead.js demandable are as follows:

- Typeahead makes it easy to implement functionality for viewing suggestions as the users start to type.
- Typeahead offers the ability to manage templates for delivering the most flexible user interface.
- It supports RTL languages and input method editors.
- The required result of the user input is accentuated within the suggestion when matched.
- Typeahead prompts custom events in order to enrich the functionality. [21]

Bloodhound.js:

Bloodhound plugs in the functionality of viewing suggestion to typeahead.js. Bloodhound is powerful, adaptable and offers several high-end features like fetching data, smart caching, quick lookups etc. The features that make bloodhound.js flexible are as follows:

- Bloodhound functions with the data embedded into the source code.
- Bloodhound fetches data while initialization to decrease the waiting time for viewing suggestions on different user inputs.
- Bloodhound smartly utilizes the local storage to minimize the network requests.
- Bloodhound backfires recommendations from remote data.
- Bloodhound limits the rate of sending and receiving data from the remote source and drops-off network requests to minimize the data load. [22]

Advanced custom field:

Advanced custom field plug-in helps to add different input fields such as text, textarea, range, checkbox, radio button, select, file attach, gallery, image, link, date picker, time picker, google map etc fields to the editor screen so that different types of content can be viewed to a page. Figure 27 shows the types of fields that can be added using advanced custom field plug-in. [23]

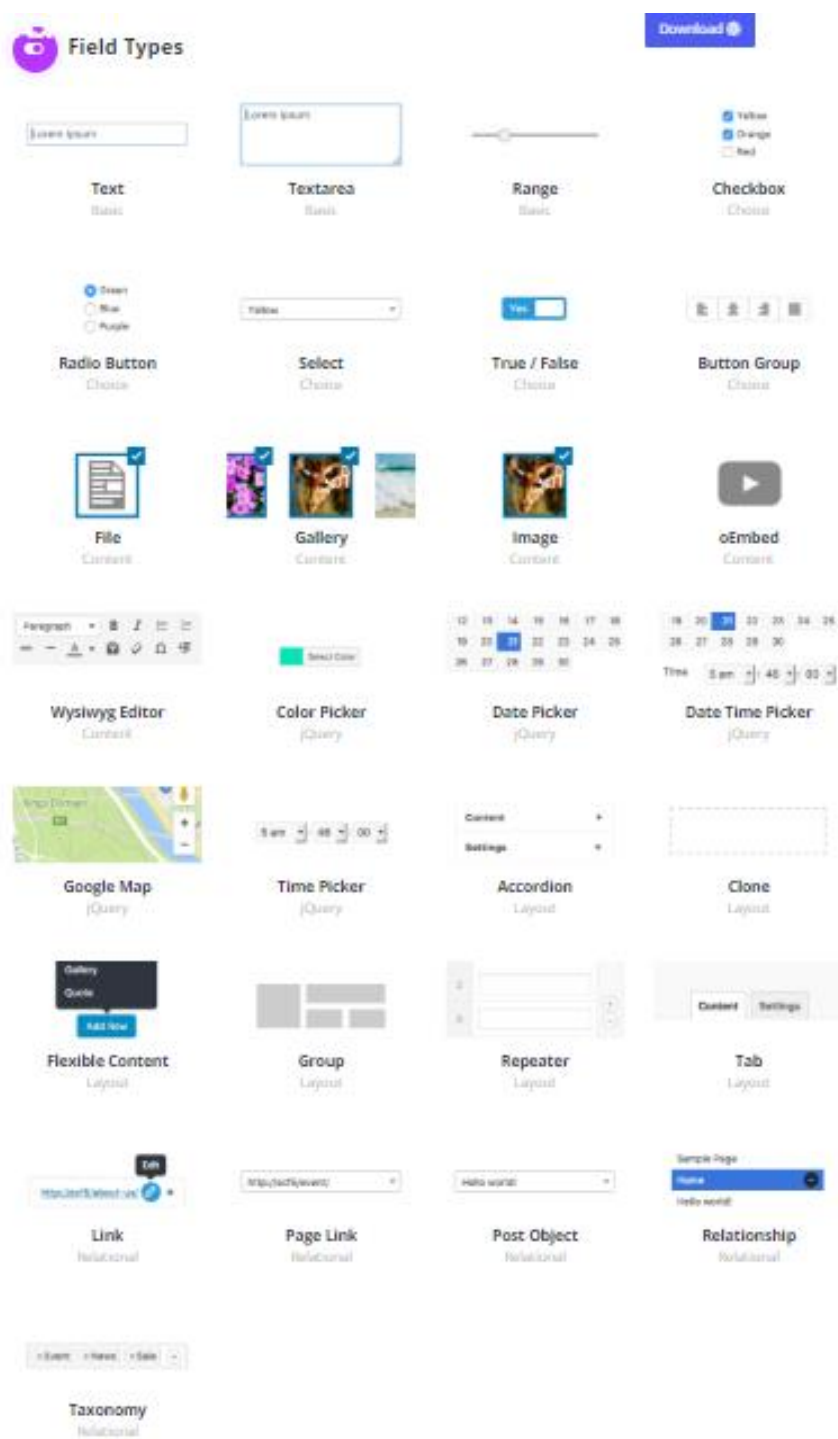


Figure 27: Different types of field offered by advanced custom field plug-in. [23]

Custom Post Type UI:

Custom post type UI plug-in offers a flexible user interface for enlisting and handling custom post types and taxonomies in a WordPress powered website. [24]

WordPress Rest API:

Introduction of WordPress Rest API leading WordPress towards the direction of being an application framework. WordPress Rest API equips feasible integrity of HTTP endpoints which makes it possible to retrieve WordPress powered web-site's data in the form of json which comprises data of users, posts, taxonomies etc. For instance, all posts of a WordPress powered web-site can be accessed in json format by simply sending a GET request by entering the web-site address in the following format:

`http://{web-site name}/wp-json/wp/v2/posts`

[25]

ACF TO WordPress API:

Adding the ACF to WordPress API plug-in to a WordPress web-site allows the advanced custom fields of posts, custom post types etc. to be viewed to the WordPress API JSON output under the key called 'acf'. [26]

6.2.1.1 Steps for implementing phase 1

Phase 1 comprises of six steps which are explained as follows:

Step 1:

The main purpose of phase 1 was to implement a contact form which includes an auto search field which shows suggestion of different properties on user input and automatically fills up some input fields on select. As a result, first step was to develop the contact form which has been done using contact form 7 plug-in. Contact form 7 has been explained in detail in section 6.2.1 of chapter 6. Appendix 1 explains the overall code

that has been used to develop the contact form as shown in figure 26 and figure 28 explains the process of adding the code in Appendix 1 to the editor of contact form 7 plug-in to generate the contact form.

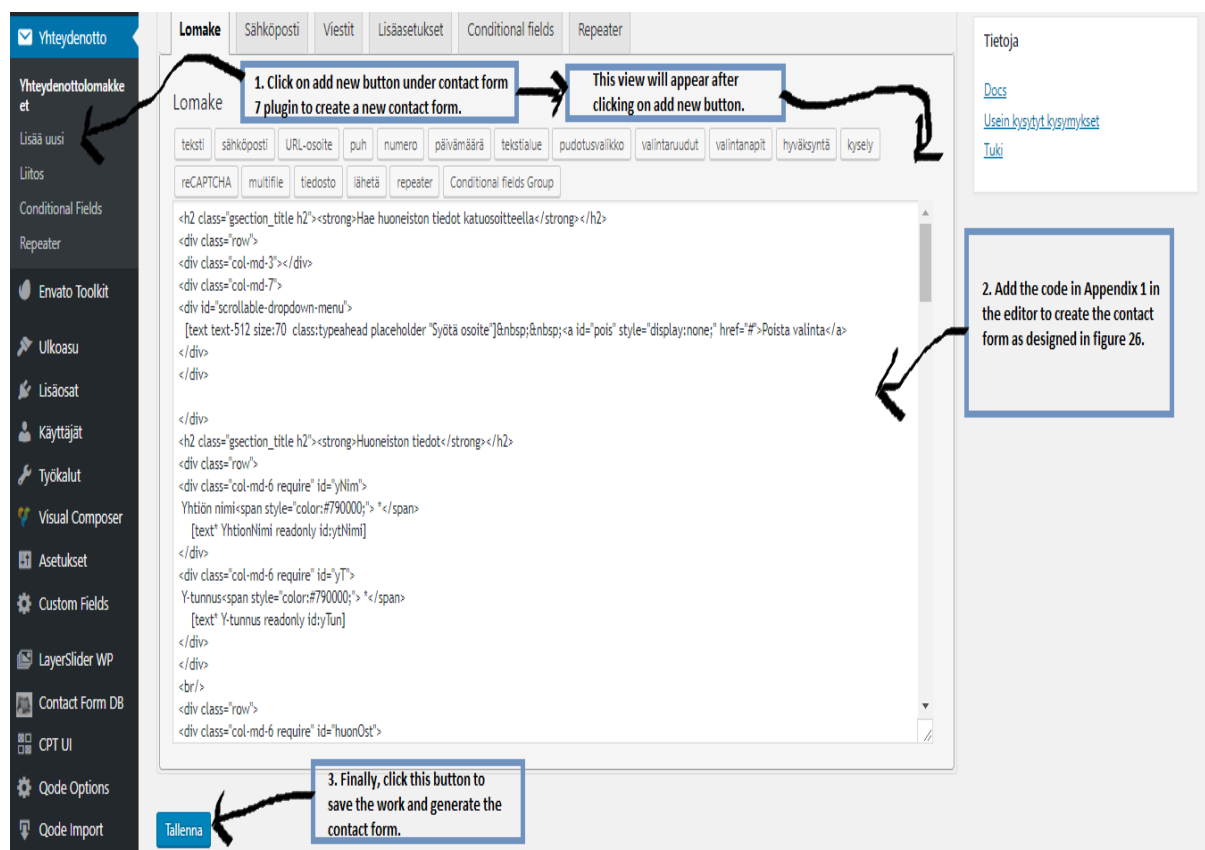


Figure 28: Process of adding the code in Appendix 1 to contact form 7 editor and generating the contact form.

Step 2:

Now that the contact form has been developed in step 1, it was required to add information of different properties of Isännöitsijäpalvelu Oy to the database which will be shown on the auto search field suggestion. This step was achieved using Custom post type UI and advanced custom field plug-in which are explained in section 6.2.1 of chapter 6.

Firstly, a custom post type was registered called 'Taloyhtiöt' using the Custom post type plug-in which has been explained in figure 28.

1. Click on Custom post type UI plugin on admin panel and then click on add/edit post types to register a new custom post type.

2. As the custom post type name is 'taloyhtiöt' so 'taloyhtiot' was typed in post type slug.

3. This field suggested to input the plural name of the custom post type so 'Taloyhtiöt' was inputted in the plural label field.

4. This field suggested to input the singular name of the custom post type so 'Taloyhtiö' was inputted in the singular label field as 'Taloyhtiö' is singular of 'Taloyhtiöt'.

5. Click on save post type to save the custom post type.

6. After the post type is saved, 'Taloyhtiöt' post type is appeared in the panel.

Additional labels

These fields will appear after clicking on add/edit post types.

Figure 29: Registering 'Taloyhtiöt' custom post type.

Now that the custom post type was registered, it was required to add custom fields using advanced custom field plug-in to the custom post type 'Taloyhtiöt' which has been explained in figure 29.

4. Finally click this button to save the customization.

Field Order	Field Label	Field Name	Field Type
1	Osoite	osoite	Text
2	Y-tunnus	y-tunnus	Text

Drag and drop to reorder

+ Add Field

Location

Rules

Create a set of rules to determine which edit screens will use these advanced custom fields

Show this field group if

Post Type is equal to taloyhtiöt and

or

Add rule group

Options

1. Click on advanced custom field plugin to add custom fields to 'Taloyhtiöt' post type.

These rows will appear after clicking the advanced custom field plugin.

2. Click on add field button to add fields which will then ask to enter field label, field name and field type as required. For example, in the 'Taloyhtiöt' post type only two input fields are required. One of which is an input field to enter a property's address and another the y-tunnus of the property. In this case both field types were assigned to text as both address and y-tunnus may have both letters and numbers. After correctly entering the required information click on save to create the fields.

3. Now select 'taloyhtiöt' so that the fields created in step 2 can appear while adding different property's information on 'Taloyhtiöt' custom post type.

Figure 30: Adding custom fields to 'Taloyhtiöt' custom post type.

Finally, information of the properties was all set to be added into the 'Taloyhtiöt' custom post type. Figure 31 explains the process in detail.

Bostadsaktiebolaget Boas Asunto-osakeyhtiö

Kestolinkki: <http://localhost/isp/taloyhtiöt=bostadsaktiebolaget-boas-asunto-osakeyhtiö> Esikatselurakenne

Osoite

Urheilukatu 20

Y-tunnus

0117450-4

2. Enter the name of the property in this field.

3. Enter the address of the property in this field.

4. Enter the y-tunnus of the property in this field.

5. Finally, click this button to save the property information.

1. Click on add new button under 'Taloyhtiöt' custom post type to add a property's information

These input fields will appear after clicking the add new button.

Julkaise

Esikatselumuutokset

Tila: Julkaistu Muokkaa

Näkyvyys: Julkinen Muokkaa

Julkaistu: 18.04.2017 klo 17:20 Muokkaa

Siirrä roskakoriin Päivitä

Artikkelikuva

Aseta artikkelikuva

Figure 31: Process for adding property information to 'Taloyhtiöt' custom post type.

Figure 31 explains the process of adding one property to the 'Taloyhtiöt' custom post type. However, figure 32 shows the list of all properties that has been added by following the same process explained in figure 31.

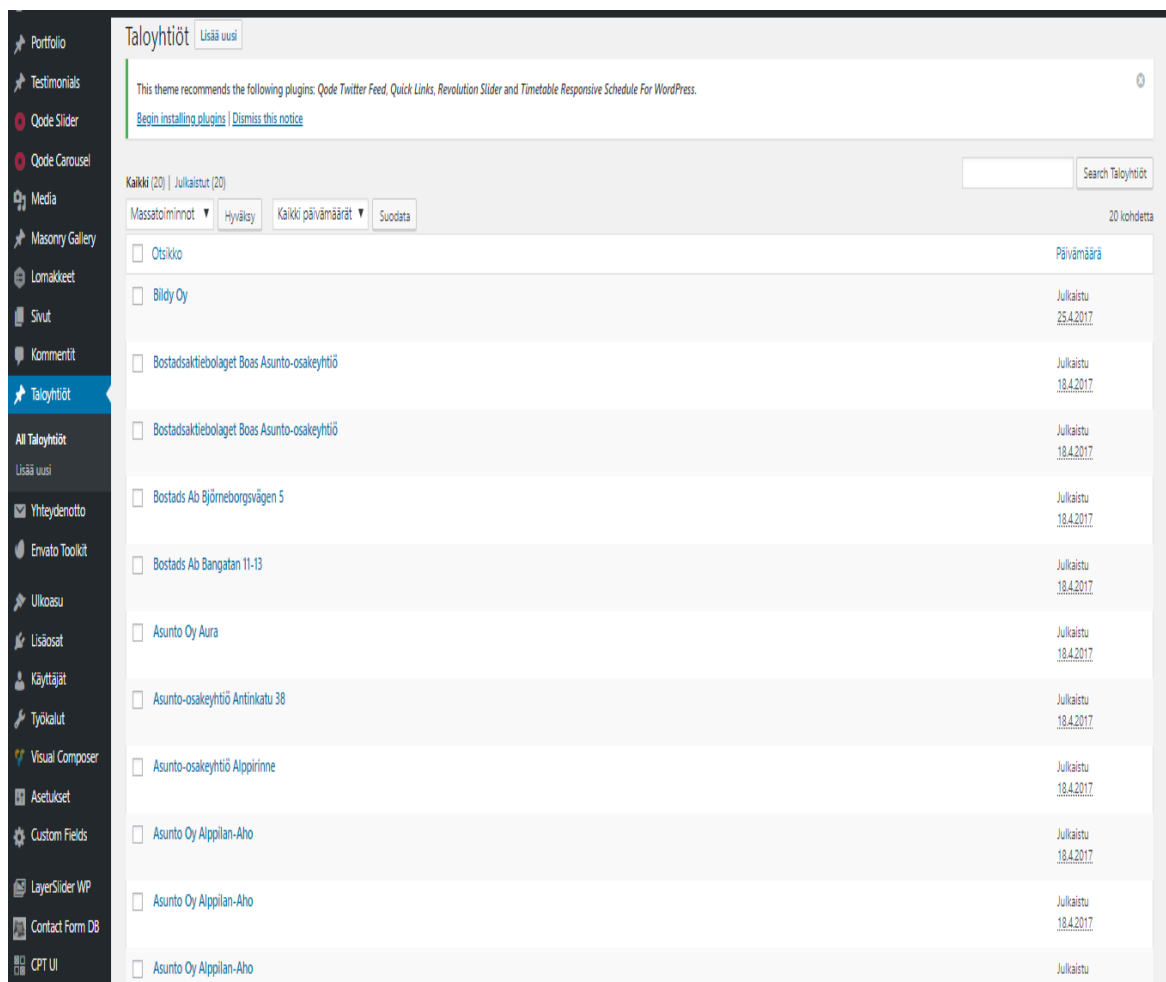


Figure 32: List of all the properties of Isännöitsijäpalvelu Oy.

Step 3:

This step explains the process of adding the 'Taloyhtiöt' custom post type to the WordPress API's HTTP endpoint. Installing the WordPress Rest API plug-in allows to retrieve the web-site's data as json output. WordPress Rest API plug-in has been discussed in detail in section 6.2.1 of chapter 6. Moreover, ACF to WordPress API plugin has made it easier to add the 'Taloyhtiöt' custom post type to the HTTP endpoint. ACF to WordPress API plug-in has been discussed in detail in section 6.2.1 of chapter 6. Simply, installing this plug-in and adding the code of Appendix 2 to the functions.php file enables the property information in the 'Taloyhtiöt' custom post type to be added to

the HTTP endpoint. Functions.php has been explained in detail in section 4.2.6 of chapter 4. As a result, the following url sends a GET request to the WordPress rest API to view the properties' information as json output:

http://www.isp.fi/wp-json/wp/v2/taloyhtiot?per_page=100

Figure 33 views the json output after sending the GET request to the WordPress rest API.

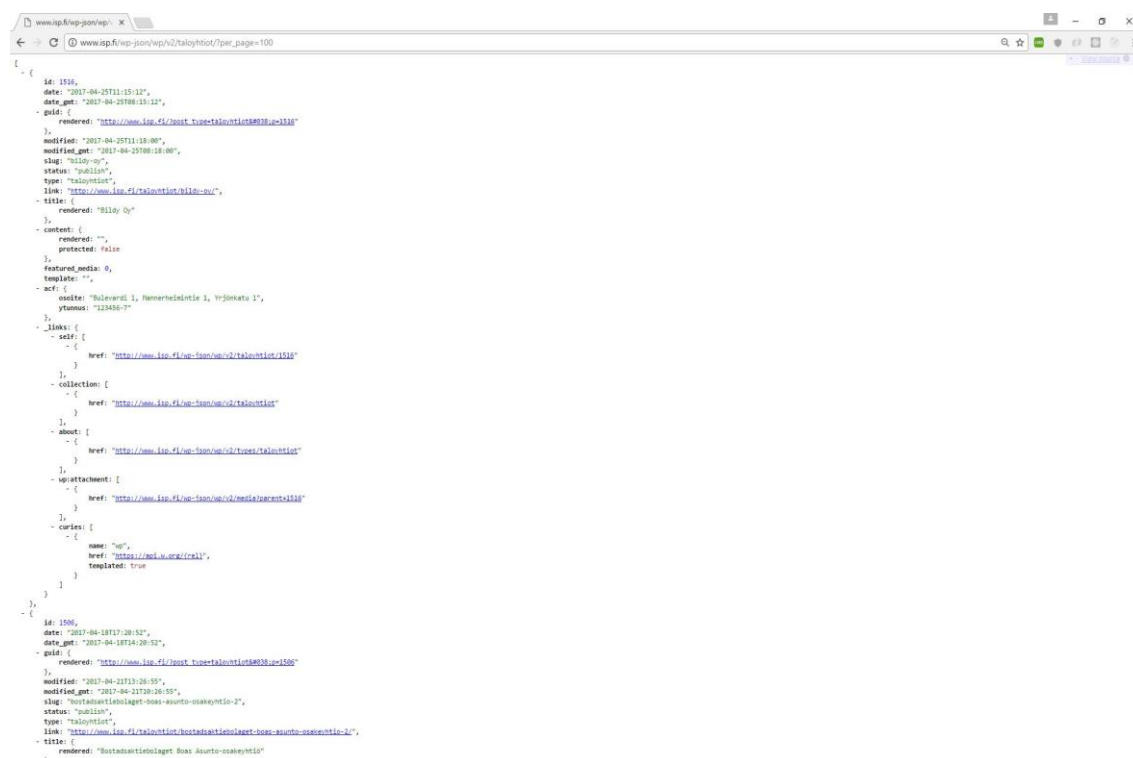


Figure 33: Json output of 'Taloyhtiot' custom post type.

Step 4:

This step comprises of adding the typeahead.js library to the active theme which has been achieved by adding the typeahead.bundle.js file, downloaded from the typeahead website, to the themes' folder inside wp-content directory and writing the following code inside the function of functions.php file as shown in Appendix 3 which finally activates the typeahead library in the active theme of the web-site:

```
wp_enqueue_script("typeahead", get_stylesheet_directory_uri() . '/typeahead.bundle.js', array(), false, true);
```

Wp-content directory has been elaborated in section 4.2.4 of chapter 4.

Step-5:

Now that the typeahead.js library has been activated in the active theme of the web-site as discussed in step-4, it was time to use the typeahead library according to the requirement which has been achieved by creating a javascript file with name 'typeaheadIntegration.js' inside the active theme's folder and adding the file to the active theme of the web-site by writing the following code to the function of functions.php file as shown in Appendix 4:

```
wp_enqueue_script("typeaheadIntegration",get_stylesheet_directory_uri().'/typeaheadIntegration.js',array('typeahead'),true,true);
```

Step-6:

This step explains the process of adding the features of typeahead library to an input field in the desired contact form. As a result, the first input field of the contact form of figure 26 has been chosen to apply the typeahead library features- an auto suggestion field. Figure 34 views the targeted input field more clearly.



Figure 34: Input field that has been used to work as an auto search field.

As per the typeahead library manual, it is required to surround the input field, that should bear the typeahead library features, within a html div tag with id 'scrollable-dropdown-menu' which can be seen in the contact form code in Appendix 1 marked with blue colour. In addition, the desired input field should also be assigned with class 'typeahead' which can be found from the contact form code in Appendix 1 marked with orange colour. Finally, the input field is now all set for customization according to the requirement. As a result, the code shown in Appendix 4 has been written inside the typeaheadIntegration.js file which made it possible to add the typeahead library features as per requirement of the customer. However, the following code of Appendix 4 has made it easier to fetch all the data of 'Taloyhtiöt' custom post type inside a variable named 'json' by sending a GET request to the WordPress rest API:

```

var json = (function () {
    var json = null;
    $.ajax({
        'async': false,
        'global': false,
        'url': "http://www.isp.fi/wp-json/wp/v2/taloyhtiot/?per_page=100",
        'dataType': "json",
        'success': function (data) {
            json = data;
        }
    });
    return json;
})();

```

Now that the data has been fetched, it was required to make the Bloodhound settings. Bloodhound works as a search engine for typeahead.js which has been elaborated in detail in section 6.2.1 of chapter 6. As a result, the code of Appendix 4 as follows was used to gather the names, addresses and y-tunnus of different properties from the fetched json data as queries inside three variables- titles, descs and tun:

// Bloodhound settings to add the names of properties as a query inside a variable from the json data

```

var titles = new Bloodhound({
    datumTokenizer: function (data) {
        return Bloodhound.tokenizers.whitespace(data.title.rendered);
    },
    queryTokenizer: Bloodhound.tokenizers.whitespace,
    local: json
});

```

// Bloodhound settings to add the addresses of properties as a query inside a variable from the json data

```

var descs = new Bloodhound({
    datumTokenizer: function (data) {
        return Bloodhound.tokenizers.whitespace(data.acf.osoite);
    },
    queryTokenizer: Bloodhound.tokenizers.whitespace,
    local: json
});

```

// Bloodhound settings to add the y-tunnus of properties as a query inside a variable from the json data

```

var tun = new Bloodhound({
    datumTokenizer: function (data) {
        return Bloodhound.tokenizers.whitespace(data.acf.ytunnus);
    },
    queryTokenizer: Bloodhound.tokenizers.whitespace,
    local: json
});

```

```
});
titles.initialize();
descs.initialize();
tun.initialize();
```

Finally, the typeahead settings were required which would enable to search the properties either with name, address or y-tunnus. The following code of Appendix 4 helped a lot to achieve the result:

// typeahead settings for searching a property either with name, address or y-tunnus

```
$('#scrollable-dropdown-menu .typeahead').typeahead({
  highlight: true
}, {
  name: 'titles',
  displayKey: function(titles) {
    return titles.acf.osoite + ", " + titles.title.rendered + ", " + titles.acf.ytunnus;
  },
  source: titles.ttAdapter()
}, {
  name: 'descs',
  displayKey: function(descs) {
    return descs.acf.osoite + ", " + descs.title.rendered + ", " + descs.acf.ytunnus;
  },
  source: descs.ttAdapter()
},
{
  name: 'tun',
  displayKey: function(tun) {
    return tun.acf.osoite + ", " + tun.title.rendered + ", " + tun.acf.ytunnus;
  },
  source: tun.ttAdapter()
}).on("typeahead:selected typeahead:autocompleted", function (ev, datum) {
  console.log(datum);
  $("#ytNimi").val(datum.title.rendered);
  $("#yTun").val(datum.acf.ytunnus);
  $("#huonOs").val(datum.acf.osoite);
}); });});
```

Lastly, the final achievement was to fill up some input fields automatically after selecting the desired property from the auto suggestion. The input fields that should be automatically filled up have been viewed in figure 35.

Huoneiston tiedot

Yhtiön nimi * Y-tunnus *

Huoneiston osoite *

Figure 35: Input fields that should be automatically filled up.

The input fields in figure 35 was assigned with ids- 'ytNimi', 'yTun' and 'huonOs' consecutively as shown in Appendix 1 with green colour. To sum up, the following part of the code in Appendix 4 was responsible for achieving the result:

```
on("typeahead:selected typeahead:autocompleted", function (ev, datum) {
  console.log(datum);
  $("#ytNimi").val(datum.title.rendered);
  $("#yTun").val(datum.acf.ytunnus);
  $("#huonOs").val(datum.acf.osoite);
});
```

6.2.1.2 Testing of phase 1

Phase 1 comprised of six steps which have been discussed in section 6.2.1.1 of chapter 6. To conclude, figures 37, 38, 39 and 40 gives the demo of the final outcome of the work that has been carried out throughout the six steps of phase 1.

Figure 37: Suggestions viewed when a property has been entered.

Hae huoneiston tiedot katuosoitteella

2. The suggestion box shows the suggestions with all the addresses highlighted with black colour that matches the address inputted by the user.

Fredrinkinkatu

Fredrinkinkatu, Bildy, 234234-Y

1. Address of a property has been entered here.

Huoneiston tiedot

Yhtiön nimi *

Y-tunnus *

Huoneiston osoite *

Osakkeenomistaja / työn teettäjä

Nimi*

Figure 38: Suggestions viewed when a property address has been entered.

Hae huoneiston tiedot katuosoitteella

2. The suggestion box shows the suggestions with the y-tunnus highlighted with black colour that matches the y-tunnus inputted by the user.

234234-Y

Fredrinkinkatu, Bildy, 234234-Y

1. Y-tunnus of a property has been entered here.

Huoneiston tiedot

Yhtiön nimi *

Y-tunnus *

Huoneiston osoite *

Osakkeenomistaja / työn teettäjä

Nimi*

Figure 39: Suggestions viewed when a y-tunnus of a property has been entered.

Hae huoneiston tiedot katuosoitteella

Fredrinkinkatu, Bildy, 234234-Y

Poista valinta

Huoneiston tiedot

Yhtiön nimi *
Bildy

Y-tunnus *
234234-Y

Huoneiston osoite *
Fredrinkinkatu

1. The desired property has been selected.

2. These fields have been filled up automatically after selecting the desired property in step 1.

Figure 40: Input fields automatically filled up on selection of the desired property.

6.2.2 Phase 2

First phase elaborated the process of developing the contact form and adding an auto search field in the form. However, this final phase elaborates the process of connecting the contact form to the customer's CRM (Customer relationship management system).

6.2.2.1 Steps for implementing phase 2

Phase 2 comprises of five steps which are explained as follows:

Step 1:

Contact form 7 plugin has a built in action hook, which should be used inside functions.php file, which allows to perform some certain codes before submitting the contact form. The following action hook of contact form 7 enables the feature:

```
add_action( 'wpcf7_before_send_mail', 'CF7_before_send' );
function CF7_before_send($cf7) {

}
```

The action hook of contact form 7 has been used to send the contact form data to the customer's CRM.

Firstly, it was required to fetch the temporary location of any attached file in the contact form from the database which was achieved by writing the following lines of code inside the CF7_before_send function as in Appendix 5:

```
$form_to_DB = WPCF7_Submission::get_instance();
if ($form_to_DB) {
    $uploaded_files = $form_to_DB->uploaded_files(); // this allows to access to upload file in the
temp location
}
```

Step 2:

This step comprises of encoding the contents of any file attached in the contact form with base64. The contents of the file are required to encode with base64 because this encoding allows binary data to transfer easily across a network. Figure 41 identifies the file uploader in the contact form. [27]



Figure 41: File uploader in the contact form.

The file uploader in the contact form has been assigned with name 'file-365' which can be deduced from the code in Appendix 1 marked with red colour. However, the following code of Appendix 5 made it convenient to encode the contents of the attached file with base64:

```
// Encoding the attached file contents with base64
$cf7_file_field_name = 'file-365'; // file upload field name
$file_name = $formData[$cf7_file_field_name]; // Getting uploaded file name
$file_location = $uploaded_files[$cf7_file_field_name]; // url of the uploaded file
$file_content = file_get_contents($file_location); // Getting contents of the uploaded file
$encoded = base64_encode($file_content); // encoding the file contents with base64
```

Step 3:

Now that the contents of the attached file has been encoded with base64 in step 1, it was required to create file name object, file content object and form submission time object by writing the following code which can be deduced from Appendix 5:

```
$_POST['LiitetiedostotNimi'] = $file_name; // creating the attached file's name object
$_POST['Liitetiedostot'] = $encoded; // creating the file content object
$_POST['LuontiPvm'] = date("Y-m-d H:i:s"); // creating the form submission time object
```

Step 4:

This step elaborated the process of removing unicode sequences from the submitted data and encoding the data to json format. Following function removes unicode sequences from the submitted data and encodes the data to json format:

```
$Cont = $_POST; // contains all the data of the submitted form

// function to remove Unicode sequences from the inputted data of the contact form and encode the data to json
function jsonRemoveUnicodeSequences($struct) {
    return preg_replace("/\\\\u{[a-f0-9]{4}}/e", "iconv('UCS-4LE','UTF-8',pack('V', hexdec('U$1')))",
    json_encode($struct));
}
$jsonCont = jsonRemoveUnicodeSequences($Cont); // contains all the data of the submitted form in json format
```

Step 5:

The company 'Isännöitsijäpalvelu Oy' has provided the url through which the contact form data should be pushed to the database of their CRM and the API key of their CRM beforehand. Therefore, these information were utilized in the following code, which can be deduced from the code of Appendix 5, which made it possible to send the submitted data of the contact form to the database of the CRM:

```

$apiKey = '7b4ec69fb5b90b5604c943a066dd1e8d'; // CRM API key
$headers = array(
    'Authorization: ' . $apiKey,
    'Content-Type: application/json',
    'Expect: ' );
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $jsonCont );
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 5);
$response = curl_exec($ch);
file_put_contents("cf7outputtest.txt", $response); // Only for test purposes: Creating a text file to the root
folder of the web-site which contains the response in order to confirm that the data has been sent success-
fully to the CRM's database
curl_close($ch);
file_put_contents("cf7outputtest.json", $jsonCont); // Only for test purposes: Creating a json file to the
root folder of the web-site which contains the json data which has been sent successfully to the CRM's
database

```

6.2.2.2 Testing of phase 2

Testing of sending the submitted data of the contact form to the database of the CRM:

The test for ensuring that the submitted data of the contact form has been successfully sent to the database of the CRM was confirmed by creating a text file in the root folder of the web-site of Isännöitsijäpalvelu Oy which contains the response. As a result, the testing was achieved by adding the following line of code which can be deduced from Appendix 5:

```

file_put_contents("cf7outputtest.txt", $response); // Only for test purposes: Creating a text file to the root
folder of the web-site which contains the response in order to confirm that the data has been sent success-
fully to the CRM's database

```

Figure 42 indicates that the text file with name 'cf7outputtest.txt' was created in the root folder of the web-site of Isännöitsijäpalvelu Oy and figure 43 shows the response that has been returned after sending the data to the database of the CRM.

Name	Date modified	Type	Size
.htpasswd	5/24/2017 6:57 AM	File folder	
cgi-bin	5/24/2017 6:57 AM	File folder	
wp-admin	5/13/2018 3:29 AM	File folder	
wp-content	5/13/2018 3:33 AM	File folder	
wp-includes	5/13/2018 3:33 AM	File folder	
.htaccess	5/25/2017 4:51 AM	HTACCESS File	1 KB
cf7outputtest.json	5/18/2017 1:48 AM	JSON File	3 KB
cf7outputtest	5/18/2017 1:48 AM	Text Document	1 KB
core.276631	12/2/2016 3:36 PM	276631 File	87,640 KB
core.430091	5/6/2017 2:40 AM	430091 File	137,096 KB
core.555338	5/5/2017 8:04 AM	555338 File	129,464 KB
core.570187	5/5/2017 8:18 AM	570187 File	120,432 KB
core.593424	5/4/2017 1:18 AM	593424 File	115,652 KB
core.729604	5/4/2017 3:02 AM	729604 File	88,936 KB
core.899639	4/26/2017 12:53 AM	899639 File	139,656 KB
error_log	5/24/2017 1:16 AM	File	1,064 KB
index	12/19/2014 3:04 AM	PHP File	1 KB
isannoitsija (6).sql	5/25/2017 4:45 AM	SQL File	12,643 KB
license	1/11/2017 2:11 PM	Text Document	20 KB
lisensi	1/11/2017 2:11 PM	HTML File	22 KB
readme	5/16/2017 4:37 PM	HTML File	8 KB
test	5/18/2017 1:48 AM	WinRAR ZIP archive	1 KB
wp-activate	12/13/2016 9:17 AM	PHP File	6 KB
wp-blog-header	4/13/2016 4:30 AM	PHP File	1 KB
wp-comments-post	12/13/2016 9:17 AM	PHP File	2 KB
wp-config	5/25/2017 2:23 AM	PHP File	3 KB
wp-config-sample	4/13/2016 4:30 AM	PHP File	3 KB
wp-cron	8/26/2015 1:05 AM	PHP File	4 KB

Text file which contains the response.

Figure 42: Text file containing the response returned after sending the submitted data to the CRM.

```

1
2 {"data":{"status":"OK."},"statusCode":200}

```

Figure 43: Response returned after sending the submitted data to the CRM.

According to the figure 43, the status is 'OK' which confirms that the data was sent successfully to the database of the CRM.

Testing that the submitted data of the contact form has been encoded to json:

This test was carried out by creating a json file with name 'cf7outputtest.json' in the root folder of the web-site of Isännöitsijäpalvelu Oy which contains all the submitted data of

the contact form in json format. Therefore, it was concluded by writing the following line of code which can be deduced from Appendix 5:

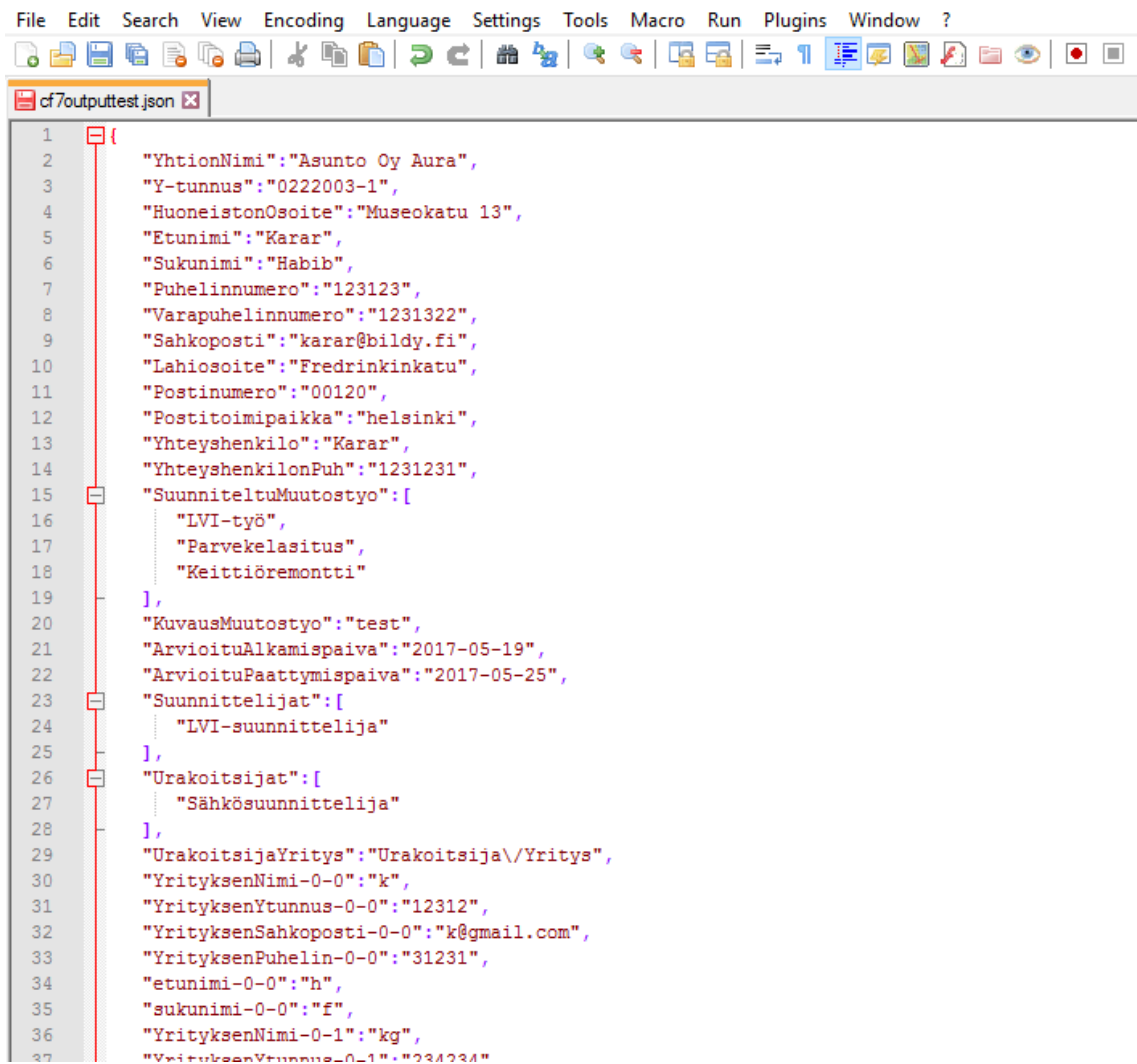
```
file_put_contents("cf7outputtest.json", $jsonCont); // Only for test purposes: Creating a json file to the root folder of the web-site which contains the json data which has been sent successfully to the CRM's database
```

Figure 44 indicates that the json file with name 'cf7outputtest.json' was created in the root folder of the web-site of Isännöitsijäpalvelu Oy whereas figure 45 displays all the submitted data of the contact form in json format.

Name	Date modified	Type	Size
.htpasswd	5/24/2017 6:57 AM	File folder	
cgi-bin	5/24/2017 6:57 AM	File folder	
wp-admin	5/13/2018 3:29 AM	File folder	
wp-content	5/13/2018 3:33 AM	File folder	
wp-includes	5/13/2018 3:33 AM	File folder	
.htaccess	5/25/2017 4:51 AM	HTACCESS File	1 KB
cf7outputtest.json	5/18/2017 1:48 AM	JSON File	3 KB
cf7outputtest	5/18/2017 1:48 AM	Text Document	1 KB
core.276631	12/2/2016 3:36 PM	276631 File	87,640 KB
core.430091	5/6/2017 2:40 AM	430091 File	137,096 KB
core.555338	5/5/2017 8:04 AM	555338 File	129,464 KB
core.570187	5/5/2017 8:18 AM	570187 File	120,432 KB
core.593424	5/4/2017 1:18 AM	593424 File	115,652 KB
core.729604	5/4/2017 3:02 AM	729604 File	88,936 KB
core.899639	4/26/2017 12:53 AM	899639 File	139,656 KB
error_log	5/24/2017 1:16 AM	File	1,064 KB
index	12/19/2014 3:04 AM	PHP File	1 KB
isannoitsija (6).sql	5/25/2017 4:45 AM	SQL File	12,643 KB
license	1/11/2017 2:11 PM	Text Document	20 KB
lisenssi	1/11/2017 2:11 PM	HTML File	22 KB
readme	5/16/2017 4:37 PM	HTML File	8 KB
test	5/18/2017 1:48 AM	WinRAR ZIP archive	1 KB
wp-activate	12/13/2016 9:17 AM	PHP File	6 KB
wp-blog-header	4/13/2016 4:30 AM	PHP File	1 KB
wp-comments-post	12/13/2016 9:17 AM	PHP File	2 KB
wp-config	5/25/2017 2:23 AM	PHP File	3 KB
wp-config-sample	4/13/2016 4:30 AM	PHP File	3 KB
wp-cron	8/26/2015 1:05 AM	PHP File	4 KB

The desired json file has been created.

Figure 44: Json file containing all the submitted data of the contact form.



```

1  {
2      "YhtionNimi": "Asunto Oy Aura",
3      "Y-tunnus": "0222003-1",
4      "HuoneistonOsoite": "Museokatu 13",
5      "Etunimi": "Karar",
6      "Sukunimi": "Habib",
7      "Puhelinnumero": "123123",
8      "Varapuhelinnumero": "1231322",
9      "Sahkoposti": "karar@bildy.fi",
10     "Lahiosoite": "Fredrinkinkatu",
11     "Postinumero": "00120",
12     "Postitoimipaikka": "helsinki",
13     "Yhteyshenkilö": "Karar",
14     "YhteyshenkilönPuh": "1231231",
15     "SuunniteltuMuutostyo": [
16         "LVI-työ",
17         "Parvekelasitus",
18         "Keittiöremontti"
19     ],
20     "KuvausMuutostyo": "test",
21     "ArvioituAlkamispäivä": "2017-05-19",
22     "ArvioituPaattymispäivä": "2017-05-25",
23     "Suunnittelijat": [
24         "LVI-suunnittelija"
25     ],
26     "Urakoitsijat": [
27         "Sähkösuunnittelija"
28     ],
29     "UrakoitsijaYritys": "Urakoitsija/Yritys",
30     "YrityksenNimi-0-0": "k",
31     "YrityksenYtunnus-0-0": "12312",
32     "YrityksenSahkoposti-0-0": "k@gmail.com",
33     "YrityksenPuhelin-0-0": "31231",
34     "etunimi-0-0": "h",
35     "sukunimi-0-0": "f",
36     "YrityksenNimi-0-1": "kg",
37     "YrityksenYtunnus-0-1": "234234"

```

Figure 45: All the submitted data of the contact form in json format

7 Conclusion

The purpose of the study were to explain WordPress in detail, the security issues or checks that should be maintained before launching a WordPress powered web-site and discussing about implementing custom integrations within WordPress.

To sum up, in this thesis, a crystal clear view of the entire WordPress system has been discussed such as about WordPress architecture, WordPress file architecture, WordPress theme architecture etc. Moreover, the security issues of WordPress has also been elaborately discussed within the paper. Lastly, an experience of a custom integration within WordPress has been illustrated which I had carried out while working

in Bildy Oy which in turn views the possibilities of making custom integrations within a WordPress powered web-site. The custom integration went live on 15th May 2017 and until now it has been proved as an essential integration for the customer.

Finally, it can be said that WordPress is a very convenient content-management system (CMS) which offers endless abilities, functionalities and flexibility.

References

1. So, what is a CMS? [online] URL:
http://www.steptwo.com.au/papers/kmc_what/ . Accessed 4 June 2017
2. The History of WordPress [online] URL:
<http://www.wpbeginner.com/news/the-history-of-wordpress/> . Accessed 4 June 2017
3. WordPress vs Joomla vs Drupal: CMS Comparison Guide [online] URL:
<https://business.tutsplus.com/articles/wordpress-vs-joomla-vs-drupal-cms-comparison-guide--cms-26581> . Accessed 10 June 2017
4. What is WordPress? [online] URL:
<https://www.pixelemu.com/documentation/wordpress-basics/what-is-wordpress> . Accessed 15 June 2017
5. WordPress Files and Directory Structure [online] URL:
<https://www.interserver.net/tips/kb/wordpress-files-directory-structure/> . Accessed 15 June 2017
6. Beginners Guide to Understanding WordPress' Internal Functioning [online] URL:
<http://www.wpexplorer.com/wordpress-internal-function/> . Accessed 4 July 2017
7. WordPress Website Architecture Explained in great detail [online] URL:
<https://www.optimizesmart.com/wordpress-ninja-15-minutes/> . Accessed 4 July 2017
8. Why your site should be using HTTPS [online] URL:
<https://www.chapterthree.com/blog/why-your-site-should-be-using-https> . Accessed 5 January 2018
9. 12 Ways to Secure Your WordPress Site You've Overlooked [online] URL:
<https://premium.wpmudev.org/blog/wordpress-security-tips/> . Accessed 5 January 2018
10. The Ultimate WordPress Security Guide- Step by Step (2018) [online] URL:
<http://www.wpbeginner.com/wordpress-security/>. Accessed 7 January 2018
11. 10 Security Steps You Should Take Before Launching Your WordPress Site [online] URL:
<https://www.seedprod.com/security-before-launching-wordpress-site> . Accessed 10 January 2018
12. How to Change the WordPress Database Prefix to Improve Security [online] URL:
<http://www.wpbeginner.com/wp-tutorials/how-to-change-the-wordpress-database-prefix-to-improve-security/>. Accessed 13 February 2018

13. How to Add Security Questions to WordPress Login Screen [online] URL: <http://www.wpbeginner.com/plugins/how-to-add-security-questions-to-wordpress-login-screen/> . Accessed 13 February 2018
14. Top tips to prevent a WordPress hack [online] URL: <https://www.acunetix.com/websitesecurity/preventing-wordpress-hack/>. Accessed 13 February 2018
15. How to Disable XML-RPC in WordPress [online] URL: <http://www.wpbeginner.com/plugins/how-to-disable-xml-rpc-in-wordpress/> . Accessed 13 February 2018
16. The Right Way to Remove WordPress Version Number [online] URL: <http://www.wpbeginner.com/wp-tutorials/the-right-way-to-remove-wordpress-version-number/> . Accessed 13 February 2018
17. Changing file permissions [online] URL: https://codex.wordpress.org/Changing_File_Permissions . Accessed 3 May 2018
18. How- and why- you should use a VPN any time you hop on the internet [online] URL: <https://www.techhive.com/article/3158192/privacy/howand-whyyou-should-use-a-vpn-any-time-you-hop-on-the-internet.html> . Accessed 3 May 2018
19. Protect the WordPress wp-config.php Configuration File [online] URL: <https://www.wpwhitesecurity.com/wordpress-security-hacks/protect-wordpress-wp-config-php-security/> . Accessed 6 May 2018
20. 10 WordPress Backup Plugins You Need to Know About [online] URL: <https://www.elegantthemes.com/blog/tips-tricks/10-wordpress-backup-plugins-you-need-to-know-about> . Accessed 10 May 2018
21. jQuery#typeahead [online] URL: https://github.com/twitter/typeahead.js/blob/master/doc/jquery_typeahead.md . Accessed 16 May 2018
22. Bloodhound [online] URL: <https://github.com/twitter/typeahead.js/blob/master/doc/bloodhound.md> . Accessed 16 May 2018
23. Documentation [online] URL: <https://www.advancedcustomfields.com/resources/> . Accessed 16 May 2018
24. Custom Post Type UI [online] URL: <https://wordpress.org/plugins/custom-post-type-ui/> . Accessed 16 May 2018
25. WP REST API [online] URL: <https://v2.wp-api.org/> . Accessed 17 May 2018

26. ACF to WP-API [online] URL: <https://wordpress.org/plugins/acf-to-wp-api/#description> . Accessed 20 May 2018
27. base64_encode [online] URL: <http://php.net/manual/en/function.base64-encode.php> . Accessed 23 May 2018

Appendix 1. Contact form 7 code for Isännöitsijäpalvelu Oy's web-site contact form

```
<h2 class="gsection_title h2"><strong>Haahuoneistontiedotkatuosoitteella</strong></h2>
<div class="row">
<div class="col-md-3"></div>
<div class="col-md-7">
<div id="scrollable-dropdown-menu">
[text text-512 size:70 class:typeahead placeholder "Syötäosoite"]
</div>
<a id="pois" style="display:none;" href="#">Poistavalinta</a>
</div>
</div>
<h2 class="gsection_title h2"><strong>Huoneistontiedot</strong></h2>
<div class="row">
<div class="col-md-6 require" id="yNim">
Yhtiön nimi<span style="color:#790000;"> *</span>
[text* YhtiönNimi readonly id:ytNimi]
</div>
<div class="col-md-6 require" id="yT">
Y-tunnus<span style="color:#790000;"> *</span>
[text* Y-tunnus readonly id:yTun]
</div>
</div>
</div>
<br/>
<div class="row">
<div class="col-md-6 require" id="huonOst">
Huoneiston osoite<span style="color:#790000;"> *</span>
[text* HuoneistonOsoite readonly id:huonOs]
</div>
</div>
<br/>
<h2 class="gsection_title"><strong>Osakkeenomistaja / työntekijä</strong></h2>
Nimi*
<div class="row">
<div class="col-md-6 require" id="etuN">
[text* Etunimiid:etu] <span style="font-size:13px;">Etunimi</span>
</div>
<div class="col-md-6 require" id="sukuN">
[text* Sukunimiid:suku] <span style="font-size:13px;">Sukunimi</span>
</div>
</div>
<br/>
<div class="row">
<div class="col-md-6 require" id="puhe">
Puhelinnumero<span style="color:#790000;"> *</span>
[text* Puhelinnumero id:puh]
```

```

</div>
<div class="col-md-6 require" id="varaPuhe">
Varapuhelinnumero<span style="color:#790000;"> *</span>
[text* Varapuhelinnumero id:varaPuh]
</div>
</div>
<br/>
<div class="row">
<div class="col-md-6 require" id="sahkoP">
Sähköposti<span style="color:#790000;"> *</span>
[email* Sahkopostiid:sahko]
</div>
</div>
<br/>
<div class="row">
<div class="col-md-6 require" id="lhOst">
Lähiosoite<span style="color:#790000;"> *</span>
[text* Lahiosoiteid:lhOs]
</div>
<div class="col-md-6 require" id="postiN">
Postinumero<span style="color:#790000;"> *</span>
[text* Postinumero:postiNumero]
</div>
</div>
<br/>
<div class="row">
<div class="col-md-6 require" id="postiP">
Postitoimipaikka<span style="color:#790000;"> *</span>
[text* Postitoimipaikka id:postiPaikka]
</div>
<div class="col-md-6">
Yhteyshenkilö (jos joku muu kuin omistaja)
[text Yhteyshenkilo]
</div>
</div>
<br/>
<div class="row">
<div class="col-md-6">
Yhteyshenkilön puhelinnumero
[text YhteyshenkilonPuh]
</div>
</div>
<br/>
<div class="row">
<div class="col-md-12 require" id="suuDiv">
Suunniteltu muutostyö / korjaus<span style="color:#790000;"> *</span>
[checkbox* SuunniteltuMuutostyo id:suunMuucheck "LVI-työ" "Parvekelasitus" "Keittiöremontti" "Terassimuutos"
"Sähkötyö" "Muu" "Wc-remontti" "Väliseinämuutos" "Saunaremontti" "Kylpyhuoneremontti" "Lattiatyö"]

```

```

</div>
</div>
<br/>
<div class="row">
<div class="col-md-12 require" id="kuvausM">
Kuvaus rakennusteknisistä muutostöistä<span style="color:#790000;"> *</span>
  [textarea* KuvausMuutostyo id:kuvausMuu]
</div>
</div>
<br/>
<div class="row">
<div class="col-md-6 require" id="alkaDiv">
Arvioitu alkamispäivä<span style="color:#790000;"> *</span>
  [date* ArvioituAlkamispaiva id:alka]
</div>
<div class="col-md-6 require" id="paattyDiv">
Arvioitu päättymispäivä<span style="color:#790000;"> *</span>
  [date* ArvioituPaattymispaiva id:paatty]
</div>
</div>
<br/>
Liitetiedostot
<div class="fileArea">
<div class="text-centercenter-block">
  [multifile file-365]</div></div>
<br/>
<div class="row">
<div class="col-md-6">
Valitse tiedossa olevat suunnittelijat / valvojat
[checkbox Suunnittelijat "Pääsuunnittelija" "LVI-suunnittelija" "Sähkösuunnittelija" "Valvoja" "Muu suunnittelija"]
</div>
<div class="col-md-6">
Valitse tiedossa olevat urakoitsijat
[checkbox Urakoitsijat "Rakennusurakoitsija" "LVI-urakoitsija" "Sähkösuunnittelija" "Muu urakoitsija"]
</div>
</div>
<br/>
Työn suorittaa? Suunnittelijoiden/urakoitsijoiden y-tunnukset ja yhteystiedot
[radio UrakoitsijaYritys use_label_element default:1 "Osakas itse" "Urakoitsija/Yritys"]
[group group-540]
[repeater add-label:TGIzw6TDpCB1dXNpIHRvaW1pdHRhamE= remove-label:UG9pc3Rh index:0 min:1 max:10 show:0
base64decode]
[/group]
<br/>
[submitid:sub "LÄHETÄ"]
<br/>

```

Appendix 2. Code to view the 'Taloyhtiöt' custom post type to the HTTP endpoint of the WordPress Rest API

// function to register acf key to the HTTP endpoint of the rest API

```
add_action( 'rest_api_init', 'slug_register_acf' );
function slug_register_acf() {
    $post_types = get_post_types(['public'=>true], 'names');
    foreach ( $post_types as $type ) {
        register_api_field( $type,
            'acf',
            array(
                'get_callback' => 'slug_get_acf',
                'update_callback' => null,
                'schema' => null,
            )
        );
    }
}
```

// function to get custom fields

```
function slug_get_acf( $object, $field_name, $request ) {
    return get_fields($object[ 'id' ]);
}
```

```
add_filter('rest_prepare_multiple_choice', 'append_acf');
```

```
add_filter('rest_prepare_vocabularies', 'append_acf');
```

// function to append acf key to the HTTP endpoint of the rest API

```
function append_acf($response) {
    $response->data['acf'] = get_fields($response->data['id']);
    return $response;
};
```

// function to view the 'Taloyhtiöt' custom post type in the rest API

```
add_action( 'init', 'my_custom_post_type_rest_support', 25 );
```

```
function my_custom_post_type_rest_support() {
    global $wp_post_types;

    //be sure to set this to the name of your post type!
    $post_type_name = 'taloyhtiot';
    if( isset( $wp_post_types[ $post_type_name ] ) ) {
        $wp_post_types[$post_type_name]->show_in_rest = true;
        $wp_post_types[$post_type_name]->rest_base = $post_type_name;
        $wp_post_types[$post_type_name]->rest_controller_class =
            'WP_REST_Posts_Controller';
    }
}
```


Appendix 3. Function to enqueue css style and javascript files to the WordPress theme of Isännöitsijäpalvelu Oy in the functions.php file

```
// function to add css and javascript files to the theme
function my_theme_enqueue_styles() {
    wp_enqueue_style( "bootstrap", get_stylesheet_directory_uri() . '/bootstrap.min.css' );
    // adding typeahead library to the theme
    wp_enqueue_script("typeahead",get_stylesheet_directory_uri() . '/typeahead.bundle.js',array(),false,true);
    // adding typeahead custom integration javascript file to the theme
    wp_enqueue_script("typeaheadIntegration",get_stylesheet_directory_uri().'/typeaheadIntegration.js',array('typeahead'),true,true);
}
add_action( 'wp_enqueue_scripts', 'my_theme_enqueue_styles' );
```

Appendix 4. Code of typeaheadIntegration.js file

```
var $ =jQuery.noConflict();
$(document).ready(function(e) {
    var $ =jQuery.noConflict();
    // function to fetch the Taloyhtiöt' custom post type data as JSON output
    var json = (function () {
        var json = null;
        $.ajax({
            'async': false,
            'global': false,
            'url': "///www.isp.fi/wp-json/wp/v2/taloyhtiot/?per_page=100",
            'dataType': "json",
            'success': function (data) {
                json = data;
            }
        });
        return json;
    })();
    // Bloodhound settings to add the names of properties as a query inside a variable from the json data
    var titles = new Bloodhound({
        datumTokenizer: function (data) {
            return Bloodhound.tokenizers.whitespace(data.title.rendered);
        },
        queryTokenizer: Bloodhound.tokenizers.whitespace,
        local: json
    });
```

```

// Bloodhound settings to add the addresses of properties as a query inside a variable from the json data
var desc = new Bloodhound({
  datumTokenizer: function (data) {
    return Bloodhound.tokenizers.whitespace(data.acf.osoite);
  },
  queryTokenizer: Bloodhound.tokenizers.whitespace,
  local: json
});

// Bloodhound settings to add the y-tunnus of properties as a query inside a variable from the json data
var tun = new Bloodhound({
  datumTokenizer: function (data) {
    return Bloodhound.tokenizers.whitespace(data.acf.ytunnus);
  },
  queryTokenizer: Bloodhound.tokenizers.whitespace,
  local: json
});

titles.initialize();
descs.initialize();
tun.initialize();

// typeahead settings for searching a property either with name, address or y-tunnus
$('#scrollable-dropdown-menu .typeahead').typeahead({
  highlight: true
}, {
  name: 'titles',
  displayKey: function(titles) {
    return titles.acf.osoite + ", " + titles.title.rendered + ", " + titles.acf.ytunnus;
  },
  source: titles.ttAdapter()
}, {
  name: 'descs',
  displayKey: function(descs) {
    return descs.acf.osoite + ", " + descs.title.rendered + ", " + descs.acf.ytunnus;
  },
  source: descs.ttAdapter()
}, {
  name: 'tun',
  displayKey: function(tun) {
    return tun.acf.osoite + ", " + tun.title.rendered + ", " + tun.acf.ytunnus;
  },
  source: tun.ttAdapter()
}).on("typeahead:selected typeahead:autocompleted", function (ev, datum) {
  console.log(datum);
  $("#ytNimi").val(datum.title.rendered);
  $("#yTun").val(datum.acf.ytunnus);
});

```

```

        $("#huonOs").val(datum.acf.osoite);
    });
    });
});

```

Appendix 5. Code which enables sending the contact form data to the CRM of the customer

```

add_action( 'wpcf7_before_send_mail', 'CF7_before_send' );

function CF7_before_send($cf7) {
    $form_to_DB = WPCF7_Submission::get_instance();
    if ($form_to_DB) {
        $uploaded_files = $form_to_DB->uploaded_files(); // this allows to access to upload file in the
        temp location
    }
    // Encoding the attached file contents with base64
    $cf7_file_field_name = 'file-365'; // file upload field name
    $file_name = $formData[$cf7_file_field_name]; // Getting uploaded file name
    $file_location = $uploaded_files[$cf7_file_field_name]; // url of the uploaded file
    $file_content = file_get_contents($file_location); // Getting contents of the uploaded file
    $encoded = base64_encode($file_content); // encoding the file contents with base64

    $_POST['LiitetiedostotNimi'] = $file_name; // creating the attached file's name object
    $_POST['Liitetiedostot'] = $encoded; // creating the file content object
    $_POST['LuontiPvm'] = date("Y-m-d H:i:s"); // creating the form submission time object
    $Cont = $_POST; // contains all the data of the submitted form

    // function to remove Unicode sequences from the inputted data of the contact form and encode the data to
    json
    function jsonRemoveUnicodeSequences($struct) {
        return preg_replace("/\\\\u([a-f0-9]{4})/e", "iconv('UCS-4LE','UTF-8',pack('V', hexdec('U$1')))",
        json_encode($struct));
    }
    $jsonCont = jsonRemoveUnicodeSequences($Cont); // contains all the data of the submitted form in json
    format

    $url = "https://customers.taimer.com/ispdemo/iface_isp.php"; // CRM url
    $apiKey = '7b4ec69fb5b90b5604c943a066dd1e8d'; // CRM API key
    $headers = array(
        'Authorization: ' . $apiKey,
        'Content-Type: application/json',
        'Expect: ' );
    $ch = curl_init($url);

```

```
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $jsonCont );
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 5);
$response = curl_exec($ch);
file_put_contents("cf7outputtest.txt", $response); // Only for test purposes: Creating a text file to the
root folder of the web-site which contains the response in order to confirm that the data has been sent suc-
cessfully to the CRM's database
curl_close($ch);
file_put_contents("cf7outputtest.json", $jsonCont); // Only for test purposes: Creating a json file to the
root folder of the web-site which contains the json data which has been sent successfully to the CRM's
database
}
```