Junnan Ge

BOOKING SERVICE DESIGNS FOR LIEKE PLATFORM

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Degree Programme in Information Technology
June 2014

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Unit	Date	Author/s
Kokkola-Pietarsaari	June 2014	Junnan Ge

Degree programme
Information Technology

Name of thesis
BOOKING SERVICE DESIGNS FOR LIEKE PLATFORM

Instructor	Pages
Kauko Kolehmainen	46

Supervisor
Kauko Kolehmainen

This thesis work was commissioned by a company called Newave Designs Oy (Ltd). Newave Designs Oy is a software company founded in 2004 and currently located in Kokkola, Finland. Their main focus is on their LIEKE platform and browser-based software built on it.

The work was about software designs for a booking system that can be used to offer and reserve timeslots in the LIEKE platform that Newave Designs Oy is building. The goal of this project was to build a time-based service in LIEKE platform for small business companies in Finland. The design part was conducted as this thesis work, and Newave Design Oy will do the implementation work.

During the writing of this thesis, the technical requirements were mentioned. Those requirements were the knowledge to understand LIEKE platform as well as software designs. LIEKE is a platform and content management system (CMS) for building dynamic websites and applications offering a broad range of features and services including user administration, translation tools, content publishing and online shopping among others. The technical terms, such as LAMP, CodeIgneiter, HMVC, Smarty, responsive design, and UML were explained. The designs of the project were done with different design tools. For visualizing the design of the system, UML was used as beginning designs. Use case and class diagram were the designs for helping implementing the software. Database designs used Microsoft Visio, and all the data will be saved in MySQL database management system. For the layout designs, the online free software called mockingbird was used.

A satisfied feedback from Newave and the customers of Newave was received about the design. According to those requirements of the customer and the software designs, the software was created in the LIEKE platform. The software is fully functional as specified and it met the usability for the LIEKE customer. Newave is expected to create a great booking application for mobile devices soon.

Key words

CodeIgneiter, CMS, HMVC, LAMP, LIEKE, MVC, UML

TABLE OF CONTENTS

1 INTRODUCTION

Nowadays, people prefer to do things online. Shopping and ordering pizza online is something that almost everyone has done. One thing that has clearly been lagging behind is booking a time-based service from a small business in Finland. Sure the big chains have online reservation for doctor's appointment, haircut and car service, but there are no good overall solutions for small businesses even the demand is growing.

This project was conducted with Newave Designs Oy, which have developed their own platform for building web applications. One topic that was mentioned often in the discussions was the missing feature of time reservation module that would allow end customers to book an available time for a chosen service from a company’s booking calendar. The goal of the project is to learn how the application platform of Newave works and to design this missing feature. This Bachelor’s thesis about the design of the booking module and the company would help in the process of implementing it.

This thesis work is a mainly about software design for a booking system that can be used to offer and reserve timeslots. It is designed as a module to LIEKE platform built by Newave Designs Oy, a company, which also commissioned this thesis work. My task was to collect requirements and based on those requirements to design the data structure and needed functions to realize this service.

This work also covers much of the theory about the subjects, which are needed to learn during the process. While LIEKE is not open source and cannot be covered in great detail, so avoid it and cover the topics that LIEKE is based on will be a good choice. In this thesis, LAMP stack was explained in chapter 2.1 and was continued to other open source tools and techniques that LIEKE is built on. From front-end side this project covered the issue of user interface (UI) development with the concept of responsive design that is used to offer support for mobile devices such as tablets and smartphones.

2 SOFTWARE ENVIRONMENT ANALYSIS

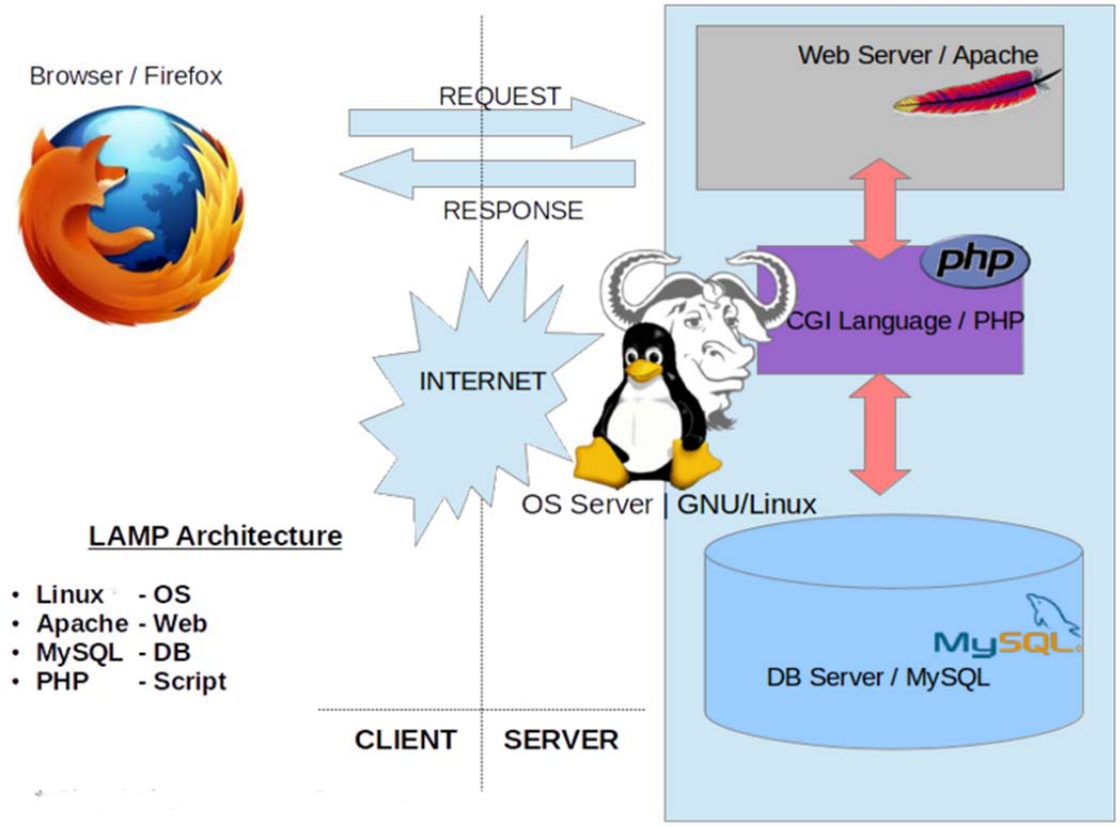
The technical requirements or the tools were used for this project were explained in this chapter. The knowledge of LAMP was mentioned at the beginning of the chapter. What the LAMP strands for, what the component of the LAMP, and how those components work with one another should be known as well. And for this project, MySQL database management system and PHP scripting language were used. The CodeIgniter with HMVC and Smarty template will be explained. LIEKE is built on CodeIgniter run as Hierarchical model–view–controller (HMVC) framework instead of traditional MVC, and uses Smarty template engine to handle dynamic layout structure that loads content modules as widgets when needed. The responsive design, which is the way of making single solution works effectively not only on desktop browsers, but also on mobile devices were described. LIEKE platform was demonstrated in the end of the chapter. This is the platform, which our booking service built for.

2.1 LAMP

LAMP is an open source Web development platform that uses Linux as the operating system, Apache as the Web server, MySQL as the relational database management system and PHP as the object-oriented scripting language. Sometimes Perl or Python is used instead of PHP. Because the platform has four layers, LAMP is sometimes referred to as a LAMP stack. Stacks can be built on different operating systems. Developers that use these tools with a Windows operating system instead of Linux are said to be using WAMP; with a Macintosh system, MAMP; and with a Solaris system, SAMP. In this project, programmers are going to use MySQL as database management system and PHP as the scripting language. (Rouse 2008.)

In order to process and develop dynamic web pages, the knowledge of three main components to creating the web page would be known. Those components are a web server, a server-side programming language, and a database. In this case, they are AMP, Apache, MySQL and PHP. GRAPH 1 shows High-level overview of LAMP components.

Internet user enters the web page address in the browser address bar, and sends request to web server (apache). Then CGI Language (PHP) works with database server (MySQL) collects the information that user needs. User's web browser uses the information that was returned from the web server to build the web page on the computer screen.



GRAPH 1. High-level overview of LAMP components (adapted from Rob 2013).

The detail and the functions of those components will be described with the following text. The first component of LAMP is Linux. Linux is an operating system that evolved from a kernel created by Linus Torvalds when he was a student at the University of Helsinki. Generally, it is conspicuous to most people what Linux is. However, both for political and practical reasons, it needs to be explained further. To say that Linux is an operating system means that it meant to be used as an alternative to other operating systems, Windows, Mac OS, MS-DOS, Solaris and others. Linux is not a program like a word processor and is not a

set of programs like an office suite. Linux is an interface between computer/server hardware, and the programs, which run on it. (Rob 2013.)

The second component of LAMP is Apache. Apache is a collaborative software development effort aimed at creating a robust, commercial-grade, and freely available source code implementation of an HTTP (Web) server (The Apache Software Foundation 2014). It is a web server that runs browser requests into resulting web pages and understands how to process PHP code. Without the web server like Apache behind it, there would be no way for web users to reach the web pages that contain the PHP language code.

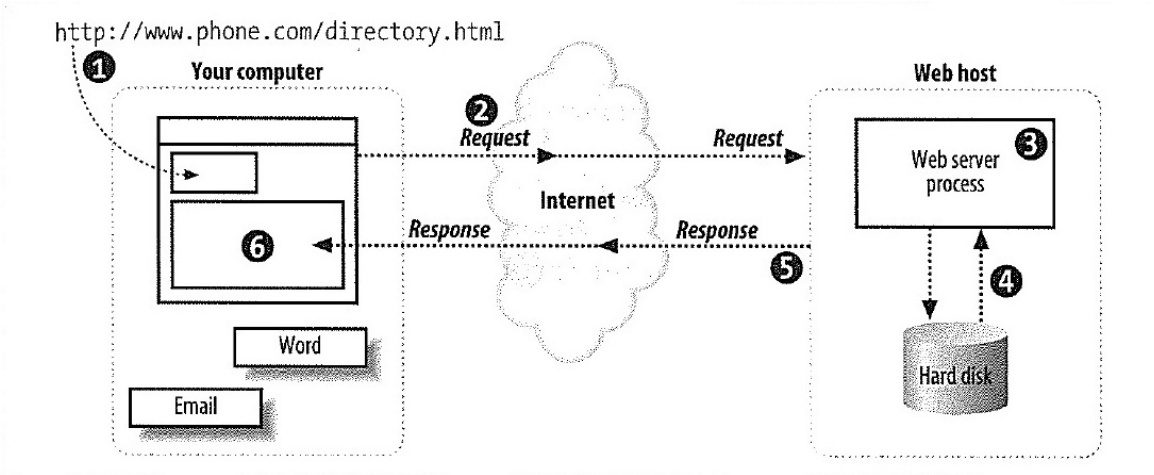
The third component of LAMP is MySQL. It is one of the main technical requirements in this project. MySQL is a database management system that stores data in separate tables rather than saving all the data in one table. This adds flexibility, as well as speed to access the data. It automates the most common tasks related to storing and retrieving specific user information based on the supplied criteria. (Davis & Phillips 2006, 6.)

The last component of LAMP is PHP. PHP is a programming language designed to generate web pages interactively on the computer serving them, called web server. PHP code runs between the requested page and the web server, adding to and changing the basic HTML output. It makes web development easy, because all the code you need is contained within the PHP framework. PHP is a great for developing web functionality. It is not a database. The database of choice for PHP developers is MySQL, which acts like a filing clerk for PHP-processed user information. (Davis & Phillips 2006, 1-2.)

MySQL is easily accessed from PHP, and they are commonly used together as they work well hand in hand. There are a lot of advantages of using PHP with MySQL. There are several factors that using PHP and MySQL together is a natural choice. First factor that is PHP and MySQL work well together. Because PHP and MySQL have been developed with each other in mind, so they are easy to use together. The programming interfaces between them are logically paired up. Working together was not an afterthought when the developers created the PHP and MySQL interfaces. Second factor is PHP and MySQL

have open source power. As they are both open source projects, PHP and MySQL can both be used for free. MySQL client libraries are no longer bundled with PHP. Advanced users have the ability to make changes to the source code, and therefore, change the way the language and programs work. Third factor is PHP and MySQL have community support. There are active communities on the Web in which user can participate and they will answer user's questions. User can also purchase professional support for MySQL if user needs it. Fourth factor is PHP and MySQL are fast. Their simplicity and efficient design enables faster processing. And the last factor is PHP and MySQL do not stall with unnecessary details. Users do not need to know all of the low-level details of how the PHP language interfaces with the MYSQL database, as there is a standard interface for calling MYSQL procedures from PHP. Online APIs at http://www.php.net offer an unlimited resource. (Davis & Phillips 2006, 3.)

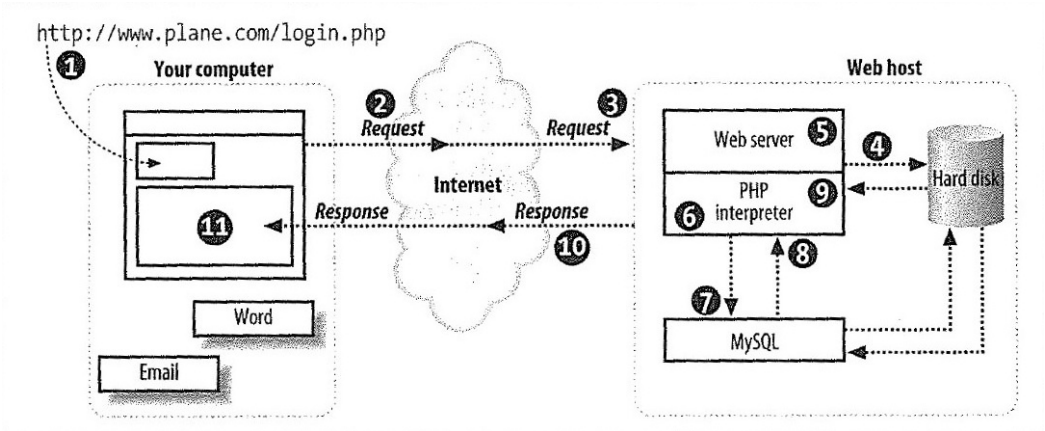
As last paragraph explained, PHP and MySQL are related to each other. The question is appeared that how they work on a web server. This processing of the PHP on the server is called server-side processing. When user requests a web page, the whole chain of events are triggered. GRAPH 2 illustrates this interaction between your computer and the web server (host of the web site).



GRAPH 2. While the user only types in a URL and hits Enter, there are several steps that occur behind the scenes to handle that request (adapted from Davis & Phillips 2006, 10).

User enters a web page address in the browser's location bar. Then the browser breaks apart that address and sends the name of the page to the web server. During next step, the address, which user types in http://www.phone.com/directory.html, and it requests the page *directory.html* from www.phone.com. After this, in the third step, a program on the web server, called the web server process, takes the request for *directory.html* and looks for this specific file. The web server reads the *directory.html* file from the web server's hard drive. Then the web server returns the contents of *directory.html* to user's browser. Finally, user's web browser uses the HTML markup that was returned from the web server to build the rendition of the web page on the computer screen. (Davis & Phillips 2006, 10.)

The HTML file called *directory.html* (requested in GRAPH 2) is called a static web page. It is static because everyone who requests the *directory.html* page gets exactly the same page. But for the web server to customize the returned page, PHP and MySQL are added to the mix. GRAPH 3 illustrates the extra steps that occur in the chain of events on the web host. (Davis & Phillips 2006, 11.)



GRAPH 3. The PHP interpreter, MySQL, and the web server cooperate to return the page (adapted from Davis & Phillips 2006, 11).

As shown in GRAPH 3, User first enters a web page address in the browser's location bar. Then user's browser breaks apart that address and sends the name of the page to the host. For example, http://www.phone.com/directory.html requests the page *login.php* from www.phone.com. The web server process on the host receives the request for *login.php*. Then the web server reads the *login.php* file from the host's hard drive. The web server detects that the PHP file is not just a plain HTML file, so it asks another process, which is the PHP interpreter to process the file. Then the PHP interpreter executes the PHP code that it finds in the text it received from the web server process. Included in that code are calls to the MySQL database. PHP asks the MySQL database process to execute the database calls. The MySQL database process returns the results of the database query. After that the PHP interpreter completes execution of the PHP code with the data from the database and returns the results to the web server process. And then the web server returns the results in the form of HTML text to the user's browser. The user's web browser uses the returned HTML text to build the web page on the user's screen. This seems quite complicated, but all of this processing happens automatically every time a web page with PHP code is requested. (Davis & Phillips 2006, 12.)

2.2 CodeIgniter with HMVC and Smarty template engine

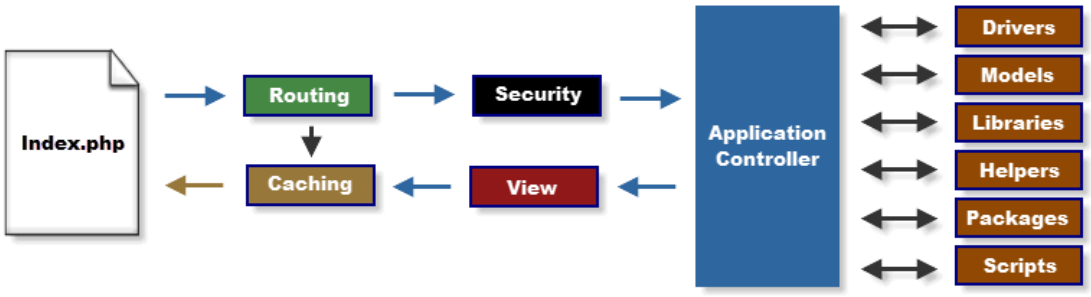
CodeIgniter is a foundation that LIEKE is build on, so it is good to know what is it and how to leverage it. Newave originally chose it, because of its clean and well-commented source code and great documentation. It is a web application framework designed for rapid development of dynamic web sites with PHP. It is licensed under an Apache/BSD-style open source license meaning that is free to use and edit even in commercial use. CodeIgniter comes with a rich set of libraries for commonly needed web development tasks, like routing, maintaining sessions and accessing a database, as well as a simple interface and logical structure to access these libraries. A base system can be easily extended through the use of the libraries, the helpers, or through the class extensions as well as the system hooks. (Ellislab Inc 2012.)

CodeIgniter uses the Model-View-Controller (MVC) design pattern (Ellislab Inc 2012), which allows separation between logic and presentation. CodeIgniter is not too strict about using MVC and it is actually possible to make software using just controller classes. MVC pattern has its limitations when it comes to building large modular applications so in LIEKE there is an extension that causes CodeIgniter run as Hierarchical model–view–controller (HMVC) framework instead of traditional MVC. HMVC structure increases code modularity, re-usability, and helps to maintain a better separation between different modules of the application. For example this booking module will have its own folder in the file system that contains every controller, model and view that is needed for booking functions and other controllers can call them when needed so there is no need for duplicating code or making external libraries or helpers to serve these functions.

CodeIgniter comes with template parser class to help to separate the actual PHP code from views. The default template parser class is very simple with limited functionality and does not offer critical functions such as extending layouts and calling custom layout functions. In LIEKE, the default template parser class is replaced by the Smarty template engine. This makes it possible to handle dynamic layout structures that load content modules as widgets when needed.

2.2.1 Dataflow in CodeIgniter

All application requests are first routed to index.php by using Apache's mod rewrite. This guarantees that URLs used by CodeIgniter are clean and search-engine friendly. CodeIgniter uses a segment-based approach for calling resources. Basic example: http://example.com/articles/1/codeigniter-routing-example. Breaks down this to segments: "example.com" is domain that runs the application and in its root contains the index.php that all requests go through. "articles" is the name of a controller that serves the articles. "1" is the id number of the article in the database and "codeigniter-routing-example" is the name of the article in it's URL form.

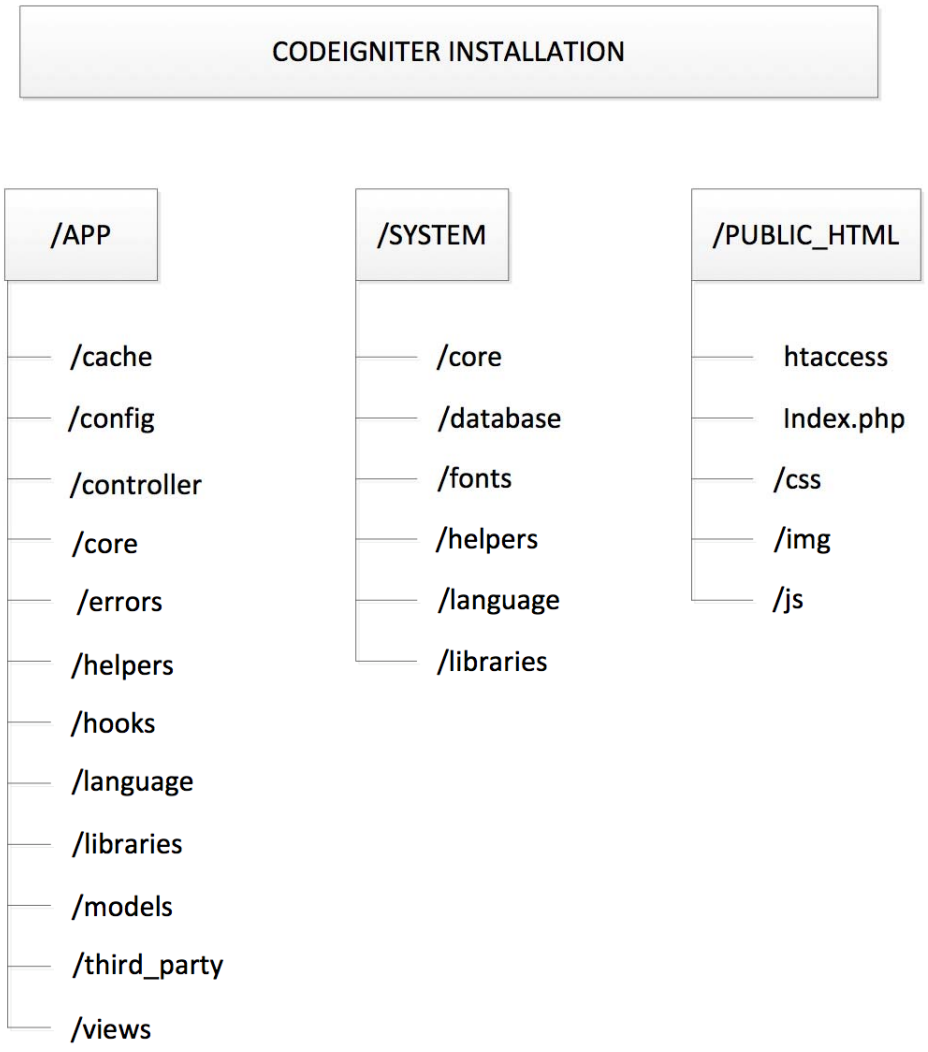


GRAPH 4. Data flow in CodeIgniter (adapted from Ellislab Inc 2012).

As shown in GRAPH 4, the index.php serves as the front controller, initializing the base resources needed to run CodeIgniter. The Router examines the HTTP request to determine what should be done with it. If a cache file exists, it is sent directly to the browser, bypassing the normal system execution. Before the application controller is loaded, the HTTP request and any user submitted data is filtered for security. The Controller loads the model, core libraries, helpers, and any other resources needed to process the specific request. The finalized view is rendered then sent to the web browser to be seen. If caching is enabled, the view is cached first so that on subsequent requests it can be served. (Ellislab Inc 2012.)

2.2.2 File structure in CodeIgniter

Following GRAPH 5 illustrates typical CodeIgniter installation that consists of 3 folders. System folder holds the CodeIgniter system files that should not be edited. App folder is where the development happens and public_html is the www-root that holds the front controller (index.php) and publicly available resources such as images, CSS and JavaScript files. It is also a common practise to place the views folder under public_html to make clear separation between backend and frontend development.

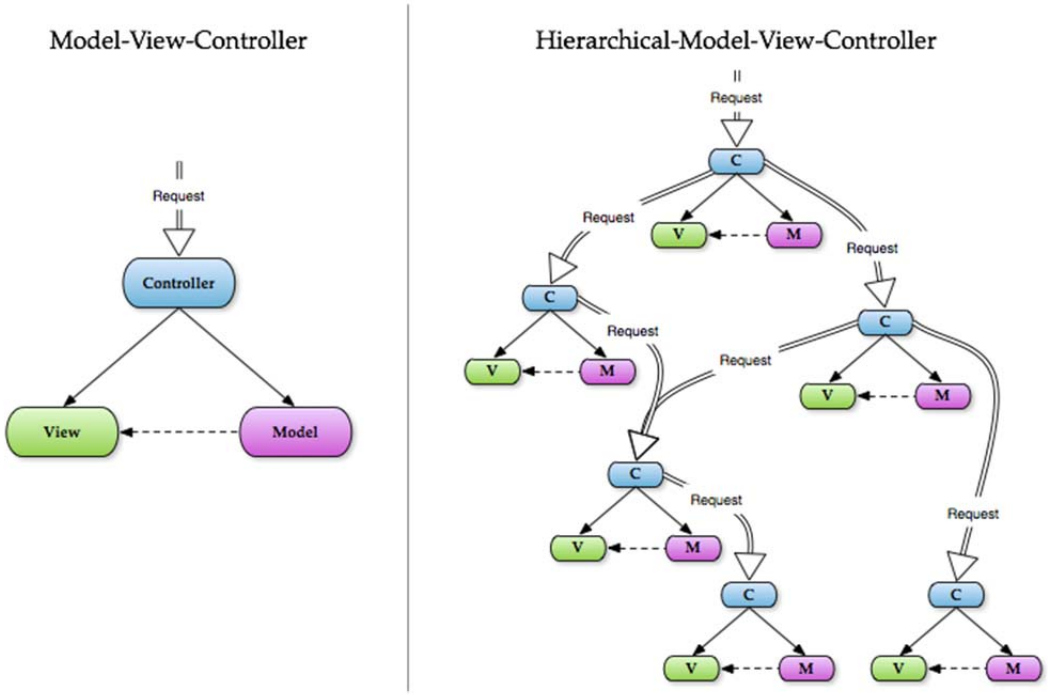


GRAPH 5. CodeIgnter filestructure

In LIEKE there is one more folder in APP that is called modules. This folder holds the modules and sub-modules that can be used inside LIEKE based applications. Another difference is that the folders under public_html contain only CSS, JavaScript and image files that are common to all LIEKE installations. There is one more folder called site that contains application specific views, CSS, JavaScript, image and another files that may be needed.

2.2.3 Hierarchical model-view-controller (HMVC)

Model-View-Controller (MVC) design pattern divides a software application into three interconnected parts (models, views and controllers), so this is to separate internal representations of information from the ways that information is presented to or accepted from the user (Techportal 2010). MVC separation helps to manage complex applications, because developer can focus on one aspect at a time. Hierarchical-MVC structure adds to this by increasing code modularity, re-usability, and help to maintain a better separation between different parts of the software.



GRAPH 6. The difference between MVC and HMVC (adapted from Techportal 2010).

The H in HMVC (Hierarchical-Model-View-Controller) means hierarchy. That means it is basically the collection of MVC (Model-View-Controller) structures that are used to modularize software in smaller chunks, each having its own models, views and controllers. GRAPH 6 illustrates this difference and how these modules can send requests to another modules.

The Model is the part of the application that handles the logic for the application data (Refsnes Data 2014). In CodeIgniter models are usually used to retrieve, update and store data from/to a database. CodeIgniter uses Active Record Database Pattern for database interactions making it possible to switch underlying database engine without breaking the application. In LIEKE all the models extend a general model that already has all the basic CRUD functions so that only the more complex multiple table's queries need to be handwritten by a developer.

The View is the part of the application that handles the display of the data (Refsnes Data 2014). In CodeIgniter views are composed out of HTML templates that are filled with data passed by Controller and usually loaded from models. LIEKE uses Smarty template engine so that designers can literally place some simple logic directly to the views. Smarty supports all basic control structures, calculations and has a good set of modifiers that can be used for example to capitalize the string that is passed on it by Controller.

The Controller is the part of the application that receives the request from browser or another controller and handles this interaction (Refsnes Data 2014). Controllers pass data to the views and then read user inputted data from a view, perform necessary validations and modification for the data and then send it to the model. The controller is responsible for responding to user input and performs interactions on the data model objects (Tutorialspoints 2014). The controller receives the input and it validates the input and then performs the business operation that modifies the state of the data model (Tutorialspoints 2014).

2.2.4 View with Smarty template engine

A template engine is a software component that is designed to combine templates with a data model to produce result documents (Niemeyer 2002). The language that the templates are written in is known as a template language. Smarty is a template engine for PHP. And

it is more sophisticated than CodeIgniter's native template parser. Smarty facilitates a manageable way to separate application logic and content from its presentation. This means that redesigning the application templates would require no change to the application logic and changes in application logic do not affect the template.

2.3 Responsive design

Depending of target audience of site or application, portion of mobile users can easily supersede desktop users making it extremely important to support variety of mobile devices. Implementing and maintaining the enormus complete different solution just for mobile users can be a correspondingly huge amount of work, and tweaking all elements to make sure they display properly and running tests across a variety of devices.

With the increasing of website and blog visitors browsing the web from smartphones and tablets, web designers need to adapt to this by creating websites that are optimized for mobile browsers (Bryant 2014). The standard way of designing mobile websites has traditionally been to create different versions of the website for tablets and smartphones, meaning that programmer may create a standard website that is intended for desktop computer viewing, a tablet website that is optimized for devices like the iPad and Android tablets, and a smartphone microsite that is optimized for devices like the iPhone and Nexus smartphones (Bryant 2014). Responsive design is a way of making a single solution that works effectively on both desktop browsers and a variety of mobile devices on the market. GRAPH 7 illustrates how the responsive architecture gives the best quality experience - whether on a smartphone, tablet, netbook or desktop, and regardless of the operating system.

GRAPH 7. Illustration of responsive layout that shares the same content, but it switches layout structure to accommodate the device view it (Bryant 2014).

People who are on-the-go have very different needs than those sitting behind a desk. Responsive solutions re-organize themselves automatically according to the device using them, so that the same application provides a great experience everywhere. Desktops get a full-blown interface with large images and animations. Smartphones get a simplified version that runs fast without the bells and whistles. Tablets and netbooks get something in between. (AllsizeWeb 2013)

Imagine a website of a railway company. One user might visit the desktop version, hoping to find a special discount for a holiday trip. At the same time, someone else might be running to catch a train while checking the mobile version, hoping to find out which platform is the right one. One solution, two situations, and two completely different user scenarios. In order to create a useful service for all users, programmers need to consider that people will use different devices in different circumstances and with different goals. This adds another layer to the traditional user-case driven software design model.

Responsive design is achieved by breaking down the layout into what is essentially a grid, and then resizing elements on that grid by percentages, rather than hard units like pixels. GRAPH 8 shows a simple web design that follows the 12-grid design system. With some

additional media queries, it is possible to rearrange and keep every part in the right position. The result is a site or application that displays fairly well on wide range of devices. The design process itself is a bit of the challenge because programmer has to think in terms of objects rather than pages. These objects or blocks of content are rearranged in different grid sizes.

GRAPH 8. 12-grid design (adapted from Newave Designs Oy 2014).

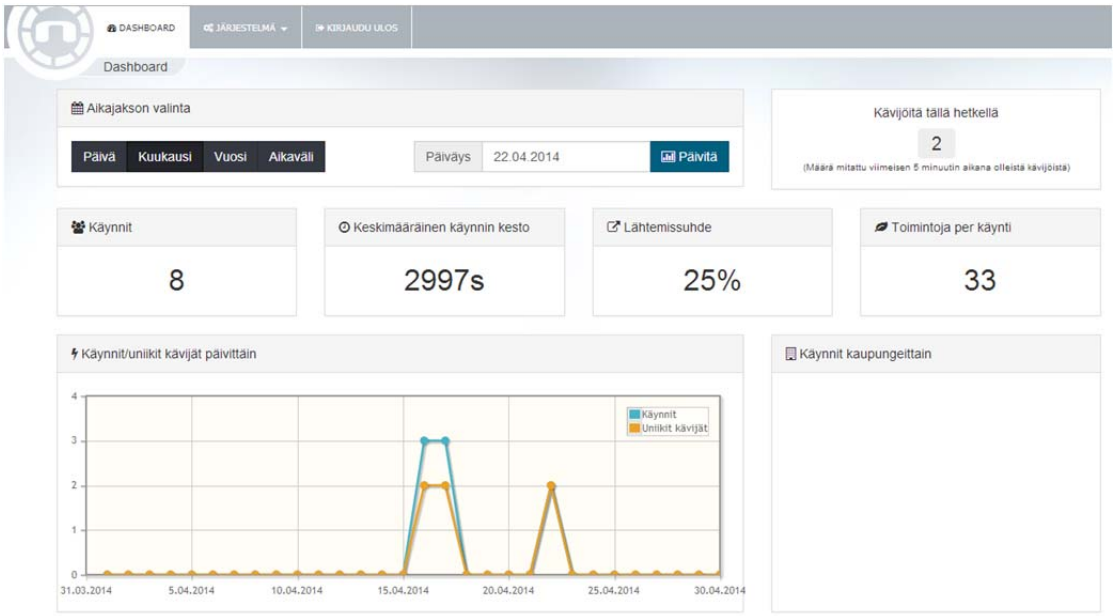
Twitter has an open source project called Bootstrap that is a free collection of tools for creating responsive websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It was developed by Mark Otto and Jacob Thornton at Twitter as a framework to encourage consistency across internal tools. In August 2011 Twitter released Bootstrap as open source. In February 2012, it was the most popular GitHub development project. (Otto 2012)

Bootstrap is the most popular front-end framework for developing responsive, mobile first projects on the web (Core team 2012). It includes a responsive, mobile first fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases. This is a big help and standardizes the layout production. Bootstrap is used in LIEKE and every solution that is built on it.

2.4 LIEKE

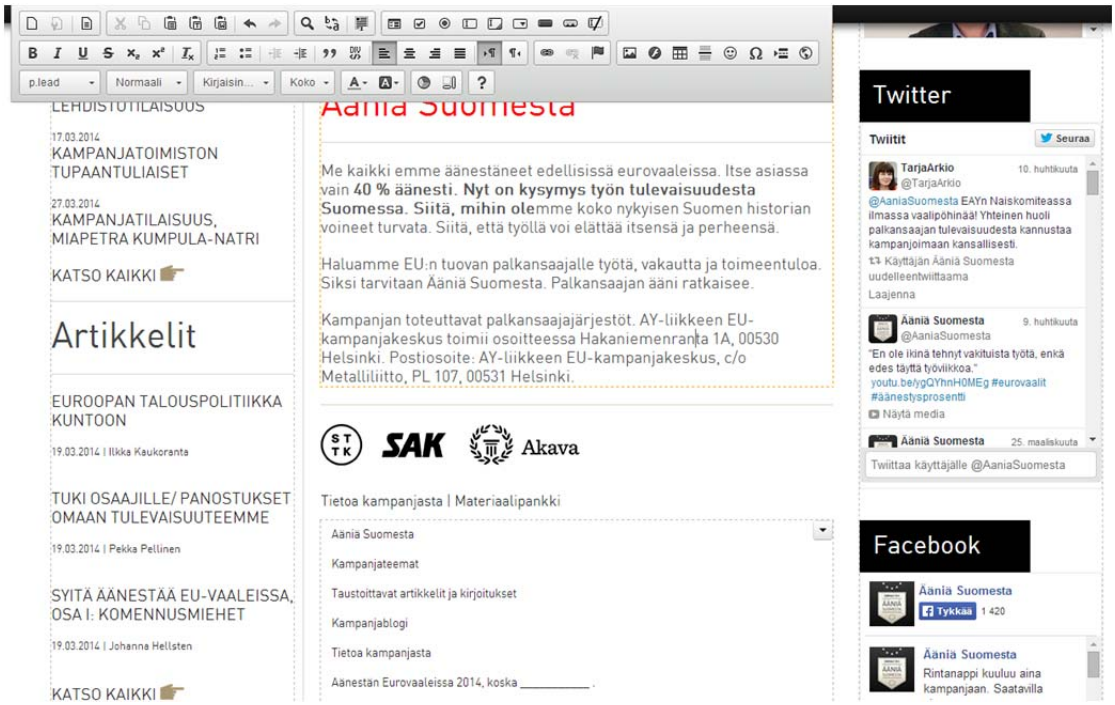
LIEKE is a platform and content management system for building dynamic websites and applications offering a broad range of features and services including user administration, translation tools, content publishing and online shopping among others. It is built by Newave in Kokkola, which is an IT company founded in 2004. LIEKE is not open source, but it is built by using many publicly available open source frameworks and libraries such as CodeIgniter in the backend and Smarty, Bootstrap and jQuery in the frontend.

Newave does not directly sell LIEKE as a product, but builds solutions to it and sells them. Its main purpose has been to speed up and standardize internal development process. The main market is in the advertisement industry where the designed campaigns can now be easily repeatedly used with a change of layout. Basic installation of LIEKE only includes dashboard (GRAPH 9), user management, translations, settings, filemanager, module management and system tools. All another components are modules that need to be enabled.



GRAPH 9. LIEKE Dashboard with usage statistics without any module installed (adopted from Newave Designs Oy 2014)

First common module that almost every project uses is the CMS (Content Management System) module. CMS module handles navigation, pages, news, image galleries and form handling. Second commonly used modules are webshop, blog and travel. As can be presented in the GRAPH 10, with CMS module of LIEKE, LIEKE customers are able to manage their website or webshop by themself. They can edit news, upload the new image into the gallery, and can also arrange the page looking.



GRAPH 10. LIEKE CMS in-place editor enables fast content editing (adapted from Newave Designs Oy 2014)

LIEKE is build to support responsive, dynamic layouts that allow extending per-page basis. This means easy manipulation of widgets and content. GRAPH11 and GRAPH12 illustrate in-place drag'n drop widgets that can occupy any space that they are placed in. Compares GRAPH 11 and GRAPH 12, LIEKE in-place editor state before and after moving banner and navigation widgets to new locations, how the navigation adapt to its new given space can be noticed.

GRAPH 11. LIEKE in-place editor state before moving banner and navigation widgets to new locations (adapted from Newave Designs Oy 2014)

GRAPH 12. LIEKE in-place editor state after moving banner and navigation widgets to new locations (adapted from Newave Designs Oy 2014)

Building a new module requires certain steps that have to be taken. Each module has its own collection of models, views and controllers. Each controller extends the LIEKE core controller that provides security and access management. Each model extends to LIEKE core model that provides basic crud functions and version control of changes in all tables.

Controllers are divided to public and non-public controllers. Public controllers provide functions that anybody viewing the site or application can freely access without requiring login. Non-public controllers require the authorized user that has privileges to the given module. Non-public controllers are usually used by solution owner, not by the end user.

Views are basically just HTML templates with the Smarty template tags. Module specific JavaScript files are also run through template engine to give them easy access to data without additional HTTP requests to obtain the initial data after the page is loaded. Views are extendable much the same action as the classes in PHP. Any block can be overwritten or new content can be appended to them before the view is rendered the return to the requesting browser. Further technical details of LIEKE are under confidentiality agreement so they cannot be covered in this thesis work.

3 DESIGNS

The design represents software design in this thesis work. It is the activity before programming, which is usually called coding or implementing. For this project, the design was for the booking service in the LIEKE platform. Before the start of the design, software designer should know the requirements from the customer. The function and views from this software/system are be considered by the designers. It is the first step to analyse requirements, ant to know what needs to be done to delight the customers.

According to requirements that customer gives, designers can start to create the designs that developers can understand. Usually, designer uses Unified Modelling Language (UML). In this project, software designer made Use case diagram, which is one of the behavioural UML diagrams, and Class diagram, which is one of the structural UML diagrams. From these two diagrams, the customer and also the developer will better understand the booking service. After built the design by using UML, designer starts to design the database, which is one of the main parts of this booking service. In the end, designer makes the layout design, which is the model of the user interface.

3.1 Requirements

Requirements are the basis of all software applications as they focus on what the application must do. Requirements are elicited from the customer outlining what they wish the system to do, and recorded in a language the customer understands. A requirements document then becomes the contract between the customer and the software developers on the product that will be delivered. (Williams & Heckman 2008.)

Requirements can be described by many manners. For formal requirements documents the most common division is into functional and non-functional requirements. Functional requirements describe how a system interfaces with its environment, i.e. the services the

system provides, how to react to inputs (both correct and incorrect and so on. Non-functional requirements describe restrictions on the system. These include time constraints for certain activities (especially important in real-time systems), security, and privacy. Other constraints on the system such as operating system, programming language, or other incorporated software applications are also included in a requirements document. Priorities can be assigned to each requirement by the customer to help define system development. (Williams & Heckman 2008.)

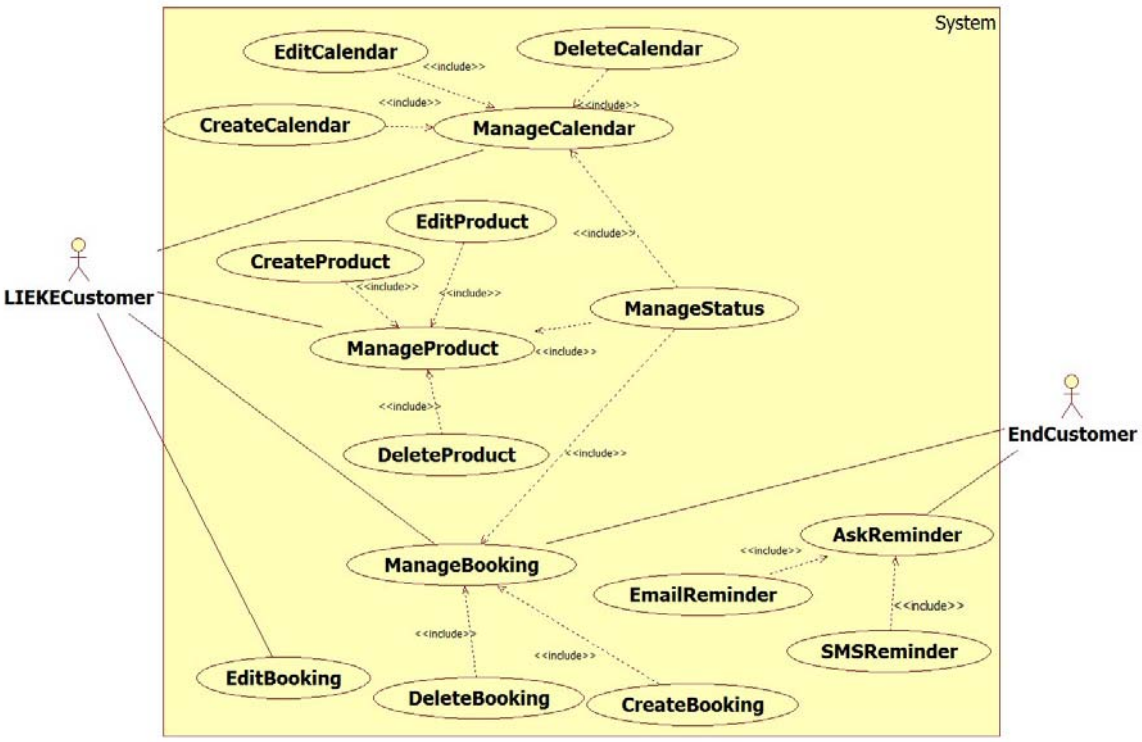
In this project, software designer needs to know the requirements elicitation, which is the client/customer's specification of the system/software. Then designer needs to analyse the specifications from the customers, which is converting 'customer-facing' requirements into equivalents that developers can understand. That usually will descript to developers by using the Unified Modelling Language (UML) to visualize the design of the software/system, which suits the requirements. The developer then can understand the requirements better and according to the design start the implementation of the software.

The LIEKE customers should be able to create and manage calendars and the calendars' status/visibility. This means that LIEKE customer can create a new calendar, edit the calendar, and also delete the calendar. They can set the calendar status/visibility to public or private. And they should be able to manage working hours, which are the start time of working hour and the end time of working hour for each day. The lunch and coffee breaks will be in certain time everyday. They should be able to create and manage their products and the status/visibility of the products. The LIEKE customer can create a new product for the calendar, edit the details of the product (name, description, duration, price, position, and so on), and they can delete the exiting product. They also can set the product status/visibility to public or private. The LIEKE customers should be able to manage the bookings. They can create/add new bookings edit booking and also delete bookings from the lists of the calendars. This means the LIEKE customers can create new booking in the calendar, and edit the booking by inserting details, such as the information of customer and product, and time, which their customer would like to book. And they can require the reminders if they want. And also can delete the bookings for their customers. They also can change the status of the bookings in the system to "coming", "process", or "done".

The end customers should be able to manage the bookings too. They can create/add new bookings and also delete bookings to the calendar they choose. This means that the end customer can create new booking in the different calendar of the software, and choose product and time, also insert their detail such as their name, email, mobile number, and leave messages. They also can choose reminders, which will remind them by sending email or SMS message if they want. And also they can delete the bookings with the booking code they saved when they make the booking, and they can explain the reason of cancelling the booking.

3.2 User Case design and documentation

Use cases and user stories are also used to document system requirements. Use cases show the interaction between users (both human and non-human) and the different tasks of the system. User stories are short task descriptions written by the customer, and they are especially useful for defining development priorities. (Williams & Heckman 2008.) In this project, the use case supports developer to understand the requirements of the customers, and help them to implement the right and suitable booking service for both LIEKE customers and the end customer. GRAPH 13 presents the use case design for the booking service in LIEKE platform.



GRAPH 13. Use case design of booking service for LIEKE platform

From the use case design above shown in GRAPH 13, both the customer and the developer understood this project, especially, for the developer to continue the future implementation stage. The customer will understand what kind of software they will get, who will be the user of the software and what kind of action or functions will be used in the software. And the developer will understand the requirements of the customer or the project better.

As the use case diagram shows, there are two actors, which are the LIEKECustomer and EndCustomer. For EndCustomer, there are two main use cases, which are AskReminder and ManageBooking. And for LIEKECustomer, there are three main use cases, which are ManageCalendar, ManageProduct and ManageBooking. There is a use case for LIEKECustomer and is separate from EndCustomer, which is EditBooking. Because that the rights of the ManageBooking are different with the EndCustomer in this software. The EndCustomer only can create and delete bookings, but LIEKECustomer also can edit bookings.

For the customer and the developer understand the use cases, the description of use case design is also one part of the use case design. The description will analyse every use case and is shown as a table with every use case. The tables below will be the descriptions for the use case designs of the booking service in LIEKE platform. Table 1 and table 2 are the description/documentation for the use cases of both the LIEKECustomer actor and the EndCustomer actor. The LIEKECustomer and the EndCustomer have the same use cases.

Table 1. Description of CreateBooking use case

The name of usecase CreateBooking
Version 1.0
Actors LIEKECustomer & EndCustomer
Frequency On request
Pre-conditions Actors have the booking information at hand.
Explanation of the use case The information of the booking needs to be inserted into system. [**EXCEPTION:** Enter wrong name form]. [**EXCEPTION:** Enter wrong email address form]. [**EXCEPTION:** Enter wrong mobile phone form].
Exceptions **ENTER WRONG NAME FORM**: User entered number in name form, such as numbers, and other characters then letter. **ENTER WRONG EMAIL ADDRESS FORM**: User did not enter validity email address. **ENTER WRONG MOBILE PHONE FORM**: User put other character then number.
Post-conditions Booking has been created in the LIEKE and waiting for next step.

Table 1 presents the CreateBooking use case, which is one of the use cases to be included in the ManageBooking use case. Both actors can create the booking. They also are required to insert the details of the bookings, such as, the calendar and the product, the start time and end time of the booking, customer's name, email, mobile number, and leave message. After the booking has been created, actors will get the booking code for deleting the booking. Because of the difference of the rights between the LIEKE customer and the End customer, only LIEKE customer can edit the booking in the future.

Table 2. Description of DeleteBooking use case

The name of usecase DeleteCalendar
Version 1.0
Actors LIEKECustomer & EndCustomer
Frequency On request
Pre-conditions Actors have the booking code at hand for delete.
Explanation of the use case The information of the booking needs to be deleted from system. [**EXCEPTION:** Code of the booking is empty]. [**EXCEPTION:** Code of the booking does not exist/ is wrong].
Exceptions **CODE OF THE BOOKING IS EMPTY**: Forgot enter the code for the booking. **CODE OF THE BOOKING DOES NOT EXIST/ IS WRONG**: The booking code does not exist/ is wrong.
Post-conditions Booking has been deleted from LIEKE.

Table 2 presents the DeleteBooking use case, which is one of the use cases to be included in the ManageBooking use case. It means both LIEKECustomer and EndCustomer can delete the booking by entering the booking code. Then the user can delete the booking, which is shown with this individual code. User should be careful with this use case. Once

user deletes the booking, it will be deleted from database at the same time. The user needs to create a new booking with same information if the user still wants the same booking after deleting the booking.

Table 3. Description of CreateCalendar use case

The name of usecase CreateCalendar
Version 1.0
Actors LIEKECustomer
Frequency On request
Pre-conditions Actor is logged in LIEKE and has new calendar name at hand.
Explanation of the use case The information of the new calendar needs to be inserted into system. [**EXCEPTION:** Name of the calendar is empty]. [**EXCEPTION:** Name of the calendar already exists].
Exceptions **NAME OF THE CALENDAR IS EMPTY**: Forgot to enter the name into the calendar. **NAME OF THE CALENDAR ALREADY EXISTS**: The calendar with the same name entered has been created already.
Post-conditions Calendar has been created in the LIEKE and waiting for next step.

From Table 3 to Table 9 show the descriptions/documentations for the use cases of the LIEKECustomer only. Table 3 presents the CreateCalendar use case, which is one of the use cases to be included in the ManageCalendar use case. It is only for creating the calendar with typing in the name of the calendar. When user creates the calendar, the information, which is the name of the calendar, will be saved in the database at the same time. The actor can edit or delete the calendar in the future (the name of the calendar that is only detail is required to edit or delete the calendar).

Table 4. Description of EditCalendar use case

The name of usecase EditCalendar
Version 1.0
Actors LIEKECustomer
Frequency On request
Pre-conditions Actor is logged in LIEKE and has the calendar information at hand.
Explanation of the use case The information of the calendar needs to be edited into system. [**EXCEPTION:** Name of the calendar already exists]. [**EXCEPTION:** Enter wrong date].
Exceptions **NAME OF THE CALENDAR ALREADY EXISTS**: The calendar with the same name entered has been created already. **ENTER WRONG DATE**: The end date is before the start date or the date in the past.
Post-conditions Calendar has been edited and saved in the LIEKE and waiting for next step.

Table 4 presents the EditCalendar use case, which is one of the use cases to be included in the ManageCalendar use case. It means LIEKECustomer can edit the calendar by filling in all the details, such as, the validity date for the calendar, the working hours for each day, the status of the calendar. After LIEKECustomer has edited the calendar, the new information of the calendar will be saved in the database at the same time. The LIEKECustomer also can edit/change the name of the calendar. The system will check the name, which actor entered has been created already or not. The actor can delete the calendar in the future (the name of the calendar that is only detail is required to delete the calendar).

Table 5. Description of DeleteCalendar use case

The name of usecase DeleteCalendar
Version 1.0
Actors LIEKECustomer
Frequency On request
Pre-conditions Actor is logged in LIEKE and has the calendar name for delete.
Explanation of the use case The information of the calendar needs to be deleted from system. [**EXCEPTION:** Name of the calendar is empty]. [**EXCEPTION:** Name of the calendar does not exist].
Exceptions **NAME OF THE CALENDAR IS EMPTY**: Forgot to enter the name into the calendar. **NAME OF THE CALENDAR DOES NOT EXIT**: The name of the calendar does not exit.
Post-conditions Calendar has been deleted from LIEKE.

Table 5 presents the DeleteCalendar use case, which is one of the use cases to be included in the ManageCalendar use case. It means LIEKECustomer can delete the calendar by searching the calendar by name (the name of the calendar that is only detail is required to delete the calendar). LIEKECustomer find the calendar by the name, and he/she can delete the calendar. Actor should be careful with this use case. Once user deletes the calendar, it will be deleted from the database at the same time. All the details of the calendar are deleted from database as well. The user needs to create a new calendar with the same information as the deleted calendar, if the user still wants the same calendar after deleting the booking.

Table 6. Description of CreateProduct use case

The name of usecase CreateProduct
Version 1.0
Actors LIEKECustomer
Frequency On request
Pre-conditions Actor is logged in LIEKE and has new product name at hand.
Explanation of the use case The information of the new product needs to be inserted into system. [**EXCEPTION:** Name of the product is empty]. [**EXCEPTION:** Name of the product already exists].
Exceptions **NAME OF THE PRODUCT IS EMPTY**: Forgot to enter the name for the product. **NAME OF THE PRODUCT ALREADY**: The product with the same name entered has been created already.
Post-conditions Product has been created in the LIEKE and waiting for next step.

Table 6 presents the CreateProduct use case, which is one of the use cases to be included in the ManageProduct use case. It is only for creating the product with inserting the name of the product to the place that product name is required. When user creates the product, the information of the new calendar, which is the name of the product, will be saved in the database at the same time. The system will check the name, which actor entered has been created already or not. The actor can edit or delete the product in the future (the name of the product that is only detail is required to edit or delete the product).

Table 7. Description of EditProduct use case

The name of usecase EditProduct
Version 1.0
Actors LIEKECustomer
Frequency On request
Pre-conditions Actor is logged in LIEKE and has product information at hand.
Explanation of the use case The information of the product needs to be edited into system. [**EXCEPTION:** Name of the product already exists]. [**EXCEPTION:** Enter wrong price form].
Exceptions **NAME OF THE PRODUCT ALREADT EXSITS**: The calendar with the same name entered has been created already. **ENTER WRONG PRICE FORM**: User inserts other character then numbers.
Post-conditions Product has been edited and saved in the LIEKE and waiting for next step.

Table 7 presents the EditProduct use case, which is one of the use cases to be included in the ManageProduct use case. It means the user can edit the product by filling in more details or changing the details, such as, the description, duration, price, position and status (which the product's visibility for customer) of the product after the product has been created. After user edited the product, the new information of the product will be saved in the database at the same time. The LIEKECustomer also can edit/change the name of the product. The system will check the name, which actor entered has been created already or not. The actor can delete the product in the future (the name of the product that is only detail is required to delete the product).

Table 8. Description of DeleteProduct use case

The name of usecase DeleteProduct
Version 1.0
Actors LIEKECustomer
Frequency On request
Pre-conditions Actor is logged in LIEKE and has the product name for delete.
Explanation of the use case The information of the product needs to be deleted from the system. [**EXCEPTION:** Name of the product is empty]. [**EXCEPTION:** Name of the product does not exist].
Exceptions **NAME OF THE PRODUCT IS EMPTY**: Forgot to enter the name for the product. **NAME OF THE PRODUCT DOES NOT EXIT**: The name of the product does not exit.
Post-conditions Product has been deleted from the LIEKE.

Table 8 presents the DeleteProduct use case, which is one of the use cases to be included in the ManageProduct use case. It means LIEKECustomer can delete the product by searching the product by name (the name of the product that is only detail is required to delete the product). LIEKECustomer find the product by the name, and he/she can delete the product. User should be careful with this use case. Once user deletes the product, it will be delete from database at the same time. All the details of the product are deleted from database as well. The user needs to create a new product with the same information as the deleted product, if the user still wants the same product after deleting the booking.

Table 9. Description of EditBooking use case

The name of usecase EditBooking
Version 1.0
Actors LIEKECustomer
Frequency On request
Pre-conditions Actor is logged in LIEKE and has the booking information at hand.
Explanation of the use case The information of the booking needs to be edited into the system. [**EXCEPTION:** Enter wrong email address form]. [**EXCEPTION:** Enter wrong mobile phone form].
Exceptions **ENTER WRONG EMAIL ADDRESS FORM**: User did not enter validity email address. **ENTER WRONG MOBILE PHONE FORM**: User put other character then number.
Post-conditions Booking has been edited and saved in the LIEKE and waiting for next step.

Table 9 presents the EditBooking use case. It means the LIEKECustomer can edit the bookings by filling in the new details of the booking, such as, the calendar and the product, the start and end time, customer's name, email, Mobile number. This use case is not included in the ManageBooking use case, because only the LIEKECustomer is able to edit the booking after it has been created. The EndCustomer cannot edit the booking after it has been created. The EndCustomer needs to delete the booking first with the booking code, and create new booking for editing new information, or the EndCustomer need to contact the LIEKECustomer to edit the new information in to the booking.

Table 10. Description of EmailReminder use case

The name of usecase EmailReminder
Version 1.0
Actors EndCustomer
Frequency On request
Pre-conditions Booking exists and the starting time of the booking is approaching for a certain time.
Explanation of the use case An email will be sent to the actor for reminding the booking for the certain time as he/she asked [**EXCEPTION:** Email address is wrong].
Exceptions **EMAIL ADDRESS IS WRONG**: Entered wrong email address when booking.
Post-conditions Remind booking email has been sent to actor for the certain time.

Table 10 and Table 11 are the descriptions/documentations for the use cases of the EndCustomer only. Table 10 presents the EmailReminder use case, which is one of the use cases to be included in AskReminder use case. It will send an email to remind the customer about the booking for the certain time, which user asked for when he/she was creating the booking. LIEKECustomer is not necessary to require the reminder in this project. Because that the LIEKECustomers are the main users for this booking service. They are the users, who order this software for their business. They will stay in their office and waiting for the customers coming by the ordering time. The booking service can be improved later if the EndCustomer requires the reminder for them also.

Table 11. Description of SMSReminder use case

The name of usecase SMSReminder
Version 1.0
Actors EndCustomer
Frequency On request
Pre-conditions Booking exists and the starting time of the booking is approaching for a certain time.
Explanation of the use case A mobile text message will be sent to actor for reminding the booking for the certain time as he/she asked [**EXCEPTION:** Mobile phone number is wrong].
Exceptions **MOBILE PHONE NUMBER IS WRONG**: Entered wrong mobile phone number when booking.
Post-conditions Remind booking text message has been sent to actor for the certain time.

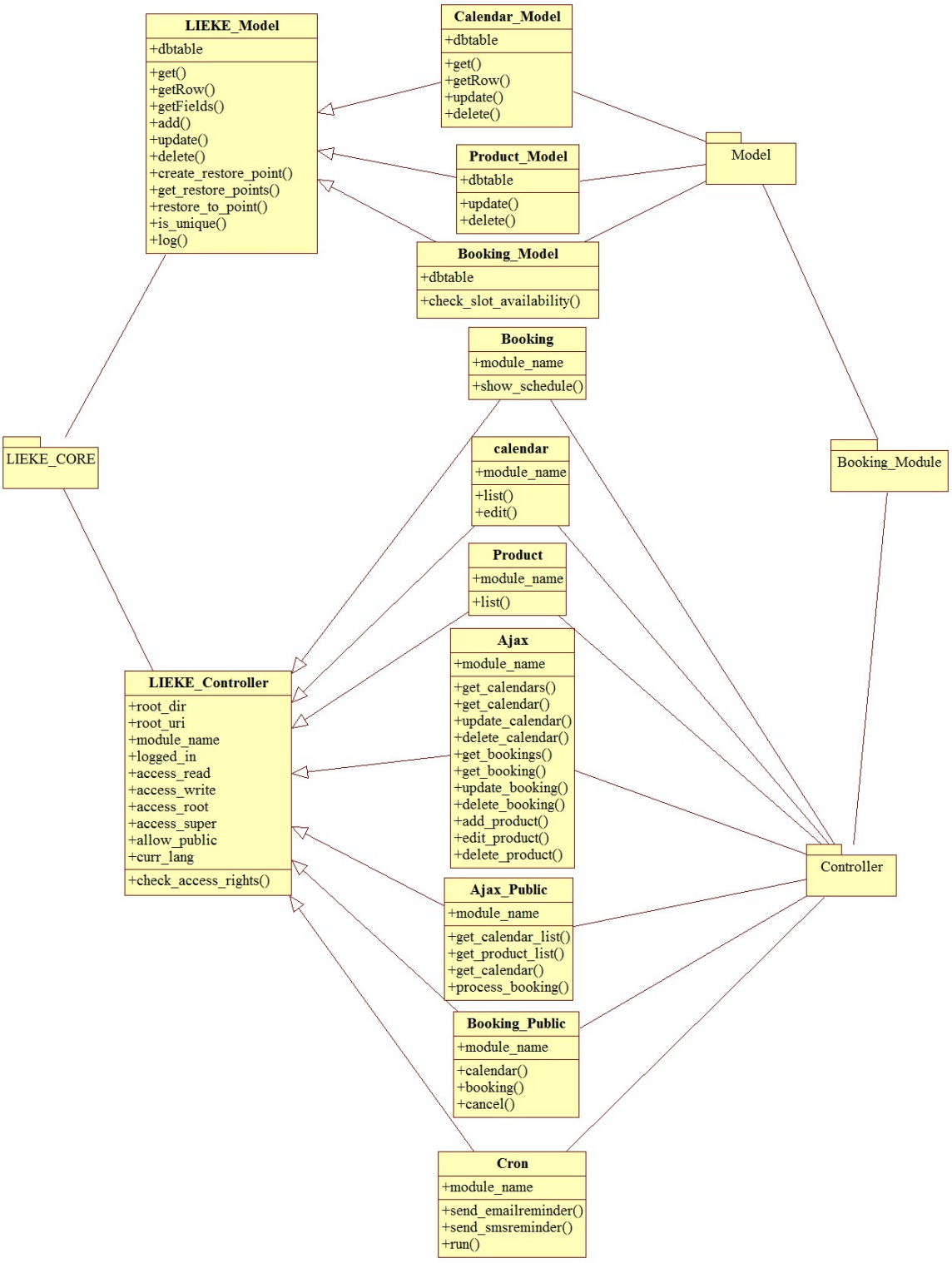
Table 11 presents the SMSReminder use case, which is one of the use cases to be included in AskReminder use case. It will send a mobile text message to remind the customer about the booking for the certain time, which the user asked for when he/she was creating the booking. LIEKECustomer is not necessary to require the reminder in this project. The system will automatically send the reminder. Because that the LIEKECustomers are the main users for this booking service. They are the users, who order this software for their business. They will stay in their office and waiting for the customers coming by the ordering time. The booking service can be improved later if the EndCustomer requires the reminder for them also.

3.3 Class Diagram

The class diagram is a static diagram. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application. It describes the attributes and operations of a class and also the constraints imposed on the system. They are used in the modelling of object-oriented systems. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram. (Tutorialspoints 2014)

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams, which can be directly mapped with object-oriented languages and widely used at the time of construction. The UML diagrams like activity diagram, and sequence diagram can only give the sequence flow of the application but class diagram is a slice different. So it is the most popular UML diagram in the coder community. (Tutorialspoints 2014)

The class diagram design of the booking service is an important design of this project. The class diagram is the basic structure of the implementation of the software. The class diagram shows the collection of the classes, which are required in the implementation of the software. The programmer can start to write the code for built the software according to the classes, which are described in the class diagram. This class diagram is the key design for the programmer to understand the purpose of developing this software, and to start implement the software step by step. The GRAPH 14 below is the class diagram design of booking service for LIEKE platform.



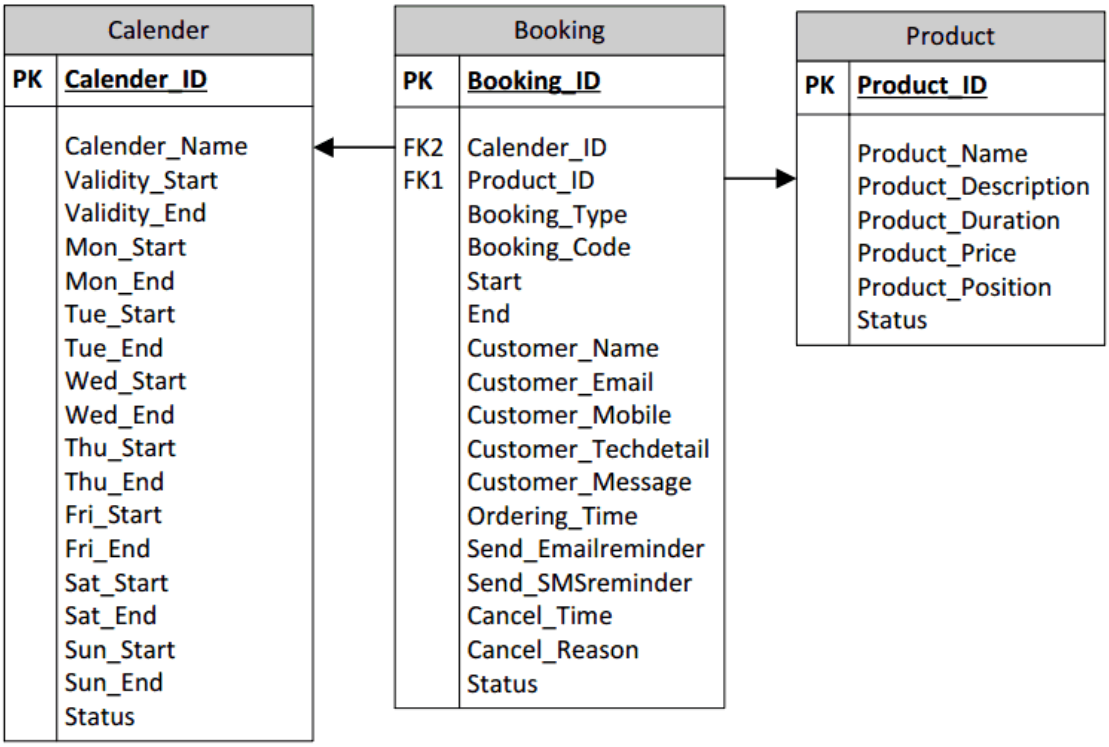
GRAPH 14. Class diagram design of booking service for LIEKE platform

As can be seen in the GRAPH 14, LIEKE_Model class and LIEKE_Controller class are associated with LIEKE_CORE package. For the model classes, the booking server has LIEKE_Model class, which is the parent class of Calendar_Model class, Product_Model class, and Booking_Model class. Basic operations/functions will be run in LIEKE_Model class, and the operations/functions in other three classes is for override the function when needed. For example, both Calendar_Model and LIEKE_Model have the operation called delete. The reason why the same operation appears in Calendar again is that in the Calendar_Model class user has to override delete operation in order to suit the specific case. The specific case is that user needs to check if the system still has undone bookings in the calendar before deleted.

For the controller class, the functions are complicated and the structure of this class is complex. The LIEKE_Controller class is for checking the user's rights, to confirm that the user is the LIEKE customer or the end customer. After the rights have been confirmed, the software will show different view according to the different rights. For example, the end customer user only can access the right to control and choose the calendar and product from the lists, and then make the booking. He/she will not have the rights for creating, editing or deleting the calendar and product. Those are the controlling rights for different users. With Ajax classes in the backend, the system makes the frontend interface much smoother.

3.4 Database Design

A database is a set of data that has a regular structure, which is organized in such a way that a computer can easily find the desired information. Data is a collection of distinct pieces of information, particularly information that has been formatted in some specific way for use in analysis or making decisions. (The Linux Information Project 2006.) For this project, the software designer needs to design the database for the booking service in LIEKE. GRAPH 15 presents the contents that the database tables should contain and also the relationships between the tables.



GRAPH 15. The database design of booking service for LIEKE platform

There are three database tables shown in GRAPH 15, which are Calendar, Booking and Product. The primary key of product database table is product ID. This unique ID is automatically set in the table when the new product is created. Every product has its own name, description, duration (how long time this product is used to process), price and the position, which the user can change for a special sale. The status is for LIEKE customer to decide if the product will show to the end customer or not. It is the product's visibility to the end customer.

The ID of calendar is the primary key for the calendar table and it set in the table automatically when the new calendar is created. Every calendar has its own name, which is easy for the customer to choose and find. The validity date for the calendar is to create the limitation of the date for the calendar. Not each calendar is available for all the time. Part of the calendars maybe just work for certain time, such as a season or a month. When the LIEKE customer creates the calendar, they will be required to insert the working time for

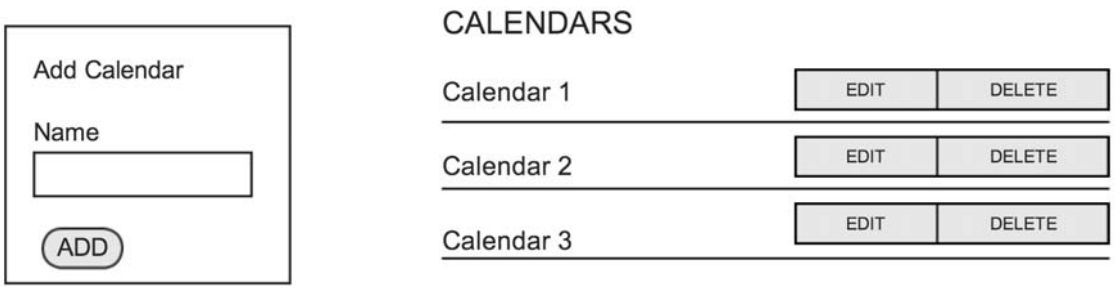
each day in the calendar-validated dates. It will help end customer to choose the suitable time. The status is for LIEKE customer to decide if the calendar will be presented to the end customer or not.

The booking table is related to the product table and also the calendar table. When the user creates a new booking, the booking ID would also automatically be created in the database, which will be the primary key in this table. With the link between the calendar ID and product ID, the user can choose the calendar they need and the product they want to book. When the user chooses the calendar and the product with selected starting time, the system will automatically fill in the end time according to the duration of the product. Also the user will fill in all the information in the table when creates the booking, such as name, email, mobile number, and leave message. The user also can choose the way he/she wants to be reminded of the booking, by email or by SMS. After the booking has been made, the user will receive a booking code, which can be use to cancel the booking with an explanation in the future if needs, and for LIEKE customer to edit the booking. The user can change the status to “coming”, “processing”, or “done”. Then the system will save the technical details, the ordering time and cancel time to the database for later confirmation.

3.5 Layout Design

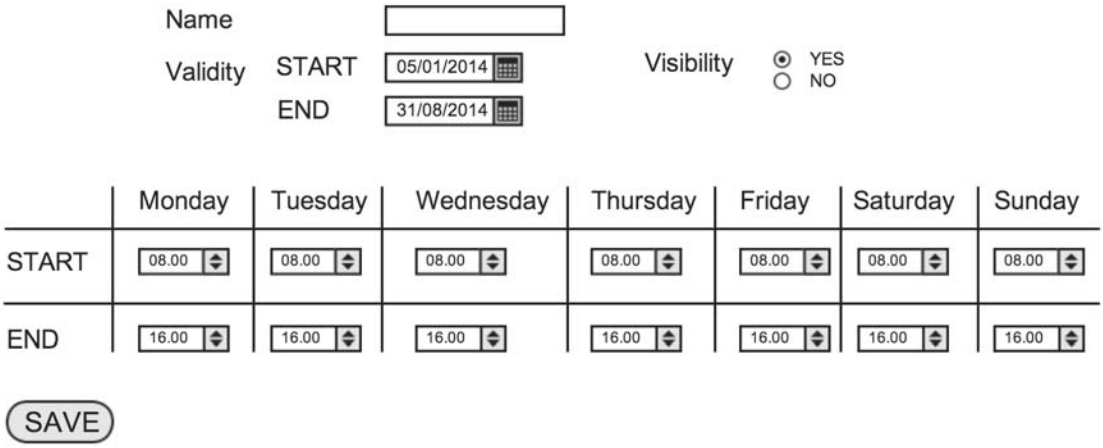
Layout design in this project is the Mockup designs for the user interface of booking service for LIEKE platform. The user interface refers to the information the program presents to the user, and control sequences the user employs to control the program. In LIEKE layouts are always designed per project basis, and layout design for this project is the views of customers when they are opening the booking service. Because different types of customers have different rights, the layout/ the view of the customer from the booking service is different. After log in LIEKE and with more rights, the user can manage more contents than the customer without log in. The graphs below are the examples of the layout design for booking service with the different user’s rights.

For the LIEKE customer, the users usually first need to log in the LIEKE platform (CMS), then the system will understand users are LIEKE customer with LIEKE customer access rights, and it will show users the views with their rights. GRAPH 16 presents the layout of creating the calendar in the booking service. Without LIEKE customer's rights, the users will not be able to see this layout.



GRAPH 16. Layout design for creating new calendar of the booking service

In order to create a new calendar, the LIEKE customer needs to type the calendar name, which customer wants to create into the textbox in the Add Calendar section. Then click ADD button. The new calendar is created. The calendar's list will display in CALENDARS section with the calendar's name, and the EDIT and the DELETE buttons. When the user wants to edit the calendar, he/she just needs to click the EDIT button. The layout design of edit calendar is shown in GRAPH 17 below.



GRAPH 17. Layout design for edit calendar of the booking service

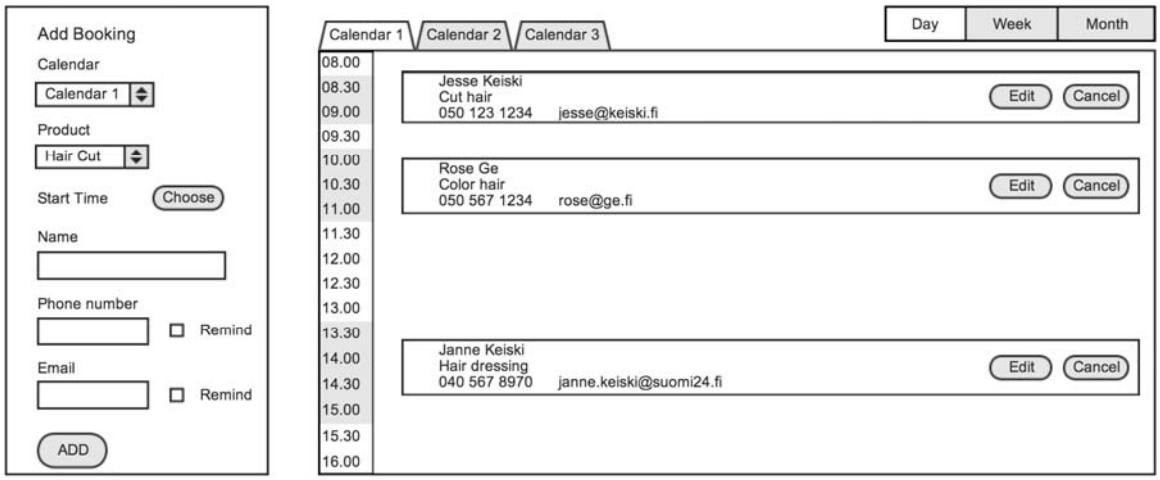
As can be seen in GRAPH 17, LIEKE customer is able to edit the calendar after it is created. The customer can change the calendar name, set the validity date for the calendar, and the working hours/time for each day of the calendar. The customer also can change the calendar's visibility by checking YES or NO. After all the details have been edited, the user can click the SAVE button. The calendar detail will be saved and the calendar is ready to be used. LIEKE customers can manage their products, too. See GRAPH 18 below.

Name	Price	Duration	Description	
Hair cut	30	30 min	cut hair	Delete
Hair color	70	60 min	color hair	Delete
Hair dressing	100	90 min	style your hair	Delete
Wedding hair	150	120 min	bride's hair package	Delete
Hair caring	200	45 min	5 times package	Delete

GRAPH 18. Layout design for managing the product of the booking service

From GRAPH 18, user can see that it is the same procedure as creating the calendar with creating a new product. The customer needs to type in the name of the product, the price of the product, set the duration of the product by using dropdown box, and insert the description of the product. Then click the ADD button; the new product has been created. The product's list will be shown as a table. In order to edit the product, the customer just needs to click on the contents in the table. For example, after the product, Hair Cut, has been created, the customer wants to change that name. He/she can edit the name by clicking on it, then write a new name, press ENTER, and the name has been changed to the new name. The same applies to price and description. For the duration, the customer only needs to choose a new time range in the dropdown box. In the end, the customer can delete the product by clicking the delete button.

The layout for the booking management is different between LIEKE customer and end customer. It is because they have different access rights. The following GRAPH 19 presents the layout of the booking management with LIEKE customer's right, which means LIEKE customer needs log in the system first. With LIEKE customer's right, the system will show more functions and options.



GRAPH 19. Layout design of booking management of the booking service for LIEKE customer

As can be seen in GRAPH 19, creating new booking has procedure almost the same as creatiing new calendar and new product. Choose the calendar and the product user wants to book from the lists, which are shown in the dropdown box. Then choose the start time for booking. When user is clicking the Choose button, a calendar with validity time marked will be displayed in a new window, and customer just need choose the time, which is available. The LIEKE customer can type in the end customer's information, such as name, phone number and email. Also the LIEKE customer can check the checkbox for send reminder to customer when needed. Then click the ADD button, and the new booking will be shown in the calendar as the right part of the graph shows. When LIEKE customer wants to edit or delete the booking, he/she just needs to click the Edit or Delete button, which are in the same column with the booking detail are displayed in the calendar. But the layout for the end customer to manage a booking is different see GRAPH 20, which is the layout design of booking management of booking service for end customer.

GRAPH 20. Layout design of the booking management of the booking service for the end customer

The end customer, as shown in GRAPH 20, can choose the different calendar by clicking on the calendar name. The calendar will be shown with week mode. Once the customer chooses the product from the product dropdown box, the working time of the week will be displayed with different colour in different period. Red colour means the time is not available. The orange ones is for the time period that is too short for the product duration time. And only the green colour areas are the available times to book. When the customer clicking the beginning time of the booking he/she wants in the calendar, according to the product's duration, the area of the booking will change from green to blue. After that, the customer only needs fill in the information, such as name, phone number, email and the message. He/she also can choose the type of the reminder that sends by email or SMS. The end customer needs to click CONFIRM THR BOOKING button to save and confirm the booking, which he/she just made.

4 CONCLUSION

The primary goal of this project was to develop a booking service, which was built on LIEKE platform. The main task of doing this thesis work was the software design for a booking system that can be used to offer and reserve timeslots and Newave will help with the implementation later. The designs have to be suitable with the LIEKE platform. The LIEKE platform and the technical requirements behind LIEKE, such as LAMP, CodeIgniter, HMVC, Smarty was explained in this thesis work. And the responsive design was a technology, which was used with the LIEKE platform to improve the customers' requirements.

In the design part, this project uses StarUML for the tool to design use case diagram and class diagram. The Microsoft Visio was used for designing the database tables and the relationship between tables. Those designs will be used for implementing this booking service in LIEKE platform under supervising by Newave. This project requires that some of the designs should be able to be understood by Newave customers and also the software developers in Newave. Those diagrams are designed with Unified Modelling Language (UML), which is the language that most of the software engineers are familiar with.

According to those requirements of the customer and the software designs, the software was created in the LIEKE platform. The software is fully functional as specified and it met the usability for the LIEKE customer. The software implementation took less time than originally planned. There are some missing columns in the database table, which should be considering before hand. On the other hand the software is modular. For example, just changing one file in the database can change the database access routines. The modularity makes the booking service easily maintainable, the quality of the code can be considered quite high. According to the feedback of the software tester, the software verification was a success. The basic functionality and the overall functionality of the software were verified in the browser-based applications. Because that the LIEKE platform has built by using Bootstrap, which secure the system working for mobile devices, too. Newave would be looking forward to create a great booking application for mobile devices soon.

In order to get feedback from the customer, the employee of MHK Isännöinti was interviewed. According to the interview the developed software has helped with the daily work and the software is very easy to use. The employees prefer using the software over pen and the calendar books, which was the method previously in use. It was said that using the software gives customers more professional and modern image from MHK. The customers of MHK can arrange the meeting online without calling and they can come the meeting on time with the reminder function in the software. It saves a lot time for both MHK and the customer of MHK.

In future, there were few areas for improvement. The software should be developed so that it would memorise the customer's information even the booking has been deleted. In case the customer booking the time again. It was also noticed that the changes to the information could be tracked. If there were a mistake, it would enable them to go back to an earlier state. The software should include the end customer log in feature. It would be logical that the end customer could edit booking, too. With the log into the system, the end customer would have more rights and could access more functions in the booking service. The software also could allow the LIEKE customer to change the time of their lunch and coffee breaks. That the LIEKE customer could manage their working time and break time the way they would like to have. More development ideas could be collected from both LIEKE customer and the end customer. Making a questionnaire for them would reveal which improvements and developments the users of the software would like to use.

5 REFERENCES

AllsizeWeb. 2013. Why responsive web design. Available: http://www.allsizeweb.com/responsive_web_design.html. Accessed 05 June 2014.

Bryant, K. 2014. Responsive design: A Crash Course and Demo. Magnetic Marketing Corp. Available: http://www.dwuser.com/education/content/responsive-design-a-crash-course-and-demo/. Accessed 10 April 2014.

Core team. 2012. Bootstrap. Available: http://getbootstrap.com. Accessed 12 June 2014.

Davis, M.E & Phillips, J.A. 2006. Learning PHP & MySQL (1st ed.). Sebastopol: O'Reilly Media, Inc.

De Freyssinet, S. 2010. Scaling web application with HMVC. Techportal. Available: http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/. Accessed 05 April 2014.

Ellislab. 2012. Application flow chart. Ellislab, Inc. Available: http://ellislab.com/codeigniter/user-guide/overview/appflow.html. Accessed 05 April 2014

Niemeyer, P. 2002. Learning Java (1st ed.). Sebastopol: O'Reilly Media, Inc.

Otto, M. 2012. Building Twiter Boostrap. Available: http://markdotto.com/2012/01/17/bootstrap-in-a-list-apart-342/. Accessed 05 June 2014.

Refsnes Data. 1999-2014. The MVC Programming Model. Available: http://www.w3schools.com/aspnet/mvc_intro.asp. Accessed 12 June 2014

Rob. 2013. What is Linux. XenForo Ltd. Available: http://www.linux.org/threads/what-is-linux.4076/. Accessed 24 April 2014.

Rouse, M. 2008. LAMP (Linux, Apache, MySQL, PHP). Available: http://searchenterpriselinux.techtarget.com/definition/LAMP. Accessed 24 April 2014.

The Apache Software Foundation. 1997-2014. Apache HTTP server project. Available: http://httpd.apache.org/ABOUT_APACHE.html. Accessed 11 June 2014.

The Linux Information Project. 2006. Database definition. Available: http://www.linfo.org/database.html. Accessed 24 April 2014.

Tutorialspoint. 2014. a/Basic MVC Architecture. Available: http://www.tutorialspoint.com /struts_2/basic_mvc_architecture.htm. Accessed 09 June 2014.

Tutorialspoint. 2014. b/UML class diagram. Available: http://www.tutorialspoint.com/uml/uml_class_diagram.htm. Accessed 24 April 2014.

Williams, L & Heckman, S. 2008. Requirements. OpenSeminar. Scoperto. Available: http://openseminar.org/se/modules/8/index/screen.do. Accessed 24 April 2014.