

Yang Cui

A Mass Email Sending System for Mainostoimisto Faarao Oy

A Mass Email Sending System for Mainostoimisto Faarao Oy

Yang Cui

Bachelor's thesis

Autumn 2010

Degree Programme in Business Information
Technology

Oulu University of Applied Sciences

School: Oulu University of Applied Science

Programme: Degree Programme in Business Information Technology

Author: Cui Yang

Year: 2010

The title of thesis: A Mass Email Sending System for Mainostoimisto Faarao Oy

Pages + appendix: 41 pages, 9 appendixes

ABSTRACT

This thesis is a project based thesis. In this thesis, the whole system which was called Ramses Mailer 2 will be introduced, including the main idea of the system, the purpose and the uses of the system, and the development of the system.

In the introduction part of the system, the reason of why the company needs the system will be explained, and also the functions and the main target markets of the system will be introduced. Moreover, the development part will contain the explanation of how to make this system by the analysis of the programming codes.

However, due to the large size of the system, it is hard to explain how to make the system step by step, but it will be divided into several main parts, and there will be some sample scripts in each part, and I will give the instruction about how to make those sample scripts with detailed codes analyze line by line.

Moreover, there will be a theoretical part, which will discuss about the techniques used in the system. For example PHP frameworks, MVC design model, version control etc.

The purpose of this thesis is to introduce PHP frameworks to the students who have already learned PHP for days, but wish to enter a higher level. The Objected-Oriented Programming Thinking will always go through the whole project, and the reader will think more than that

thinking after reading. Furthermore, the project development explanation part will be a good tutorial for the reader to understand the operation method of PHP Frameworks.

As a result of the thesis, Mainostoimisto Faarao Oy will have a good introduction of the technique part of the system, and this can be used to be a part of the instructions of the system which will be used when promote the system to the market.

Keywords: PHP, Object-Oriented Programming, PHP Frameworks, MVC

TABLE OF CONTENT

| | |
|--|----|
| A Mass Email Sending System for Mainostoimisto Faarao Oy | 1 |
| ABSTRACT | 2 |
| 1 INTRODUCTION | 5 |
| 1.1 Introduction of the system | 5 |
| 1.1.1 Email | 5 |
| 1.1.2 Mass email sending system | 6 |
| 1.2 System Analysis | 7 |
| 1.3 Developing Requirement Analysis | 8 |
| 2 THEORATICAL ANALYSIS | 9 |
| 2.1 PHP Frameworks | 9 |
| 2.1.1 Introduction | 9 |
| 2.1.2 Object-Oriented Programming Thinking | 10 |
| 2.1.3 MVC design model | 11 |
| 2.1.4 CodeIgniter | 13 |
| 2.2 JavaScript | 13 |
| 2.2.1 AJAX | 14 |
| 2.2.2 jQuery | 15 |
| 3 SYSTEM DEVELOPMENT | 17 |
| 3.1 Database design | 17 |
| 3.2 Initialization | 20 |
| 3.2 Login part | 23 |
| 3.3 Listing | 30 |
| 3.4 Create campaign tool | 34 |
| 3.5 Emails Delivering | 36 |
| 4 CONCLUSION | 38 |
| 5 DISCUSSION | 39 |
| REFERENCE | 40 |
| APPENDIX | 41 |

1 INTRODUCTION

Mainostoimisto Faarao Oy is a company which provides varieties of IT solutions of web applications. There are two main business models in the company, one model is to provide customized web applications to the customers, such as company websites, business software and so on. Another model is to provide a CMS (Content Management System) the name of which is Ramses CMS, the system can help customers to build their own personal or small company websites very easily and quickly, and the company will also help the customers to buy and host the web space and domain.

With the business growth up, the company wants to have more services under their own brand instead of making project for the other companies. Therefore, in the company's future plan, they will develop a big system which includes all kinds of software (which is a small or medium sized company needs), such as domain reserving tool, website hosting tool, website building tool and some office software. The mass email sending system which will be introduced in this thesis is one of the office software of that system.

1.1 Introduction of the system

The system was called Ramses Mailer 2. Ramses is the main product series of Mainostoimisto Faarao Oy, at the moment it has two sub-products, they are Ramses CMS (Content Management System) and Ramses Mailer.

Ramses Mailer is a mass email/campaign delivering system, and it was made in early 2009. However, the idea was immature at that time, so as the time goes by, Ramses Mailer can no longer fulfill all the customers' demands, the functions were limited and not intelligent enough. Therefore, the company decided to build Ramses Mailer 2 instead of Ramses Mailer.

1.1.1 Email

At the beginning, the concept of email needs to be understood. Email is widely used nowadays,

and it is a part of daily life of a normal human being.

Electronic mail is an asynchronous messaging technology. The person that you are trying to reach does not have to be available that instant. She will receive your message the next time she checks her email. This simple fact allows nearly every netizen to send a message to any other at any time. Electronic mail is used in business, government departments, universities, schools, homes, kiosks, and on mobile laptops, pagers, and personal digital assistants. While the Web distributes information, email keeps us in touch with one another. (David Wood 1999, 1.)

Email sending is a main function of the Ramses Mailer 2 which works totally the same as the normal sending process of electronic mail. But usually it's used to send hundreds of emails at one time, and each of the email is delivered individually, which means it sends the emails to the receivers one by one. This kind of system was called mass email sending system.

1.1.2 Mass email sending system

The main difference between the normal email system and the mass email sending system is that the mass email system focuses on sending emails to large quantities of receivers at once, but the email will be sent to each receiver individually. It means that every receiver will get an email which appears to be sent to him/her only, therefore he/she will not see a long receivers list in the information of the email.

This kind of mass email sending system is quite useful for commercial uses, for example the advertisements and promotions sending. If a company which sells shoes wanted to let all its customers know that there would be a discount event next week, they can write an email with the discount information, and then send the email to their customers, or even some potential customers whose emails are known by the company. If the normal email system were used here, it would be hard to send hundreds of emails by resending and resending again.

However, the receivers may not be happy with this kind of advertisement emails, and some email systems may define these emails as junk mails by defaulting. In this case, the way of sending the email and the content of the email will become very important.

Moreover, there are some other ways of advertising online, for example Really Simple Syndication (RSS), it can help people to get the newest updates of the source which they are following

1.2 System Analysis

Ramses Mailer 2 is based on Ramses Mailer's concept, but it is a totally different system. It will have much more powerful functions and friendlier interface. The purpose or the main function of the Ramses Mailer is to provide a better and easier way for business people to create campaigns for personal or company use, the campaigns can be used for notice, announcement, promotion or some other commercial uses. Therefore, in the Ramses Mailer 2, there are 3 main parts.

The first one is the campaigns handle part, which includes campaign creation tool and campaigns list. By the campaign creation tool, user can create a campaign which is based on existing templates. And then the user can modify the content of the campaign by many easy-to-use tools provided by the system.

In the campaigns list, users can view all the campaigns divided into 3 different lists, which are already sent campaigns, scheduled campaigns and saved campaigns. Users can sort the campaigns by different order or search for the wanted campaign, and also he/she can click on the campaign to see the details or resend, edit and delete the campaign.

The second part will be the contacts handle part, where users can create new contacts, or modify the existing contacts. The system will also provide a mass adding tool, which helps users to add thousands of contacts via .csv file at once. Users just need to follow the instruction and examples, and then they can easily create a .csv file which contains all his existing contacts, and then they

can import all those contacts to the Ramses Mailer 2. Moreover, in the contact list, users can view all the contacts in a list, which provides sorting and searching tools and users can also edit or delete the contact by click on different function buttons.

The last part would be group. A group is made up of contacts. When the user tries to send a campaign to other users in the same group – for instance, friends – he/she could now check the group 'Friends', instead of clicking on every single contact within. Of course, all contacts in group Friends would have been appointed and could be altered bt the user himself/herself at any given time.

1.3 Developing Requirement Analysis

The company's main direction of the production is the web application, which means the application can be accessed by the internet only. Normally, PHP, ASP.NET and JavaEE are considered to be the best programming languages for web applications.

Mainstoisimisto Faarao Oy is a medium-small sized company, so actually the target of the company market is not the huge web application developing market., but some medium or small sized projects.

In this case, JavaEE usually uses in big projects, and it is too complicated to small projects. ASP.NET is provided and supported by Microsoft, and it only works under Windows Operation System, which means the operation costs will be much higher than PHP.

For companies, PHP means cheaper operation costs. Because it works under Linux Operation System, which is mostly free. In addition, the appearance of LAMP also makes it easy to choose and build a free but efficient environment and platform. LAMP stands for Linux, Apache, MySQL and PHP. Linux is the operation system of the server, apache works for file system and MySQL holds the database. Consequently, LAMP was chosen to be the developing environment of Ramses Mailer 2.

2 THEORITICAL ANALYSIS

Before starting the project, the techniques which will be used during the developing period need to be understood absolutely. To understand the exactly advantages and disadvantages of a technique will help the developer to avoid potential risks, and always find the most suitable solutions. In this chapter, some main concepts will be discussed.

2.1 PHP Frameworks

As it is mentioned before, there are many frameworks of PHP, by the last statistics of phpframeworks.com, they are Akelos, ash.MVC, CakePHP, CodeIgniter, DIY, eZ Components, Fusebox, PHP on TRAX, PHPDevShell, PhpOpenbiz, Prado, QPHP, Seagull, Symfony, WACT, WASP, Yii, Zendframework, and ZooP.

They're different generally, but they're similar partly. To understand why frameworks are used in nearly every PHP projects, then it's important to know what does PHP framework exactly means and the advantages of it first.

2.1.1 Introduction

From the original meaning of the word "framework", it can be understood as a kind of structure. And for PHP, it can mean a "way of coding". Frameworks are not another technique, but it has changed the style of PHP definitely. It can be explained shortly as a code structure of making a PHP project. Normally, a PHP project will contains some frontend files and backend files, those frontend files would be the interface of the project, and the backend files do the logical operation.

For example, nearly every website has the feedback form. In that feedback page, the form and submit button which can be seen by the users was made by HTML and CSS codes, this is frontend. Consequently, in some cases, frontend files can be understood as the files which can be seen, and on the contrary, backend file means the files which cannot be seen.

Continue with the example, after user filled the form, and click on the submit button, the backend file will be activated; it gets the post from that frontend file, and insert those information filled by the user to the database (Figure 1).

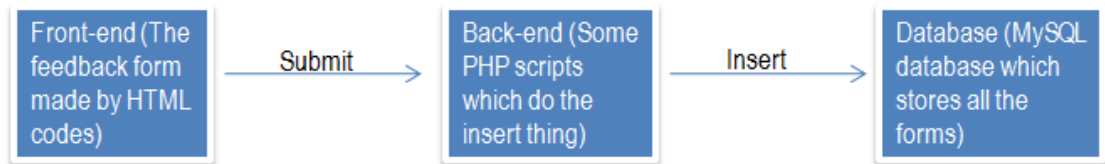


Figure 1 - Process of a simple feedback form submits.

However, if the project is huge sized, there are hundreds of this kind of insert, update or delete actions to the database which are controlled by the user from the frontend, this kind of structure will become very complicated, and the later development or maintain will be very hard. Therefore, a solution for huge sized PHP projects was being invented, which is called MVC design model.

2.1.2 Object-Oriented Programming Thinking

However, before starting working with MVC design model, the concept of Object-oriented programming must be understood. Because the MVC design model was based on the Object-oriented programming thinking.

According to Janet Valade (2004, 177), Object-oriented programming (OOP) is an approach to programming that uses objects and classes, and it's widespread today, and many universities teach object-oriented programming in beginning programming classes.

Though Object-oriented programming is a kind of hard programming style for beginners, which has more complicated and strict rules, but in fact, Object-oriented programming uses the same way as the human being thinks. In fact, OOP is invented by the brief of "code less, do more", and in Model – View – Controller design model, the OOP was fully used in the models.

2.1.3 MVC design model

... The acronym MVC stands for Model, view and controller. These three distinct components of the pattern represent the data model, the rendering of that data model (the view), and the logic that accepts input and contains the logic to manipulate the model and view (the controller). The relationship between these three components is shown in Figure 2. (John Cogeshall & Morgan Tocker. 2009, 4.)

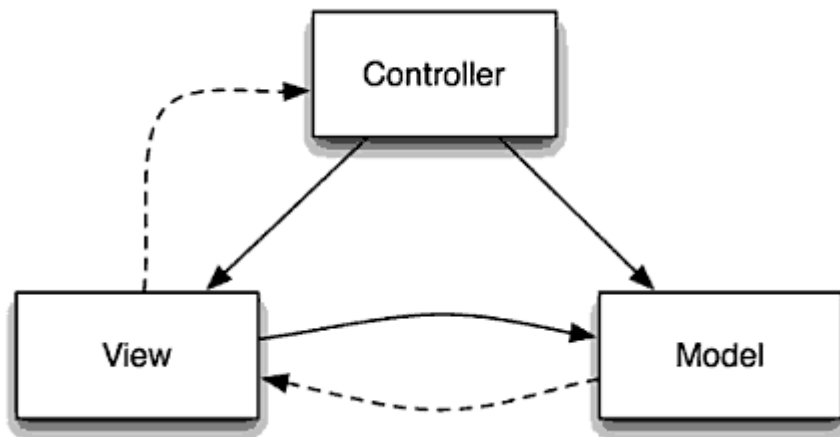


Figure 2 – The high-level relationship between a model, view, and controller within an MVC framework. (John Cogeshall & Morgan Tocker. 2009, 4.)

The example of feedback submitting which has been discussed before can be also designed with Model – View- Controller model. In this case, the frontend page which shows the feedback form will be showed by the controller. It means in the controller there's a function which does the operation of print out the form page. After the user clicked on the submit button, the post data will go to the controller again, but this time, the controller will call another model function to finish the rest actions. (Figure 3).

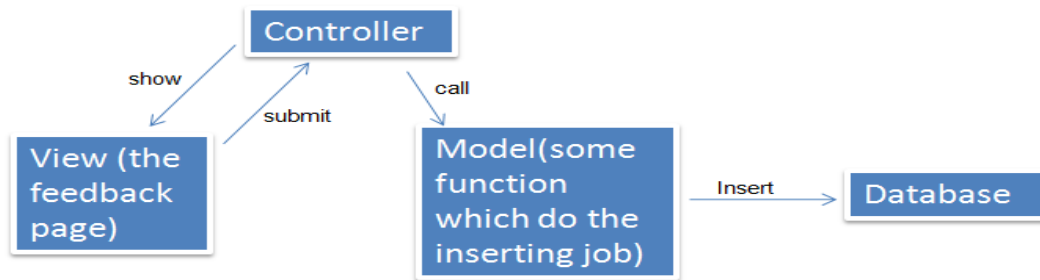


Figure 3 – Process of a simple feedback-form submits designed by Model – View - Controller structure.

It seems that it's more complicated by using Model – View - Controller structure, but if this is a huge project, there won't be only this feedback form, but also many other forms, and many actions such as login, registration, create and delete, modify and edit etc. Those actions which require high traffic between databases, in the other words, there're many different frontend and backend files which are different.

In this situation, certainly the normal designer model still works, but the complex file structure will make the debug and upgrade very hard and many codes were iterative written.

On the contrary, in Model – View - Controller structure, different frontend pages can be handled by the same controller, including those functions which do the related actions. For example, a website has several parts, one part is news, to handle everything of this part, a controller named news can be created, and the web pages can show the news list, and the pages show the news content, and the functions do the inserting and deleting thing when create or delete news – can be all handled by that controller, which means all those functions can be write inside that controller, and for some other part, like products, then all the functions related products can be written in the controller products.

The biggest advantage is when the system needs to be upgraded, or giving some bug fix, and it's very clearly to the programmer that which part should be modified. Moreover, it's also much easier for the graphic designer to make the user-interface to the frontend pages, because all the

frontend pages are called by the controller, so all the logical operations are done inside the function. In the view page, all the variables just need to be printed out, and no more logical codes, this is what we called separation of interface and codes. As a result, the graphic designer who doesn't know much about programming, the web page is much more clearly, and that makes it much easier to design the page.

Another big advantage of Model – View – Controller is for developer, whatever in controllers or models, the codes is written by Object-Oriented Programming, which makes the codes more readable and editable.

In conclusion, a framework is a structure with Model – View – Controller, and programmed with Object-Oriented Programming Thinking, as the result of analysis of the mass email system, a framework must be used. In Mainostoimisto Faarao Oy, a PHP framework named CodeIgniter was used for every project.

2.1.4 CodeIgniter

CodeIgniter is a powerful PHP framework with a very small footprint, built for PHP coders who need a simple and elegant toolkit to create full-featured web applications. (Ellislab, Referred: 27.8.2010)

CodeIgniter is loosely based on Model – View – Controller design model, and the speed is the best feature of it compare with the other frameworks. And because the Ramses Mailer 2 will be a sub system of Ramses CMS, and Ramses CMS is build on CodeIgniter, so CodeIgniter will be the framework for Ramses Mailer2.

2.2 JavaScript

With the coming of the WEB 2.0 time, JavaScript has become more and more important in web development. It helps to improve the users' experience of a website, and it's a very good binder

between HTML codes and PHP codes.

JavaScript is most commonly used in web browsers, and, in that context, the general purpose core is extended with objects that allow scripts to interact with the user, control the web browser, and alter the document content that appears within the web browser window,. The embedded version of JavaScript runs scripts embedded within HTML web pages. It is commonly called client-side JavaScript emphasize that scripts are run by the client computer rather than the web server. (David Flanagan 2006, 1.)

Moreover, the most useful technology which can highly improve the user experience is AJAX. It can keep the user always stay in current page, but do all the requests and actions in the backend side.

2.2.1 AJAX

Ajax (Asynchronous JavaScript and XML) encompasses much more than the technologies that make up the catchy acronym. The general term Ajax describes the usage of various Web technologies to transform the sluggish batch submission of traditional Web application into a highly responsive near desktop-software-like user experience. (Tomas A. Powell. 2008, 3.)

Normally it can be understood that AJAX do everything behind the users' sight. The main purpose of using AJAX technology in a web application is to give better user experience to the client, because AJAX can handle most of the users' requests in the backend side.

However, whatever JavaScript or AJAX, they both have a very complicated coding "grammas", what is not easy to learn in a short time, and also low-efficient. Therefore, jQuery as a JavaScript library will be used instead of puring JavaScript and AJAX.

2.2.2 jQuery

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript. (jQuery. Referred: 4.9.2010)

Although jQuery is just a library of JavaScript, it has fully changed the way of coding JavaScript. A code example will explain this more clearly. For example there is an input field:

```
<input type="text" id="input_field" value="input_value" />
```

And if the value of the input field needs to be gathered, by JavaScript, it can be write like this:

```
var value = document.getElementById("input_field").value;
alert(value)
```

However it's much easier to do the same action by jQuery:

```
var value = $("#input_field").val();
alert(value);
```

From the example above it's clear that the coding style is different between JavaScript and jQuery,

but because jQuery is just a library of JavaScript, so it will not affect the JavaScript, the JavaScript codes can be written together with jQuery-styled codes, like the "alert();" in last example.

3 SYSTEM DEVELOPMENT

After analyzing the system, the developing can be started. Firstly, the normal processes of building a web application need to be discussed. Usually, a web application, which is made by PHP, have 3 main parts, they're frontend, backend, and database (Figure 4).

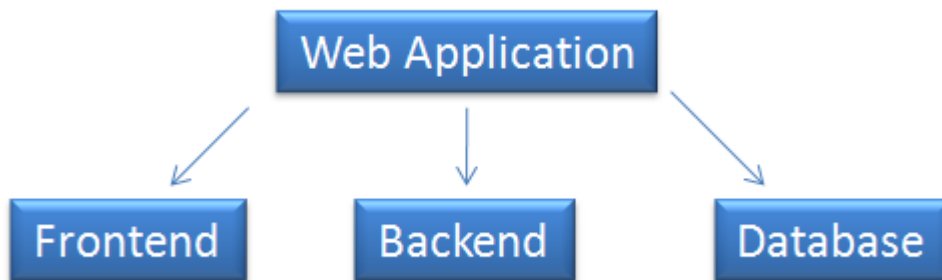


Figure 4 – Normal structure of a web application based on PHP.

In the frontend part, there're interface of the software, that's the only part which users can see and control. In the backend part, there is the logical layer of the system; all the functionalities are operated in this part. And in the database, there stores all the data which is needed by the system or insert by the users.

3.1 Database design

The first part which needs to be done is the database part, because the Ramses Mailer 2 is part of Ramses CMS System. The database needs to be based and related to the original database, so the company Mainostoimisto Faarao has already designed the database (Figure 5-7). Moreover all the mysql database creation codes were in APPENDIX 1. The Ramses Mailer 2 will access to at least 2 databases, one is the Ramses CMS's database, because they use the same tables for users' information. And each user of Ramses Mailer will have his/her own database, which means when the company sell the product to a new customer. They will also create a database for that customer.

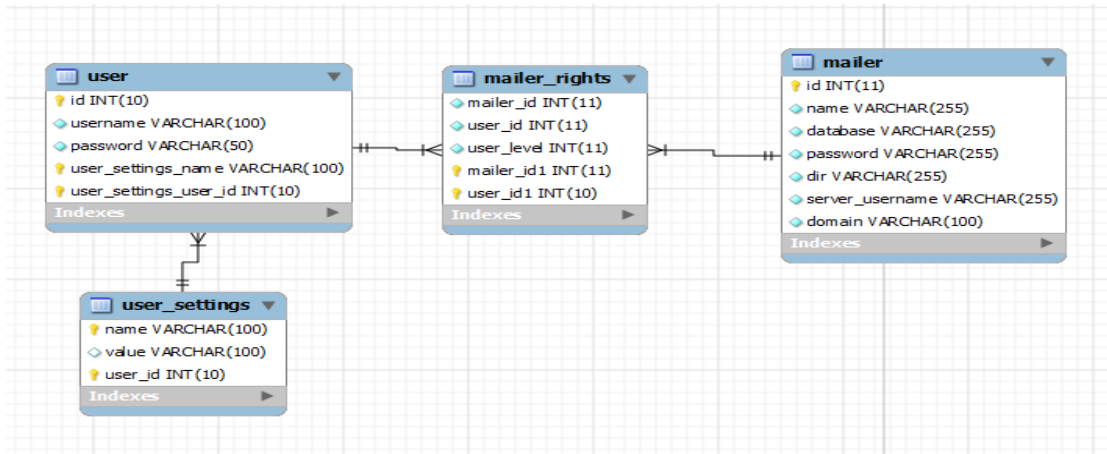


Figure 5 – Tables which Ramses Mailer will use in Ramses CMS's database

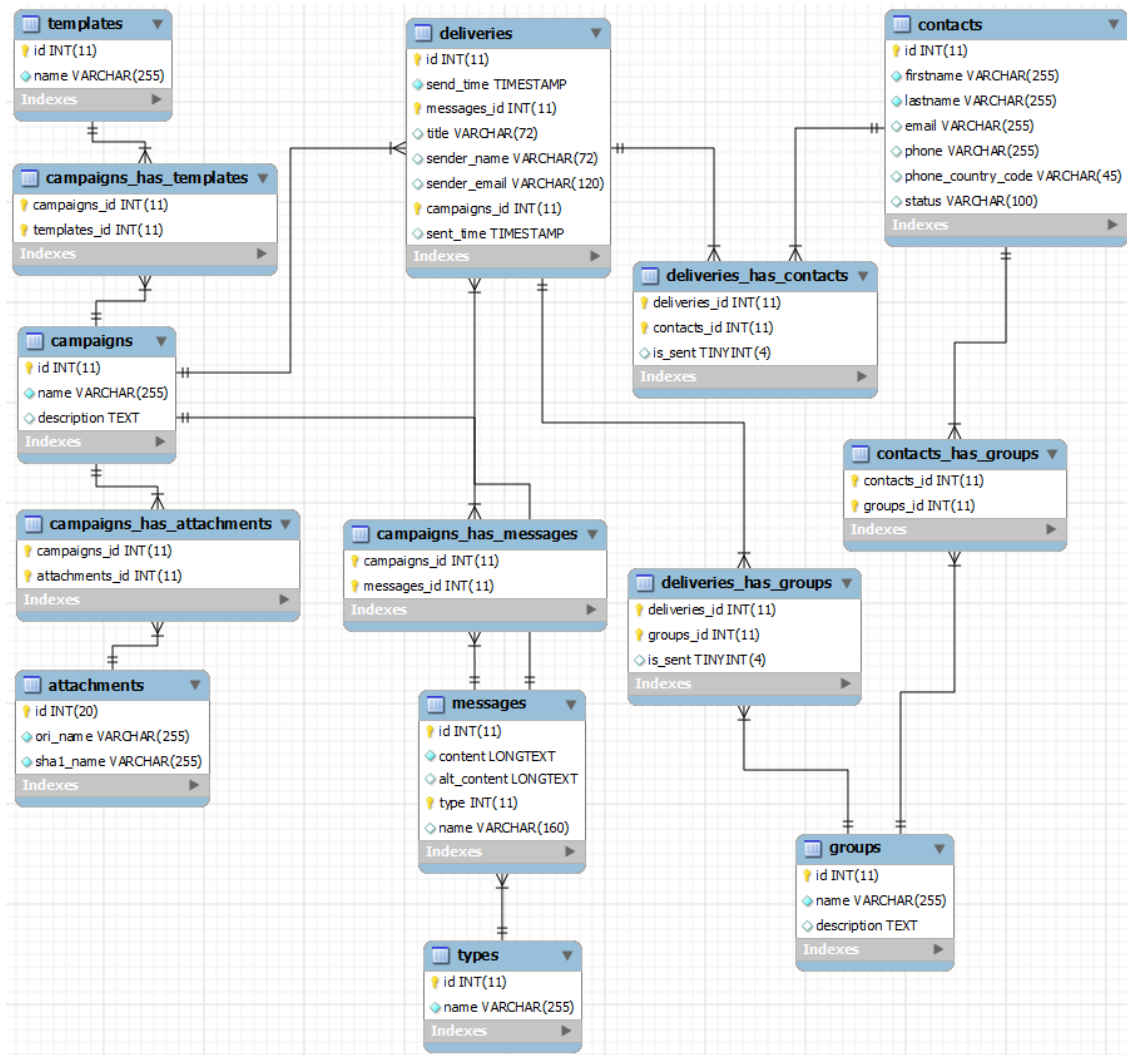


Figure 6 – Main part of Ramses Mailer 2's database tables

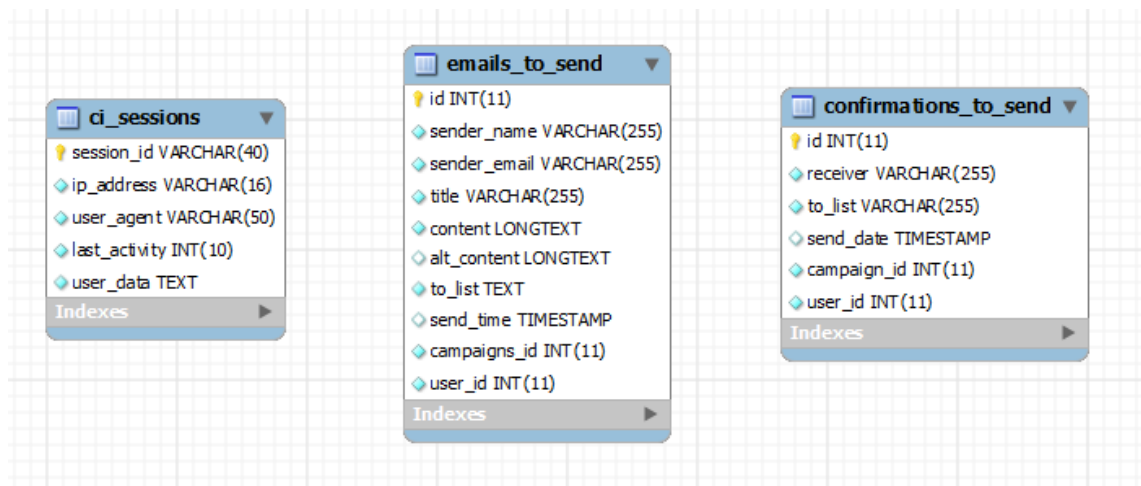


Figure 7 – Rest part of Ramses Mailer 2's database tables

The figure 10 shows the tables in Ramses CMS's database which will be used in Ramses Mailer 2, the table named *user* stores the user id, username and the password of the user, and in the table named *user_settings*, there are the personal settings or information such as first name, last name, address of the user. In the table named *mailer*. There stores the information of the Ramses Mailer 2, like the Ramses Mailer 2's database name, and the username & password which can access to that database. The table named *mailer_rights* connects the user and mailer, records which user has the access right to the mailer.

In the figure 11, there's the database structure of the Ramses Mailer 2, as it is mentioned before, each user of the Ramses Mailer 2 will has his/her own database, which means each user will has a database like this.

The main table is the *deliveries*, it records the campaign id, sending time, and the receivers. Via the campaign id, the campaigns can be found in the table named *campaigns*. Each campaign has a template which was recorded in the table named *templates*, and the campaigns might have some attachments, that will be recorded in the table named *attachments*. Also, the content of the campaign was in the table named *messages*.

The receivers can be either a group or a contact, or both. And a group or contact information are recorded in the table named *groups* and *contacts*. Moreover, the members of the groups are the

contacts, so there's a table named *contacts_has_groups* which handles the relationship between contacts and groups.

In the figure 12, there are several individual tables. The first one *ci_sessions* is a specific table required by CodeIgniter framework, because CodeIgniter uses a different session store method from the normal PHP sessions. And the last two tables records the emails or confirmation emails which will be sent. Because sending emails to a large number of receivers may take a lot of time, every time after the user clicked on “deliver” button, those emails will not be sent immediately. But insert into these two tables. There's a script which checks the emails which have arrived the sending time, and then send them, this script will run every minutes just in the server with the help of a software named crontab which installed in the Linux-based server.

For the system, as analyzed in the theoretical part, it will have several main parts: campaigns, groups and contacts. And each of them has some sub-parts (Figure 8).



Figure 8 – Function structure of Ramse Mailer 2

3.2 Initialization

Before starting the coding, some settings need to be done to the CodeIgniter. Firstly, the files of CodeIgniter need to be downloaded. After unzipped the downloaded file, several files and folders can be found (Figure 9).





| | | | |
|---|-----------------|--------|------|
|  system | 2010/9/11 22:31 | 文件夹 | |
|  user_guide | 2010/9/11 22:31 | 文件夹 | |
|  index.php | 2009/7/3 0:23 | PHP 文件 | 4 KB |
|  license.txt | 2010/7/12 16:21 | 文本文档 | 3 KB |

Figure 9 – Files and folders of CodeIgniter

The file named *index.php* and the folder named *system* were needed to be uploaded to the server. And remember to create a folder named *Mailer2* in the server before uploading, and then put everything into the folder. After that, download a plug-in of CodeIgniter which name is *CI template*, this plug-in will help the framework to generate the interface with specified structure, and it will be much easier for the UI designer to modify the UI.

After downloaded, unzip the file, and copy all the folders to the server, under system -> application folder. After that, open the system -> application -> config -> config.php. Change the *base_url* to the link which the system located. This setting defines the base link address of the index page of the system.

```
$config['base_url']="http://yang.faaao.fi/Mailer2/";
```

Then, open the file named *autoload.php* which was also in the folder *config*. And load the libraries and helpers which will be used in the system.

```
$autoload['libraries'] = array(
    'database',
    'session',
    'template',
    'email'
```

```
);

$autoload['helper'] = array(
    'url',
    'html',
    'form',
    'file',
    'text',
    'email'
);
```

Libraries and helpers in CodeIgniter are some groups of functions which can make the coding more convenient. For example the library named *database* contains many functions which do the database actions, and after loaded this library, the actions related to database can be coded with very short and convenient way.

The next step is modify the database settings. Open the file *database.php* and set the database connection informations. The username and password should be the username and password of the mysql user, and the database will be the *ramsestcms*, which is the database name of Ramses CMS.

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "some_username";
$db['default']['password'] = "some_password";
$db['default']['database'] = "ramsestcms";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
```

```
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
```

In addition, in the system, every page will have the same header and footer, so the header and footer files can be made first. Both of the header and footer are already designed by the company.

Create a file named *header.php* in *views* folder, and codes of *header.php* were in APPENDIX 2. And some codes need to be explained here. The first one is the `<?= some codes ?>`. In normal PHP codes, If there is something need to be shown in the screen, `<?php echo "something"; ?>` will be written, but it's a little bit inconvenient, so `<?= "something" ?>` can instead of that. The second one is the *base_url()*, this will output the link address of the system which were set in *config.php*. Moreover, the *img()* in the codes is a function in CodeIgniter, is can instead of the HTML `` tag, for example `img("assets/pic/example.jpg")` equals to ``.

Then create a file named *footer.php* in *views* folder, and the codes were in APPENDIX 3. This part makes the 3 main navigator link of the system, they are campaigns, contacts and groups. And in this part, the *anchor()* need to be explained, it can instead of HTML `<a>` tag, for example `anchor("somepage", "go")` equals to `<a href="<?= base_url ?>index.php/somepage">go`.

3.2 Login part

After designed the database and defined the settings, the system development can be started. The first part will be the login part. As mentioned before, the CodeIgniter was based on MVC design model, so in CodeIgniter's application folder, there're directories which were designed for MVC (Figure 10).

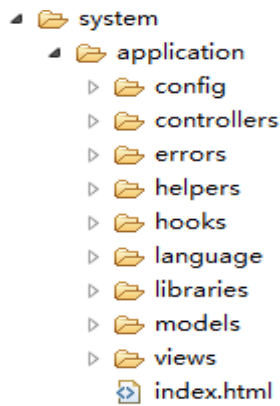


Figure 10 – Directories in application folder.

Some of the directories will be used for this project. The first one is the *config* folder which stores the configuration files, like the files which were modified in initialization part. The *helpers* and *libraries* directories store the helper or library files of the developers self. And the *models*, *views* and *controllers* directories were used for MVC.

Furthermore, for the purpose of *do more and code less*, before creating the first controller, a super controller which “controls” the entire controllers needs to be created first. So create a file named *MY_Controller.php* in the folder *libraries* and write the codes below. The class *MY_Controller extends Controller* means this library class which named *MY_Controller* will be the child the class of *Controller*, the class *Controller* were coded by CodeIgniter, and doesn’t need to be modified. The function *__construct()* defines the construct function of the class, construct function can be understood as the “root” function. The *parent::Controller();* means all the functions in the class *Controller* will be inheritance here, and can be used directly in this class.

```
class MY_Controller extends Controller{  
    function __construct() {  
        parent::Controller();  
    }  
}
```

The benefit of creating such a controller is all the JavaScript and CSS files can be loaded here. And there is no need to be loaded again and again in different controllers. For this project, all the CSS files has been already made by the UI designer, and as mentioned in the theoretical part, jQuery will be used in this project, and jQuery file need to be loaded before everything, so firstly some files can be loaded at the moment.

Firstly, the jQuery file needs to be downloaded from the official website of jQuery. And create a folder named *assets* in the main folder which named *Mailer2*, and then create subfolders *js*, *css* and *iconsets* under the new folder. Upload the jQuery file to the folder *js*, and the *css* file to the folder *css*, the folder *iconsets* stores the icons which will be used later, those icons were provided by the company.

When those steps above are finished, the codes which work for loading files can be written inside the `__construct` function of the *MY_Controller.php*. The `add_css()` or `add_js()` were functions provided by the CI Template plug-in which were installed before, and they do the jobs of load the *css* or *js* files at the beginning.

```
$this->template->add_css("assets/css/core.css");  
$this->template->add_js("assets/js/jquery-1.4.2.js");
```

Next, the controller for login part can be created. Create a file named *login.php* in the directory *controllers*. And write the codes below:

```
class Login extends MY_Controller {  
    function index() {  
        $this->template->write_view(  

```

```

        'header', 'header'
    );
    $this->template->write_view(
        'content', 'login/login_view'
    );
    $this->template->write_view(
        'footer', 'footer'
    );
    $this->template->render();
}
}

```

As the same in *MY_Controller.php*, the *extends* means the class before the word *extends* will be the child class of the parent class which follow the word. In this case, *Login* will be the child class of *MY_Controller*, so the files loaded in *MY_Controller* will be also loaded here.

The *index()* function is the root function of *Login*. In CodeIgniter, a function in the controller, means a “page”. For example, the base link address of the project is <http://yang.faarao.fi/Mailer2>, and there is a function named *test()* in the controller named *login*, the access address to that function will be <http://yang.faarao.fi/Mailer2/index.php/login/test>, but *index()* function will be the controller itself, to access the *index()* function, the link address will be just <http://yang.faarao.fi/Mailer2/index.php/login>.

In addition, here in the *index()* function, the *\$this->template-> write_view()* was a function provided by the CI Template plug-in, it was used to load the page file and set the file as a part of the whole page. The first parameter in *write_view()* will be the position of the file, and the second one will be the path to the file. In this situation, there are three parts of a page, they are header, content and footer, the header and footer will be the files which were made early before, and the content file will be the file which shows the login form and will be created later,

`$this->template->render()` will render and show the page.

The next step is to create the *view* file. Create a folder named *login* in *views* folder, and create a file named *login_view.php* in that folder. This file will contain the login form. The codes were in APPENDIX 4.

In the codes, *form_open()* can instead of the HTML `<form>` tag, also *form_input()*, *form_password()* and *form_submit()* can instead of the HTML `<input />` tag, but they are in different type, for example *form_password()* equals to `<input type="password">`. These are all provided by *form* helper of CodeIgniter. And this form will be submitted to the link "login/login_check", which means the *login_check* function in controller *login*. So a function name *login_check* need to be created in *login.php*.

```
function login_check(){
    $loginCheck=$this->login_model->checkUser($_POST['user
    name'], $_POST['password']);
    if( $loginCheck == "ok" ){
        redirect('campaign');
    }else{
        redirect('login');
    }
}
```

Normally, there's no rules for code balance of controllers and models, which means controllers can contains most of the codes, but also models can take more. But for readable reasons, the "thin controllers, fat models" rule is suggested. That is to say, controllers just do the jobs like "control", but most of the action codes will be written in the functions inside the models, and controllers just call those functions when needed.

Therefore, in this *login_check()* function, the part of checking whether the user has typed the right username and password will be done by the function named *checkUser* which is located in the model named *login_model*, and after checked, if both the username and password were correct, the function *checkUser* will return “ok”, otherwise it will return false.

However, the Ramses Mailer 2 has to access to 2 different databases, the Ramses CMS's database which named *ramsestcms* were set as the default database, but another database which belong to the user haven't been set yet. So one model which can switch the database needs to be created before create the *login_model*. So create a model named *dbswitch_model.php* in the folder models. And write the codes below:

```
class Dbswitch_model extends Model{
    protected $_altdb;
    function __construct(){
        parent::Model();
        if(!$this->session->userdata('db_username')
        || !$this->session->userdata('db_password')
        || !$this->session->userdata('db_database')) {
        } else {
            $dbconfig['hostname'] = "localhost";
            $dbconfig['username'] =
                $this->session->userdata('db_username');
            $dbconfig['password'] =
                $this->session->userdata('db_password');
            $dbconfig['database'] =
                $this->session->userdata('db_database');
            $dbconfig['dbdriver'] = "mysql";
            $dbconfig['dbprefix'] = "";
        }
    }
}
```

```

        $dbconfig['pconnect'] = TRUE;
        $dbconfig['db_debug'] = TRUE;
        $dbconfig['cache_on'] = FALSE;
        $dbconfig['cachedir'] = "";
        $dbconfig['char_set'] = "utf8";
        $dbconfig['dbcollat'] = "utf8_general_ci";
        $this->_altdb = $this->load->database($dbconfig,
        TRUE);
    }
}
}

```

Like in the controllers, this model also will be the child model of the CodeIgniter's Model. And `$this->session->userdata()` is equals to PHP code `$_SESSION['']`, it's a CodeIgniter's session, and it was stored in the table named `ci_sessions` in the database. So here after detecting username, password and the database name of the user (which means the user has logged in), the database will be switched to the user's own database.

After this model has been done, the `login_model` can be created. Firstly create a file named `login_model.php` in the models folder, and then write the codes in APPENDIX 5.

Firstly, for the security reason, all the password in the database are encrypted by a PHP function named `sha1()`. And because the `login_model` has extended to `dbswitch_model`, so here a line `$oridb = $this->load->database('default', true);` need to be wrote to load the database connection settings which were set in the database's configuration file. Followed that, there's the database selection actions with the CodeIgniter style, these codes equals to the mysql commands below:

```
SELECT
```

```

        user.id AS uid,
        user.username AS username,
        mailer.id AS mid,
        mailer.name AS db_username,
        mailer.password AS db_password,
        mailer.database AS db_databse,
        mailer.dir AS dir,
        mailer.domain AS domain

FROM user

JOIN mailer_rights

ON user.id = mailer_rights.user_id

JOIN mailer

ON mailer_rights.mailer_id = mailer.id

WHERE username = $username

AND password = $password

```

But it's much more clearly to use CodeIgniter's style to do the database actions, and it is easy to modify in the future. These codes gather the user's information from the database by the username and password which was typed by the client. If it didn't find matched user, a false response will be returned. But if it is found, all the information will be store in sessions by the function `$this->session->set_userdata()`, and return a string "ok".

Back to the `login_check()` function in the `login` controller, after getting the return value from the `checkUser()` function, the system will be redirected to different page, if the return is false, means login failed, then the system will jump back to the login page, but if the value is "ok", then the client will be redirected to the campaign page.

3.3 Listing

In the page after the user logged in, there will be a list of all the campaigns which have been

delivered already. In this case, a table which shows the information of each delivered campaigns will be needed, and it should also support the user to search or sort the data. Therefore, a plug-in of jQuery named jqgrid was chose to handle this issue.

Firstly, download the jqgrid files and upload the files to the server, and write the codes below to load the files in the system.

```
$this->template->add_js(
    "assets/js /i18n/grid.locale-en.js");
$this->template->add_js(
    "assets/js/ jquery.jqGrid.min.js");
$this->template->add_css("assets/css/ui.jqgrid.css");
$this->template->add_css(
    "assets/css/ jquery-ui-1.8.2.custom.css");
```

Then the controller and model need to be created for campaign part, and they are similar as the controller and model in login part, but the name will be *campaign.php* and *campaign_model.php*, in controller *campaign*, write the function which shows the page of delivered campaign list.

```
function index(){
    $this->template->write_view('header', 'header');
    $this->template->write_view(
        'content', 'campaign/campaign_list');
    $this->template->write_view('footer', 'footer');
    $this->template->render();
}
```


Then create a folder named *campaign* in the directory *views*, and create a file named *campaign_list.php*, and fill the codes below to make the table and pagination bar:

```
<table id="campaign_list" width="100%"></table>
<div id="campaign_pager"></div>
```

Next step is to create a file named *campaign.js* in the directory *Mailer2* -> *assets* -> *js*, and write the codes in APPENDIX 6 to load the data of the table, and also load the file in *MY_Controller*.

This is a kind of AJAX technology, the link

http://yang.faarao.fi/Mailer2/index.php/campaign/get_campaign_data is the function which loads all the data required from the database.

In the codes, *colNames* contains the *<td>* content of the table's *<th>*. And *colModel* contains the settings of those *<td>*, moreover, *rowNum* defines how many rows in one page, and *rowList* defines the selection options of rows in a page, *sortname* and *sortorder* defines the default sorting rules, in this function, the data will be sort based on *sent_time* in descent order. Next, create the function *get_campaign_data*, write the codes below to the controller *campaign*.

```
function get_campaign_data(){
    $params = array(
        'page' => $this->input->post('page'),
        'limit' => $this->input->post('rows'),
        'sidx' => $this->input->post('sidx'),
        'sord' => $this->input->post('sord'),
        'searchOn' => $this->input->post('_search')
    );
}
```

```

        $data = $this->campaign_model->getCampaignList(
            $params, $_POST);
        echo $data;
    }

```

In the codes, `$this->input->post()` equals to the PHP function `$_POST[""]`, and here get all the post values from the user's actions, like the page number of the table, and the rows number in one page, and the sorting item & order, and if the user has searched something, that will be also get here.

Then create the function `getCampaignList()` in `campaign_model`. And write the codes in APPENDIX 7 there. These codes firstly store the post values to variables, and if the default sorting item didn't defined, it will be automatically defined to the first one. And if the user has searched something, get the search item and search key words, and stores in an associative array named `$condition`. For example if the user searched "test" for campaign's name, then the array will be `$condition['campaigns.name'] = "test"`.

Then it will calculate the number of rows and the number of pages. It will first select information of the campaign from the tables `deliveries`, `campaigns` and `messages`, and by the codes `$this->_altdb->where("UNIX_TIMESTAMP(deliveries.sent_time) <", time())`, it compare the sent time of the campaign with the current time in Unix timestamp, and if the sent time is earlier than the time now, it means the campaign has already been delivered. At last, it will push the required data into an array, and encode the array to JSON style. After this, the list of delivered campaign has been finished.

Moreover, all the other lists, such as the scheduled campaign list, saved campaign list, contact list and group list can be made with the same way.

3.4 Create campaign tool

Every campaign must be based on a template, and those templates were hard-coded by the company. So after creating a new campaign by selected an existing template, the campaign will already be a integrated email except all the contents haven't been edited. In this situation, a campaign editing tool was needed.

Firstly, the HTML codes of the template need to be analyzed, the codes can be found in APPENDIX 8. In the codes, the `<a>` tags inside the `<td>` tags with the class name "editHandle" were the tool links which used to edit the campaign. For example if the `copy_row` tool link needs to work, the codes below need to be added to the file `campaign.js`.

```
$(".copy_row").live("click", function() {  
    var content = $(this).parent().parent().html();  
    $(this).parent().parent().after(  
        '<tr class="record">' + content + '</tr>');  
    return false;  
});
```

These codes detects the "click" event of the mouse on the `<a>` tag which has the class name "copy_row" (the copy row button), firstly it get the content of the row which will be copied, `$(this)` means the button element itself, and `parent()` means the parent element of the button, and from the template code it was cleared that the parent element of the button's parent will be the `<tr class="record"></tr>`, and `html()` get the all html codes inside the element, and then add one more `<tr class="record"></tr>` with the same content after the origin `<tr>` element. And all the other actions can be done with the similar way, for example change the content text. Use the code below to create a simple editor:

```
<textarea id="editor"></textarea>
<input type="button" id="finish" value="submit" />
```

When the client clicked on the *edit text* button, the codes below will get the original content, and show the content into the editor for the clients to modify.

```
$("#edit_button").live("click", function(){
    var content = $(".txt-component").html();
    $("#editor").html(content);
    return false;
});
```

And after the client finished the modification of the content, then he/she can click on the *submit* button, and the codes below will change the content to the one which was modified by the client.

```
$("#finish").live("click", function(){
    var content = $("#editor").val();
    $(".txt-component").val(content);
    return false;
});
```

This method can be used for all the rest of the tools, the normal process is get the content or image, or even style settings of the *<tr>* where the button being clicked was located in, and show that in a place where the client can modify. After the client finished the editing, get the newest

content or image or style settings and instead of the old ones.

3.5 Emails Delivering

As introduced before, all the emails will not be delivered immediately, but stores in a specific table.

Then, a script will run every minute to check which emails can be delivering now, and deliver it.

Here are the codes of that script which named *cron.php* (controller):

```
public function send_email(){
    $oridb = $this->load->database('default', true);
    $oridb->select("mailer.database AS db_database");
    $oridb->from("mailer");
    $query=$oridb->get();
    $array = array();
    foreach($query->result() as $row){
        array_push($array, $row->db_database);
    };
    $array = array_unique($array);
    foreach($array as $a){
        $this->cron_model->send_campaign($a);
    }
}
```

This function will be ran by the *crontab* every minute, it get all the Ramses Mailer 2's databases, and set the databases' names as a parameter of the function *send_campaign*. And the codes of function *send_campaign* were in APPENDIX 9.

In the codes, first it get all the emails which stores in the table "emails_to_send", and check first if the email's send time is earlier than the time now, if it is, then do the delivery action.

`$this->email->from()` set the senders information, the first parameter defines the sender's email address, and the second parameter defines the sender's name, `$this->email->to()` set the receiver's email address, `$this->email->subject()` and `$this->email->message()` set the title and the content of the email, and the last one `$this->email->send()` will send the email at once.

4 CONCLUSION

Currently, emails play a more and more important role in advertising, and with the support of HTML, the emails look like more and more similar with the web pages. However, to make a HTML based email advertisement is not that easy, the designers need to have enough skill level with HTML and CSS coding. Obviously, normal business person doesn't have such skills. And for companies, they need to hire more people who make the emails only, that's an unnecessary human power cost.

Consequently, a software which helps people to create beautiful and professional HTML based emails without webpage design skills were needed, and the market will be widely. On the contrary, this Ramses Mailer 2 system is a product which has real market demands, and it's a good solution for especially single business person and small companies. It has the advantages such as easy to use, simple structured and much cheaper than hire a web designer.

In addition, in the PHP development nowadays, frameworks were more important than ever before. The MVC structure helps to separate the PHP codes from the HTML codes, which has highly improved the developing speed of a project, because the PHP programmers and the web designers can work individually, and merge their work very quickly after everything was done.

Furthermore, whatever which framework, they all defines the code structure in a specific and strict rule. So if a company uses one framework for all its products, those products will have a same coding style and that will be helpful when upgrading a product. Because no programmers will focus on one product forever, in this case, if one product were developed by a programmer two years ago, but it need to be upgraded recently by another programmer, even the reading of the original codes will take very long time, but if that product was made by the framework which the new programmer was familiar with, then he doesn't need to read too much and can understand the whole project very quickly.

5 DISCUSSION

I've been started to learn PHP from about 4 years ago, and it has attracted me a lot even at the very beginning. This project is the first project I made not by normal PHP coding but PHP frameworks, at the beginning it's not a good feeling of writing every code in object-oriented programming style, it was complicated. But with the project going on, I found that frameworks improve the quality of the PHP codes.

Because PHP is a script language, it was not like pure Object-Oriented languages such as C++ or Java. Sometimes the codes of PHP were mixed with HTML, and hard to read and modify, many unreasonable bugs occur always, and there's no good ways to track the bug except check the codes line by line. However, when coding with frameworks, those problems won't happen, and there are also many useful functions bind with CodeIgniter, and those functions decreased the time of coding a lot.

The developing period of the project is quite long. It takes more than 6 months, because the project is quite huge and everything except the user interface needs to be done by myself. Therefore, it is hard to explain every step of making the system in the thesis. And the PHP frameworks are not that easy to use for the PHP beginners. It requires medium level and experience with PHP, so in this thesis only the main parts were introduced, and the explanation of making project with MVC structure was set as the first purpose. And it's hoped that after reading the thesis, the readers can understand how to make simple projects or scripts with CodeIgniter, and get the basic concept of jQuery.

During the process, many problems happened. The biggest one is that coding with frameworks was quite different from coding normal PHP scripts, so many bugs were caused by the old-styled PHP codes. Moreover, to fit the MVC, the codes were quite complicated sometimes, but after familiar with the framework after several months, the problems were solved.

REFERENCE

Printed sources

David Wood. 1999. Programming Internet Email. Cambridge: O'Reilly Media, Inc.

David Flanagan. 2006. JavaScript: the definitive guide. Cambridge: O'Reilly Media, Inc.

Jon Loeliger. 2009. Version control with Git. Cambridge: O'Reilly Media, Inc.

Janet Valade. 2004. PHP 5 for dummies. New York City: For Dummies.

John Coggeshall & Morgan Tocker. 2009. New York City: Springer.

Thomas A. Powell. 2008. New York City: McGraw-Hill Professional.

Electronic sources

Ellislab. CodeIgniter [Referred: 27.8.2010]

Available: <http://codeigniter.com/>

jQuery. [Referred: 5.9.2010]

Available: <http://jquery.com/>

APPENDIX

TABLE OF CONTENT

| Appendix Number | Description |
|-----------------|--|
| APPENDIX 1 | The database creation codes of Ramses Mailer 2 |
| APPENDIX 2 | The codes of header.php |
| APPENDIX 3 | The codes of footer.php |
| APPENDIX 4 | The codes of login_view.php |
| APPENDIX 5 | The codes of Login_model.php |
| APPENDIX 6 | The codes of campaign.js |
| APPENDIX 7 | The codes of function getCampaignList |
| APPENDIX 8 | The codes of a normal template |
| APPENDIX 9 | The codes of function send_campaign |

The database creation codes for Ramses Mailer 2:

```
CREATE TABLE IF NOT EXISTS `attachments` (  
  `id` int(20) NOT NULL auto_increment,  
  `ori_name` varchar(255) NOT NULL,  
  `shal_name` varchar(255) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;  
  
CREATE TABLE IF NOT EXISTS `campaigns` (  
  `id` int(11) NOT NULL auto_increment,  
  `name` varchar(255) NOT NULL,  
  `description` text,  
  PRIMARY KEY (`id`),  
  KEY `name_INDEX` (`name`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;  
  
CREATE TABLE IF NOT EXISTS `campaigns_has_attachments` (  
  `campaigns_id` int(11) NOT NULL,  
  `attachments_id` int(11) NOT NULL,  
  PRIMARY KEY (`campaigns_id`,`attachments_id`),  
  KEY `fk_campaigns_has_attachments_campaigns1` (`campaigns_id`),  
  KEY `fk_campaigns_has_attachments_attachments1` (`attachments_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
CREATE TABLE IF NOT EXISTS `campaigns_has_messages` (  
  `campaigns_id` int(11) NOT NULL,  
  `messages_id` int(11) NOT NULL,  
  PRIMARY KEY (`campaigns_id`,`messages_id`),
```

```

KEY `fk_campaigns_has_messages_campaigns1` (`campaigns_id`),
KEY `fk_campaigns_has_messages_messages1` (`messages_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

```

CREATE TABLE IF NOT EXISTS `campaigns_has_templates` (
  `campaigns_id` int(11) NOT NULL,
  `templates_id` int(11) NOT NULL,
  PRIMARY KEY (`campaigns_id`,`templates_id`),
  KEY `fk_campaigns_has_templates_campaigns1` (`campaigns_id`),
  KEY `fk_campaigns_has_templates_templates1` (`templates_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

```

CREATE TABLE IF NOT EXISTS `ci_sessions` (
  `session_id` varchar(40) NOT NULL default '0',
  `ip_address` varchar(16) NOT NULL default '0',
  `user_agent` varchar(50) NOT NULL,
  `last_activity` int(10) unsigned NOT NULL default '0',
  `user_data` text NOT NULL,
  PRIMARY KEY (`session_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

```

```

CREATE TABLE IF NOT EXISTS `confirmations_to_send` (
  `id` int(11) NOT NULL auto_increment,
  `receiver` varchar(255) NOT NULL,
  `to_list` varchar(255) NOT NULL,
  `send_date` timestamp NULL default NULL,
  `campaign_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

```

```
CREATE TABLE IF NOT EXISTS `contacts` (  
  `id` int(11) NOT NULL auto_increment,  
  `firstname` varchar(255) NOT NULL,  
  `lastname` varchar(255) NOT NULL,  
  `email` varchar(255) default NULL,  
  `phone` varchar(255) default NULL,  
  `phone_country_code` varchar(45) default NULL,  
  `status` varchar(100) default NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;  
  
CREATE TABLE IF NOT EXISTS `contacts_has_groups` (  
  `contacts_id` int(11) NOT NULL,  
  `groups_id` int(11) NOT NULL,  
  PRIMARY KEY (`contacts_id`,`groups_id`),  
  KEY `fk_contacts_has_groups_contacts` (`contacts_id`),  
  KEY `fk_contacts_has_groups_groups1` (`groups_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
CREATE TABLE IF NOT EXISTS `deliveries` (  
  `id` int(11) NOT NULL auto_increment,  
  `send_time` timestamp NOT NULL default CURRENT_TIMESTAMP on update  
CURRENT_TIMESTAMP,  
  `messages_id` int(11) NOT NULL,  
  `title` varchar(72) default NULL,  
  `sender_name` varchar(72) default NULL,  
  `sender_email` varchar(120) default NULL,  
  `campaigns_id` int(11) NOT NULL,  
  `sent_time` timestamp NULL default NULL,
```

```

PRIMARY KEY (`id`,`messages_id`,`campaigns_id`),
KEY `fk_deliveries_messages1` (`messages_id`),
KEY `fk_deliveries_campaigns1` (`campaigns_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

CREATE TABLE IF NOT EXISTS `deliveries_has_contacts` (
  `deliveries_id` int(11) NOT NULL,
  `contacts_id` int(11) NOT NULL,
  `is_sent` tinyint(4) default NULL,
  PRIMARY KEY (`deliveries_id`,`contacts_id`),
  KEY `fk_deliveries_has_contacts_deliveries1` (`deliveries_id`),
  KEY `fk_deliveries_has_contacts_contacts1` (`contacts_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `deliveries_has_groups` (
  `deliveries_id` int(11) NOT NULL,
  `groups_id` int(11) NOT NULL,
  `is_sent` tinyint(4) default NULL,
  PRIMARY KEY (`deliveries_id`,`groups_id`),
  KEY `fk_deliveries_has_groups_deliveries1` (`deliveries_id`),
  KEY `fk_deliveries_has_groups_groups1` (`groups_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `deliveries_has_receivers` (
  `deliveries_id` int(11) NOT NULL,
  `receivers_id` int(11) NOT NULL,
  `is_sent` tinyint(4) default NULL,
  PRIMARY KEY (`deliveries_id`,`receivers_id`),
  KEY `fk_deliveries_has_receivers_deliveries1` (`deliveries_id`)
  KEY `fk_deliveries_has_receivers_receivers1` (`receivers_id`)

```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `emails_to_send` (
  `id` int(11) NOT NULL auto_increment,
  `sender_name` varchar(255) NOT NULL,
  `sender_email` varchar(255) NOT NULL,
  `title` varchar(255) NOT NULL,
  `content` longtext NOT NULL,
  `alt_content` longtext,
  `to_list` text NOT NULL,
  `send_time` timestamp NULL default NULL,
  `campaigns_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=36 ;

CREATE TABLE IF NOT EXISTS `groups` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(255) NOT NULL,
  `description` text,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

CREATE TABLE IF NOT EXISTS `images` (
  `id` int(11) NOT NULL auto_increment,
  `original_name` varchar(255) NOT NULL,
  `new_name` varchar(255) NOT NULL,
  `width` int(11) NOT NULL,
  `height` int(11) NOT NULL,
  PRIMARY KEY (`id`)
```

```
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

```
CREATE TABLE IF NOT EXISTS `messages` (
```

```
  `id` int(11) NOT NULL auto_increment,
```

```
  `content` longtext NOT NULL,
```

```
  `alt_content` longtext,
```

```
  `type` int(11) NOT NULL,
```

```
  `name` varchar(160) default NULL,
```

```
  PRIMARY KEY (`id`,`type`),
```

```
  KEY `fk_messages_types1` (`type`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

```
CREATE TABLE IF NOT EXISTS `receivers` (
```

```
  `id` int(11) NOT NULL auto_increment,
```

```
  `name` varchar(255) default NULL,
```

```
  `email` varchar(255) NOT NULL,
```

```
  `phone` int(20) NOT NULL,
```

```
  `phone_country_code` int(11) NOT NULL,
```

```
  PRIMARY KEY (`id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

```
CREATE TABLE IF NOT EXISTS `settings` (
```

```
  `name` varchar(255) NOT NULL,
```

```
  `value` varchar(255) NOT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `sms_to_send` (
```

```
  `id` int(11) NOT NULL auto_increment,
```

```
  `content` text NOT NULL,
```

```
  `confirm_phone` bigint(20) NOT NULL,
```



```
`phone_list` text NOT NULL,  
`send_time` timestamp NULL default NULL,  
`campaigns_id` int(11) NOT NULL,  
PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;  
  
CREATE TABLE IF NOT EXISTS `templates` (  
  `id` int(11) NOT NULL auto_increment,  
  `name` varchar(255) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;  
  
CREATE TABLE IF NOT EXISTS `templates_has_variables` (  
  `templates_id` int(11) NOT NULL,  
  `variables_id` int(11) NOT NULL,  
  PRIMARY KEY (`templates_id`,`variables_id`),  
  KEY `variables_id` (`variables_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
CREATE TABLE IF NOT EXISTS `types` (  
  `id` int(11) NOT NULL auto_increment,  
  `name` varchar(255) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;  
  
CREATE TABLE IF NOT EXISTS `variables` (  
  `id` int(11) NOT NULL auto_increment,  
  `name` varchar(255) NOT NULL,  
  `value` varchar(255) NOT NULL,  
  PRIMARY KEY (`id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

ALTER TABLE `campaigns_has_attachments`

  ADD CONSTRAINT `campaigns_has_attachments_ibfk_2` FOREIGN KEY
(`attachments_id`) REFERENCES `attachments` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `campaigns_has_attachments_ibfk_1` FOREIGN KEY
(`campaigns_id`) REFERENCES `campaigns` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE;

ALTER TABLE `campaigns_has_messages`

  ADD CONSTRAINT `campaigns_has_messages_ibfk_1` FOREIGN KEY
(`campaigns_id`) REFERENCES `campaigns` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE,

  ADD CONSTRAINT `campaigns_has_messages_ibfk_2` FOREIGN KEY
(`messages_id`) REFERENCES `messages` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE;

ALTER TABLE `campaigns_has_templates`

  ADD CONSTRAINT `campaigns_has_templates_ibfk_2` FOREIGN KEY
(`templates_id`) REFERENCES `templates` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE,

  ADD CONSTRAINT `campaigns_has_templates_ibfk_1` FOREIGN KEY
(`campaigns_id`) REFERENCES `campaigns` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE;

ALTER TABLE `contacts_has_groups`

  ADD CONSTRAINT `contacts_has_groups_ibfk_1` FOREIGN KEY
(`contacts_id`) REFERENCES `contacts` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE,
```

```
ADD CONSTRAINT `contacts_has_groups_ibfk_2` FOREIGN KEY
(`groups_id`) REFERENCES `groups` (`id`) ON DELETE CASCADE ON UPDATE
CASCADE;
```

```
ALTER TABLE `deliveries`
```

```
ADD CONSTRAINT `deliveries_ibfk_1` FOREIGN KEY (`messages_id`)
REFERENCES `messages` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
```

```
ADD CONSTRAINT `deliveries_ibfk_2` FOREIGN KEY (`campaigns_id`)
REFERENCES `campaigns` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

```
ALTER TABLE `deliveries_has_contacts`
```

```
ADD CONSTRAINT `deliveries_has_contacts_ibfk_1` FOREIGN KEY
(`deliveries_id`) REFERENCES `deliveries` (`id`) ON DELETE CASCADE
ON UPDATE CASCADE,
```

```
ADD CONSTRAINT `deliveries_has_contacts_ibfk_2` FOREIGN KEY
(`contacts_id`) REFERENCES `contacts` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE;
```

```
ALTER TABLE `deliveries_has_groups`
```

```
ADD CONSTRAINT `deliveries_has_groups_ibfk_1` FOREIGN KEY
(`deliveries_id`) REFERENCES `deliveries` (`id`) ON DELETE CASCADE
ON UPDATE CASCADE,
```

```
ADD CONSTRAINT `deliveries_has_groups_ibfk_2` FOREIGN KEY
(`groups_id`) REFERENCES `groups` (`id`) ON DELETE CASCADE ON UPDATE
CASCADE;
```

```
ALTER TABLE `deliveries_has_receivers`
```

```
ADD CONSTRAINT `deliveries_has_receivers_ibfk_1` FOREIGN KEY
(`deliveries_id`) REFERENCES `deliveries` (`id`) ON DELETE CASCADE
ON UPDATE CASCADE,
```

```
ADD CONSTRAINT `deliveries_has_receivers_ibfk_2` FOREIGN KEY
(`receivers_id`) REFERENCES `receivers` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE;
```

```
ALTER TABLE `messages`
```

```
ADD CONSTRAINT `messages_ibfk_1` FOREIGN KEY (`type`) REFERENCES
`types` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

```
ALTER TABLE `templates_has_variables`
```

```
ADD CONSTRAINT `templates_has_variables_ibfk_2` FOREIGN KEY
(`variables_id`) REFERENCES `variables` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE,
```

```
ADD CONSTRAINT `templates_has_variables_ibfk_1` FOREIGN KEY
(`templates_id`) REFERENCES `templates` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE
```

The codes of header.php :

```

<div id="sub-navigation">
  <div class="container_12">
    <div class="grid_12">
      <ul>
        <li>
          <span>
            <? = img(array('src'=>base_url()
              assets/iconset/email_add.png',))."Create New
Campaign" ?>
          </span>
          <img src='<? =
base_url() ?>assets/pic/gfx/bluepipe.gif' alt='' />
        </li>
        <li>
          <span>
            <? = img("assets/iconset/email_go.png")."Sent
Campaigns" ?></span>
            <img src='<? =
base_url() ?>assets/pic/gfx/bluepipe.gif' alt='' >
          </li>
          <li>
            <span>
              <? =
                img("assets/iconset/time.png")."Schedu
led Campaigns" ?>
            </span>

```

```

        <img src='<?=
base_url()?>assets/pic/gfx/bluepipe.gif' alt='' />
    </li>
    <li>
        <span>
            <?= img("assets/iconset/disk.png")."Saved
Campaigns" ?>
        </span>
        <img src='<?=
base_url()?>assets/pic/gfx/bluepipe.gif' alt='' />
    </li>
</ul>
<ul>
    <li>
        <span>
            <?=img(array('src'=>base_url().'assets/iconset/user_add.png'))
            ."Create New Contact" ?>
        </span>
        <img src='<?=
base_url()?>assets/pic/gfx/bluepipe.gif' alt='' />
    </li>
</ul>
<ul>
    <li>
        <span>
            <?=
            img(array('src'=>base_url().'assets/ic
            onset/group_add.png'))."Create New
            Group" ?>
        </span>

```

```
        <img src='<?=
base_url()?>assets/pic/gfx/bluepipe.gif' alt='' />
</li>
    </ul>
</div>
</div>
</div>

<div class="clear"></div>

<div id="content">
    <div class="container_12">
        <div class="grid_12">
```

The codes of footer.php :

```

<div id="footer">
  <div class="container_12">
    <div class="grid_12">
      <span class="align_right">
        <a href="#">Tietoturvaseloste</a> |
        <a href="#">Rekisteriseloste</a>
      </span>
      <span class="align_left">
        &copy; 2010 <a href="http://www.ramses.fi/">Ramses</a>
      </span>
    </div>
    <div class="clear"></div>
    <div id="social" class="grid_12">
      <a href="http://facebook.com/faarao" target="_blank">
</a>
      <a href="http://twitter.com/faarao" target="_blank">
        </a>
      </div>
    </div>
  </div>

  <div id="navigation">
    <div class="container_12">
      <div class="grid_12">
        <ul>

```



```
<li>
  
  <span>
    <?= anchor("/campaign", "Campaign") ?>
  </span>
</li>
<li>
  
  <span>
    <?= anchor("/contact", "Contact") ?>
  </span>
</li>
<li>
  
  <span>
    <?= anchor("/group", "Group") ?>
  </span>
</li>
</ul>
</div>
</div>
</div>
<div class="clear"></div>
```

The codes of login_view.php :

```

<table width="100%" height="100%">
  <tr>
    <td align="center" valign="middle">
      <?= form_open("login/login_check") ?>
      <table width="40%">
        <tr>
          <td width="45%" align="right">Username</td>
          <td width="55%" align="left">
            <?= form_input('username') ?>
          </td>
        </tr>
        <tr>
          <td align="right">Password</td>
          <td align="left">
            <?= form_password('password') ?>
          </td>
        </tr>
        <tr>
          <td align="right"><?= form_hidden('action',
'login') ?></td>
          <td align="left">
            <?= form_submit('login',"login".nbs(1)
            .$this->uri->segment(3) ?>
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>

```

```
        <?=  
        form_close() ?>  
    </td>  
</tr>  
</table>  
</div>  
</div>  
</div>
```

The codes of Login_model.php :

```

class Login_model extends Dbswitch_model {
    function checkUser($username, $password){
        $password=sha1($password);
        $oridb = $this->load->database('default', true);
        $oridb->select("user.id AS uid");
        $oridb->select("user.username AS username");
        $oridb->select("mailer.id AS mid");
        $oridb->select("mailer.name AS db_username");
        $oridb->select("mailer.password AS db_password");
        $oridb->select("mailer.database AS db_database");
        $oridb->select("mailer.domain AS domain");
        $oridb->from("user");
        $oridb->join("mailer_rights", "user.id =
mailer_rights.user_id");
        $oridb->join("mailer", "mailer_rights.mailer_id =
mailer.id");
        $oridb->where("username", $username);
        $oridb->where("password", $password);
        $query=$oridb->get();
        $row=$query->last_row();
        if(empty($row)){
            return false;
        }else{
            $sessionArray=array(
                'user_id' => $row->uid,
                'mailer_id' => $row->mid,

```

```
'username' => $row->username,  
'db_username' => $row->db_username,  
'db_password' => $row->db_password,  
'db_database' => $row->db_database,  
'domain' => $row->domain  
);  
$this->session->set_userdata($sessionArray);  
return "ok";  
}  
}  
}
```

The codes of campaign.js :

```
$(document).ready(function(){
    jQuery("#campaign_list").jqGrid({
        url:'http://yang.faarao.fi/Mailer2/index.php/campaign/get_
campagin_data',
        datatype: 'json',
        mtype: 'post',
        colNames:[
            "Campaign ID",
            "Campaign Name",
            "Campaign Description",
            "Sent Time",
            "Action"
        ], colModel :[
            {name:'id', index:'id', align:'center', align:'center',
hidden:true},
            {name:'name', index:'name', align:'center', width:200},
            {name:'description', index:'description', align:'center',
width:300},
            {name:'sent_time', index:'sent_time', align:'center',
searchoptions:{
                dataInit:function(el){
                    $(el).datepicker({dateFormat:'yy-mm-dd'});
                }
            }, width:300},
            {name:'resend', index:'resend' , align:'center',
search:false, sortable:false}
```

```
    ],  
    pager: '#campaign_pager',  
    rowNum:50,  
    rowList:[50,100,150,1000],  
    sortname: 'sent_time',  
    sortorder: 'desc',  
    viewrecords: true,  
    caption: "Delivered Campaign List",  
    height: 400,  
    autowidth: true  
  }).navGrid('#campaign_pager',{  
    edit:false,add:false,del:false,search:false,refresh:true  
  })  
  .filterToolbar();  
});
```



```

    }}}
$this->_altdb->select("*");
    $this->_altdb->from("deliveries");
    $this->_altdb->join("campaigns", "deliveries.campaigns_id =
campaigns.id", "left");
    $this->_altdb->join("messages", "deliveries.messages_id =
messages.id");
    $this->_altdb->where("deliveries.sent_time !=", "0000-00-00
00:00:00");
    $this->_altdb->where("UNIX_TIMESTAMP(deliveries.sent_time)
<", time());
    $this->_altdb->like($condition);
    $query=$this->_altdb->get();
    $count = $query->num_rows();
    if($count>0 && $limit>0){
        $total_pages = ceil($count/$limit);
    }else{
        $total_pages = 0;
    }
    if($page>$total_pages){
        $page = $total_pages;
    }

    $start = $limit * $page - $limit;
    if($start<0){
        $start=0;
    }

    $table="";
    if( $sidx=="send_time" OR $sidx=="sent_time" ){

```

```

        $table="deliveries";
    }else{
        $table="campaigns";
    }
    $this->_altdb->select("campaigns.id AS id");
    $this->_altdb->select("campaigns.name AS name");
    $this->_altdb->select("campaigns.description AS description");
    $this->_altdb->select("deliveries.send_time AS send_time");
    $this->_altdb->select("deliveries.sent_time AS sent_time");
    $this->_altdb->select("messages.type AS type");
    $this->_altdb->from("deliveries");
    $this->_altdb->join("campaigns", "deliveries.campaigns_id =
campaigns.id", "left");
    $this->_altdb->join("messages", "deliveries.messages_id =
messages.id");
    $this->_altdb->where("deliveries.sent_time !=", "0000-00-00
00:00:00");
    $this->_altdb->where("UNIX_TIMESTAMP(deliveries.sent_time) <",
time());
    $this->_altdb->like($condition);
    $this->_altdb->order_by($table.".".$sidx, $sord);
    $this->_altdb->limit($limit, $start);
    $query2=$this->_altdb->get();
    $responce->page = $page;
    $responce->total = $total_pages;
    $responce->records = $count;
    $i=0;
    foreach($query2->result() as $row){
        $responce->rows[$i]['id']=$row->id;
        $responce->rows[$i]['cell']=array(

```

```
        $row->id,  
        anchor("campaign/view_campaign/".$row->id,  
$row->name),  
  
        $this->function_model->shortContent($row->description, 30),  
        $row->sent_time,  
        $this->lang->line("common_mtype_".$row->type),  
        anchor(  
            "campaign/resend_sent_campaign/".$row->id,  
  
img(array('src'=>'assets/iconset/arrow_redo.png')),  
  
array('title'=>$this->lang->line("common_resend"))  
        )  
    );  
    $i++;  
}  
return json_encode($response);  
}
```

The codes of a normal template :

```

<table id="custom-css" cellpadding="0" cellspacing="0"
width="630px">
  <tbody>
    <tr class="record">
      <td style="display: none;" class="editHandle"
valign="top">
        <a class="change_img" href="###" title="Change Image">
          <img
src=http://yang.faarao.fi/Mailer2/assets/iconset/im
age_edit.png />
        </a><br>
        <a class="change_color" href="###" title="Change BG
Color">
          </a><br>
        <a class="copy_row" href="###" title="Copy Row">
          
          <img src=
"http://yang.faarao.fi/Mailer2/assets/iconset/cross.png" />
        </a>
      </td>
      <td class="img-component" colspan="3"

```



```
</td>
<td class="txt-component" colspan="3" valign="top">
  <p>
    This is the example text. This is the example
    text. This is the example text. This is the
    example text. This is the example text. This
    is the example text. This is the example text.
    This is the example text. This is the example
    text. This is the example text. This is the
    example text. This is the example text. This
    is the example text. This is the example text.
  </p>
</td>
</tr>
</tbody>
</table>
```

The codes of function send_campaign :

```

function send_campaign($db) {
    $oridb = $this->load->database('default', true);
    $oridb->select("emails_to_send.id AS id");
    $oridb->select("emails_to_send.sender_name AS sender_name");
    $oridb->select("emails_to_send.sender_email AS sender_email");
    $oridb->select("emails_to_send.title AS title");
    $oridb->select("emails_to_send.content AS content");
    $oridb->select("emails_to_send.to_list AS to_list");
    $oridb->select("emails_to_send.send_time AS send_time");
    $oridb->select("emails_to_send.campaigns_id AS campaigns_id");
    $oridb->select("emails_to_send.user_id AS user_id");
    $oridb->select("campaigns.name AS campaigns_title");
    $oridb->select("campaigns.description AS
campaigns_description");
    $oridb->from($db."emails_to_send");
    $oridb->join($db."campaigns",
        "emails_to_send.campaigns_id=campaigns.id", "left");
    $query=$oridb->get();
    foreach($query->result() as $row) {
        if(!empty($row)) {
            $this->email->clear(TRUE);
            if($row->send_time<date("Y-m-d H:i:s")) {
                $sender_name = ucwords($row->sender_name);
                $sender_email = strtolower($row->sender_email);
                $title = $row->title;
                $to = $row->to_list;
            }
        }
    }
}

```

```
        $content=$row->content;
$this->email->from($sender_email, $sender_name);
        $this->email->to($to);
        $this->email->subject($title);
        $this->email->message($content);
        $this->email->send();
        $oridb->where("id", $row->id);
        $oridb->delete($db.".emails_to_send");
    }
}
}
return true;
}
```