

Building a responsive web application with the MVC PHP framework

LAHTI UNIVERSITY OF APPLIED
SCIENCES
Degree Programme in Information
Technology
Bachelor's Thesis
Spring 2018
Jevgeni Anttonen

Lahti University of Applied Sciences
Degree Programme in Information Technology

ANTTONEN, JEVGENI: Building a responsive web application
with the PHP MVC framework

Bachelor's Thesis in Software Engineering, 71 pages, 2 pages of
appendices

Spring 2018

ABSTRACT

The main purpose of this thesis was to follow the process of developing a mobile friendly web application by utilizing the Model View Controller design pattern and the latest available set of technologies for responsive web application development. The application was developed by using Codeigniter 3 PHP framework in combination with Bootstrap 3 frontend component library and MySQL database. Additional technologies such as AngularJS, jQuery, HTML5 and CSS were also used in order to build the user interface and frontend functionality of the application.

The theoretical part of the thesis presents technologies that were used during its creation with the main focus on the server-side functionality and Codeigniter 3 framework. In the empirical section the process of application development is explained step by step and examples of technologies and different application components are provided.

Key words: MVC design pattern, Responsive Web Design, Codeigniter 3, Bootstrap 3, MySQL, AngularJS, Web Development

Lahden ammattikorkeakoulu
Tietotekniikka

ANTTONEN, JEVGENI:

Responsiivisen web-sovelluksen
toteuttaminen PHP
MVC-ohjelmistokehityksellä

Ohjelmistotekniikan opinnäytetyö, 71 sivua, 2 liitesivua

Kevät 2018

TIIVISTELMÄ

Opinnäytetyön tavoitteena on seurata responsiivisen ja mobiiliystävällisen web-sovelluksen kehittämisprosessi. Sovelluksen kehityksessä hyödynnetään Model-View-Controller suunnittelumallia ja muita ajankohtaisia teknologioita tarkoitettuja web-sovelluksen responsiivisen rakentamiseen. Sovellus on tehty käyttäen Codeigniter 3 PHP sovelluskehystä, Twitter Bootstrap 3 käyttöliittymän toteuttamiseen tarkoitettua työkalua ja MySQL tietokantaa. Työssä myös otetaan huomioon muita teknologioita ja työkaluja, joita käytettiin sovelluksen käyttöliittymän ja frontend- toiminnallisuuden toteutuksessa.

Teoriaosuudessa käydään läpi kaikki projektissa käytetyt teknologiat ja työkalut. Tässä vaiheessa kiinnitetään erityistä huomiota sovelluksen palvelinpuoleen ja Codeigniter 3 sovelluskehukseen. Toiminnallisessa osuudessa kerrotaan miten valittuja teknologioita käytetään projektin toteutuksessa ja tutkitaan sovelluksen kehittämisprosessin askel askeleelta.

Asiasanat: MVC suunnittelumalli, Responsiivinen Web-suunnittelu, Codeigniter 3, Bootstrap3, MySQL, AngularJS, verkko-ohjelmointi

TABLE OF CONTENTS

1	INTRODUCTION	1
2	BASIC CONCEPTS	3
2.1	Responsive Web Design	3
2.2	Model-View-Controller	5
3	BACKEND ARCHITECTURE	9
3.1	Codeigniter 3	9
3.2	Features of Codeigniter 3	10
3.3	File Structure of Codeigniter 3	12
3.4	Application flow	13
3.5	Routing	14
3.6	Caching	16
3.7	Securing application with Codeigniter	17
3.7.1	Best Practices	17
3.7.2	SQL Injections	18
3.7.3	XSS and CSRF attacks	18
3.8	Model-View-Controller	20
3.8.1	Controller	20
3.8.2	Model	21
3.8.3	View	22
3.9	Codeigniter additional files	22
3.9.1	Helpers	22
3.9.2	Libraries	23
3.9.3	Hooks	24
4	USER INTERFACE	26
4.1	Twitter Bootstrap 3	26
4.2	File Structure of Bootstrap 3	28
4.3	Responsive Web Design with Twitter Bootstrap 3	29
4.3.1	Grid system	29
4.3.2	Flexible images	32
4.3.3	Entry forms	33
5	ADDITIONAL TECHNOLOGIES	35
5.1	HTML5, CSS3, JS and jQuery	35

5.2	MySQL	36
5.3	AngularJS	36
6	PRACTICAL IMPLEMENTATION	38
6.1	Concept and requirements	38
6.2	Application database	41
6.3	Installation and configuration	43
6.4	MVC components	47
6.4.1	Controllers	47
6.4.2	Models	50
6.4.3	Views	55
6.5	Application overview	58
6.5.1	Login page	58
6.5.2	User personal information	60
6.5.3	Internship navigation list	62
6.5.4	Internship plan and information	64
6.5.5	Internship goals	65
6.5.6	Career goals	67
6.5.7	Reports and references	69
7	CONCLUSION	70
	LIST OF REFERENCES	72
	APPENDIXES	77

1 INTRODUCTION

There are many hidden aspects and production risks related to the process of developing and maintaining of any complicated web application and these aspects and risks have to be taken into account before getting started. There are client-side user interface building and frontend design decisions to be made that have to satisfy the end-user requirements. There is server-side business logic and hidden functionality, required to control the flow of data between end-users and an application database, and also perform other specific tasks behind the scene. And finally, there is the continuous post-release process of tracking issues and improving application performance in order to stay up-to-date with modern web industry standards.

The main goal of this thesis is to follow the process of development of a responsive and mobile-friendly web application relying on Model-View-Controller pattern principles to build its server-side functionality. The application in question contains three core components: relational database designed and implemented using the MySQL database management system, backend functionality built with the Codeigniter 3 MVC framework and frontend design using the Twitter Bootstrap 3 component library. This thesis mostly focuses on the server-side application development with less attention being paid to the database and frontend design.

The project described in this thesis was initially started in 2014 as an internship portal for Lahti University of Applied Sciences (LUAS). It was dedicated for students who wanted to store and manage their internship-related information in digital form as well as for LUAS placement coordinators and teachers to keep track and control the internship status of their students.

The thesis consists of two main sections: theoretical background and practical implementation. In the theoretical part, both Codeigniter 3 and Bootstrap 3 frameworks, their features, components and, in the case of

Codeigniter 3, application flow structure will be explained in detail, providing additional code examples when necessary. Additional technologies, such as the MySQL database management system or the AngularJS framework, which were also used in the project but are out of the scope of this thesis research, will be only briefly described in this section.

The practical implementation part includes the creation of a relational database and its integration with the Codeigniter 3 framework and the process of building and interconnecting different MVC components. It will also briefly describe the process of designing the application's UI using Twitter Bootstrap 3 components in combination with HTML, CSS and JavaScript elements. Different parts of the application prototype such as authentication, internship data management and file upload systems will also be explained in this part.

2 BASIC CONCEPTS

2.1 Responsive Web Design

Responsive Web Design is a term that became popular in modern web development since the introduction of Web 3.0, following the growth of social media and mobile device industry. The term “responsive” describes a modern approach to web design according to which the application is considered responsive if all its web pages and a summary of their content can render and display their data correctly on a wide variety of devices and screen resolutions (Bhaumik 2015, 29). Therefore, it means that all web applications following the responsive web design pattern should be able to provide the same quality of content, design decisions and overall performance across all supported devices indiscriminately in order to ensure better accessibility and end-user satisfaction.

The list of supported devices may include for example smartphones, tablets, laptops and normal desktop PCs. A responsive application layout will automatically adjust itself to fit these devices and their screen resolutions as is shown in Figure 1.

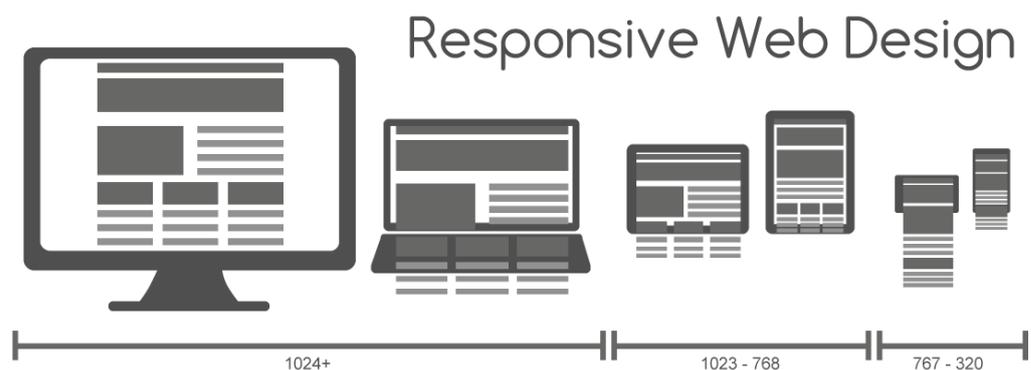


Figure 1. Responsive Web Design

There is a core set of elements which application developers have to apply in order to make its frontend layout more responsive. The core elements of

responsive web design include fluid grids, flexible images and CSS3 media queries.

Fluid grid layout represents a technique where the majority of the application components is described in relative units and can thus be easily adjusted to fit any display resolution. When compared to more traditional fixed content representation techniques, fluid grid design generally uses percentage values rather than pixels, points or other absolute units. (De Graeve 2011.)

In addition to fluid grids, flexible images and similar media sources are also sized in relative percentage values. The easiest way to prevent a specific image from exceeding the width of its container element is to set its CSS max-width as 100%. Images with their max-width set to 100% will automatically scale down to the same size as their container components. Alternatively, images that are either smaller or the same size as their container components will display their full size. (Bradley 2011.)

Media queries are a special CSS feature first introduced in CSS 3. They can be used to apply different CSS rules to the existing web page components depending on the capabilities of the device these components will be displayed on. The list of capabilities includes for example the height and width of the device, its screen resolution, aspect ratio and orientation. (De Graeve 2011.)

A faster and more efficient way for web developers to implement these responsive web design techniques in their applications is to use an appropriate frontend framework that will provide all necessary functionality out of the box. There are quite a few mobile friendly frameworks available in the industry at the moment and one of them, called Twitter Bootstrap 3, is used in this project.

Twitter Bootstrap is a popular frontend framework based on the “mobile first” philosophy, released in August 2010 by Twitter. In this thesis the theoretical basis for Twitter Bootstrap 3 core file structure and components will be presented. There will also be a practical example of how to develop

responsive web applications using this framework in combination with Codeigniter 3.

2.2 Model-View-Controller

Model-View-Controller or MVC is the name of a design pattern that separates an application core structure into three logical component types: the models, the views and the controllers. Using program languages such as PHP, with a strong focus on object-oriented programming paradigms, it is a powerful tool frequently utilized in modern backend development to create scalable and extensible applications, able to quickly adapt to changes and developer needs. (Tutorialspoint 2017a.)

The clean structure of the Model-View-Controller pattern provides a good starting ground for the responsive web application development in which separate program components can be further exchanged, revised or reused with minimum effort from the application developer. In a properly designed MVC application these operations will also have a small impact on the application source code or its other components.

A faster and more efficient way to develop MVC-based applications is to utilize a dedicated web framework that comes with all necessary functionality out of the box. There are multiple factors to consider when selecting an appropriate MVC framework for the needs of a specific project. According to Garbade (2016), it may be useful to narrow the selection by answering a few related questions first:

- What are the main features and functionality of the framework?
- What is its learning curve?
- How actively developed and supported is the framework in question?
- Does it provide all required documentation?
- Does it have any strong and active community support?

For example large and complex web frameworks with a rich set of features may be better suitable for big and long-term maintenance projects but are not necessarily an optimal choice for smaller ones. Similarly, frameworks

with a steep learning curve, while often being able to provide more built-in functionality, will take more time and resources to learn how to utilize them properly. It is also important to keep in mind that even if a particular framework is up-to-date at the moment it may quickly become outdated receiving no further support from developers. (Garbade 2016.)

Another important factor to consider is how actively the framework in question is supported by the community. Having a strong and active community of developers and end-users is important to secure further growth and longevity of the framework. Sharing user experiences, creating new resources, writing guides and making extensions is just a small part of what an active framework community can do. (Garbade 2016.)

With that in mind, there are many popular PHP frameworks in the industry that support the MVC pattern. The initial framework selection for this thesis is based on the results of “The Most Popular Framework of 2015” survey held by SitePoint (Skvorc 2015) and limited to the few widely known and actively supported frameworks. The results of this survey in the “PHP Framework Popularity in Personal Projects” category are presented in Figure 2.

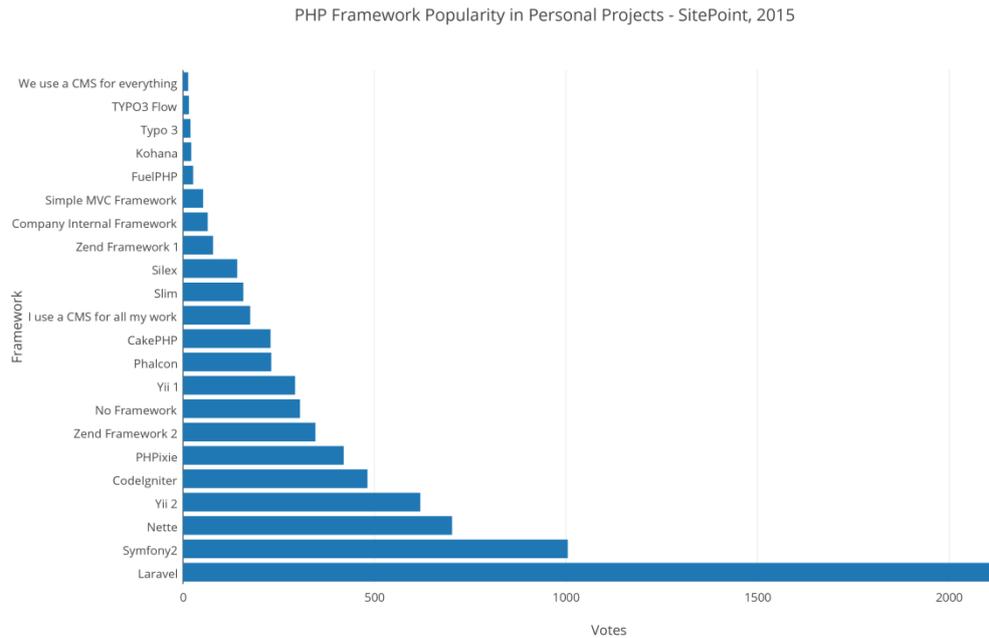


Figure 2. PHP Framework Popularity in Personal Projects

Codeigniter 3, Laravel, Yii 2 and Zend are the four main PHP frameworks that were researched and considered for this application development during the planning phase of the project. The feature-based comparison between these frameworks can be seen in Table 1.

Table 1. PHP Framework Comparison

Features	Codeigniter	Laravel	Yii	Zend
License	MIT-license	MIT-license	BSD license	BSD license
Programming language	PHP	PHP	PHP, JavaScript	PHP
Operating system	Cross-platform	Cross-platform	Cross-platform	Cross-platform
MVC support	Yes	Yes	Yes	Yes
MySQL support	Yes	Yes	Yes	Yes
ORM	Yes	Yes	Yes	Yes
Caching	Yes	Yes	Yes	Yes
Template Engine	PHP, 3d parties only	Blade, fabric	Twig, Smarty	PHP, Twig, Smarty
Programming Paradigm	Component-Oriented	Object-oriented Event-driven, Functional	Object-oriented Event-driven	Object-oriented Event-driven

In the end, the Codeigniter 3 framework was selected for its smaller size, tidiness and simplicity still satisfying all basic project requirements, being able to provide good performance without increasing development time or

cost. Its clear documentation and large active community also made it easier to find all necessary information through dedicated forums and blogs.

Codeigniter 3 is a compact, fast and very lightweight framework developed by Ellis Lab. It is an open source MVC framework which is well suited for small and medium scale projects. Codeigniter comes with a clear interface and offers a wide selection of built-in libraries and extensions for many common developing tasks. In addition, it has a logical file structure with no restrictive coding rules to access and utilize these libraries efficiently. Just like all the frameworks mentioned above, Codeigniter 3 can be used to manage relational databases by providing its own functional database class with many methods that can also be further extended.

The next section of this thesis provides a brief explanation of the history, main features and the application architecture of the Codeigniter 3 framework based on the information available in its documentation (BCIT 2017) and related information sources.

3 BACKEND ARCHITECTURE

3.1 Codeigniter 3

Codeigniter is a PHP-based MVC framework developed by Rick Ellis and released by EllisLab on February 28, 2006. Initially it was based on the collection of refactored classes written for ExpressionEngine, a commercial content management system also developed by EllisLab. Eventually transformed into an independent frontend framework, Codeigniter was presented by EllisLab as a simple and efficient tool designated for rapid development of web sites and applications. (EllisLab 2017.)

On January 28, 2011, version 2.0 was released and the development process was separated into two main branches: Codeigniter Core and Codeigniter Reactor. Codeigniter Core was a heavily tested and slowly updated official version maintained entirely by EllisLab, while Codeigniter Reactor was a branch managed by the Codeigniter community of developers. The main intention of the Codeigniter Reactor project was to merge in possible community modifications without drastically changing the core structure of the framework. (Codeigniter 2010.)

On July 9, 2013, EllisLab officially announced that the Codeigniter framework will not receive any further updates and they were seeking a new owner for it. On October 6, 2014, EllisLab once again announced that framework will continue its growth under the stewardship of the British Columbia Institute of Technology (BCIT). On March 30, 2015, Codeigniter version 3.0.0 was released (EllisLab 2017).

According to the information provided by TechNet Experts portal (PHPTeam 2016), there were multiple changes in Codeigniter 3.0 compared to older versions including:

- License change from proprietary to the MIT license
- Introduction of a default database driver known as 'mysqli'
- New and updated list of supported MIME types
- New and updated list of libraries, helpers and other specific files

The most recent version of Codeigniter is 4.0, currently in alpha development phase and yet to be officially released.

3.2 Features of Codeigniter 3

As a PHP framework Codeigniter 3 has a relatively small code structure. According to Codeigniter developers, it was intentionally designed this way to maximize its simplicity, performance and flexibility through minimal automation and a limited set of built in libraries compared to similar PHP web frameworks (BCIT 2017). It is intended for developers who prefer speed and tidiness over complexity of heavyweight frameworks, as well as for people who just start studying PHP frameworks and are not very familiar with the MVC pattern architecture.

Table 2 provides an overview of the basic specifications of the Codeigniter 3 PHP framework:

Table 2. Codeigniter 3 Basic Specifications (1&1 2017)

Codeigniter	
Developer	British Columbia Institute of Technology (BCIT)
Current release	Version 3.1.2
Design pattern	MVC/HMVC, Active Record, Chain of Responsibility
Necessary knowledge	PHP, Object-Oriented Programming (OOP)
Programming language	PHP 5.6 or higher
License	MIT License
Database support	MySQL (5.1+), Oracle, PostgreSQL, MS SQL, SQLite, CUBRID, Interbase/Firebird, ODBC
ORM	Only through third parties
Caching	Yes
Template engine	No
Namespaces	No
PHP auto-loading	No
Search machine-friendly URLs	No
Security features	Cross-Site-Request-Forgery (XSRF), Cross-Site-Scripting (XSS), SQL-Injection
Testing library	PHP Unit

As can be seen from the table above, Codeigniter 3 is a lightweight framework with mostly basic functionality and has some disadvantages that may limit its usefulness for large scale application development.

Codeigniter's default library selection is lacking compared to more advanced PHP frameworks such as Symfony or Larafel. While it leads to a smaller size, better loading speed and less complexity, it may also be a limiting factor when building applications that require access to specific libraries in order to work properly. Many of these libraries, however, can still be integrated from third party sources. (1&1 2017.)

Codeigniter has no native Object-Relational Mapping (ORM) support, which means that ORM in Codeigniter can only be provided via third party extensions (1&1 2017). ORM is an object-oriented programming method dedicated to map application database tables into objects that can be used to manipulate data from the database directly via PHP code.

Finally, Codeigniter 3 does not have a fully functional built-in template engine, instead providing a limited template parser class. The template parser can still be used to assign a simple text substitution for pseudo-variables, but its functionality, compared to modern template engines, is intentionally restricted in order to maintain maximum performance. Codeigniter still has access to more advanced template engines such as Twig through third party library extensions. (1&1 2017.)

3.3 File Structure of Codeigniter 3

The file structure of Codeigniter is demonstrated in Figure 3.

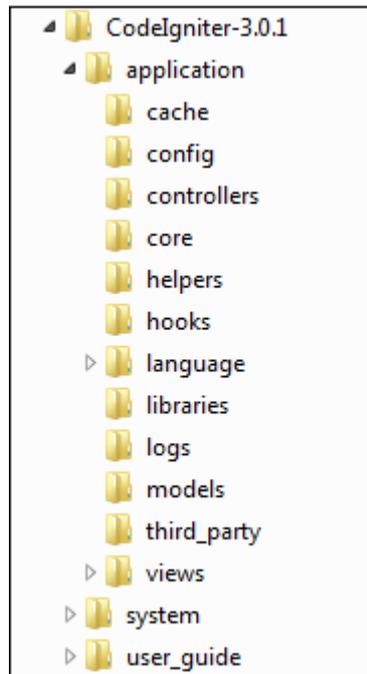


Figure 3. File structure of Codeigniter 3 (Tutorialspoint 2017b)

Files in Codeigniter 3 are divided between three main directories:

- Application files: this directory contains all of the application-related source code files.
- System files: this directory stores Codeigniter core components and source code files such as libraries and helpers.
- User Guide: this directory contains an offline version of the Codeigniter User Guide.

Codeigniter is easy to set up and start working with out of the box. After uploading all Codeigniter folders and files to the web server, the minimal configuration requires only setting up the config.php and index.html page access files. Both these files, along with any additional configuration files, can be found in the application/config folder of the Codeigniter root directory. Codeigniter's config.php file may be used to define a lot of application starting options, including base site URL routing, index file name, default language and character set, log file settings, data formats,

cookie policy and many others (BCIT 2017a). If database connection is required at that point, it can be configured via `database.php`. It is also recommended to manually rewrite or create a new `.htaccess` file to remove 'index.php' from the application's URL links. More information about Codeigniter installation and setup will be given in the practical part of this thesis.

3.4 Application flow

This section focuses on the application data flow and covers multiple application parts and interactions between them inside the Codeigniter PHP framework. The flow structure of an application is made up of multiple interconnected components, as is shown in Figure 4.

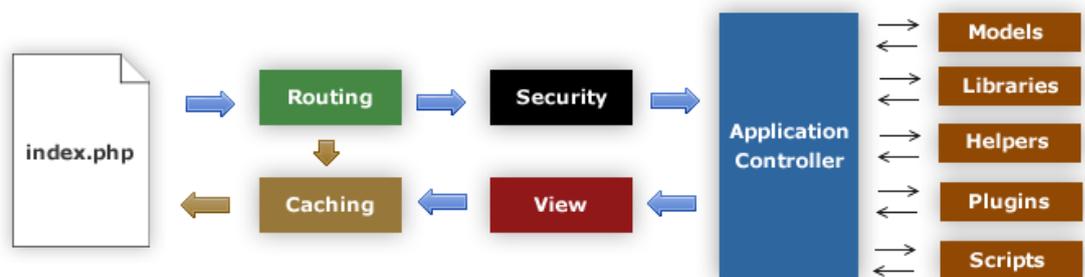


Figure 4. Application flow (Tutorialpoint 2017b)

First all HTTP requests go through the `index.php` file, which Codeigniter uses as its front controller, initializing base resources needed to start the application. From `index.php`, Codeigniter sends each HTTP request to the router in order to determine how to process them correctly. The router will first check if the cache file for each specific request exists. If it does, then the request will be cached and delivered back to the browser bypassing any further system interaction. If it does not, Codeigniter will determine which controller class will respond to the request, but before the controller is loaded, the HTTP request and any additional data must be processed by a security filter. The security filter is the core part of the Codeigniter PHP framework that will try to detect and intercept any harmful actions or hacking

attempts. Any uncached HTTP request in Codeigniter has to undergo a security filter check first. (BCIT 2017b.)

From there on the normal MVC pattern is launched. First, once the correct controller is executed, it will load all views, models, libraries, helpers, plugins and other additional files and extensions required to answer that specific browser request. Once all collected data has been processed, generated and stored in the view file, it can be rendered and sent back to the web browser. If the caching option is enabled at that point and no active cache file exists, the view will be cached first so the router can directly resend the same data to the browser on all subsequent HTTP requests. (BCIT 2017b.)

3.5 Routing

By default, Codeigniter decides which controller class and method should be loaded by comparing a given URL string to its corresponding controller class and methods. Normally it follows the “class/method/parameter” URL structure. In some cases, however, this basic path structure can be manually adjusted. For example a “class/method” route can be replaced with a single word “product”. In order to do that the new routing rules have to be defined first in the routes.php file located in the application/config folder of the Codeigniter root directory. This file allows developers to define their own routing criteria by manually setting specific \$route-arrays. All routes inside the routes.php file will be executed in order of appearance, starting from default routes. (BCIT 2017c.)

There are three default routes in the routes.php file as is shown in Figure 5:

```
/* default routes */
$route['default_controller'] = "auth";
$route['404_override'] = 'error/_404';
$route['translate_uri_dashes'] = FALSE;
```

Figure 5. Default routes

The first route defines the standard controller of the application. The standard controller will be displayed to the user if no additional routing information has been given. This route accepts the “controller/method” string as its first and second parameter and will use the “index” method by default. If there is no specific controller defined as standard, Codeigniter will instead redirect visitors to the 404-error page. (BCIT 2017c.)

For example, in the code above the routing rule will define the “auth”-class as a standard controller of the application and redirect any visitor with no specific webpage selected back to the authentication page. After that, as there is no second parameter specified, it will call the default “index” method to proceed with an authentication procedure.

The second route specifies which controller will be called if the requested route is not found. By default it will show the 404-error page. It is usually recommended to avoid using error pages without necessity as it makes it harder for search engines to access relevant data and may have a negative impact on the end-user’s site navigation experience. (BCIT 2017c.)

The last route is a Boolean type option and by default it will replace any found dashes with underscores inside both controller and method names. Dashes are not allowed to be used in class and method names and will cause a fatal error if not replaced. (BCIT 2017c.)

Specific routing rules for dynamic URLs in Codeigniter can be defined by using either Wildcards or Regular Expressions.

Wildcard routes are specifically made to respond to dynamic parts of URI requests and serve as placeholders for normal routing entries. By default Codeigniter supports two types of wildcards: ‘num’, which is used to match segments containing only whole numbers and ‘any’, which is used to match segments containing characters or strings. (BCIT 2017c.)

Another way to define routing rules in Codeigniter is to use regular expressions. Codeigniter accepts all valid regular expressions and

back-references and allows mixing and matching both wildcards and regular expressions in the same sentence. (BCIT 2017c.)

3.6 Caching

Web caching is a technology that can be used to temporarily store application responses to HTTP requests in order to make subsequent HTTP requests faster and, therefore, improve application's performance. This is especially useful in heavily used web applications designed to operate with a large amount of dynamic data and process multiple subsequent HTTP requests at the same time.

Codeigniter 3 allows developers to cache their web pages. If web caching is enabled and the cache files are allowed to be written in the file permission settings, then for the first time the specific web page is loaded, Codeigniter will create a new cache file for it in the application/cache folder. After that, on any subsequent page request the existing cache file will be resent to the requesting web browser bypassing any further controller interaction in the process. The cache option can be enabled for each controller separately on a limited time basis and once the cache file has expired it will be deleted and recreated on the following page request. (BCIT 2017d.)

To enable caching in Codeigniter a special tag must be placed in the requested controller method, as is shown in Figure 6, where \$n variable defines a number of minutes the web page will be cached before being refreshed.

```
$this->output->cache($n); // set cache on
```

Figure 6. Enabling the cache file in Codeigniter

In order to delete existing cache file it is enough to remove the caching tag from the selected method and it will no longer be refreshed after being normally expired. Alternatively, Codeigniter 3 allows deleting cache files manually by using corresponding output class methods. (BCIT 2017c.)

3.7 Securing application with Codeigniter

Application security methods in Codeigniter can be loosely divided into three main categories: 'best practice' features, tools to deal with SQL injections and tools to prevent Cross Site Scripting (XSS) and Cross-site Request Forgery (CSRF) attacks.

3.7.1 Best Practices

By default Codeigniter provides some features that follow the application security best practices and helps web developers to integrate these features into their applications. Internal security features in Codeigniter include many different components such as:

- URI character restrictions
- native PHP error settings
- password handling tools
- form validation and string escape methods

For example URI character restriction settings in Codeigniter 3 provide a set of rules that can be used to define specific characters inside the URI strings. The default Codeigniter URI rules are rather restrictive and allow URIs to contain only standard characters such as Latin letters, numbers, tildes, percent signs and underscores. These restrictions help to minimize the possibility that harmful data will pass through the browser to the application. (BCIT 2017e.)

Another important aspect of application security is password handling. This section contains different utilities, tools and algorithms related to the password hashing, encoding and encryption processes. Codeigniter developers strictly recommend using only strong hashing algorithms such as BCrypt to prevent user passwords from being decrypted or decoded. (BCIT 2017e.)

In addition, Codeigniter 3 comes with a built-in Form Validation Library that can be used to validate, filter and manage user input data. It's

recommended to always validate all input data passing through the application and confirm that it follows certain input patterns. (BCIT 2017e.)

3.7.2 SQL Injections

SQL injections are SQL database queries sent from the client to the application. If being successful the injection can access specific data from the database, read and modify it (OWASP 2016). There are a few methods that the Codeigniter 3 framework can use to prevent or restrict SQL injections. These methods include for example escaping queries and query bindings.

Escaping queries can be used to escape data before submitting it into the database. For example the 'db->escape ()' method will automatically escape the string data type. It will also automatically add single quotes around the data. (BCIT 2017e.)

Query bindings help to manage the query syntax by putting queries together automatically and by replacing the question marks with actual data strings in the process. An example of query binding is demonstrated in Figure 7:

```
/* query binding */  
$qry = "SELECT * FROM table WHERE id = ? AND username = ?";  
$this->db->query($qry, array(5, 'Pete'));
```

Figure 7. Query binding

The main benefit of this method for preventing SQL injections is that all values inside the query will be automatically escaped in the process (BCIT 2017e).

3.7.3 XSS and CSRF attacks

Cross site section attacks attempt to inject malicious JavaScript code pieces from the client into the application. It can be further used to send this code in the form of client-side scripts to end-user computers using injected

application as a messenger. XSS attacks can be very dangerous if not detected and prevented quickly, especially in large scale applications. They may access user cookie files, browsing history and other sensitive information related to the web browser activity. (OWASP 2017a.)

Cross-site request forgery (CSRF) is a more straight-forward attack method that tries to trick users to execute unintended operations using the web application they are currently authenticated in. Compared to XSS, CSRF usually requires from users to take specific actions in order to trigger state changing requests. (OWASP 2017b.)

Codeigniter 3 offers an entire Security class, which contains different methods to prevent both XSS and CSRF attempts. These methods include for example XSS filters and CSRF security tokens.

XSS filters are a prevention mechanism that will search the input data for all commonly used XSS attack types. XSS filters can be used to filter any incoming data passing through the application. In Codeigniter 3 the 'xss_clean()' method is used to filter an incoming data through the XSS filter. (BCIT 2017e.)

Cross-site request forgery protection tokens can be enabled by switching the parameter called csrf_protection in the config.php file from FALSE to TRUE (Figure 8). After initialization the tokens can be either recreated on each individual submission, which is set by default, or kept alive through the entire lifespan of a CSRF cookie. This behavior can be changed by editing the csrf_regenerate parameter inside the same config.php file.

```
$config['csrf_protection'] = FALSE;
$config['csrf_token_name'] = 'csrf_test_name';
$config['csrf_cookie_name'] = 'csrf_cookie_name';
$config['csrf_expire'] = 7200;
$config['csrf_regenerate'] = TRUE;
$config['csrf_exclude_uris'] = array();
```

Figure 8. CSRF protection token

Once CSRF protection is active and the `form_open` method is used to create a HTML form tag it will automatically insert a hidden CSRF field in that form. Alternatively it is also possible to create CSRF tokens manually. (BCIT, 2017e.)

3.8 Model-View-Controller

This chapter provides a short overview of the three main MVC components available in Codeigniter 3. It will also list their roles in the application flow and possible interactions with each other. More detailed explanation of each component and their functions will be given in the practical section of this thesis.

Discussing MVC implementation in Codeigniter 3, it should be mentioned that while both view and controller components are mandatory to build an application, its connection to the database via models is entirely optional and can be ignored. This loose MVC approach means that if developers at some point consider that maintaining the complete model structure requires more resources than they can afford, they can just skip it and build their application minimally using only controllers and views. (1&1 2017.)

3.8.1 Controller

Simply put the controller is the main component of any MVC-based application that connects models and views together. Controllers serve as a buffer between Model and View components, isolate the business logic of the model from the user interface layout of the view and handle data interactions between them.

Model-View-Controller basic interaction can be seen in Figure 9.

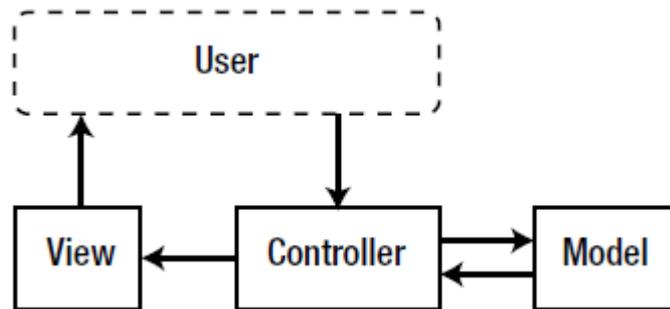


Figure 9. MVC Interaction (Pitt 2012, 2)

Controllers can also be considered as a first point of entry in MVC-based applications. An incoming browser request will be passed to the dedicated controller first, which will try to process it and instantiate correct models and views in order to respond to it properly. (Pitt 2012, 1.)

3.8.2 Model

The model is a MVC component, in which the application business logic is stored. The term 'business logic' refers to all data and business functionality that the specific application works with. It describes how the application in question stores and manages its data or uses third party services in order to fulfill its business requirements. (Pitt 2012, 1.)

In practice this means that outside of few possible exceptions all database-related code will be placed inside the model classes. For example, when application retrieves specific data from the database, updates and puts it back into the database, it requires an appropriate model or set of models to define these interactions.

In Codeigniter 3 models are optional classes designed to operate with the application database and provide different methods to access and manage its data. Models can be loaded and called within the controller by using appropriate load-class methods. Alternatively models can be auto-loaded during the system initialization phase by adding the specific model to the auto-load array in the autoload.php configuration file. (BCIT 2017f.)

3.8.3 View

The view is the frontend part of an application in which all user interface components are stored. Anything that users can see or interact with through web browsers can be found in view files. The list includes for example HTML markup, CSS stylesheet and JavaScript files. All frontend and JavaScript based frameworks that work with data presentation are also presented here. (Pitt 2012, 1.)

Being a presentation layer of the application the view is basically a normal HTML page or a page segment that includes dynamically generated content via PHP code. The main difference of the MVC views from the normal web pages is that views, unlike their traditional counterparts, can never be called directly: they can only be loaded by calling their representative MVC controller methods. Multiple views can be combined with each other or embedded (nested) within other views in order to build more complex page presentation. Specific views such as headers, footers and RSS feeds can also be reused between multiple application controllers.

3.9 Codeigniter additional files

Codeigniter has a number of additional file types that can be used in the application development to improve its functionality and make it easier for developers and end-users to perform specific tasks. The most common additional file types in Codeigniter 3 are helpers and libraries.

3.9.1 Helpers

Helper files are a collection of common functions, which, as the name suggests, are assigned to help developers with routine application tasks. For example Codeigniter core package includes a HTML helper file that can be used to organize and manage HTML files. The main difference of the helper files from other specific file groups is that helpers don't use object oriented programming methods in their source code: they are simple procedural functions and, therefore, can be utilized in the same way as

normal PHP functions. Each helper file is designated to assist with one specific task at a time with no reliance on other helpers or functions (BCIT 2017g).

Helper files are stored in either global system/helpers or local application/helpers directory. Codeigniter will first check if the requested helper file can be found in the application/helpers directory. If the directory does not exist or the particular helper file is not found, Codeigniter will look for it in the global/helpers directory next. (BCIT 2017g.)

By default Codeigniter does not load any helper files so they have to be loaded manually. As with all similar file types defining helpers in the controller constructor method will make them globally available inside that controller. Alternatively helpers can be initialized locally inside the methods that require them. Finally, helpers can be initialized globally throughout the entire application by adding them to the array list in the “autoload.php” file. Along with helpers Codeigniter 3 auto-loader can be used to automatically load many different file types including libraries, custom config files, language packages and models. (BCIT 2017g.)

Even though helpers are not object oriented classes depending on the project requirements they can be manually extended. To extend an existing helper file it is enough to add a new file in the application/helpers folder, with the same name as the core Codeigniter helper, but prefixed with MY_. After that, it's possible to create a function inside this file with the same name as the original helper function to overwrite it. Writing new helper functions is also allowed. (BCIT 2017g.)

3.9.2 Libraries

Libraries in Codeigniter are object oriented classes. They provide a set of abstract methods that can be utilized across multiple web application components. Compared to helpers, libraries are larger scale and more complex development tools that can greatly extend the core application

functionality and make it easier to re-use already written code. (BCIT 2017h.)

Codeigniter comes with many pre-built core libraries, and even more are regularly released, extended and shared by the Codeigniter developer community. For example one of the most frequently used Codeigniter libraries is the Form Validation library, which provides different methods to validate HTML forms and work with data passing through them.

Application specific libraries are located in the application/libraries folder, while core Codeigniter libraries can be found in the system/libraries folder. Codeigniter 3 allows developers to extend its core libraries in order to add or rewrite specific functionality by placing a new version with the same name as the existing system library in the application/libraries folder. Developers are also allowed to create their own library files in the same folder. All library names and class declarations in Codeigniter must be capitalized and match each other. (BCIT 2017h.)

Libraries may include multiple specific files in their core packages. The special case of the library class in Codeigniter 3 is called driver. As object oriented classes, drivers always have one parent class, which is usually the original library class and can have an unlimited number of child classes. Child classes of the same driver can contact with their representative parent class but cannot contact with each other. Drivers can be found inside the “system/libraries” directory in their own sub-folder which name is identical to their parent library class name. (BCIT 2017i.)

3.9.3 Hooks

Hooks are an event-like mechanism that can be called at eight different stages also known as hook points during program execution. Hooks help developers manually adjust application flow logic without changing the framework core functionality. In order for hooks to be called the hook points should be defined first. The hook point is basically a trigger condition that defines when and how the hook in question will be called. For example

'pre_controller' hook point will be triggered prior to any of the application controllers being loaded. After being enabled hooks can be initialized in the application/config/hooks.php file. (BCIT 2017j.)

4 USER INTERFACE

4.1 Twitter Bootstrap 3

Twitter Bootstrap is a frontend framework which technically can be described as the collection of flexible web design elements and methods. Bootstrap allows developers to choose the components they want to include in their projects and ensures that selected components will not conflict with each other.

The first version of the Twitter Bootstrap framework was designed as a style guide with the main purpose to help solve design inconsistencies within Twitter's projects. It was first introduced at Twitter in mid-2010 by two (currently former) Twitter engineers Mark Otto and Jacob Thornton. On August 19, 2011, Bootstrap was officially released as an open source project under the MIT license on GitHub. It was well received and within few months contributed by thousands web developers across the world. (Utterback 2014.)

Two years after, in August 2013 Bootstrap 3 was released. This version introduced multiple new features including new design direction and original UI theme, additional components and customization options, better dependency support and built-in error handling. Since 3.0 both mobile first and responsive web design approaches became the core part of the Twitter Bootstrap framework. (Bootstrap Blog 2013.)

As is shown in Figure 10 the basic structure of the Twitter Bootstrap 3 framework consists of CSS stylesheet files, JavaScript plugins and internal built-in components.

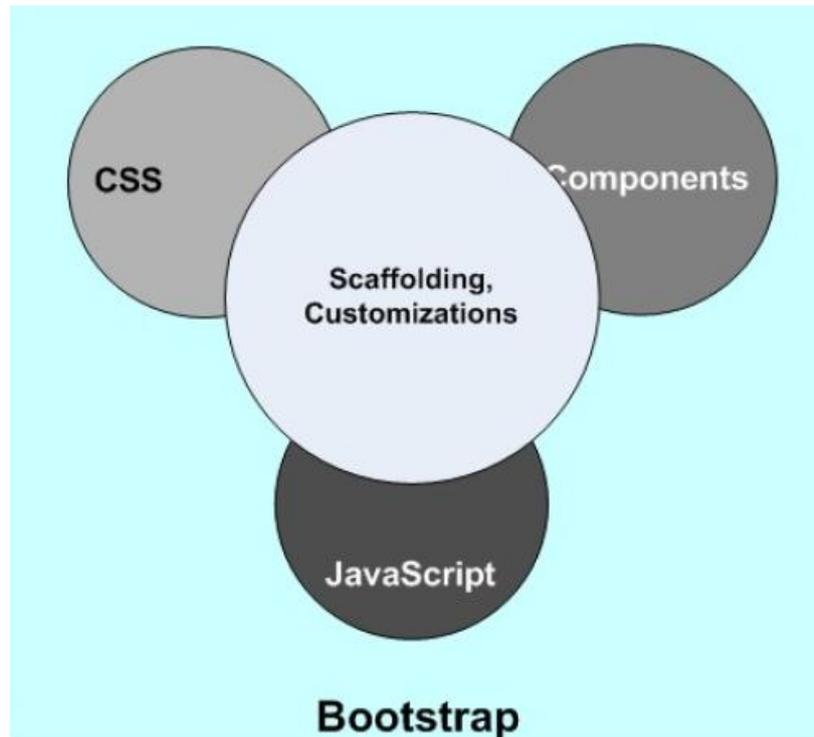


Figure 10. Twitter Bootstrap Core structure (Bhaumik 2015, 34)

In the core CSS package Bootstrap 3 provides global CSS settings and many common HTML elements further restyled and extended. It also comes with media queries and classes that are used to build advanced grid systems. Bootstrap 3 was designed to be mobile friendly out of the box and, therefore, all required mobile styles are already included in its core functionality. These files can be found through the entire Bootstrap 3 file library and don't require any separate tools to utilize them. (Otto & Thornton 2017.)

The JavaScript package comes with a number of ready-to-use jQuery Plugins that can be integrated into the project either individually or all at once. These files contain various jQuery scripts that can be used to display and interact with different HTML components. All JS plugins stored in this folder require jQuery core source file in order to work properly. (Otto & Thornton 2017.)

In addition, Bootstrap 3 offers a rich set of customizable built-in components. These components are technically custom CSS classes and unique HTML elements, each designed with specific purpose in mind. There

are many different UI components in Bootstrap 3, such as glyphicons, navigation bars, buttons, list groups, panels, tooltips and others. (Otto & Thornton 2017.)

4.2 File Structure of Bootstrap 3

There are two main methods to download and install Twitter Bootstrap 3, either as a precompiled bundle or as a full source code package. Each method has its own pros and cons depending on the project the Bootstrap framework is intended to be used in.

The file structure of precompiled Twitter Bootstrap 3 is demonstrated in Figure 11:

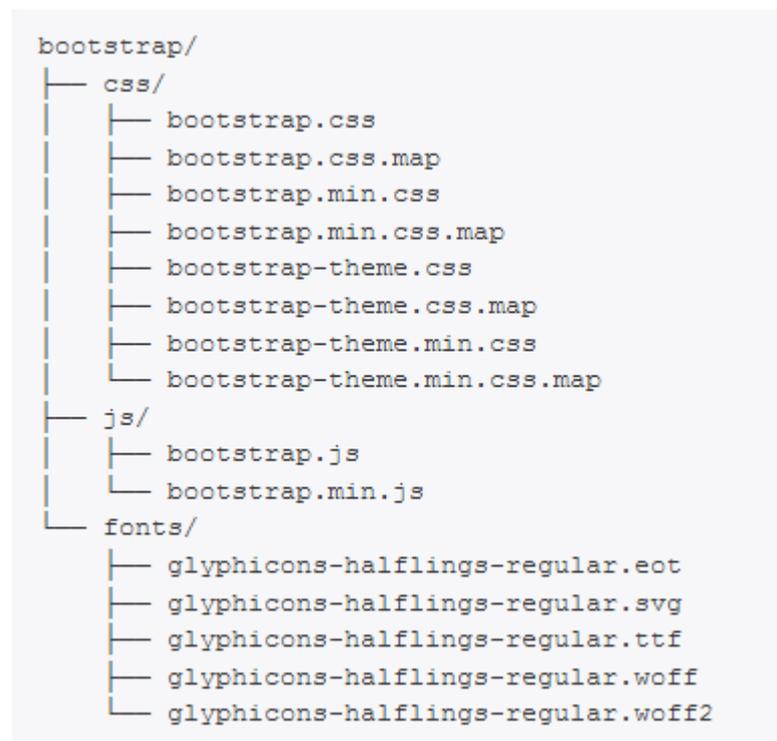


Figure 11. Bootstrap 3 precompiled (Otto & Thornton 2017)

The precompiled Bootstrap bundle contains all necessary CSS, JavaScript and font files, already compiled and minimized. It also includes Bootstrap 3 main UI theme but does not include any documentation or original source code files. It's essentially the most basic form of Bootstrap 3: the fully

functional version that can be loaded and immediately used in nearly any web application (Otto & Thornton 2017).

Compared to the precompiled bundle, the source code version has to be installed and compiled manually using the LESS compiler. Bootstrap source code version comes with LESS, JS and font source files and also provides full user documentation as is shown in Figure 12.

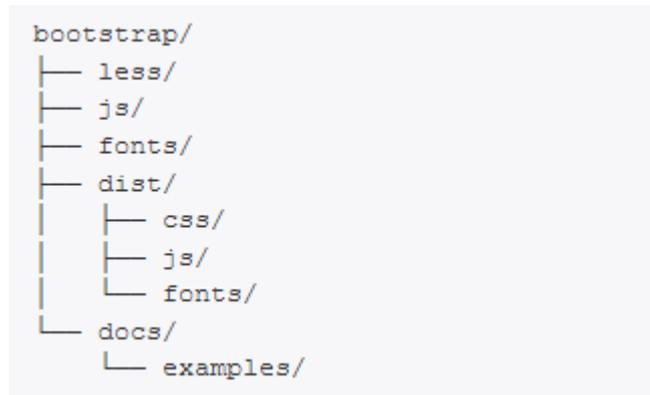


Figure 12. Bootstrap 3 source code (Otto & Thornton 2017)

The LESS files located in the less folder require a separate CSS preprocessor to compile them properly. All compiled output files will then be stored in the dist folder, which content will be exactly the same as in the precompiled version. (Otto & Thornton 2017.)

4.3 Responsive Web Design with Twitter Bootstrap 3

Bootstrap 3 presents many tools that can be used to make the frontend layout of the web application more responsive. Three techniques that have been frequently taken advantage of during the practical implementation of this thesis will be discussed in this chapter.

4.3.1 Grid system

In responsive web design the grid system represents a flexible page layout, which provides a set of rows and columns that can be used to store page contents. This grid layout is fully responsive and all its elements will

dynamically scale and re-arrange themselves based on the screen size and orientation of the device. Bootstrap 3 comes with a mobile-first approach based grid system that allows up to 12 columns per web page to be displayed as is demonstrated in Figure 13. (W3S 2017a.)

span 1											
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

Figure 13. Bootstrap Grid page layout (W3S 2017a)

According to W3S tutorial (2017a), Bootstrap 3 grid system contains four screen size defining classes and each class comes with a number arranging from 1 to 12. The numbers show how much space the element will take from the available 12 columns. These classes are listed below:

- .col-xs: for mobile devices, with display resolution of 768px or less
- .col-sm: for tablets and similar mobile devices, whose width is equal or greater than 768px
- .col-md: for small laptops and desktop devices, with the width equal or greater than 992px
- .col-lg: for large desktop computers and other devices with the screen size greater than 1200px

The grid system classes can be freely combined with each other in order to make application screen layout more responsive (W3S 2017a). For example, let's say there are two different HTML <div> elements in the same row: the first is defined as “.col-md-3 .col-lg-6” and the second as “.col-md-9 .col-lg-6”. In this case Bootstrap 3 will dynamically distribute these elements across the page in a 25%/75% split on devices with the screen resolution varying from 992 to 1200px and in a 50%/50% split on devices with the

screen resolution greater than 1200px. The HTML code will then look as follows:

```
<div class="container-fluid">
  <div class="row">
    <div class="col-md-3 col-lg-6">
      <p>first row content</p>
    </div>
    <div class="col-md-9 col-lg-6">
      <p>second row content</p>
    </div>
  </div>
</div>
```

Figure 14. Screen size class declaration in Bootstrap 3

As is demonstrated in Figure 14 the grid classes have to be placed in the class defined as 'row' in order to create a horizontal set of columns. After that, all row-elements must also be wrapped by the .container class. There are two types of container classes in Bootstrap 3: the .container class that will render stored content with a fixed width and the .container-fluid class that will expand its content to fill all available page width (Otto & Thornton 2017).

Table 3 summarizes how the Bootstrap Grid system works across multiple devices and screen resolutions.

Table 3. The overview of Bootstrap 3 Grid system (W3S 2017a)

	Extra small <768px	Small ≥768px	Medium ≥992px	Large ≥1200px
Class prefix	<code>.col-xs-</code>	<code>.col-sm-</code>	<code>.col-md-</code>	<code>.col-lg-</code>
Suitable for	Phones	Tablets	Small Laptops	Laptops & Desktops
Grid behaviour	Horizontal at all times	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints
Container width	None (auto)	750px	970px	1170px
# of columns	12	12	12	12
Column width	Auto	~62px	~81px	~97px
Gutter width	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)
Nestable	Yes	Yes	Yes	Yes
Offsets	Yes	Yes	Yes	Yes
Column ordering	Yes	Yes	Yes	Yes

4.3.2 Flexible images

To make normal images in Bootstrap 3 more responsive it is enough to add the `.img-responsive` class to their containing `` HTML element. Bootstrap 3 will then automatically apply `'max-width: 100%;'`, `'height: auto;'` and `'display: block;'` tags to the images and will make them scale to the size of their parent container element (W3S 2017d).

In addition, images in Bootstrap can be reshaped by adding corresponding image shaping classes to their `` tag. For example the `.img-rounded` tag will add rounded corners to the image, `.img-circle` will make it fully rounded and `.img-thumbnail` will shape the image to the thumbnail. It should be taken into attention that some image reshaping options such as `img-rounded` and `img-circle` are not supported by the Internet Explorer browser version 8.0 or lower. (W3S 2017d.)

4.3.3 Entry forms

Being an integral part of many web applications, web forms are used to gather information from the end-users. By default Bootstrap will automatically apply some global style settings to multiple form control elements such as `<textarea>` and `<select>`. This can be achieved by adding `.form-control` class to the corresponding form element. (W3S 2017e.)

Bootstrap 3 supports three types of form layouts: vertical forms, horizontal forms and inline forms. These form layouts can also be mixed together to make the page presentation more diverse.

Vertical forms in Bootstrap are set by default without any serious changes in the original HTML markup. Vertical form controls will be presented with left aligned labels on the top of form input fields. Any created HTML form with no further class definition will be considered by Bootstrap 3 as vertical. (Tutorial Republic 2017.)

In horizontal forms, labels are aligned on the same line next to their input fields. To make the form controls horizontal the form HTML element must be tagged with the `.form-horizontal` class and form `<label>` elements must be tagged with `.control-label`. Bootstrap will only render horizontal forms for devices with large and medium screen sizes. On small screens, with the width of 768px and below, horizontal forms will be automatically transformed back to vertical. (Tutorial Republic 2017.)

Finally, the inline form presentation method will place all form elements side by side left aligned, which will make forms look more compact. To make a specific form inline it is enough to add the `.form-inline` class to its `<form>` element. (Tutorial Republic 2017.)

To further modify the form presentation in Bootstrap 3, by applying additional HTML elements such as text, buttons and glyphs to the form input field, the `.input-group` classes can be used. For example the `.input-group-addon` class will tell Bootstrap that the text or glyph can be

attached next to the input field. Similarly the `.input-group-btn` class can be used to attach buttons.

5 ADDITIONAL TECHNOLOGIES

5.1 HTML5, CSS3, JS and jQuery

This section gives a brief overview of additional web technologies used in the process of creation of this application. The list of core web development technologies used in this project includes the HTML5 markup language, the CSS3 stylesheet engine and the Java Script language, which is represented by the jQuery frontend framework.

The “HTML” term stands for Hyper Text Markup Language and is used to describe the structure of web pages using HTML markup elements. Each HTML markup element covers a part of the actual content of the web page that will be interpreted, rendered and displayed to the end-users by their web browsers. HTML5 is the fifth and the latest revision of the HTML standard. It introduces a large list of new semantic attributes and input types that can be used to make document structure and semantics more organized and easier to understand. (W3S 2017b.)

The “CSS” term stands for Cascading Style Sheets. CSS files are responsible for visual presentation of web pages and are used to describe how each particular HTML element will be displayed on the screen. CSS3 used in this project is the latest version of the CSS language and introduces many new and useful features, such as selectors, rounded corners, border images, shadows, gradients, flexible boxes, grid layouts and media queries.

JQuery is a small, ‘write less - do more’-concept based JavaScript framework commonly used to build the frontend functionality of web sites and applications. JQuery presents a list of easy-to-use methods for specific frontend operations, which otherwise would require many lines of the JavaScript code to accomplish. It makes many complicated JavaScript techniques, such as AJAX calls and DOM element manipulations much simpler and faster to execute. JQuery also supports third party plugins that allow developers to extend its core functionality and add new methods required for specific tasks. (W3S 2017c.)

5.2 MySQL

MySQL is an open source relational database management system (DBMS) developed, distributed and maintained by Oracle Corporation. The database is technically a structured collection of data and in order to access, process or transfer this data between client and server components of the application the database management system such as MySQL is necessary (Oracle 2017).

Being the relational database management system MySQL stores data in separate tables and provides developers with all necessary tools to manage these tables. In addition the database management system can be used to set up rules and to define the relationships between different table fields. (Oracle 2017.)

To access data inside the database tables MySQL utilizes Structured Query Language also known as SQL. SQL can be either sent to the database directly in the form of SQL queries, through the code written in another language such as PHP, or by using language-specific APIs, that will hide the SQL syntax from the user (Oracle 2017).

MySQL DBMS uses the client-server architecture, which means that in order to function properly it requires a dedicated MySQL server or mysqld. The server will track the network connections from user client programs and manage their access to the data. The client on the other hand is a program installed locally on the end-user machine from which it will try to establish connection to the server.

5.3 AngularJS

AngularJS is a structural JavaScript based frontend framework dedicated for dynamic web application development (Lerner 2013, 8). While AngularJS and Bootstrap 3 frameworks are not directly tied to each other they can be used together in the same project as the frontend part of the application.

The AngularJS framework further extends the normal HTML markup syntax with directives and can bind data to HTML with expressions. AngularJS directives can be defined as markers directly attached to the DOM elements ordering these elements to follow the rules specified by the AngularJS's HTML compiler. AngularJS provides multiple built-in directives, prefixed with 'ng', such as ngBind, ngModel and ngClass. (Tutorialspoint 2017c.)

Different directives can be used to extend the basic functionality of DOM elements in multiple ways. The list of what AngularJS directives can do includes:

- Data binding: a synchronization mechanism between multiple components of the AngularJS data model
- Creation and management of DOM control structures for displaying and hiding DOM fragments
- Additional support for entry forms and form validation techniques
- Organization of individual HTML elements into reusable components

Compared to other JavaScript frameworks such as jQuery, the AngularJS framework works by separating the application business logic from the data presentation on the client side, similar to what PHP frameworks such as Codeigniter can do on the server-side of the application (Lerner 2013, 9). Therefore, the same logic that was used to describe the MVC architecture of backend frameworks can be applied here.

6 PRACTICAL IMPLEMENTATION

6.1 Concept and requirements

The first concept of this project was born in 2014 in cooperation with the LUAS placement coordinator and the finished application was supposed to eventually replace the currently used LUAS internship portal. The initial aim for this project was not fully achieved due to time restrictions and circumstances unrelated to the development of this application. Therefore the final version was produced as a demonstration of technologies that were used in the process with only partial product requirements being fulfilled.

The main requirement that was set during the first planning phase of the application was to change the previous version of the internship portal into something that meets modern web application development standards and to make it more suitable for long term improvements and support.

The rest of the requirements set by the placement coordinator can be divided into functional and non-functional requirements. Functional requirements are a list of features that specify what an application in question is supposed to do. Functional requirements include for example the business logic of the application and define the actions that the application users are allowed to perform. Some functional requirements that the internship portal was supposed to fulfill are listed below.

- Users are allowed to log into the application.
- Users are allowed to change their personal user information with the exception of their student number and group ID.
- Users are allowed to start a new practical activity if all previous activities have been complete.
- Users are allowed to send their internship activities to the placement coordinator (administrator) for her approval
- Users are not allowed to proceed until the placement coordinator gives her approval and permission to move to the next part

- Users are allowed to upload their internship-related documents such as reports and references and attach them to their current practical activity
- Users are allowed to confirm the status of their current internship at any time

Non-functional requirements describe an application's overall functionality rather than specific actions. Non-functional requirements that were set for this project are listed below:

- The application must work properly on all modern web browsers.
- The application must render properly on all modern devices and screen resolutions.
- The application must make use of all current web standards such as HTML 5 and CSS 3.
- The application structure must be intuitive and easy to understand.
- The application must be reasonably fast and responsive.
- The application must have at least minimal built-in security such as password encryption.
- The application must provide all necessary functionality for further support and improvements.

By default the application allows 4 groups of users: administrators, substance teachers, language teachers and general users or students. The administrator or placement coordinator is a role at the top of the user group hierarchy and is allowed to manage other user groups, keep track and approve their activities and utilize all unrestricted application functions. The administrator is also allowed to assign substance and language teachers. Substance and language teachers both have access to the activities of the students they are assigned to and can approve them for further evaluation. Finally users or students are the largest group. They are allowed to manage their own personal information, create and process new internship activities as well as keep track of their old internships. The application flow chart and possible interactions between different user groups can be seen in Appendix 1.

The process of implementation of this project can be separated into three main parts:

- Database and tables
- Background functionality
- Frontend and user interface

The MySQL database management system is responsible for storing and managing data in the application database. Background functionality was built using the Codeigniter 3 MVC framework. The frontend functionality and user interface were designed by using Bootstrap 3, AngularJS and jQuery frameworks in addition to normal HTML 5 and CSS elements. In following chapters the main parts of the application will be discussed in more detail.

6.2 Application database

The entity relationship diagram for the project database was designed using Microsoft Visio software. The database model can be seen in Figure 15.

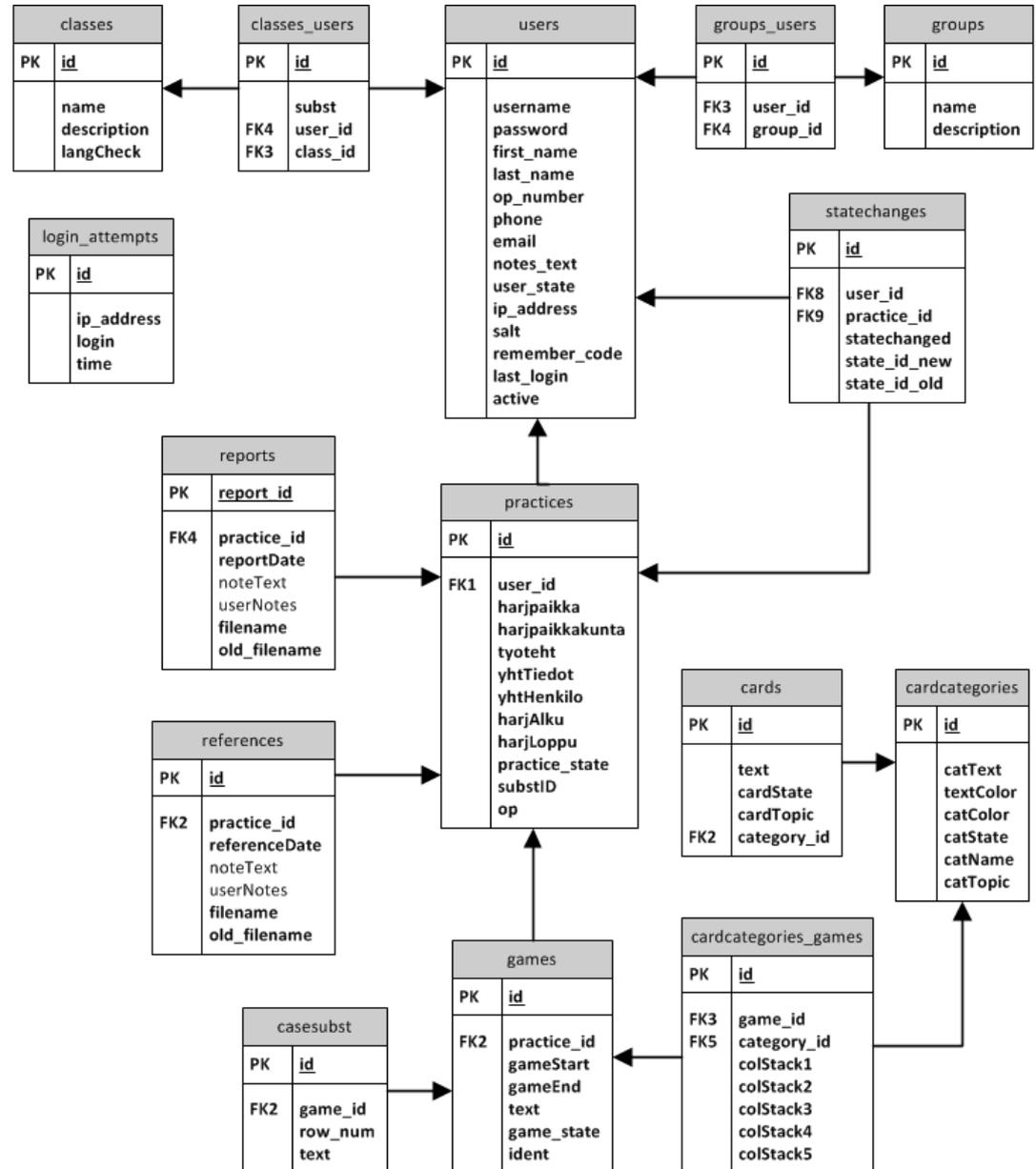


Figure 15. Database model

In addition to the fields shown in Figure 15, each relational table in the database contains two mandatory `TIMESTAMP` fields: `updated_at` and `created_at`. These fields are used by Codeigniter models to automatically keep track of the table creation and modification records.

Table 4 provides a brief description of all tables in the database and explains their role in the application structure according to the project's initial requirements.

Table 4. Database tables

Table name	Description
users	Contains personal information of each application user. Personal information includes user credentials, first and last name, student number, phone and email address. In addition, it keeps track of user's current status and previous login attempts.
groups	Stores and defines user groups. User groups include students, teachers and administrators.
groups_users	Intermediate table between groups and users. Shows which user group the user belongs to.
classes	Stores and defines user class information. Classes are used to organize users into groups based on their degree program.
classes_users	Intermediate table between students and classes. Shows which class the user attends. "Subst" is a Boolean value which checks if the user in question is assigned as a substance teacher for this class.
practices	Stores user internship related information. The fields include internship location and date, assignments and possible contact information.
statechanges	Intermediate table between users and practices. Stores data related to the internship state and keeps track of possible state changes.

reports	Shows all reports attached to the internship. The fields for this table include a date when the report was sent, its name and any additional information.
references	Similarly to the reports table, this table stores all references attached to the internship.
games	Defines the state of the card game (the list of goals the user would like to achieve) attached to the internship activity.
cards	Stores information about all cards that can be used in the card game.
cardcategories	Defines the categories that cards can be attached to.
cardcategories_games	Intermediate table between card categories and games. Stores information about all user decisions made through the card game.
casesubst	Stores additional internship related information (career goal list).
login_attempts	This table is required by Ion_Auth authorization library to keep track of user login attempts.

6.3 Installation and configuration

In order to start working with Codeigniter 3 a dedicated web server must be installed first. According to the Codeigniter 3 documentation (BCIT 2017m), the server should meet the following requirements:

- PHP version 5.6 or higher

- MySQL database version 5.1 or higher with built-in mysqli and PDO driver support

There are two web servers that were mainly used in this project: a local WAMP server and a LUAS student server. This thesis will focus on building the application using the local WAMP server.

A WAMP server is a small web development environment designed to quickly build and test various web applications on a local Windows machine. A WAMP server comes with the latest versions of PHP, Apache and MySQL DBMS already preinstalled and there is no need to install them manually. After downloading a WAMP server from its official web page, it can be installed on any modern Windows platform by following installation instructions.

After installation is complete the WAMP server will automatically create a new folder called “www” within its root directory. This folder can be used to store all PHP files and additional resources required for web application development. After downloading all Codeigniter 3 core package files it is enough to place them into the WAMP “www” folder. If everything is done correctly, a default Codeigniter page will be displayed by opening the web browser and typing in “localhost/codeigniter_folder_name” (Figure 16).

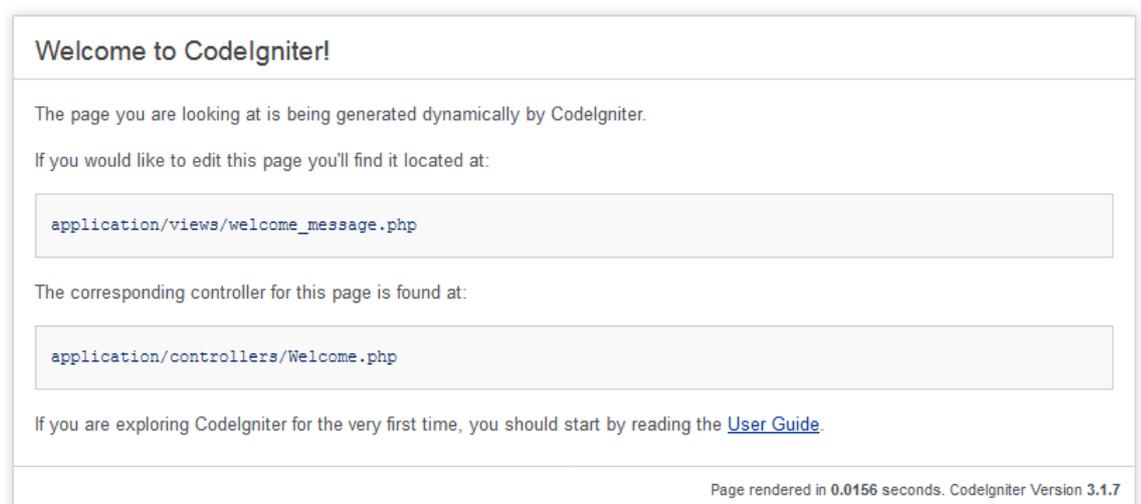


Figure 16. Codeigniter 3 Welcome page

If installation was successful, the Codeigniter folder can be renamed to correspond to the actual project name. It is also recommended to open the Codeigniter configuration file located in the application/config directory and manually set the base URL path of the project to the Codeigniter root directory.

Before getting started, the Codeigniter 3 framework must be configured for the first time. The minimal configuration process consists of connecting to a MySQL database, defining routes and routing rules for default controllers and rewriting the htaccess file. In addition, some models, helpers and libraries can be placed in the autoloader file to tell Codeigniter that these files should be initialized automatically.

The application database can be configured by opening the database configuration file and adding in connection values (Figure 17):

```
$db['default'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'practice',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => TRUE,
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

Figure 17. Codeigniter 3 database configuration

The values that are directly used to establish the database connection are hostname, username, password and the name of the database. The

additional value that may be important at this point is dbdriver, which specifies the type of driver and the database being used in the application. In the code above the database driver is set to “mysqli”, which means that the application uses MySQL DBMS.

By default Codeigniter will include an index.php line in all application URL requests. The .htaccess file can be modified to remove index.php part of the URL by applying additional rules (Figure 18). In order to modify the .htaccess file a mod_rewrite Apache Module inside the WAMP server must be enabled first. It can be enabled in Apache module configuration settings by selecting the rewrite_module option and turning it on.

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule .* index.php/$0 [PT,L]
```

Figure 18. Removing index.php from site URL

Finally, to start adding new custom controllers and to make them correspond to specific browser URL requests it is recommended to define the routing rules first. Routing rules in Codeigniter 3 are stored in the routes.php file. Some routing rules can be defined immediately; others will be extended eventually with an addition of new controllers. Figure 19 shows an example of the routing rules that were made for this project.

```
/* default routes */
$route['default_controller'] = "auth";
$route['404_override'] = 'error/_404';
$route['translate_uri_dashes'] = FALSE;

/* not default routes */

/* authentication */
$route['login'] = 'authorization/login';
$route['logout'] = 'authorization/logout';

/* information pages */
$route['index'] = 'info/index';
$route['faq'] = 'info/faq';

/* user and practice related forms */
$route['userinfo'] = 'userdata/personal';
```

Figure 19. Codeigniter 3 routing rules

As can be seen in the figure above the Codeigniter will define auth/index as the default controller/method of the application and maintain custom routing rules for additional application controllers and their methods.

6.4 MVC components

6.4.1 Controllers

Controllers in Codeigniter are basically user-defined classes that can be called by sending an appropriate HTTP request. The request should contain a specific URI that the controller in question is associated with. All controller classes in Codeigniter 3 must start with an uppercase character, though their respective URIs are usually case insensitive and can be written in lowercase. (BCIT 2017k.)

The most basic controller class in Codeigniter can be seen in Figure 20:

```
/* controller */  
class Controller extends CI_Controller {  
    public function __construct() {  
        parent::__construct();  
    }  
}
```

Figure 20. Basic controller class

All new user-defined controller classes must have a parent controller class specified that they will extend in order to inherit their non-private methods, with a Codeigniter core controller class called “CI_Controller” being at the top of the class hierarchy. Codeigniter 3 allows developers to make their own core controller classes, which will extend the native CI_Controller class functionality. The new core controller class must be named “MY_Controller” and placed in the application/core folder. This new core controller class can then be used to add new functionality or extend some of the existing methods of the CI_Controller class. (BCIT 2017k.)

Any user-defined controller class in Codeigniter must contain a constructor method. This constructor will be used to call a parent class constructor method and load specific libraries, helpers, models and other items that will be globally accessible by all methods inside the controller class.

Figure 21 demonstrates a part of the controller class dedicated to work with incoming user data in this project:

```

class Userdata extends CI_Controller {

    function __construct() {
        parent::__construct();
        $this->load->database();
        $this->load->library(array('ion_auth',
            'form_validation', 'session'));
        $this->load->helper(array('url', 'language', 'form'));
        $this->load->model(array('users_model', 'groups_model',
            'classes_model', 'practices_model'));
        $this->form_validation->set_error_delimiters($this->
            config->item('error_start_delimiter', 'ion_auth'),
            $this->config->item('error_end_delimiter', 'ion_auth'));
        $this->lang->load('auth');
        if (!$this->ion_auth->logged_in()) {
            //redirect them to the login page
            redirect('auth/login', 'refresh');
        }
    }
}

```

Figure 21. Controller initialization

As is shown in this code, the controller class is defined as Userdata and extends the functionality of the CI_Controller core class. It will then execute the parent constructor method, establish connection with the application database and globally load all required libraries, helpers and models. Additionally, it will check if the user is currently logged in. If not, the controller will redirect the user back to the login page.

The body of each application controller contains one or more methods, used to load application views, access and pass data to the models and work with different helpers and libraries. Any public method written for the controller class can be passed through the URI request as a second segment following the controller name unless this behavior was changed in routing rules. If URI has more than two segments, further segments will be passed to the selected controller method as additional parameters. Private controller methods are only available inside their representative classes and cannot be inherited or accessed via URI requests. (BCIT 2017k)

To start adding new controllers in Codeigniter 3 projects it is enough to create a separate PHP class for each of them inside the application/controllers folder. The application/controllers folder also permits the creation of sub-directories. Sub-directories can be used to hierarchically

organize different controllers depending on their roles inside the application. Figure 22 demonstrates the content of the application/controllers folder, which contains a list of controllers that were made for this project.

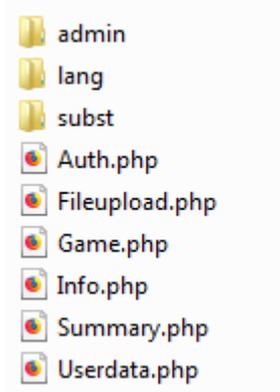


Figure 22. Application controllers

Each controller presented in the list above responds to a specific part of the application. Auth-controller is responsible for user authentication and login functionality. It defines, organizes and utilizes all models and views necessary to build and maintain the authorization system of the application. Info-controller manages a secondary set of views unrelated to the user internship or personal information. These views include i.e. application FAQ and Index pages. Userdata-controller is responsible for managing users, user groups, user classes and any sensitive personal user data. Game-controller is the main application controller class that manages views and models related to the user internship activities. Fileupload is a secondary controller class connected to the Game controller and is used to upload report and reference files to the web server. Additional folders are intended to store controllers for administrator, language and substance teacher specific tasks.

6.4.2 Models

Each model class created for this project refers to the particular table in the database which data this model interacts with. Each model class name contains the name of the table the model refers to followed by “_model”

suffix. Figure 23 demonstrates the list of all models that were created for the application.

```
Cardcategories_model.php
Cards_model.php
Casesubst_model.php
Classes_model.php
Example_model.php
Games_model.php
Groups_model.php
Ion_auth_model.php
Practices_model.php
Refers_model.php
Reports_model.php
Statechanges_model.php
Users_model.php
```

Figure 23. Application models

Model class names in Codeigniter must have their first letter in an upper-case with the rest of the name being lowercase and the file name of the model must also match their class name (BCIT 2017f).

The model class core structure is mostly similar to that of the controller. Model classes are located in the application/models folder of the Codeigniter root directory. The folder also accepts creation of sub-directories for better file organization. Figure 24 demonstrates a code for the most basic model class in Codeigniter 3:

```
/* Model */
class Model extends CI_Model {
    public function __construct() {
        parent::__construct();
    }
}
```

Figure 24. Basic model class

As is shown in the figure above, a new model class can be initialized by defining its class name and extending Codeigniter core model class called "CI_Model". In order to load existing databases and other necessary

resources it must also contain a class specific constructor, which will be used to execute the parent class constructor method.

To add new database queries to the models in Codeigniter, a Query Builder class is used. The Query Builder class contains different methods that copy the functionality of SQL queries and allows for the manipulation of data stored in the database. Query Builder methods can mimic normal SQL statements such as SELECT, INSERT, UPDATE and DELETE, specific conditions such as WHERE and LIKE and different data grouping, ordering and limiting functions. An example of a SELECT statement in Codeigniter 3 with the corresponding Query Builder method that will return user's first and last names from the "users" table can be seen in Figure 25.

```
$this->db->select('first_name, last_name');  
$query = $this->db->get('users');
```

Figure 25. SELECT in Query Builder

In addition to Query Builder basic functionality, Codeigniter allows developers to add their own query building methods by extending application core model class. To apply a new model class that will extend the Codeigniter core model the model class must be named MY_Model and placed in the application/core folder of the Codeigniter framework root directory. (BCIT 2017f.)

In this project it was decided to extend the Codeigniter core model class by using the MY_Model class provided by Adrian Voicu in his GitHub project. This model extension adds full CRUD base support for all database interactions, an event-based observer system, intelligent table name guessing and the option for soft delete (Voicu 2015). It does not fully replace Query Builder class functionality but makes query building for smaller queries within model classes faster and, to some extent, more logical.

To add MY_Model functions into a normal model class it has to be defined as an extension of the MY_Model class first. The new model class that will extend MY_Model can be defined as follows (Figure 26):

```

class Users_model extends MY_Model {

    public $table = 'users'; // table name
    public $primary_key = 'id'; // primary key of the table
    public $fillable = array();
    public $protected = array();

    function __construct() {

        parent::__construct();
    }
}

```

Figure 26. Defining a new model

MY_Model also allows changing model settings inside the constructor method before calling parent constructor. The settings can be used to establish separate database connection for each particular model, enable and disable some built-in functions such as timestamp and soft delete and set different return values for model query results.

MY_Model also allows establishing relationships between the current application table/model and any tables/model that is related to it. The relationships include ONE TO ONE, ONE TO MANY and MANY TO MANY and are similar to the normal MySQL database table interactions. An example of relationships between the “practices” model and models that are related to it can be seen in Figure 27.

```

//$this->has_one['...'] allows establishing ONE TO ONE
//or more ONE TO ONE relationship(s) between models/tables
$this->has_one['users'] = 'Users_model';
$this->has_one['games'] = 'Games_model';

// $this->has_many['...'] allows establishing ONE TO MANY
// or more ONE TO MANY relationship(s) between models/tables
//$this->has_many['statechanges'] = 'Statechanges_model';
$this->has_many['reports'] = 'Reports_model';
$this->has_many['refers'] = 'Refers_model';

```

Figure 27. Practices_model relationships

The code part provided in Figure 28 shows an example of the methods used to interact with the application database in Practices_model after the MY_Model class was extended.

```
function insert_practice($data) {
    return $this->practices_model->insert($data);
}

function remove_practice_by_id($id) {
    return $this->practices_model->
        where('id', $id)->delete();
}

function remove_practice_by_userid($userid) {
    return $this->practices_model->
        where('user_id', $userid)->delete();
}

function update_practice_by_id($data, $id) {
    return $this->practices_model->
        where('id', $id)->update($data);
}

function update_practice_by_userid($data, $userid) {
    return $this->practices_model->
        where('user_id', $userid)->update($data);
}
```

Figure 28. Query building methods

The first method will insert values into the table, passing data-parameter as an object. The second and third methods are used to delete specific data existing in the table similar to what SQL's DELETE statement can do. They use the "where" method to specify which table rows will be affected and the "delete" method to send the query. The last two methods are used to update database tables in the same fashion. The query result will be passed back to the model in the form of an object that can later be processed by the application controllers. The other application model classes for this project are built using similar query building methods.

6.4.3 Views

View files in Codeigniter 3 are stored in the application/views directory. To load a particular view inside the controller a “load->view (‘name’)” method can be used where the ‘name’ string represents the name of the view (BCIT 2017I). If multiple view calls inside the controller method are written in direct order, Codeigniter 3 will consider them being connected and render them together in order of appearance starting from the top view (Figure 29):

```
public function index() {  
    $this->load->view('layout/include/header');  
    $this->load->view('home/index', $data);  
    $this->load->view('layout/include/footer');  
}
```

Figure 29. Combination of views

In the example above Codeigniter will first load a header view, followed by an index.php file and footer view at the bottom of the web page. A \$data parameter inside the second method is a dynamically added data type such as an array or an object that can be passed from the controller to the view as an additional parameter. This data can be later accessed from the view file directly.

The content of the views folder created for this application can be seen in Figure 30:

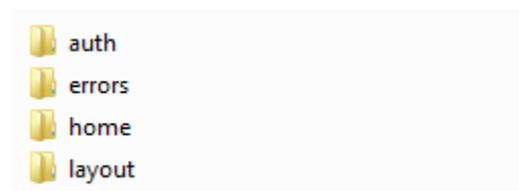


Figure 30. Application views

The auth folder contains views that are intended for user authentication. This folder includes views for many user authentication-related tasks such as user creation and deactivation, user password change or reset, and others.

The errors folder stores views that will be displayed when specific application error has occurred. The list of application errors in Codeigniter includes database errors, error 404, errors shown by the exception handler as well as general and PHP specific errors. By default, Codeigniter 3 will display all application errors and keep log files for all occurring errors. This behavior can also be changed by using Codeigniter's error handling class methods.

Home is the main directory that keeps all regular views utilized by application controllers. This folder contains views for each application part, such as index, FAQ and login page, as well as specific views related to internship activities. Figure 31 demonstrates the content of a home view sub-folder which stores views connected to the user's internship activity:

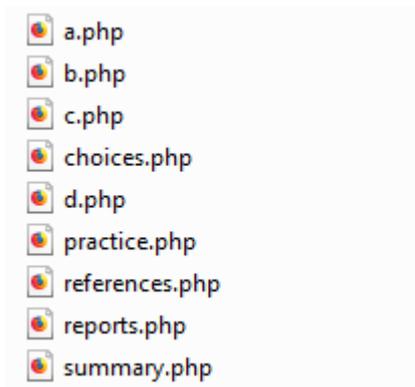


Figure 31. Practice activity view list

The last folder contains layout specific views, such as headers and footers shared between multiple application pages through controllers to avoid redundancy in code. Header views contain for example application Meta tags and the list of CSS and JS files that will be used by other views. This list also includes Bootstrap 3, AngularJS and jQuery framework files as well as custom fonts and page specific files. Additionally, header views contain a Bootstrap 3 navigation bar, which will be displayed at the top of the page as is demonstrated in Figure 32.

FAQ Subheading

[Home](#) / [FAQ](#)

Figure 32. Header view and navigation bar

The content of a regular view's body is mostly similar to that of a normal HTML file. It may contain regular HTML and CSS elements, jQuery scripts and PHP code pieces as well as unique components provided by frontend frameworks and libraries. The code below (Figure 33) demonstrates a part of the header view file from above responsible for rendering this navigation bar.

```

</div>
<!--Collect links,forms and other content for toggling-->
<div class="collapse navbar-collapse"
  id="bs-example-navbar-collapse-1">
  <ul class="nav navbar-nav navbar-right">
    <li class="<?php
      if ($this->uri->segment(1) == "game" ||
        $this->uri->segment(1) == "practice" ||
        $this->uri->segment(1) == "a" ||
        $this->uri->segment(1) == "b" ||
        $this->uri->segment(1) == "c" ||
        $this->uri->segment(1) == "d" ||
        $this->uri->segment(1) == "choices" ||
        $this->uri->segment(1) == "reports" ||
        $this->uri->segment(1) == "references" ||
        $this->uri->segment(1) == "summary") {
          echo "active";
        }
      ?>">
      <a href="game">Practice</a>
    </li>
  </ul>
</div>

```

Figure 33. Navigation bar view code

This particular code segment uses the combination of regular HTML tags, Bootstrap 3 tags and PHP code to confirm if any of the views from the

internship activity list is active. In the PHP part it uses the Codeigniter 3 URI class to retrieve information from URI string and compare its first segment to the name of a view from the internship activity list.

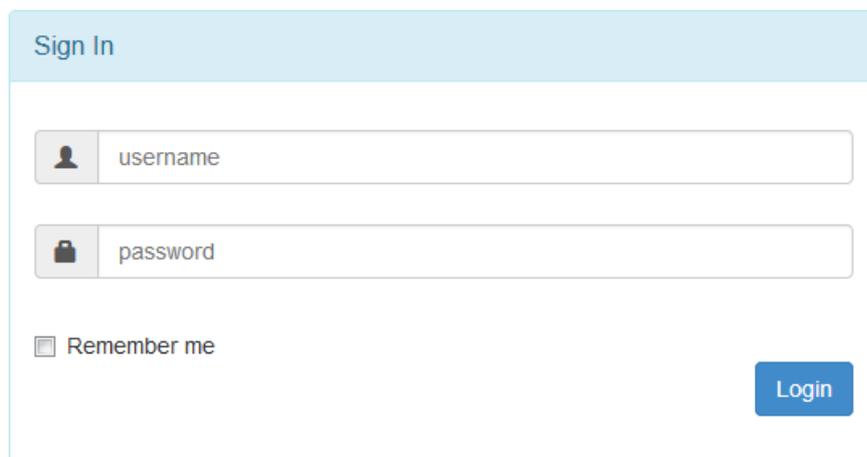
6.5 Application overview

This section briefly reviews some of the main parts of the application combining descriptions of server-side functionality and user interface decisions.

6.5.1 Login page

The login page of this project was built using the Codeigniter Ion Auth library. The default controller and model classes and views for the Ion Auth library were created by Ben Edmunds as a part of his Ion_Auth authentication system GitHub project (Edmunds 2015). Some of them were also manually extended or adjusted for this project.

Figure 34 shows a login window that will ask users to enter their personal, case sensitive login credentials in order to access the application. The Ion Auth uses password hashing methods to keep user passwords protected.



The image shows a 'Sign In' form with a light blue header. It contains two input fields: 'username' (with a person icon) and 'password' (with a lock icon). Below the password field is a checkbox labeled 'Remember me'. A blue 'Login' button is positioned at the bottom right of the form.

Figure 34. Login window

The system was supposed to transfer user personal information, including LUAS authentication credentials into the application database directly using some sort of third party functionality that was out of the scope of this project's initial aims. Therefore the system provides no option for new users to register or change their login data manually. However it does keep track of all registered users and allows them to access the application as long as their authentication data is valid and exists in the database. A part of the login method for the auth controller is shown in Figure 35:

```

$this->data['title'] = "Login";
//validate form input
$this->form_validation->set_rules('identity', 'Identity', 'required');
$this->form_validation->set_rules('password', 'Password', 'required');

if ($this->form_validation->run() == true) {
    //check for "remember me"
    $remember = (bool) $this->input->post('remember');
    if ($this->ion_auth->login($this->input->post('identity'),
        $this->input->post('password'), $remember)) {
        //if the login is successful
        $this->session->set_flashdata('message',
            $this->ion_auth->messages());
        redirect('/', 'refresh');
    } else {
        //if the login is un-successful
        $this->session->set_flashdata('message',
            $this->ion_auth->errors());
        redirect('auth/login', 'refresh');
    }
}
}

```

Figure 35. Login method

The login method first sets both login form fields as mandatory and checks if they both contain user credential data. It will then confirm that both the user name and the password are valid and add session data using Codeigniter's Session class "set_flashdata" method accordingly. If credentials are valid, the user will be redirected back to the auth/index page and from there to the internship portal main page.

6.5.2 User personal information

This section is available for application users immediately after they are logged in and lets them confirm and correct their personal information with the exception of their degree program and student number.

Index Harjoittelu **Omat Tiedot** FAQ Logout

Info Subheading

Home / Info

Omat tiedot

Omat Tiedot:

Etunimi test

Sukunimi test

Sähköposti test@mail.com

Puhelinnumero 434

Opiskelijanumero 333

Ryhmä

Talleta

© Lahti University of Applied Sciences 2015

Figure 36. User personal information

Figure 36 shows an example of the personal information page for a user currently logged in. The page view uses Bootstrap 3 form control styles to make forms look consistent between different devices and screen resolutions. Both student number and user class fields are disabled allowing the user to confirm their content but not interact with them.

The Userdata controller class is responsible for managing the content of this personal information view. Figure 37 shows the part of its main method, which sets validation rules for each field, retrieves current user data from the database and sends it to the view for the user to confirm. If the data was updated by the user according to the form validation rules, the controller will

update the users-table with newly obtained user data. Similar methods are also used for managing other application view forms.

```

$data['userinfo'] = $this->users_model->
    get_userinfo_by_id($user['userid']);
$data['usergroup'] = $this->users_model->
    get_usergroup_by_id($user['userid']);
//set validation rules
$this->form_validation->
    set_rules('first', 'Firstname',
        'trim|required|callback_alpha_only_space');
$this->form_validation->
    set_rules('last', 'Lastname',
        'trim|required|callback_alpha_only_space');
$this->form_validation->
    set_rules('email', 'Email', 'trim|required|valid_email');
$this->form_validation->
    set_rules('phone', 'Phone', 'trim|required|numeric');
if ($this->form_validation->run() == FALSE) {
    //fail validation
    $this->load->view('layout/include/header');
    $this->load->view('home/userinfo', $data);
    $this->load->view('layout/include/footer');
} else {
    //pass validation
    $userdata = array(
        'first_name' => $this->input->post('first'),
        'last_name' => $this->input->post('last'),
        'email' => $this->input->post('email'),
        'phone' => $this->input->post('phone')
    );
    //update record
    $this->users_model->
        update_user_by_username($userdata, $user['username'] );
    redirect ('userinfo');
}

```

Figure 37. Userdata controller

Form validation rules shown in this code are part of a Codeigniter's Form Validation class and are used to restrict the content that the form field will accept from the user. The "set_rules" method used to define the form validation rules takes three main parameters: the field name it works with, the name that it will show to the user in the error message text, and the validation rule list. For example the "valid_email" validation rule means that the field must contain a valid email address.

The Form Validation class also allows declaring custom validation rules, such as “callback_alpha_only_space“, as is shown in the code above. This validation rule will call a separate “alpha_only_space” method for user name validation. It will also pass any existing form field data as a variable that the callback method will be able to process.

The “alpha_only_space” method in turn uses regular expressions to restrict the validation function to accept only letters and space inputs as is shown in Figure 38.

```
function alpha_only_space($str) {  
    if (!preg_match("/^([-a-z ])+$/i", $str)) {  
        $this->form_validation->set_message('alpha_only_space',  
            'The %s field must contain only alphabets or spaces');  
        return FALSE;  
    } else {  
        return TRUE;  
    }  
}
```

Figure 38. Custom validation rule

6.5.3 Internship navigation list

This is the main page of the application that displays the list of active and finished internships. Currently it only displays the last active internship for each user, but keeps track of all finished internships in the application database. An active internship navigation window is demonstrated in Figure 39:

Active:

▼ 2. Harj. jakso
▼ Suunnitelu
Perustiedot
➤ Yleistavoitteet
Amm. tavoitteet
Yhteenvedo
Raportti
Työtodistukset

Figure 39. Internship navigation list

This navigation window utilizes Bootstrap 3 list-group components to display its content to the user. It allows creation of more complex navigation lists with custom HTML components such as glyphs nested within these lists in any given order. The part of the view used to display the internship navigation window can be seen in Figure 40.

```

<a href="#item-2-1" class="list-group-item" data-toggle="collapse">
  <i class="glyphicon glyphicon-chevron-down"></i>Suunnitelu
</a>
<div class="list-group collapse in" id="item-2-1">
  <a href="practice" class="list-group-item">Perustiedot</a>
  <a href="#item-2-1-1" class="list-group-item" data-toggle="collapse">
    <i class="glyphicon glyphicon-chevron-down"></i>Yleistavoitteet
  </a>

```

Figure 40. List group

Navigation list provides access to multiple parts of the application related to the internship activities. These parts include the internship plan and information page, the main goals page also known as a card game, the

additional career goals page, a summary and the file upload page for reports and references.

6.5.4 Internship plan and information

Functionally this section is similar to the user personal information section and is used to check and manage Practices_model data. Users must confirm their internship status and information here before they are able to continue with the next part. Additionally, all internship-related data was supposed to be passed down to the placement coordinator in order to approve or reject it and only allow the user to proceed to the next section if the previous one was approved. This functionality, however, is not currently implemented.

This section uses the Bootstrap 3 Date and Time picker form component to handle the starting and finishing dates of the internship. The interface of the internship plan view can be seen in Figure 41.

Figure 41. Internship plan and information page

6.5.5 Internship goals

This part of the application, accessible through multiple sections of the active internship list, is dedicated for users to define their main goals and set priorities during the course of their current internship. The main idea was to provide users with a number of so called cards that they would be able to place freely into a few given categories in order of importance starting from the highest. The data will then be stored in the application database and attached to the current active internship list.

The user interface for one of the four currently implemented internship goal sections can be seen in Figure 42.

Oppiminen Subheading

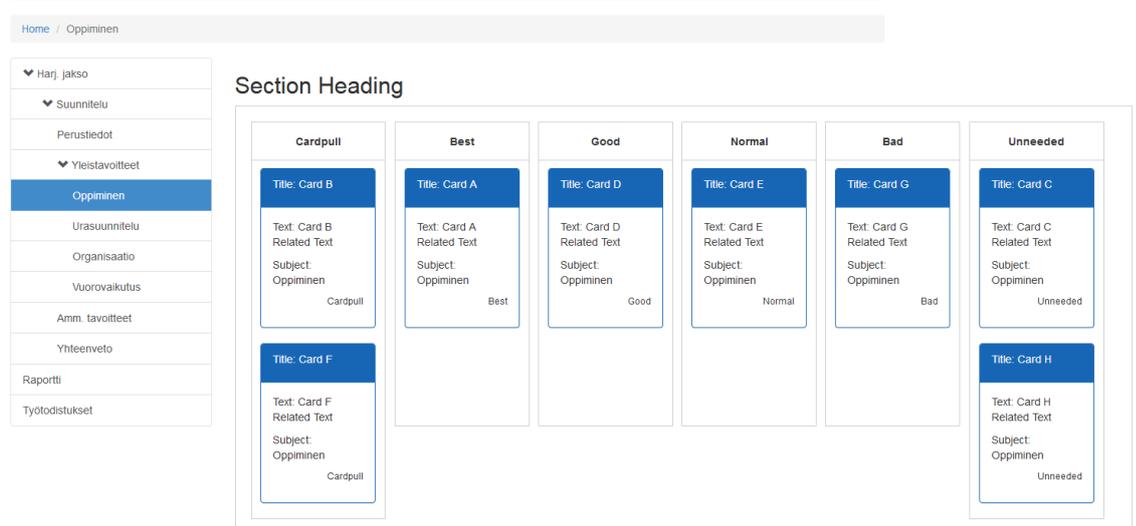


Figure 42. Card game page

Cards for this section are located in the “cards” table of the database. The “cardcategories” table determines the category in which each particular card belongs. Each internship goal page of the application uses its own card category with a number of cards attached to it. There is currently no built-in controller functionality to add or remove cards from the game or switch them between different categories, though the models for this section contain all necessary database queries to make it possible in the future.

In addition to Bootstrap 3, the view files for this section use the AngularJS framework with a ngDraggable drag-and-drop module connected to it. This method allows free movement of selected cards between columns. The logic for this section works as follows:

1. The Codeigniter game-controller method renders all necessary views for the page according to the HTTP request.
2. The loaded AngularJS module makes a separate server request with http.get to the game-controller (see Appendix 2).
3. The requested controller method gathers all card game data through models.
4. The controller method sends the data to the AngularJS module in a form of a JSON object.
5. The AngularJS module processes received data and fills the list with it.

The controller will send the data from the model to the AngularJS module in the form of a JSON object as is shown in Figure 43. The data from the database is gathered based on the category ID of selected cards.

```

▼ cards:
  ▼ 0:
    id: "1"
    category_id: "1"
    cardTitle: "cardA"
    cardText: "random text for card A"
    cardState: "1"
    cardTopic: "OP1"
    created_at: "2016-02-02 01:07:26"
    updated_at: "0000-00-00 00:00:00"

```

Figure 43. Card as a JSON object

Figure 44 displays a view code responsible for rendering cards in the first column of the list presented in Figure 42. On each successful drag or drop action set by ng-drag-success and ng-drop-success directives, AngularJS will launch a respective function from its module. It will also repeatedly redraw each object (card) for a given section according to the ng-repeat

directive and fill it with the data provided by the game controller. This process will repeat itself for each card in the list.

```

<div class="cardWrapper" ng-drop="true"
  ng-drop-success="onDropComplete1($data,$event)">
  <div class="cardcat"><p>{{categories[0]}}</p></div>
  <div ng-repeat="obj in droppedObjects1"
    ng-drag="true" ng-drag-data="obj"
    ng-drag-success="onDragSuccess1($data,$event)"
    ng-if="obj.category == 'cardpull'"
    ng-click="getData(obj)" class="{{obj.subject}}">
    <div class="portlet">
      <div class="panel panel-primary">
        <div class="panel-heading">
          <div class="portlet-header">
            <p class="title">title: {{obj.title}}</p>
          </div>
        </div>
        <div class="panel-body">
          <div class="portlet-content">
            <p class="text">text: {{obj.text}}</p>
            <p class="subject">subject: {{obj.subject}}</p>
            <p class="catRight">{{obj.category}}</p>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figure 44. Cardpull code

6.5.6 Career goals

This section is tied to the internship main goal section and allows users to list their career goals and other self-assessment information. The page presents up to ten flexible textarea elements allowing users to freely write down their notes in any given order. The career goal view UI in the browser can be seen in Figure 45:

▼ Yleistavoitteet	1.	Do this and that
Oppiminen	2.	And maybe that too
Urasuunnittelu	3.	And don't forget about this one
Organisaatio	4.	"Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro
Vuorovaikutus		
Amm. tavoitteet		
Yhteenveto		
Raportti		
Työtodistukset		

Figure 45. Career goals page

The controller method for this section sets up to five form fields as required in form validation rules. After that, if they are filled correctly by the user, it will insert them directly into the database. Alternatively, if the data already exists in the database, the method will update casesubst-table for the last active internship. The code used by the controller to insert new lines or update an already existing career goal list is shown in Figure 46:

```

if (!$data['choices']) {
    for ($i = 1; $i <= 10; $i++) {
        $casesubstdata = array(
            'game_id' => $game['id'],
            'row_num' => '' . $i,
            'text' => $this->input->post('' . $i),
        );
        $this->casesubst_model->
            insert_casesubst($casesubstdata, $i);
    }
} else {
    for ($i = 1; $i <= 10; $i++) {
        $casesubstdata = array(
            'row_num' => '' . $i,
            'text' => $this->input->post('' . $i),
        );
        $this->casesubst_model->
            update_casesubst_by_gameid($casesubstdata, $game['id'], $i);
    }
}
//update record
$this->load->view('layout/include/game_header');
$this->load->view('home/game/choices', $data);
$this->load->view('layout/include/footer');

```

Figure 46. Insert or update career goals

6.5.7 Reports and references

These two functionally identical sections are dedicated for the users to upload their report and reference files to the server. The application utilizes the Blueimp jQuery File Upload plugin and its UploadHandler PHP file as a third party library in Codeigniter 3. Figure 47 displays the user interface for the report upload page.

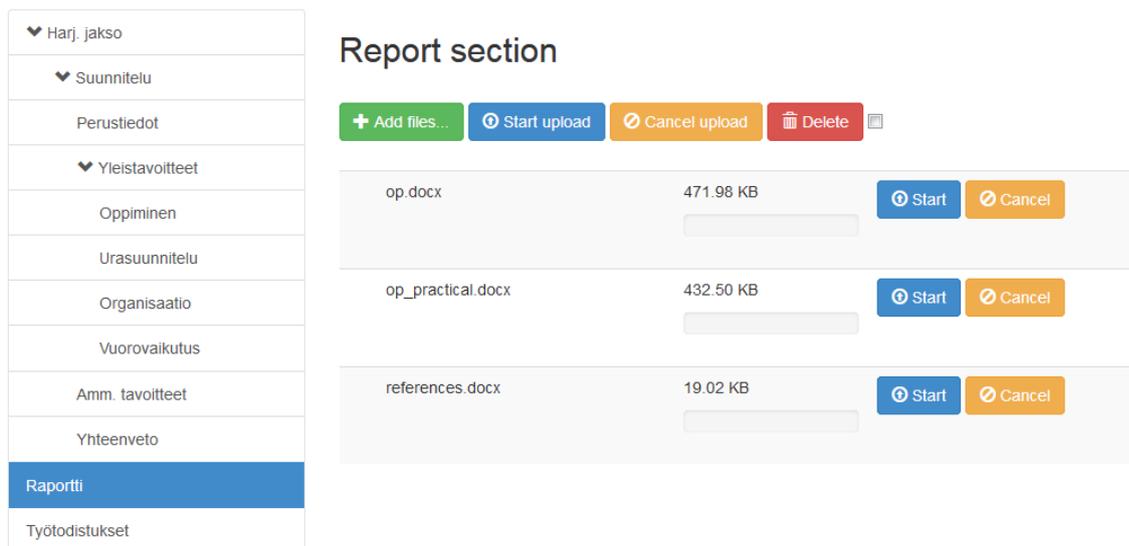


Figure 47. Report page

The controller for this view uses two different methods to store reports and references into two separate folders. To make the File Upload plugin work in Codeigniter it is necessary to set up the jQuery File Upload widget first by sending it the URL link to the method responsible for initializing the UploadHandler library (Figure 48). The method itself must also contain the path to the folder where all uploaded files will be stored and pass it to the UploadHandler library as a second parameter.

```
$options = [
    'script_url' => site_url('practice/upload'),
    'upload_dir' => APPPATH . '../uploads/reports/'
];
$this->load->library("UploadHandler", $options);
```

Figure 48. UploadHandler initialization

7 CONCLUSION

The initial aim set for this project was the development of an internship portal application able to provide users with necessary tools to control and manage their internship activities. The process of building the application described in this thesis could be loosely divided into four main phases:

- sorting client requirements into functional and non-functional requirement lists
- planning the application based on the analysis of the client requirements
- building the application database
- writing backend functionality and frontend user interface for the application, using appropriate tools and frameworks

While not being able to fulfill the entire list of functional requirements set for this project the thesis was able to successfully achieve its main goal: demonstrate the process of creation of a responsive and mobile friendly web application with a strict distinction between its backend and frontend technologies. By using the Model-View-Controller pattern as a starting ground for backend development and following its core principles it succeeded in building the application with a well-structured and easily extensible list of MVC components. The thesis therefore was also able to successfully demonstrate the advantages of building dynamic web applications using the Codeigniter 3 MVC framework.

Working on this project I faced several challenges. The most notable one was probably the scale and the complexity of the project, which I was not able to fully comprehend at the initial stages of planning and development. Judging it now retrospectively, the project would probably have required a dedicated team of developers with a strict distribution of tasks to make it in time. The second problem arising from the first was my personal lack of knowledge and practical experience when it comes to building applications of that scale.

Thinking of framework selection, I am glad I decided to use Codeigniter 3 and Bootstrap. The amount of useful information and community support made it much easier to understand and properly utilize some of the best aspects of MVC architecture, responsive UI design and mobile friendly application development. If I would consider changing the MVC framework right now I would probably go with Laravel instead, but back in 2015 when this project was initially developed I think Codeigniter 3 was a solid choice.

Overall, I think that this project, while not being completely successful in achieving its initial goals, provided me, and hopefully readers of this thesis, with some valuable experience how to build dynamic applications by utilizing the MVC pattern.

LIST OF REFERENCES

Published references

Bhaumik, S. 2015. Bootstrap Essentials. Birmingham: Packt Publishing LTD.

Griffiths, A. 2010. Codeigniter 1.7 Professional Development. Birmingham: Packt Publishing LTD.

Lerner, A. 2013. Ng-book – The Complete Book on AngularJS. Fullstack io.

Pitt, C. 2012. Pro PHP MVC. Apress.

Electronic references

1&1 2017. Codeigniter – The Lightweight of the PHP Frameworks [accessed 1 November 2017]. Available at:
<https://www.1and1.com/digitalguide/websites/web-development/codeigniter-the-lean-php-framework/>

Bootstrap Blog 2013. Bootstrap 3 released [accessed 21 November 2017]. Available at: <http://blog.getbootstrap.com/2013/08/19/bootstrap-3-released/>

Bradley, S. 2011. How To Create Flexible Images And Media In CSS Layouts. Vanseodesign [accessed 28 October 2017]. Available at: <http://vanseodesign.com/css/flexible-images/>

British Columbia Institute of Technology (BCIT). 2017. Codeigniter User Guide. Codeigniter [accessed 28 October 2017]. Available at: https://www.codeigniter.com/user_guide/

British Columbia Institute of Technology (BCIT). 2017a. Installation instructions. Codeigniter [accessed 28 October 2017]. Available at: https://www.codeigniter.com/user_guide/installation/index.html

British Columbia Institute of Technology (BCIT). 2017b. Application Flow Chart. Codeigniter [accessed 28 October 2017]. Available at: https://www.codeigniter.com/user_guide/overview/appflow.html

British Columbia Institute of Technology (BCIT). 2017c. URI Routing.

Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/routing.html

British Columbia Institute of Technology (BCIT). 2017d. Web Page Caching.

Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/caching.html

British Columbia Institute of Technology (BCIT). 2017e. Security.

Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/security.html

British Columbia Institute of Technology (BCIT). 2017f. Models. Codeigniter

[accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/models.html

British Columbia Institute of Technology (BCIT). 2017g. Helper Functions.

Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/helpers.html

British Columbia Institute of Technology (BCIT). 2017h. Creating Libraries.

Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/creating_libraries.html

British Columbia Institute of Technology (BCIT). 2017i. Using Codeigniter

Drivers. Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/drivers.html

British Columbia Institute of Technology (BCIT). 2017j. Hooks – Extending the Framework Core. Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/hooks.html

British Columbia Institute of Technology (BCIT). 2017k. Controllers.

Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/controllers.html

British Columbia Institute of Technology (BCIT). 2017l. Views. Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/views.html

British Columbia Institute of Technology (BCIT). 2017m. Server requirements. Codeigniter [accessed 28 October 2017]. Available at:

https://www.codeigniter.com/user_guide/general/requirements.html

Codeigniter. 2010. Codeigniter in 2011: Reactor, Core and UserVoice. Facebook [accessed 28 October 2017]. Available at:

<https://www.facebook.com/notes/codeigniter/codeigniter-in-2011-reactor-core-uservice/10150096079313760/>

De Graeve, K. 2011. HTML5 – Responsive Web Design. Microsoft Developer Network [accessed 24 October 2017]. Available at:

<https://msdn.microsoft.com/en-us/magazine/hh653584.aspx>

Edmunds, B. 2015. Codeigniter-Ion-Auth. GitHub [accessed 12 June 2015].

Available at: <https://github.com/benedmunds/CodeIgniter-Ion-Auth>

EllisLab. 2017. About Codeigniter. EllisLab, Inc. [accessed 28 October 2017]. Available at: <https://ellislab.com/codeigniter>

Garbade, M. 2016. How to choose a PHP framework. Opensource.com [accessed 28 October 2017]. Available at:

<https://opensource.com/business/16/6/which-php-framework-right-you>

Open Web Application Security Project (OWASP). 2016. SQL Injection [accessed 15 November 2017]. Available at:

https://www.owasp.org/index.php/SQL_Injection

Open Web Application Security Project (OWASP). 2017a. Cross-site Scripting (XSS) [accessed 15 November 2017]. Available at:

https://www.owasp.org/index.php/SQL_Injection

Open Web Application Security Project (OWASP). 2017b. Cross-Site Request Forgery (CSRF) [accessed 15 November 2017]. Available at: [https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))

Oracle Corporation. 2017. MySQL: Reference Manual [accessed 7 December 2017]. Available at: <https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html>

Otto, M. & Thornton, J. 2017. Bootstrap: Getting started. Bootstrap [accessed 28 October 2017]. Available at: <https://getbootstrap.com/docs/3.3/getting-started/>

PHPTeam. 2016. New Features in Codeigniter 3. TechNetExperts [accessed 28 October 2017]. Available at: <http://www.technetexperts.com/web/new-features-in-codeigniter-3/>

Skvorc, B. 2015. The best PHP Framework for 2015: SitePoint Survey Results. SitePoint [accessed 7 December 2017]. Available at: <https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/>

Tutorial Republic. 2017. Creating Forms with Bootstrap [accessed 1 November 2017]. Available at: <https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-forms.php>

Tutorialspoint 2017a. MVC Framework – Introduction. Tutorialspoint [accessed 28 October 2017]. Available at: http://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

Tutorialspoint 2017b. Codeigniter – Application Architecture. Tutorialspoint [accessed 15 November 2017]. Available at: https://www.tutorialspoint.com/codeigniter/codeigniter_application_architecture.htm

Tutorialpoint 2017c. AngularJS Tutorial. Tutorialspoint [accessed 7 December 2017]. Available at:

https://www.tutorialspoint.com/angularjs/angularjs_overview.htm

Utterback, B. 2014. What is Bootstrap? – The History and the Hype. PrestaShop [accessed 21 November 2017]. Available at:

<https://www.prestashop.com/en/blog/what-is-bootstrap>

Voicu, A. 2015. Codeigniter-MY_Model. GitHub [accessed 12 June 2015].

Available at: https://github.com/avenirer/CodeIgniter-MY_Model

W3S 2017a. Bootstrap Grids. W3Schools.com [accessed 21 November 2017]. Available at:

https://www.w3schools.com/bootstrap/bootstrap_grid_basic.asp

W3S 2017b. HTML5 Tutorial. W3Schools.com [accessed 2 January 2018].

Available at: https://www.w3schools.com/html/html5_intro.asp

W3S 2017c. jQuery Tutorial. W3Schools.com [accessed 2 January 2018].

Available at: https://www.w3schools.com/jquery/jquery_intro.asp

W3S 2017d. Bootstrap Images. W3Schools.com [accessed 21 November 2017]. Available at:

https://www.w3schools.com/bootstrap/bootstrap_images.asp

W3S 2017e. Bootstrap Forms. W3Schools.com [accessed 21 November 2017]. Available at:

https://www.w3schools.com/bootstrap/bootstrap_forms.asp

Appendix2. AngularJS card game module

```

angular.module('cardGame', ['ngDraggable'])
.controller('categoryCtrl', ['$scope', function($scope) {
  $scope.categories =
    ["cardpull", "best", "good", "normal", "bad", "unneeded"];
}])
.controller('cardgameCtrl', ['$scope', '$http', '$element',
function($scope, $http, $element) {
  $http.get('http://localhost/practice/cards').then(function(res) {
    $scope.events = res.data;
    $scope.draggableObjects = $scope.events.cards;
    $scope.droppedObjects1 = $scope.draggableObjects;
    $scope.droppedObjects2 = $scope.draggableObjects;
    $scope.droppedObjects3 = $scope.draggableObjects;
    $scope.droppedObjects4 = $scope.draggableObjects;
    $scope.droppedObjects5 = $scope.draggableObjects;
    $scope.droppedObjects6 = $scope.draggableObjects;
    $scope.onDropComplete1 = function(data, evt) {
      var index = $scope.droppedObjects1.indexOf(data);
      var selectCategory = $scope.$parent.categories[0];
      data.category = selectCategory;
      if (index == -1)
        $scope.droppedObjects1.push(data);
    }
    $scope.onDragSuccess1 = function(data, evt) {
      var index = $scope.droppedObjects1.indexOf(data);
      if (index > -1) {
        $scope.droppedObjects1.splice(index, 1);
      }
    }
    $scope.onDropComplete2 = function(data, evt) {
      var index = $scope.droppedObjects2.indexOf(data);
      var selectCategory = $scope.$parent.categories[1];
      data.category = selectCategory;
      if (index == -1)
        $scope.droppedObjects2.push(data);
    }
  }
  // repeat for each part

```

