OULU UNIVERSITY OF APPLIED SCIENCES

Leon Forte

BUILDING A MODERN WEB APPLICATION USING AN MVC FRAMEWORK

BUILDING A MODERN WEB APPLICATION USING AN MVC FRAMEWORK

Leon Forte
Bachelor's Thesis
Spring 2016
Degree Programme in Business
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Technology

Author: Leon Forte
Title of Bachelor´s thesis: Degree Programme in Business Information Technology
Supervisor(s): Pekka Ojala
Term and year of completion: Spring 2016 Number of pages:

Since the web 2.0 renaissance, the development demands of online content has increased significantly as it became more and more user generated. To ease the development burden many developers faced, web application frameworks were created. Such frameworks come with learning curves though, and learning one can take time away from an already busy schedule.

This thesis will attempt to introduce web developers to popular development frameworks using version three of a beginner friendly framework called CodeIgniter. The thesis explores the components of the framework as well as the model-view-controller (MVC) object oriented architectural design pattern it is based upon. To demonstrate the framework in action, a prototype application was created along with development documentation. At the end of the thesis the author shares the experience of learning CodeIgniter and the value of using an MVC web application framework when developing a modern, data intensive web application.

Keywords: Web application, MVC, CodeIgniter, Web development framework.

CONTENTS

1 INTRODUCTION

At the end of 2005, a framework for developing web applications was released. This framework was called Ruby on Rails (commonly called Rails) and was powered by the Ruby programming language. (Wikipedia 2016j, cited 18.5.2016.) It provided Ruby web developers with a new way of web application development based on a 1970s software architectural pattern originally for implementing user interfaces in desktop software applications. This design pattern is called Model–view–controller (MVC) and is the foundation of such frameworks (Wikipedia 2016i, cited 18.5.2016). Due to the popularity of Rails, it would later spawn a rush of similar web application framework projects created with different programming languages (Wikipedia 2016g, cited 18.5.2016). As the complexity of world wide web pages, now commonly known as web applications, has evolved, these web frameworks serve as an essential tool to meet modern demands, and work well with project management processes like agile and Kanban.

However, there are still many software companies and developers that are somewhat perplexed about MVC frameworks, and still don't know if these kinds of frameworks are the right choice for a specific project (Cheshire 2009, cited 18.5.2016). For example, companies developing with ASP.NET and WebForms. After reading the thesis, the author hopes people in this frame of mind will have a better understanding behind the development practice, and will be able to evaluate whether or not an MVC web development framework is suitable for a specific development project.

In order to do this, the author intends to document the value of writing a web application from a freelance perspective, using the newly updated CodeIgniter3 PHP framework to achieve this —as recommended by my current education institution as an entry level web development framework (Auer, Juntunen and Ojala 2014, cited 18.5.2016). The project will be complex enough to effectively evaluate the value of the framework's libraries, MVC architecture, and act as a guide to those who are considering using an MVC web framework for future projects.

In order address the thesis goal, the thesis will give an overview of the MVC design pattern, and attempt to analyse the benefits of MVC frameworks in regards to separation of concerns in the codebase as well as code reuse.

As the thesis author will be working like a freelancer and not as part of a software development team, he will not be able to objectively analyse other benefits of MVC design, such as developer specialisation and focus, and parallel development by separate teams. It is also important to note that because the author will only be evaluating the development of a web application using just one framework, he is not evaluating the framework itself or making comparisons, but rather the common MVC design pattern methodology and various features shared throughout all MVC frameworks.

2 THE EVOLUTION OF THE WEB

2.1 History

In 1969 the Advanced Research Projects Agency (ARPA), working for the US department of defence, developed ARPANET (Advanced Research Projects Agency Network). ARPANET used NCP (network control protocol) and originally connected US research centres and universities — eventually expanding to include other government sectors, more universities, and international hosts (Wikipedia 2016d, cited 18.5.2016). In 1981 the TCP/IP internet protocol, the replacement of NCP, was standardised, which changed ARPANET to The Internet. This allowed any computer that supported this protocol to be networked together. (Daigle 2016, cited 18.5.2016.)

Web 1.0

The standardisation of TCP/IP which changed ARPANET to The Internet led to Tim Berners Lee to conceive the world wide web (WWW), and he went on to create the first web site running on the first web server in 1990. The earliest web featured static web sites which could be viewed on a web browser that supported the early HTML specification.

In the mid-90s personal computers (PCs) dropped enough in price for most people to afford, and enabled the masses to begin to connect online using dial-up modems (Knight 2014, cited 18.5.2016). In 1995 Amazon opened their online marketplace about the same time as scripting languages (JavaScript and JScript) first appeared in web browsers, and in 1998 Google was founded, providing a popular method to searching the web. It was about this time the web began to provide technologies that improved performance e.g. faster web servers, web caching, content delivery networks etc. (Wikipedia 2016n, cited 18.5.2016; US FCC 2005, cited 18.5.2016.) In 1996 broadband was introduced in North America (Canada), but it took several more years to start to replace dial up modems in the US (Zickuhr and Smith 2012, cited 18.5.2016).

Web 2.0

Just after 2000, wikis, content management systems, and social networks emerged enabling the general user to create web content. In 2005 Ajax, a technology incorporating asynchronous JavaScript, XML or JSON, became popular after Google started demonstrating its usefulness in

their Maps and Gmail web apps. (Wikipedia 2016b, cited 18.5.2016.) Other sites followed suit and the web became much more interactive, data intensive, and started to feel more like applications people were used to using on the desktop.

In 2006, Amazon opened their EC2 cloud computing platform and started to offer its services for websites. This allowed businesses to migrate their IT costs by letting Amazon handle all or part of the online services. (Amazon 2016, cited 18.5.2016.) Around 2007, the emergence and adoption of the 3G service in mobile devices spawned an always connected generation increasing the worth of the web platform (Wikipedia 2016a, cited 18.5.2016).

Web 3.0

With the emergence of HTML5 and other technologies, the web is currently improving and is expanding into new areas like audio/video communication, client storage, location sharing, and web linked data (the semantic web) (WC3 2016, cited 18.5.2016). There is also the emergence of machine facilitated understanding of information AKA The intelligent web —artificial feedback and recommendations for online products and services (Wikipedia 2016m, cited 18.5.2016).

2.2 The need for web application frameworks

Definition of a web application

A web application, or specifically a web-based application, is an application that can be accessed using a web browser (Magic web solutions 2016, cited 18.5.2016). In comparison, a desktop application is an individual application installed on the host platform usually accessing a database over its network.

Although the distinction between a website and a web application can be unclear, the application should have similar functionality to desktop software to be called a web application. There is also a trend for many mobile apps to run on “site specific browsers” (hiding the web browser user interface and usually using HTML5’s offline storage capability), therefore such mobile applications could also be referred to as web applications. (Wikipedia 2016l, cited 18.5.2016.)

According to Borodescu (2013, cited 18.5.2016) about the difference between a website and web app, where he interviewed several prominent figures in the web technology domain, a defining attribute of a web app is that it is action-oriented rather than information oriented.

Some of the first desktop application replacements to be successfully adopted by the public and business community were email apps, starting with Hotmail (replacing MS Outlook) and Gmail. Later, this extended to full office suites, computer aided design applications, and even video editing applications. Of course, web applications can also replace small in-house applications or tools.

Business benefits

According to Magic Web Solutions (2016, cited 18.5.2016) There are many business benefits to creating web applications instead of desktop applications.

- A web application runs in a web browser, which is ubiquitous with PCs and mobile devices today. As web browsers are cross platform, a web app will run on any OS (operating system) and any OS version that one can install a modern web browser onto.
- Web apps usually only require an up-to-date web browser and don't need to be installed individually. Changes to a web app doesn't require installing a software update for every client using it.
- If required, web applications are usable at any time now that one can develop them to operate offline, using local storage.
- Web application user interfaces are easier to create and offer a greater design scope than desktop applications. They can also be easily customised for multiple device screens and change layout responsively.
- As internet technologies are based on open industry-wide standards, interoperability between different web applications is usually much easier to implement.
- A number of flexible core technologies (stacks) are available to fit specific business requirements.
    - Java-based solutions (J2EE) such as JSP and Servlets from Oracle.
    - Microsoft .NET platform such as Active Server Pages, SQL Server and .NET scripting languages.
    - Open Source platform (various open source programming languages and databases).

The technology behind web applications

Depending on the requirements of the web application in question, it can utilise any of the W3C standards defined in the Open Web Platform for application development (WC3 2016, cited 18.5.2016). A web app is commonly organised into two sides —a client side and server side. On the client side, most web apps today use HTML5 for structuring and presenting content; Cascading Style Sheets version 3 (CSS3) for styling the presentation of the content; and JavaScript to control client side logic and update parts of the page. On the server side, a number of programming languages can be used to process HTTP requests generated by the client side, and can interact with database applications, for example MySQL. (Klimushyn 2015, cited 18.5.2016.)

If the web application or website being created is minimal, one can simply use these technologies as is, with minimal maintenance issues. However, if the project is complex, requiring many forms of creating, reading, updating, and deleting of data (known an CRUD operations), the developer can often turn to a web application framework to support the development and maintenance process.

Web application frameworks

In web application development, a web application framework (or framework) is a collection of libraries and best practices designed to alleviate repetitive, common activities, improve security, increase development speed, and provide a more modularised code base to aid maintenance. Different frameworks can vary on how they achieve this, but generally adhere to this basic principle.

Common features of frameworks include a simplified and cross platform way of connecting to, interacting with, and validating database actions; provide methods to manage session data; authorise users and protect authenticated sessions; cache pages to reduce server load; map or re-route URLs to provide cleaner, more user friendly URLs; a templating system to reduce the amount of client side mark-up; and scaffolding, to automatically generate some common database backed code structure. (Srinivasan 2014)

Model-View-Controller

Most web application frameworks make use of the Model-View-Controller (MVC) architectural pattern to separate an application's logic from its data and presentation. For a long time, this pattern has been a popular choice amongst developers designing web applications.

MVC was formulated into the Smalltalk-76 programming system by Trygve Reenskaug in the 1970s when the conception of programming for desktop graphical user interfaces began to take root (Wikipedia 2016i, cited 18.5.2016).

The MVC design architectural pattern promotes cohesion (classes that represent or define only one type of object) and aims for low coupling (a measure of how much classes are interrelated). In UI object oriented programming, it is considered bad practice to mix presentation and data code. Instead, classes called controllers are defined to mediate communication between presentation and data classes. These controller classes decouple the presentation layer and the data layer classes, consequently allowing them to be defined more specifically. (Ambler 2001; Davis 2008, cited 18.5.2016.)

Advantages of high cohesion are said to be reduced module complexity, increased system maintainability, and increased module reusability (Wikipedia 2016f, cited 18.5.2016). High cohesion creates a separation of concerns in the code base allowing developer specialisation and focus, and parallel development if working in teams (Berkeley 2004, cited 18.5.2016).

Model-View-Controller is typically made up of three classes as mentioned in its name. The controller is the intermediary class between the model and the view classes. It controls the flow of information by accepting user input from the view, and instructs both the model and view to perform actions based on that information (See figure 1.). The model is responsible for the data management routines in the application. These are commonly create, read, update, and delete (CRUD) database operations. The view is responsible for presenting the data from the model, and normally contains mostly mark-up displayed as web pages, or for example, RSS feeds. (Thorpe 2011, cited 18.5.2016.)

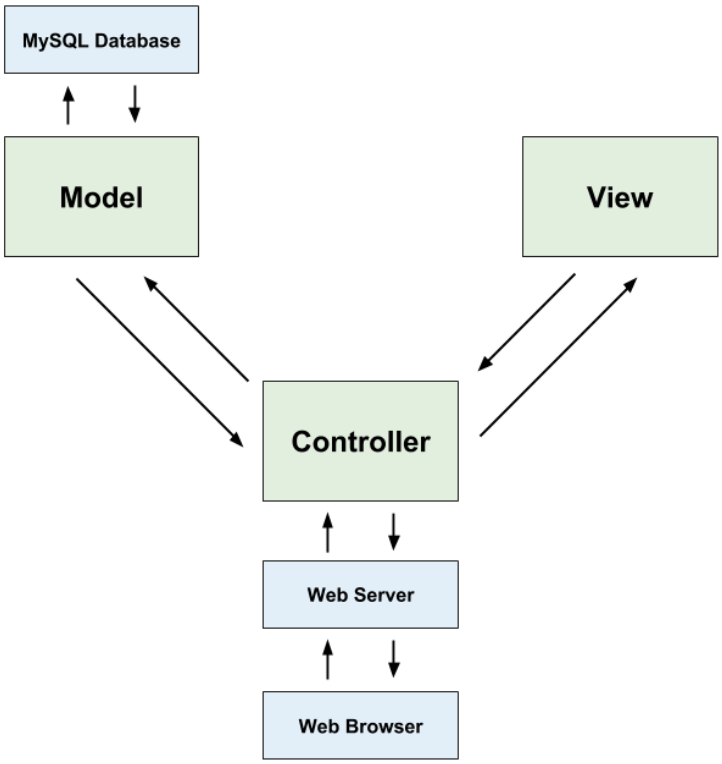

FIGURE 1. The MVC process

Appearance of MVC in web frameworks

In 2002, the use of the MVC pattern in web application design became popular after it was introduced in the Spring web development framework for the Java programming language. It was then implemented for the Ruby programming language in 2006 under the name Ruby on Rails, and later as Django for the Python programming language; ASP.NET MVC for C#; along with numerous implementations for PHP. These frameworks have evolved/adapted the original MVC pattern into slightly different variations to better fit their concept of application design for the web. Regardless of these variations, they are still commonly known as MVC web development frameworks. (Wikipedia 2016g, cited 18.5.2016.)

Definition of a design pattern

Although widely used in all kinds of software development, design patterns have no formal classification or standardisation in the development world. The following is a generally agreed definition.

Object oriented design patterns are categorised into three classifications —creational patterns, structural patterns, and behavioural patterns. Under each category classification are various

design patterns. The modern classification of the MVC pattern is defined under the structural pattern category as a front controller pattern. The front controller pattern is popular in web application design to provide a centralized entry point for handling requests. During this thesis the MVC pattern is often referred to as a design pattern, but in its broader definition, it is better classified as an architectural pattern, which is "A general, reusable solution to a commonly occurring problem in software architecture within a given context" (Wikipedia 2016k, cited 18.5.2016). Therefore, it could be said that MVC is more comprehensively defined as a software architectural design pattern (Wikipedia 2016c, cited 18.5.2016).

Benefits of using the MVC pattern in web frameworks

In this broader definition it is easier to define the benefits of the MVC pattern in web development, and it begins with organisation. Separating code into categories such as models, views and controllers, allows the developer to develop specific areas of project functionality. This will then lead to an easier to manage, more readable code base, when a part of the application needs to be fixed, tested, or extended. This is commonly known as separation of concerns, and if one is working as part of a team, it is especially useful for separating developer workloads as one can have for example, three developers working on different areas in the same part of the application —the model, the view, and the controller. (Berkeley 2004, cited 18.5.2016.)

Most web development projects feature heavily on database backed user interaction through multiple user interfaces (web pages) capturing input known as CRUD operations. As a practical example of this, during the author's professional work experience on such a project (using the C# ASP.NET MVC framework), one found the use of a web development framework and the object oriented architectural design pattern it facilitated, helped break the project down into smaller, more manageable problems, easing the development and problem solving. It also made it easier to work with team members as it emphasised separating the presentation and data access layers, allowing more than one person to work on the same application area (for example, the login area) without code conflict. Finally, helper modules/classes (for example, page data pagination) made it extremely easy to add functionality that would have taken time to design, implement and test.

Framework examples

MVC frameworks are usually constructed with a single object oriented programming language, and therefore they are categorised by language. The following is a list of common server side frameworks commonly used in web development:

- PHP: CakePHP, CodeIgniter, Symfony, Laravel, and Zend Framework.
- Java: Spring, Play, Struts.
- Ruby: Ruby on Rails, Sinatra.
- Python: Django, Flask.
- C#: ASP.NET MVC.

3 CODEIGNITER

This thesis uses CodeIgniter as the framework to demonstrate the building of the web application mentioned in its statement. CodeIgniter is often publicised as easy for beginners to quickly comprehend. For this reason, it is the framework currently used by OUAS for the student developed OpixManager application, and has been used to teach students in the CS164 course at Harvard in 2012. The information here is a condensed version of the CodeIgniter documentation, which is extensive. It focuses on the parts of the framework most likely to be exposed to the beginner, and tries to present it in a more elementary way. A more practical demonstration of CodeIgniter can be found in the next chapter.

History

Released in 2006 by Rick Ellis of EllisLab, CodeIgniter is an open source, web application development framework written in PHP which is praised for its speed, agility, and simplicity. It comes from ExpressionEngine, EllisLab's flagship CMS, and is essentially a collection of refracted classes from the ExpressionEngine backend with its application specific functionality removed. It quickly became one of the most popular frameworks because of its thorough documentation, and ease at which it could be setup on shared hosting accounts (which host many small company's web presences). (EllisLab 2016b, cited 18.5.2016.)

CodeIgniter is an object oriented PHP framework, and like many others like it, is loosely based on the popular MVC architectural pattern which is preferred in applications that implement user interfaces —traditionally desktop graphical user interfaces.

In 2009 EllisLab released version 2 of CodeIgniter (EllisLab 2014a, cited 18.5.2016), but in 2013 EllisLab stopped development of the framework, and announced it was seeking new ownership of it, citing a lack of resources for the decision (providing only security updates from then on). In 2014 they announced stewardship to be handed over to the British Columbia Institute of Technology, who subsequently released version 3 in 2015. (Wikipedia 2016e, cited 18.5.2016.)

During the year EllisLab had CodeIgniter on hiatus, many questions from its community were raised about its future, but after BCIT quickly released version 3 and are currently discussing

improvements for version 4, it is generally agreed that the continued development of the framework is secure.

Goals and features of CodeIgniter

CodeIgniter has various features that may or may not appeal to developers and companies. The framework states, “Our goal for CodeIgniter is maximum performance, capability, and flexibility in the smallest, lightest possible package.” In order to meet these objectives, its components and routines only run and execute when requested. This they call dynamic instantiation, and makes the system very light-weight by default. Ultimately, the developer will need to invoke components and routines as they call code that requires them to be loaded. The framework also tries to adhere to the MVC principles of loose coupling and high cohesion within its design. (British Columbia Institute of Technology 2016c, cited 18.5.2016.)

Some of the specific things that appeal to developers are as follows. Local development setup is trivial. Just download and extract the zip file to a localhost development environment, and change a few configuration parameters. After local development is complete, the app can be quickly installed onto a client’s server or hosting provider’s server by simply changing a few of the configuration files to suite the client’s server requirements, and uploading the files with, for example, an FTP client. Things then just normally work because of CodeIgniter’s ability to run on a variety of PHP versions and configurations.

The framework does not adhere to restrictive coding rules, and although this could then result in the developer not writing ‘best practice’ code, it allows them to quickly create something that works —allowing them to refactor it at a later date (British Columbia Institute of Technology 2016m, cited 18.5.2016). Unlike some other frameworks, CodeIgniter does not require one to use a terminal command line, or rely on tools like package managers, or components from other frameworks (Wikipedia 2016h, cited 18.5.2016).

The framework takes its security seriously. As well as an in-house team (CI project council) dedicated on keeping it secure, it has also affiliated itself with the HackerOne Project to recognise the importance of the security community in general (HackerOne 2016, cited 18.5.2016). CodeIgniter has an extensive list of features that can be reviewed here:
http://www.codeigniter.com/user_guide/overview/features.html

Exploring the general parts of CodeIgniter

Although MVC is an integral part of CodeIgniter, a web development framework is not just its architectural design pattern. It is made up of other important parts (See figure 2) responsible for acting upon data sent around the system.

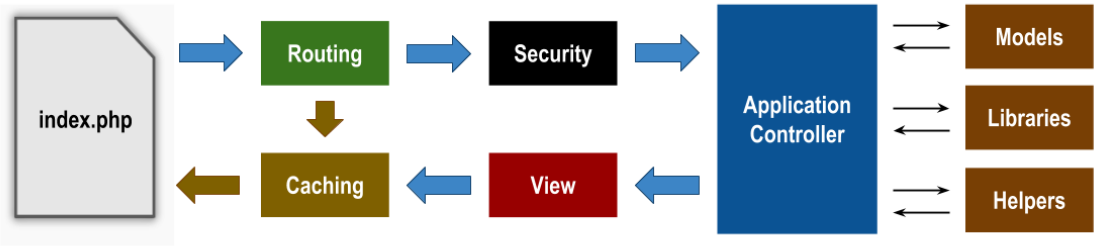

FIGURE 2. Workflow of a typical CodeIgniter application consisting of various modules and classes (revised British Columbia Institute of Technology 2016p, cited 18.5.2016)

URLs and Routing

In traditional web applications, Uniform Resource Locators (URLs) usually used standard query string parameters to 'get' and 'post' data. For example:

```
http://www.example.com/Blog/Posts.php?Year=2015&Month=10&Day=11
```

In CodeIgniter and other MVC frameworks, the segment-based approach is mainly used. The segments, also known as Uniform Resource Identifier (URI) segments, are added to the domain name (including any directory path) and represent the controller class, its method, and any arguments belonging to that class method. For example:

```
http://www.example.com/blog/posts/2015/10/11
```

In the above URL, 'blog' is the controller class, 'post' is its method, and 'year/month/day' are the method's arguments. The arrangement of the segment approach allows URLs to be more human reader, and search engine friendly (British Columbia Institute of Technology 2016a, cited 18.5.2016).

When a framework, such as CodeIgniter, performs routing on a URL, it modifies or rewrites the default action it has with the URI segment. Routing can be set up in the configuration file application/config/routes.php For example:

```
$route['product/(:num)'] = 'catalog/product_lookup_by_id/$1';
```

Here the URI segment can be remapped to “product” and its number, instead of exposing the user to the “catalog” controller class and the descriptive, but verbose “product_lookup_by_id” method and its argument, represented by the “$1” variable. (British Columbia Institute of Technology 2016j, cited 18.5.2016.)

Caching

Caching or web page caching is a method of storing part of a page (specifically view output) that is not usually subject to a change during refreshes, thus allowing faster subsequent reloads of an app. In CodeIgniter, caching is done on a page by page basis. Caching is optional and not set by default, and therefore requires setting in a controller’s method using a number of minutes as the expiration argument to its method call. Cached files are usually set in a folder called application/cache/ and require setting writable permissions to that directory on the web server. Files accessed from this cache folder are said to achieve performance that nears that of static web pages. (British Columbia Institute of Technology 2016l, cited 18.5.2016.)

Security

CodeIgniter has various internal security features designed to apply ‘best practices’ regarding web security. For example, it restricts the type of characters allowed in URI segment strings in order to thwart malicious data being injected into the application.

It also unsets global variables (see PHP’s register_globals) found in $_GET, $_POST, $_REQUEST and $_COOKIE during system initialisation. This prevents variables in various HTTP request methods mixing with other locally defined variables, or worse, users trying to manipulate code in, for example, $_GET requests to bypass various logic checks like authentication. This security measure is effectively the same as register_globals = off in PHP, but forces the setting in case some old server configurations still have it set as “on”. (Dimitrios 2010, cited 18.5.2016.)

Receiving error messages during development is essential, but once the app goes into production and is running publically, developers are advised to turn this feature off, else risk potentially sensitive information being made available to the end user. CodeIgniter provides an easy way to set this this in the application, instead of having to manipulate the php.ini file on the server. The various running modes are development, testing, and production and are set in the ENVIRONMENT constant in the index.php file located in the application’s root folder.

CodeIgniter provides a method to filter possible malicious JavaScript output (Cross site scripting - XSS) that attempts to hijack user browser sessions. It is therefore advised to clean all output coming from third parties (like user comments, or even possible infected image uploads) that will appear to other users. For example:

```
$data = $this->input->post('comment');
$data = $this->security->xss_clean($data);
```

Another type of attack that can affect users is the cross-site request forgery (CSRF). When using the form_helper's form_open() method, CodeIgniter automatically inserts a hidden CSRF field into web forms. This thwarts CSRF attacks by giving web forms a one-time key token that is extremely difficult for an attacker to guess when they are trying and duplicate the form's action.

CodeIgniter's form validation library provides a more concise, easier way to validate or prepare data instead of explicitly calling the many individual validation or preparation functions provided with PHP. Validation can be defined in the controller or model, but is usually called in the controller. The method set_rules() performs various checks on submitted data to verify its type, length, presence, similarity, etc. The following is an example of the set_rules() method.

```
$this->form_validation->set_rules(
'username', 'Username',
'trim|required|min_length[5]|max_length[12]|is_unique[users.username]'
);
```

This example passes three arguments to the set_rules() method. The form field's name, its label, and the rules to be performed —separated by the pipe character. As one can see in the example, the last rule check even accesses a database column's requirements to check, in this case, if username column's data is unique. When validation_errors() is placed in a view, one can automatically receive error messages if form validation fails. By default, validation error messages use the value in the second argument of set_rules() (the form's label) to reference where the error took place and the type of error that occurred. Error messages are also customisable. Using the set_data() method for form values in the view, one can also repopulate a form with the original input after returning an error. Examples of validation will be demonstrated in the next chapter.

CodeIgniter offers various ways to deal with malicious database injection attempts. As well as offering various class methods to escape data when building database queries, it also offers a simplified query binding (also known as prepared data), which is a way of letting the system put

together and escape the database queries for the developer by separating the database syntax and the data when preparing the statement. For example:

```
$sql = "SELECT * FROM your_table WHERE id = ? AND status = ?";
$this->db->query($sql, array(7, 'available'));
```

Finally, it features a query builder class (similar to the Active Record pattern), which provides a very simple interface to perform database operations that are both automatically escaped, and work automatically with many different database engines. For example:

```
$query = $this->db->get_where('positions', array(
'project_id' => $project_id,
'day' => $day
));
```

Is equivalent to the MySQL statement:

```
SELECT * FROM positions WHERE project_id = $project_id AND day = $day;
```

(British Columbia Institute of Technology 2016i, cited 18.5.2016.)

Views

Views are an integral part of the MVC architectural pattern. They play their part in separating application logic (business logic, data access logic, and validation logic) from presentation. Views can be simply thought of as web pages, or fragments of a web page (header, footer, menu etc...). However, the view itself (unlike traditional web applications), is not directly called by the end user. They are always loaded by the controller as instructed by the developer —allowing for greater flexibility of presentation types. As mentioned earlier, if a view is fragmented, CodeIgniter will intelligently handle multiple calls to load each view fragment by appending them in the order each fragment was called. When variable data is required inside the view, it is defined and passed from the controller by way of an associative array or an object. When passed as an associative array, the key values are directly used as variables, or as arrays (if they were an array type). CodeIgniter views support its own special templating syntax which is meant to condense or 'prettify' regular PHP, which is already a HTML templating syntax. However, CodeIgniter's templating syntax requires parsing which makes applications slower, and of course, it also requires time for the developer to learn. (British Columbia Institute of Technology 2016n, cited 18.5.2016.)

Models

Models, like views are an integral part of the MVC architectural pattern. In essence, a model is a class designed to work with information in a database. Due to the flexibility of the CodeIgniter framework, it is not compulsory to use them. However, developers following the traditional MVC approach typically will make use of models. If a model exists, it is called in the controller class like so:

```
$this->load->model('my_model_name');
```

One can then access them by their class name:

```
$data['query'] = $this->my_model_name->fetch_database_info();
```

Although the model is usually for working with data access logic, it is also common for validation rules to be defined in the model and then called by a controller. An example of this will be demonstrated in the next chapter. (British Columbia Institute of Technology 2016h, cited 18.5.2016.)

Controllers

A controller is a class, and like the model and view classes, is part of the MVC architectural design pattern. The job of the controller is to act as a data intermediary between the view and model classes, but it also determines how HTTP requests should be handled. When the user interacts with an application, they will be accessing the controller, so controllers are also thought of as the access point of an application. To expand upon the concept of an access point, in the URLs and routing sub chapter, it was explained what a URI segment was. To reiterate, a URI segment is simply a part of a URL that can call a method (with optional arguments) defined inside of the controller class. This class method is usually written to load specific libraries and helpers, orchestrate calls from the model, and load a particular view while passing data to it, which is ultimately returned to the user. (British Columbia Institute of Technology 2016b, cited 18.5.2016.)

Libraries

Libraries are classes that extend the functionality of (in the case of CodeIgniter) the framework. As mentioned earlier, a full stack web application framework is not just an architectural pattern, but also its libraries. They can be thought of as a toolkit, or as add-ons, and are essentially features that will aid in the rapid development of a web application. CodeIgniter's libraries reside in the systems/libraries directory, while user defined, or third party libraries reside in application/libraries. Libraries are initialised in the controller, for example:

```
$this->load->library('library_class_name');
```

Once loaded, the developer can use its methods as documented. (British Columbia Institute of Technology 2016k, cited 18.5.2016.)

Helpers

Helpers help a developer with various tasks when developing a web application. They are in this sense similar to libraries, but instead of being classes, they are simple, procedural functions similar to PHP's built-in functions. It is for this reason helper functions perform specific tasks and have no dependencies.

CodeIgniter is a lightweight framework, so it's no surprise the developer has to load the required helper inside of a controller, or even a view when needed. This however is quite trivial, for example:

```
$this->load->helper('url');
```

After which the helper can be used like any other PHP function:

```
echo current_url();
```

There are currently over twenty helpers in the framework. Examples of helpers are the CAPTCHA helper, date helper, url helper, form helper, email helper. Helpers that come with the framework are stored in the system/helpers directory. One can also define their own helpers or extend the framework's existing helpers. In this case, custom or extended helpers are stored in application/helpers. (British Columbia Institute of Technology 2016f, cited 18.5.2016.)

Chapter conclusion

In this chapter, the key components of the CodeIgniter framework have been generally discussed at an introductory level. It is apparent that the framework is well organised and there are many important features to facilitate the rapid and secure development of a web application. The framework functionality however is quite extensive, and it is not in the scope of this thesis to cover all of it. If the reader is keen to know more about the depth of the framework, it is advised to browse the extensive documentation available online.

4 DEVELOPING THE WEB APPLICATION

In the first chapter, the need for web application frameworks was promoted by writing about the evolution of the web from a historical perspective. In the second chapter, the workings of the CodeIgniter web application framework was introduced to serve as an example of such a framework. In this chapter, CodeIgniter will be demonstrated using real code for a real project. It is in this demonstration, along with the previous chapters, the author hopes to justify to himself and to the reader the usefulness of web development frameworks to gauge what kind of projects they best suit. This will be discussed in the conclusion.

4.1 The design phase

To demonstrate usefulness of an MVC web framework, a prototype product has been developed with CodeIgniter. The author is a proponent of the build-measure-learn feedback loop of lean start-up culture, and what has been developed is known as a minimal viable product (MVP). MVPs allow a developer or business to release something with the core features to demonstrate and evaluate to potential users or buyers the value of the application. Based on the feedback, the product can then be developed further by adding features that were needed, but not considered essential. This allows the product to have the highest investment return and the lowest investment risk, because both client and developer find out early on not to waste time and money developing a fully featured product people don't find useful. (The Lean Startup 2016, cited 18.5.2016.)

The product has been defined by the thesis author's client as an availability calendar, whereas the author defines it as a people organiser. Whatever the definition, it will be targeted at various organisations and groups to simplify the organisation of people and roles they need to perform within some kind of activity. It redefines an existing product the author made for the same client's organisation using procedural PHP, by building upon the original concept. What was a quickly developed application for a single project running on just one day of the week, will now be extensible to any project defined by the organisation's organisers to run on one or more days of the week. An attempt to simplify the existing user interface (UI) will also be tested.

The vision for the new availability calendar is for it to be more generic, and thus usable by other organisations and groups. If not immediately adoptable by others, it is the hope that the object oriented MVC design of the calendar will make it easily extensible to accommodate various customisations.

During the client meeting phases, requirement gathering documents, use cases, and user stories were drawn up, and feedback from previous users was also reviewed. Some of those documents will be available to the reader to show the design process and to illustrate the idea of the application in general. The following is a list defining the various requirements at this stage of the application. The list has been abridged to meet the requirements of the prototype.

Functional requirements: (What must the app do)

- Feature a user management system for admin, and a super admin.
- Allow an admin to create/edit/delete projects, set the days of the week those projects happen on, set descriptive position requirements (title/description) for a project, and set a maximum number of participants allowed to participate for each position.
- Allow the admin to save an unfinished project as a draft, or publish the project (seen publically to users).
- Allow an admin to create/edit/delete a user account.
- Allow only super admin to set the user account as an admin account.
- Allow the user to view the calendar by navigating the projects and days the projects run on.
- Allow the user to add/remove their availability for a specific task on a specific day.
- Block user input if a maximum volunteer reservation limit is reached.
- Allow a user to change their password.
- Allow a user to request a change to their information (name, phone number, email).
- Allow only a super admin to delete an admin account.
- Forbid an admin to delete their own account.
- Allow a user/admin to contact an admin.
- Allow the admin to maintain the FAQ, user guide, information page.

Non-functional requirements: (Features that are needed but not required to function)

- Adhere to object orientated principles and use an MVC framework for easy extensibility and maintenance.
- Conform to Latest web standards; HTML5, CSS3, and run on recent versions of Chrome, Firefox, Safari, IE, and Opera.
- Use PHP and MySQL (required by the client's hosting provider), and not require shell access.
- Will use JavaScript or jQuery for some client side input verification.
- Will use English as the source language.
- Provide a responsive UI design for mobile device access.
- Provide a minimal interface to quickly access the system.
- Encrypt passwords stored in the database.

Use case diagram

Before any code was written, a use case diagram was created. A use case diagram is a way to illustrate various types of user interaction within the system. Each relationship is called a use case and can help modularise the application at the early stage of development, as well as clarify the functionality of the application between the client and developer at a high level of abstraction.

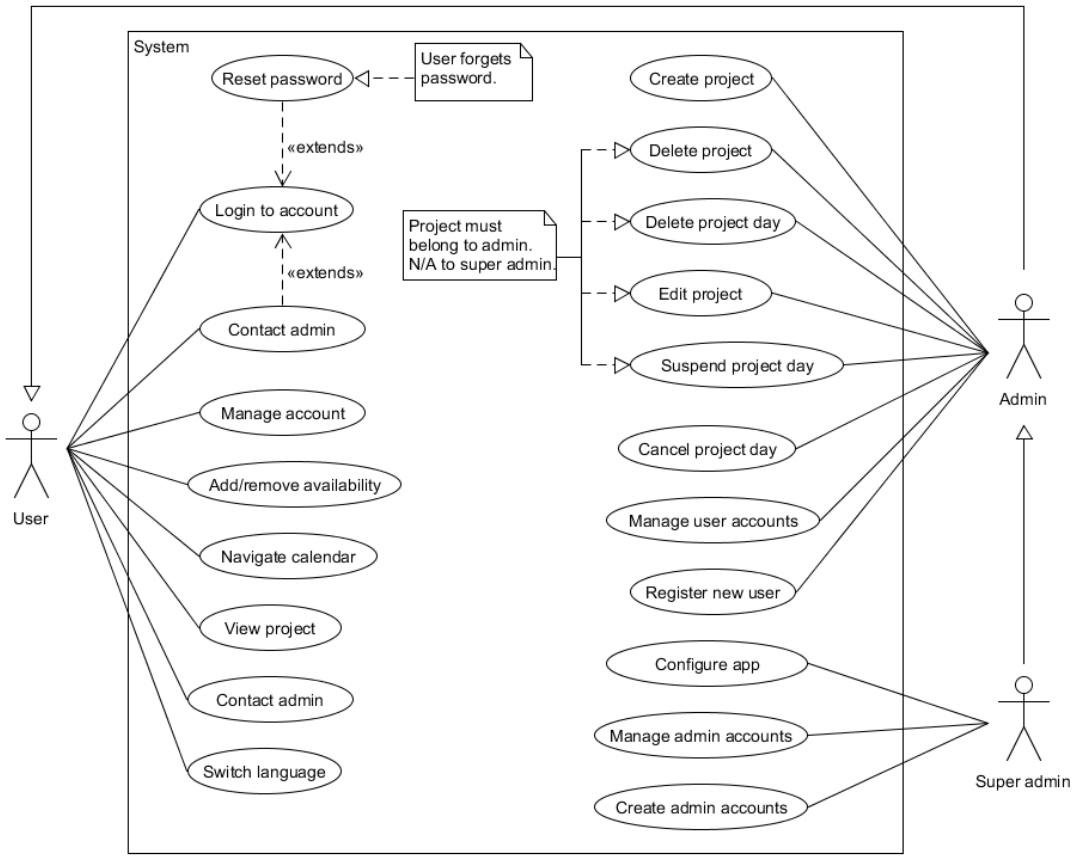

FIGURE 3. Use case diagram for the availability calendar web application

Database design

Following the requirements, a MySQL relational database was designed in accordance with the non-functional requirements of the client. A database design tool called MySQL Workbench was used for this stage. MySQL Workbench is a visual tool for developers to quickly create relational tables using a GUI, then export their tables as database schema to be implemented in the MySQL server. They can also connect to existing databases and perform queries from within the editor. (Oracle 2016, cited 18.5.2016.)

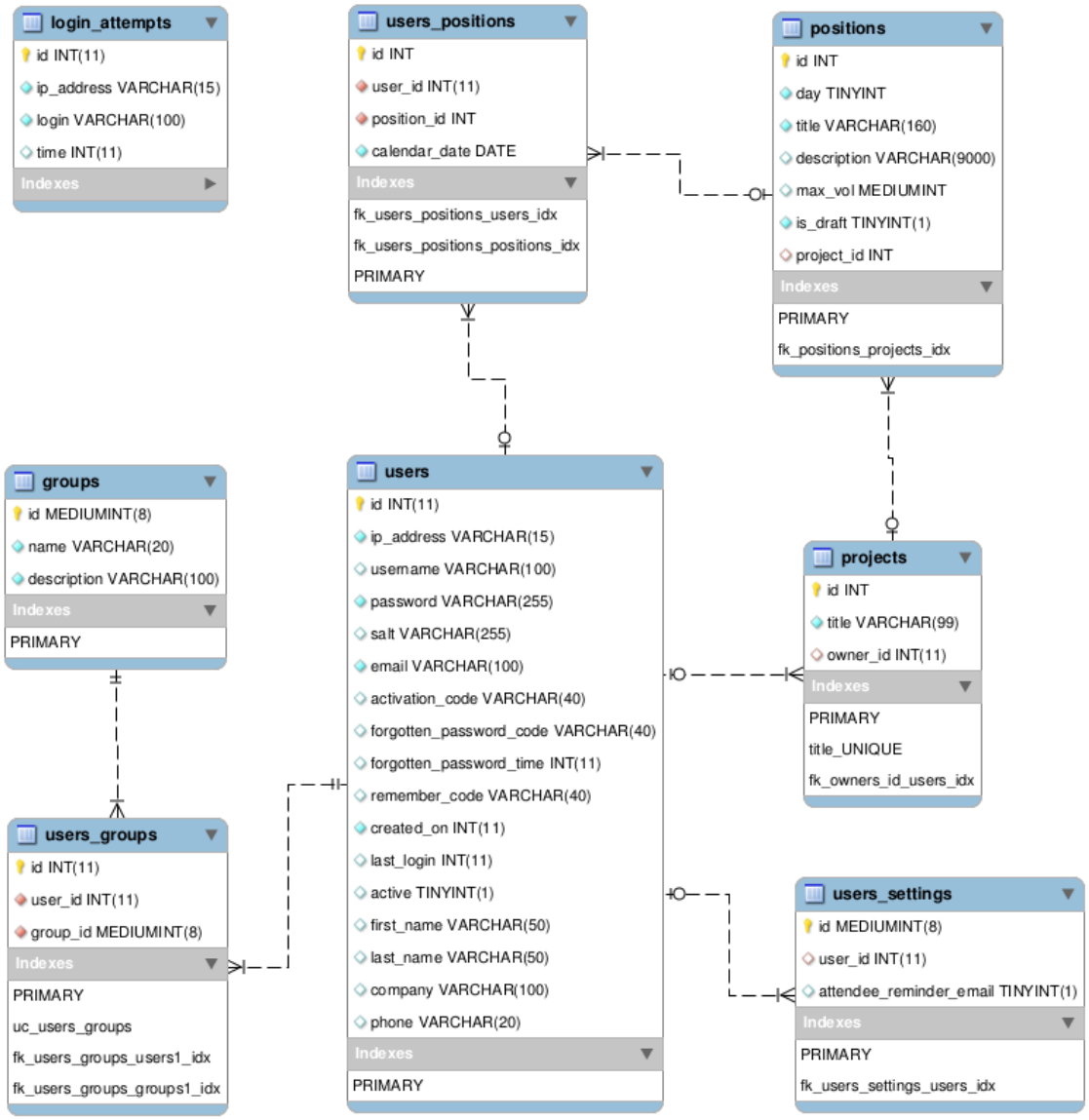

FIGURE 4. Prototype application's relational database design

4.2 The preparation phase

CodeIgniter installation and setup

To develop a CodeIgniter application, all that is needed is a text editor. However, to develop and test on the local development environment, the PHP programming language, a supported web server (for example, Apache2), and database (for example, MySQL) is required to be running on the host operating system. In this thesis, the author has setup a virtual hosts folder on Apache, along with MySQL running on Linux. Apache virtual hosts allows a single web server to serve multiple sites based on an IP address or a domain name (The Apache Software Foundation 2016, cited 18.5.2016). This also makes it possible for a developer to develop many different

sites on one local server (ie. their development PC) giving each project their own domain name and folder for organisation.

After the web server and database are setup, open the CodeIgniter homepage and select the download link from the menu. Select to download the latest version and after download has completed, unzip the file to the local web server's virtual host directory. In the virtual host directory, open application/config/config.php and set the base URL. To do this, locate the line that has…

```
$config['base_url'] = '';
```

...and enter the name of the virtual hosts domain name. CodeIgniter has helpful comments above each setting to help developers understand what each setting is for. Note, when uploading the project to a live web server, the base URL will need to be changed to the website's domain name.

To set up the database, open application/config/database.php and set the database settings. This assumes the developer has created a database. Once again, there are comments about the setting to explain what each setting is for, and when uploading the finished application to the live web server, the setting will have to reflect that of the web server's database. (British Columbia Institute of Technology 2016g, cited 18.5.2016.)

If all goes smoothly, opening a web browser and typing in the domain name of the virtual host will display the CodeIgniter welcome page. The welcome page is displayed by default because the framework routes the base URL to the 'welcome' controller. To change this default behaviour, open application/config/routes.php and change

```
$route['default_controller'] = 'welcome';
```

...to the desired controller and controller method. If the method is called 'index' it can be omitted as is the case of the welcome controller. This is because each controller class inherits some functionality from the parent CI_Controller class. One of those methods is a special index method that diverts absence of a method to the default index method. In the case of this application, the base URL routes to the default login controller and login method ie. 'login/login'. Note, the developer cannot store the default controller inside any other folders within the controller folder. However, there are workarounds if this is requirement is essential.

Remove the need for index.php

CodeIgniter bootstraps itself through a file called index.php found the root directory. When calling a controller, the following must be called by default.

```
example.com/index.php/controller/method
```

To remove this requirement, the Apache2 .htaccess file is edited with the following appended.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

Any HTTP request other than those for existing files and directories is treated as a request for the index.php file. Note, the mod_rewrite module has to be enabled in Apache. Now all controller calls can be written like so.

```
example.com/controller/method
```

Third party authentication library

By default, CodeIgniter does not include an authentication library. However, there are plenty to choose from. For this application, the Ion Auth authentication library was chosen for its simplicity and because its features fit the requirements needed for the application. It is written and maintained by Ben Edmunds who is responsible for Security as a member of the CI project council. The library can be downloaded from https://github.com/benedmunds/CodeIgniter-Ion-Auth The documentation can be found at http://benedmunds.com/ion_auth/.

Installing Ion Auth is straightforward. After downloading the zip file, copy the relevant files in the various folders to the equivalent folders in CodeIgniter. Note, the files in the Controller and View directories are example files and are not required for the installation. In the sql directory, the database file ion_auth.sql was executed to create the database tables required for Ion Auth. The various class methods are now available for the application to use. (Edmunds 2016, cited 18.5.2016.)

Autoloader

In order for CodeIgniter to be as lightweight as possible, the framework does not load many resources by default. The autoloader file config/autoload.php globally defines which systems the developer would like loaded with every server request. The following was added to the autoloader

as they were needed most of the time and thus saved having to manually call them each time in the controller file.

```
$autoload['libraries'] = array('ion_auth', 'form_validation',
'session', 'database');
$autoload['helper'] = array('url', 'form', 'language');
$autoload['config'] = array('cms_settings');
$autoload['language'] = array('auth');
```

Extending the controller core class

When a developer finds they are writing the same code across different controllers, it is best to extend a core controller class. To do this, create a class file inside the framework's core folder and prefix it with "MY_" and have it extend the CI_Controller class. A common name for such a controller is MY_Controller. As the default prefix "MY_" (see $config['subclass_prefix']) is defined in config.php, classes with this prefix will be automatically loaded after the core library. (British Columbia Institute of Technology 2016o, cited 18.5.2016.) Amongst CodeIgniter developers, it is common to further extend MY_Controller into child classes such as Public_Controller and Admin_Controller to facilitate different functionality for users logged in and not logged into the system. (Sturgeon 2010, cited 18.5.2016.) This can be seen in the following example MY_Controller.php

Controller

```
// a base controller for admin and public controllers.
class MY_Controller extends CI_Controller
{
  protected $data = array();

  function __construct()
  {
    parent::__construct();
    $this->data['before_head'] = ''; // inject page specific js or
styles
    $this->data['before_body'] = '';
    $this->data['current_user_menu'] = ''; // define a user specific
menu
    $this->data['page_title'] = 'Set a page title!'; // appears if you
forgot
    $this->data['calendar_days'] = ''; // n-day appearing in a
clanendar month where n = Mon, Tues...
  }

  // set page templates and pass construct data when loading the view
  protected function render($the_view = NULL, $template = 'master')
  {
    if(is_null($the_view)) {
      $this->data['the_view_content'] = '';
    }
    else {
```

```
      $this->data['the_view_content'] = $this->load->view($the_view,
$this->data, TRUE);
    }

    $this->load->view('templates/'.$template.'_view', $this->data);
  }

}

// the area accessable by logged in users
class Admin_Controller extends MY_Controller
{
  function __construct()
  {
    parent::__construct();
    // allow further site access only to those logged in, or refuse
further
    // access if a session has expired
    if(!$this->ion_auth->logged_in()) { redirect('login/login',
'refresh'); }

    // user_name and group_name is used in the navbar to display the
current
    // user and group menu
    if(isset($this->ion_auth->user()->row()->first_name)) {
      $this->data['user_name'] = $this->ion_auth->user()->row()-
>first_name;
    }
    if(isset($this->ion_auth->get_users_groups()->row()->name)) {
      $this->data['group_name'] = $this->ion_auth->get_users_groups()-
>row()->name;
    }

    // user is logged in. Get their details
    $this->data['current_user'] = $this->ion_auth->user()->row();
    // add the admin menu if they belong to admin or super_admin
groups
    if($this->ion_auth->in_group('admin') || $this->ion_auth-
>in_group('super_admin'))
    {
      // load these menu items if user is an admin
      $this->data['current_user_menu'] = $this->load-
>view('templates/_parts/user_menu_admin_view.php', NULL, TRUE);
    }

    // Initiate the projects view with project named menu items
    $this->load->model('menu_model');
    $this->data['menu_projects'] = $this->menu_model->get_projects();
  }

  protected function render($the_view = NULL, $template =
'admin_master')
  {
    parent::render($the_view, $template);
  }
}

// the area accessable by the public
class Public_Controller extends MY_Controller
{
  function __construct()
```

```
  {
    parent::__construct();
    // if user is already logged in and tries to access a public
class,
    // log them out before continuing
    if($this->ion_auth->logged_in()) { $this->ion_auth->logout(); }
  }

}
```

Another thing to note here is the custom method "render". This improves upon the `$this->load->view()` method that comes with CodeIgniter by setting the correct page templates and passing construct data from MY_Controller when loading the view. The various page templates are defined in the views folder.

Custom configurations

Developers can also create their own custom configuration files and reconfigure default configurations. Two examples used in this application were cms_settings.php and form_validation.php both defined within application/config. The file cms_settings.php was created to allow various parts of the application to be quickly customised. In this case

```
$config['cms_title'] = 'acal';
$config['cms_super_admin_email'] = 'acal@acal.com';
```

...which allows any adopter to easily tailor the core elements of the application to suite their own requirements. After creating a custom configuration file, it is added to the config array in autoload.php

```
$autoload['config'] = array('cms_settings');
```

The file form_validation.php was created to reconfigure existing form validation library variables. This file is not created by default in CodeIgniter, but if the developer chooses to do so, it is not required to be added to autoload array like with the previous example. In this application, the author customised the default error delimiters to align with Bootstrap's CSS styling. (British Columbia Institute of Technology 2016e, cited 18.5.2016) For example,

```
$config['error_prefix'] = '<div class="alert alert-danger">';
$config['error_suffix'] = '</div>';
```

There are also other form validation customisations the developer can set in this file, such as defining form validation rules.

4.3 The implementation phase

Now that the client requirements have been laid out, the development environment has been set up, and a database design has been implemented, a select number of use cases will be chosen to describe the various scenarios, along with the CodeIgniter code. The first example demonstrates the login functionality.

4.3.1 Login

Title	Account Login
Actor	User, admin, super admin
Scenario	1. System requests actor to enter credentials. 2. Actor enters correct credentials. 3. System validates credentials and redirects the actor to the main page.
Alternative	1. Actor forgets password. 2. Actor clicks forgotten password link. 3. System opens password reset page.

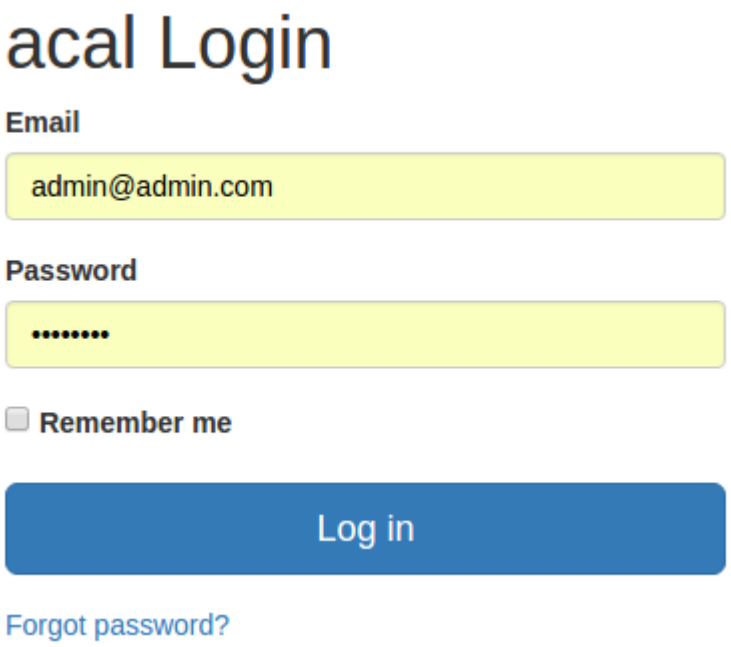

FIGURE 5. The Login screen

Note, there is no model coded by the developer in this example because connecting to the database and verifying the user is handled by the Ion Auth library model. A developer could define the validation rules in their own model, but as this is a straightforward example, CodeIgniter allows flexibility to define input validation in the controller.

Controller code

```
// Access point for the user
class Login extends Public_Controller
{
  function __construct()
  {
    parent::__construct();
    $this->data['page_title'] = 'Login';
  }

  public function login()
  {
    // in case the root domain is entered and the user is already
logged in,
    // this will prevent the login screen appearing in the view
    if($this->ion_auth->logged_in()) {
redirect('calendar/calendar/home', 'refresh'); }

    if($this->input->post())
    {
      $this->form_validation->set_rules('identity', 'Identity',
'required|valid_email');
      $this->form_validation->set_rules('password', 'Password',
'required');
      $this->form_validation->set_rules('remember','Remember
me','integer');

      if($this->form_validation->run()===TRUE)
      {
        $remember = (bool) $this->input->post('remember'); // remember
me checkbox

        if ($this->ion_auth->login($this->input->post('identity'),
$this->input->post('password'), $remember))
        {
          redirect('calendar/calendar/home', 'refresh');
        }
        else
        {
          $this->session->set_flashdata('message', $this->ion_auth-
>errors());
          redirect('login/login', 'refresh');
        }
      }
    }
    $this->render('admin/login_view');
  }

}
```

Within the controller construct, the page's tab title (header tag) has been set. It has been defined here and not in the view, because the application view page will load itself from a series of templates. The view here is concerned only with the body content and therefore is separate from the headers, menu, and footer parts of the page.

Within the login method, it first checks for post variables. If not found, the login view page is called. If post variables are found, it then checks against a series of validation rules defined in form_validation. If the validation passes, Ion Auth processes the login, and redirects the user to the appropriate page, else the user is returned to the login page and a one-time session variable error message is set.

The view

```
<div class="row">
  <div class="col-lg-4 col-lg-offset-4">
    <h1 style="margin-top:45px;"><?php echo $this->config-
>item('cms_title');?> Login</h1>
    <?php
      echo $this->session->flashdata('message'); // output any
flashdata session messages
      echo form_open('',array('role'=>'form'));
    ?>
      <div class="form-group">
        <?php
          echo form_label('Email','identity');
          echo form_error('identity'); // field specific error - like
required
          echo form_input('identity','','class="form-control"');
        ?>
      </div>
      <div class="form-group">
        <?php
          echo form_label('Password','password');
          echo form_error('password');
          echo form_password('password','','class="form-control"');
        ?>
      </div>
      <div class="form-group">
        <label>
          <?php echo form_checkbox('remember','1',FALSE);?>
          Remember me
        </label>
      </div>
      <?php echo form_submit('submit', 'Log in', 'class="btn btn-
primary btn-lg btn-block" style="margin-bottom:15px;"');?>
      <p>
        <?php echo anchor('admin/forgot_password', 'Forgot
password?');?>
      </p>
    <?php echo form_close();?>
  </div>
</div>
```

In the H1 tag, the framework pulls the name of the application from the custom configuration file. Just below the H1 tag, any session error messages will be outputted with the flashdata function. The form helper provides form_open and form_close functions to define the web form. It also provides functions to define form elements like input, label, and submit, as well as support for error placeholders, and CSS styling.

4.3.2 Menu (including document templates)

Various scenarios for the loading of the menu are dependent on whether the user is logged in, and whether or not they are an admin, or a regular user/member. All these scenarios are controlled by MY_Controller (see Extending the core class).

Title	Menu
Actor	User, admin, super admin
Scenario	1. Actor is not logged in. 2. System does not display the user menu. 3. Actor successfully logs in. 4. System displays user's name, public projects and project days, and a user menu.
Alternative	1. Actor logs in and is an admin. 2. System appends admin menu items to the user menu.

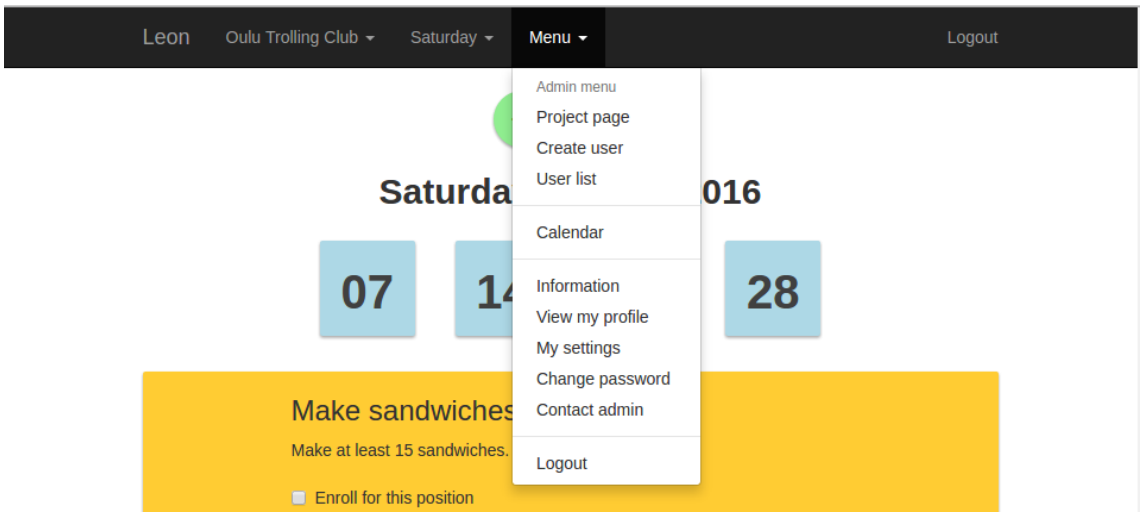

FIGURE 6. The main menu

Template

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title><?php echo $page_title;?></title>

<!-- BootStrap -->
<link href="<?php echo site_url('assets/css/bootstrap.min.css');?>"
rel="stylesheet">
<!-- acal custom styles -->
<link href="<?php echo site_url('assets/css/acal/acal-custom.css');?>"
rel="stylesheet">

<!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and
media queries -->
<!-- WARNING: Respond.js doesn't work if you view the page via file://
-->
<!--[if lt IE 9]>
  <script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script
>
  <script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
<![endif]-->

<?php
  // optional styles or js
  echo $before_head;
?>

</head>

<body>

<nav class="navbar navbar-inverse navbar-static-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#navbar"
aria-expanded="false" aria-controls="navbar">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <!-- Site title -->
      <a class="navbar-brand" href="<?php echo
site_url('admin/user/profile');?>"><?php echo ucfirst($user_name);
?></a>
    </div>

    <div id="navbar" class="collapse navbar-collapse">

      <!-- PROJECT -->
      <ul class="nav navbar-nav">
        <li class="dropdown">
```

```
          <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-expanded="false"><?php
if(isset($_SESSION['selected_project_name'])){ echo $this->session-
>selected_project_name;} else {echo 'Select Project';} ?> <span
class="caret"></span></a>
          <ul class="dropdown-menu">
            <?php
              // get projects from the database output them here
              // $menu_projects is defined in the admin controller
              if (empty($menu_projects)) {
                echo '<li><a href="javascript:;">No projects added
yet</a></li>';
              }
              else {
                foreach ($menu_projects as $project) {
            ?>
                  <li><a href="<?php echo
site_url('menu/set_project/'.$project->id);?>"> <?php echo $project -
>title; ?></a></li>
            <?php
                }
              }
            ?>
          </ul>
        </li>
        <!-- VIEWS -->
        <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-expanded="false"><?php
if(isset($_SESSION['selected_project_day'])){ echo jddayofweek($this-
>session->selected_project_day, 1);} else { echo 'Calendar View'; } ?>
<span class="caret"></span></a>
          <ul class="dropdown-menu">
            <?php
              // get projects from the database output them here
              if (!isset($_SESSION['menu_views'])) {
                echo '<li><a href="javascript:;">Choose a project with
days</a></li>';
              }
              else {
                foreach ($this->session->menu_views as $day) {
            ?>
                  <li><a href="<?php echo
site_url('menu/set_menu_view/'.$day);?>"> <?php echo jddayofweek($day,
1); ?></a></li>
            <?php
                }
              }
            ?>
          </ul>
        </li>
        <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-expanded="false">Menu <span
class="caret"></span></a>
          <ul class="dropdown-menu">
            <?php
              echo $current_user_menu; // insert user specific menu
(just admin at this time)
              // insert shortcut to calendar. Populate with session
data if previously setup by user
```

```
                if(isset($_SESSION['selected_project_id']) &&
isset($_SESSION['selected_project_day'])){
            ?>
                <li><a href="<?php echo
site_url('calendar/calendar/home/'.$_SESSION['selected_project_id'].'/
'.$_SESSION['selected_project_day']); ?>">Calendar</a></li>
            <?php
                }
                else { ?>
                <li><a href="<?php echo
site_url('calendar/calendar/home');?>">Calendar</a></li>
            <?php
                }
            ?>
            <li role="separator" class="divider"></li>
            <li><a href="<?php echo
site_url('info');?>">Information</a></li>
            <li><a href="<?php echo
site_url('admin/user/profile');?>">View my profile</a></li>
            <li><a href="<?php echo
site_url('admin/user/user_settings');?>">My settings</a></li>
            <li><a href="<?php echo
site_url('admin/user/change_password');?>">Change password</a></li>
            <li><a href="<?php echo
site_url('contact/contact');?>">Contact admin</a></li>
            <!-- <li><a href="<?php echo
site_url('#');?>">Contact</a></li> -->
            <li role="separator" class="divider"></li>
            <li><a href="<?php echo
site_url('admin/user/logout');?>">Logout</a></li>
          </ul>
        </li>
      </ul>

      <ul class="nav navbar-nav navbar-right">
        <li><a href="<?php echo
site_url('admin/user/logout');?>">Logout</a></li>
      </ul>


    </div>
    <!--/.nav-collapse -->
  </div>
</nav>
```

The admin_master_header_view.php file provides content defining the document type, the inclusion of third party libraries, and the menu. Variables such as $page_title and $before_head are loaded, and the Bootstrap styled menu is laid out. Within the menu container, the user's first name is fetched and linked to their profile page. The script then checks session variables to see if the user has selected a project and project day, and displays them if they are found. It then populates the project dropdown will all public projects. If a project appears in the session variable, it will populate all the days that project runs on in the project days' dropdown. Finally, the

standard user dropdown menu is displayed, and if the user is an admin or super admin, the content from $current_user_menu will be prepended to the standard user menu.

4.3.3 Sending email

Throughout the application, the email class is used to notify users when an admin has created an account for them, and to send them password reset links. In this simple demonstration of how to send email, the application's contact feature has been chosen. The email class supports multiple email protocols along with a host of other email features. Email setup is easiest done by creating an email configuration file in the application's config folder. (British Columbia Institute of Technology 2016d, cited 18.5.2016.) For local development, the author has used his Gmail SMTP account, but SMTP is also used on the client's host provider and just requires changing a few variables in the config file when moving the application to production. Here is an example of the config file.

```
$config['mailtype'] = 'html'; // default text
$config['charset'] = 'utf-8';
$config['protocol'] = 'smtp';
$config['smtp_port'] = 465; // default is 25 (no ssl)
$config['newline'] = "\r\n"; // essential to use double quotes here!!!

// gmail
$config['smtp_host'] = 'ssl://smtp.gmail.com';
$config['smtp_user'] = 'account@gmail.com';
$config['smtp_pass'] = 'gmail_password';
```

Controller

```
class Contact extends Admin_Controller
{
  function __construct()
  {
    parent::__construct();
    $this->data['page_title'] = 'Contact';
  }

  function index()
  {
    // validation rules
    $this->form_validation->set_rules('title', 'Title',
'trim|required');
    $this->form_validation->set_rules('message', 'Message',
'trim|required');

    if ($this->form_validation->run() == FALSE) {
      // set any errors and display the form
      $this->data['message'] = (validation_errors()) ?
validation_errors() : $this->session->flashdata('message');
      $this->render('contact/contact_view');
```

```
    }
    else {
      $sender = $this->ion_auth->user()->row()->email;
      $sender_name = $this->ion_auth->user()->row()->first_name.'
'.$this->ion_auth->user()->row()->last_name;
      $admin = $this->config->item('cms_dev');

      $this->email->from($sender, $sender name);
      $this->email->to($admin);
      $this->email->subject('['.$this->config->item('cms_title').']
'.$this->input->post('title'));
      $this->email->message($this->input->post('message'));

      if($this->email->send()){
        $this->session->set_flashdata('message', '<div class="alert
alert-success"><strong>Mail sent. </strong>A reply will be sent to
your user email address.</div>');
      }
        else {
        $this->session->set_flashdata('error', '<div class="alert
alert-danger"><strong>Sorry, </strong>unable to send email.</div>');
        }

      $this->render('contact/contact_view');
    }
  }

}
```

The email class provides all the appropriate methods. To send email, the email of the current user has been extracted from Ion Auth and passed to the “from” method. The value of cms_dev from the applications custom configuration file is passed to the “to” method. The application title is then prepended to the value of the form’s title input when passed to the “subject” method. This will help identify where email is coming from. Finally, the value of the message field is passed to the “message” method. The send method returns either true or false (success or failure), enabling it to be used conditionally.

4.3.4 Projects

The projects page is responsible for creating projects, editing their names and deleting them. Once a project has been created, its positions can also be accessed.

Title	Projects
Actor	Admin, super admin
Scenario	1. Admin actor selects projects from the admin menu. 2. System opens projects page and lists all projects.

	3. System gives option to create new projects. 4. System only allows a project owner to make modifications to it.
Alternative	1. Super admin actor selects projects from the admin menu. 2. System lists all projects along with editing rights to all of them.

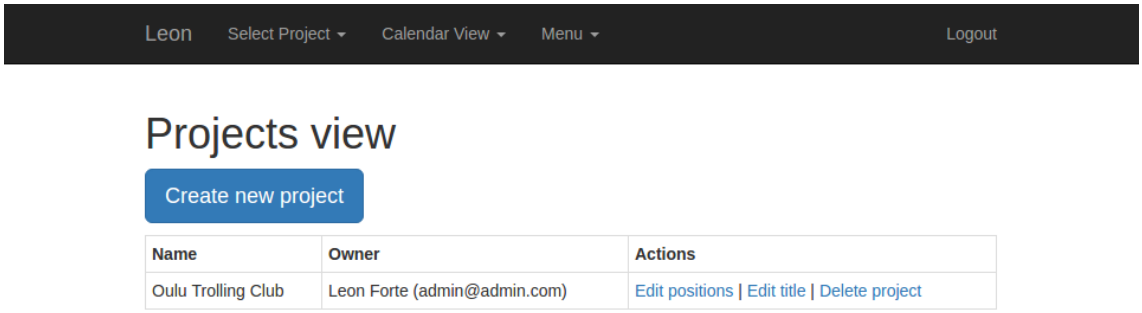

FIGURE 7. Projects main view

Controller

```
class Projects extends Admin_Controller
{
public function __construct()
{
  parent::__construct();
  $this->load->model('projects/projects_model');
}

// list the projects
public function index()
{
  $this->data['page_title'] = 'Projects page';
  $this->data['projects'] = $this->projects_model->get_projects();
  $this->render('projects/list_projects_view');
}

// create a new project (title)
public function create()
{
  $validate = $this->projects_model->validate['create_project']; //
validation validate
  $this->form_validation->set_rules($validate); // set the validation
rules
  // validate form submission or form has loaded for the first time
  if($this->form_validation->run() === FALSE)
  {
    $this->render('projects/create_project_view'); // return the user
  }
  else
  {
    $this->projects_model->create_project_title();
    redirect('calendar/projects', 'refresh');
  }

}
```

```
// edit the title of the project
public function update_title($project_id = NULL, $owner_id = NULL)
{
  // use the same validation rules as create project title
  $validate = $this->projects_model->validate['create_project'];
  $this->form_validation->set_rules($validate);

  if($this->form_validation->run() === FALSE)
  {
    $this->data['project_name'] = $this->projects_model-
>get_project_name($project_id);
    $this->render('projects/update_project_title_view'); // return the
user
  }
  else
  {
    // call the model method
    $this->projects_model->update_project_title($project_id,
$owner_id);
    //Redirect to positions page
    redirect('calendar/projects', 'refresh');
  }
}

public function delete_project($project_id = NULL, $owner_id = NULL)
{
  $this->projects_model->delete_project($project_id, $owner_id);
  redirect('calendar/projects', 'refresh');
}

}
```

The controller's index method fetches project details using the get_projects method from the projects_model. In the create method, a different approach has been made to form validation. Instead of defining the validation rules in the controller, the rules have instead been defined in the model and passed to the set_rules method as an array. If validation (still tested in the controller) is successful, the controller calls the create_project_title method from the model to create the database entry. In the update_title method, the validation rules are shared with those of create_project_title, but calls a different method to update the entry in the database. It's also worth noting that in this example, the controller calls different views depending on the action (ie. create new project, and edit project title).

Model

```
class Projects_model extends CI_Model
{
  public $current_user_id; // current user's id

  public function __construct()
  {
    parent::__construct();
```

```
    $this->current_user_id = $this->ion_auth->user()->row()->id; //
get their id.
  }

  // all the rules of all the project forms. This keeps the controller
light.
  public $validate = array(
    'create project'=>
      array(
        'title' => array(
          'field'=>'project_title',
          'label'=>'Project name',

'rules'=>'trim|required|max_length[99]|is_unique[projects.title]'
        )
      )
  );

  // get a list of project titles and their owners
  public function get_projects()
  {
    $query = $this->db->get('projects'); // select the table with
ActiveRecord get() same as SELECT * FROM test_table
    return $query->result(); // return the rows selected
  }

  // set new a project title and their owner
  public function create_project_title()
  {
    $data = array(
      'title' => $this->input->post('project_title'),
      'owner_id' => $this->current_user_id,
    );
    return $this->db->insert('projects', $data);
  }


  public function update_project_title($project_id = NULL, $owner_id =
NULL)
  {
    // prevent modifing something without an ID OR if the user id
doesn't match the owner id
    if( is_numeric($project_id) && is_numeric($owner_id) && ($this-
>current_user_id == $owner_id || $this->ion_auth-
>in_group('super_admin')) )
    {
      $data = array(
        'title' => $this->input->post('project_title')
      );

//https://www.codeigniter.com/userguide3/database/query_builder.html#u
pdating-data
      return $this->db->update('projects', $data, array('id' =>
$project_id)); // 3rd arg is the where clause
    }
    $this->session->set_flashdata('message', "<p>1 ID check failed:
$project_id / $owner_id </p>");
    redirect('calendar/projects', 'refresh');
  }

  // get the name of a project via its id
```

```
  public function get_project_name($project_id = NULL)
  {
    if($project_id === NULL || !is_numeric($project_id)){
redirect('calendar/projects', 'refresh'); }
    // pull one record from the db
    $row = $this->db->get_where('projects', array('id' =>
$project_id))->row();
    return $row->title;
  }

  public function delete_project($project_id = NULL, $owner_id = NULL
)
  {
    if( is_numeric($project_id) && is_numeric($owner_id) && ($this-
>current_user_id == $owner_id || $this->ion_auth-
>in_group('super_admin')) )
    {
      return $this->db->delete('projects', array('id' =>
$project_id));
    }
      redirect('calendar/projects', 'refresh');
  }

}
```

The projects model, demonstrates the ability to define validation rules in the model. To do this an associative array called $validate is defined and the rules are entered as a series of key value pairs. All the methods listed in the class perform various database calls and the type of calls can be deduced from their method name.

The preference to use CodeIgniter's query builder is also demonstrated as the preferred method to make these database calls. The query builder methods get, insert, update, get_where and delete can be seen. These all translate into standard escaped MySQL queries. Some custom validation rules are also added to the model methods, so that certain security measures are not bypassed, for example whether the current user is the owner of a project being modified/deleted.

View

```
<div class="row">
  <div class="col-lg-12">
    <h1>Projects view</h1>
    <?php echo $this->session->flashdata('message'); ?>
    <a href="<?php echo site_url('calendar/projects/create');?>"
class="btn btn-primary btn-lg">Create new project</a>
  </div>
</div>

<div class="row">
  <div class="col-lg-12" style="margin-top: 10px;">
  <?php
  if(!empty($projects))
  {
```

```
    echo '<table class="table table-hover table-bordered table-
condensed">';
    echo '<tr><th>Name</th><th>Owner</th><th>Actions</th></tr>';
    foreach($projects as $project)
    {
      echo '<tr>';
      $project_user = $this->ion_auth->user($project->owner_id)-
>row();
      $owner = "$project_user->first_name $project_user->last_name
($project_user->email)";
      echo '<td>'.$project->title.'</td><td>'.$owner.'</td><td>';
      // Rules:
      // if the user is in the group admin, they can only modify their
own projects.
      // if the user is a super admin, they have all access rights on
projects.

      // user is super admin
      if($this->ion_auth->in_group('super_admin'))
      {
        echo
anchor('calendar/project_positions/update_position/'.$project-
>id.'/'.$project->owner_id,'Edit positions').' |
'.anchor('calendar/projects/update_title/'.$project->id.'/'.$project-
>owner_id,'Edit title').' |
'.anchor('calendar/projects/delete_project/'.$project-
>id.'/'.$project->owner_id,'Delete project');
      }
      // user is admin && user owns the project
      else if($this->ion_auth->in_group('admin') && $current_user->id
== $project->owner_id)
      {
        echo
anchor('calendar/project_positions/update_position/'.$project-
>id.'/'.$project->owner_id,'Edit positions').' |
'.anchor('calendar/projects/update_title/'.$project->id.'/'.$project-
>owner_id,'Edit title').' |
'.anchor('calendar/projects/delete_project/'.$project-
>id.'/'.$project->owner_id,'Delete project');
      }
      else
      {
        echo 'Access not granted.';
      }
      echo '</td>';
      echo '</tr>';
    }
    echo '</table>';
  } else { echo '<div class="alert alert-info"><h3>No projects found
:(</h3></div>'; }
  ?>
  </div>
</div>
```

The example of the projects view demonstrates outputting view data into tables. It shows the use of the anchor function from the URL helper, to easily build a series of URIs, by iterating through view data. While the URIs are being built, it uses Ion Auth to control what is allowed to appear, in

order to meet the ownership and super admin rights requirements. These rights are also checked on the server side in case a URI segment is manipulated by the user.

4.3.5 Edit positions

Once a project has been created, the owner or super admin can access the project's positions page. This is where they can specify the days of the week that project runs on, and the positions the attendees are required to fill for each day. Once defined, the owner or super admin can publish a project's day (making it appear on the calendar page), or save it as a draft.

Title	Edit positions
Actor	Admin, super admin
Scenario	1. Admin actor selects a project to edit from the project page. 2. System displays name of the project, the default day of the week, and the dynamic positions form. 3. System offers the actor to publish, save as draft, or delete the project.

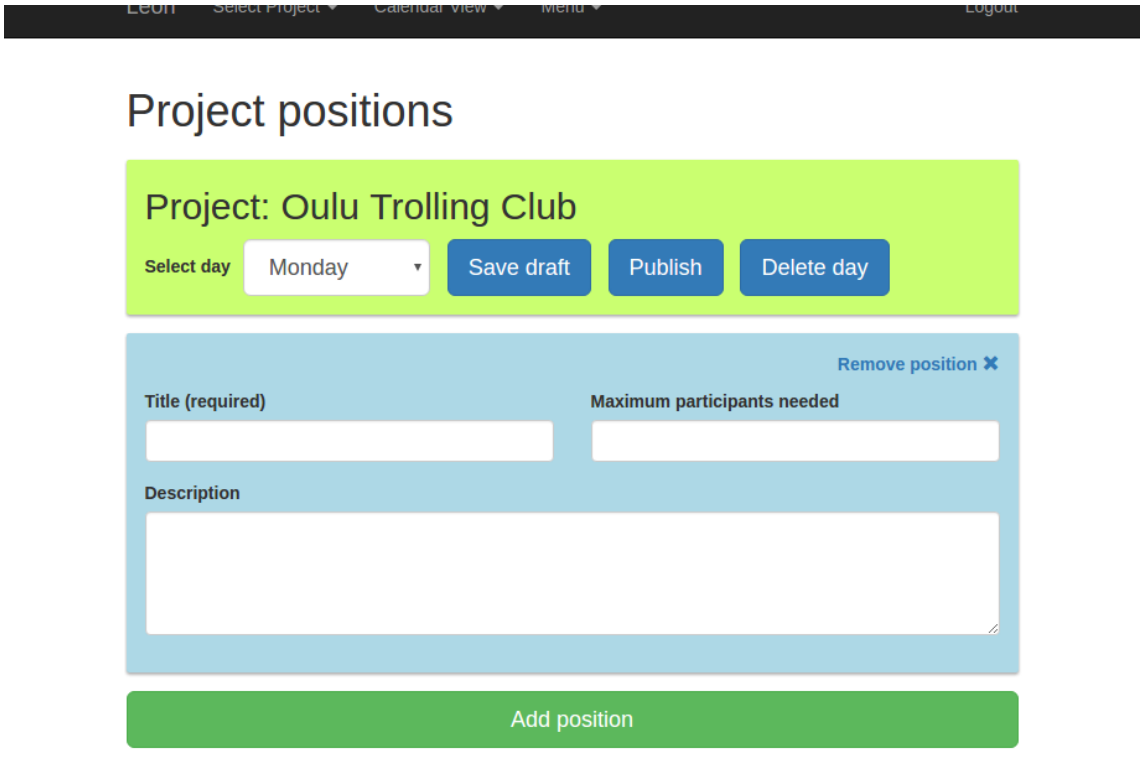

FIGURE 8. Project positions view

Controller

```
class Project_positions extends Admin_Controller
{

public function __construct()
{
  parent::__construct();
  $this->load->model('projects/positions_model');
  $this->load->model('projects/projects_model');
  $this->data['page_title'] = "Project positions";
  $this->data['before_body'] = '<script
src="'.site_url('../assets/js/acal/project_positions.js').'"></script>
';
}

// populates the admin's project position view.
// also works with the select day form element to pull the positions
for a selected day
public function update_position($project_id = NULL, $owner_id = NULL,
$day = NULL)
{
  if($day == NULL){ $day = 0; } // set day to 0 == Monday (default
view)
  $this->data['project_id'] = $project_id;
  $this->data['day'] = $day;
  // get model data
  $this->data['project_positions'] = $this->positions_model-
>get_positions($project_id, $day); // all positions by project_id
  $this->data['project_name'] = $this->projects_model-
>get_project_name($project_id);

  // form validation
  $validate = $this->positions_model->validate['update_position'];
  $this->form_validation->set_rules($validate);

  if($this->form_validation->run() === FALSE) {
    $this->render('projects/project_positions_view');
  }
  else {
    // check which action the user requested (see UI buttons -> Draft,
Publish, Delete)
    // $_POST['submit'] holds the string draft, submit, or delete;
    if($this->input->post('submit') == 'draft') {
      $this->positions_model->update_position();
    }
    else if ($this->input->post('submit') == 'publish') {
      $this->positions_model->update_position();
    }
    else if ($this->input->post('submit') == 'delete') {
      $this->positions_model->delete_position();
    }
    else {
      echo "<h2>No action detected.</h2>";
      exit();
    }


redirect('calendar/project_positions/update_position/'."$project_id/$o
wner_id/$day");
  }
```

```
}

}
```

The positions controller demonstrates the inclusion of client side JavaScript being linked to the before_body element in the $data associative array. There are times when the developer may want to reuse a model method instead of recreating it in the model they are associating the current controller with. This controller demonstrates the possibility of calling multiple models in the constructor to achieve this. In the update_position method, there is a good example a method accepting and processing multiple arguments. Finally, if the page form has been submitted the submit post variable is checked and the appropriate model method call is taken.

View

```
<div class="row">
  <div class="col-lg-8 col-lg-offset-2">

    <h1 style="padding-bottom:10px">Project positions</h1>
    <?php echo $this->session->flashdata('updated');?>
    <?php echo validation_errors(); ?>

    <!-- FORM -->
    <?php
      $hidden = array('day'=>$day,'project_id'=>$project_id);
      echo form_open('',array('role'=>'form'),$hidden);
    ?>

    <div class="form-inline pos-control-menu cal-card-hover">
      <h2>Project: <?php echo $project_name;?></h2>
      <div class="form-group">
        <?php
        // this select is used to call other day results.
        echo form_label('Select day', '', array('style'=>'margin-
right:10px;'));
        echo form_error(''); // see set_rules($field_name[,
$field_label[, $rules]])
        $day = array( '0' => 'Monday', '1' => 'Tuesday', '2' =>
'Wednesday', '3' => 'Thursday', '4' => 'Friday', '5' => 'Saturday',
'6' => 'Sunday' );
        echo form_dropdown('', $day, $this->uri->segment(6),
'id="select_day" style="margin-right:10px;" class="form-control input-
lg"'); // http://stackoverflow.com/questions/11133540/adding-class-
and-id-in-form-dropdown
        ?>
      </div>
      <!-- Function buttons -->
      <button name="submit" type="submit" value="draft" class="btn
btn-primary btn-lg acal-confirm" title="Do not make available
publically">Save draft</button>
      <button name="submit" type="submit" value="publish"
style="margin-left:10px;" class="btn btn-primary btn-lg acal-confirm"
title="Save and publish the positions publically">Publish</button>
```

```
      <button name="submit" type="submit" value="delete"
style="margin-left:10px;" class="btn btn-primary btn-lg acal-confirm"
title="Delete the positions for this day">Delete day</button>
    </div>

    <!-- POSITION CARD CONTAINER -->
    <div class="acal-position-container">
    <?php
    if(count($project_positions) == 0)
    { // display default form
    ?>
    <div class="acal-card-container">

      <input type="hidden" name="id[]" value="">
      <div class="acal-card cal-card-hover">
        <div style="text-align:right;margin-bottom:10px;">
          <a href="javascript:;" class="remove-position-
card"><strong>Remove position </strong><span class="glyphicon
glyphicon-remove"></span></a>
        </div>
        <div class="row">
          <div class="form-group col-sm-6">
            <?php
            echo form_label('Title (required)','title[]');
            echo form_error('title[]');
            echo form_input('title[]','','class="form-control"
required');
            ?>
          </div>
          <div class="form-group col-sm-6">
            <?php
            echo form_label('Maximum participants
needed','max_vol[]');
            echo form_error('max_vol[]');
            $max_vol = array('name'=>'max_vol[]', 'type'=>'number',
'min'=>'1', 'max'=>'999', 'class'=>'form-control');
            echo form_input($max_vol);
            ?>
          </div>
        </div>
        <div class="form-group">
          <?php
          echo form_label('Description','description[]');
          echo form_error('description[]');
          echo form_textarea('description[]','','style="height:100px;"
class="form-control"'); // works just like form_input
          ?>
        </div>
      </div>

    </div>
    <?php
  } else { // populate the positions with the existing data
    $max_vol = array('name'=>'max_vol[]', 'type'=>'number',
'min'=>'1', 'max'=>'999', 'value'=>'', 'class'=>'form-control');

    foreach ($project_positions as $p)
    {
    ?>
    <div class="acal-card-container">
```

```
      <input type="hidden" name="id[]" value="<?php echo $p->id; ?>">
      <div class="acal-card cal-card-hover">
        <div style="text-align:right;margin-bottom:10px;">
          <a href="javascript:;" class="remove-position-
card"><strong>Remove position </strong><span class="glyphicon
glyphicon-remove"></span></a>
        </div>
        <div class="row">
          <div class="form-group col-sm-6">
            <?php
            echo form_label('Title (required)','title[]');
            echo form_error('title[]');
            echo form_input('title[]',$p->title,'class="form-control"
required');
            ?>
          </div>
          <div class="form-group col-sm-6">
            <?php
            echo form_label('Maximum participants
needed','max_vol[]');
            echo form_error('max_vol[]');
            $max_vol['value'] = $p->max_vol; // update the value key
            echo form_input($max_vol);
            //echo form_input('max_vol[]',$p->max_vol,'class="form-
control"');
            ?>
          </div>
        </div>
        <div class="form-group">
          <?php
          echo form_label('Description','description[]');
          echo form_error('description[]');
          echo form_textarea('description[]',$p-
>description,'style="height:100px;" class="form-control"'); // works
just like form_input
          ?>
        </div>
      </div>

    </div>
    <?php
    }
  } // end form generation
    ?>
  </div>

    <!-- Submit button -->
    <input type="button" id="add_position_card" style="margin-
top:15px;" class="btn btn-success btn-lg btn-block" value="Add
position">

    <?php echo form_close();?>
  </div>
</div>

<?php
  // create the controller uri base for the day select element
argument to append to
  $controller_uri = base_url().$this->uri->slash_segment(1).$this-
>uri->slash_segment(2).$this->uri->slash_segment(3).$this->uri-
>slash_segment(4).$this->uri->slash_segment(5);
```

```
?>
<script>
  // These need to be set globally since we can't set php in linked js
files
  // it's then used by the linked js file project_positions.js
  var controller_uri = "<?php echo $controller_uri ?>";
  // get the day of the select dropdown
  var select_day = document.getElementById('select_day'); //
select_day.value
</script>
```

In this view example, the “hidden” argument of the open_form helper function is demonstrated and shows how easy it is to create an array of hidden form inputs with dynamic values. The form_dropdown function from the form helper is also demonstrated, and uses the segment method from the URI class to pull a value from a URI segment position to act as the select dropdown’s default value. The view then goes on to decide which layout to display based on the count of project positions coming from the database. Finally, the $controller_uri variable uses the base_url function from the URL helper, along with the URI segment methods to create a URL to pass to an external jQuery script which is responsible for submitting a request when the user changes the day.

4.3.6 Calendar

The calendar part of the application allows the user to browse the dates a specific project runs on each month. It then allows the user to select a specific date and view the positions they can mark their availability for. It finally allows the user to add/remove their desire to participate for a position on a particular date. There will be further functionality added to this area of the application at a later date.

Title	Calendar
Actor	User, admin, super admin
Scenario	1. Actor selects one or more unchecked availability steps. 2. Actor selects to save/commit the input. 3. System adds the record, and refreshes the page, confirming the commit.

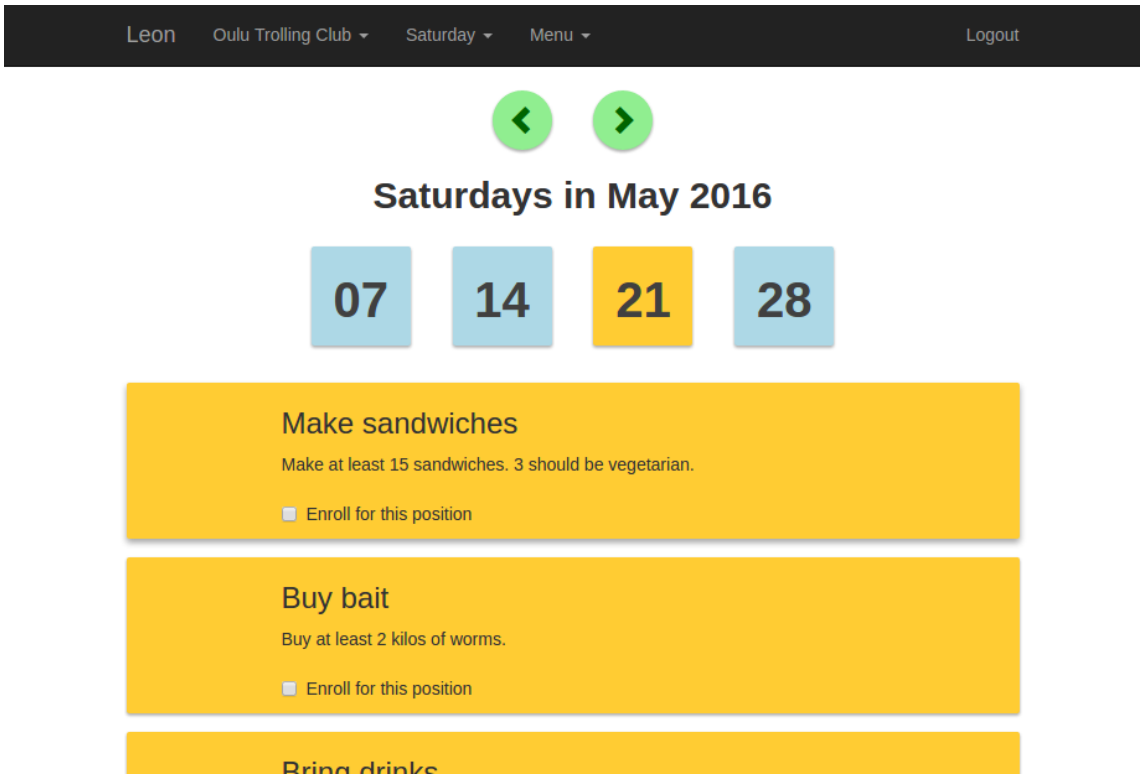

FIGURE 9. The calendar view

Controller

```
class Calendar extends Admin_Controller
{
public function __construct()
{
  parent::__construct();
  $this->data['page_title'] = 'Calendar';
  $this->load->model('menu_model');
  $this->load->model('projects/calendar_model');
  $this->data['before_body'] = '<script
src="'.site_url('../assets/js/acal/calendar_home.js').'"></script>';
}

// this controller is used as default application view after the user
has logged in
public function home($project = NULL, $day = NULL, $month = NULL,
$year = NULL, $date = NULL)
{
  if($year === NULL ) { $year = date("Y");  }
  if($month === NULL ) { $month = date("n"); }
  if($day === NULL ) { $this->session->selected_project_day = $day; }
// important for when user comes from link calendar/calendar/home with
no arguments

  // Use $this->session->selected_project_id
  // if $month = NULL, month is this monthYear
  // $year = NULL, year is this year
  $calendar_days = $this->getAllDaysInAMonth($year, $month, $day);
  $this->data['calendar_days'] = $calendar_days;
  $this->data['calendar_month'] = $month;
  $this->data['calendar_year'] = $year;
```

```
  // set the highlight date. create a date unless the user selects a
date -- then use that date
  if ($date === NULL) {
    $today = date('Y-m-d');
  }
  else {
    $today = $date;
  }

  // calendar day 'boxes' with the next-time/project high-lighted with
a special class
  foreach ($calendar_days as $value) {
    $is_next_time = $this->dateDiff($today, $value->format('Y-m-d'));

    if ($today == $value->format('Y-m-d')) {
      $this->data['is_next_date'] = $value->format('Y-m-d');
    }
    elseif ($today < $value->format('Y-m-d') && $is_next_time < 7) {
      $this->data['is_next_date'] = $value->format('Y-m-d');
    }
  }

  if($date !== NULL) {
    // user has selected a date -- request the positions form for that
date
    $this->data['calendar_positions'] = $this->calendar_model-
>get_positions($this->session->selected_project_id, $day, $date);
  }
  elseif($day !== NULL) {
    // user has arrived in the default view and the next date is
visible and highlighted.
    // request the positions form for the next available date
(hightlighted date).
    if (isset($this->data['is_next_date'])) {
      $date = $this->data['is_next_date']; // highlighted date
      $this->data['calendar_positions'] = $this->calendar_model-
>get_positions($this->session->selected_project_id, $day, $date);
    }
  }

  // form validation
  $validate = $this->calendar_model->validate['update_position'];
  $this->form_validation->set_rules($validate);
  // handle form submission
  if($this->form_validation->run() === FALSE) {
    $this->render('calendar/calendar_home_view');
  } else {
    $this->calendar_model->update_users_positions($this->input-
>post('date'));

redirect("calendar/calendar/home/$project/$day/$month/$year/$date");
  }

}

// return the difference between two days. Returns: number of days.
private function dateDiff($start, $end) {
  $start_ts = strtotime($start);
  $end_ts = strtotime($end);
  $diff = $end_ts - $start_ts;
```

```
  return round($diff / 86400);
}

private function getAllDaysInAMonth($year, $month, $day) {
    // loop through the array and output the days as 'date' => 'day',
'4' => 'Monday'
    // sort the keys in numerical order
    $dateString = 'first '.jddayofweek($day,1).' of '.$year.'-
'.$month;

    if (!strtotime($dateString)) {
        throw new Exception('"'.$dateString.'" is not a valid
strtotime');
    }

    $startDay = new DateTime($dateString); // datetime object

    $days = array();
    // http://php.net/manual/en/datetime.modify.php
    while ($startDay->format('Y-m') <= $year.'-'.str_pad($month, 2, 0,
STR_PAD_LEFT)) {
        $days[] = clone($startDay);
        $startDay->modify('+ 7 days');
    }

    return $days;
}

}
```

The calendar controller demonstrates an example of using private methods to break up a more complicated URI segment method. This is not a feature of CodeIgniter itself, but rather an example of better organisation within a controller. The “home” controller (named because it is used as the default controller after the user has logged in) takes many arguments. The first two, $project and $day are responsible for getting the correct project and day. The $day, $month and $year are used by the calendar navigation in the view, and also for fetching the positions from the database. Finally, the $date argument is specifically used to post form data to the calendar model. If there are arguments missing from the URI, the controller checks and acts appropriately by filling in default data, for example, the current date.

Model

```
class Calendar_model extends CI_Model
{
  public $current_user_id; // current user's id

  public function __construct()
  {
    parent::__construct();
    $this->current_user_id = $this->ion_auth->user()->row()->id; //
get their id.
  }
```

```
  public $validate = array(
    'update_position'=>
      array(
        'attendee_checkbox' =>
array('field'=>'attendee_checkbox[]','label'=>'','rules'=>'trim|intege
r')
      )
  );

  // Display a list of *published* *project* *positions* for a
position *day* and *date*.
  // If no date is specified, the current date is used.
  public function get_positions($project_id = NULL, $day = NULL, $date
= NULL)
  {
    /*
      Equivalent SQL query:
      SELECT title, description, max_vol, IF(up.id IS NULL, "FALSE",
"TRUE") volunteered
      FROM positions AS p
      LEFT JOIN users_positions AS up
      ON p.id = up.position_id
      AND up.user_id = user_id
      AND up.calendar_date = '2016-04-04' # note the quotes
      WHERE p.day = 0
      AND p.project_id = 3;
    */
    if ($date === NULL) { echo '$date === NULL. Cant call
calendar_model database'; exit(); }
    // current_user_id comes from ion_auth session data.
    $this->db->select('up.id AS up_id, p.id AS pos_id, title,
description, max_vol, IF(up.id IS NULL, "FALSE", "TRUE")
volunteered');
    $this->db->from('positions AS p');
    $this->db->join('users_positions AS up', "p.id = up.position_id
AND up.user_id = \"$this->current_user_id\" AND up.calendar_date =
\"$date\"", 'left');
    $this->db->where('p.day', $day);
    $this->db->where('p.project_id', $project_id);
    $query = $this->db->get();

    return $query->result(); // return the rows selected
  }

public function update_users_positions($date = NULL)
{
  if (!isset($date)) { echo "update_users_positions() is missing
args."; exit(); }
  $posted = $this->input->post(NULL, FALSE); // assign app POST vars
to $posted

  // delete previous rows from users_positions table
  $up_id_array = $posted['up_id'];

  foreach ($up_id_array as $up_id) {
    $this->db->delete('users_positions', array('id' => $up_id));
  }

  // update with latest data
  $attendee_checkbox_array = $posted['attendee_checkbox'];
```

```
  foreach ($attendee_checkbox_array as $attendee_checkbox) {
    if (!empty($attendee_checkbox)) {
      $data = array(
        'user_id' => $this->current_user_id,
        'position_id' => $attendee_checkbox,
        'calendar_date' => $date
      );
      $this->db->insert('users_positions', $data);
    }
  }

  $this->session->set_flashdata('updated', '<div class="alert alert-
success"><strong>Success! </strong>Positions have been
updated</div>');
}

}
```

In the calendar model, the get_positions model demonstrates a more complex query, like a left join, is possible with CodeIgniter's query builder.

5 CONCLUSION

Since the introduction of 3G mobile networks, the majority of people have the internet on their mobile devices as well as on PCs. This has created an always online culture. Business and society have taken advantage of this and therefore require fast, secure, and maintainable solutions to their development needs. Web frameworks achieve this by allowing developers with only intermediate knowledge and experience to cater for these needs. Thanks to libraries, helper functions, the MVC pattern, URI segment schemes, and active record pattern, making a web application is faster, easier, more maintainable, and more secure when developing using a web framework, but it's worth noting it is not a complete solution. There are still major parts of the application it will not help with, such as object modelling, database design, responsive design, styling, and front-end scripting. Thankfully, there are tools and methodologies that can ease that burden. The author acknowledges UML, MySQL Workbench, Bootstrap and jQuery technologies, which helped migrate these difficulties.

The author approached the development goal as a single (freelance) developer, and therefore could not evaluate the "separation of concerns" benefit the MVC pattern claims it has for developers working as part of a team. However, it is not too far stretched to imagine that all the roles the author filled would have been successfully managed by a team member working on one specific functioning area of the application i.e. the model, view or controller.

As this thesis has only demonstrated and therefore evaluated one MVC framework, it is also important to mention there are other frameworks available. While frameworks vary in features and solutions, they also vary in programming language. This can be essential for an individual developer's success when attempting to quickly master a framework, although learning a new language could also be a rewarding and beneficial if the developer can be more productive in the new language.

It is hoped that this thesis has helped a developer or company interested in MVC frameworks to gauge their effectiveness and deduce whether or not it would be relevant for future projects. It is also hoped that it accelerated the learning process of such a framework. The author himself has strengthened his understanding of MVC frameworks by writing this thesis, and is more confident

to recommend it as a solution to projects similar in complexity as the one demonstrated in this thesis.

6 DISCUSSION

In the beginning of the thesis it was stated that the intention was to document the value of writing a modern web application from a freelance perspective, using the newly updated CodeIgniter3 PHP framework. Before the task was undertaken, I had had previous experience with procedural PHP, and had recently taken a course module on web application development using the ASP.NET MVC web development framework. This was followed by professional work practice training where myself, and other students, developed a web application for a local company using ASP.NET MVC. I found it took a surprisingly short amount of time to learn CodeIgniter, although my previous experience with ASP.NET had probably helped. I appreciated the rich amount of classes and helper functions that CodeIgniter features, and think this makes for a good extension of the PHP language. I favour the MVC pattern as it seems logical to separate code in accordance to this pattern, which makes navigating around the codebase undemanding as there is less code than on a single page to understand. I also feel that using URI segments to interact with class methods over a network to be more direct, and therefore more helpful when finding exact paths of execution, while the system of routing allows the developer to mask URL complexities and verbose method calls —ultimately benefiting search engines and the end user. As I aspire to be a web developer and not a web security expert, I appreciate the built in features the framework offers to better protect the system, and also its users, from various attacks and exploits. This allows a developer more time to focus on development.

During the early stages of the development, UML methodologies like the activity diagram and use cases were produced. Although UML is not part of a framework, it is an important part of the design process as it clarifies the requirements between client and developer, and serves as a good start point when creating classes and class methods.

Minor updates are often available for frameworks. While some of these updates just improve or add new features, some are security patches and are therefore essential. This involves active maintenance of existing applications, but mostly doesn't require more than a command line update, or in the case of CodeIgniter, overwriting a system folder. However, during major version updates, frameworks can completely change how they do things. As older versions of the framework become legacy code, eventually it will reach its end-of-life, and a framework development team will stop providing security updates. Major updates are often rewrites of the

codebase and may implement new ways of doing things. This often requires changing parts of an existing application in order for it to run without error on updated versions.

The complete prototype application was successfully built, but I have only included parts of the application in the thesis to demonstrate the features of the framework. This was for brevity and to avoid repetition of code that fundamentally does the same thing. After the thesis is complete, I will release the full project code to GitHub after some further development. To find the project code, it is advised to search GitHub by developer name.

7 REFERENCES

Amazon. 2016. *What Is Amazon EC2?* Accessed 5 18, 2016. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

Ambler, S. 2001. “The Object Primer.” 170-172. Cambridge University Press.

Auer, Liisa, Jouni Juntunen, and Pekka Ojala. 2014. “OpixProject.” *Opiskelija Projektit.* Accessed 5 18, 2016. http://opixproject.opiskelijaprojektit.net/images/mindtrek_auerjuntunenojala.pdf.

Berkeley, UC. 2004. *Model-View-Controller: A Design Pattern for Software.* 06. https://ist.berkeley.edu/as-ag/pub/pdf/mvc-seminar.pdf.

Borodescu, C. 2013. *Web Sites vs. Web Apps: What the experts think.* Accessed 5 18, 2016. http://www.visionmobile.com/blog/2013/07/web-sites-vs-web-apps-what-the-experts-think/.

British Columbia Institute of Technology. 2016a. *CodeIgniter URLs.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/urls.html.

—. 2016b. *Controllers.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/controllers.html.

—. 2016c. *Design and Architectural Goals.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/overview/goals.html.

—. 2016d. *Email Class.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/libraries/email.html.

—. 2016e. *Form Validation.* Accessed 5 18, 2016. https://codeigniter.com/user_guide/libraries/form_validation.html#changing-the-error-delimiters.

—. 2016f. *Helper Functions.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/helpers.html.

—. 2016g. *Installation Instructions.* Accessed 5 18, 2016. https://codeigniter.com/user_guide/installation/index.html.

—. 2016h. *Models.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/models.html.

—. 2016i. *Security.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/security.html.

—. 2016j. *URI Routing.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/routing.html.

—. 2016k. *Using CodeIgniter Libraries.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/libraries.html.

—. 2016l. *Web Page Caching.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/caching.html.

—. 2016m. *Welcome to CodeIgniter.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/welcome.html.

—. 2016n. *Views.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/views.html.

—. 2016o. *Creating Core System Classes.* Accessed 5 18, 2016. http://www.codeigniter.com/user_guide/general/core_classes.html#extending-core-class.

—. 2016p. *Application Flow Chart.* Accessed 05 18, 2016. http://www.codeigniter.com/user_guide/overview/appflow.html.

Cheshire, J. 2009. *ASP.NET MVC: What is it and should I use it?* Accessed 5 18, 2016. https://blogs.msdn.microsoft.com/webtopics/2009/09/01/asp-net-mvc-what-is-it-and-should-i-use-it/.

Daigle, L. 2016. *30 Years of TCP -- and IP on everything!* Accessed 5 18, 2016. https://www.internetsociety.org/blog/2013/01/30-years-tcp-and-ip-everything.

Davis, I. 2008. *What Are The Benefits of MVC?* Accessed 5 18, 2016. http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc.

Dimitrios, T. 2010. *What is register_globals and why is it a security risk?* Accessed 5 18, 2016. https://tournasdimitrios1.wordpress.com/2010/11/09/what-is-register_globals-and-why-is-it-a-security-risk/.

Edmunds, B. 2016. *Ion Auth Documentation.* Accessed 5 18, 2016. http://benedmunds.com/ion_auth/#install.

EllisLab. 2014a. *ChangeLog.* Accessed 5 18, 2016. https://ellislab.com/codeigniter/user-guide/changelog.html.

—. 2016b. *CodeIgniter.* Accessed 5 18, 2016. https://ellislab.com/codeigniter.

HackerOne. 2016. *CodeIgniter security research.* Accessed 5 18, 2016. https://hackerone.com/codeigniter.

Klimushyn, M. 2015. *Web Application Architecture from 10,000 Feet, Part 1 – Client-Side vs. Server-Side.* Accessed 5 18, 2016. https://spin.atomicobject.com/2015/04/06/web-app-client-side-server-side/.

Knight, D. 2014. *Personal Computer History: The First 25 Years.* Accessed 5 2016, 18. http://lowendmac.com/2014/personal-computer-history-the-first-25-years/.

Magic web solutions. 2016. *The benefits of web-based applications.* Accessed 5 18, 2016. http://www.magicwebsolutions.co.uk/blog/the-benefits-of-web-based-applications.htm.

Oracle. 2016. *WorkBench.* Accessed 5 18, 2016. https://www.mysql.com/products/workbench/.

Srinivasan, A. 2014. "Cloud Computing." Chap. 15.5. Pearson India.

Sturgeon, P. 2010. *CodeIgniter Base Classes: Keeping it DRY.* Accessed 5 18, 2016. https://philsturgeon.uk/codeigniter/2010/02/08/CodeIgniter-Base-Classes-Keeping-it-DRY/.

The Apache Software Foundation. 2016. *Apache Virtual Host documentation.* Accessed 5 18, 2016. https://httpd.apache.org/docs/current/vhosts/.

The Lean Startup. 2016. *The lean startup methodology.* Accessed 5 18, 2016. http://theleanstartup.com/principles.

Thorpe, S. 2011. *Untangling MVC with CodeIgniter.* Accessed 5 18, 2016. https://www.sitepoint.com/untangling-mvc-with-codeigniter/.

US FCC. 2005. *Making the Connections .* Accessed 5 18, 2016. https://transition.fcc.gov/omd/history/internet/making-connections.html.

WC3. 2016. *Standards.* Accessed 5 18, 2016. https://www.w3.org/standards/.

Wikipedia. 2016a. *3G.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/3G.

—. 2016b. *Ajax (programming).* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Ajax_(programming).

—. 2016c. *Architectural pattern.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Architectural_pattern.

—. 2016d. *ARPANET.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/ARPANET.

—. 2016e. *CodeIgniter.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/CodeIgniter.

—. 2016f. *Cohesion (computer_science).* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Cohesion_(computer_science).

—. 2016g. *Comparison of web frameworks.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Comparison_of_web_frameworks.

—. 2016h. *Laravel.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Laravel.

—. 2016i. *Model–view–controller.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

—. 2016j. *Ruby on Rails.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Ruby_on_Rails.

—. 2016k. *Software design pattern.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Software_design_pattern#Classification_and_list.

—. 2016l. *Web application.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Web_application.

—. 2016m. *Web intelligence.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/Web_intelligence.

—. 2016n. *World Wide Web.* Accessed 5 18, 2016. https://en.wikipedia.org/wiki/World_Wide_Web.

Zickuhr, Kathryn, and Aaron Smith. 2012. *Digital differences.* Accessed 5 18, 2016. http://www.pewinternet.org/2012/04/13/digital-differences/.