CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

DEVELOPING A CMS PHP FRAMEWORK WITH SYMFONY

A thesis submitted in partial fulfillment of the requirements

For the degree of Master of Science in Software Engineering

by

Justin Khoi Bao Han-Nguyen

December 2016

The thesis of Justin Khoi Bao Han-Nguyen has been approved by:

Professor Richard Covington	Date Approved
Professor Li Liu	Date Approved
Professor Taehyung "George" Wang, chair	Date Approved

California State University, Northridge

PREFACE

This thesis is submitted for the degree of Master of Science in Software Engineering at the California State University at Northridge. The work described herein was done under the supervision and advice of a committee headed by Professor Taehyung "George" Wang in the department of Computer Science between January 2016 and December 2016.

This work is done for the purpose of creating an easy to use PHP framework for web developers looking to develop Content Management Systems (CMS). Since most PHP frameworks are made generically, there is nothing focused for CMS development. This work is original to the best of my knowledge, except where acknowledgements and references are made.

Justin Khoi Bao Han-Nguyen

December 2016

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Professor Taehyung “George” Wang, for his support and supervision. I would like to thank my committee members, Professor Richard Covington and Professor Li Liu, for taking their time to be available to help me with my thesis, providing me their valuable input and ideas.

I would like to thank the friends and peers whom I’ve met at the College of Engineering and Computer Science at the California State University at Northridge for providing me with

Finally, I would also like to thank my family and friends for supporting me with my decision to give up my pursuit of a medical degree in order to pursue my passion in programming and engineering.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

DEVELOPING A CMS PHP FRAMEWORK WITH SYMFONY

By

Justin Khoi Bao Han-Nguyen

Master of Science in Software Engineering

Open-source programming is very popular in the computer science world today. There are various open-source technologies out there that are currently in use – libraries, frameworks, APIs, etc. Web developers who develop in PHP have a choice for frameworks to use – most of which are open-source. However, there is a small amount, if not none, of open-source frameworks that are focused on CMS solutions.

This project sought to develop a CMS focused framework using Symfony, a PHP framework, as the base. Development was supported by a tutorial and Symfony's detailed documentation, aided by the use of mostly open-source software, such as Toad for MySQL, Atom.io, and XAMPP.

CHAPTER 1

INTRODUCTION

There are many PHP frameworks that are popular and widely used to develop websites and applications. The most commonly used open-source PHP frameworks are Laravel, CodeIgniter, CakePHP, and Symfony; however, these are generic and do not focus on any particular type of system. Although there are commercial PHP frameworks that are focused on delivering Content Management Systems or other types of management systems, the objective of this project is to create a framework similar to those commercial frameworks and make them available via open-source.

1.1 The Objectives

The main objective of this project was to create a functional PHP framework that could be used by developers to quickly and efficiently develop CMS solutions. The framework needed to pass these criteria:

- The CMS framework shall consist of multiple standalone modules to facilitate development.
- The modules shall be generic, usable for different types of data.
- The modules shall not require other modules to function.
- The framework shall be used to develop a basic CMS webpage.
- This basic CMS webpage shall have used all of the functionality listed in our documentation.
- The developed webpage shall perform every function without error.

- The amount of time needed for the development of the webpage using the framework shall be recorded and compared to average values.

1.2 The Motivation

The motivating factor behind this development project is the lack of open-source PHP frameworks focused on business applications. Commercial PHP frameworks have a variety of costs and could be useful in developing applications that require CMS services, but there are many PHP developers who do not have the capital to buy licenses for these commercial PHP frameworks. Commercial PHP frameworks are also part of a niche market.

As PHP developers, we wanted to develop this new framework in order to provide ourselves, as well as other PHP developers, a foundation that we would lay down for all future projects, used for clients or companies. With this project complete, all of our future projects would have their foundations laid out before the development process even began.

1.3 The Approach

The Symfony PHP framework can be leveraged to create a more focused and specialized framework. To facilitate this, the documentation for Symfony also has an exceptionally detailed tutorial to get the development started. We utilized this tutorial to create the backbone of the framework and then expanded on it with our own designs and classes to produce the modules with the functionality to meet our requirements.

Since the finished product would be a framework, we believed the best testing process for each module of this project would make use of the framework to develop a

simple website that utilized the completed module. We thoroughly tested each module to ensure that it met the user stories and our requirements before beginning development on the subsequent modules.

When all modules were developed and each module was thoroughly tested, we created a website that required the functionality of every module and verified that there were no errors, bugs, or functional anomalies.

1.4 The Tools

We used Symfony PHP as a base to begin development of this CMS focused framework. We also used a simple text editor called Atom.io to write our code and develop the framework. To test our code, we created a local LAMP-like stack using the software XAMPP. XAMPP is an open-source package that installs Apache, MariaDB, PHP, and Perl onto Windows, Linux, or Mac OS X. For our database testing, we used Toad for MySQL. We also used Dreamweaver CS 5 to develop a test web site that utilizes the project framework and then we debugged the web site with the Google Chrome web browser's developer tools.

1.5 The Next Chapters

In the next chapter, we will be reviewing the popular PHP frameworks and explaining our decision for not using those frameworks for creating CMS solutions. Chapter 3 discusses the requirements the project must meet. We will detail our design of the solution in Chapter 4.

Beginning with Chapter 5, we will be discussing the various testing suites and our testing process, laying out the reasons we made those decisions. We will expand on the tools we used in Chapter 6, fully explaining Symfony PHP, Atom.io, and XAMPP. Chapter 7 will discuss the value this framework provides to the end user, the web developer. Finally, we will conclude with our findings and the results of our work in Chapter 8.

CHAPTER 2

REVIEW OF POPULAR PHP FRAMEWORKS

There is a plethora of PHP frameworks available in the PHP community, such as Laravel, CakePHP, CodeIgniter, Symfony, and Zend Framework. These frameworks make development of web applications and web sites faster as they provide an implementation of a Model-View-Controller (MVC) architecture pattern, making the routing easy to set up by changing some configuration settings. Of all the available frameworks, the most popular frameworks are Laravel, CakePHP, CodeIgniter, and Symfony. We will review and discuss Laravel, CakePHP, and CodeIgniter in this chapter, and explain why we chose Symfony as our base framework to build upon.

2.1 The Common Problem

The PHP frameworks we are discussing all share a common problem: they were developed to hasten the development of web applications and websites for all applications, so they are all generic and without focus in any particular area. Because of this, using these frameworks would prove to be faster for certain developers over others. A CMS solution developer, for example, would need more time to complete his project than a simple static website developer.

2.2 Laravel Framework

Currently, Laravel is considered to be the most popular PHP framework. [Winspire and Hitesh] In order to understand its popularity in the PHP community, we must review

the features. Hitesh and Winspire Web Solution described Laravel as the number one PHP framework, but they each discussed Laravel's "number one" status in different aspects – Hitesh looked at it from a viewpoint of popularity while Winspire discussed it from the viewpoint of the "best" functionally. [Winspire and Hitesh]

The official website for Laravel discussed how Laravel's documentation and screencasts provide the developer with tutorials and help with getting started – this allows the developer or development team to deliver "rapid … [and] amazing" applications. [Laravel] In today's software development culture, rapid delivery of quality applications is a main focusing point, especially with more technologies implementing continuous integration and continuous delivery processes. [CGI] Laravel is able to fit in today's culture because it can facilitate a rapid delivery, even if it cannot be used with the continuous delivery process.

Rapid delivery of applications is not the only feature that makes Laravel popular and respected. There is an inherent database version control provided by Laravel. It uses "restful routing … [which] connects resources smoothly." Laravel is also bundled with Eloquent, an object-relational mapping tool, and Blade, a templating engine. With built-in unit testing, Laravel allows developers to develop anything from simple websites to enterprise-level applications. [Winspire and Hitesh] These benefits were taken into consideration when we were deciding which framework to use.

2.3 CakePHP Framework

Like Laravel, CakePHP is an open-source PHP framework designed for rapid development. CakePHP was one of the more popular frameworks in the period between

2006 and 2010, but the popularity of CakePHP, along with Zend Framework, CodeIgniter, and other frameworks, began to decline as the popularity of Laravel began to increase. [Winspire] Even with the decline, CakePHP 3.0 is considered to be within the top 7 PHP frameworks by the community. It is still widely used by developers and companies, such as Bavarian Motor Works, blendtec, and Billabong. [CakePHP]

CakePHP, at its core, offers the same features as Laravel, such as MVC conventions, security, and built-in functions: database access, caching, validation, authentication, etc. [CakePHP] Laravel is powerful due to the availability of its templating engine, Blade, which allows developers to build the product faster, while CakePHP does not have a templating engine. Instead, it provides code generation and scaffolding, allowing developers to build prototypes quickly; Laravel does not have code generation. [CakePHP]

The features provided by CakePHP support more traditional software development processes with the rapid prototyping capabilities. In contrast, the features provided by Laravel support more modern software development processes, using both continuous integration and delivery such as Agile and Scrum. Since we were not focused on prototype building, we decided that the benefits of using CakePHP were not attractive enough for the project.

2.4 CodeIgniter Framework

Unlike CakePHP, CodeIgniter's popularity did not change drastically. CodeIgniter is still considered to be in the top 5 PHP frameworks. [Winspire] The reason it kept a steady grip on its popularity is the ease of learning and becoming proficient with it. According to CodeIgniter, it has "no restrictive coding rules, … no need for templating language, …

[and focuses on] simple solutions over complexity." [CodeIgniter] The simplicity, in conjunction with its detailed documentation, makes this framework a great beginner level PHP framework.

While CodeIgniter encourages the MVC pattern, it does not force developers to conform to it in the same manner as CakePHP and Laravel. It also allows developers to use their own coding and naming conventions provided that they do not create class name conflicts with the built-in modules. With virtually zero configuration needed, CodeIgniter can simply be downloaded – 2 MB download size including the user guide – and used on most hosting platforms. CodeIgniter's exceptionally high performance and maintenance, alongside its other features, are other reasons it is considered to be the best framework for beginners. [Winspire]

Because CodeIgniter was the first PHP framework we had ever used, we felt more comfortable with the idea of utilizing it for our project. The familiarity with CodeIgniter was one of the main reasons we had placed it on our list of possible frameworks we could leverage, even though we considered the features of Laravel and CakePHP to be superior to CodeIgniter. We also thought that the performance and maintainability of CodeIgniter made it a competitor in our search.

2.5 Why Symfony?

Ultimately, we made our decision based on the needs of the project, choosing the Symfony framework as our "canvas." So, why did we choose Symfony? It features similar built-in features as the other frameworks discussed above. However, the deciding factor was the flexibility of Symfony to be adapted – Symfony lets developers customize every

part of the framework to their specifications. [Symfony] This was in line with our project and fulfilled our needs the best compared to the other frameworks – the documentation contained a step-by-step tutorial on how to create your framework by customizing the base framework.

CHAPTER 3

THE REQUIREMENTS

Software engineering has implemented many different software development processes, from traditional waterfall to Agile and Scrum. However, all of the development processes have a phase or activity of gathering the software requirements and specifications. This is the step in which the development team meets with the customers or users to determine what they are expecting from the software. The requirements are formally written down and separated into functional and non-functional requirements.

3.1 Who Are the Users?

In the PHP community, open-source tools are almost a necessity. Most of the available PHP frameworks are open-source, which makes them easier for developers to access. The PHP frameworks previously discussed in Chapter 2 are generic, so developers need to customize them when they incorporate a framework. This project's goal was to develop niche framework for CMS developers. Therefore, the users of our finished product will be web developers using the PHP language to develop CMS solutions. We will be the first users of this product, making our CMS development work much simpler to rapidly build for current and future clients.

3.2 Gathering Requirements

Working with fellow CMS developers and coworkers, we discussed the issues that come with developing a CMS solution using the common popular PHP frameworks.

Together, we came up with a list of inadequacies that we felt would make our development work easier. Line by line, we transformed each inadequacy into a requirement, if possible. We will discuss these requirements in the following section.

3.3 Functional Requirements

These requirements were compiled together with fellow CMS developers and coworkers. The functional requirements for this project are:

- The framework must provide an API, or class, to access the database through MySQL.
- The database API must allow the developer to set up the database connection dynamically.
- The database API must allow table creation and revisions through an API call.
- The database API must provide API calls for the different types of queries.
- The database API must provide API calls for manipulating views and procedures.
- The database API should be made generic to accommodate any database design.
- The database API should be extendable.
- The database API must be secure from SQL injection.

3.4 Non-functional Requirements

The non-functional requirements of this project are:

- The database API should have a faster processing time.
- The database API should have robust documentation.

- The database API should not change the innate database engine of the Symfony framework base.

These requirements were used to write the specifications for the framework design. We will discuss our design in the following chapter.

CHAPTER 4

FRAMEWORK DESIGN

From the requirements, we fine-tuned the specifications, then we used the specifications to design the database API to fulfill all of the "must have" functional requirements, then used UML diagrams to depict our designs. This chapter will show the finalized design with an explanation for every design decision we made.

4.1 Design Decisions and Structure

The finished framework in Symfony's tutorial can only handle requests and routing, but other functionalities can be added to it as needed. We decided that we only needed to build upon the initial tutorial framework. Figure 4.1 shows the UML class diagram for the framework we created. CMS solutions are built on top of databases, so the essential functions are database queries and construction, so we made sure that our framework could handle all of the most common database functions, in a generic manner.

In conjunction with the framework, we needed a class to encapsulate the database connection, as pictured in Figure 4.2 below. The class needs a variable to contain the path to the configuration file and a variable to contain the database connection object. We made the decision to only support MySQL for the first release, so the database connection object is of type mysqli_link, created by the mysqli PHP system class.

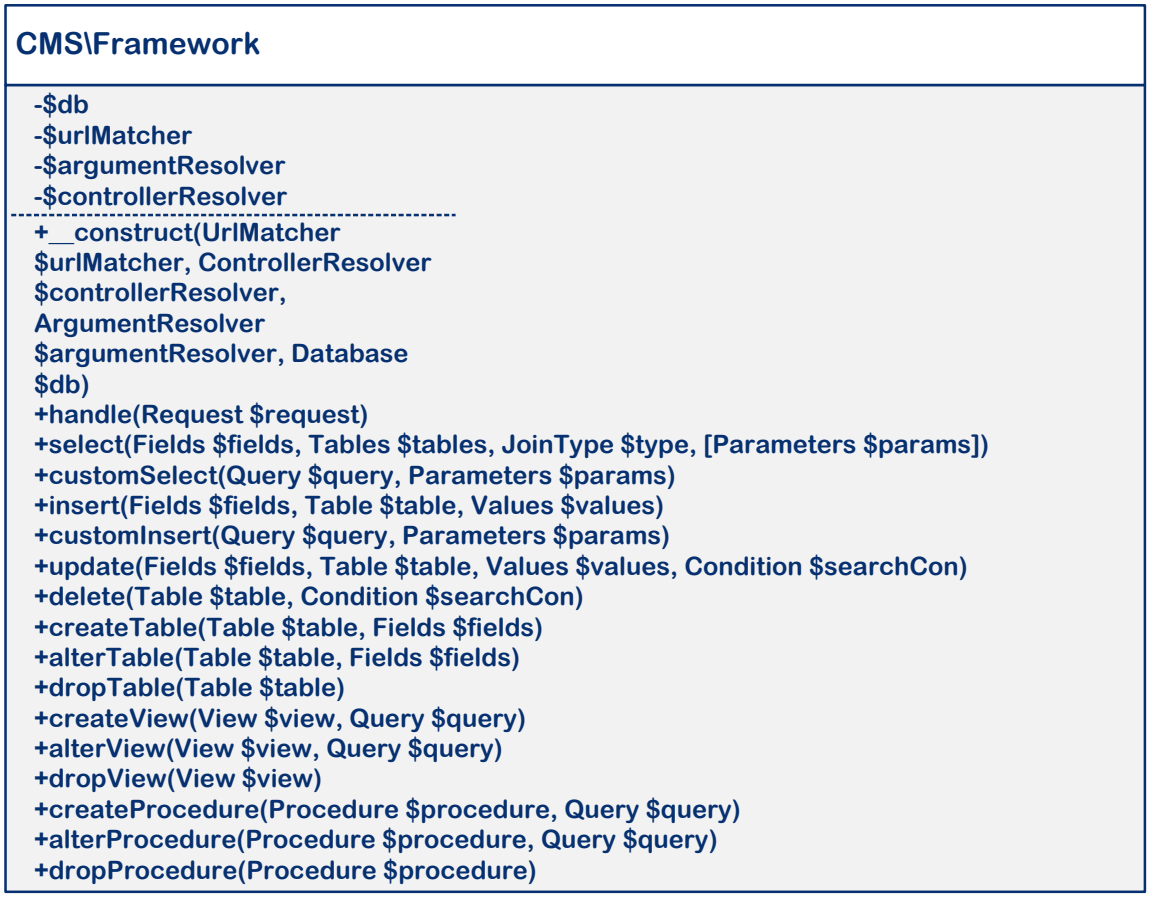

Figure 4.1 CMS Framework UML Diagram

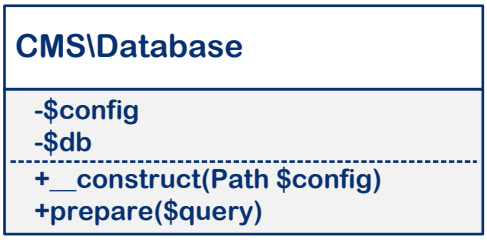

Figure 4.2 CMS Database Object UML Diagram

The framework also requires a controller for the MVC pattern implementation of Symfony. We kept the default model, controller, and view in addition to adding a custom controller and view, providing a user interface for the developer to initially set up the database, for the release. Once the framework has been "installed" to their machine, the developers can add their own controllers and views when they start the development on

their CMS solutions. The developers will need to reference Symfony's documentation if they have any questions on how to implement their own models, controllers, and views.

4.2 High Level Overview

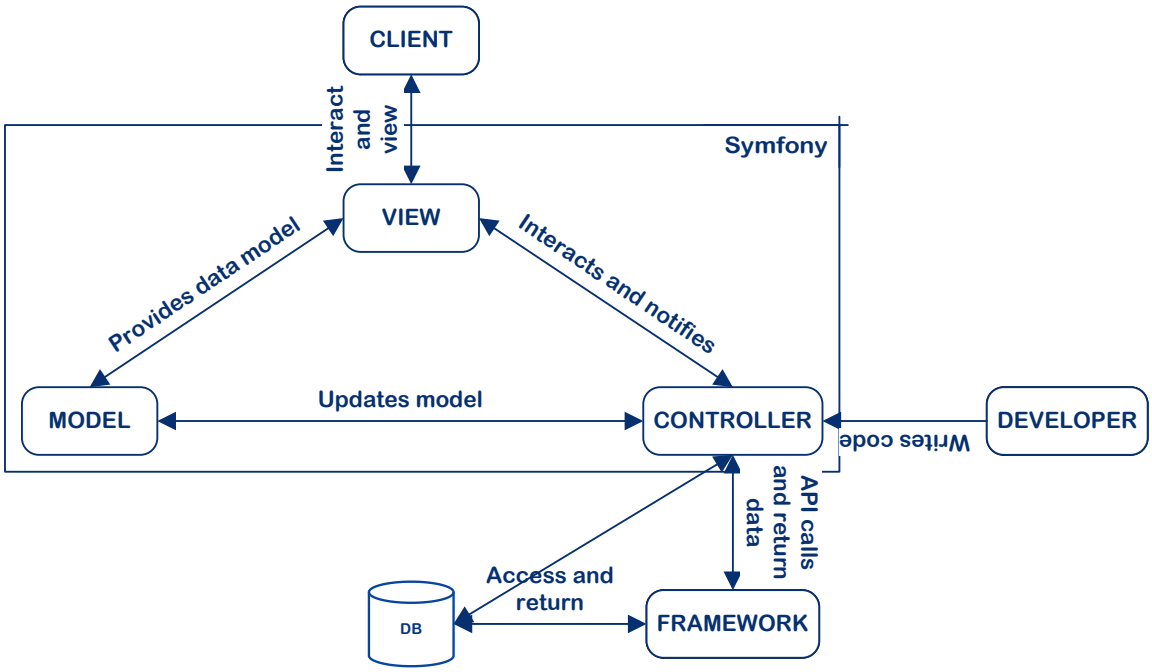

Figure 4.3 High Level Overview of the Framework

Figure 4.3 (above) depicts the high level overview of the framework developed for this project. Symfony and its MVC architecture is the base of the framework – the CMS framework provides an API for the controllers of the Symfony framework to perform the database functions. The end user for this framework is the web (application) developer. The developer interacts with the API through custom controllers that the Symfony MVC architecture will use to display the views and update the models. Only the framework and the controllers will be able to access the database, with the controller updating the model and database as changes are made by the client through the view.

4.3 Inside the Code

The CMS\Database class contains the database connection and the database configuration information. When a controller handles an event, such as table creation, the CMS\Database class is instantiated with a file path passed into the constructor to connect to the database configured in the configuration file. After the connection has been established, the framework uses the connection create prepared statements to perform the SQL statements in a secure manner.

The framework must be instantiated for it to be usable – the instantiation code is depicted in Figure 4.4 below. The constructor for the framework requires a MySQL connection object as a parameter, so the CMS\Database class must be instantiated prior to instantiating the framework. Then the instance of the framework saves the connection within a protected member variable – this allows the user to make API calls through the instance without the need to establish a database connection every time the call is made.

```
$framework = new CMS\Framework($url, $contRes, $argRes)
```

Figure 4.4 Framework Instantiation Code

Each API call will generate a MySQL statement based on the functionality that it is responsible for – for example, the insert call will generate a MySQL insert statement using the parameters passed into it as a guide. A prepared statement is then created by the database object and the variables passed through the call are bound as parameters. This binding is what minimizes the risk of SQL injections. Figure 4.5 shows the partial code of how this process is implemented. After the parameters are bound to the prepared statement, the statement is then executed and error checking verifies whether the execution was a success or failure. The user interface will display the results of the verification to the user

so they will know if their API call was completed successfully or not. Appendix B is a longer excerpt of the Select API call.

```
$stmt = $this->db->prepare($query);
if($hasParams){
    $a_param = array();
    $a_param[] .= & $pstring;
    for($i < 0; $i < $pcount; $i++){
        $a_param[] = & $pvalue[$i];
    }
    call_user_func_array(array($stmt, 'bind_param'), $a_param);
}
```

Figure 4.5 Excerpt of code from Select API Call

The user is able to read the source code for our custom controller to learn how to use our framework within the controller. It is possible for them to make changes to the source code, but we hope that the end user will realize that any changes to our controller may cause the create table user interface to cease functioning.

4.4 Behind the Reasoning

We will now discuss our reasoning behind each design decision discussed in the previous sections. First, we decided to create the framework by using the framework built in the tutorial as a base and expanding it by adding functions to it. We made this decision because we needed to make up for time lost due to the gathering and refining of our requirements. We also made this decision because the tutorial had everything set up in a certain way and we did not wish to spend more time than necessary on essential parts.

The next decision we made was the decision to only support MySQL as the database query language. Since Oracle and MySQL have the highest market share right now [DB-Engines Ranking], we believed that the target audience, open-source developers,

would be more likely to use MySQL than any other database engine. Later, we would like to add support for other database engines, such as NoSQL, DB2, and PostgreSQL.

This framework would come with a custom controller and view that facilitates the creation of the tables in their defined database. The user interface for this view allows the user to define table and field names and the script behind the controller would create the table as We had made this design decision because we wanted to stay in line with the other frameworks since they all had default controllers and views in their release builds.

Symfony 3.0 requires PHP 5.5.9 or newer, which may pose an issue for some developers since migrating to newer PHP versions requires refactoring and, in some cases, complete code rewrite because some features have been deprecated or removed completely. However, we decided to use Symfony 3.0 and PHP 5.5.9 or newer because the users of this framework will, usually, be looking to begin new CMS projects, not updating, upgrading, or migrating existing projects. As such, we wanted the users to use recent technology for their new projects and not rely on old technology.

PHP has a mysqli class library for connecting to and using MySQL databases. The mysqli library has great tools and functions; however, in order to use some of these functions, the server hosting the PHP code must be running on mysqlnd (MySQL Native Driver). There are methods of verifying that mysqlnd is running – these methods can be referenced by the user through a web search. We decided to use the mysqlnd because it gave us more functions for dealing with prepared statements in PHP. In Appendix B, there is a portion after the execute where it gets the result from the query – the get_result function requires mysqlnd to be usable.

CHAPTER 5

TESTING PROCESS

To make sure the project was completed to our specifications, we needed a rigorous testing plan involving unit testing for our modules to verify that the functions were being called correctly and functioned as designed and developed. In order to accomplish this, we looked into PHP testing frameworks, such as Codeception, PHPUnit, and SimpleTest. The utilities offered by each framework were considered; however, the utilities added too much overhead to the project for the functionality we needed. The first few sections will discuss each testing framework and why we decided not to use them. Following that, we will provide a discussion of the testing process we did implement and an explanation on our reasoning for choosing this testing plan.

5.1 Codeception

Codeception is a PHP testing framework that is capable of acceptance testing, functional testing, and unit testing. [Codeception] We had no interest in acceptance testing because our project "users" were the developers and not a web application end user. We believed that the acceptance testing procedures covered in the Codeception documentation would not be helpful to us in our testing plan. While Codeception's acceptance testing can be run on any website and can test JavaScript and ajax requests [Codeception], our project needed only to test our CMS related functionality.

We looked at functional testing with Codeception and we thought that we could utilize it for our project. When we considered the interactions between our project and a

test application, we realized that it would create more overhead to test the test application with Codeception. We would have had to create two applications, one to call the framework objects and another to test that application – this lead to too much work so we decided to forego this plan.

Codeception was built upon PHPUnit and will have a similar usage to PHPUnit. If unit testing with Codeception is similar to PHPUnit, we believed that it would be easier to test our modules using PHPUnit instead. [Codeception]

5.2 PHPUnit

Since we knew that Codeception was built upon PHPUnit, we decided to review PHPUnit as a possible testing framework we could leverage for the purposes of testing our project's functionality. To do this, we perused the documentation for the possible test functionalities we could use. We focused our attention on the database testing API because our framework deals with managing the content that is saved in our database.

To our disappointment, the database testing API provided with PHPUnit required the usage of the table names and fields within the test code. [Bergmann] This was not a possibility for us because our system would allow the developer to use general functions to interact with the database. The functionalities include, but are not limited to, creating tables, changing tables, querying data, and deleting information. With the API functions available [Bergmann], we would not be able to test the functionalities of our framework. Due to this issue, we believed that we could not utilize PHPUnit to its fullest potential and that using PHPUnit would add more overhead to the project.

5.3 SimpleTest

SimpleTest is a PHP testing framework that can handle unit testing and web testing. Similar to the previously mentioned testing frameworks, we believed that we would be unable to using SimpleTest for our testing process. SimpleTest is presented as a tool to make the testing of the "common but fiddly PHP tasks, such as logging into a site, [easier]." [SimpleTest] The documentation for SimpleTest shows that the functionality of it is even simpler than that of PHPUnit. For this reason, we decided that SimpleTest would not be able to fulfill our needs and requirements.

5.4 Our Testing Process

The testing frameworks that we reviewed did not meet our needs and requirements so we ultimately had to come up with our own testing process. We had to separate our testing plan into sections, focusing on one test at a time. During the development of the framework, we needed to test the functionality of each subclass we created. To test each subclass, we created simple scripts, each script using a particular function, and executed them. Then we checked the database and/or output to make sure we arrived at our expected results. We executed these scripts multiple times to ensure there were no unexpected bugs. Once we were satisfied with the results, we would work on the next subclass.

The main objective of this project was to create a framework that alleviates developers' workloads when they are developing a CMS solution for a client or for their company, so we decided to test our framework by creating a simple CMS solution using the framework. This solution would have a simple textual user interface to keep the focus on the functionality of the framework. This allowed us to make sure there were no bugs or

errors. Since this solution is a functional application, we could use one of the testing frameworks mentioned above to run tests to make sure that the test system used the frameworks correctly, thereby giving us the expected results. We felt that running unit tests on our test system would prove to give us a low return on investment of time.

CHAPTER 6

TOOLS OF THE TRADE

The end product of this project is a PHP framework for web CMS development, so we needed a web server environment for development and testing, an IDE for coding, and a web browser to test the code. We used XAMPP to create the environment. We used Atom.io as our development text editor, Dreamweaver CS 5 as our testing IDE, and Google Chrome for our web browser as well as a debugging tool. For reference, we used the documentation for Symfony PHP found on their website. We will go into detail about their use and purpose in this chapter.

6.1 XAMPP

To save money and resources, we did not dedicate a server; instead, we used XAMPP to create a virtual Apache web environment on our Windows machine. XAMPP is an open source package that installs an Apache web server distribution with MariaDB, PHP, and Perl. The package can be downloaded from the Apache Friends web site, located at http://www.apachefriends.org/index.html, and installation is as easy as running an executable file. Once installed, the settings can be accessed through the control panel, shown in Figure 6.1 below. The settings can be configured by editing the *.ini files for each component using any text editor. [Apache Friends]

We chose XAMPP because there is a community available through the Apache Friends forum, allowing for support whenever a problem arises. The Apache Friends web site also recommended Stack Overflow as a resource to get answers and support. These

readily available resources for help combined with the cost-free nature of being an open source package made it an easy decision to use XAMPP for our project.

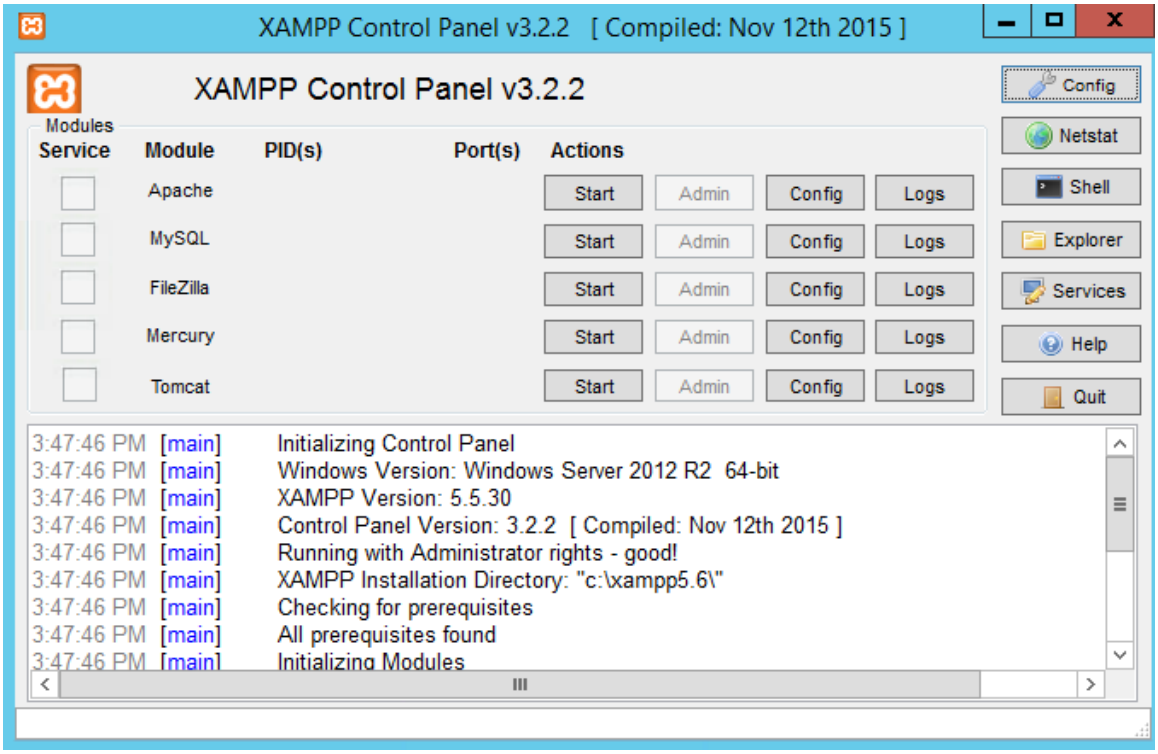

Figure 6.1 XAMPP Control Panel

6.2 Atom.io for Text Editing

For our text editor, Atom.io proved to be a very useful application. Shown in Figure 6.2 below, Atom.io is a text editor that color codes special keywords that are specific to various languages based on the extension of the file. Atom.io is also great for styling the code in an easy to read manner. An example of a file that was opened in Atom.io is shown in Figure 6.2 below. The file in the figure is a PHP script with the *.php file extension; as you can see, special keywords are color coded differently from variables and data.

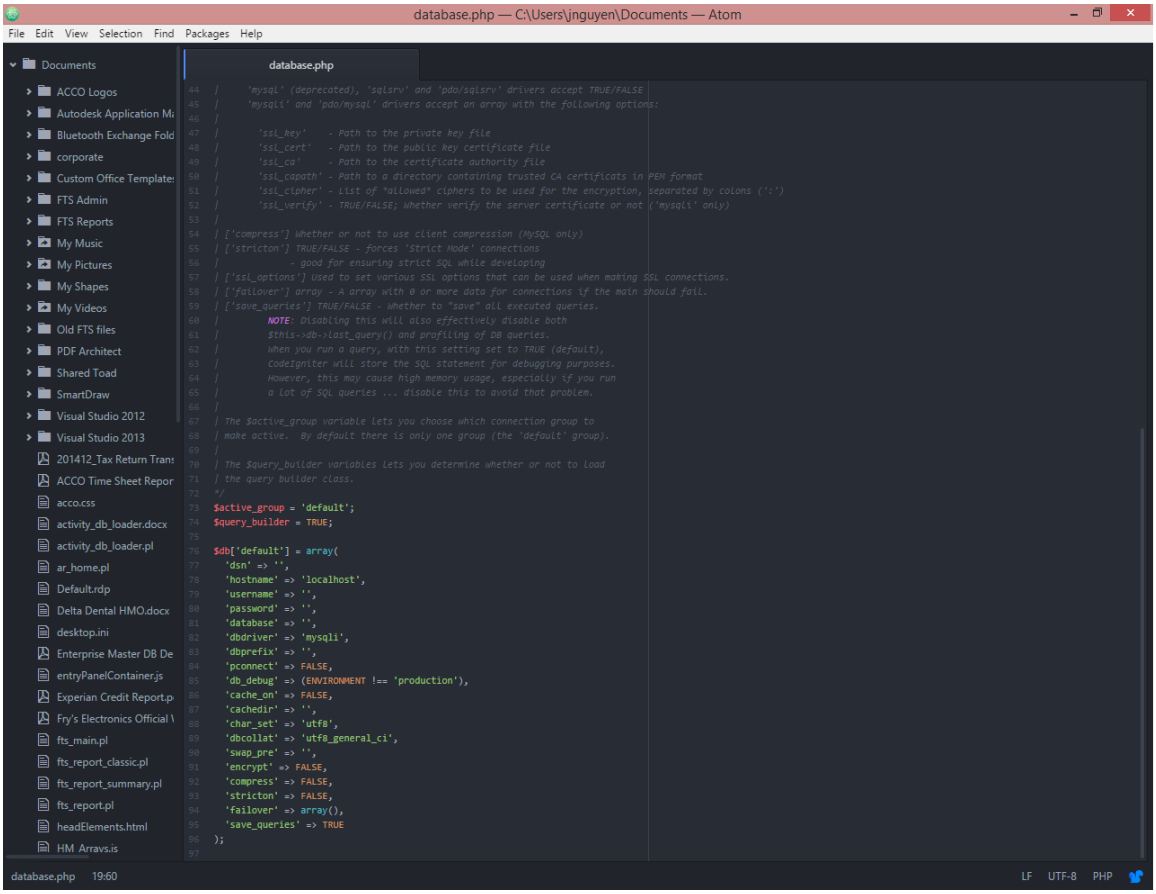

Figure 6.2 Atom.io UI and text example

6.3 Dreamweaver CS 5

Our user acceptance testing process involved developing a simple CMS website, so we used Dreamweaver CS 5 for the development. Dreamweaver has its own FTP engine so synchronizing to the web server can be done with a few mouse clicks. Dreamweaver also provides a dual view mode, where the page can be viewed in code mode and design mode simultaneously, as well as separately. Seen in Figure 6.3 below, the design mode only worked well for html pages linked to CSS pages, and not PHP pages.

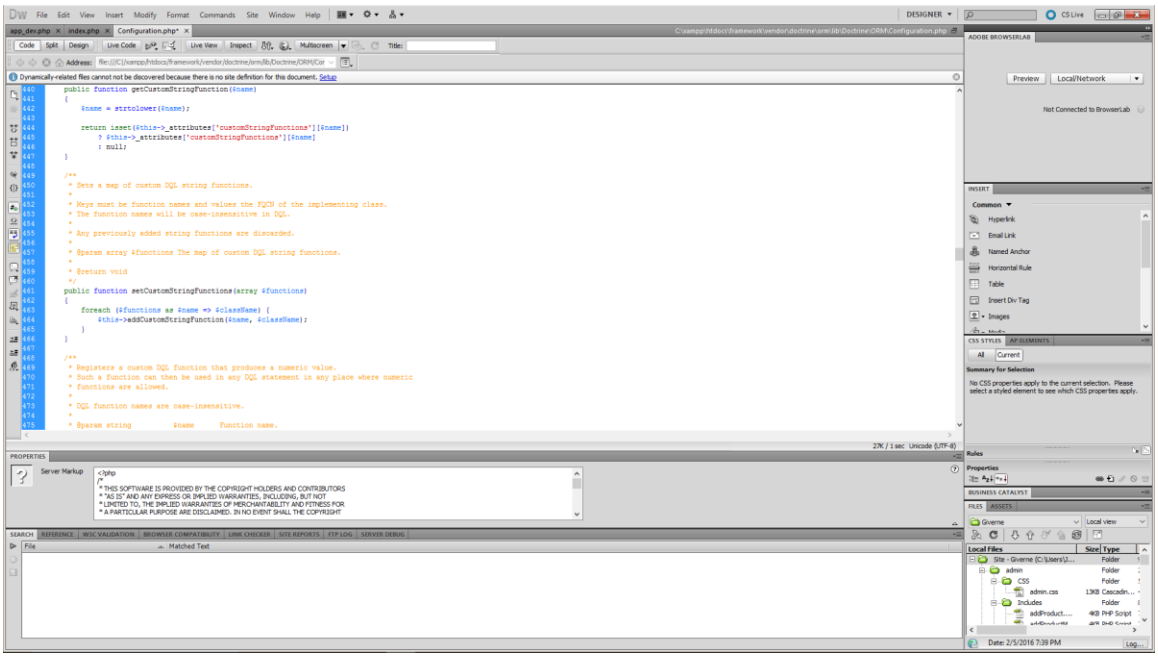

Figure 6.3 Dreamweaver CS 5

6.4 Google Chrome for Debugging

Google Chrome is a very commonly utilized web browser – it has the highest browser market share currently, at 54.41% of the total market share [Netmarketshare]. Taking this large market share into consideration, we decided to focus on confirming that our project works on Google Chrome. We used the native Developer Tools for tracking the data that was being sent to and from the server; we were able to ascertain the correct data and signals were being sent to our framework. Figure 6.4 below shows an example of the Developer Tools interface as well as the functionalities it can track.

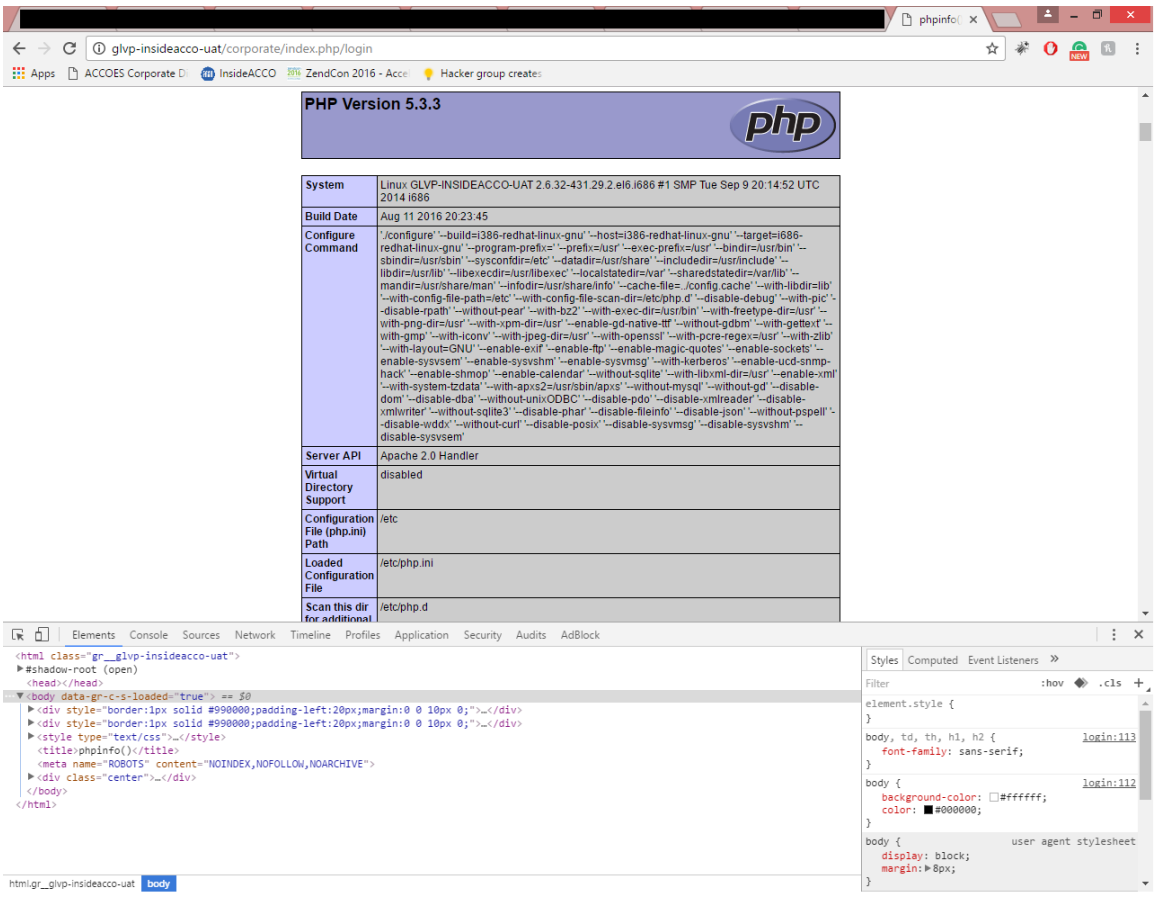

Figure 6.4 Google Chrome Developer Tools for Debugging

6.5 Toad for MySQL

Database management was necessary to test our framework, since a CMS solution needs to manage content, which is almost always stored in a database. While we could use a CLI tool for this, we decided to keep the project management easy by using a GUI based database tool. There are many GUI based database tools out there, both open-source and commercial. We chose Toad for MySQL for multiple reasons, one of which is the open-source nature of the tool. Figure 6.5 below depicts the GUI of Toad for MySQL, which shows the schema for a table as well as an explorer to easily change between database schemas to which the current logged in user account has access. [ToadWorld]

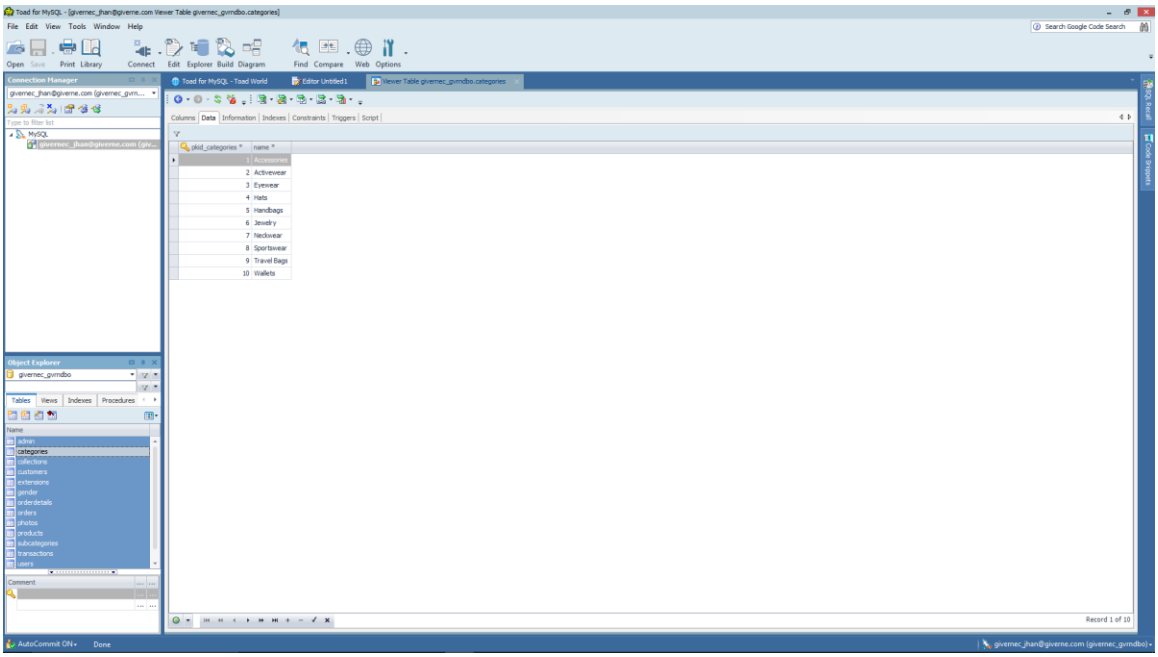

Figure 6.5 Toad for MySQL UI

Toad for MySQL is a freeware tool distributed by Dell and is constantly updated by a team of developers [ToadWorld]. Since MySQL is always being improved and updated by Oracle [MySQL], having a constantly updated freeware tool for managing the databases is a necessity for open-source developers. This is why we chose to use Toad for MySQL for our project.

6.6 Symfony PHP Reference Documentation

Lastly, we used the tutorial and documentation of Symfony PHP to develop this project. The documentation, including the tutorial, can be found on the Symfony website at http://symfony.com/doc/current/create_framework/introduction.html#why-would-you-like-to-create-your-own-framework, where the steps are outlined in detail. Support can also be found on the website, within their community forums.

CHAPTER 7

IMPACT ON END USER

The purpose of the project was to create a framework on top of the Symfony PHP framework. This new framework will decrease the amount of time it takes web developers to start the development of a CMS solution. An important part of a CMS solution is the database. Thus, well-developed and well-maintained CMS solution has a database that is normalized, easy to maintain, and designed for growth. This framework aimed to help the developer with the creation and maintenance of the database.

7.1 Level of Experience Required

Although this framework was designed with the CMS developer in mind, there is a requirement for the developer to understand: (a) how MySQL (or relational) databases are designed, and (b) their functionality. Not all web developers are familiar with the concept of normalized relational databases – usually only DBAs, database administrators, are knowledgeable about database design and the implications of performance and maintainability that the design holds. However, lately, a larger amount of web developers, typically full stack developers, are becoming experienced in database design.

Knowledge of a database's design should be considered necessary for a CMS solution developer. We believe the developer can only develop a CMS well if they have a firm understanding of the underlying database's topology and design. Without this understanding, the application that is developed cannot fully utilize the database in an efficient manner. However, many application developers do not have any database design

experience, or knowledge, and this could lead to databases that are unable to grow according to the changing scope and functionality of the application. If the database cannot adapt to the changes in the application's scope and functionality, then the data model will become a major obstacle in the future.

Though it is designed to help web developers with faster development of CMS solutions, the first build of the framework will only be of use to developers with experience in database design and management. This was a design decision made to have a focus on facilitating the LAMP stack developers' work, allowing them to quickly begin developing the application.

7.2 Installation Method

Setting up Symfony on any development machine required using either the Symfony installer or Composer, the dependency manager. Although these two methods are not difficult to use, it still requires more time to set up than unloading a zip file or running an executable file. For example, CodeIgniter is fast to download and fast to set up. It is a 2 MB zip file, downloaded from the official CodeIgniter home page, which you unarchive using an unzipping tool, such as WinZip, 7z, or the tool built into recent versions of Windows.

The framework will be released in an archived, prepackaged version of Symfony, similar to CodeIgniter. The end user would only be required to download the zip file from a source and then unarchive the entire folder structure inside to their server, usually in the directory "/var/www/html" for Linux servers. After the framework has been transferred to the server, the developer can immediately begin developing on the framework.

7.3 Value to the End User

Like other applications and tools, there must be value to the end user. If there is no value provided by an application, then it will have been a pointless venture, making the entire project a failure. In this section, we will discuss the value this framework adds and why it will be a great tool for more experienced CMS developers.

One feature of this framework is the ability to create a table in the database by filling out a form according to the documentation. The form is shown in Figure 7.1 below. The interface starts the user with only field, but allows the user to add more fields dynamically. The PHP script within the controller will verify each input field to make sure there are no empty fields and then it will run the script, making an API call to the function createTable($table, $fields) passing the parameters generated from the input fields on the form. The $table variable must be a string and the $fields variable must be an array of arrays where each element contains the name, attributes and type. The script uses prepared statements to limit the risk of SQL injections.

This feature allows the developer to quickly create tables if the database has been designed and the tables and their fields have been detailed, decreasing the amount of time spent on database creation and allows the developer to reallocate that saved time to another task. The developer can also alter and drop the table using the API call depicted by the example code in Figure 7.2.

Similarly, procedures and views can be created using the corresponding API calls depicted in Figure 7.2. All of the depicted API calls function similarly to one another. Although the API calls for views and procedures can only handle simple views and

procedures, it is useful for the initial set up of the database since more complex views and procedures can always be created or updated at a future time.

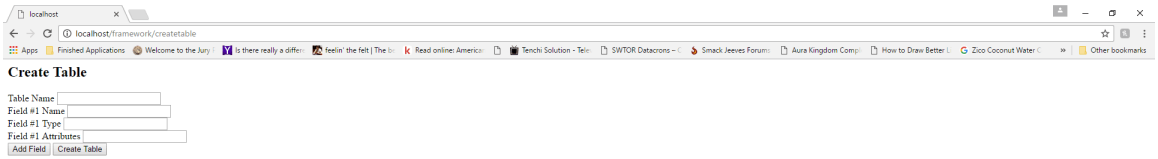

Figure 7.1 Using createtable controller to create a new table

```
$framework = new CMS\Framework($urlMatch, $contRes, $argRes);


$view = "viewA";
$query = "SELECT * FROM tableA LEFT JOIN tableB ON (tableA.fieldA = tableB.fieldC)";
$framework->createView($view, $query);
$framework->alterView($view, $query);
$framework->dropView($view);


$procedure = "procA";
$query = "DECLARE @name string;
      SELECT @name = t.name FROM tableA t WHERE id='1';
      UPDATE tableA t SET t.name = @name WHERE id='10'";
$framework->createProcedure($procedure, $query);
$framework->alterProcedure($procedure, $query);
$framework->dropProcedure($procedure);


$fields = array();
$fields[] = {"name" => "fieldA", "type" => "INT(11)", "attributes" => "NOT NULL AUTO_INCREMENT,
PRIMARY KEY (fieldA)"};
$fields[] = {"name" => "fieldB", "type" => "varchar(60)", "attributes" => "NOT NULL"};
$fields[] = {"name" => "fieldC", "type" => "DATETIME", "attributes" => ""};
$table = "tableA";
$framework->createTable($table, $fields);
$framework->alterTable($table, $fields);
$framework->dropTable($table);
```

Figure 7.2 Sample Code of API Calls

7.4 Value to the Project Stakeholders

The framework provides value to not only the end users, but it may provide value to other project stakeholders, such as the project manager, the lead developer, and the development team manager. These stakeholders are responsible for managing the time spent and quality of the product. This framework would not just benefit stakeholders who are interested in time management as this framework can benefit stakeholders who are interested in quality management.

Since the time spent on the initial database setup is reduced, the extra time can be allotted to development or automated test script development which could result in the early completion of the project. The time would be valuable to the project manager and maybe the development team manager. If the amount of time saved was recorded for each project, the managers could see a trend in the time saved, allowing them to use this information in their estimations for project time of completion. A consistent trend of time saved would increase the accuracy of their estimated time of completion projections based on the level of effort required for that project.

Symfony has coding standards and conventions that are recommended and encouraged but not forced. If all the developers on a development team were using the same coding standards and conventions, then code reviews would be completed more efficiently and the quality of the code would be ensured provided that every developer is strict in sticking to the standards. Debugging other developers' code would also be improved because the code will all be conforming to a single "format." Another problem with code reviews and debugging other developers' code is the difference in the logic used

by different people. Strict coding standards may help to alleviate this issue, which would also increase the efficiency of code reviews and debugging, thus increasing the quality of the source code.

7.5 Possible Value

Cost is a factor in the decision of which tool or product to use (i.e. is it expensive, cheap, or free?) so this framework may have value to the financial/business stakeholders of the project. Since this framework was built on an open-source package, we will be releasing this as an open-source package as well. There will be zero cost to use our framework, which may make it attractive to small businesses or individual developers.

CHAPTER 8

THE FINALE

We have arrived at the end of this creative undertaking, and now we have a finished product that we can distribute and use in the future. Although we finished the project, we met with difficulties along the way. In this final chapter, we will discuss the problems that arose, our thoughts on the entire project, and the future work that we could integrate.

8.1 The Problems

During the project, we ran into a few problems that prevented us from continuing with development. The first problem we encountered was an unexpected one: when we were roughly three-quarters complete, the hardware used for development suffered a crash, causing us to lose the work. We had multiple backups at different levels of completed work, but we believed all of our backups became corrupted and we would have to start from the beginning. It turned out that only some of the more recent backups became corrupted. We were forced to use an earlier backup. While it did not contain our entire work, it had most of our initial configuration work saved, so we only had to re-implement our designs.

The second problem we had was gathering requirements for this project from multiple perspectives. Contacting developers in the community to get their ideas was harder than initially thought. We also tried contacting developers in our own company, as well as project managers. Unfortunately, not many people were well versed in frameworks to know what they wanted in a CMS framework.

8.2 Thoughts on the Project

The project was, to our surprise, required more work than we had initially thought it would. We had initially estimated the time it would take to gather the requirements at around two to three days, but it actually took almost a week due to the busy schedules of those involved. Keeping in contact with everyone required us to write emails daily, asking for possible useful features that this CMS framework would need to make projects easier and faster to build.

In hindsight, using a framework we had never used previously proved to add some difficulty to this project because we were forced to learn the framework as we implemented our design. The existence of the tutorial helped with the learning process but there were more topics that were not covered in the tutorial. We had to learn about those topics to make sure what we were doing would not affect the functionality and stability of the framework as a whole. It may have been a better idea to use a framework we were familiar with for this project.

After development completed, we reviewed PHPUnit again, this time under greater scrutiny than we had done previously. As a result, we discovered that PHPUnit's database testing API would have served us better as a testing suite than the testing method we had decided to use. Initially, we thought that we needed to have static table and field names and that we could not use dynamically created tables and fields in the testing API calls. However, that was completely opposite to what PHPUnit was designed for because the user passes the database information, including table and field names, as parameters to the database testing API.

This would have significantly reduced the amount of time we needed to spend on testing the functionality. We still needed a page to make the calls to our framework so the overhead for the creation of the page remains the same, but the time spent on double checking the database for verification that the procedure ran correctly and there were no bugs or errors would have greatly been reduced. The automated testing would just require an API call to PHPUnit and we would have verified whether or not the database was changed according to the functionality.

Overall, we thought this project was a success because we managed to create a product that met most of our initial functional requirements. We will definitely be using our product in our future CMS developments for clients. However, this project is not at its end because there are more features we can improve upon, thereby adding to our future workload for new builds and releases.

8.3 Future Work

This is just the initial build that we have completed. There are more features that we would like to add to this project. In the future, we would like to improve this framework by overhauling the process while keeping the main API calls the same. In this future build, we want to make it even easier for the end users to create the database for their CMS solution. The framework will have an additional controller and corresponding view, which will house the user interface. Users will then be able to define the data structures they wish to capture in the database and the framework will employ the best practices of database design and create a normalized database for the end user. This functionality would most likely only be able to support the MySQL database engine, unless we get developers

proficient with other database engines to join the project. With this value added, we believe that it would be a great framework for beginner developers to use, requiring minimal knowledge of databases.

Another impactful change we would like to add to the framework is the support of other database engines, such as NoSQL, DynamoDB, and MariaDB. This will be a necessary addition because database engines are consistently being improved and new engines are being introduced. Currently, we do not have the knowledge of these database engines necessary to add this support. We hope to garner the attention of developers knowledgeable in various database engines in order to add this change to the framework.

There are more features that could make this framework more robust. To discover what other features should be added, we need to distribute this current build to other developers and have them use the build to develop a CMS solution. We would like them to write down their opinions and thoughts during the development process – as they experience it, they may come up with new ideas, making it the best time to gather their feedback. Afterwards, we would use their feedback and create a list of possible features we could add and make decisions on which, if any, features would be implemented in the next build.

Performance is always an important aspect that needs to be evaluated when making a decision on what tool, product, vendor, etc. to use for any specific project. The importance is even more prevalent in the field of software development because society is expecting technology to improve its performance regularly. As such, we must evaluate this framework's performance and improve it regularly because the end user will only find value in a product that is faster, more reliable, and easier to use than its competition.

Unfortunately, we did not get a chance to evaluate any of the performance aspects of this framework; therefore, prior to release, we would like to evaluate the performance of all of the API calls and compare them to the performance of another database tool, such as Toad for MySQL or MySQL Workbench, running the same functionality.

The last bit of work we would like to complete prior to our first release would be the refactoring of the entire source code we developed. In doing so, we hope to increase the performance, as refactoring can affect the performance of a system. [Fowler] Fowler has shown that refactoring could potentially improve the performance if refactoring was done correctly. Fowler used an example where splitting one loop into two loops improved its performance. [Fowler]

8.4 Conclusion

At the end of our journey, we have come to realize some things about our project and the subsequent framework resulting from our development work. First, automated testing is always better than manual testing if there is a possibility to do so. Second, when working with a diverse group of users, especially those who cannot be contacted at the same time, extra time must be allotted in the event that busy schedules prevents requirements engineering from occurring according to plan. Third, it is important to acknowledge that there are many technologies.

As discussed earlier, automated testing would have reduced the amount of time we spent on verification of the functionality had we understood the full capabilities of PHPUnit and its database testing API. This was a needed lesson we had to learn. Another lesson we learned was requirements gathering could become difficult to perform when all the users

cannot be contacted in a timely manner. This had thrown our development off schedule, leading us to believe that our group of users was too spread out. Lastly, new technologies are being developed every day; current technologies are being updated every day. This must be taken into account when decisions regarding which technologies to support.

Overall, we thought the project was a success. We accomplished what we had set out to do, thanks the detailed tutorials and documentation of Symfony. Although we met some difficulties along the way, we did not let it get our morale down and stop the project. If we had stopped, we would not have created this product that we can now release once we fully license the work. We hope future PHP developers can develop and build upon what we have created, making it a larger open-source project.

Works Cited

1. "The Agile Cultural Shift: Why Agile Isn't Always Agile." *The Agile Cultural Shift: Why Agile Isn't Always Agile* (2016): n. pag. *CGI IT Services*. CGI, 2016. Web. 3 Nov. 2016. <https://www.cgi.com/sites/default/files/white-papers/agile-culture-white-paper.pdf>.
2. "Apache Friends." *Apache Friends RSS*. N.p., n.d. Web. 04 Dec. 2016. <https://www.apachefriends.org/index.html>.
3. Bergmann, Sebastian. *PHPUnit Manual*. N.p.: PHPUnit, 30 Oct. 2016. PDF.
4. "CakePHP - Build Fast, Grow Solid | PHP Framework | Home." *CakePHP*. CakePHP, n.d. Web. 04 Nov. 2016. <https://cakephp.org/>.
5. "Codeception." *Codeception*. N.p., n.d. Web. 02 Mar. 2016. <http://codeception.com/>.
6. "CodeIgniter Rocks." *CodeIgniter Web Framework*. CodeIgniter, n.d. Web. 05 Nov. 2016. <http://www.codeigniter.com/>.
7. "Create Your Own PHP Framework (current)." *Create Your Own PHP Framework (current)*. Symfony, n.d. Web. 02 Nov. 2016. <http://symfony.com/doc/current/create_framework/index.html>.
8. "DB-Engines Ranking." *DB-Engines Ranking - Popularity Ranking of Database Management Systems*. N.p., n.d. Web. 10 Nov. 2016. <http://db-engines.com/en/ranking>.
9. Fowler, Martin, and Kent Beck. *Refactoring: Improving the Design of Existing Code*. Boston: Addison-Wesley, 2000.

10. Hitesh, Ramoliya. "Why Laravel Is Best Php Framework In 2016?" *Why Laravel Is Best Php Framework In 2016?* LinkedIn, 27 Jan. 2016. Web. 1 Nov. 2016. <https://www.linkedin.com/pulse/why-laravel-best-php-framework-2016-ramoliya-hitesh>.
11. "Market Share Reports (Source of Analytics Data)." *Market Share for Mobile, Browsers, Operating Systems and Search Engines*. N.p., n.d. Web. 10 Nov. 2016. <http://netmarketshare.com/>.
12. "Oracle MySQL." *MySQL | The Most Popular Open-Source Database | Oracle*. N.p., n.d. Web. 04 Dec. 2016. <https://www.oracle.com/mysql/index.html>.
13. Otwell, Taylor. "Love Beautiful Code? We Do Too." *Laravel*. Laravel, n.d. Web. 03 Nov. 2016. <https://laravel.com/>.
14. "SimpleTest - Unit Testing for PHP." *SimpleTest - Unit Testing for PHP*. N.p., n.d. Web. 02 Apr. 2016. <http://simpletest.org/>.
15. "Toad for MySQL - Toad World." *Toad for MySQL - Toad World*. N.p., n.d. Web. 04 Dec. 2016. <http://www.toadworld.com/products/toad-for-mysql>.
16. Winspire Web Solution. "7 Best PHP Frameworks for 2015." *7 Best PHP Frameworks for 2015*. LinkedIn, 3 June 2015. Web. 1 Nov. 2016. <https://www.linkedin.com/pulse/7-best-php-frameworks-2015-winspire-web-solution>.

APPENDIX A

GLOSSARY

API – *Application program interface*: set of definitions, protocols, and tools for connecting two pieces of software.

CLI – *Command-line interface*: a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

CMS – *Content management systems*: a computer application that supports the creation and modification of digital content using a simple interface to abstract away low-level details unless required, usually supporting multiple users working in a collaborative environment.

CSS – *Cascading style sheets*: a style sheet language used for describing the presentation of a document written in a markup language, such as HTML.

GUI – *Graphic user interface*: a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation.

HTML – *Hypertext markup language*: the standard markup language for creating web pages and web applications.

IDE – *Integrated development environment*: a software application that provides comprehensive facilities to computer programmers for software development, typically consists of a source code editor, build automation tools and a debugger.

SQL – *Structured query language*: a special-purpose programming language designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

UML – *Unified modeling language*: general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

APPENDIX B

CODE EXCERPT OF SELECT API CALL

```
$tnames = array();
$talias = array();
$tcount = 0;
foreach($tables as $t){
    $tnames[$tcount] = $t["name"];
    $talias[$tcount] = (!empty($t["alias"])?$t["alias"]:"");
    $tcount++;
}

if($fnames != "*"){
    $fnames = array();
    $fcount = 0;
    foreach($fields as $f){
        $fnames[$fcount] = $f;
        $fcount++;
    }
}

$hasParams = false;
if($params != null || !empty($params)){
    $hasParams = true;
    $pfield = array();
    $pvalue = array();
    $pcount = 0;
    foreach($params as $p){
        $pfield[$pcount] = $p["name"];
        $pvalue[$pcount] = $p["value"];
        $pcount++;
    }
}

$hasJoins = false;
if($type != null || !empty($type)){
    $hasJoins = true;
    $tsize = sizeof($type);
}

$query = "SELECT ";
if($fnames != "*"){
    for($i = 0; $i < $fcount; $i++){
        if($i != $fcount - 1 && $i != 0){
            $query .= ", ";
        }
        $query .= $fnames[$i]." ";
    }
}
else{
    $query .= "* ";
}

$query .= "FROM ".$tnames[0]." ".($talias[0] != ""?$talias[0]." ":"");
if($hasJoins){
    $index = 1;
    foreach($type as $join){
        $query .= $join["type"]." ".$tnames[$index].($talias[$index] !=
""?$talias[$index]." ":"")."ON (".$join["match"].") ";
        $index++;
```

```
    }
}

if($hasParams){
    $pstring = "";
    $query .= "WHERE ";
    for($i = 0; $i < $pcount; $i++){
        if($i != $pcount - 1 && $i != 0){
            $query .= "AND ";
        }
        $query .= $pfield[$i]." = ? ";
        $pstring .= "s";
    }
}

$stmt = $this->db->prepare($query);
if($hasParams){
    $a_param = array();
    $a_param[] .= & $pstring;
    for($i < 0; $i < $pcount; $i++){
        $a_param[] = & $pvalue[$i];
    }
    call_user_func_array(array($stmt, 'bind_param'), $a_param);
}

if($stmt->execute()){
    $res = $stmt->get_result();
    $data = array();
    while($row = $res->fetch_assoc()){
        array_push($data, $row);
    }
    $stmt->close();
    return $data;
}
else{
    return "Error: SELECT call encountered an error with the execution.";
}
```