ASHESI UNIVERSITY COLLEGE

Using Model-View-Controller Architectural Pattern in Building a new PHP Micro-Framework

APPLIED PROJECT

B.Sc. Computer Science

Oliver Mensah

2018

ASHESI UNIVERSITY COLLEGE

Using Model-View-Controller Architectural Pattern in Building a new PHP Micro-Framework

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfillment of the requirements for the award of Bachelor of Science degree in Computer Science

Oliver Mensah

2018

DECLARATION

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date: ……

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied project laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date: ……

ACKNOWLEDGEMENT

I thank God Almighty for the breath of life to see this project through. I would also like to extend my gratitude to Mr. David Sampah for his guidance, support and helpful supervision during the process of this project.

Not forgetting my mentor, Kent C. Dodds, for his encouragement, suggestions, support and advice to choose this kind of project.

Other thanks go to staff and student of various departments in Ashesi University College for their immense contribution and response within a limited time frame.

To all friends, family and colleagues who supported me throughout the period of my study, I say thank you.

Finally, I would like to thank MasterCard Foundation for their sponsorship in pursuing my undergraduate's degree. Without all of you, I would not be at this stage of my life. I say a big THANK YOU to you all. GOD BLESS YOU.

Contents

TABLE OF FIGURES

ABSTRACT:

This project is about building a minimal web framework in PHP for developers who use framework for building complex application. I had the opportunity to work with PHP and some of its framework in production level. Some of these frameworks are awesome to work with but depending on the kind of project a developer will be working on, you end up tweaking the internals of the framework which a developer with less knowledge of the framework will find it very difficult to do. There is also much learning curve in using this these frameworks

This project seeks about building a minimal web framework in PHP where developers start from small and many functionalities are needed, developers can make use existing libraries on top of the framework to achieve those tasks. In this way, libraries are added only when they are demanded by the developers rather than coming with the frameworks in a bundled form.

A feasibility study would be carried out to find if this framework would be feasible. Stakeholders would also be made to confirm functionalities and suggest new functions as well.

Upon completion of this project, it's the aim of the developer that the framework would help other developers to build applications with it.

Chapter 1: Introduction:

1.1 Introduction:

Web applications development started with composing simple and static Hypertext Markup Language (HTML) pages mainly to share the information over the internet (Ping, Kontogiannis, &Lau, 2003). In recent times, many organizations and change in business needs have led to massive changes to how web applications are built. Henceforth, Modern Web systems are now designed by multi-tier architectures and supported by open standards, new technologies and design patterns (Ping, Kontogiannis, &Lau, 2003). This paper presents Model-View-Controller(MVC), a software design and architectural patterns applied in modern software development, which is analyzed and used to build a new simple and efficient PHP: Hypertext Preprocessor (PHP) web framework. By using this framework, an online internship placement platform is created. This application will help students to get internship offers created by companies.

1.2 Background

With the rapid development and penetration of the Internet, developing applications for the web platform has been more and more popular. Web applications were built statically with just HTML pages. It got harder for such applications to be maintained with content changing dynamically (Zhang et al., 2013). This problem has brought forward a higher demand of efficiency, reliability, maintainability and scalability. PHP is one of such languages used today in building applications with dynamic content, it is easier to maintain and scale. However, writing pure PHP code without following any architectural design pattern makes it easier to write spaghetti code (Tommi & Antero, 2008); meaning the code for data access, business logic processing and presentation of view layer are mixed together. It was an actual problem in when working on web applications (Zhang et al., 2013).

Over time, practitioners in the software industry have developed patterns that helps in building application to ensure clean separation of database, business logic and interface component. MVC architectural design pattern has proven to be useful in dealing with separation of concerns (Wang, 2011). MVC is built on the idea that the logic of an application should be separated from its presentation. It divides the application into three interconnected parts, namely, model, view and controller. This is done to separate the internal representations of information from the ways information is presented to and accepted from the user. It is becoming the definitive architecture of web application frameworks due to its stability and extensibility. Though these frameworks are awesome to work with, depending on the type of application to be built; using some of these frameworks seem bloated (Wang, 2011). Instead of having framework bundled with a lot of libraries that might not be needed on demand for some applications, this project seeks to employ the MVC pattern to provide a minimalistic framework called "barePHP". Due to it being minimal in nature, it does not offer a complete solution like what some other frameworks like Laravel do. Hence, it makes it easier for developers to use any third-party library of the choice to build fully-fledged web applications. (Wang, 2011)

1.3 Problem Description and Motivation

PHP is one of the most used web application development languages. According to W3Tech, PHP runs about 83.1% of all the websites with server-side programming language (W3Techs - World Wide Web Technology Surveys, 2017). However, it is easier to write PHP code with no separation of concerns thus, mixing the code for data access, the processing of business logic, and web presentation layer. This makes it difficult to have maintainable and scalable PHP based web application. With the evolution of architectural design patterns like the Model View Controller Pattern, it makes it much easier to separate internal representation of information from the way

information is presented to and from clients. This decouples components and allows efficient code reuse. MVC brought revolution in how most building software, tools for the web typically in building frameworks (RashidahF Olanrewaju, Thouhedul Islam and Nor'ashikin Bte.Ali, 2015). There are many MVC-based web frameworks implemented in different programming language platforms like PHP, JavaScript, etc. Each of these frameworks has its own pros and cons. Generally, some of the advantages of using these frameworks are; efficiency, security, integration and among others. However, there are issues of steep learning curve, cost of development, learning bias and performance when using some of these framework (Wang, 2011)s. This framework aims to make it easy for developers at any level; beginners, intermediate or experienced to build any PHP based web applications. With composer, packages repository for PHP libraries, developers can make use of any library with barePHP framework to build any application of their choice.

1.4 Aims and Objectives

Using complete frameworks to build applications requires much investment of time to learn the framework before developing the application. Also, these frameworks do not allow you to have full control over your implementations. This is where the barePHP framework comes in. This framework is for PHP developers to build web application with full control of their solutions. The barePHP separates the data, view and control of web application to prevent a of these components, thereby helping to achieve loosely coupled application that are highly reliable, maintainable and scalable. The framework reveals simple features for developers to interact with database, rich Uniform Resource Locator (URL) routing, rendering views. With these, any software developer with PHP knowledge who understands the concept of Object Oriented Programming(OOP) can be able to write a maintainable and scalable application without necessarily investing much time in learning about the ecosystem of the framework before building the application.

Chapter 2: Related Works

2.1 Related Works

There are a lot of PHP Web Application Frameworks that use the Model View Controller Architecture (RashidahF Olanrewaju, Thouhedul Islam and Nor'ashikin Bte.Ali, 2015). For example, CodeIgniter, CakePHP, Laravel, Symfony, and others. In 2015, SitePoint, one of the popular website that provides only tutorials for developers around the web as well as forum to discuss software related issues, made a global survey to find the valuable insights about various PHP frameworks. One of the key questions include why developers favor one framework over the other (Skvorc, 2015). Figure 1 shows the graph of developers using various frameworks in production.

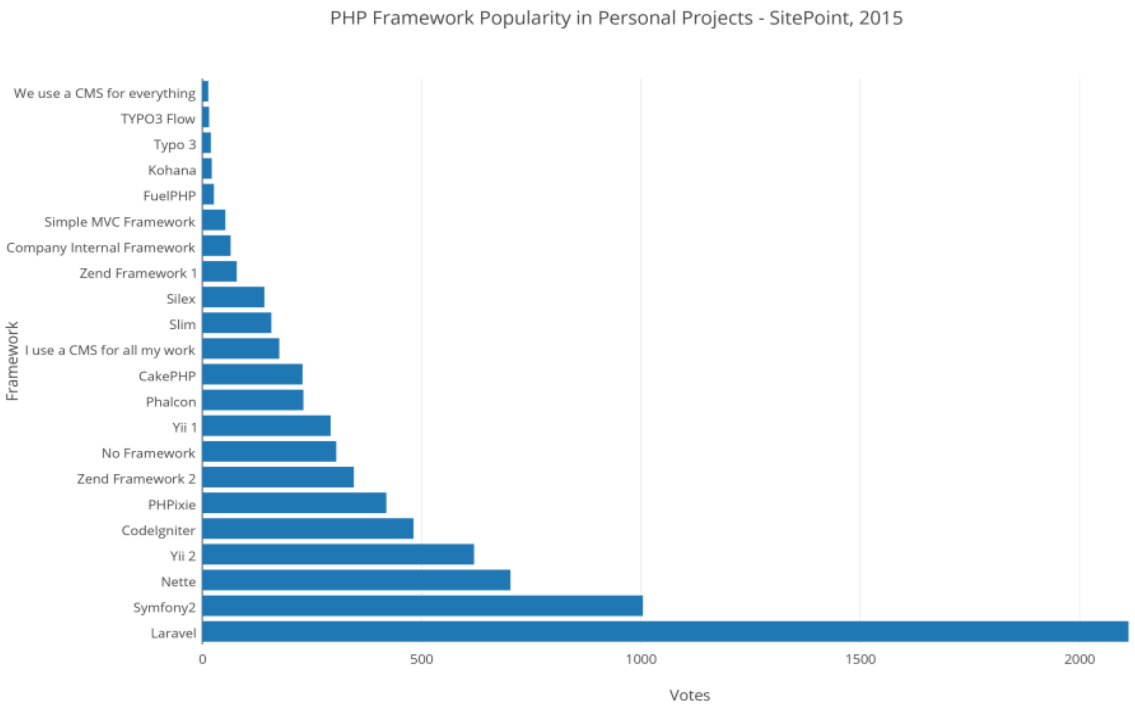


Figure 1 SitePoint PHP Framework 2015 Survey

Obviously, Laravel is the most used Framework among the all the various frameworks, then follow by the Symfony. In 2017, the same platform, worked on the state of PHP MVC Frameworks and following their results, the had two categories of PHP MVC frameworks; namely;

1. Symfony or Laravel, depending on your needs
2. The rest

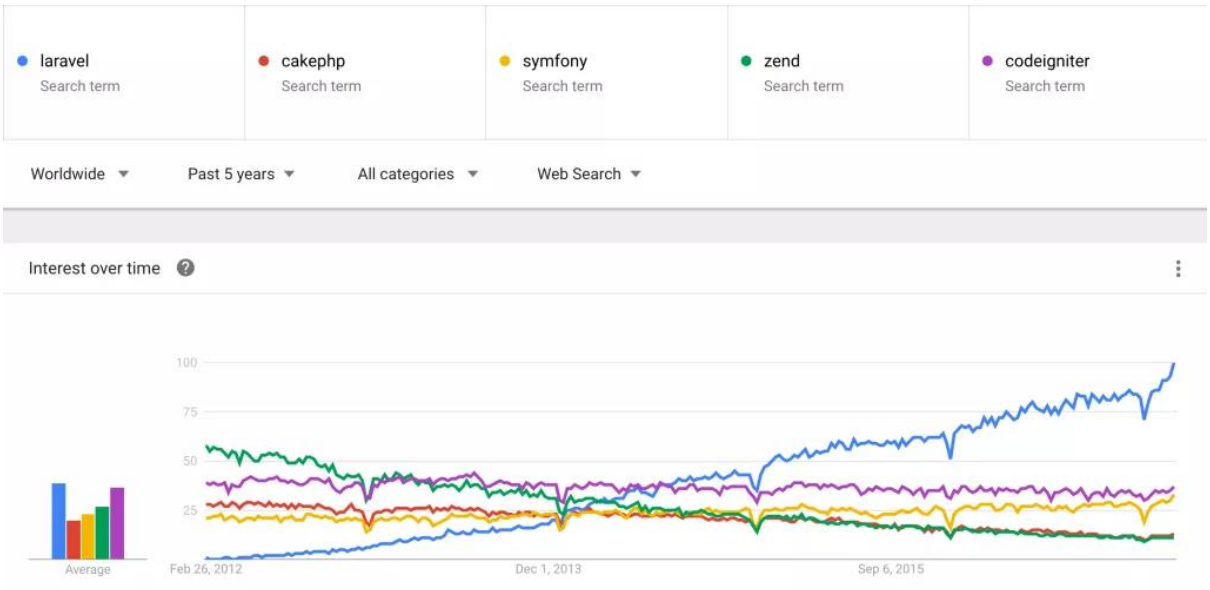


Figure 2 SitePoint PHP Framework 2017 Survey

From Figure 2 and Figure 8, the Laravel and Symfony continue to reign complete PHP frameworks for building web applications and Application Programming Interfaces(API's). Henceforth, this chapter will focus on these two frameworks, the ecosystem, why developers love to use them, the problems that developers face in using these frameworks and the reason behind the creation of the barePHP microframework.

2.1.1 Laravel Framework

It is an open source PHP web framework intended for the development of web applications following the MVC architectural pattern. Though Laravel is a complete solution but it leaves the architecture open for third party plugins. In this case, if a developer adds more third-party libraries to the already existing structure, it makes the bloated. Laravel provides convenient way to develop

Application Programming Interfaces (API's) and build complex backend system. Below gives the various pros and cons of Laravel over Symfony:

2.1.1.a Pros of Laravel

- Laravel makes it easy to work the management of database by providing easy means of migrating database systems as well as providing multiple database objects.
- It makes it easy to autoload files using PHP built-in autoloading facility
- It has easy to use pre-built authentication system. Developers are just allowed to configure models, database migrations

2.1.1.b Cons of Laravel

Though Laravel is awesome solution for building API's and complex backend system however, there is so much to do when it comes to using the framework. As a developer, you need to know about Blade templating, Gulp, Node Package Manager (NPM), Bower, Composer, Eloquent and Laravel architecture itself. Hence it might be steep learning curve(Kobilansky, 2017).

2.1.2 Symfony Framework

With Symfony, it is not just a framework but also has numerous modular components that can be separately be used and works magically with any framework that it bounds together with.

2.1.2.a Pros of Symfony

- Developers can make use of many easy scalability options that Symfony offers. Since it has several individual components, it makes it easy for developers to incorporate into their projects
- Flexibility in setting up projects.

2.1.2.b Cons of Symfony

- While Symfony is a fantastic framework, beginners find it complicated. Because of its complex structure, it seems to be "bulky" and hence needs to be configured. This can be challenging to understand and learn at first (Lukač, 2015).

2.1.3 The Core of both Laravel and Symfony

Laravel and Symfony, the leading two frameworks (Skvorc, 2015) of PHP have common core. Both frameworks are built on top of libraries bundled together. These set of libraries are called "The Symfony Components" (Narożny, 2017). These tools are excellent tools, but they have large learning curve and are "large, insular, kitchen-sink solutions" (Nawaz, 2016) for some projects.

2.2 Why A new PHP framework - BarePHP

Both top tier frameworks were created by bundling of independent modular components as a complete solution (Narożny, 2017). This makes them suitable for complex projects and using them for simple applications come at a cost of are "large, insular, kitchen-sink solutions" (Nawaz, 2016). Hence, the barePHP micro framework is proposed to provide simple and rapid development of simple PHP web-based application. The barePHP seeks to leverage either these

independent modular components to create simple and large applications or writing pure vanilla PHP code.

Chapter 3: Requirement and System Architecture

3.1 Requirement Design Overview

When developing software, defining requirements before starting the development phase saves a lot of time and sometimes reduces financial costs (Tran, 1999). Gathering software requirements helps to clearly define everything that the software will accomplish and serves as the basis for designing and the development process. This chapter provides an overview of the design methods and architectural pattern used in coming out with this framework.

3.2 Introduction to MVC Architecture

MVC, was first described in 1979 by Trivet Reinking, while working on Smalltalk at Xerox PARC. The Model-View-Controller pattern is the most used pattern for today's world web applications because it has been proven as the effective way to develop applications. The MVC pattern separates an application into three separate layers: model, view and controller that work separately to produce the same result. The controller handles the model and view layers to enable them work together. The **controller** is responsible for receiving a request from the client, and invokes the **model**, responsible for interacting with a data source, to perform the requested operations and presents the data to the **view**. which is responsible for rendering web pages.

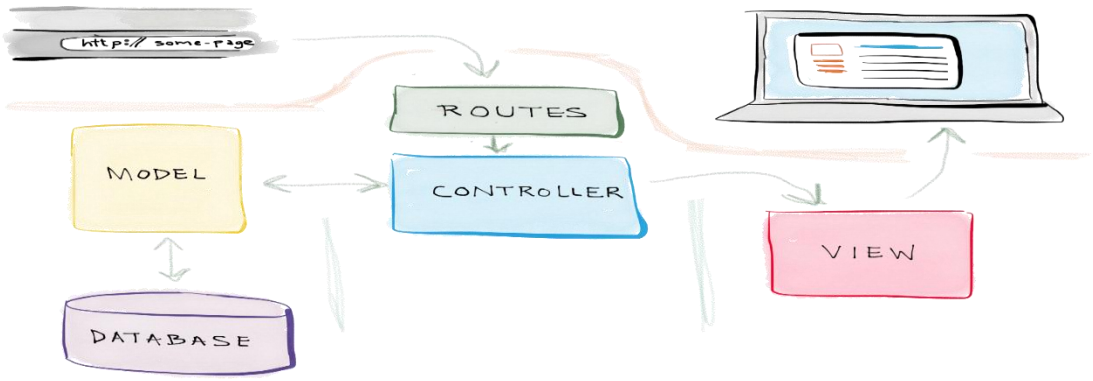


Figure 3 MVC Pattern Mechanism

Figure 3 depicts the MVC pattern through the interaction of an end user and the application with just a single flow of data. Here, the user makes a request to view a page through a client system using the browser by entering a URL. Behind the system, the application matches the URL to a predefined route. The controller action associated with the route is evoked. The controller action uses the models to retrieve all the necessary data from a data source, places the data in a data structure and loads a view, passing along the data structure. The view accesses the structure of data and uses it to render the requested page, which is then presented to the user in their browser (RashidahF Olanrewaju, Thouhedul Islam and Nor'ashikin Bte.Ali, 2015).

3.3 The core of barePHP Framework

This chapter describes the design and workflow of the framework, which is considered as a lightweight MVC framework. The barePHP framework is written in PHP. The project structure has a root file called .htaccess. This .htaccess file is a configuration file used on web servers running the Apache Web Server software, hence they are easily detected and executed by the software. This file contains a redirect functionality that redirects all the client requests to a public folder which has an index.php, to initialize core features of the framework. This index.php also requires a file that bootstraps the resources needed for the framework to function. Figure 4 provides a visual representation of how the framework works behind the scene with the help of the MVC architectural pattern.

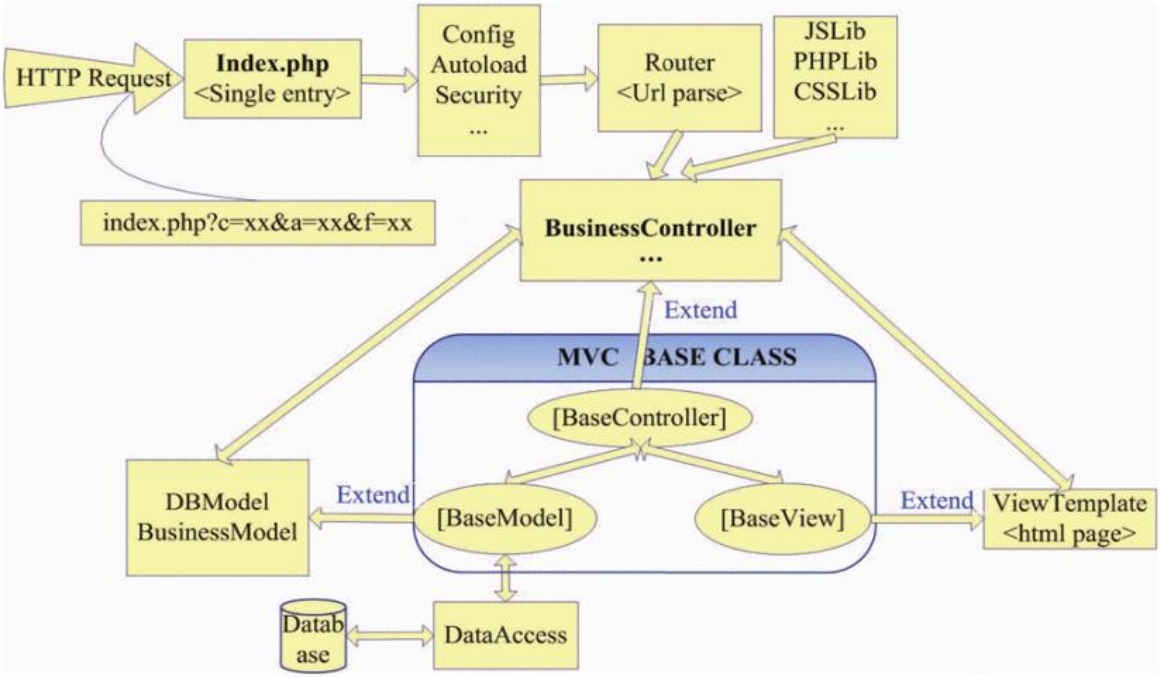


Figure 4 The workflow of the framework

3.4 System Description

The barePHP framework is a lightweight MVC micro framework for building web applications in PHP. Contrast to some other PHP framework which offer developers with complete solution, developers who need simpler solution oftentimes end up extracting a lot of functionality from those opinionated advanced frameworks. The barePHP framework takes the opposite approach.

This framework provides developers the foundational layer; the base functionality for MVC-like applications. This makes it easier to organize code, routes, business logic in a nicer and better manner. With MVC at its core, it makes it very easy to build a modularized application, which is easier to maintain. With barePHP framework, developers start small, minimal and then as the application progresses and advances, then they can add complex features either by using third party libraries that have been made available at the PHP package central repositories like

Packagist(Composer), Pear or by writing their own modules to handle those tasks they want to implement.

3.4.1 Features

BarePHP framework has brought a lot of common utilities used in development of systems. Some of these utilities employed in this framework include; MVC Pattern, class inheritance, convention over configuration and built-in validation

I. MVC Pattern

With MVC Pattern incorporation in barePHP, it helps ease the web development process. Here the pattern allows the developer to know the exact location of database operations like insertion, deletion, modification and selection, handling of business logic like processing user request as well as rendering user response through html webpages.

II. Class Inheritance

At the core of barePHP is extendibility. Since applications deals with CPU and IO bound operations, barePHP has two core modules that helps to create reusable code for performing IO operations such as data insertion, with the model component. The model component must be extended by any functinality that has to do with database activities. Also, the controller component can be extended by other controllers to handle CPU operations.

All Application specific controllers extend BaseController class with predefined methods to load a view and a model.

III. Convention over configuration

BarePHP employs zero configuration, hence developers can easily deploy applications with no complicated configuration files like XML, JSON, INI, YAML to help in application staging. The only thing to do while using barePHP is the database connection settings, setting URLROOT to replace any root-relative links "/" and APPROOT to replace any include/require links that uses the $SERVER["DOCUMENT_ROOT"] variable in PHP.

IV. Built-in Validation

The barePHP framework has utilities to validate incoming Hypertext Transfer Protocol(HTTP) request. By default, barePHP uses a Validation Class that provides variety of convenient methods to make available high validation rules for developers to handle HTTP requests.

3.5 Implementation Support(Tools)

The underlying tools used in creating this framework are the PHP programming language and apache ".htaccess" file. PHP is a server-side scripting language used to build web applications like any other Common Gateway Interface(CGI) program can do; such as collection of form data, dynamic page generation, or send and receive cookies. ".htaccess" is a configuration file for web servers, executed on Apache Web Server software. By default, ".htaccess" file in any directory is detected and executed by the Apache Web Server software. These ".htaccess" files can be used to alter the configuration of the Apache Web Server software to enable/disable additional functionality and features that the Apache Web Server software has to offer. In relation to this project, the ".htaccess" and the PHP programming language are used to build a fully functional web framework from scratch.

3.6 Operating Environment

To work with this framework, there should be a supporting environment for the framework to work. Any environment can be setup in any device provide the device has support to run a webserver that can interpret PHP code. Since the framework uses .htaccess for Mod_Rewrite, to setup custom and simplified URLs as needed for user friendly URLs in MVC, it will be important to use the framework to use an environment that runs an Apache Web Server software.

3.7 Functional Requirements

This provides an overview of how software developers can use the framework to build web applications. It presents the various requirement from setting up application environment configurations to finishing up with the entire application they intend to build with the framework. The functional requirements here describe the end users' workflow when working with the barePHP framework.

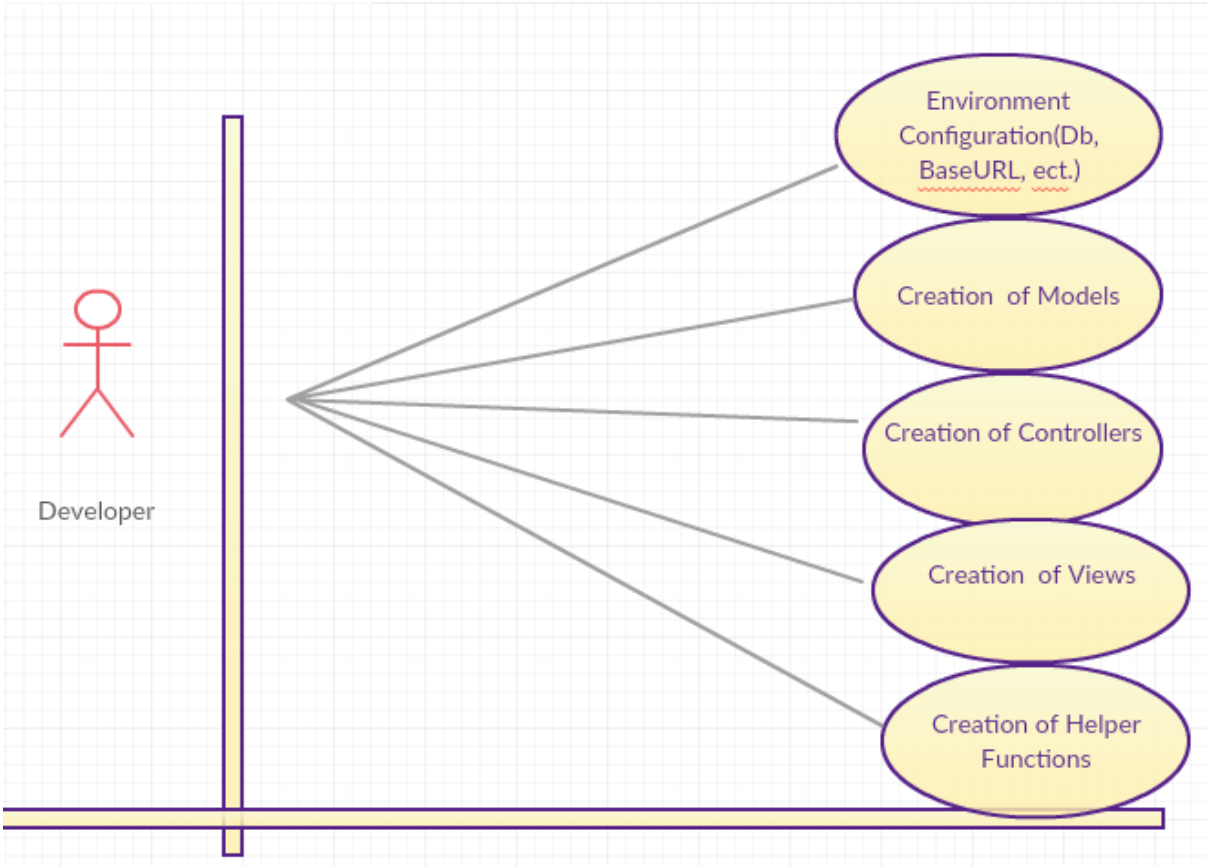


Figure 5 Principal requirements for the developer

Figure 5 shows the workflow of a developer using the framework in production. The following depicts the principal workflow of using this framework:

3.8 Non-Functional Requirements

3.8.1 Product Requirements

This section describes the product requirement of the framework. It helps to define the components of an operational product and the method in which these components must integrate to achieve

desired results. Here it consists of security, reliability and safety and performance of the framework.

3.8.1.1 Security

The framework makes it easy for the developer to have control on what to protect in terms of security threats. For instance, when a developer is making Hypertext Transfer Protocol (HTTP) Request with the post method, the developer has the capacity to handle it the way she/he wants.

3.8.1.2 Reliability

The framework should be able to display errors in the development in a readable and understandable format. In this case, it makes it easier to debug errors that occurred during development.

3.8.1.3 Safety and Performance

Determining the performance of this web framework can be inferred from the various web application that will be built on top of the framework. Below shows the various performance metrics that can be used to determine how the framework performs in relation to the applications that run on top of it;

I. Average Response Time: This determines the amount of time an application takes to return response to a user. It can be rendering of page or sending endpoint data in either JavaScript Object Notation (JSON) or Extensible Markup Language(XML) format especially when creating Application Programming Interface(API). This metric will help to find out how much time the framework takes to handle user request and return their responses.

II. Error Rates: Here, it is the percentage of problems(errors) that an application encounters. It can be a user trying to access a page which exists on the server but returns an error of file not found. This helps to determine the reliability of the application.

III. Application and Server Central Processing Unit (CPU) : As the application needs certain resources to execute, the relationship between the amount of server CPU needed to execute the application is Application and Server CPU metric. This metric will help determine whether the application built with this framework uses amount of CPU during its execution.

IV. Request Rate: It tells how many actions are sent to the target server by this application and how long it takes to process the request by the server. The metric here will help to determine how the framework handles request to web services.

3.8.2 Organizational Requirements

3.8.2.1 Operational Requirement

The framework makes greater use of .htaccess files to help rewrite URLs into MVC friendly URLs. This is only available on Apache webserver. This then restricts the framework as to what server is needed to execute the programs that are built with the framework.

Chapter 4: Implementation

This chapter provides clear implementation of the barePHP framework from provision of basic configurations to the implementation of the MVC architecture.

4.1 The Core Implementation of barePHP framework

The implementation of the barePHP framework can be group into the following categories;

I. Files and Folders Structure: This provides how files are organized in the framework;
II. MVC Architectural Pattern: This provides a detailed implementation of the pattern in the barePHP framework.
III. URL Manipulation: MVC friendly URL's is one of the core concepts that comes in handy when employing the MVC architectural pattern for crafting a software. URL manipulation provides how to transform all routes into MVC friendly URL's.
IV. The Registry: The framework comes with core functions, such as database access, URL routing, user authentication, etc. The registry pattern will help to keep a central repository of objects associated with the framework.

4.1.1 The Files and Folder Structure.

Any software project starts with file structuring by creating directories. The root directory of this framework contains application and public folder as well as .htaccess file as shown in Figure 6.

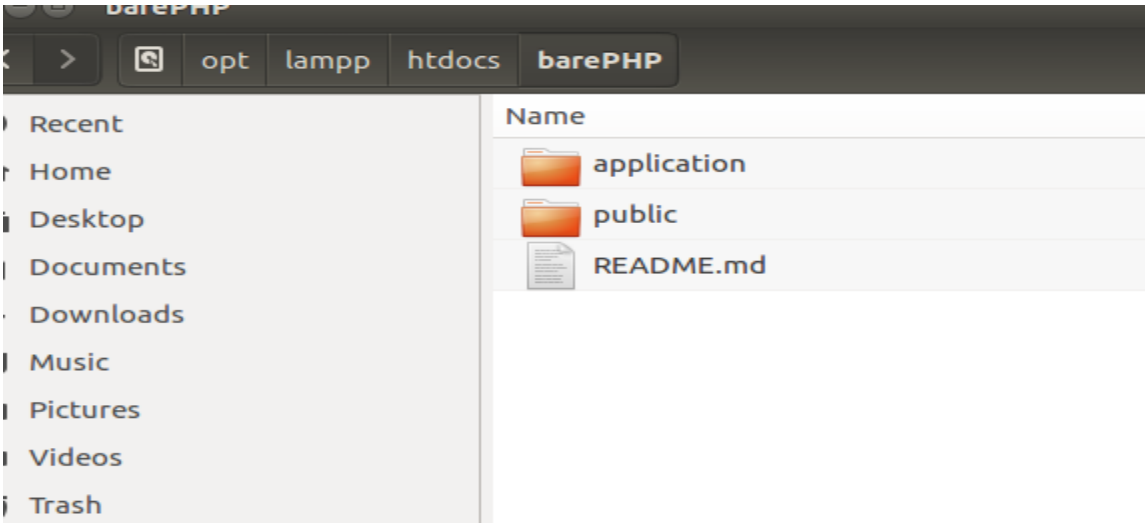


Figure 6 Framework Files and Folders Structure

4.1.1. 1 .htaccess File in the Root Directory

The .htacesss file reroutes the incoming request into the public folder as well as prevent access to those root files and folders.

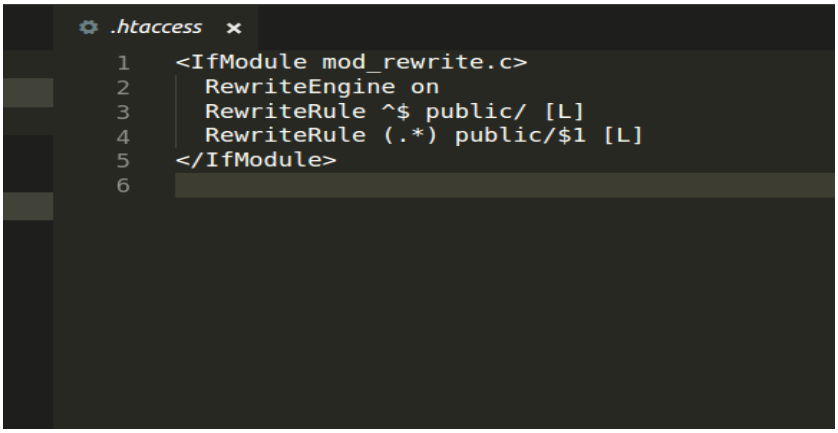


Figure 7 The .htaccess file in the root directory

4.1.1. 2 The Public Folder in the Root Directory

This directory contains index.php files, which is the entry point for all HTTP requests of the application. This directory also houses your assets such as images, JavaScript and Cascading Style Sheets(CSS) files. Aside that, it contains .htaccess map incoming URL as a querying string to index,php, create

4.1.1. 2 The Application Folder in the Root Directory

This is where most of the framework structure are housed and that is where developers will be using a lot to build their applications. It consists of other folders, .htaccess file and a bootstrap file. These provide utilities encompassing the entire framework, loading the files needed to start the framework;

I. 404 folder: Any incoming URL which are not associated with any controller action are reroutes to this folder to inform the user that the requested page does not exist in this application.
II. Config folder: This folder houses the configuration file. The file has information concerning database connection settings, “URLROOT”, “APPROOT” and the name for the project.
III. Controllers Folder: The “C” in the MVC happens in this folder. All controllers associated with an application are created here.
IV. Core Folder: This contains file called Core.php that loads controllers, format incoming URL’s into MVC friendly URL. Also, it contains BaseController.php file that helps to load views and models. The remaining file in this folder is Database.php that provides an abstract layer for database operations.

V. Helpers Folder: This is where other utilities needed by developers to perform specific functions are created and housed.

VI. Models Folder: The folder will house all files needed to perform and maintain data and business logic.

VII. Views Folder: This folder serves as the directory where all files needed to be served to the client are housed.

VIII. .htaccess File: The .htaccess file helps to map the incoming URL as a querying string. With the help of .htaccess file in the root directory the domain name appends public to the URL to help get access to the starting file. For instance if the domain name is "http://oliver.com", it becomes "http://oliver.com/public". The .htaccess file in this directory then helps to reverse the newly created URL to the old URL.

IX. bootstrap.php File: This file loads all the files and instantiate the class needed to start the entire project.

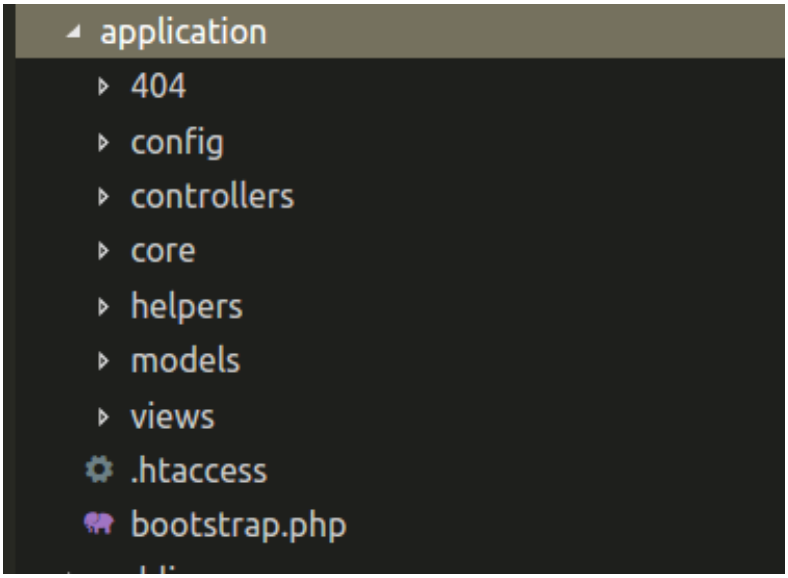


Figure 8 Application Folder Root Directory

4.1.1 The MVC Architectural Pattern

As discussed in chapter 3.2, Introduction to MVC Architecture, this section will provide the implementation of the of the pattern this framework.

4.1.1.1 The Controller.

Each HTTP request comes in as a route. The route as reformatted into MVC friendly URL's create an association between the route and the controller. The localhost/placeInterns form the hostname of the application. The remaining form the controller action. Here, the name of the controller is called Users and has a method called login.

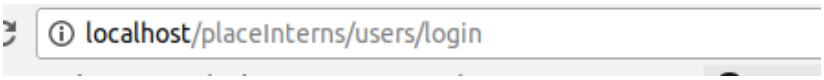


Figure 9 MVC Friendly URLs from controller action

```php
<?php
class Users extends BaseController{
    public function __construct(){
        $this->userModel = $this->loadModel('User');
    }
    public function login(){
      // Check if logged in
      if($this->isLoggedIn()){
        redirect('posts');
      }

      // Check if POST
      if($_SERVER['REQUEST_METHOD'] == 'POST'){
        // Sanitize POST
        $_POST  = filter_input_array(INPUT_POST, FILTER_SANITIZE_STRING);
```

Figure 10 Creation of Controller Class

Within the controller action, thus the methods of the controller, two main things typically occur: the models are used to retrieve all of the necessary data from a data store; and that data is passed to a view, which renders the requested page. The data retrieved via the models is generally added to a data structure (like a list or dictionary), and that structure is what's sent to the view.

4.1.1.2 The Model.

When a controller needs to get data to present to the client, the controller is not concerned with how the data is obtained, all it cares about it the data. The model is responsible for all processes regarding to data. In this framework, the default data store is MySQL. However, the framework provides the convenience to use other data store like SQLite, PostgreSQL. To achieve flexible switching between data store, the framework made use of PDO. PDO stands for PHP Data Object. It is the preferred way to connect to databases because it provides consistent API to do so. Consistent API means that if the framework is not using PDO, then need functions like mysqli_connect, etc. is needed to interact with MySQL data store, and if using PostgreSQL something like pgsql_connect to connect with PostgreSQL data store. With PDO, all these kind of go away because regardless of the data store needed, PDO provides simple API to do so and can easily switch between different data stores.

```php
<?php
/*
 *PDO Database Class
 * Connect to Database
 * Create Prepared Statements
 * Bind Values
 * Return rows and results
 */

declare(strict_types=1);

class Database {
    private $host = DB_HOST;
    private $user = DB_USER;
    private $pass = DB_PASS;
    private $dbname = DB_NAME;

    private $dbh;
    private $error;
    private $stmt;

    public function __construct() {
        // Set DSN
        $dsn = 'mysql:host=' . $this->host . ';dbname=' . $this->dbname;
        $options = array (
            PDO::ATTR_PERSISTENT => true,
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
        );
        // Create a new PDO instanace
        try {
            $this->dbh = new PDO ($dsn, $this->user, $this->pass, $options);
        }       // Catch any errors
        catch ( PDOException $e ) {
            $this->error = $e->getMessage();
        }
```

Figure 11 The Database Class

```php
<?php
/**
 * @author Oliver Mensah <https://omensah.github.io/resume>
 * @link https://github.com/OMENSAH/barePHP
 * Model Class
 */
class User {
    private $db;

    public function __construct(){
      $this->db = new Database;
    }

    // Add User / Register
    public function register($data){
      // Prepare Query
      $this->db->query('INSERT INTO student (name, email,password)   VALUES (:name, :email, :password)');
      // Bind Values
      $this->db->bind(':name', $data['name']);
      $this->db->bind(':email', $data['email']);
      $this->db->bind(':password', $data['password']);
      //Execute
      if($this->db->execute()){
        return true;
      } else {
          echo "BAd";
        return false;
      }
    }
```

Figure 12 Creation of Model Class

4.1.1.3 The View

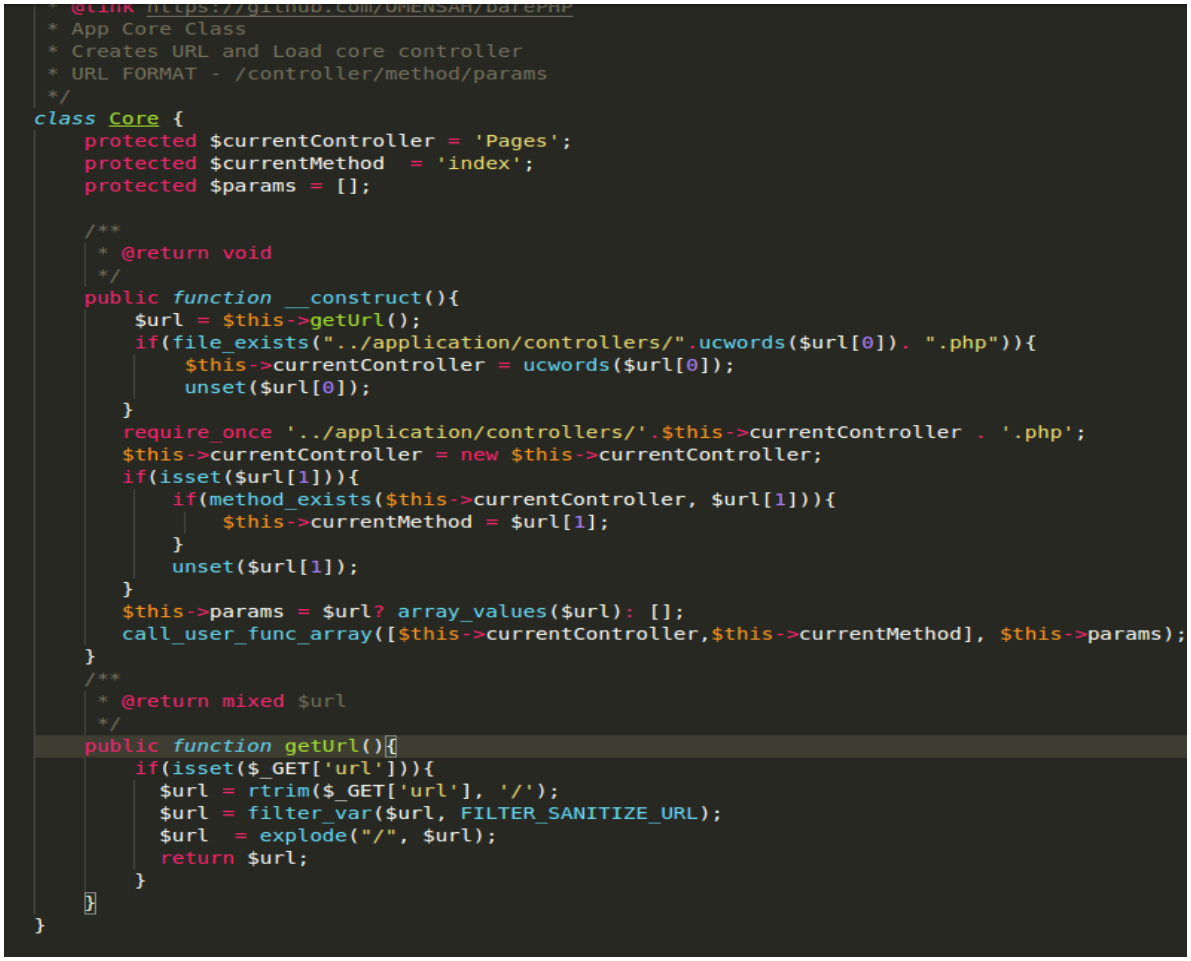
Once the controller method has completed any (Input Output) IO or CPU bound operation, the controller continues to load the view to be rendered to the client as an Hypertext Markup Language(HTML)webpage

```php
<?php require APPROOT . '/views/includes/header.php'; ?>
<div class="row">
    <div class="col-md-6 mx-auto">
      <div class="card card-body bg-light mt-5">
        <?php flash('register_success'); ?>
        <h2>Login</h2>
        <p>Please fill in your credentials to login.</p>
        <form action="<?php echo URLROOT; ?>/users/login" method="post">
          <div class="form-group">
              <label>Email:<sup>*</sup></label>
              <input type="text" name="email" class="form-control form-control-lg <?php echo (!empty($data['email_err'])) ?
               'is-invalid' : ''; ?>" value="<?php echo $data['email']; ?>">
              <span class="invalid-feedback"><?php echo $data['email_err']; ?></span>
          </div>
          <div class="form-group">
              <label>Password:<sup>*</sup></label>
              <input type="password" name="password" class="form-control form-control-lg <?php echo (!empty($data
              ['password_err'])) ? 'is-invalid' : ''; ?>" value="<?php echo $data['password']; ?>">
              <span class="invalid-feedback"><?php echo $data['password_err']; ?></span>
          </div>
          <div class="form-row">
            <div class="col">
              <input type="submit" class="btn btn-success btn-block" value="Login">
            </div>
            <div class="col">
              <a href="<?php echo URLROOT; ?>/users/register" class="btn btn-light btn-block">No account? Register</a>
            </div>
          </div>
        </form>
      </div>
    </div>
</div>
```

Figure 13 Creation of View

4.1.2 The URL Manipulation

Anytime there is an HTTP request, the, .htacess file in the root directory of the public folder maps the corresponding URL as a querying string parameterized by a URL variable as key and the incoming URL as the value. The Core.php file in the core folder of the application directory retrieves the incoming URL sent as a querying string, parses it and generate MVC friendly URL as well as instantiate the corresponding controller. When the controller executes the associated method to fulfill the request made.



```
 * @link https://github.com/JMENSAH/barePHP
 * App Core Class
 * Creates URL and Load core controller
 * URL FORMAT - /controller/method/params
 */
class Core {
    protected $currentController = 'Pages';
    protected $currentMethod  = 'index';
    protected $params = [];

    /**
     * @return void
     */
    public function __construct(){
        $url = $this->getUrl();
        if(file_exists("../application/controllers/".ucwords($url[0]). ".php")){
            $this->currentController = ucwords($url[0]);
            unset($url[0]);
        }
        require_once '../application/controllers/'.$this->currentController . '.php';
        $this->currentController = new $this->currentController;
        if(isset($url[1])){
            if(method_exists($this->currentController, $url[1])){
                $this->currentMethod = $url[1];
            }
            unset($url[1]);
        }
        $this->params = $url? array_values($url): [];
        call_user_func_array([$this->currentController,$this->currentMethod], $this->params);
    }
    /**
     * @return mixed $url
     */
    public function getUrl(){
        if(isset($_GET['url'])){
            $url = rtrim($_GET['url'], '/');
            $url = filter_var($url, FILTER_SANITIZE_URL);
            $url  = explode("/", $url);
            return $url;
        }
    }
}
```

Figure 14 URL manipulation

4.1.3 The Registry

The framework comes with core functions, such as database access, URL routing, user authentication, etc. The registry pattern will help to keep a central repository of objects associated with the framework.

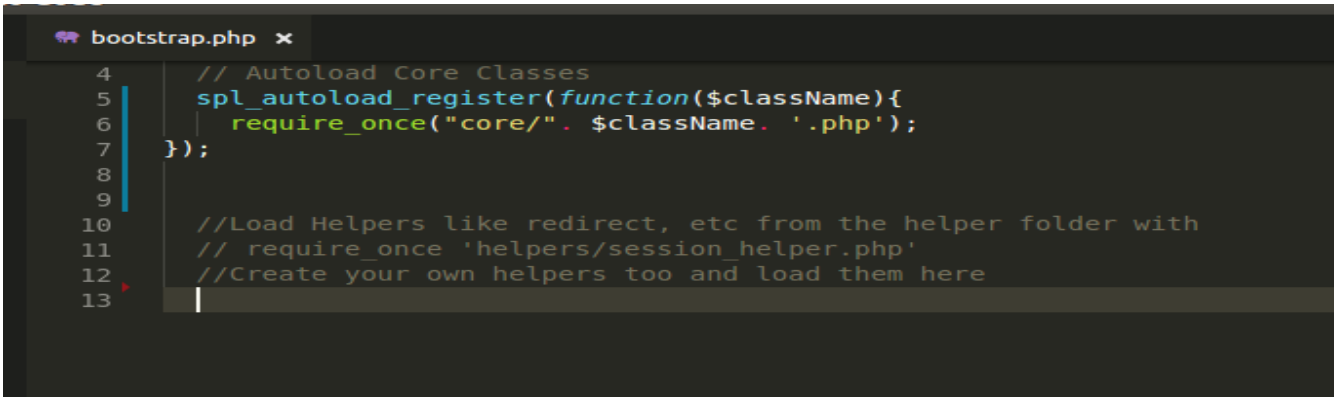


Figure 15 The Framework entry point

Chapter 5: System Testing

This chapter focuses on system testing. This is to ensure that the system meets expectation. When a developer builds an application with this framework, the developer expects to see that the incoming request hits the expected URL or endpoint. The data being retrieved by the control to be displayed through the view should display the exact content as expected (HTML format). To achieve these expectations, functional or integration testing approach is used to test the entire application rather than testing each component separately.

5.1 Functional Testing with PHPUnit, Goutte and Guzzle.

Functional testing looks at how the entire application functions at a whole when all the components are put together. PHPUnit is a programmer-oriented testing framework for testing PHP built applications (Bergmann., 2001). Guzzle is a PHP HTTP client that makes it easy to send HTTP requests (Dowling, 2015). Goutte on the other hand is a simple PHP web Scraper which provides nice API to extract information form the HTML responses (Potencier, n.d.). The Goutte will help make a request, and the response coming as HTML content will be scrap with Goutte. Finally, the output is tested with PHPUnit to find out if the response meets the expected results.

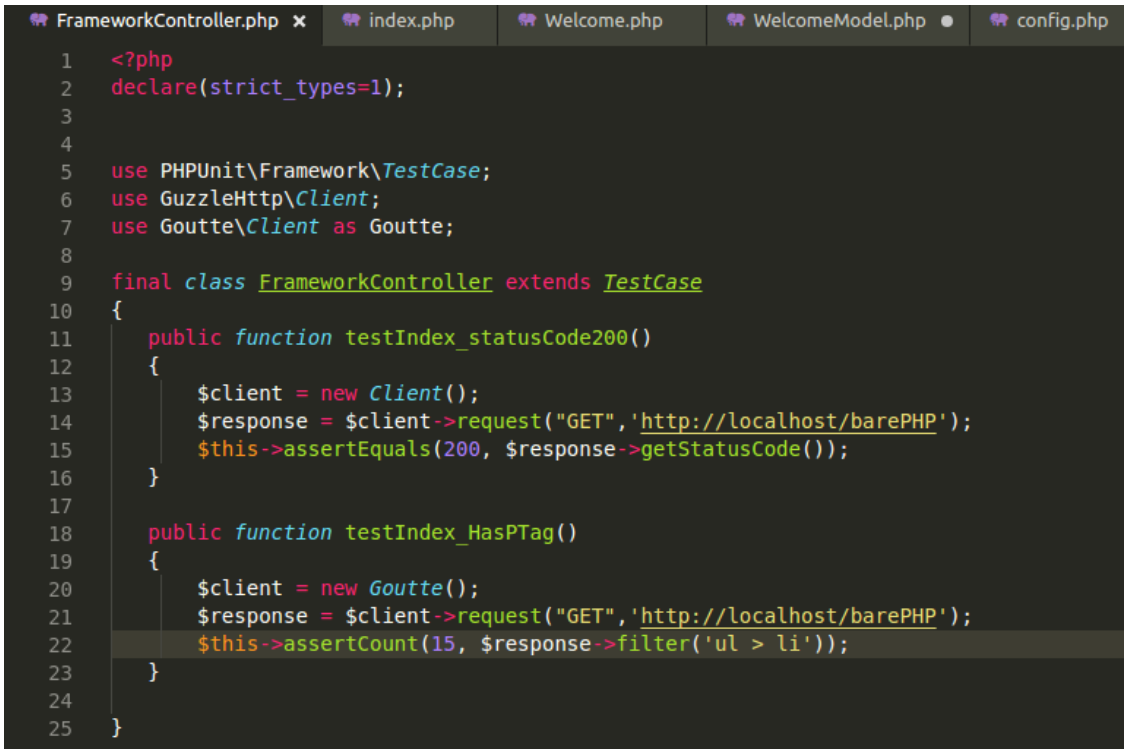


Figure 16 Functional Testing

The two assertions from the test framework work as expected according to the test result. Henceforth, the framework will be performing as expected when used in an application development and on production.

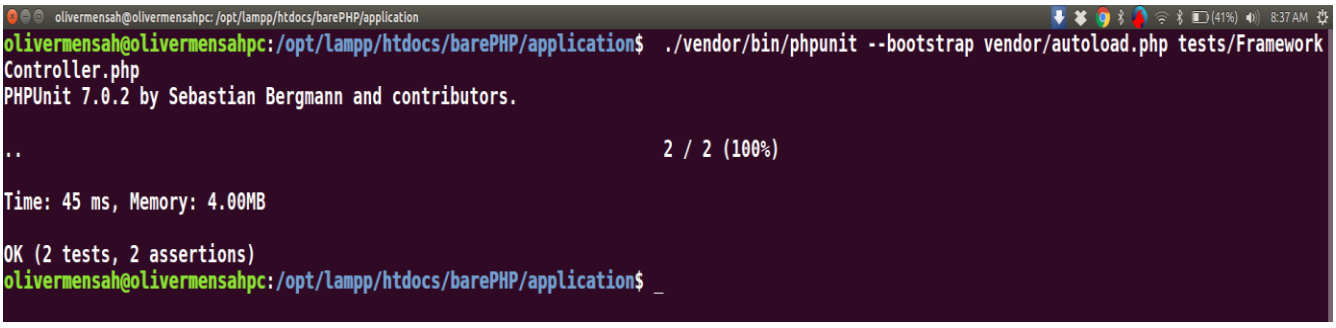


Figure 17 Results of Functional Testing

5.2 Application Performance

About chapter 3.6.1.3, Safety and Performance, it was stated that Average Response Time, Error Rates, Application and Server CPU Usage, Request Rate will be the main metric for measuring performance of any web application built with this micro-framework.

From the test output, the Average Response Time is 45ms, which is fairly a good performance because the requester waits for a small time for ring processing to complete (Laravel Forum, 2014). Also, the minimum amount of memory used in buffering any request is 4.00MB. This request affirms that less amount of memory is needed to buffer request that will not cause contention of data to backpressure subsequent requests. Both Error Rate and Request Rate yields positive outcome hence the application displays the requested data without any error as well as making sure all requests are being processed.

Chapter 6: Conclusion and Recommendation

6.1 Conclusion

The main purpose of this project is to use software engineering principles, methodologies and system architecture to provide a minimal framework for developers to build any simplified web application in PHP. With this approach, any developer can take the barePHP framework and build on top their application, and as more complexity is demanded, they have the flexibility to use any libraries available to accomplish this task.

6.2 Limitation

The framework is not yet a matured framework like Laravel and others. There a lot of complexity that needs to be handled such as it not easier now to switch between different database like from MySQL to NoSQL database. For now, the framework works with only MySQL database as well as execute on only Apache Server.

6.2 Recommendation

The first version of the barePHP framework has an implementation of the design and functionalities described in the requirement specification. Due to the time limitation (one semester), the implementation of the application is not matured as compared to other frameworks that have been in existence over years and has a huge community of developers always adding features and updating functionalities. In future works, updated versions of this framework will be implemented where different design principles will be adopted to add extra features to the framework to make it fully matured. For example, the **S**ingle responsibility **O**pen/closed **L**iskov substitution **I**nterface segregation **D**ependency

inversion) SOLID principle can be implemented by focusing on Inversion of Control and Dependency Injection to allow developers to choose any database to work with. Also, it would be more appropriate to make this framework works well on other servers such NGINX servers.

References

Bergmann., S. (2001). *PHPUnit*. Retrieved from PHPUnit: https://phpunit.de

Dowling, M. (2015). *Guzzle*. Retrieved from Guzzle: http://docs.guzzlephp.org/en/stable/

Kobilansky, V. (2017, March 03). *PHP*. Retrieved from SitePoint: , https://www.sitepoint.com/the-state-of-php-mvc-frameworks-in-2017/)

Laravel Forum. (2014, April 03). Retrieved from Laravel Response Executon Time: https://laravel.io/forum/04-03-2014-laravel-responseexecuton-time

Lukač, I. (2015 , November 13). *Blog | A few thoughts about the Symfony framework and PHP in general*. Retrieved from Netgenlabs: https://www.netgenlabs.com/Blog/A-few-thoughts-about-the-Symfony-framework-and-PHP-in-general

Narożny, H. (2017, 2 June). *Code and Tools*. Retrieved from Merixstudio: https://www.merixstudio.com/blog/laravel-vs-symfony-clash-frameworks/

Nawaz, S. (2016, July 29). *Blog*. Retrieved from Cloudways: https://www.cloudways.com/blog/josh-lockhart-interview/

Potencier, F. (n.d.). *Friends Of PHP*. Retrieved from Goutte: https://github.com/FriendsOfPHP/Goutte

RashidahF Olanrewaju, Thouhedul Islam and Nor'ashikin Bte.Ali. (2015). An Empirical Study of the Evolution of PHP MVC Framework. *In Advanced Computer and Communication Engineering Technology - Proceedings of the 1st International Conference on Communication* , pp. 399-410.

Skvorc, B. (2015 , February 27). *PHP*. Retrieved from SitePoint: https://www.sitepoint.com/best-php-framework-2015-survey/

Tommi, M., & Antero, T. (2008). Web Applications – Spaghetti Code for the 21st Century. *2008 Sixth International Conference on Software Engineering Research, Management and Applications* (pp. 319-328). Prague : Institute of Electrical and Electronics Engineers.

W3Techs - World Wide Web Technology Surveys. (2017, December 1). Retrieved from W3Tech Web Technology Survey: https://w3techs.com/technologies/details/pl-php/all/all

Wang, G. (2011). Application of lightweight MVC-like structure in PHP. *2011 International Conference on Business Management and Electronic Information* (pp. 74-77). Guangzhou: Institute of Electrical and Electronics Engineers.