CHAPTER 3

METHODOLOGY

3.1 INTRODUCTION

This chapter describes the methodological aspect and ways involved in the development of an enhanced php framework addressing the evolving needs of modern web applications. This chapter will cover the research design, development approach, and various methodologies used throughout the project. This methodology section serves as a roadmap, explaining the choices and processes acknowledged and addressing encountered limitations. It provides a clear understanding of the steps taken to achieve the objectives of creating an advanced PHP framework for developing web applications.

The main primary objective of developing an enhanced PHP framework is to address the evolving needs of modern web applications, offering a scalable, flexible, and performance-optimized solution.

Developing web applications can be a complex and time-consuming process. PHP frameworks aim to address this challenge by providing a structured foundation and pre-built components, allowing developers to focus on building specific functionalities rather than reinventing the wheel.

3.2 RESEARCH DESIGN

The php framework design started by evaluating well-known php frameworks weakness, downsides, usability issues and design patterns.

From the research made, it was discovered that most php frameworks tends to have some drawbacks which this topic research and design will fix later in the chapters:

1. **Steeper learning curve:** Frameworks like Laravel, Symfony can introduce a steep learning curve for beginners, as they require understanding additional concepts and patterns beyond basic PHP, this can be a barrier for new developers or those unfamiliar with the chosen framework.
2. **Increased dependency:** Relying on a framework can create a dependency on its specific structure and conventions, which frameworks like Zend engine, Phalcon and Laravel seems to have, this can make it difficult to switch frameworks in the future or integrate code from other frameworks.
3. **Configuration Complexity:** Frameworks like Symfony can require extensive configuration, especially for complex applications, which can be time-consuming and error-prone.
4. **Large Footprint:** Frameworks like Zend Engine and Symfony can add significant overhead to small projects due to its large codebase and extensive dependencies.

Building and developing a new custom php framework depends on which model or architectural/design pattern is adopted.

3.2.1 List of Available Design Patterns:

Here is the list of models considered for this project:

1. Model–view–controller (MVC)
2. Model–view–template (MVT).

in this project case and some in-person survey conducted, the MVC architecture seems to offer clean folder directory than MVT, which is adopted by some other php frameworks.

3.3 SYSTEM ARCHITECTURE

The system architecture of the PHP framework typically follows the Model-View-Controller (MVC) design pattern according to the figure 3.1. the pattern typically separates the concerns of the application into three distinct layers:

1. **Models:** The model represents the application's data and business logic. It handles data access, validation, and manipulation.
2. **Views**: The view is responsible for rendering the user interface (UI) based on the data provided by the model and rendered through the controller. It typically generates HTML templates or other markup formats.
3. **Controllers**: The controller acts as the intermediary between the model and the view. It receives user requests, interacts with the model to retrieve or update data, and then selects the appropriate view to render the response to the end users.

This separation of concerns makes the PHP framework more easy, maintainable, and testable. Each layer can be developed and tested independently, and the framework can be easily extended with new components and third-party libraries.

Figure 3.1 diagram of model-view-controller architecture

**3.3.1 ARCHITECTURAL PATTERNS**

In addition to the MVC pattern, PHP frameworks often employ other architectural patterns, such as:

1. Routing System: this is used for mapping URLs to specific controllers and actions and it also enables the framework to interpret and handle different types of requests.
2. Database Interaction: Includes components for database connectivity and interaction and ORM (Object-Relational Mapping) may be used to map database tables to model objects.
3. Middleware: Provides a way to filter HTTP requests and responses, middleware components can perform tasks such as authentication, logging, caching and many more.
4. Template Engine: this manages the presentation layer and generates dynamic content, it allows developers to embed programming logic within HTML templates or PHP file.
5. Template Engine: this manages the presentation layer and generates dynamic content and allows developers to embed programming logic within HTML templates.
6. Utilities and Helpers: this is a collection of tools and helper functions for common task, it streamlines development by providing pre-built functions for tasks like form validation, string manipulation, etc.

Here's a diagram illustrating the architecture and workflow of Fast-volt PHP framework:

Figure 3.2 Fast Volt Framework Workflow

**3.4 Framework Prerequisites**

The custom PHP framework involves several prerequisites to ensure a smooth development process and the creation of a robust and maintainable framework. Here are some key prerequisites and extensions required in developing the framework:

1. PHP Version 8.x
2. Composer (Dependency Manager).
3. MySQL (only if needed)
4. Apache or Nginx Server
5. Ctype PHP Extension
6. cURL PHP Extension
7. DOM PHP Extension
8. Fileinfo PHP Extension
9. Filter PHP Extension
10. Hash PHP Extension
11. Mbstring PHP Extension
12. OpenSSL PHP Extension
13. PCRE PHP Extension
14. PDO PHP Extension.

**3.4.1 PHP Version 8**

PHP 8 is a major release of the PHP language, released in November 2020. It introduced several significant changes and improvements, which includes Just-in-time (JIT) compiler that can improve performance by translating code into machine instructions at runtime and it also enforces stricter type checking by default, improving code accuracy and reducing runtime errors.

As regarding this project, php 8 is preferred because of the features it offers and mainly because it adopts JIT compiler, which is needed for fast code translations and performance.

**3.4.2 Composer**

Composer is a dependency manager for PHP applications. It allows a way to declare the libraries that a project needs and installs them automatically. This simplifies the process of managing third-party code and ensures integrity while working on a project.

Composer is the only package manager php has for now, so, for this project, most third-party libraries/ dependencies will be installed using it.

**3.4.2.1 Dependencies**

The framework development involves some advanced tasks like environment variable loading, testing frameworks and many more, which can only be installed through a dependency manager (composer). Here are the lists of dependencies installed to get the framework development started:

1. Smarty Template: is a template engine for PHP, facilitating the separation of presentation (HTML/CSS) from application logic.
2. Phpdotenv: loads environment variables from .env file to getenv(), $\_ENV and $\_SERVER automagically.
3. PhpUnit: is a programmer-oriented testing framework for PHP (recommended).
4. PHP Cs Fixer: a tool to automatically fix PHP Coding Standards issues.

**3.4.2.2 Hardware Requirements:**

To develop this framework, hardware requirement is very important to be considered. Table 1.1 shows the hardware requirements to develop the framework without running into high CPU usage or any other related hardware issues:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Hardware | Description | Cost |
| 1 | A Laptop will at Least 2 – 3 GB Ram and 2.5 processor speed | Workstation for the project and this operation will be memory intensive. | N 120,000 |
| 2 | Hard Disk Space (500MB) | To save framework data | nil |
| 3 | Good Condition Laptop Cooling Fan | More memories will be used on creating the framework and testing, so that may resolve in overheating and we will need a cooling fan to minimize that problem. | nil |

Table 1.1 Hardware requirements

**3.4.2.3 Server/Development Requirements:**

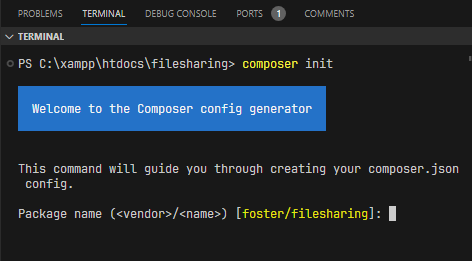
After hardware requirements, server requirements is very important to be considered, table 1.2 shows the server and development software needed to get started in developing the php framework from scratch.

|  |  |  |  |
| --- | --- | --- | --- |
| No | Software | Description | Cost |
| 1 | Xampp (Apache, Mysql) | This tool has almost all things needed to launch a server and has a lot of packages pre-installed which includes Apache Server, PHP and MariaDB, these are the core thing needed in starting a php framework development. | Free |
| 2 | IDE (Vs Code) | There are many IDEs out there, but in my case, we will use Vs-Codes, because It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application. | Free |

Table 1.2 Software requirements

**4.1 Framework Implementation and Development**

The framework development started with getting the server started using Xampp and installing the composer software, then the development configuration started by running the below command on terminal:



This step helps in speeding up development process by generating a `composer.json` file, which contains the installed dependencies and helps distribute them across the framework project workspace by a process called autoloading.

**4.1 Autoloading**

Autoloading is a mechanism that automatically loads classes, interfaces, traits, or other code files as they are needed, without requiring manual inclusion or require statements for each file. Autoloading simplifies the process of managing dependencies and ensures that the necessary code is available when it is needed.

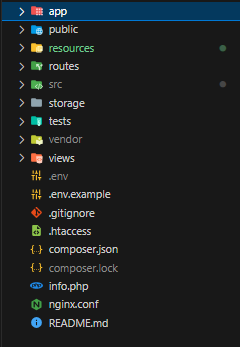
**4.2 Framework Request Router**

A core component of any framework is the Router, which is used to dispatch incoming requests to the right controller action, this is how frameworks like Symfony and Zend knows how to route `/blog/12` URI to a specific class.

In this project, the request router was created from scratch with security and performance in mind, without installing any support libraries to prevent overheads.

**4.3 Framework Directory Structure**

The proposed application structure in this framework is basically the structure of folders, sub-folders and files included in the project. Once the application is installed, we get an overview of the application structure as shown in the image here, which includes various sub-folders and files. The analysis of folders and files, along with their functional aspects is given below:



**4.3.1 Framework App Directory**

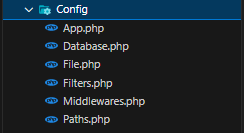
The app directory is created as the main directory for core development, it holds all sections needed for app logics, database interaction, configurations and many more.

Here is the list of proposed subfolders created under App directory:

1. **Config Directory:** it contains all app configurations including App Config, Database Config, Middleware Config, Filters Config and File Configuration
2. **Database**: it controls database table migrations
3. **Enums**: this folder is created to contain any enumeration operations in the application
4. **Http Directory**: this folder was created with two subfolders, which is the **Controller directory** and Middleware directory, the purpose of this sub-directory will be explained better in chapter **3.5.4.1** and **3.5.4.2** respectively.
5. **Model Directory:** this folder was created to contain all database table models needed for the application to function and work properly with the db.

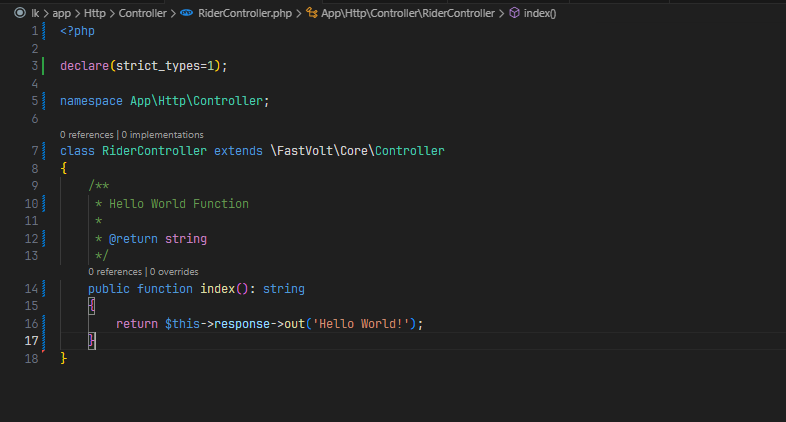
**4.3.1.1 Framework Config Folder**

The config folder includes various configurations and associated parameters required for the smooth functioning of the application. Various files included within the config folder are as shown in the image here. The filenames work as per the functionality associated with them.

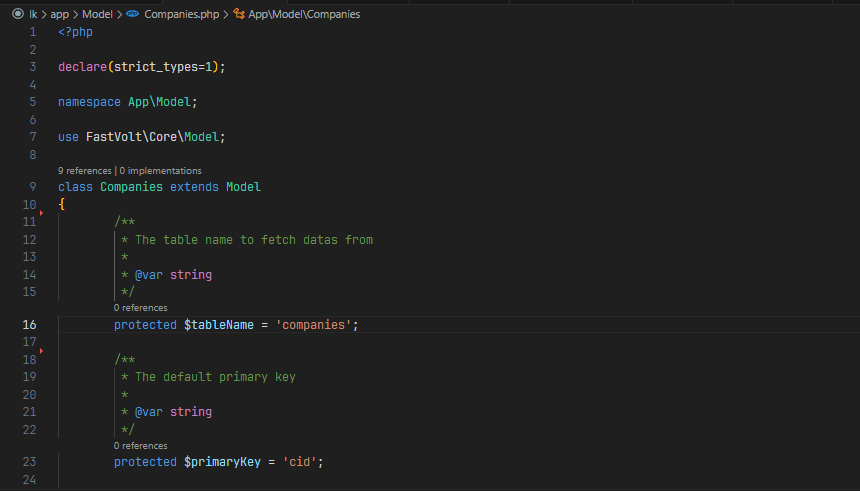


**4.3.1.2 Framework Controller Directory**

The controller directory is created to contain all logics needed to render responses to the client and end-users. A controller files act as a PHP function that either reads information from the request object and returns a Response object or it serves as a place to perform server-side logics. The controller object response could be an HTML page, JSON, XML, a redirect, a 404-error page or anything else. All controller logics are performed through the controller directory in `**app/Http/Controller**` directory, which was created to house all Controller object files. The below picture shows how the http controller file looks like:

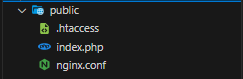


**4.3.1.3 Framework Model Directory**

The model directory is created to contain all database table model objects. A database model file is a single class with various properties like table name and table’s primary key, here is an image demonstration of how a model file looks like:

**4.3.2 Framework Public Directory**

The public folder is the primary root folder which helps in initializing the framework application. It includes the following files and folders:



1. .**htaccess file:** This file gives the server configuration for Apache servers.
2. **index file:** This file is required for the initialization of the framework bootstrap file.
3. **Nginx conf file**: this is the default configuration file for Nginx servers.

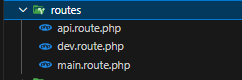
**4.3.3 Framework Resources Directory**

The resources directory/folder contains the files which enhances the framework web application. The sub-folders included in this directory and their purpose is explained below:

1. JS: this folder contains the Javascript files for the application.
2. CSS: this folder contains the CSS files required for styling the application.
3. Images− The assets folder include files such as images, icons and any other graphical file types.

**4.3.4 Framework Routes Directory**

The framework routes directory contains all of the route definitions for the application. By default, several route files are included with the framework, which includes: main.route.php, api.route.php, and dev.route.php.



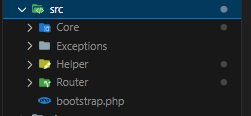
The route files are preloaded by the framework Route Dispatcher method in the source file and processed to the application source bootstrap file.

**4.3.5 Framework Source Directory**

This directory is either pre-installed or included in the composer.json file, which install all dependencies when the install command is triggered by the framework end-client users.

This directory holds the codes behind the framework functionalities and the documentation can be accessed through this url: <https://fastvolt.gitbook.io/docs> and the codes can be accessed through the framework github repository: <https://github.com/fastvolt> .

The directory structure looks like this:



**4.3.6 Framework Storage Directory**

This is the folder that stores all the logs and necessary files which are needed frequently when the framework application is running. The sub-folders included in this directory and their purpose is given below:

* **Caches**: this stores all application cache related files
* **Logs** – this folder stores the exceptions and error logs triggered by the application.
* **Recovery**: this folder stores the default files needed by the application to function, it acts on the rollback function to restore configuration files.

**4.3.7 Framework Tests Directory**

All the unit test cases are included in this directory. The naming convention for naming test case classes is camel-case and follows the convention as per the functionality of the class, the test files in this directory are required to extend PHP-Unit Testcase class, the below image explains further on how a test file should look like in this framework:

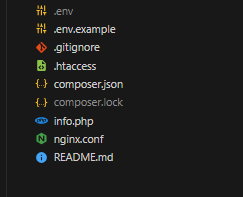


**4.3.8 Framework Vendor Directory**

This framework section is completely based on Composer dependencies, for example to install this framework setup or to include some third-party libraries, etc. Composer installer will be used and it will store all the files in `vendor` directory.

This framework also includes some other files which play a primary role in various functionalities such as GitHub configuration (.gitignore), Apache configuration (.htaccess), Nginx server configuration (.nginx), Application environment configuration (.env, .env.example), Composer configuration (composer.json, composer.lock) and the ReadMe file.

The files included in the application structure are shown below:



**5.1 Conclusion**