Modification MVC Architecture in PHP using Basedata Service Display Pattern

Mohammad Robihul Mufid, Yogi Pratama, Arna Fariza, Saniyatul Mawaddah

Department of Computer Science and Informations Technology
Politeknik Elektronika Negeri Surabaya, Indonesia
mufid@pens.ac.id, yogipratama.ut@gmail.com, arna@pens.ac.id, saniyatul@pens.ac.id

Abstract. Model View Controller (MVC) is a design pattern of structuring the folder structure used to separate between files that have the function of interacting with the database (Model), sending data obtained by the database with an interface page (Controller), and displaying data with an interface page (Views). In the MVC pattern, there is a problem with the base architecture which does not have direct integration with security packages, requires manual activation, and does not provide a special directory for developing security packages. This study aims to develop a pattern known as Data Based Service Display (BSD). BSD was developed to optimize the security package integration process by providing dedicated development directories and space to tolerate the complexities of package injection independently of the MVC pattern. From the results of the tests conducted, it was found that the use of the BSD pattern has a better loading time in rendering scripts than using the MVC pattern and has a tokenization process when submitting data to minimize code sabotage.

Keywords. Design Pattern, Model-View-Controller (MVC), PHP, Basedata Service Display (BSD)

1. Introduction

In the world of website development, there are several tools developed by developers to make it easier for users to create or develop a website effectively and efficiently. Among these tools is the development of a PHP framework which is currently rampant and widely used by developers. Frameworks that are currently popular include the Laravel framework, CodeIgniter, Symfony, Zend, and so on [1]. These frameworks use an architectural pattern that we know as the MVC pattern [2].

MVC architecture is an extension of Model, View, and Controller. MVC is an architecture in website development that divides between data structures, system logic, and interface displays. Where those related to data will be handled by the Model, while those related to logic functions will be handled by the controller, and those related to the display to the user will be handled by the view [3]. Making websites using the MVC architecture can be faster because developers can focus more on working on certain parts. However, this MVC also has drawbacks, including the lack of direct integration

M. U. H. Al Rasyid and M. R. Mufid (eds.), *Proceedings of the International Conference on Applied Science and Technology on Engineering Science 2023 (iCAST-ES 2023)*, Advances in Engineering Research 230,
https://doi.org/10.2991/978-94-6463-364-1_2

and a special directory for the security package system. And if you want to use the security system, you need to activate it manually.

Actually there are several studies that discuss MVC and the use of MVC in developing an application. Among them is research from D. Dobrean et al. [4] who proposed a hybrid approach to analyze the existing layers of the MVC architecture. The approach taken is by combining machine learning methods and static calculations. From the approach proposed, it can provide an understanding to users that the level of accuracy of a system depends on the external library used and how the code is structured to comply with existing guidelines.

Then there are other studies that utilize the MVC architecture to develop a framework, one of which is from M.R. Mufid et al. [5] who proposed an extension of the MVC pattern to be implemented in the Flask framework. The research being carried out is to develop an architectural model, view, and controller to separate functions and folders in the process of handling data, logic, and display interfaces that do not yet exist in the Flask framework. The results of the research conducted explain that the full loading time when implementing the MVC concept in the Flask framework is better than not using the MVC concept.

In addition, there are also other studies that have the same objective as this study, namely to develop an MVC architecture which has several drawbacks, including a complicated structure and time for development and maintenance. The study is from S. I. Ahmad et al. [6] who proposed a model-based approach to simplify the MVC framework in web creation. The approach uses UML Profiles and a model-to-text transformation engine. The results obtained indicate that the approach taken produces MVC-based source code that is more flexible in web development.

Meanwhile, to overcome the security problems found in the MVC architecture [6, 7, 8], this research proposes an architectural approach known as Data Based Service Display (BSD). BSD was designed to simplify the process of integrating security packages by providing a dedicated development directory and storage space to accommodate the complexities of adding packages that are independent of the MVC pattern.

2. Overview Of Mvc Architecture

This section will explain the architecture of MVC (Model View Controller). Where in Figure 1 it is explained that the data structure in the MVC architecture is divided into 3 parts, namely the view which acts to display data in graphical form to the user, then the model which acts to search and process data in the database, and the controller which acts to regulate how the data will be displayed to views.

The workflow of the MVC concept starts with the user requesting data through the graphics in the view, then it is received by the Controller to be forwarded to the model so that the data requested by the user is found. After the data is found, the data will be sent back to the Controller to be regulated. And when the data is ready, the controller will send the data back to the view to be displayed again in graphical form to the user.

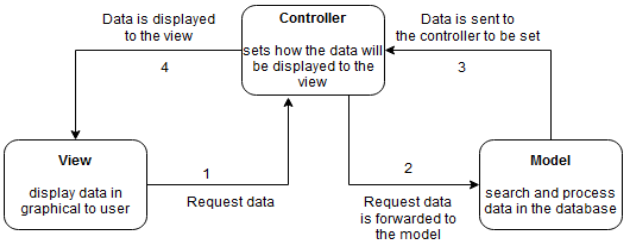

Fig. 1. MVC architecture workflow

3. System Design

In this section we will explain the architecture and workflow of the system directory which will be implemented using the Basedata Service Display (BSD) method in the PHP native framework. Figure 2 shows a diagram of the BSD system folder architecture. There are 6 main directories in the BSD architecture.

1. Devise, is the main folder that contains logic from programming made by programmers involving query programs that will be stored in the Basedata directory, interface programs displayed on the client side that are stored in the Display directory, logic connecting code between programming code from both Basedata, Display and other packages stored in the Service directory, and Devise is a directory that contains the basic interface programs and core programs that will be applied to the classes in each Basedata Service Display directory.
2. Public is a directory that contains files and programs that will be loaded when the program is run and files in the public folder will be published and only in this folder all certain configurations will be displayed in the client browser.
3. Router is a directory that contains programming logic to navigate and run instances of object programs.
4. TempSTR is a special directory that is used as a storage container for files submitted from forms. Files that can be stored in this directory by default are images and documents in pdf format.
5. Vendor is a directory that contains packages and external dependencies taken from the publisher composer which functions to provide flexibility to programmers to be able to add certain packages needed to work on the project.

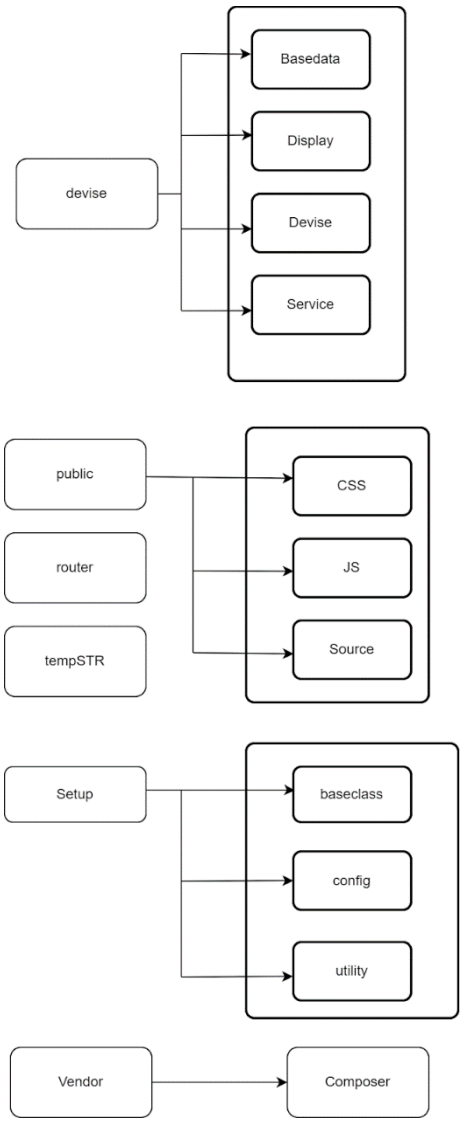

Fig. 2. BSD Architecture Folder

In this study, the implementation of the development of the BSD design pattern in the PHP framework will be divided into several stages.

A. Creating a Package Generator Using Composer

At this stage, we take advantage of the composer feature to save the configuration folder that was previously created. In figure 3, is the folder architecture of the BSD pattern.

Fig. 3. Folder Structure after is generated

Then, in the composer.json file in figure x, information is added regarding the project name, package type, autoloader, author information, and default packages to be installed to support the creation and performance of the BSD design pattern. The information stored in the composer.json file will be used as a reference by Composer to classify packages when they are published to the Composer publisher.

Fig. 4. Composer Generator file using composer.json

B. Implement PSR-4 standard Autoloading Namespace

At this stage, each directory can be automatically integrated with the associated folder architecture thanks to the autoloading system provided by composer. In figure 5, we define a virtual folder that has the value of the actual directory of the BSD architecture. In use, PSR-04 is intended for the implementation of namespace autoloading which aims to provide a virtual path to compromise a file so that it does not experience a crash when encountering the same file name. With the namespace concept, a file will be placed in a virtual path where the actual value of the virtual path is the original path address of the file stored.

```
    "autoload": {
        "psr-4": {
            "devise\\": "devise/",
            "setup\\": "setup/",
            "router\\": "router/"
        }
```

Fig. 5. PSR-4 Autoloading Mechanism

In figure 6, is the process of mapping the file path of the namespace that has been registered in the PSR-04 autoloader which has the original value from the base directory combined with the value of the path in the namespace that was created in the previous composer file.

```
vendor > composer > autoload_psr4.php > ...
<?php

// autoload_psr4.php @generated by Composer

$vendorDir = dirname(__DIR__);
$baseDir = dirname($vendorDir);

return array(
    'setup\\' => array($baseDir . '/setup'),
    'router\\' => array($baseDir . '/router'),
    'devise\\' => array($baseDir . '/devise'),
    'RingCentral\\Psr7\\' => array($vendorDir . '/ringcentral/psr7/src'),
    'React\\Stream\\' => array($vendorDir . '/react/stream/src'),
    'React\\Socket\\' => array($vendorDir . '/react/socket/src'),
    'React\\Promise\\Timer\\' => array($vendorDir . '/react/promise-timer/src'),
    'React\\Promise\\Stream\\' => array($vendorDir . '/react/promise-stream/src'),
    'React\\Promise\\' => array($vendorDir . '/react/promise/src'),
    'React\\Http\\' => array($vendorDir . '/react/http/src'),
    'React\\EventLoop\\' => array($vendorDir . '/react/event-loop/src'),
    'React\\Dns\\' => array($vendorDir . '/react/dns/src'),
    'React\\Cache\\' => array($vendorDir . '/react/cache/src'),
    'Psr\\Http\\Message\\' => array($vendorDir . '/psr/http-message/src'),
    'Fig\\Http\\Message\\' => array($vendorDir . '/fig/http-message-util/src'),
);
```

Fig. 6. Conversion Namespace to actual Path

In figure 7, is the flow of the PSR-04 autoloader in which, a class file that uses a namespace in the specified directory, then automatically, the class file will be registered in the classmap. Then if you want to use another file in the class file, then all you have to do is call the namespace of the class you want to call, then PSR-04 will automatically map to provide another class file called by class file x.

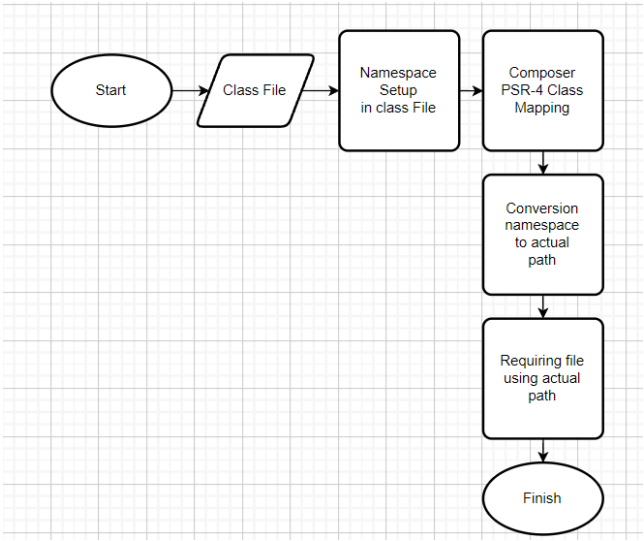

Fig. 7. Conversion Flow PSR-04 Autoloader

C. How the BSD Design Patterns Work in PHP

In figure 8 and figure 9 is the system design of the BSD structure which is implemented in the native PHP Framework. Based on this architecture, data from users will be sanitized before being sent to the Service via the Gemstone process. Apart from sanitizing data, there is a data encryption process in GemStone. Then the data will be sanitized and prepared for auto queries with a late binding scheme using Xgen queries so that on Basedata, users only have to run instances to manipulate data in the database. Then, to display data from the database, the first phase of the data will be fetched via Xgen autoquery in the form of an assoc array, then in the service the data will be managed into a simpler array. Then through Gemstone, the data will undergo an encryption process before being forwarded to the display for display.

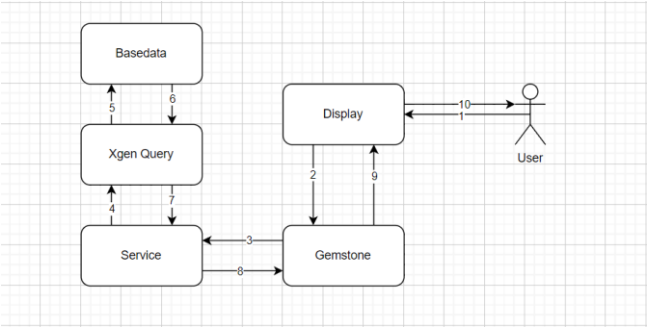

Fig. 8. Arsitektur Directory

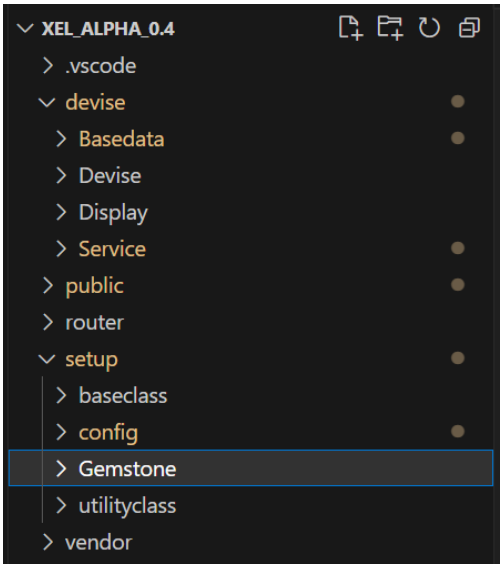

Fig. 9. BSD Work Flow

D. Gemstone Feature Work in PHP

Gemstone is a mineral hardness algorithm that calculates a safety scale ('diamond', 'ruby', 'topaz', 'lapiz lazuli'). Named Gemstone because of the naming of the stone according to the level of hardness of the stone. The higher the rock hardness, the higher the level of data security. Based on Figure 10, the gemstone will start from a data sanitization process to ensure a value does not contain special characters that could potentially be script injectors. Then gemstone will perform level 1 encryption using the basic Gemstone Algorithm, then proceed with encryption using openssl, and finally convert the results of openssl into a token using bin2hex. And data that has passed the Gemstone will be forwarded to the intended Service.

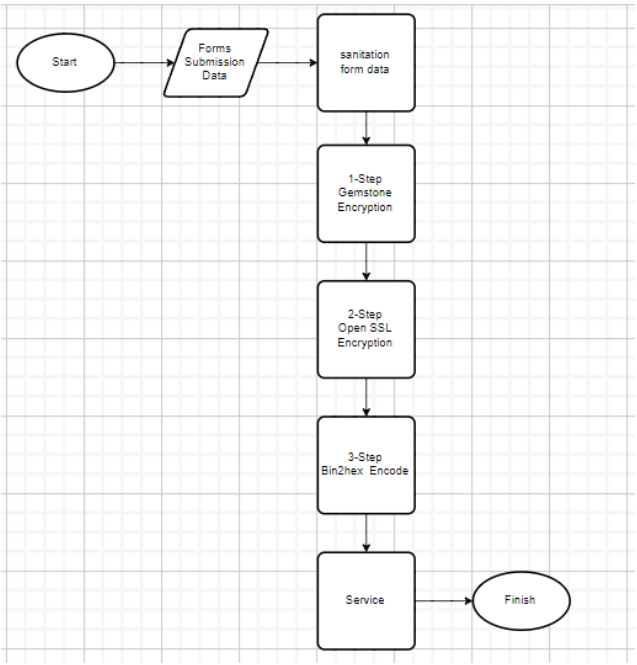

Fig. 10. Gemstone WorkFlow

E. Xgen Query Feature Work in PHP

In figure 11 is the workings of the Xgen feature which in its implementation will start from the service that runs the instance on Xgen which query from that instance will be filled with data, the data will be sanitized first, then the data will be bound first and prepared in a prepared statement for later the function will be triggered in basedata to manipulate the database.

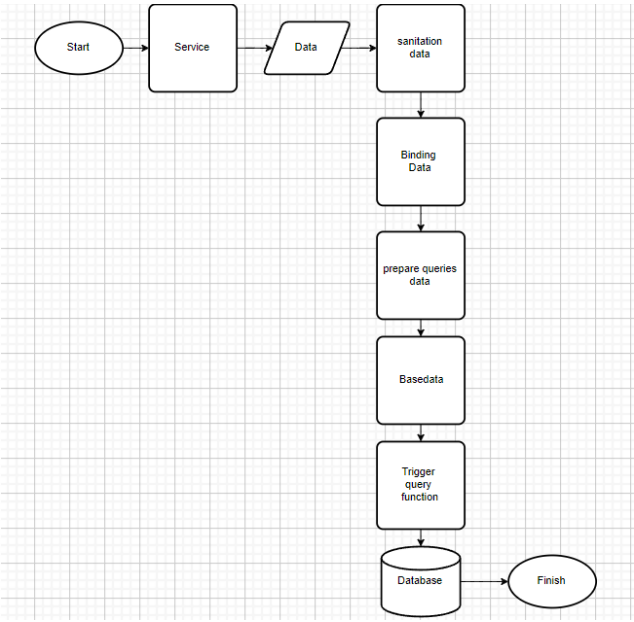

Fig. 11. XgenQuery Builder

II. PERFORMANCE EVALUATION

In this section, testing will be carried out related to the BSD architecture using the PHP programming language. In this test, it will be divided into 3 parts, namely Comparison Result Between MVC and BSD Pattern , BSD Pattern Installation on PHP Native Framework, Performance Test and performance comparison of the BSD architecture with MVC.

A. Comparison Between MVC and BSD Pattern

The result of study show that the Basedata Service Display(BSD) Pattern can improve the maintainability, testability, flexibility, and scalability of applications. The following summarizes the key different between MVC before and after modifications using BSD Pattern :

Table 1. Comparison result between MVC and BSD

No	*Feature*	*MVC*	*BSD*
1	Direct interaction between the view and controller	Direct	Using Subprocess
2	Display Subprocess	not have	Gemstone Process for handling data
3	Data Storage	Directly handled in controller	Stored and Encrypted

No	*Feature*	*MVC*	*BSD*
			on Session
4	Middleware Use	Reliance on middleware for handling data	Reduced dependency on middleware

in this revised table, have included additional details such as specific aspects of data handling and reduce reliance on middleware due to the enhancement in the BSD pattern.

B. BSD Pattern Implementation on PHP Native Framework

In this first part are the steps to do how to implement the BSD pattern in a native PHP framework. Where to implement it there are 3 stages, namely installation via composer, running the development server, and testing access to the landing page.

1) Install via composer

The first step for implementing the BSD pattern is to install it via composer. Figure 12 shows the command to create a project which is "composer create-project –stability=dev xel/BSD framework". Before writing the command, you must first ensure that the location of the project to be installed is in the appropriate position, namely in the xampp/htdocs folder. The "BSD" command above shows the name of the project to be created. Figure 13 shows the BSD folder structure that we managed to create in the xampp/htdocs folder.

```
tirpitz@DESKTOP-3HINETM MINGW64 /c/xampp/htdocs
$ composer create-project --stability=dev xel/framework BSD
```

Fig. 12. Command to create a BSD project

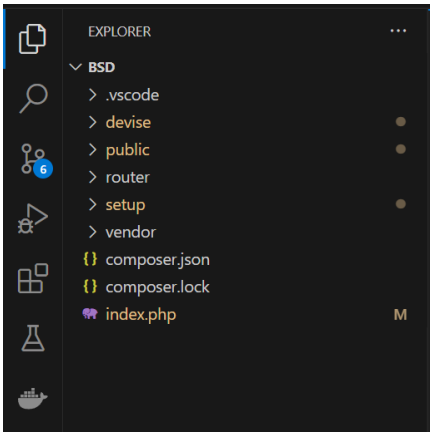

Fig. 13. Structure folder BSD

2) Running the Development server

After we have successfully installed the project, the next step is to run the project. Figure 14 shows how to run the project using the command “php -S localhost:8000”.

Fig. 14. Structure folder BSD

3) Test the Page's landing page access

If the project is successfully executed, then the website that we are building can be started immediately. To start the BSD web, we can go to the landing page by opening a browser, then writing "localhost:8000". Figure 15 shows the landing page view of the BSD project.

Fig. 15. Plant landing page BSD

C. Performance Test and Performance Comparison of Architectures

This section will discuss the analysis of PHP programming performance using the BSD and MVC patterns using the GTmetrix tools. On the websites that we have provided using the MVC and BSD patterns, performance will be measured using the Grade Matrix, , LCP, CLS, Full Load Time parameters.

1) Head To Head Performance Overview

Based on figure 16, the BSD architecture has an advantage in the LCP section where the loading time in rendering scripts is slightly better than the MVC architecture. then on CLS, BSD has a slightly better advantage in terms of changing the dimensions of the content loaded by the website. And even though BSD has a larger page size than MVC, it can still compete with better realtime.

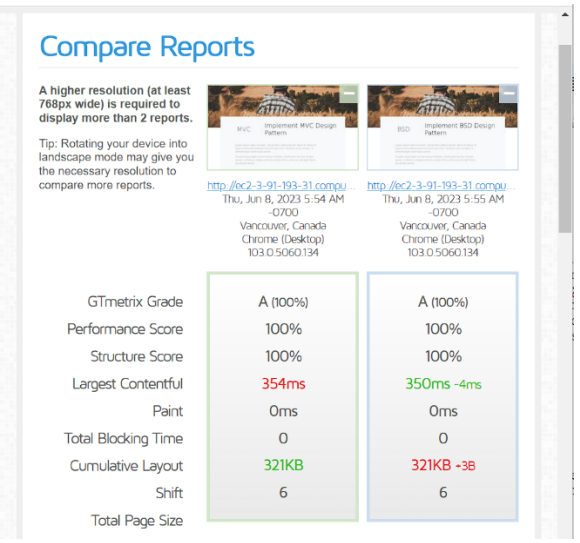

Fig. 16. Overview of Performance

2) Strength of BSD in Performance

In figure 17, the total load time has the same duration, but for the consistency of each process, BSD tends to be more consistent in each process.

Fig. 17. Overview of Performance Load Time

3) Benchmark test for handling Request

In figure 18 and 19, the BSD pattern also impacts performance in Request Per Second handler. In this test using 2 models, the test includes

PHP with MVC and BSD pattern to return GET requests with 1000 connections and using a single thread. In terms of RPS BSD has a lot of requests handling 860 over 98 per second. In terms of latency the mvc wins and have a different time of 0.09 second. and at last the transfer rate on BSD have more data per second than mvc at least on this has 2.95 Megabyte over 431,25 Kilo Bytes.

```
[tirpitz@fedora ~]$ wrk -t1 -c1000 -d10s http://localhost:8000/auth/
Running 10s test @ http://localhost:8000/auth/
  1 threads and 1000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.09s   233.51ms   1.28s    90.25%
    Req/Sec     0.86k    99.76     1.02k    83.00%
  8560 requests in 10.07s, 29.70MB read
  Socket errors: connect 0, read 8560, write 0, timeout 0
Requests/sec:    849.67
Transfer/sec:      2.95MB
[tirpitz@fedora ~]$
```

Fig. 18. Benchmark Test in PHP native framework with BSD

```
[tirpitz@fedora ~]$ wrk -t1 -c1000 -d10s http://localhost:8000/api/documentation
Running 10s test @ http://localhost:8000/api/documentation
  1 threads and 1000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.00s   574.53ms   1.99s    57.95%
    Req/Sec    97.74     4.63   101.00     74.00%
  974 requests in 10.10s, 4.25MB read
  Socket errors: connect 0, read 974, write 0, timeout 779
Requests/sec:     96.44
Transfer/sec:    431.25KB
```

Fig. 19. Benchmark Test in PHP native framework with MVC(Lumen)

4) First Secure In form Submission CSRF Token By default

In figure 20 automatically in the BSD design pattern it has integrated with the CSRF protection security package which will add tokens to the data submission process both authorized and non-authorized to minimize code sabotage which contains requests from outside the website to penetrate the server and data sent.

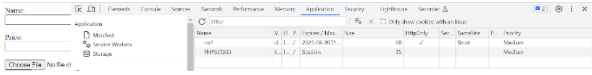

Fig. 20. First Secure CSRF

4. Conclusion

MVC is a concept that is popularly used by developers in website development because it makes it easier to work on and divide tasks. However, this MVC concept also has drawbacks in the integration process with security packages. This study aims to approach and develop the MVC architecture known as Data Based Service Display (BSD). The BSD design pattern is one of the methods used in managing the folder architecture and placing the core program code that facilitates related built-in protection packages. BSD has an architecture that is focused on managing data transmission security at the data delivery layer from display to service or vice versa and provides shortcut features to make it easier to sanitize data without the need to define code repeatedly when you want to use it on a different architecture. The test also found that using

the BSD pattern has better loading times for rendering scripts than using the MVC pattern and has a tokenization process when sending data to minimize code tampering.

References

1. H. Abutaleb, A. Tamimi, and T. Alrawashdeh,: "Empirical Study of Most Popular PHP Framework," In 2021 International Conference on Information Technology (ICIT), pp. 608-611. IEEE, (2021).
2. M. Laaziri, K. Benmoussa, S. Khoulji, K. M. Larbi, and A. E. Yamami, : "A comparative study of laravel and symfony PHP frameworks," International Journal of Electrical and Computer Engineering, vol. 9, no. 1, pp. 704, (2019).
3. A. Subari, S. Manan, and E. Ariyanto, : "Implementation of MVC (Model-View-Controller) architecture in online submission and reporting process at official travel warrant information system based on web application," In Journal of Physics: Conference Series, vol. 1918, no. 4, p. 042145. IOP Publishing, (2021).
4. D. Dobrean, and L. Diosan, : "A Hybrid Approach to MVC Architectural Layers Analysis," In ENASE, pp. 36-46. (2021).
5. M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, and I. F. Rochimansyah, : "Design an mvc model using python for flask framework development," In 2019 International Electronics Symposium (IES), pp. 214-219. IEEE, (2019).
6. S. I. Ahmad, T. Rana, and A. Maqbool. : "A Model-Driven Framework for the Development of MVC-Based (Web) Application." Arabian Journal for Science and Engineering 47, no. 2 (2022): (1733-1747).
7. M. R. Mufid, A. Basofi, I. Syarif, and F. Sanjaya. : "Estimated vehicle fuel calculation based on Google map real time distance." In 2019 International Electronics Symposium (IES), pp. 354-358. IEEE, (2019).
8. M. R. Mufid, N. R. K. S. Putri, and A. Fariza. : "Fuzzy Logic and Exponential Smoothing for Mapping Implementation of Dengue Haemorrhagic Fever in Surabaya." In 2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC), pp. 372-377. IEEE, (2018).