# **LAVALUST DEVELOPMENT**

Contents

[**LAVALUST DEVELOPMENT** 1](#_Toc151070529)

[INSTALLATION 3](#_Toc151070530)

[CONFIGURATION 3](#_Toc151070531)

[FIRST RUN 4](#_Toc151070532)

[REMOVING INDEX.PHP 4](#_Toc151070533)

[PARTS OF URL 5](#_Toc151070534)

[CSRF 5](#_Toc151070535)

[Default Controller & Method 6](#_Toc151070536)

[CONTROLLER 7](#_Toc151070537)

[ROUTE 8](#_Toc151070538)

[Views 9](#_Toc151070539)

[Passing data from Controller-View and View-Controller 10](#_Toc151070540)

[Controller to View 10](#_Toc151070541)

[View to Controller 11](#_Toc151070542)

[OPTIMIZING OUR CODE 13](#_Toc151070543)

[DATABASE CONFIGURATION 15](#_Toc151070544)

[Database Library 15](#_Toc151070545)

[MODELS & QUERY BUILDER 16](#_Toc151070546)

[Loading the Model 17](#_Toc151070547)

[INSERT INTO (CRUD) 17](#_Toc151070548)

[SELECT STATEMENT (CRUD) 18](#_Toc151070549)

[SELECT only one (1) row (CRUD) 19](#_Toc151070550)

[Helpers 20](#_Toc151070551)

[Loading Helper 21](#_Toc151070552)

[Built-in Helpers 22](#_Toc151070553)

[Using Autoload 22](#_Toc151070554)

[Create your own Helper 23](#_Toc151070555)

[Links, and Public / Assets (css, js, images, etc) form Views 24](#_Toc151070556)

[Public / Assets 24](#_Toc151070557)

[Links 26](#_Toc151070558)

[Library 28](#_Toc151070559)

[Form Validation Library 28](#_Toc151070560)

[(IO) Input and Output Class 30](#_Toc151070561)

[Session Library 30](#_Toc151070562)

[Retrieving Session Data 30](#_Toc151070563)

[Adding Session Data 31](#_Toc151070564)

[Removing Session Data 31](#_Toc151070565)

[Security 32](#_Toc151070566)

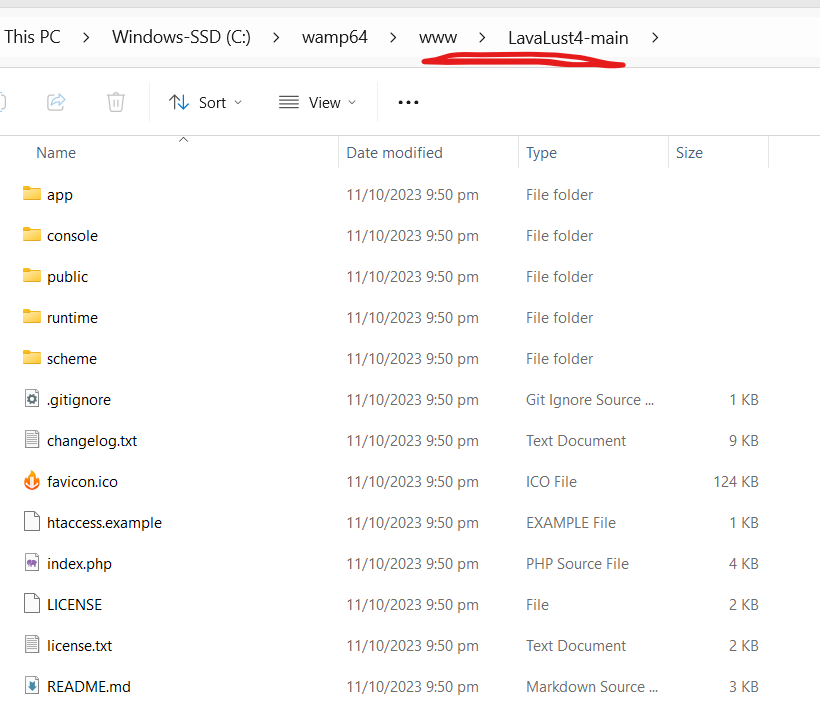
[Before Deploying your application 32](#_Toc151070567)

INSTALLATION

* **DOWNLOAD**

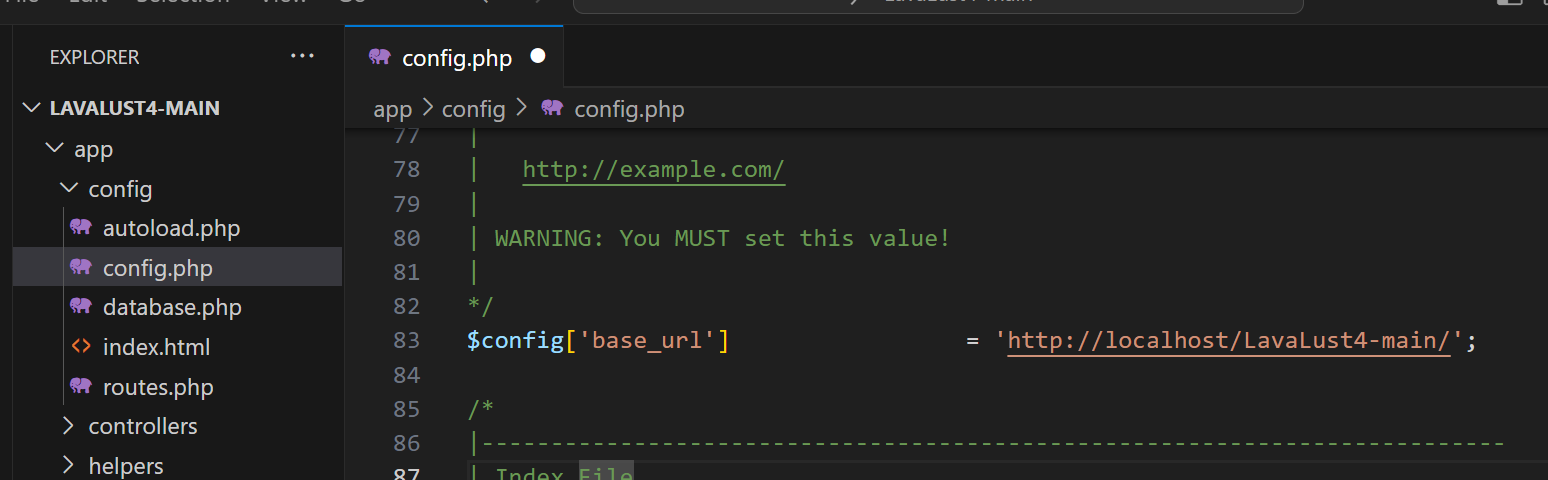
<https://github.com/ronmarasigan/LavaLust4>

* Extract in **www** folder (for wamp) or **htdocs** folder (for xamp).



CONFIGURATION

* Open the **app/config/config.php**
* Go to **$config[‘base\_url’] = ‘’;** and add your local project url



FIRST RUN

* Open your browser and go to your local project url <http://localhost/LavaLust4-main/>



REMOVING INDEX.PHP

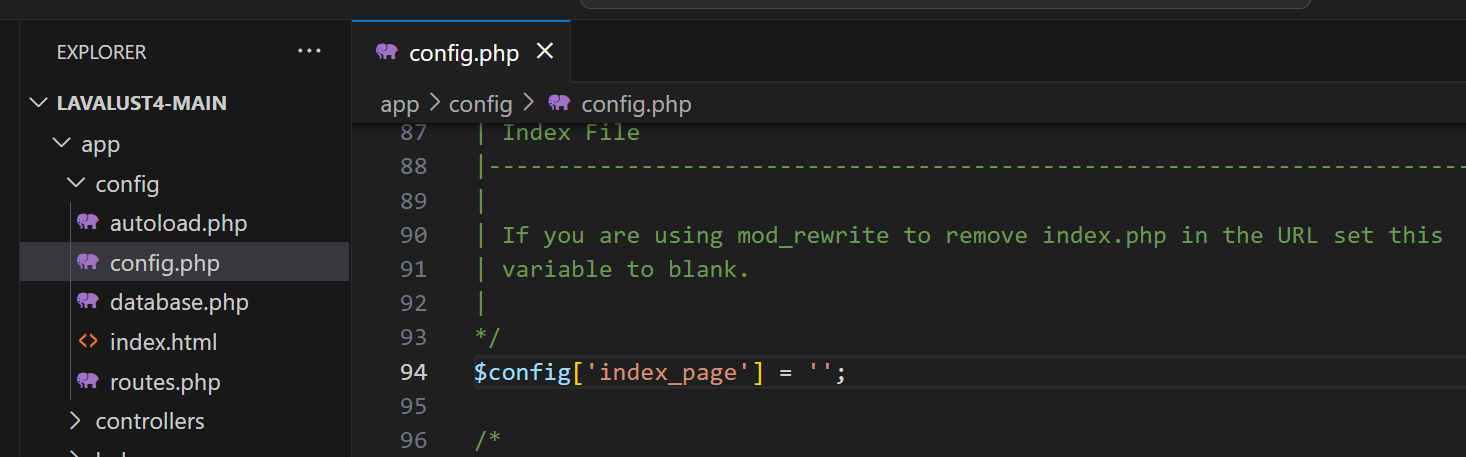
* **Removing the index.php file**

By default, the **index.php** file will be included in your URLs:

example.com/*index.php*/news/article/my\_article

* Empty the value of **$config['index\_page'] = 'index.php';**

*Note: Visit* [*https://lavalust.netlify.app/*](https://lavalust.netlify.app/) *for more details*



* Add/create **.htaccess** to your project root directory and add the following code.

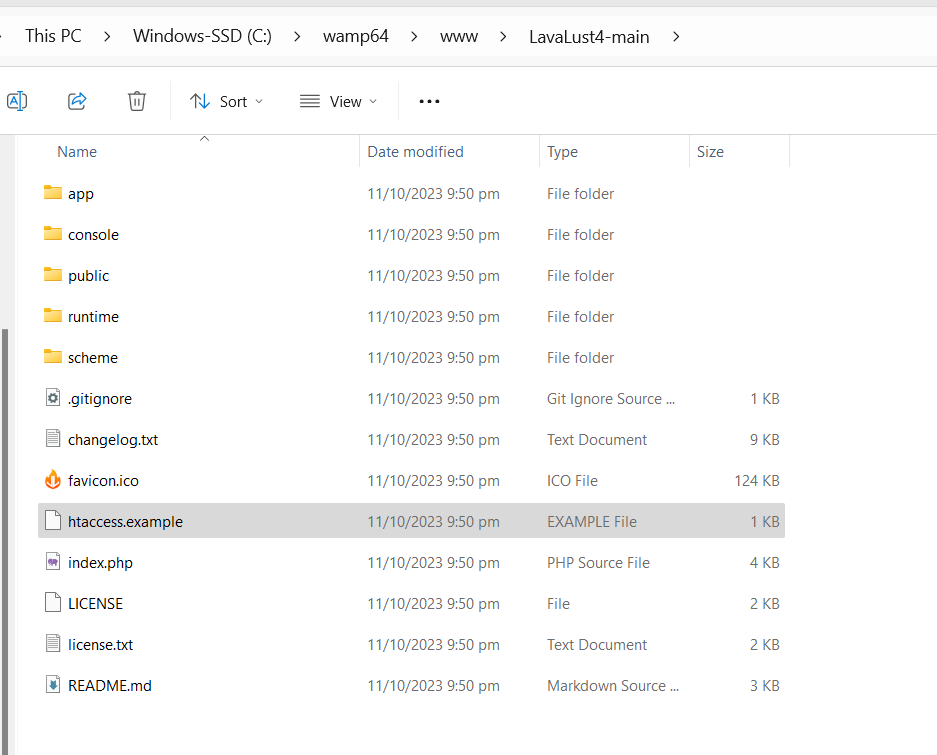
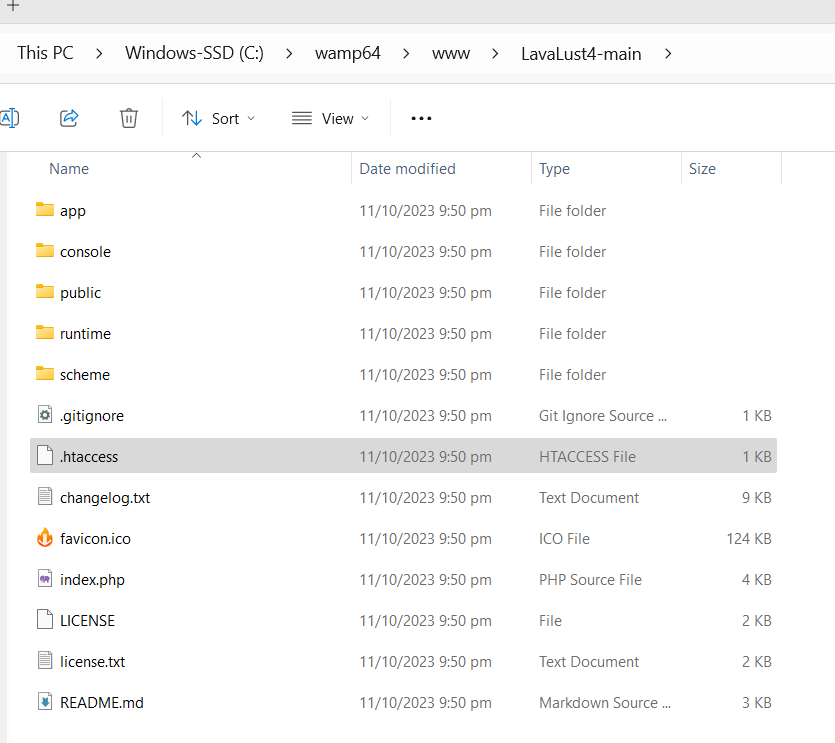
RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.\*)$ index.php/$1 [L]

* In Lavalust, just rename the **htaccess.example** file into **.htaccess**

PARTS OF URL

* **URI Segments**

<https://Example.com/Class/function/id>

<https://Example.com/ClassController/method/parameterValue>

<https://Example.com/User/profile/21>

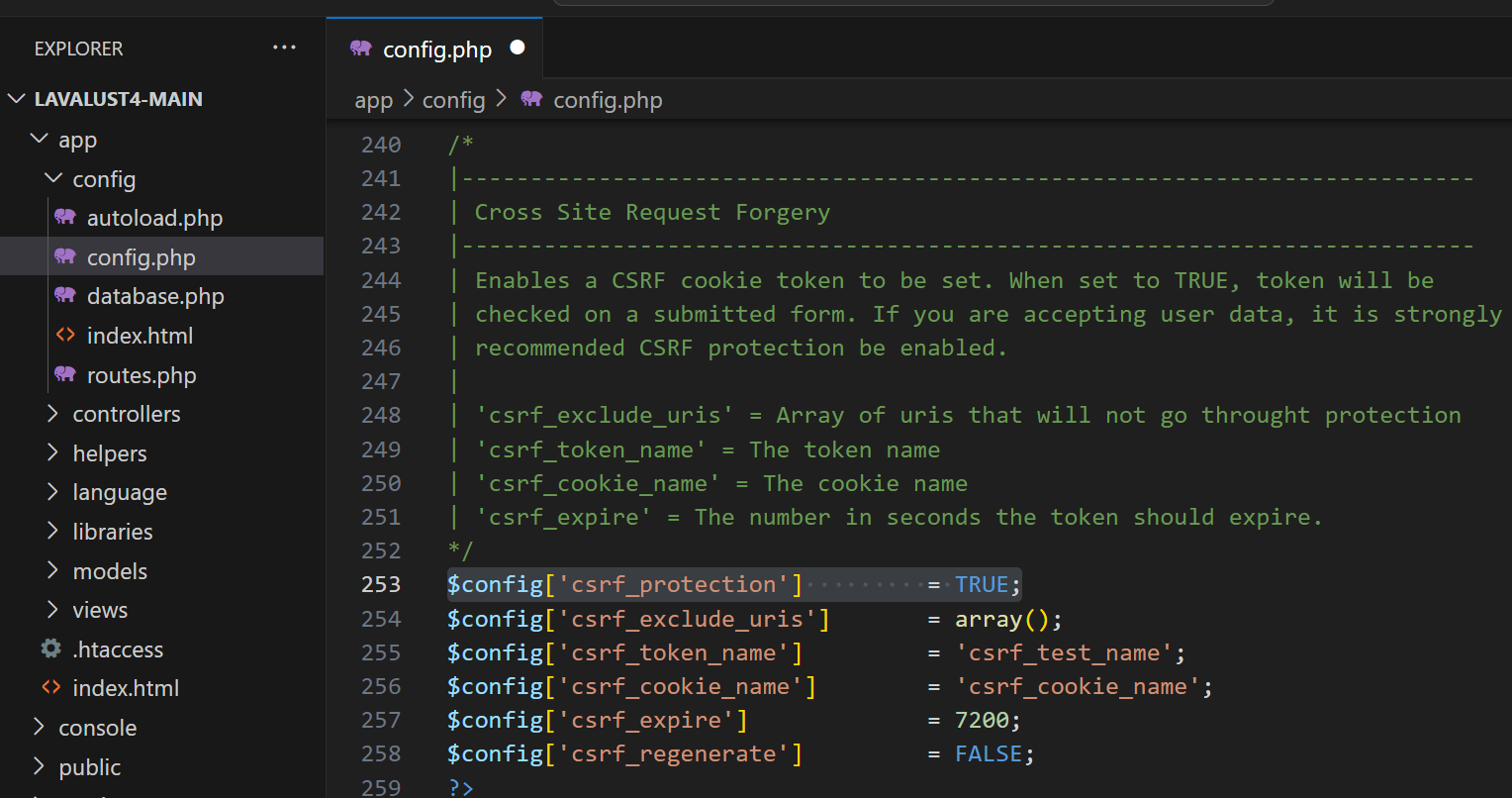
The segments in the URL, in following with the MVC approach, usually represent:

* + The first segment represents the **controller class** that should be invoked.
  + The second segment represents the class **function**, or **method**, that should be called.
  + The third, and any additional segments, represent the **ID** and **any** variables that will be passed to the controller.

CSRF

* Other important security that you must use in **config.php** is the CSRF protection

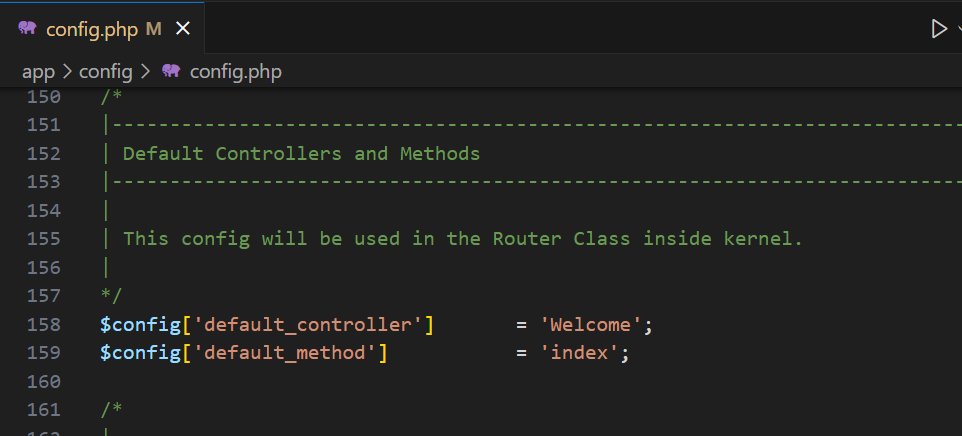
Just change the value into **TRUE**



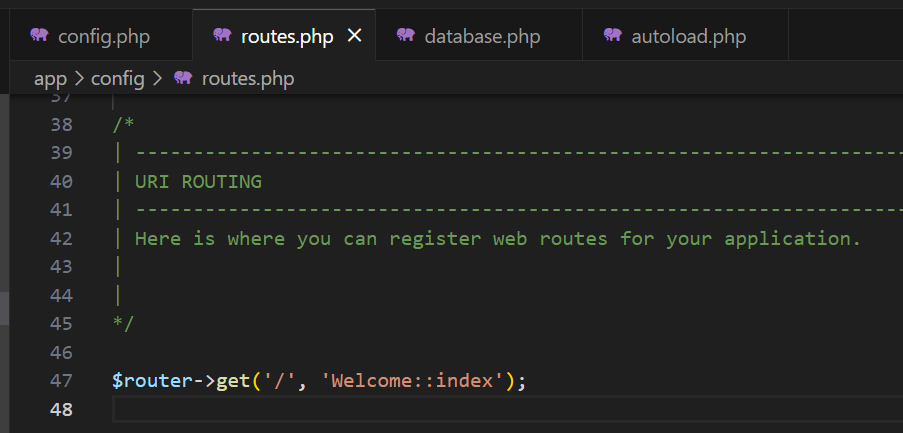
Default Controller & Method

The default controller & method is the page that display/execute first when you access the base URL without specifying the controller & method.

In LavaLust3 the default controller & method is in Config/**Config.php** file, and you can access all your controller in the URL without using Config/**Routes.php**



While in LavaLust 4, you need to create routes for all of your controllers & method before you access it in URL.

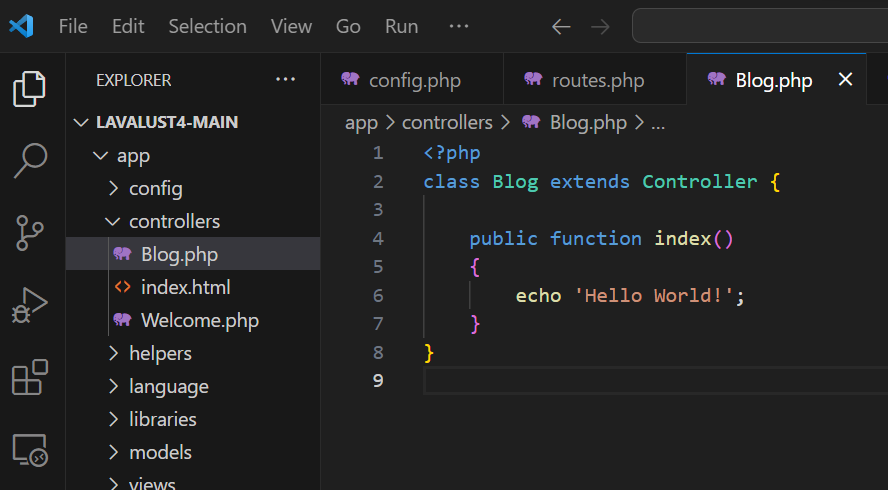


With that, the Welcome controller & index method will execute even not specified in URL (e.g. <http://localhost/LavaLust4-main/>

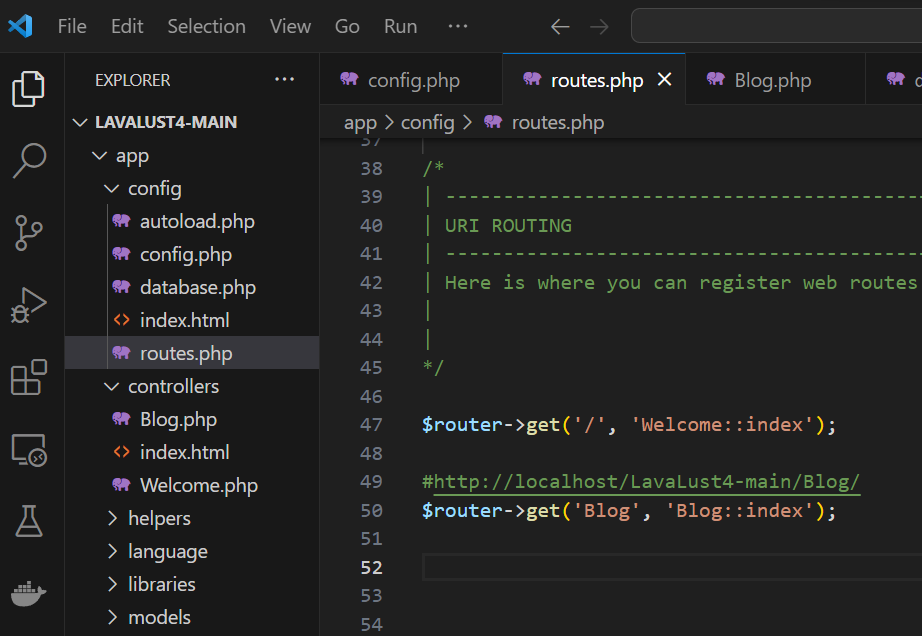
For more info about URI Routing, visit <https://lavalust.netlify.app/#item-4-8>

CONTROLLER

To create a controller, go to **app/controllers** folder and create a **php file**. It is recommended that your controller **file name** must be your controller **class name.** You must also extends/inherit the class **Controller** from the LavaLust scheme/system.

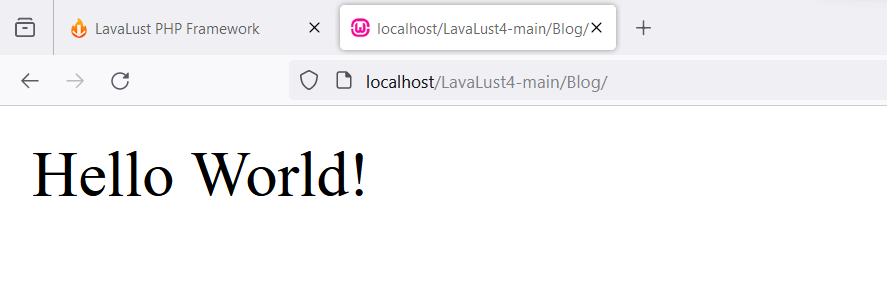


Next is to create a **route** for your **Blog** controller and **index()** method.



The **1st** argument of the **get()** method will be use in your URL (e.g. <http://domain.name/Blog/>). The **2nd** argument is the **Controller** & **method** that will execute when you call the **Blog** (1st argument) in your URL.

**Output:**



ROUTE

Documentation: <https://lavalust.netlify.app/#item-4-8>

**Setting your own routing rules**

Routing rules are defined in your ***app/config/routes.php*** file. In it you’ll see an object **$router** with several **methods** that permits you to specify your own routing criteria.

There are several types of methods use for routing:

1. **get()** – used for Http Request method **Get**
2. **post()** – used for Http Request method **Post**
3. **match()** – used for both **Get** & **Post** or other Http Request method

The **get()** & **post()** method have 2 parameters:

1. The name used in URL
2. The Class/method from your Controller

**Example:**

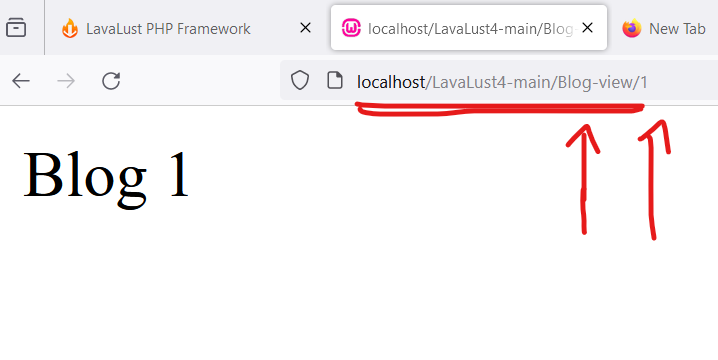
**Route**



**Controller**

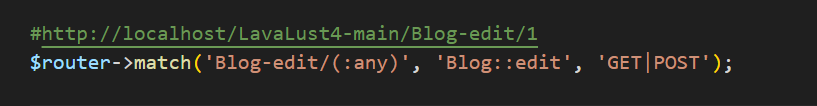


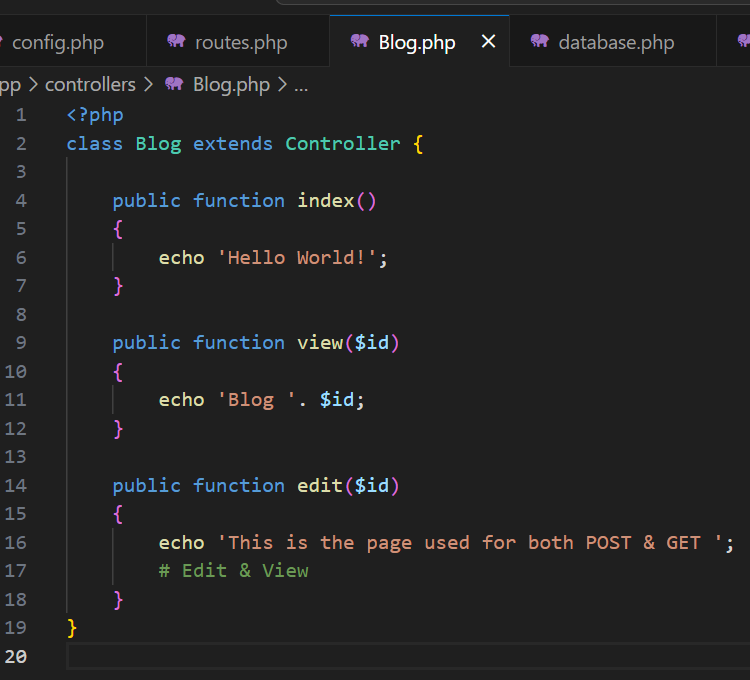
**Output**



The **match()** method have three (3) parameters:

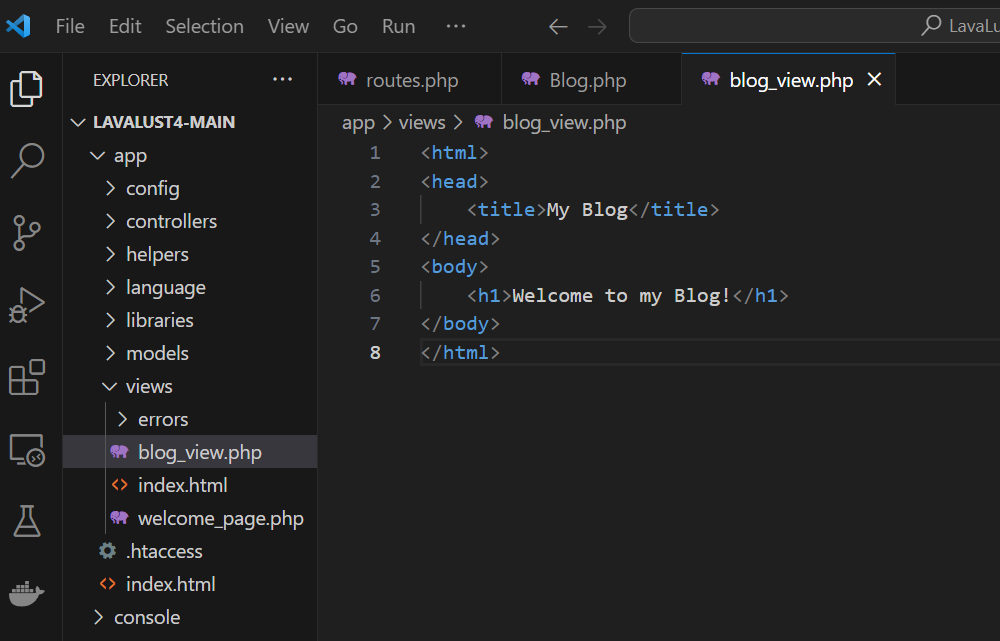
1. The name used in URL
2. The Class/method/ from your Controller
3. And the Http Request (e.g. POST, GET, POST|GET, etc.)



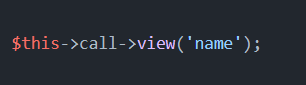
 

Views

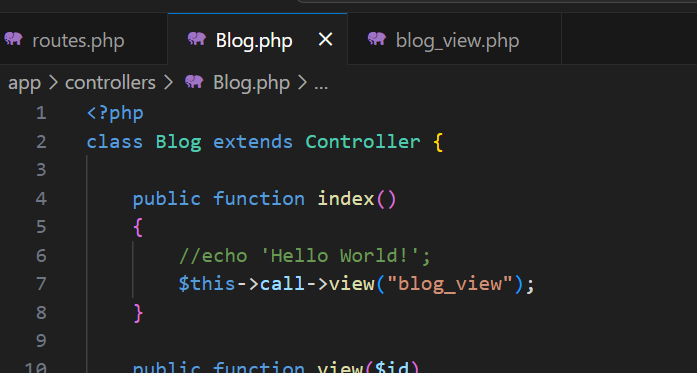
The view is a file or page you see in your web browser. This contains HTML code. Views are never called directly, they must be loaded by a controller. Create a file called **blog\_view.php**, Then save the file in your **app/views/** directory.



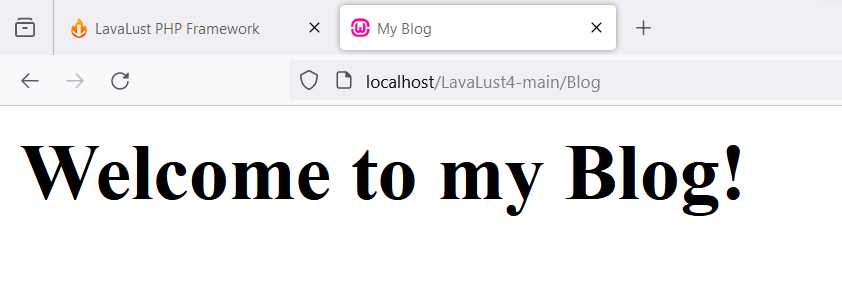
To load a particular view file, you will use the following method inside your controller:



Let’s call it in our Controller.



Output:



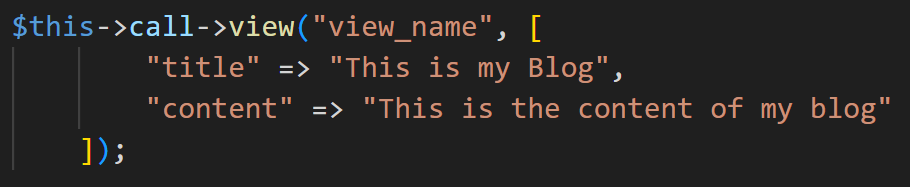
Passing data from Controller-View and View-Controller

Controller to View

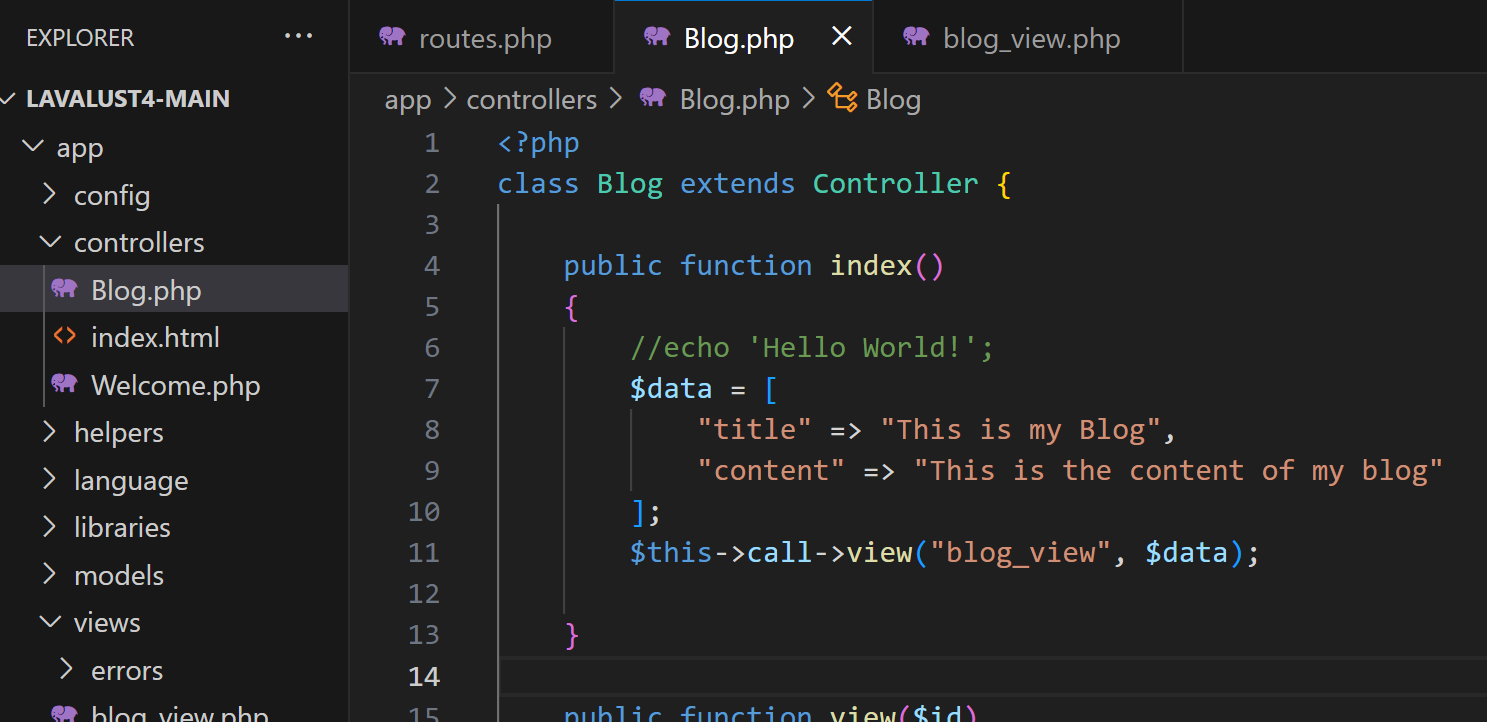
Remember when we render a view file from our Controller, we use **$this->call->view(“view\_file\_name”)** method. The first parameter will be our View File Name.

To pass data from controller to view, we put a second parameter in Associative Array format.

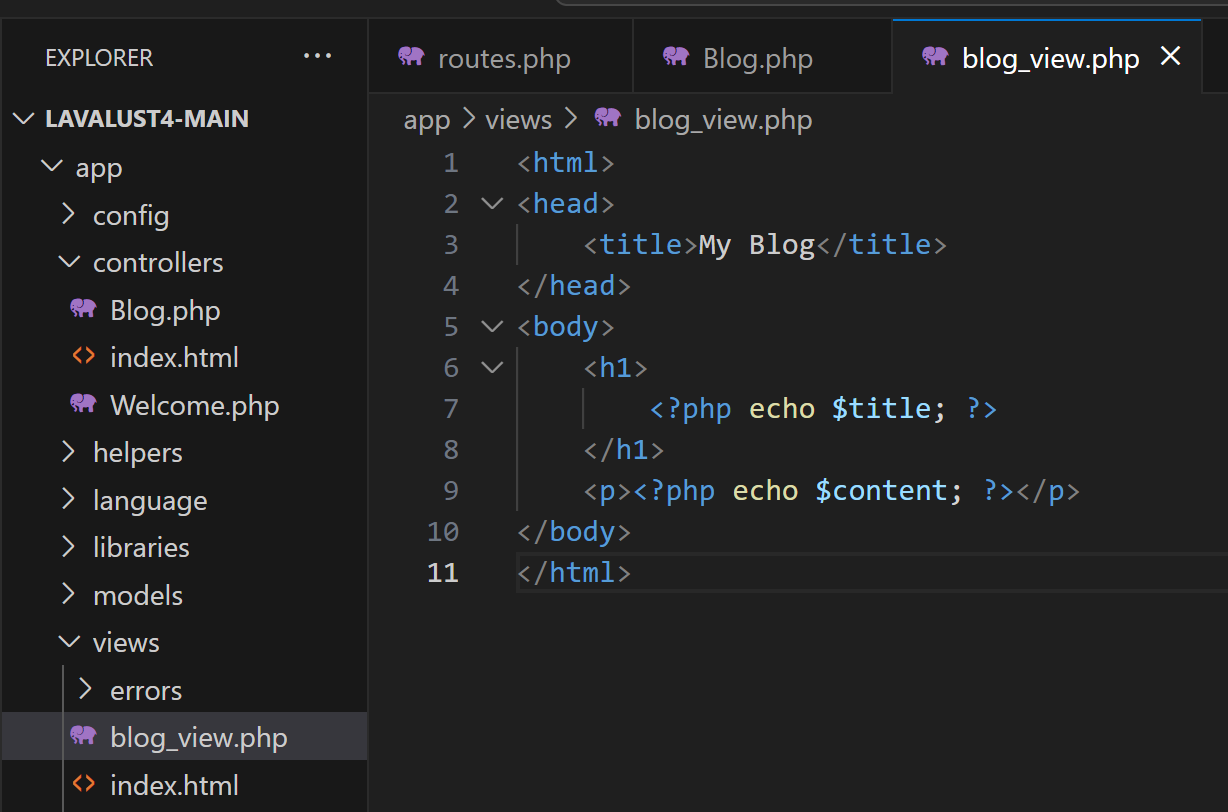
**Example:**



Let’s try it in our **Blog/index** controller



Let’s used the array key name “**title**” & “**content**” to our view file as variable.



**Output:**



View to Controller

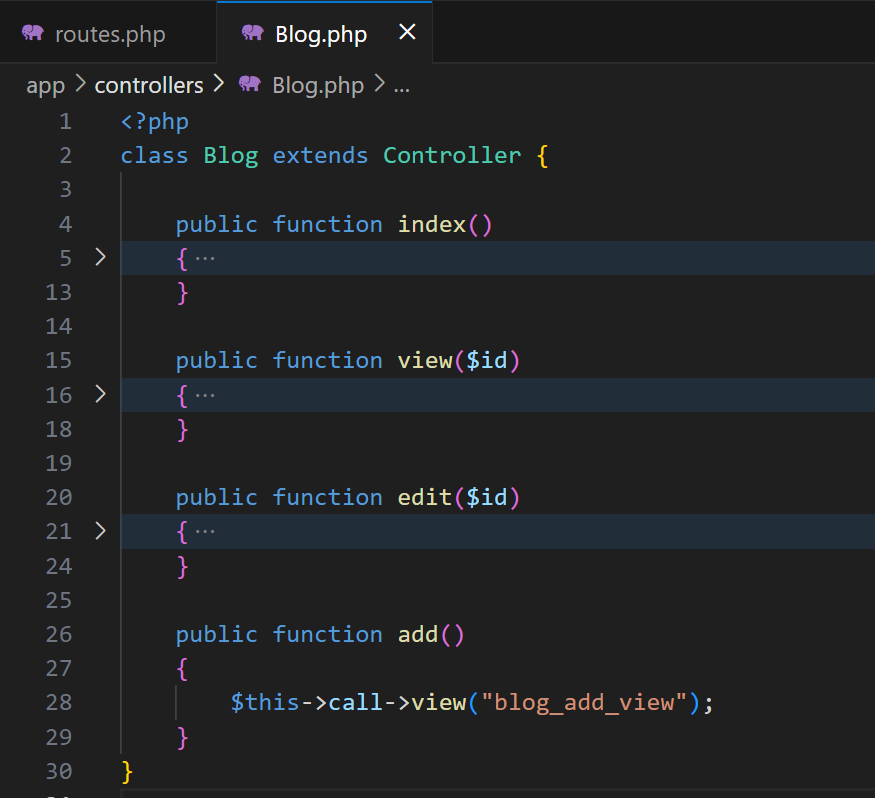
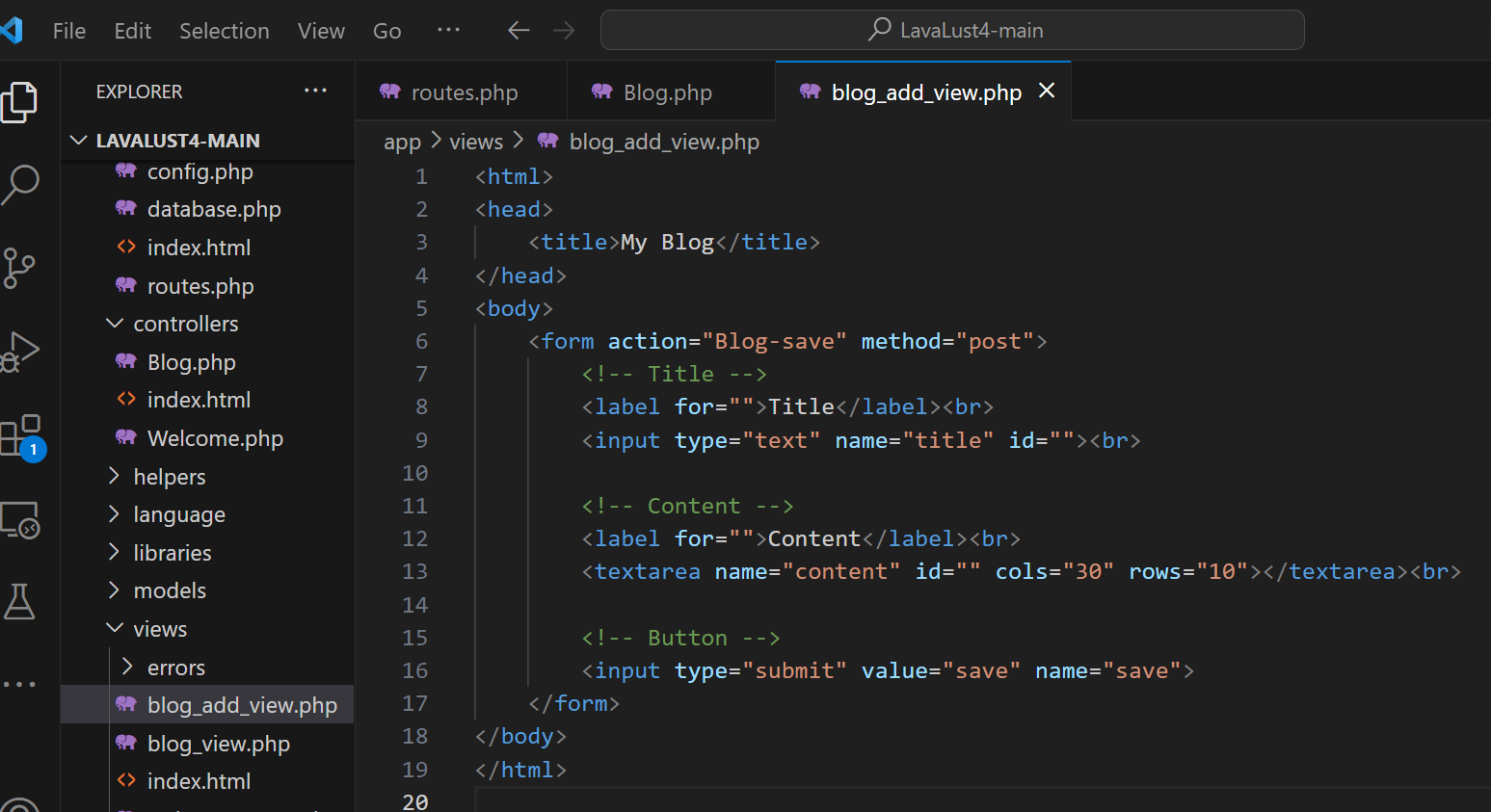
Now, let’s try to pass a data from our **View** file into **Controller**. This time we will used *http request* and *HTML Form*.

**Example:**

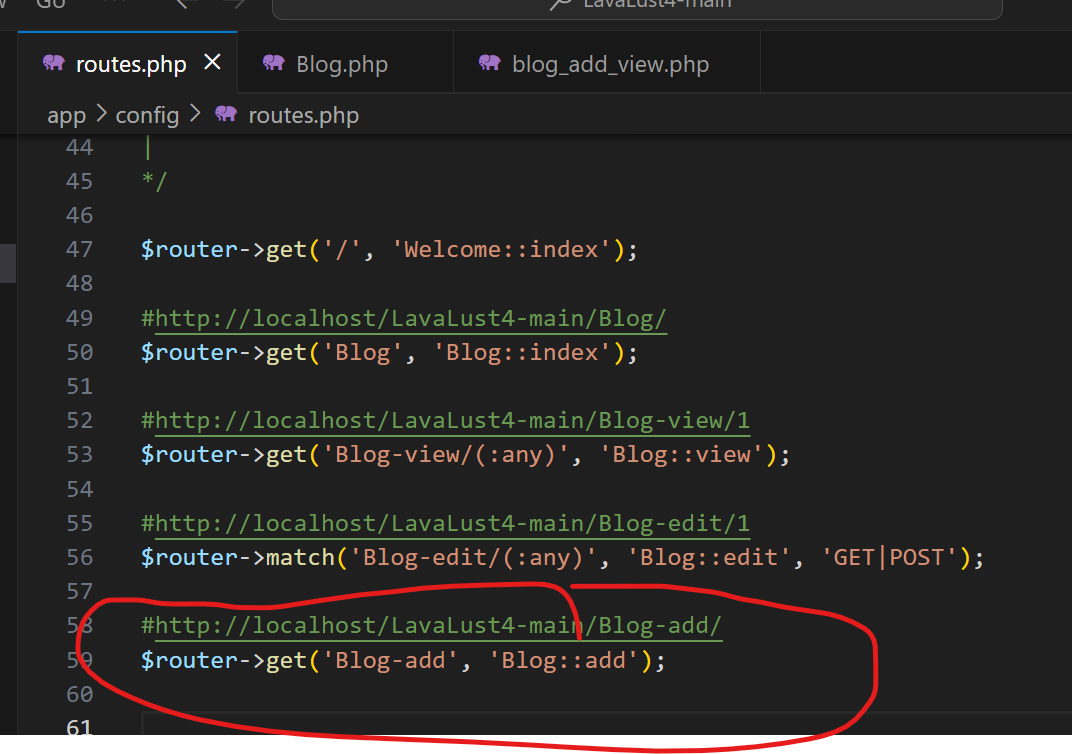
Let’s add first a **add()** method inside the **Blog** controller. Then create a **view** file **blog\_add\_view.php**, add *HTML Form* with *POST* method, *input tag* for Title and Content and a button.

Note: The Action’s value of the Form tag will be the route of the page after we submit the **Form** from our *Blog/add*. When we submit the form, the browser will redirect to the **action=”Blog-save”** or <http://localhost/LavaLust4-main/Blog-save>

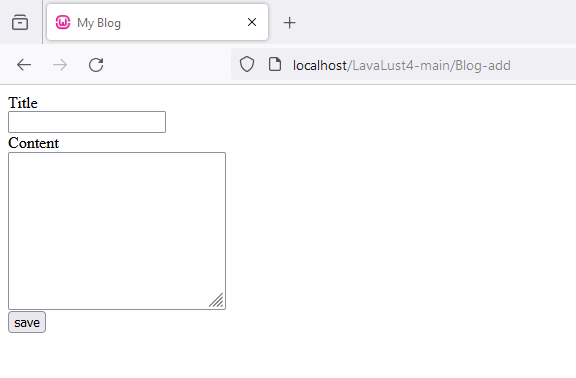
Check the image below:

**Create our route!**

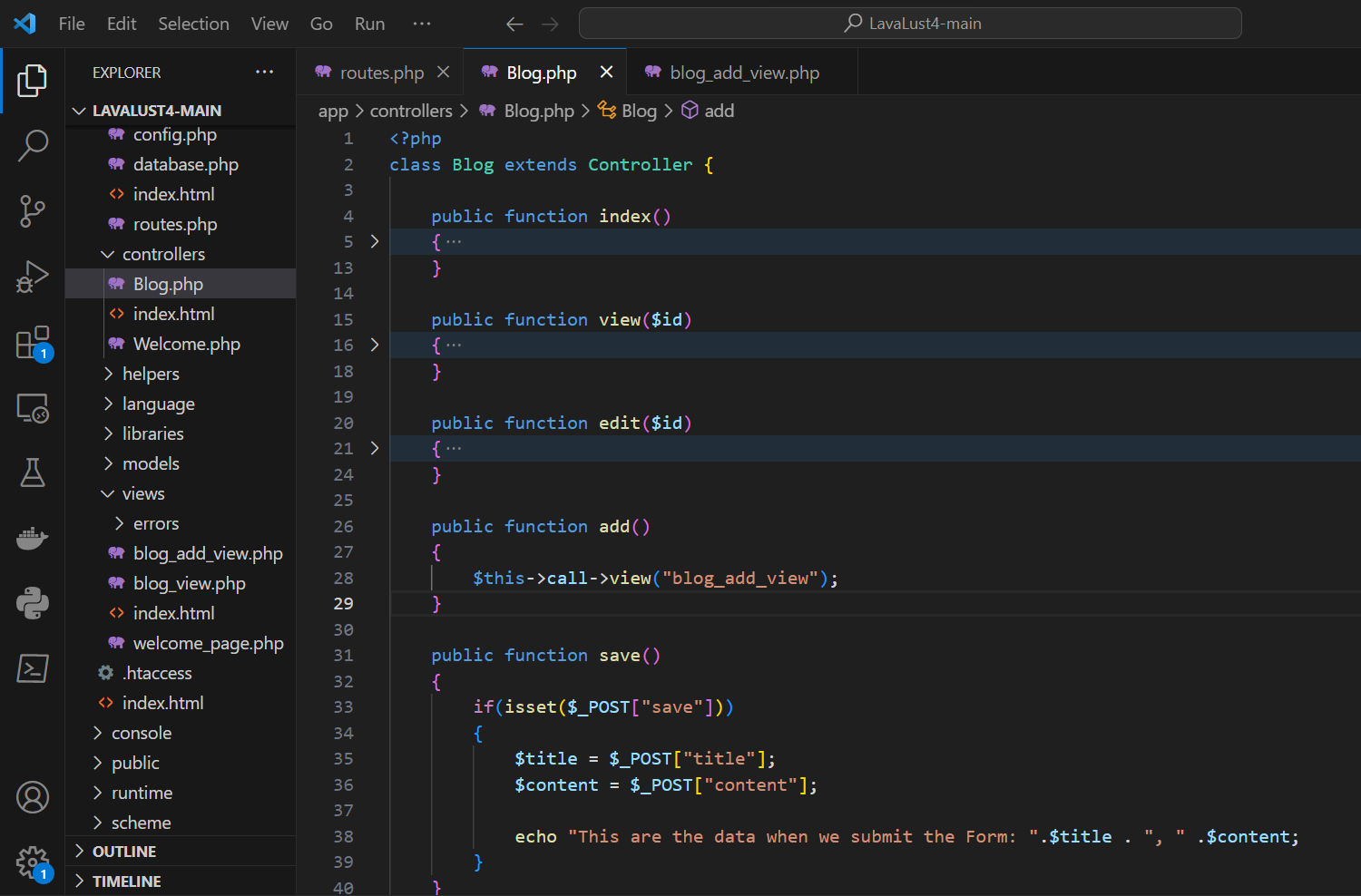


**Output:**



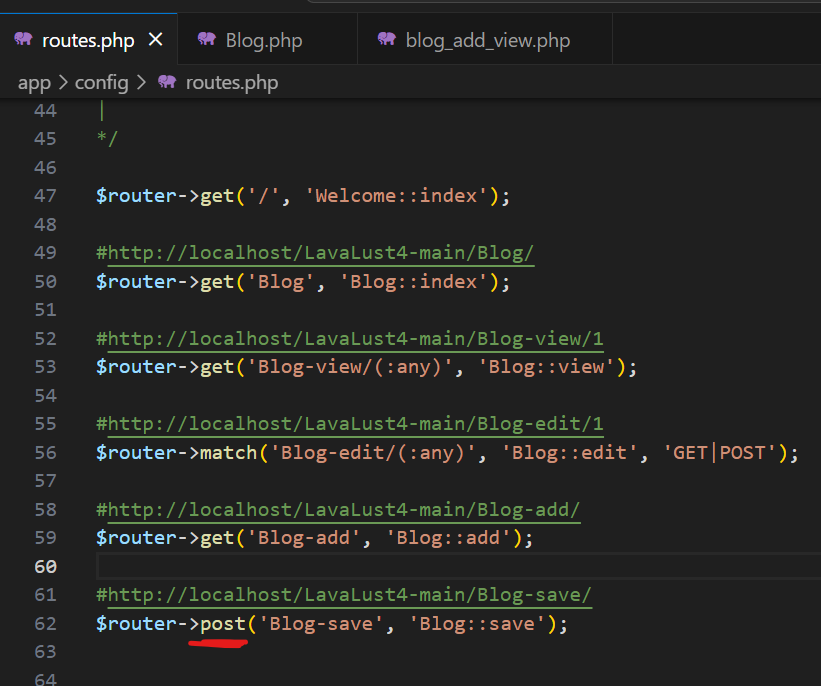
Note: Again, when we submit the form, the browser will redirect to the <http://localhost/LavaLust4-main/Blog-save>

Now let’s create the **Blog-save** page after we submit the Form, then get the data from our HTML inputs.

**Controller:** 

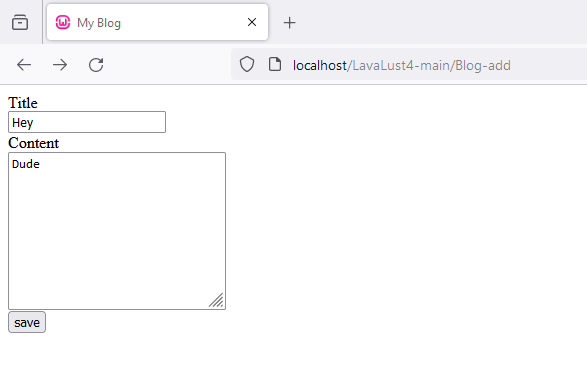
**Route:**

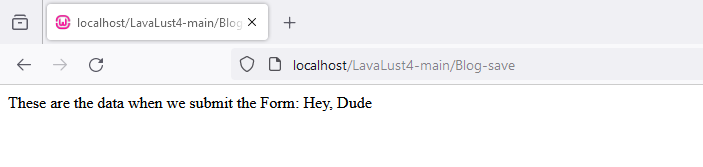
This time, we will used **POST** method for our route.



**Output:**

Let’s submit the form

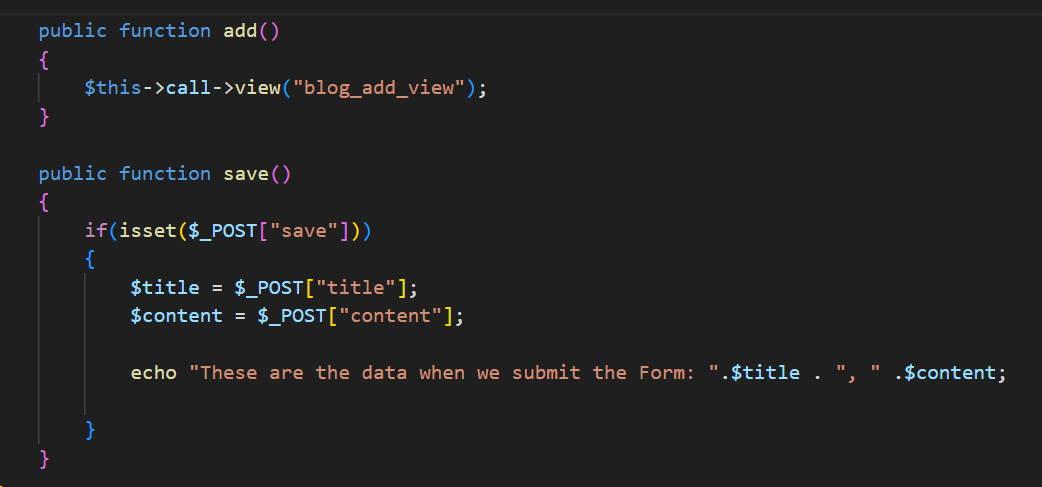




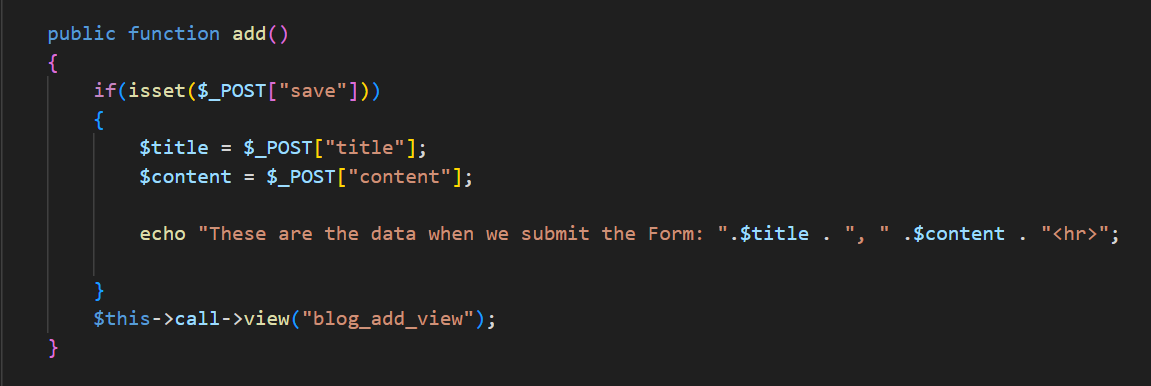
OPTIMIZING OUR CODE

Let’s merge our **Blog-add** page and **Blog-save** page for better organization of our code.

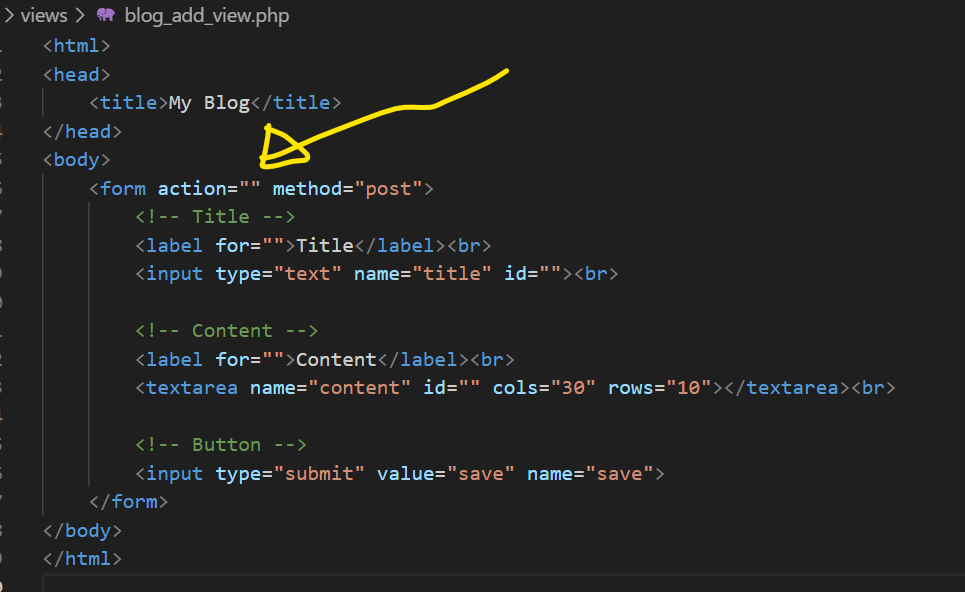
From this:



To this:



Now, empty the action attribute value, so it will redirect to the same page.

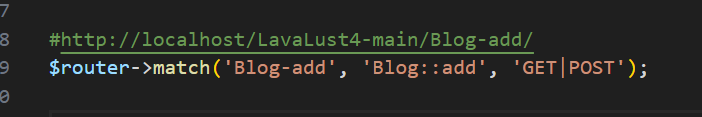


**Let’s modify the Routes**

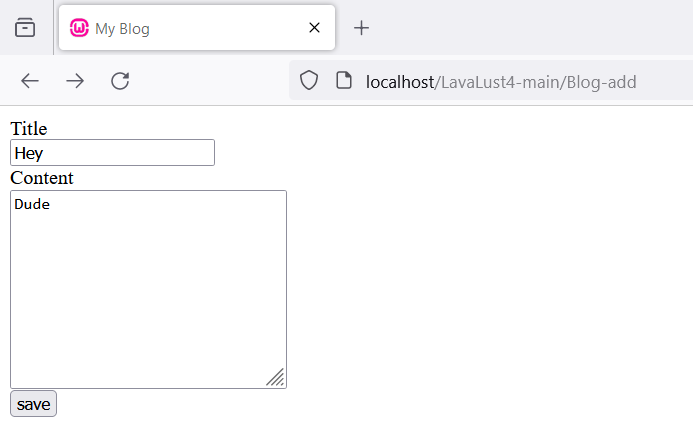
From this:



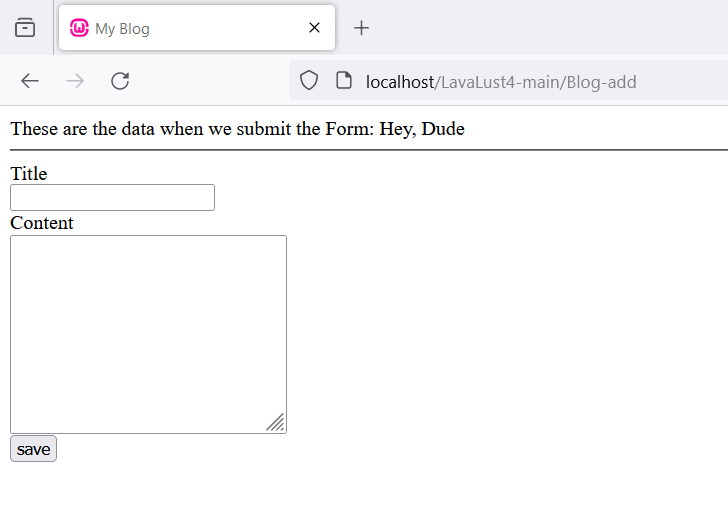
To this: (this time, we will used match() method both for GET & POST)



**Now let’s submit this form:**

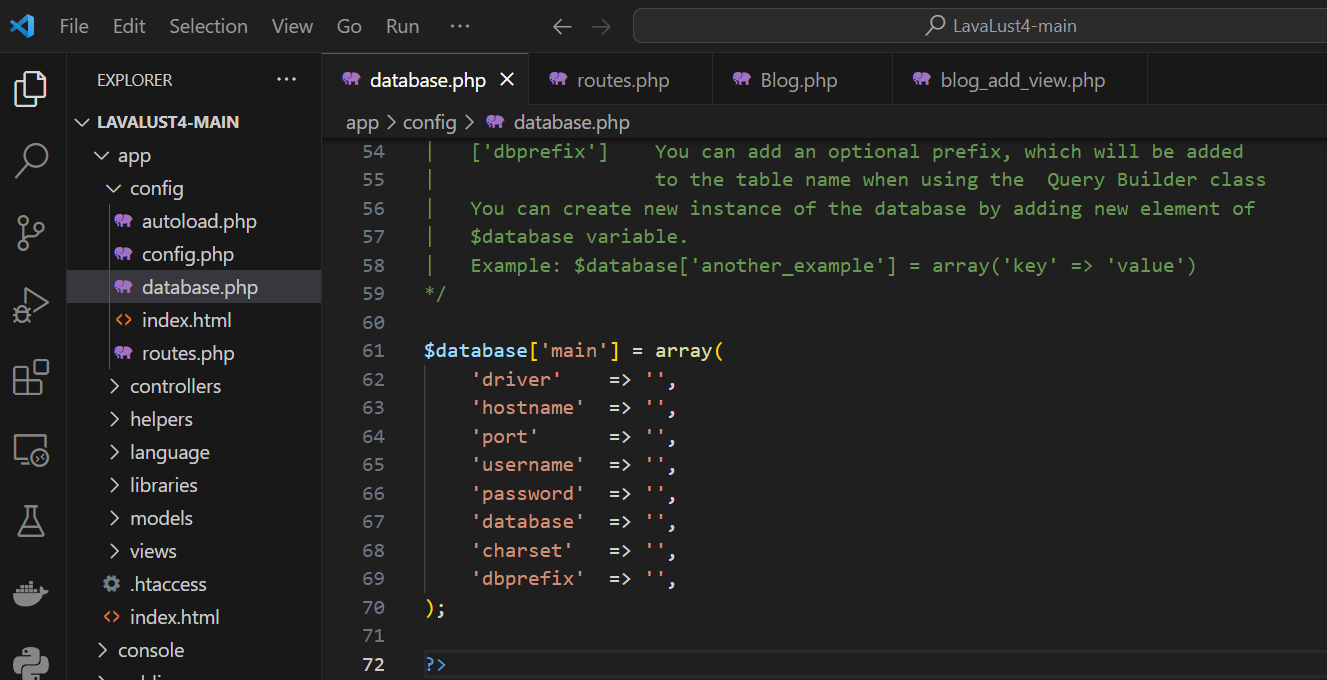


**And this will be the output:**

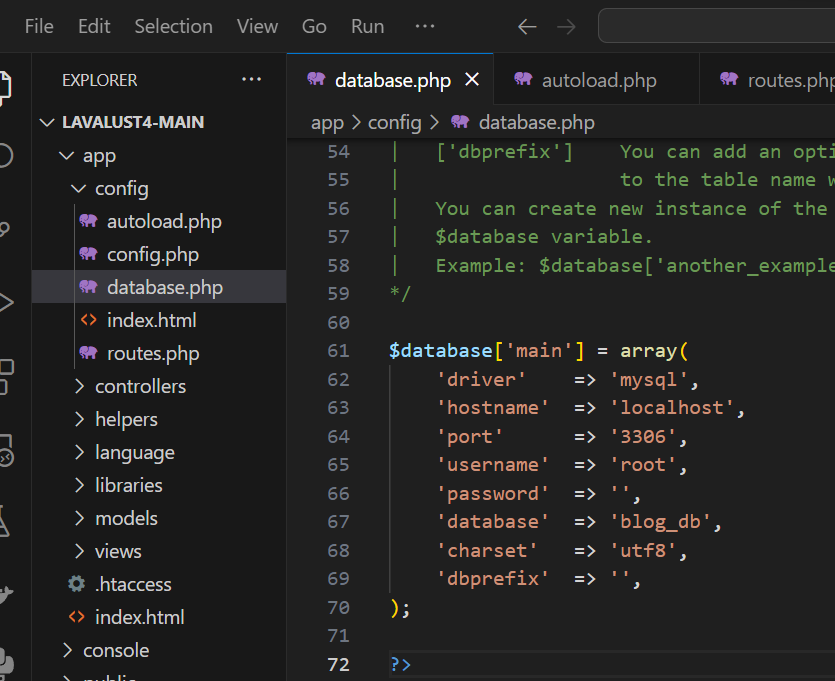


DATABASE CONFIGURATION

The database settings of LavaLust will be in ***database.php*** inside config folder.



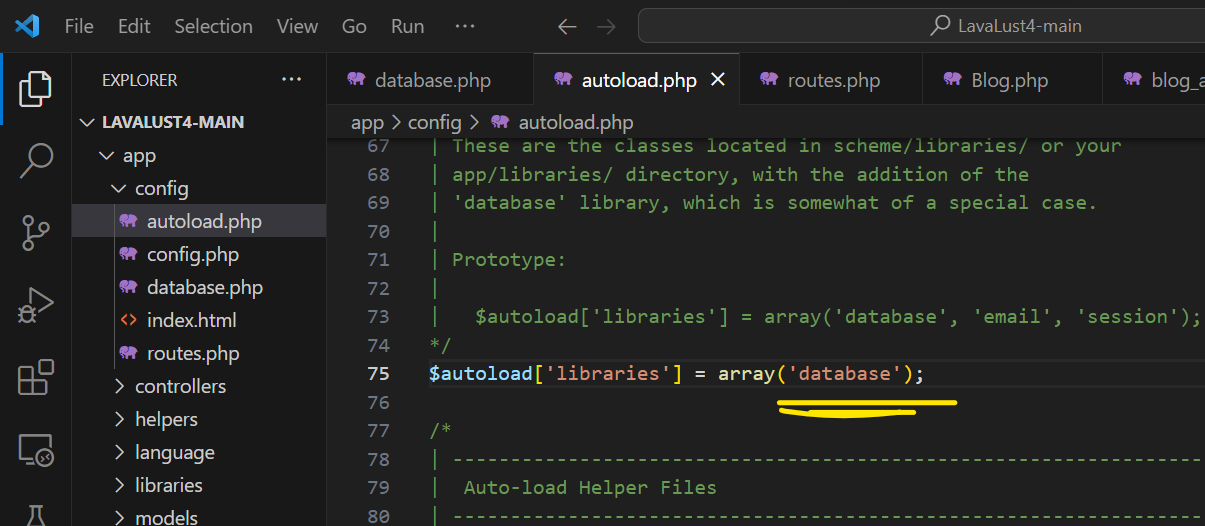
Let’s add our database configurations



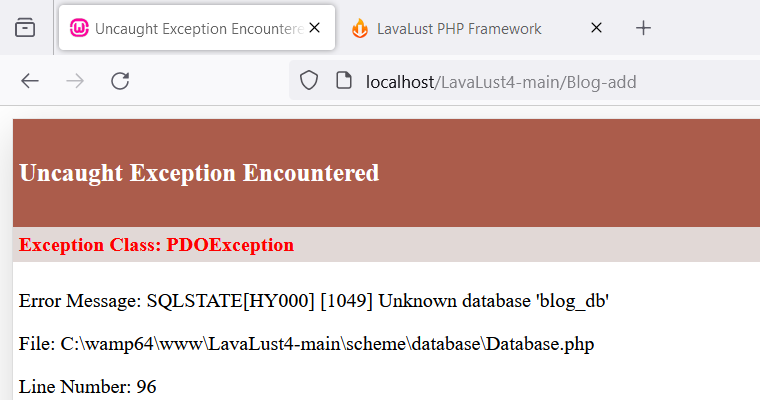
Database Library

In an MVC Framework, we will use library for using database. To use that library, we need to load that library to our ***autolad.php*** file inside the ***config*** folder. The autoload.php file is used as configuration to load all the needed resources or initialized automatically every time the system runs. See <https://lavalust.netlify.app/#item-4-7> about **Autoload.**

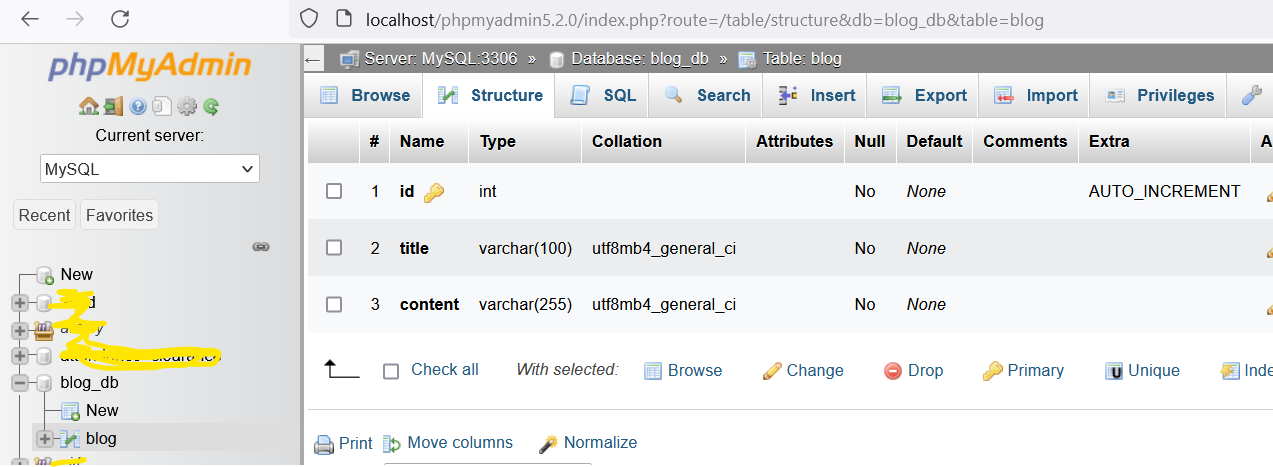
**Let’s autoload the library called “database”.**



**Now, reload the browser and it will get an error unknown database.**

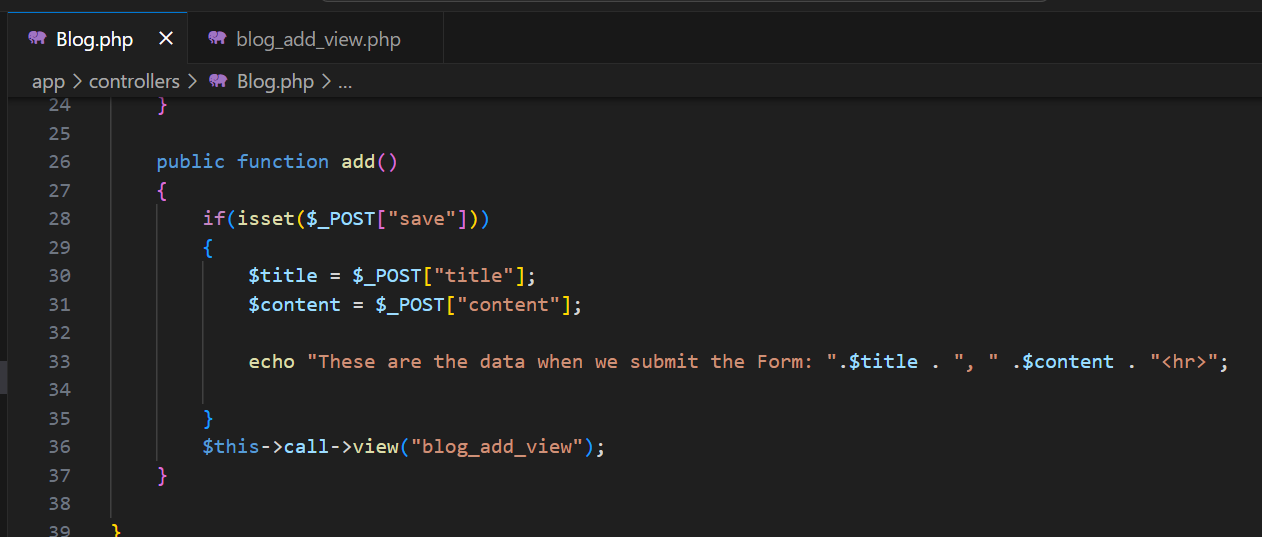


**Create the database.**

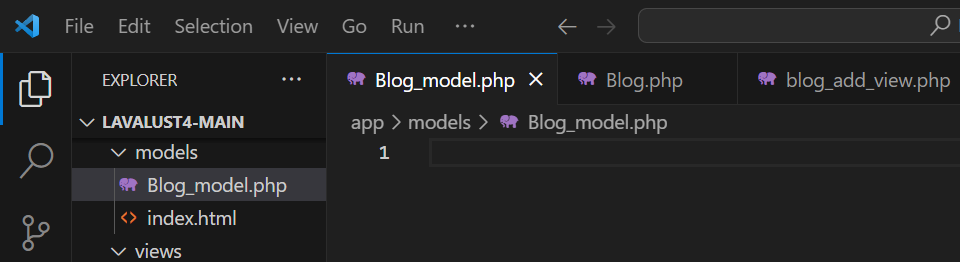


MODELS & QUERY BUILDER

Remember the page below (Blog/add)? Let’s save the **title** and **content** to our database instead.

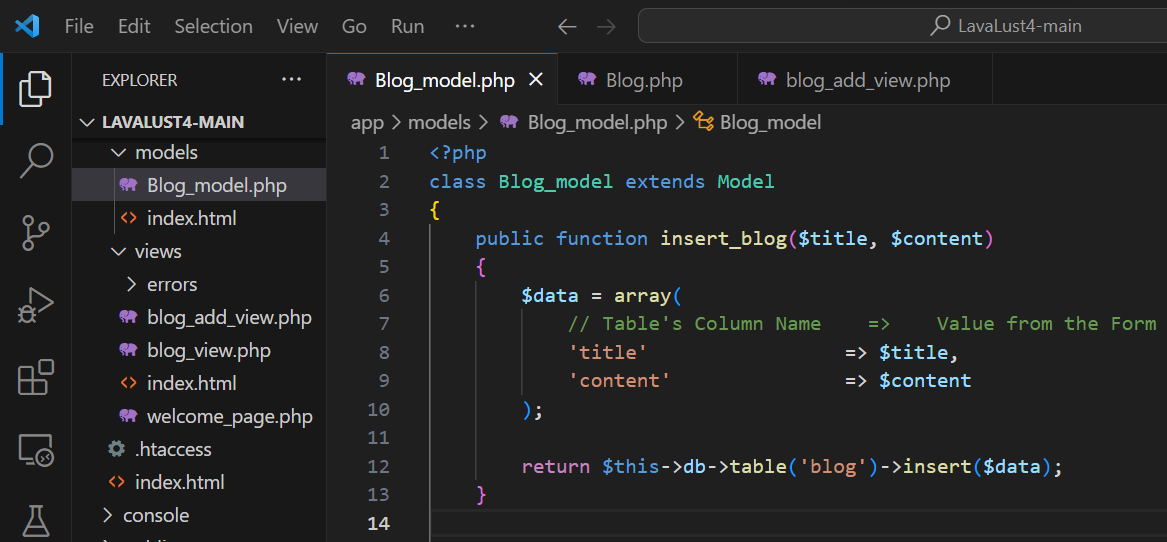


First is create a **Model** called ***Blog\_model.php***



**Models** are PHP classes that are designed to work with information in your database. See <https://lavalust.netlify.app/#item-4-7> for more info about Model.

Now, create a method for inserting the blog info to our database. The method I created has two parameters (*depends on number of information needs to save in database*).



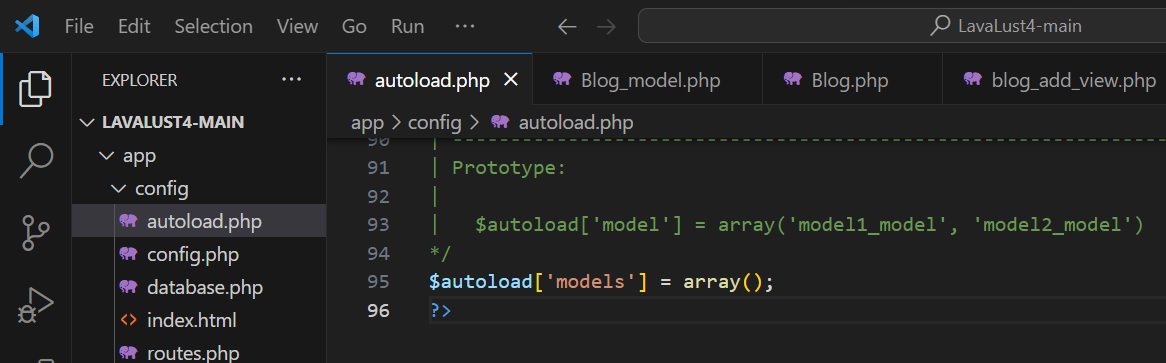
***NOTE:*** *Visit* [*https://lavalust.netlify.app/#item-6-3*](https://lavalust.netlify.app/#item-6-3) *about query builder.*

Loading the Model

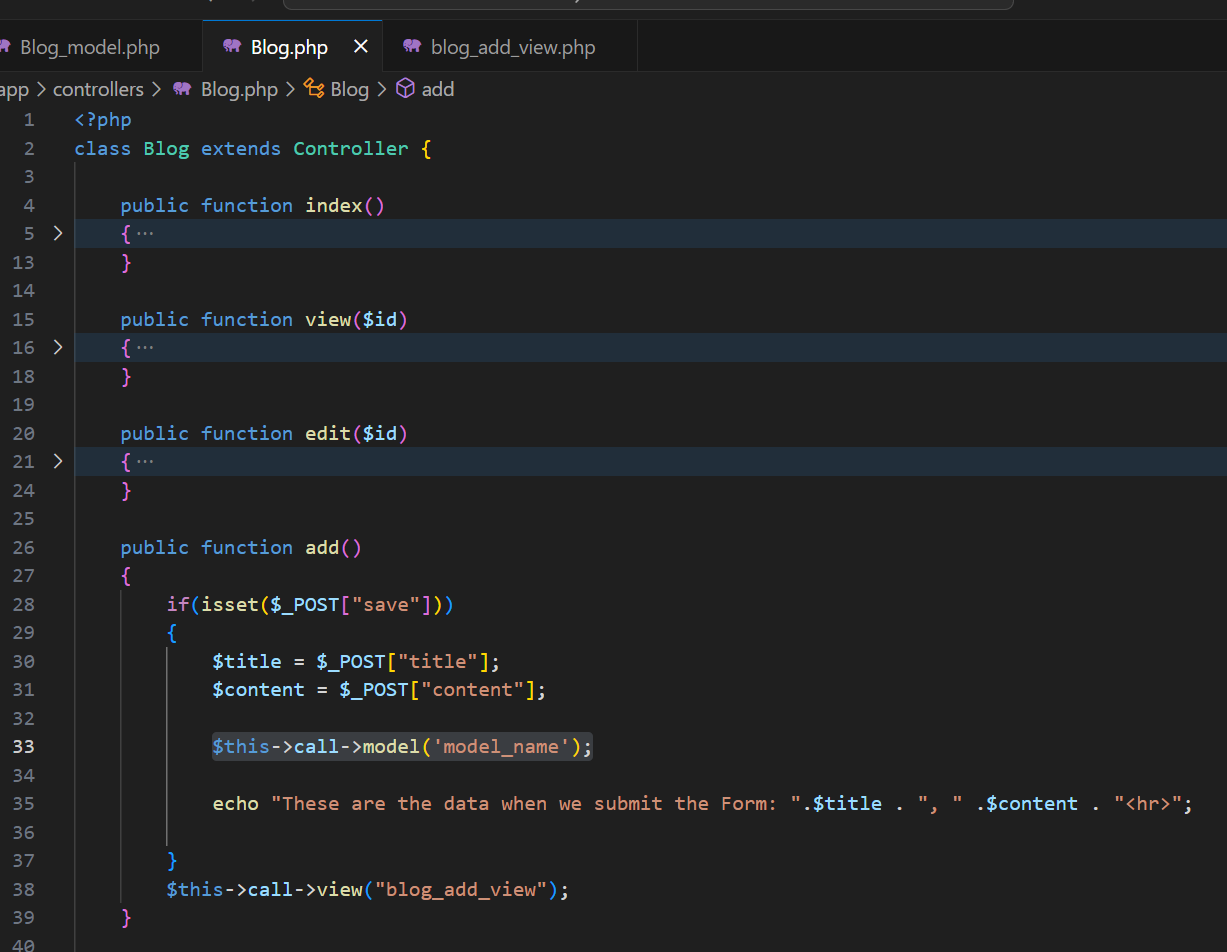
Now, let’s use our **Model** to our **Controller.**

**There are two ways of calling/loading/using the Model:**

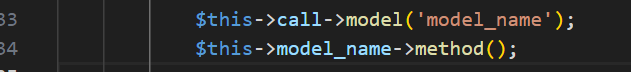
1. Using ***Autoload.php***,



1. **Manual loading**. *Your models will typically be loaded and called from within your Controller methods.*

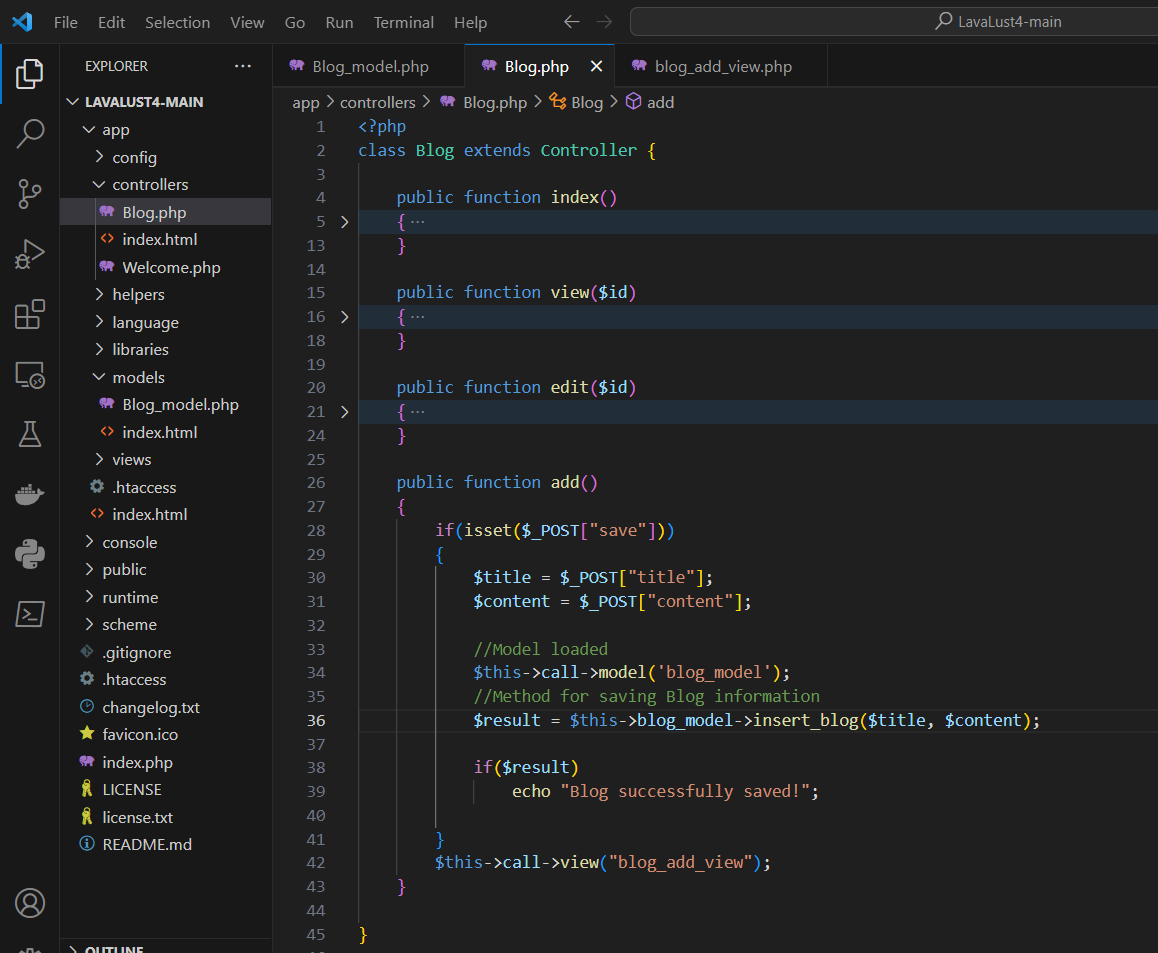


Once loaded, you will access your model methods using an object with the same name as your class:

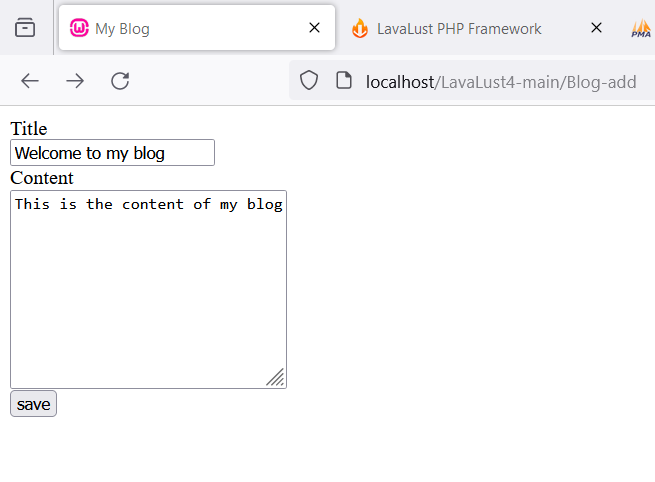


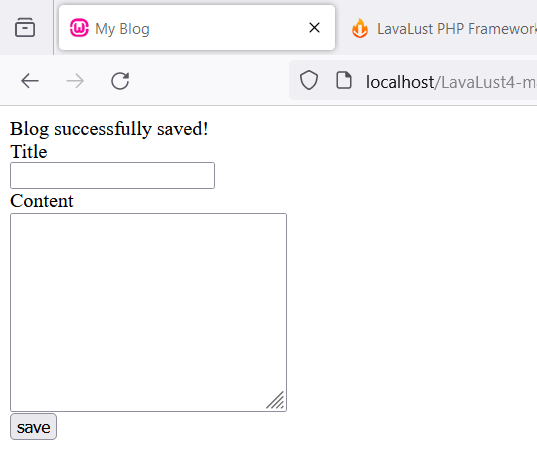
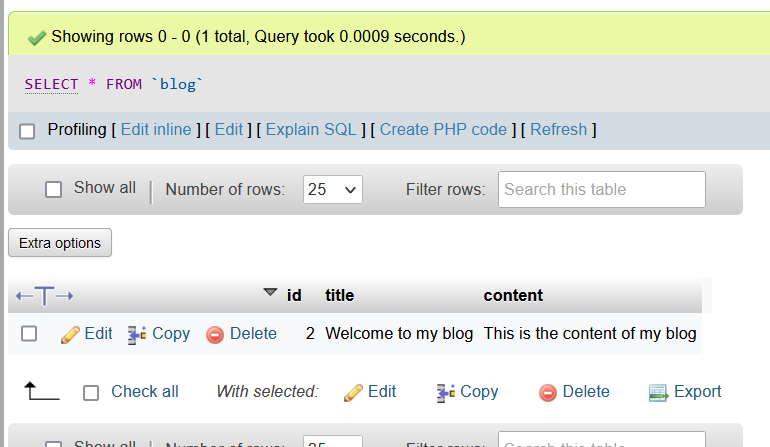
INSERT INTO (CRUD)

Let’s back to our Controller **Blog.php**, and load our model then used the method for inserting blog information.



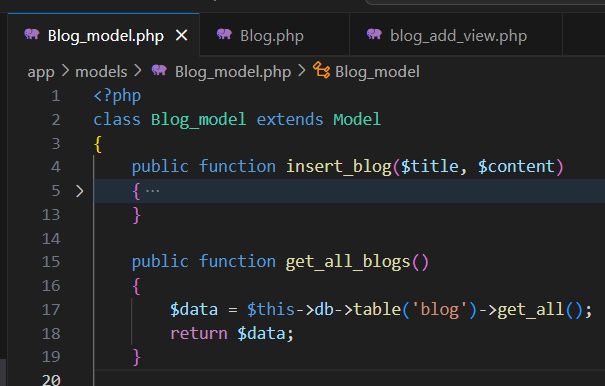
Let’s try and submit some information:



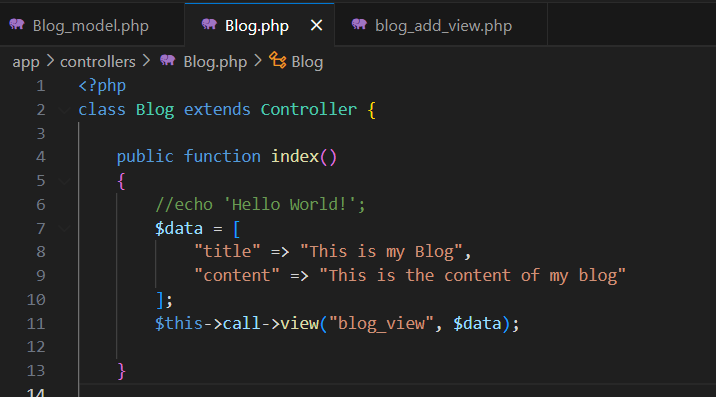
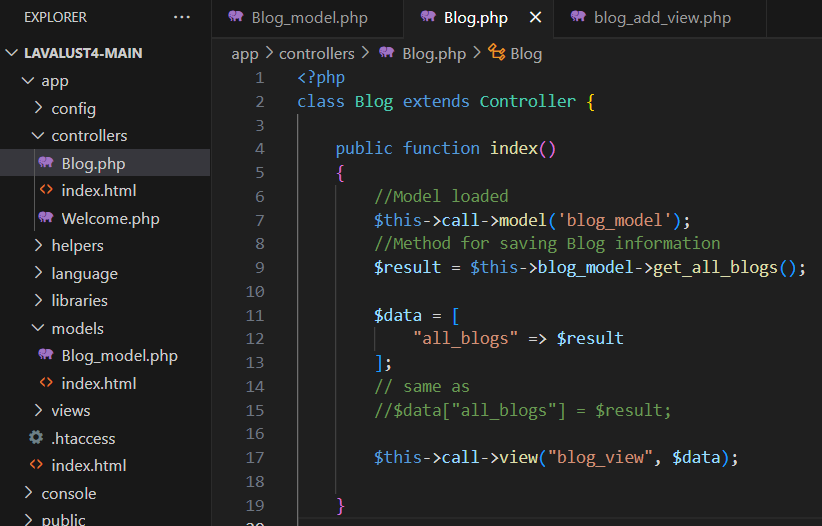
Output**:** ****

SELECT STATEMENT (CRUD)

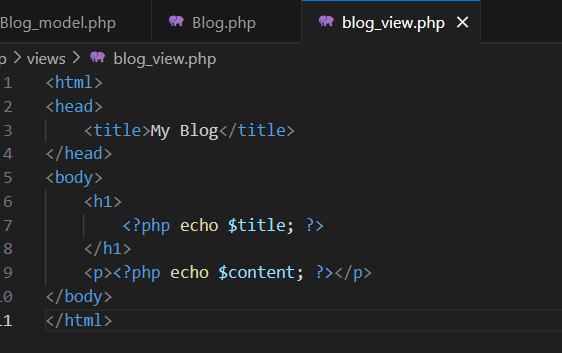
**Model** (select \* from blog)

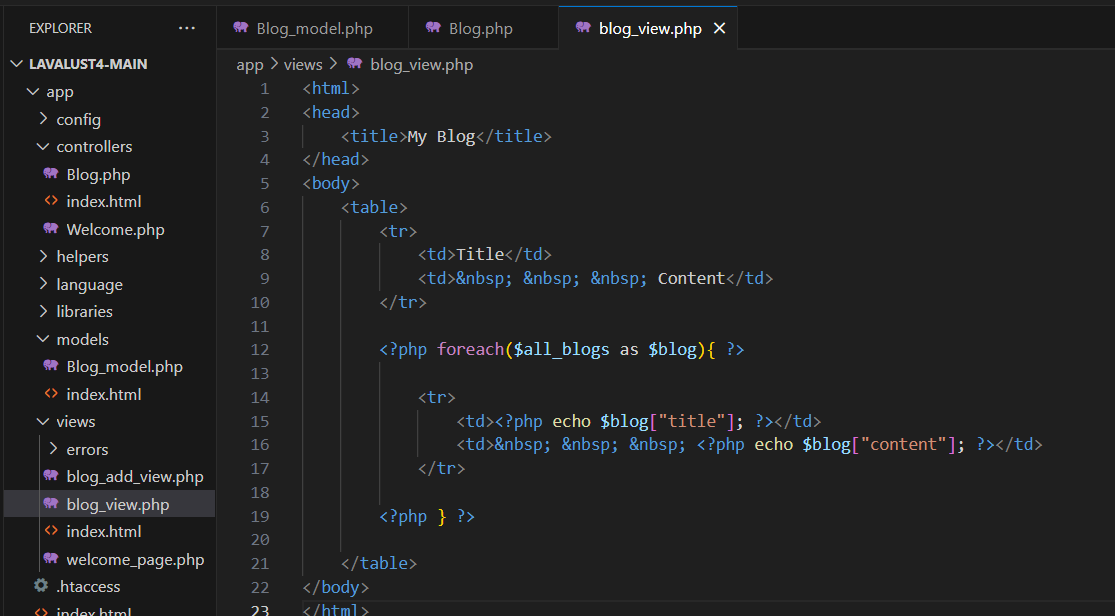
****

**Controller** (Blog/index)

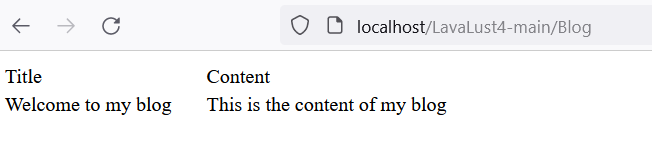
From this: **** To this: 

**Views** (Blog\_view.php)

From this: 

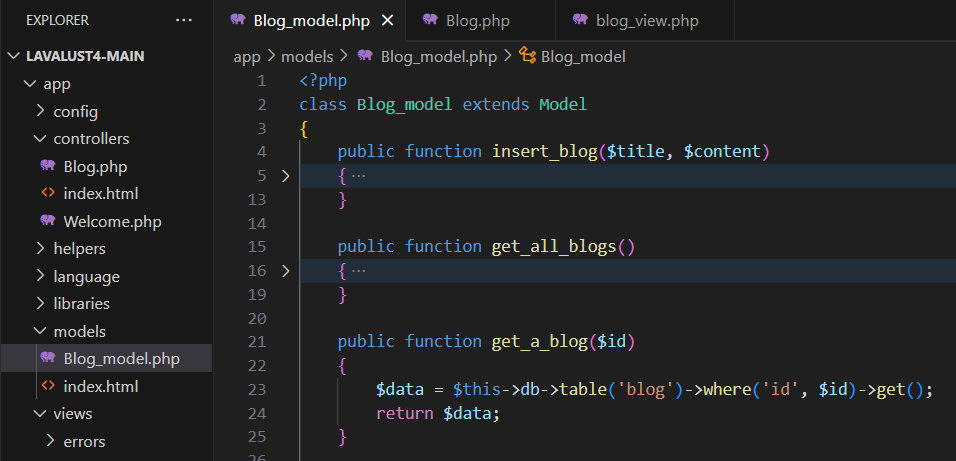
To this: 

**Output:**

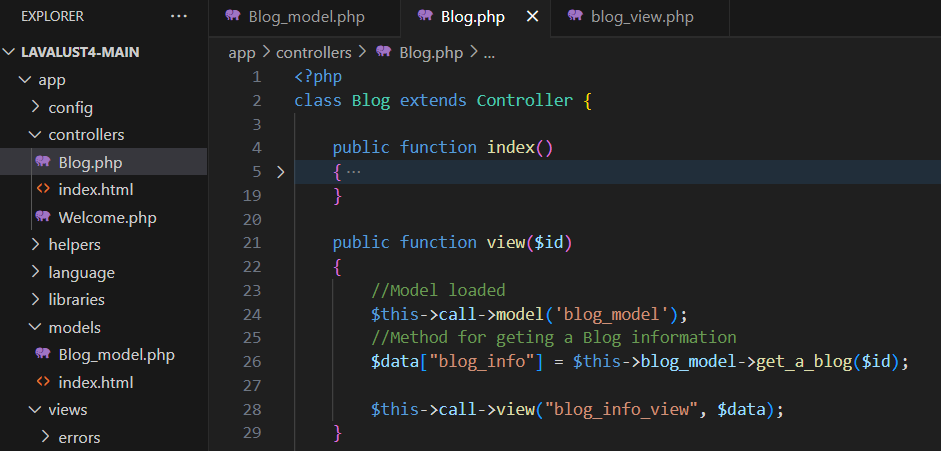


SELECT only one (1) row (CRUD)

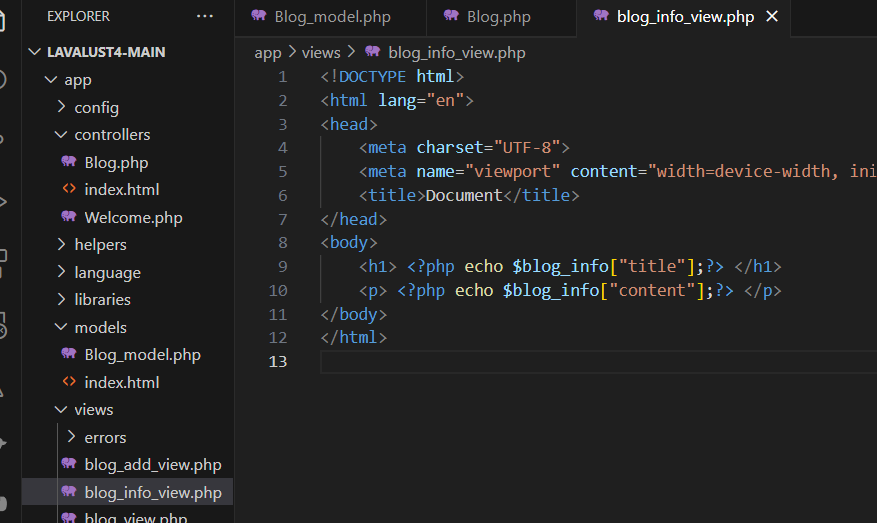
**Model** (select \* from blog where id=?)



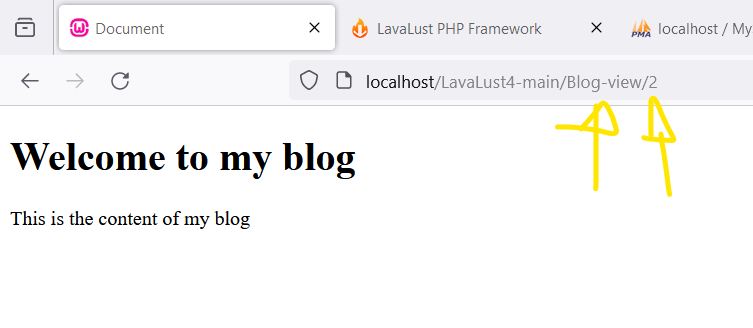
**Controller** (Blog/view/$id)



**View** (blog\_info\_view.php)

****

**Output:**

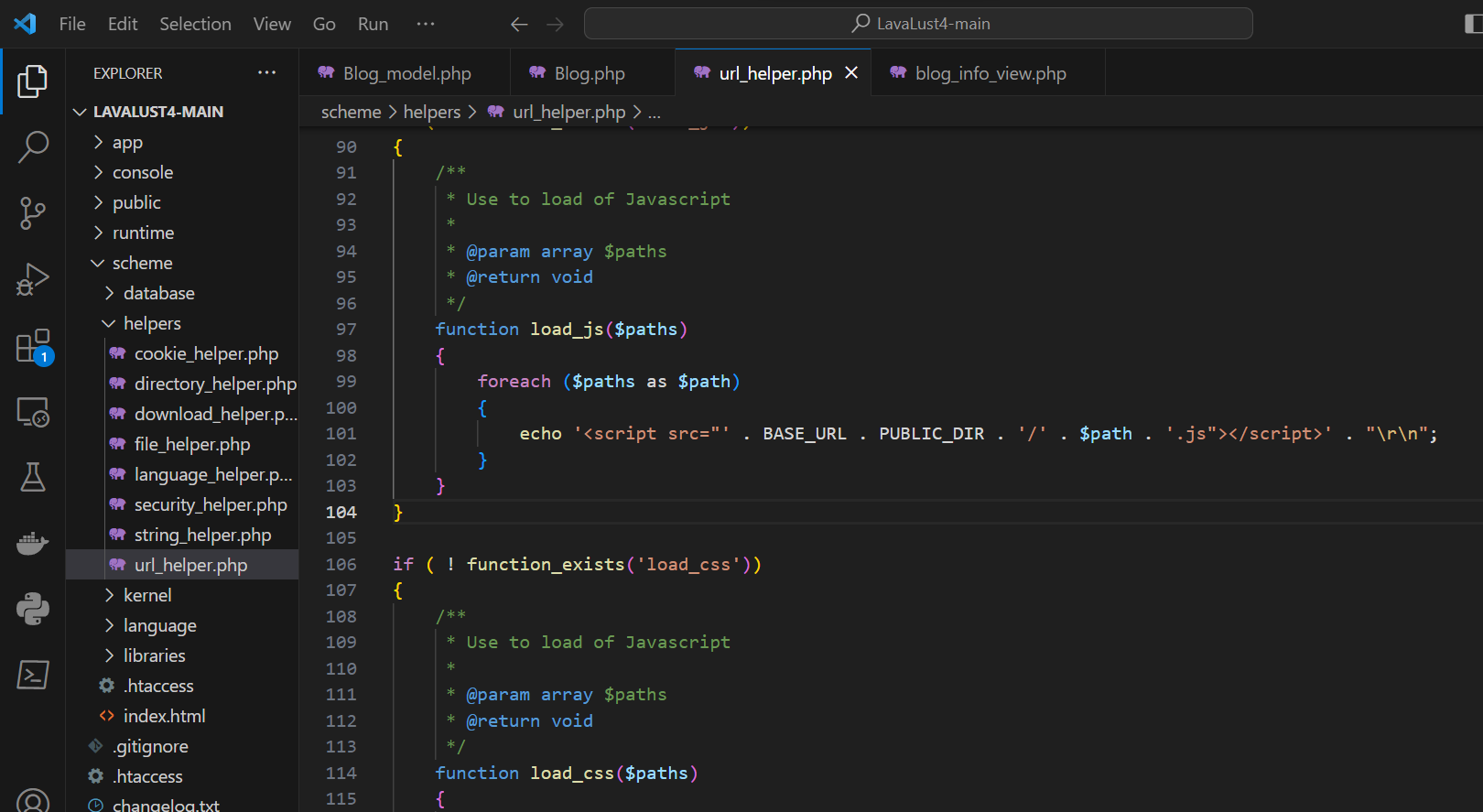
****

***YOUR TURN:*** *Apply the idea of the* ***Blog-view*** *&* ***Blog-add*** *page to create Edit/Update (CRUD).*

Helpers

**Built-in Helpers.** LavaLust have built-in helpers help you with tasks. Each helper file is simply a **collection of functions** in a particular category.

Helpers are typically stored in your **scheme/helpers**, or **app/helpers** directory.

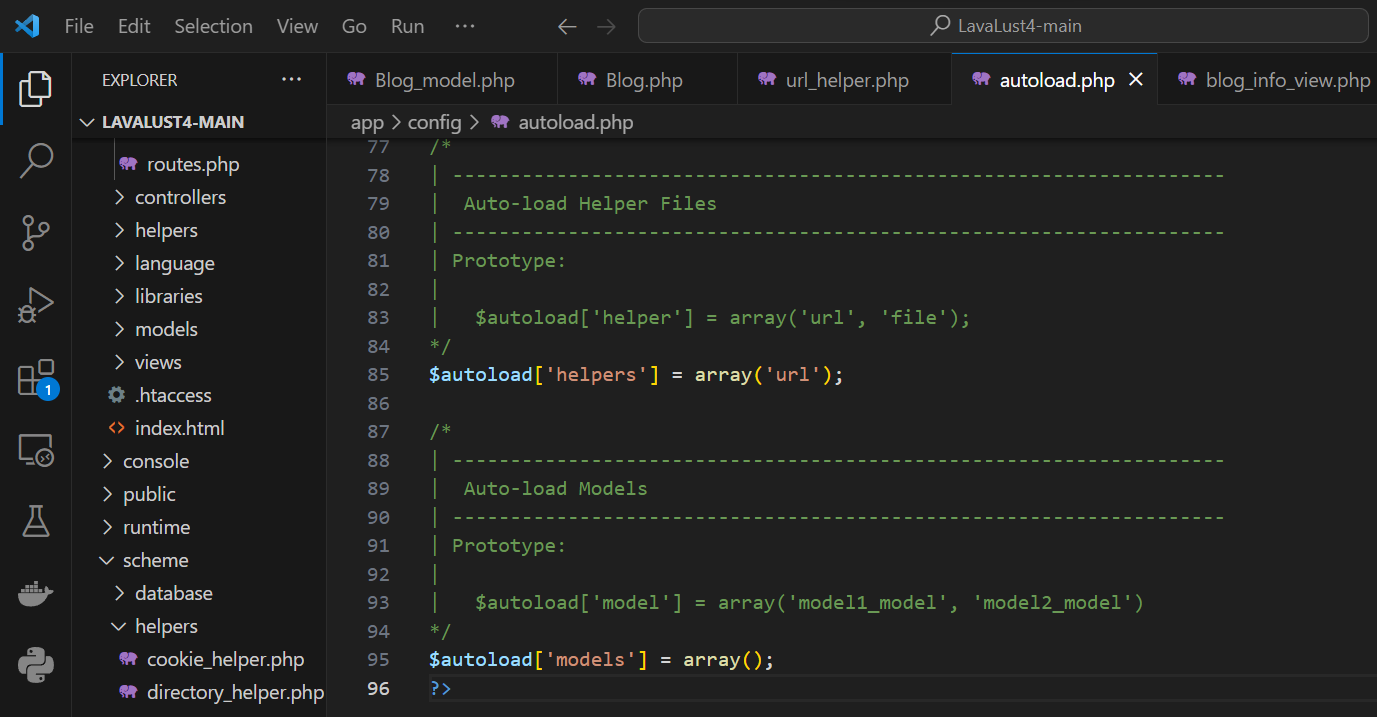


LavaLust does not load Helper Files by default, so the first step in using a Helper is to **load** it. Once loaded, it becomes **globally available** in your **controller** and **views**.

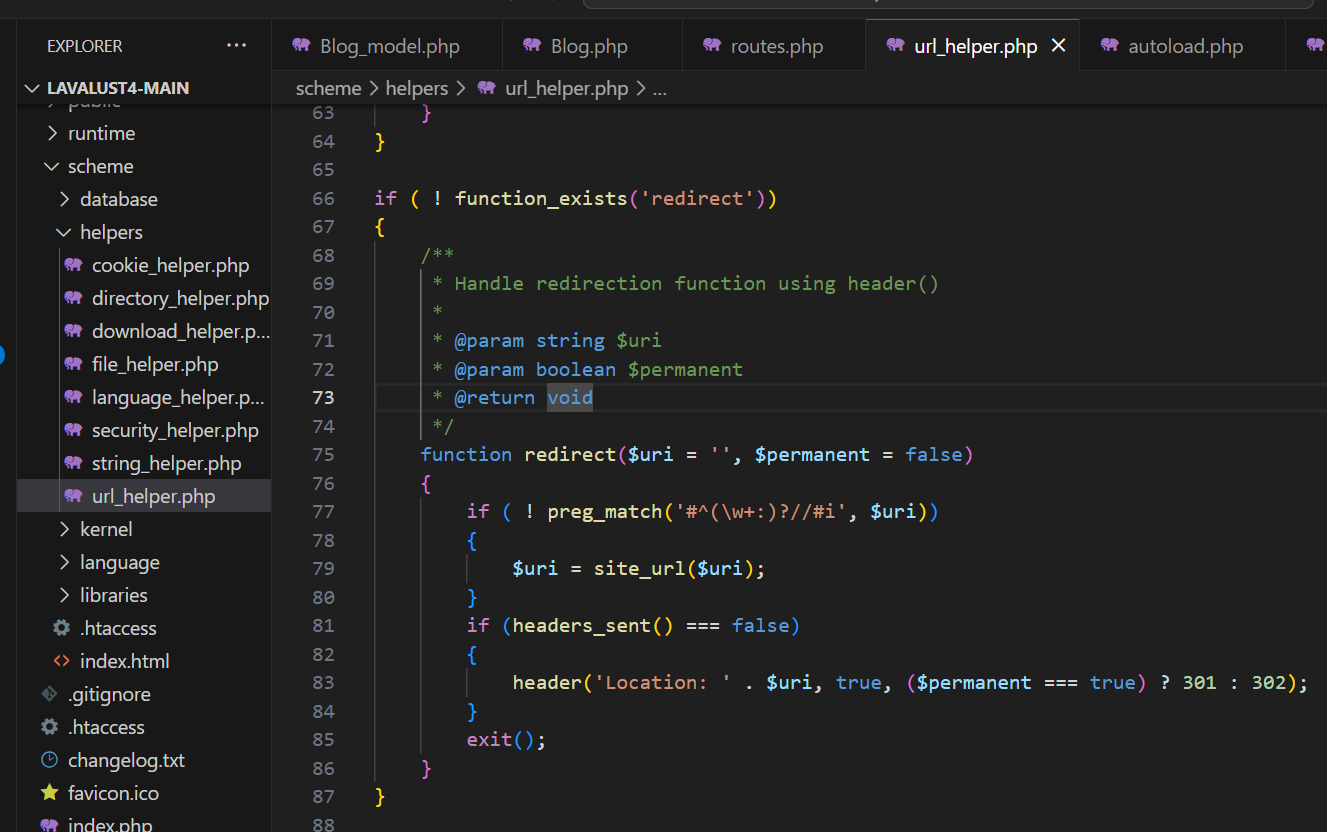
Loading Helper

Load the helper in **autoload.php**

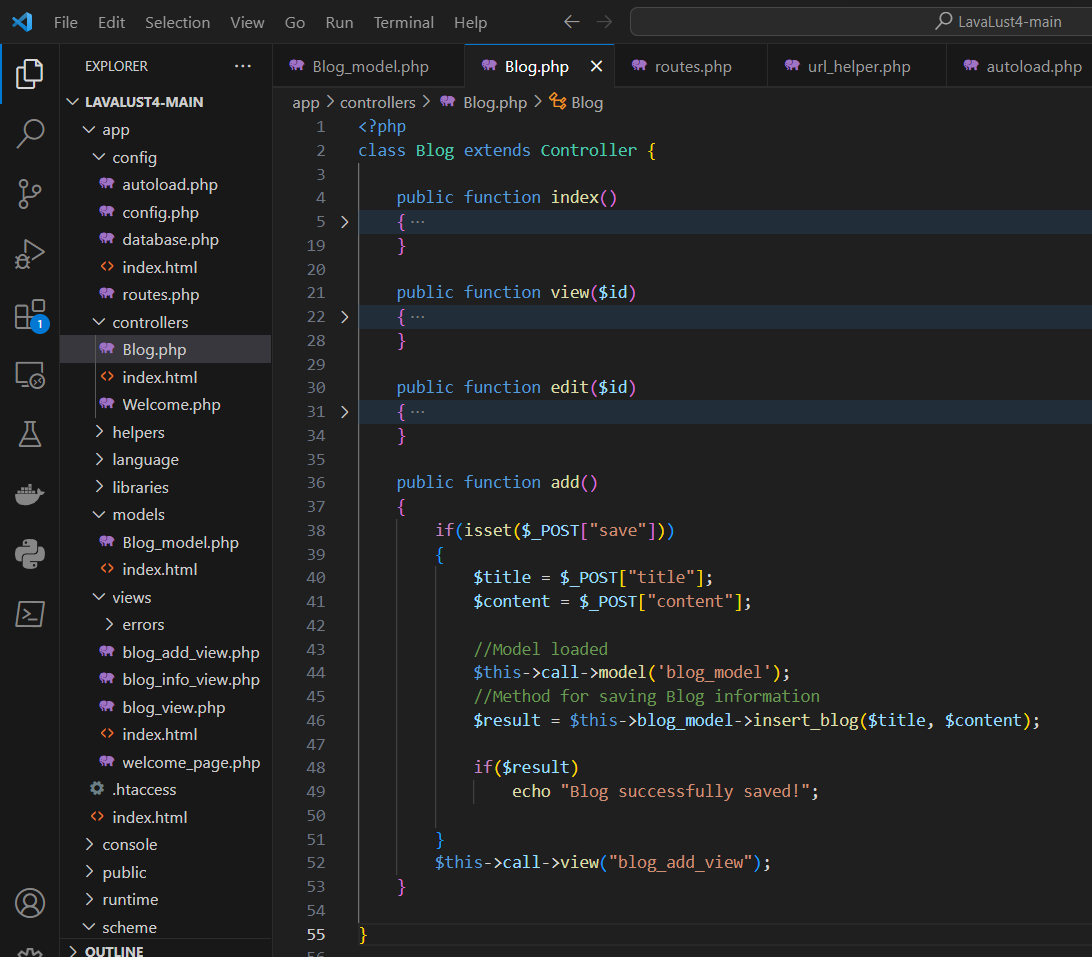
We will use the **URL** helper that assist us in creating links

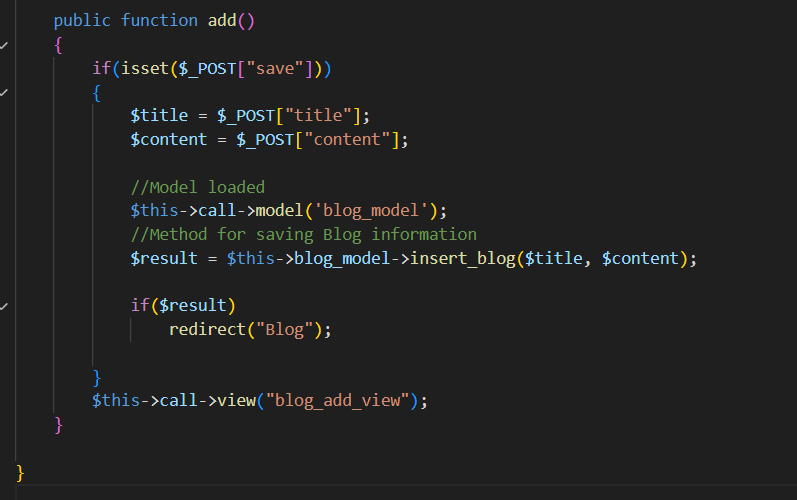


Now we can use the **redirect()** method from **url\_helper** to redirect into another page.



Let’s try in **Blog** Controller:

Before:

After: 

***Now, try to submit our form.***

Built-in Helpers

* URL Helper
  + base\_url() – return the value of base url in config.php
  + redirect($url) – redirect to specified route
  + load\_js($array\_paths) - Use to load of Javascript
  + load\_css($array\_paths) - Use to load of CSS
  + active($current\_url, $css\_class=’active’) – Set menu as active
* Security Helper
* String Helper
* Cookie Helper
* Etc.

Using Autoload

LavaLust comes with an “Auto-load” feature that permits **libraries**, **helpers**, and **models** to be initialized automatically every time the system runs. If you need certain resources globally throughout your application you should consider auto-loading them for convenience.

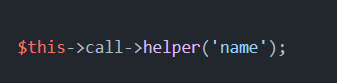
The following items can be loaded automatically:

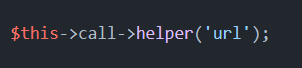
* Classes found in the ***libraries/*** directory
* Helper files found in the ***helpers/*** directory
* Models found in the ***models/*** folder

To autoload resources, open the **app/config/autoload.php**

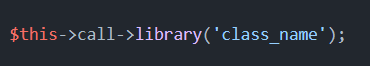
**Note:** Libraries, Helpers, and Models can also be loaded inside our **Controller**. (without using the autoload)

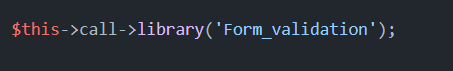
**Helper:**



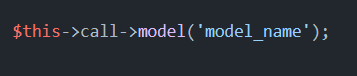
Example: 

**Library:**



Example: 

**Model:**

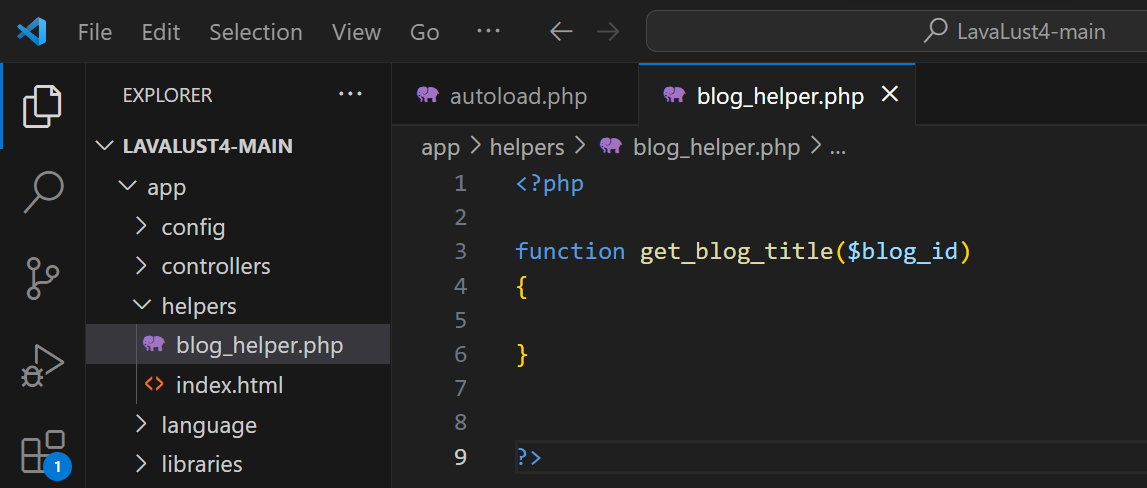


Example: 

Create your own Helper

You can also create your own Helper & Libraries. To create your own helper, create a php file (i.e.: helpername\_helper.php) and save it inside **app/helpers/** folder.

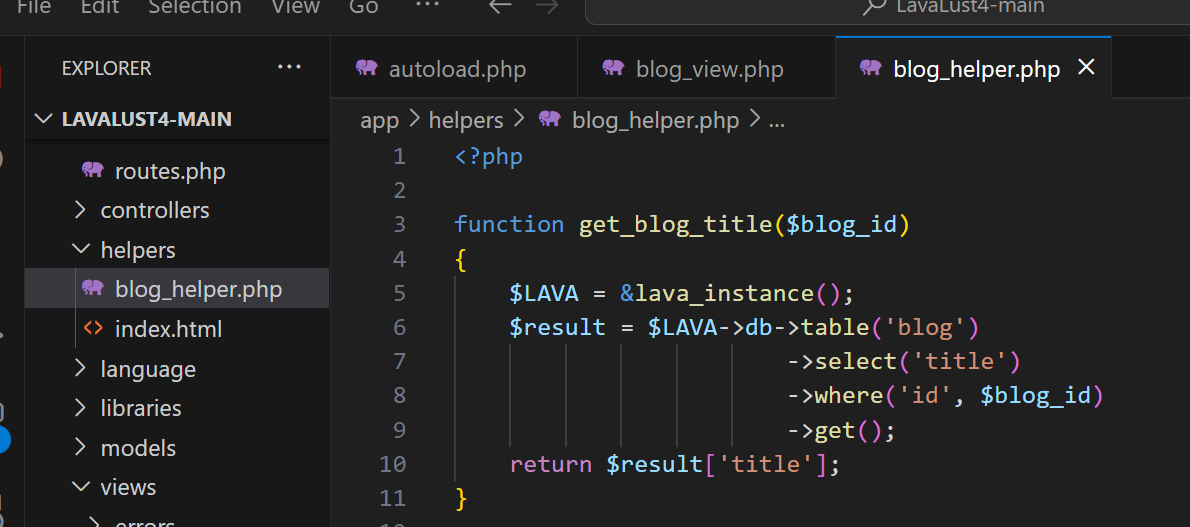
**Example:** blog\_helper



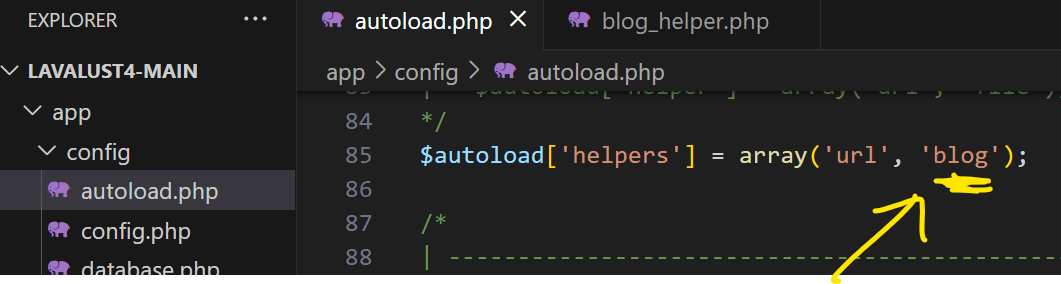
Now we will use the **get\_blog\_title()** function to get from the database the title of a specified **blog\_id**.

**Note:** we can also use the **&lava\_instance();** to access the **database** library in our helper. With that, we can use our helper same as the model.

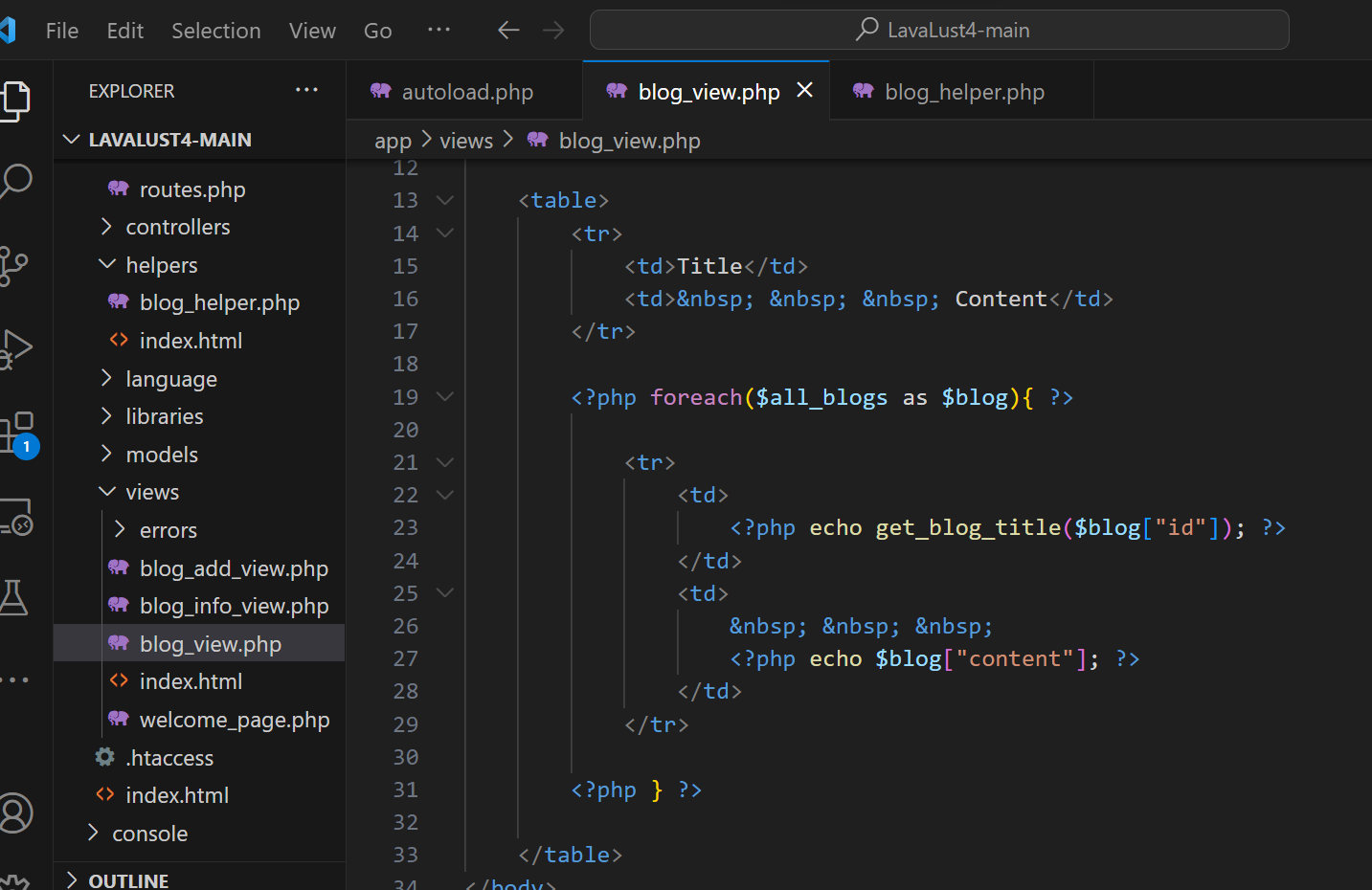
**Example:**



Now, let’s load our helper into ***autoload.php***



When the helper is loaded, we can use **all** the **functions** inside the ***blog\_helper***. Let’s try to use the ***get\_blog\_title()*** function in our views.



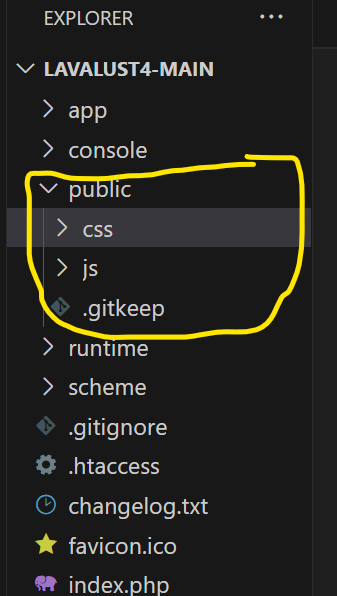
**Output:**



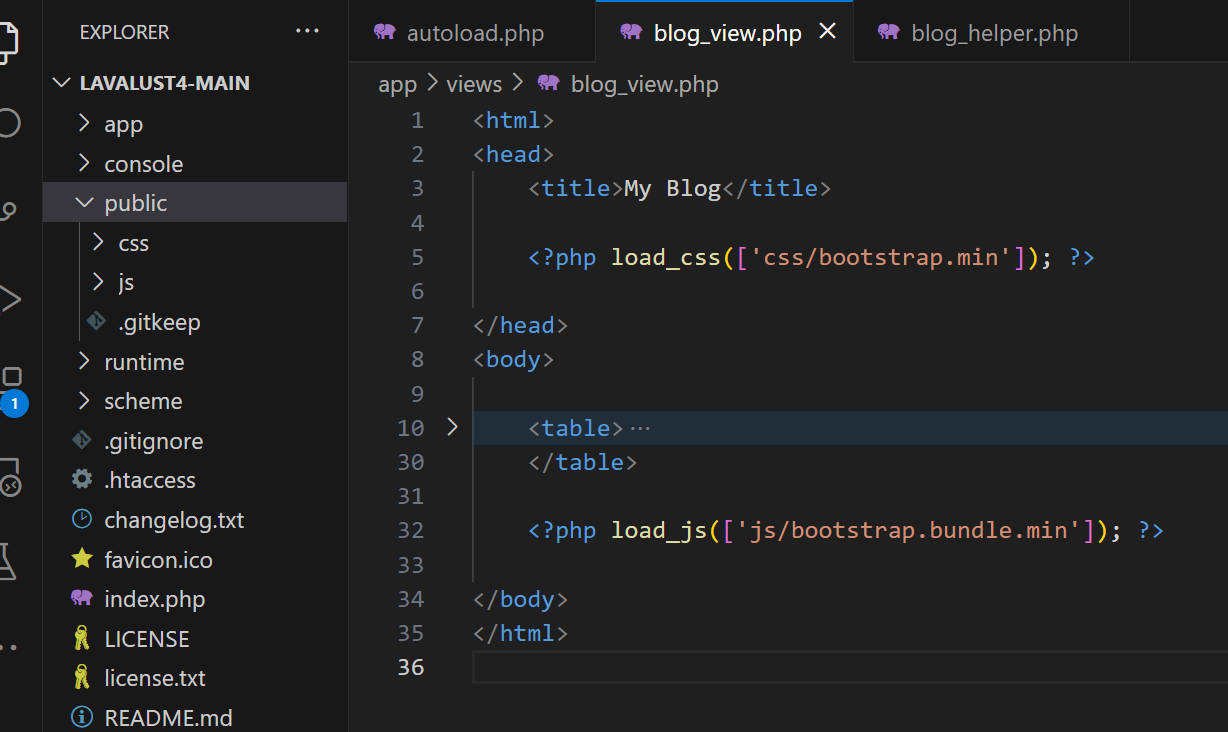
Links, and Public / Assets (css, js, images, etc) form Views

Public / Assets

To use css, js, and other assets, store it inside the ***public*** folder.

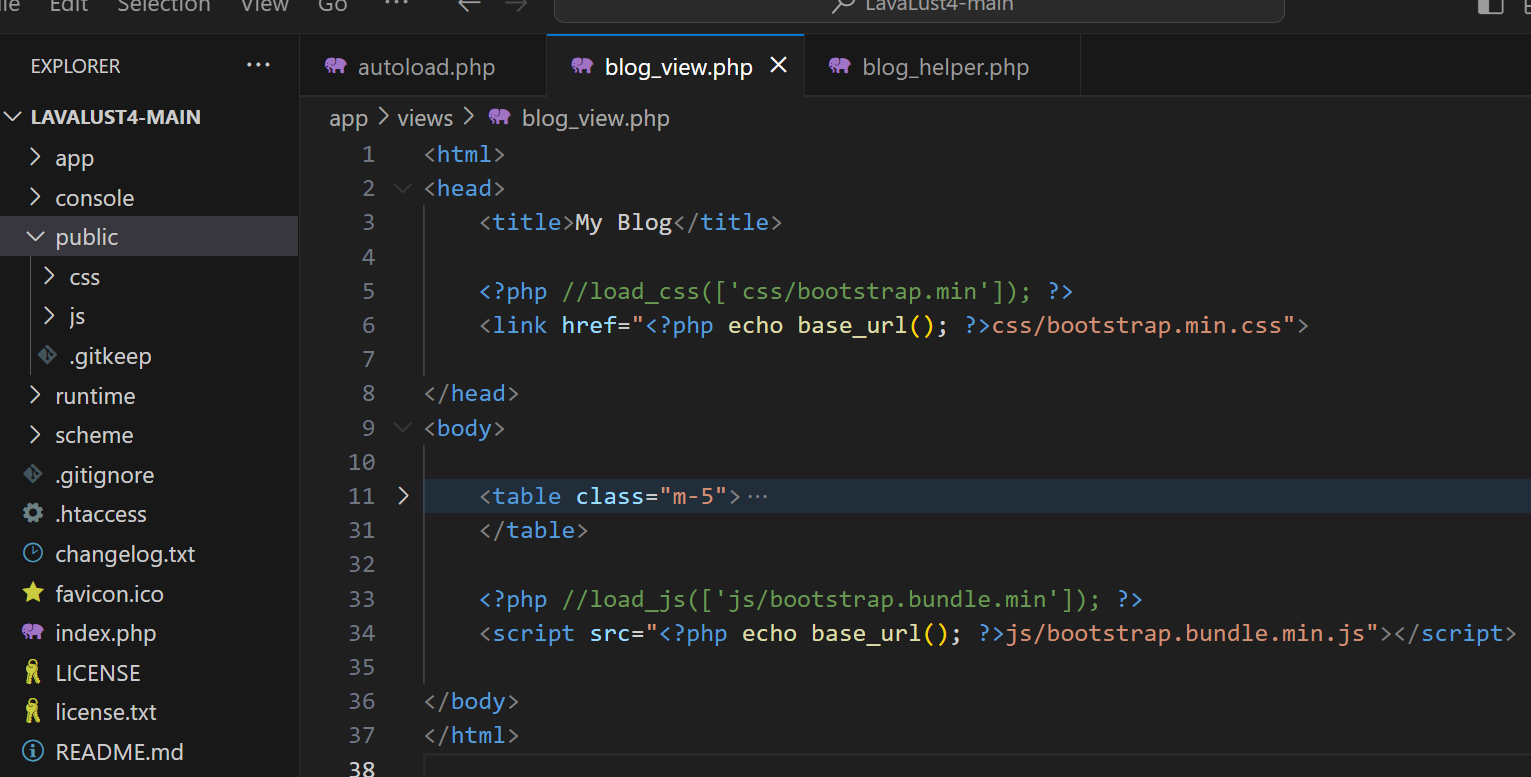


Now, use the load\_css() and load\_js() function from url\_helper.

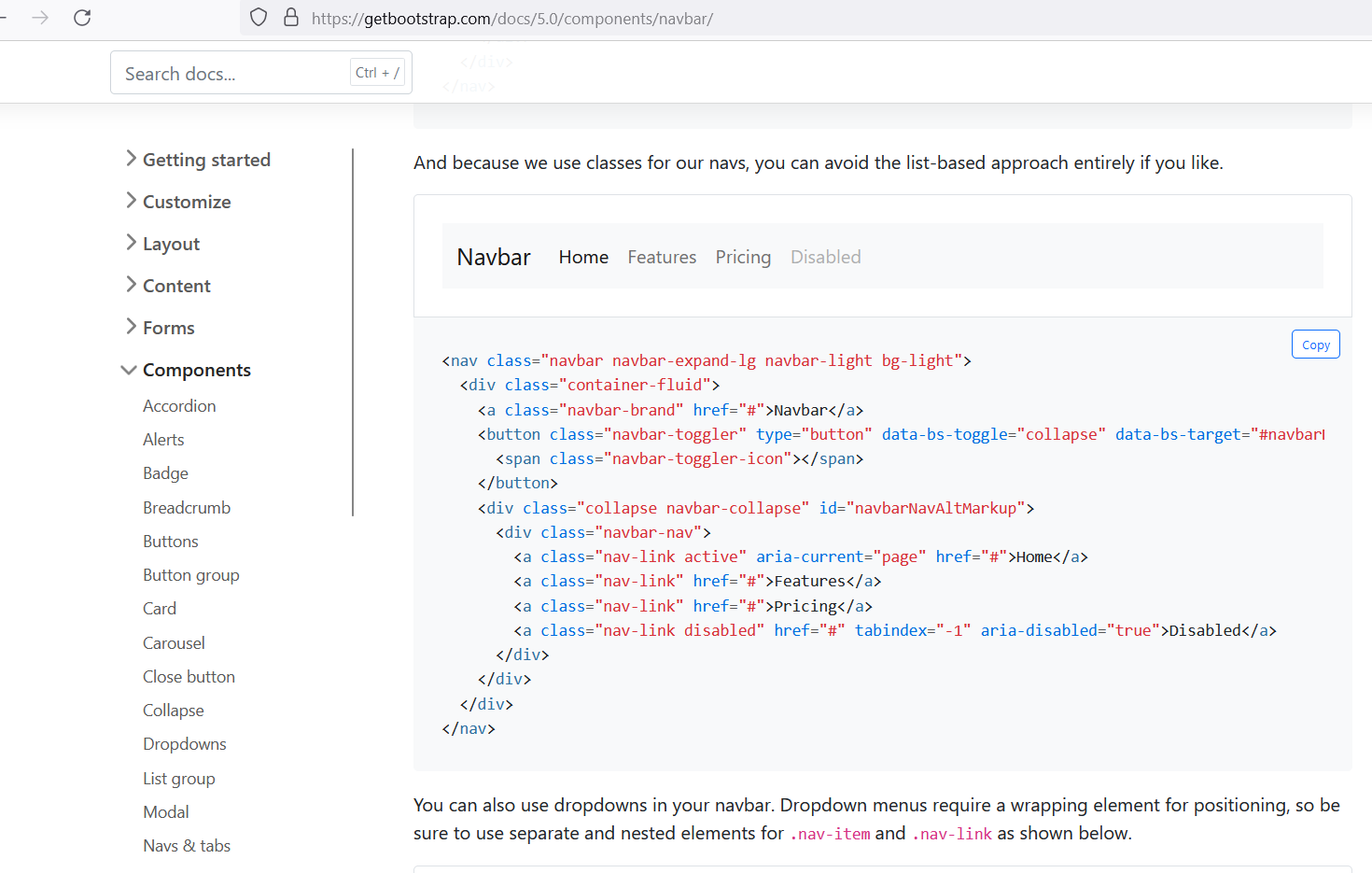


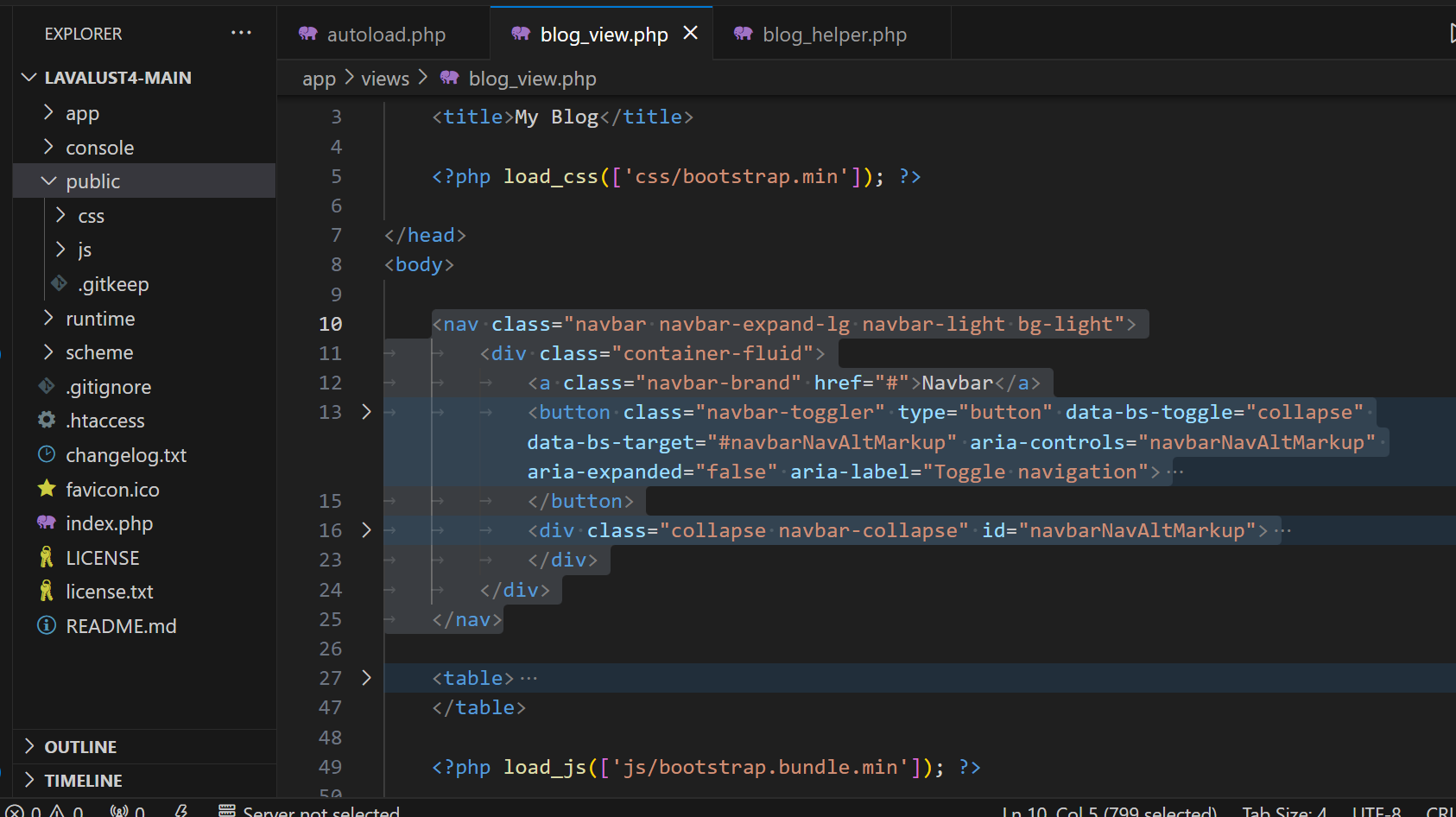
***Note:*** Other way of accessing public assets is using the **base\_url()**.

Example:



OK! Let’s copy some code from Bootstrap5 documentation.





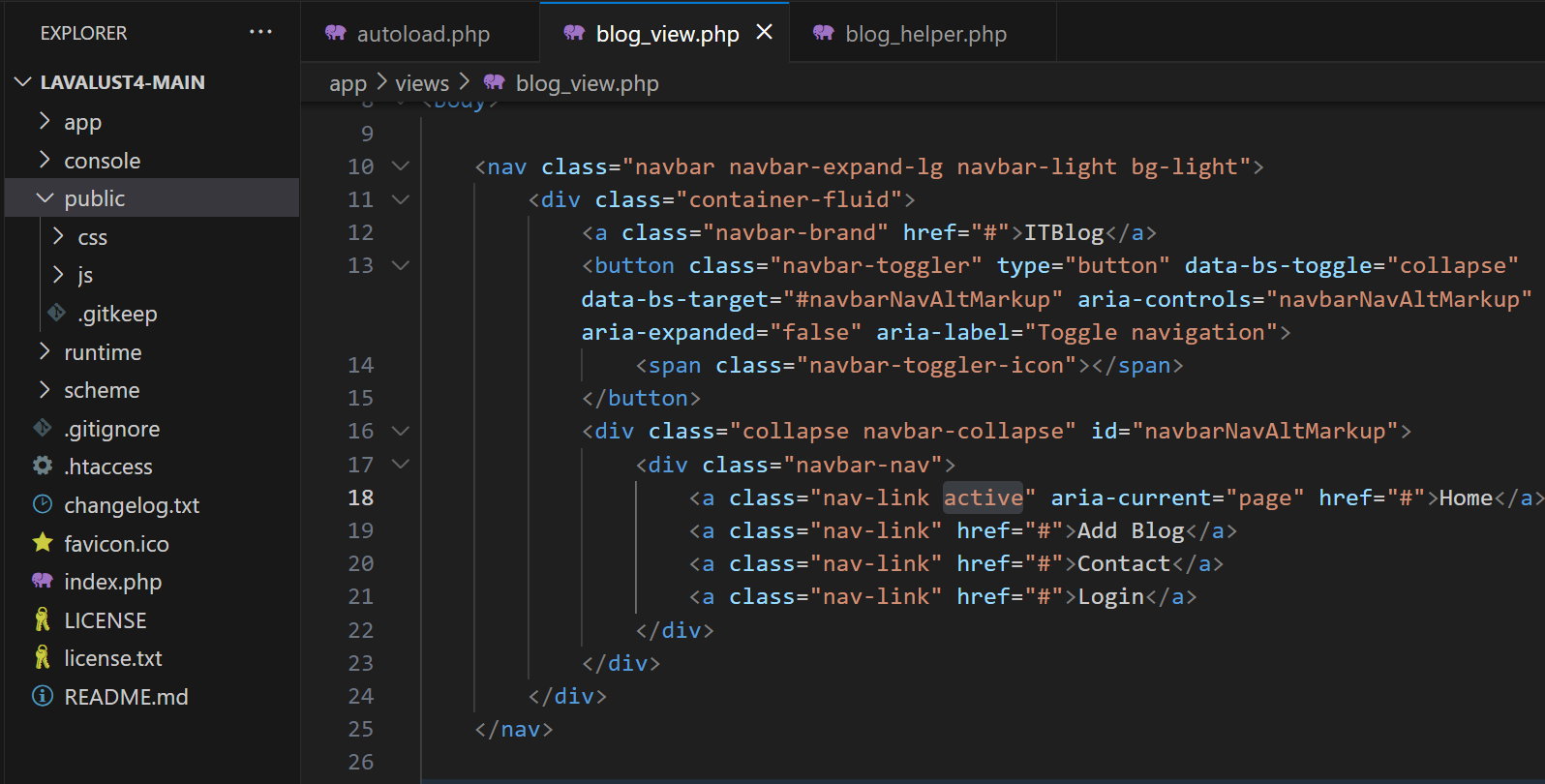
**Output:**

Now, we have a navigation bar. Let’s customized it!



Links

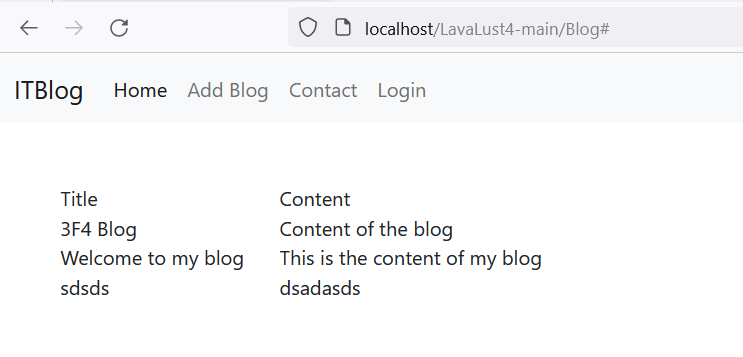
**Navigation Bar**



Let’s add the **route** for every menu/navigation.



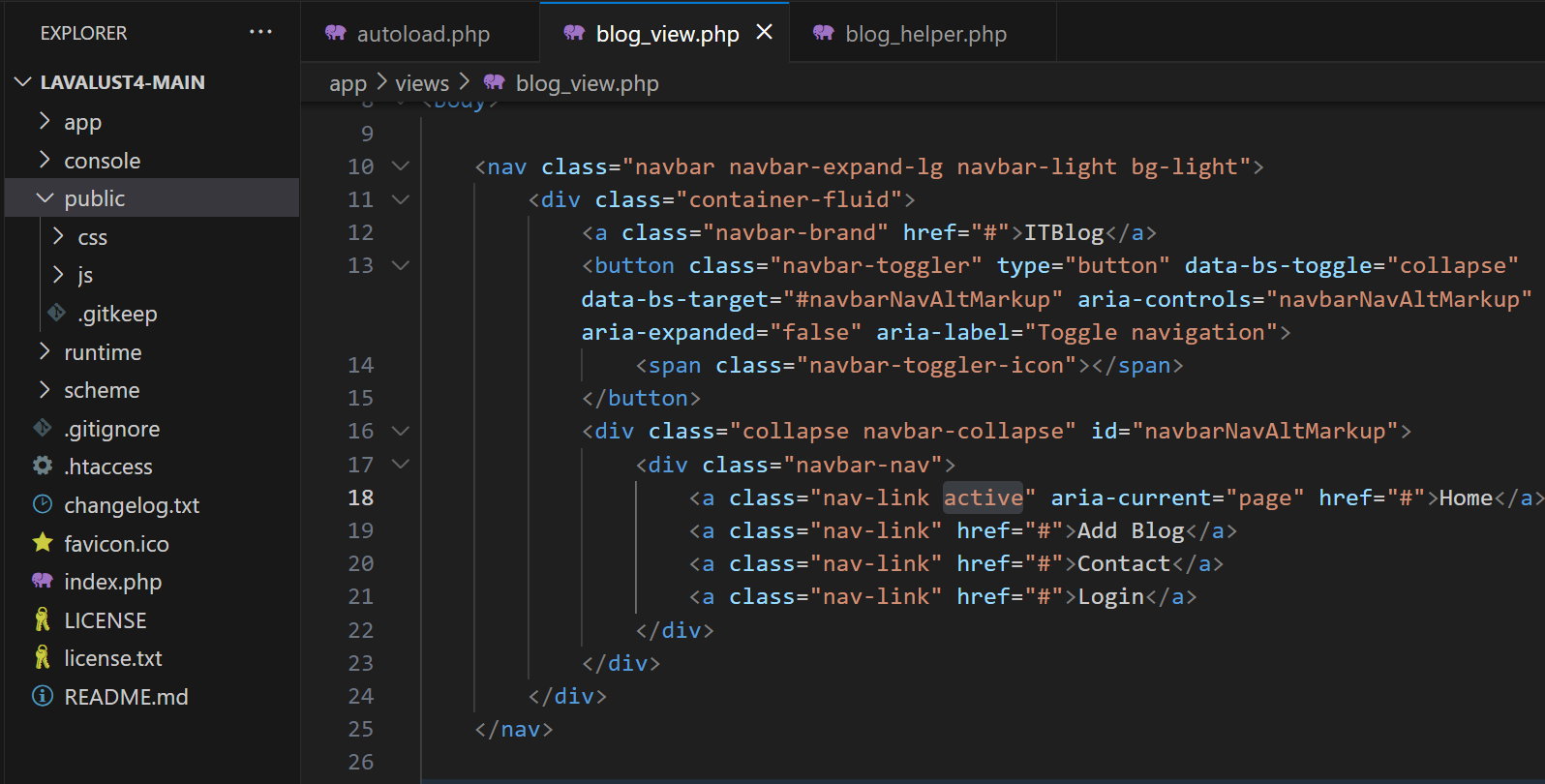
Now, we can switch to different page using our Navigation Bar.



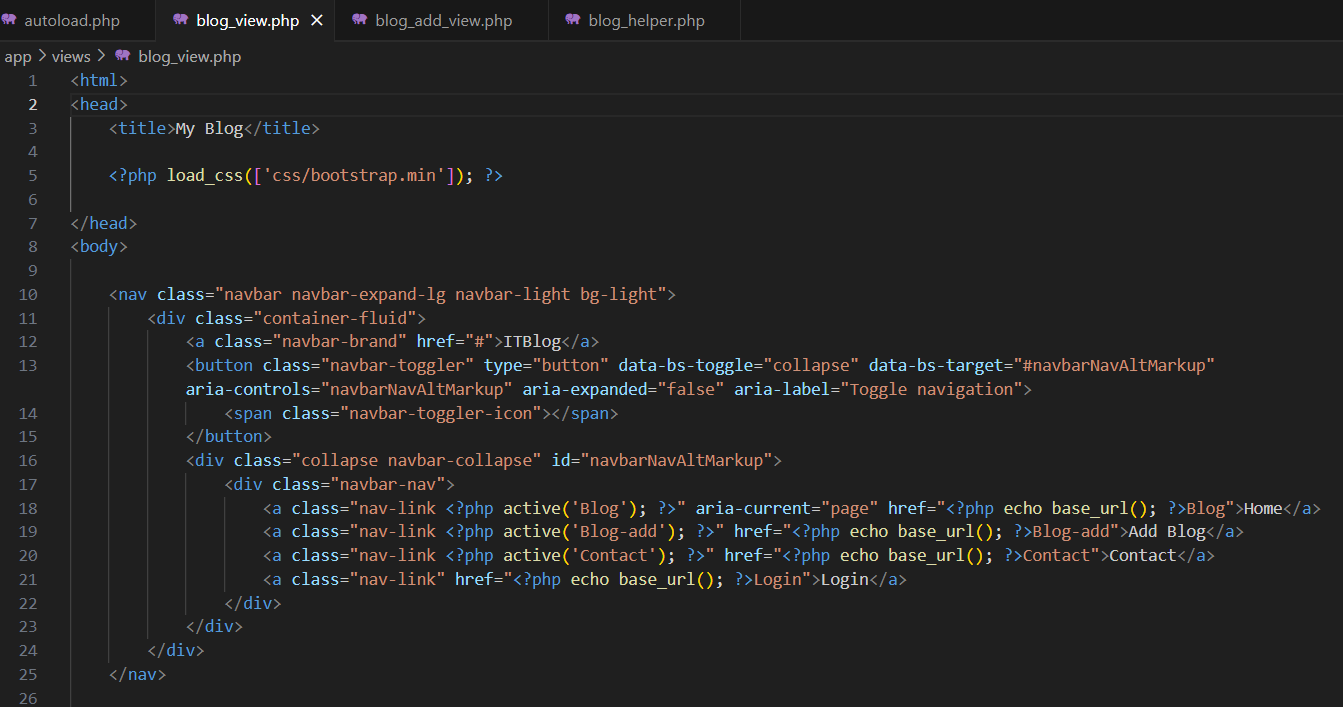
Now, copy the navigation bar to all view file.

**Active**

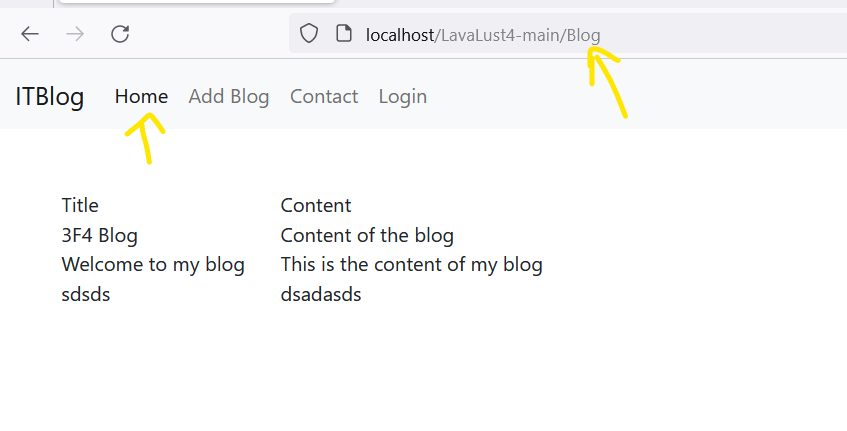
If you see the “**active”** class inside the anchor tag, this is a css class use to highlight the active nav.

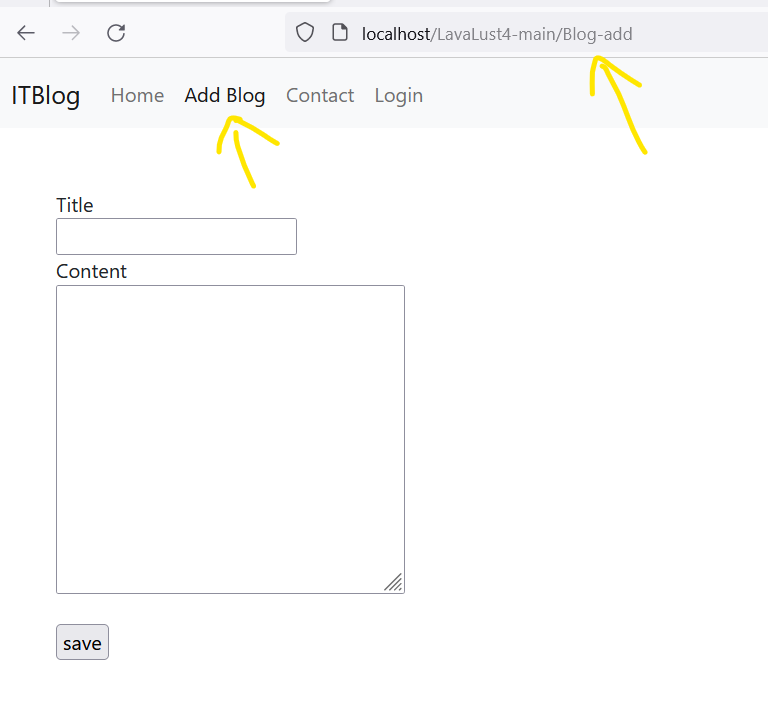


Let’s use the **active()** function to make it dynamic.



**Output:**

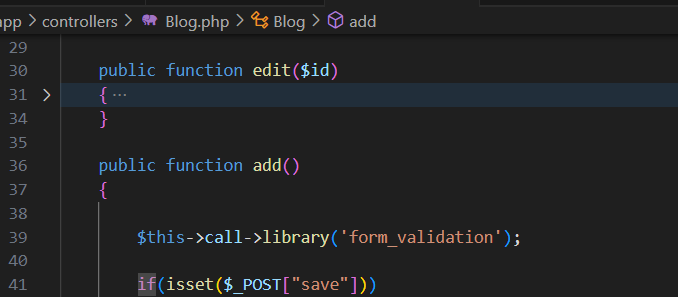




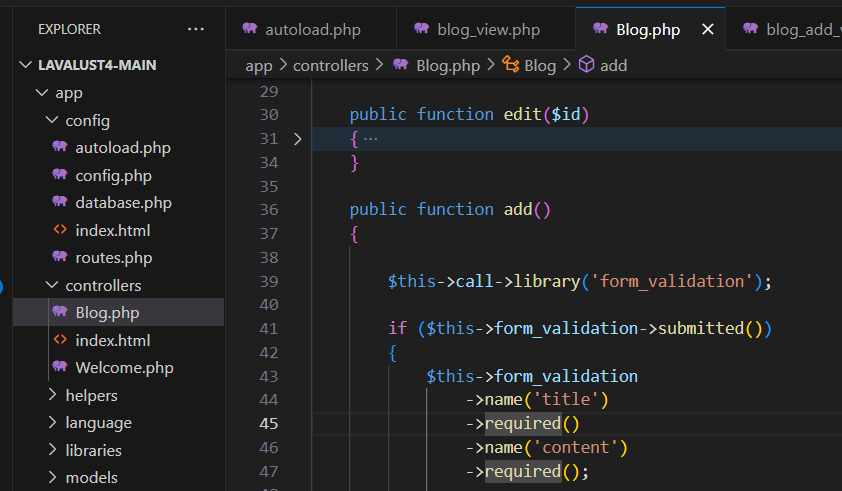
Library

Form Validation Library

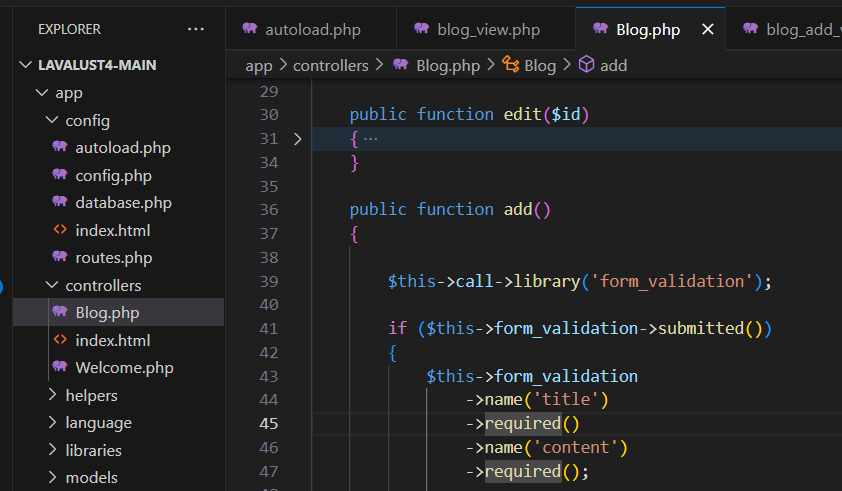
Load the form\_validation library. This time, we load it manually inside controller.



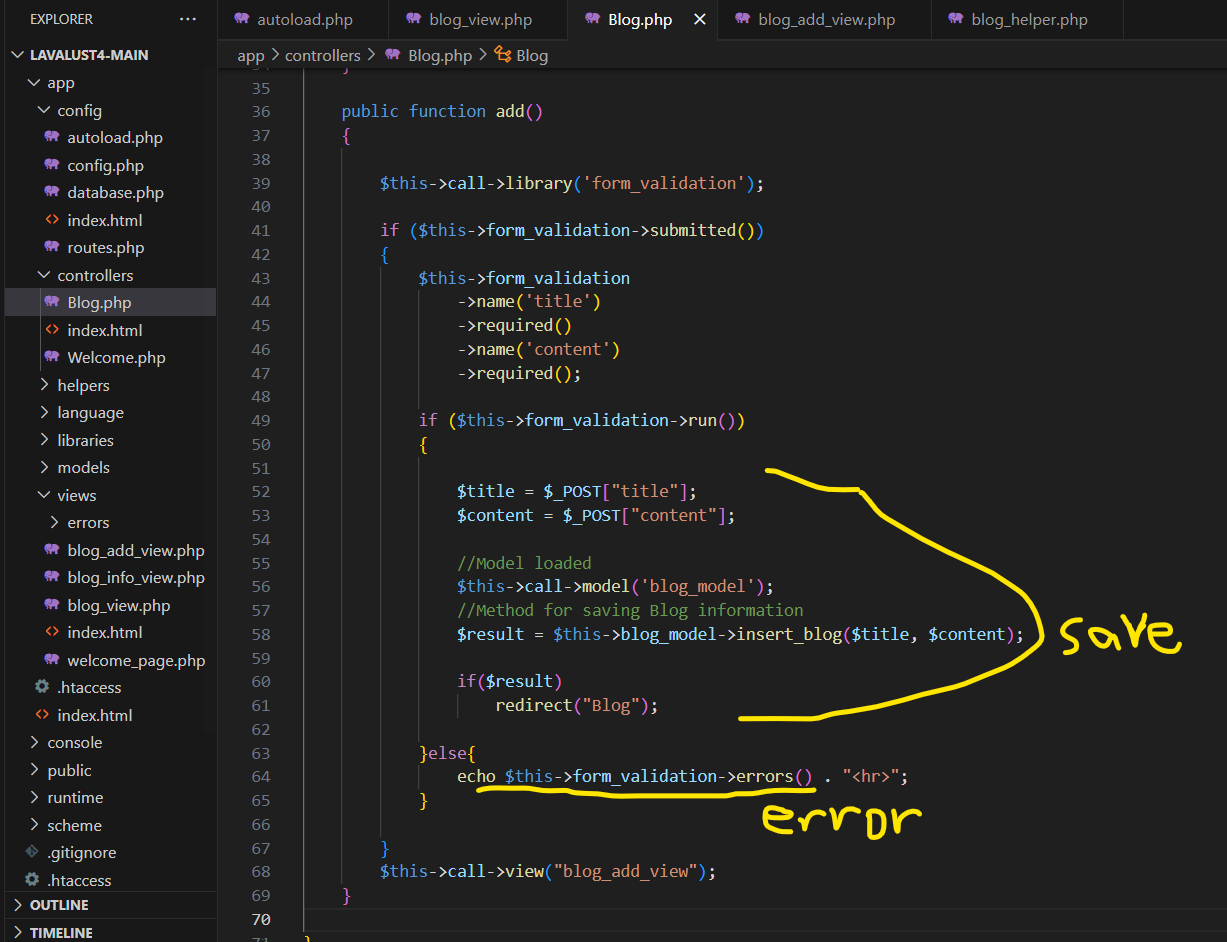
Use the **submitted()** method to replace the condition **if(isset($\_POST[“save”])){**

****

Let’s validate our inputs (title & content)



The above validation will check the **value** of **title** and **content** input if **empty** or not. To test the validation, use **run()** method of the form\_validation class.



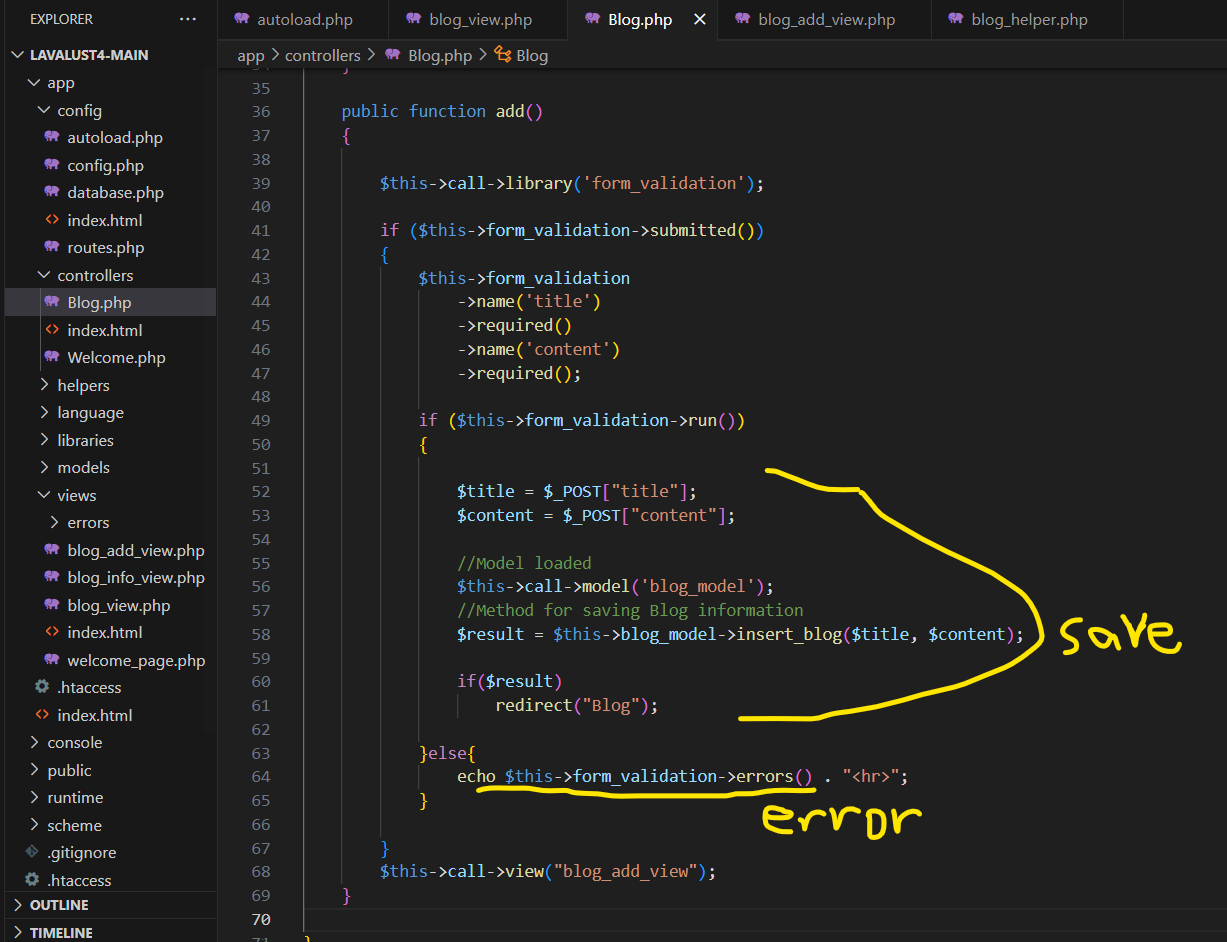
If the validation returns an error, it will display what kind of input error or else if no error, the data will save in database.

For more validation methods: <https://lavalust.netlify.app/#item-5-4>

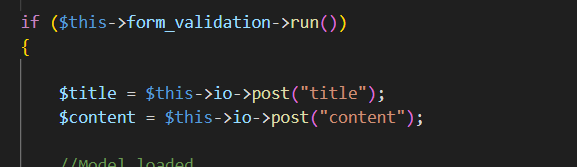
(IO) Input and Output Class

This class is initialized automatically by the system so there is no need to do it manually.

In the image below, input will fetch directly the data without checking if the item is set and return NULL if not. <https://lavalust.netlify.app/#item-5-8>

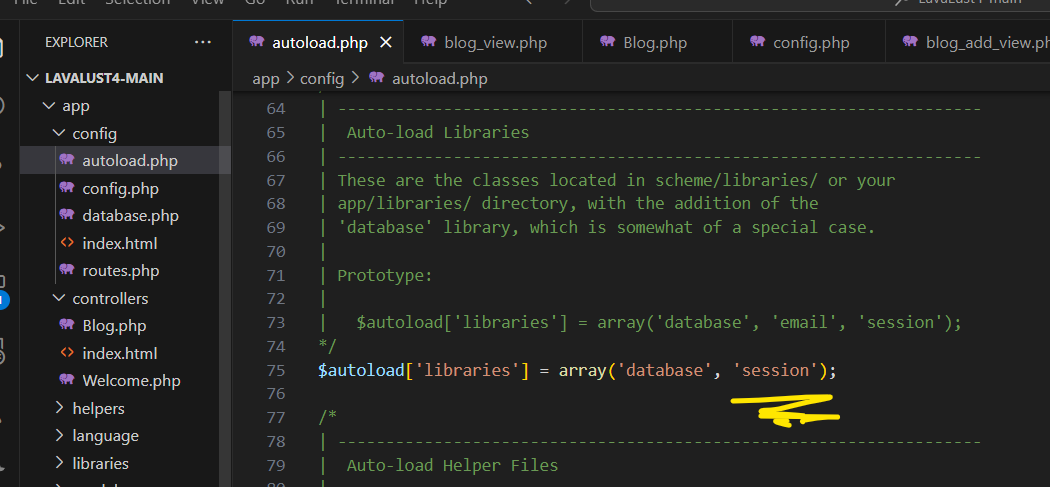


Now, let’s use the methods from the **io** class.



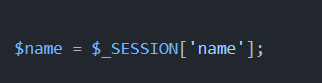
Session Library

To use the session, load it first in ***autoload.php***

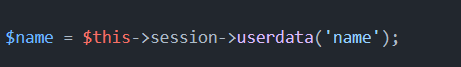


Retrieving Session Data

**$\_SESSION** superglobal

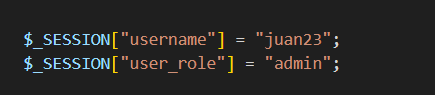


Using **Session** library

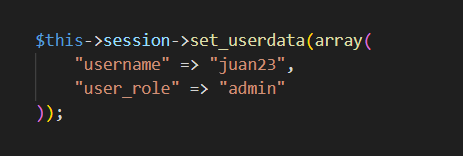


Adding Session Data

**$\_SESSION** superglobal

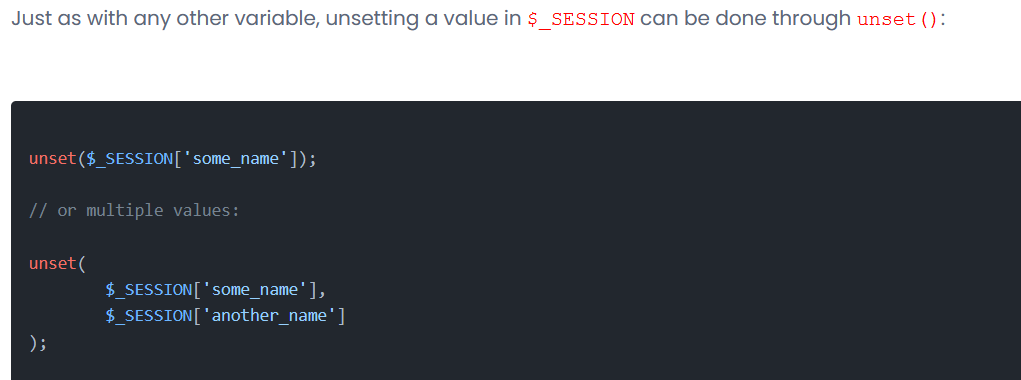


Using **Session** library

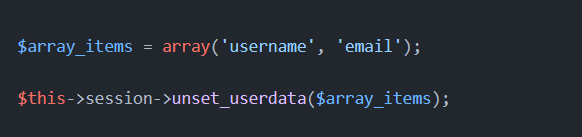


Removing Session Data

**$\_SESSION** superglobal



Using **Session** Library



Security

* **Identify sensitive data to be protected.**
  + Password encryption
  + Session Role
* **Validate all incoming data.**
  + Form Validation
* **Review and secure File Permissions.**
* **Test for Cross Site Scripting (XSS)**
* **Check for SQL Injection vulnerabilities.**
* **Secure file uploads.**
* **Update all PHP versions to the latest**
* **Disable display\_errors.**
  + **ENVIRONMENT** = “Production” in *config.php*
* **Ensure important security headers such as X-Content-Type and X-XSS-Protection are implemented.**
* **Check for Cross-Site Request Forgery**
  + **CSRF Protection** in *config.php*

Before Deploying your application

* **Disable display\_errors**