LAVALUST DEVELOPMENT

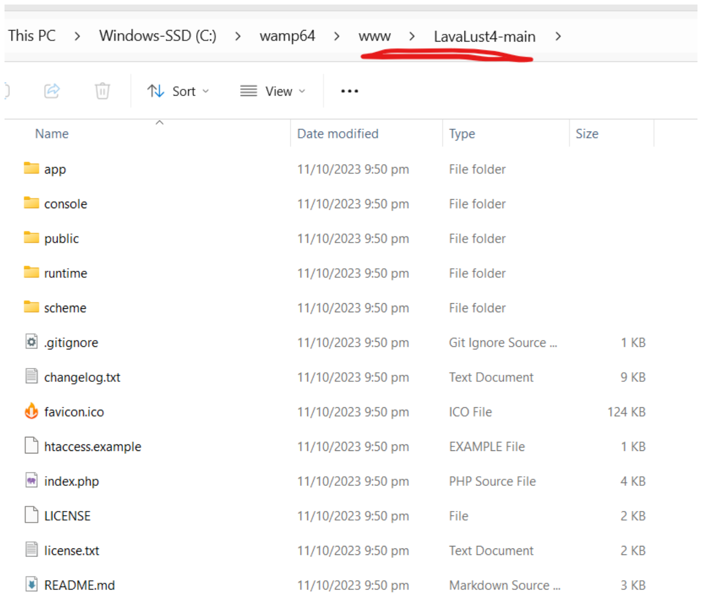
Contents

INSTALLATION

- ✓ **DOWNLOAD**

  https://github.com/ronmarasigan/LavaLust4

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

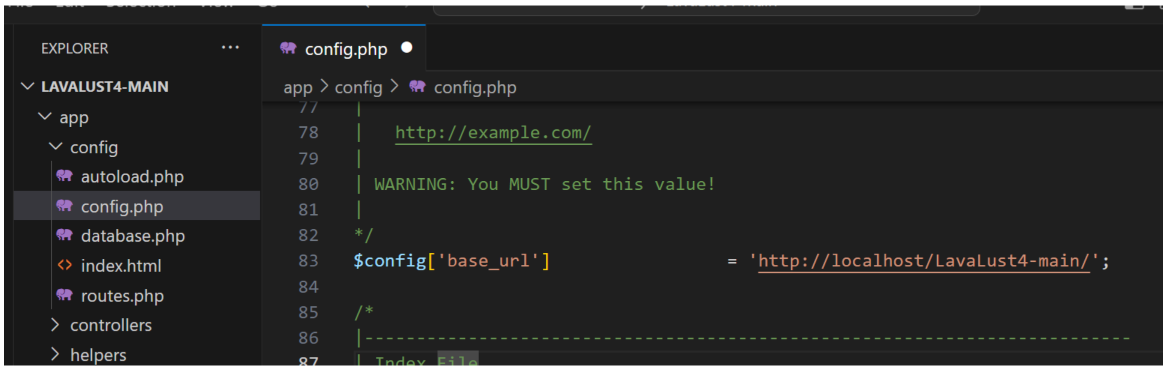


This PC > Windows-SSD (C:) > wamp64 > www > LavaLust4-main >

Name	Date modified	Type	Size
app	11/10/2023 9:50 pm	File folder	
console	11/10/2023 9:50 pm	File folder	
public	11/10/2023 9:50 pm	File folder	
runtime	11/10/2023 9:50 pm	File folder	
scheme	11/10/2023 9:50 pm	File folder	
.gitignore	11/10/2023 9:50 pm	Git Ignore Source ...	1 KB
changelog.txt	11/10/2023 9:50 pm	Text Document	9 KB
favicon.ico	11/10/2023 9:50 pm	ICO File	124 KB
htaccess.example	11/10/2023 9:50 pm	EXAMPLE File	1 KB
index.php	11/10/2023 9:50 pm	PHP Source File	4 KB
LICENSE	11/10/2023 9:50 pm	File	2 KB
license.txt	11/10/2023 9:50 pm	Text Document	2 KB
README.md	11/10/2023 9:50 pm	Markdown Source ...	3 KB

CONFIGURATION

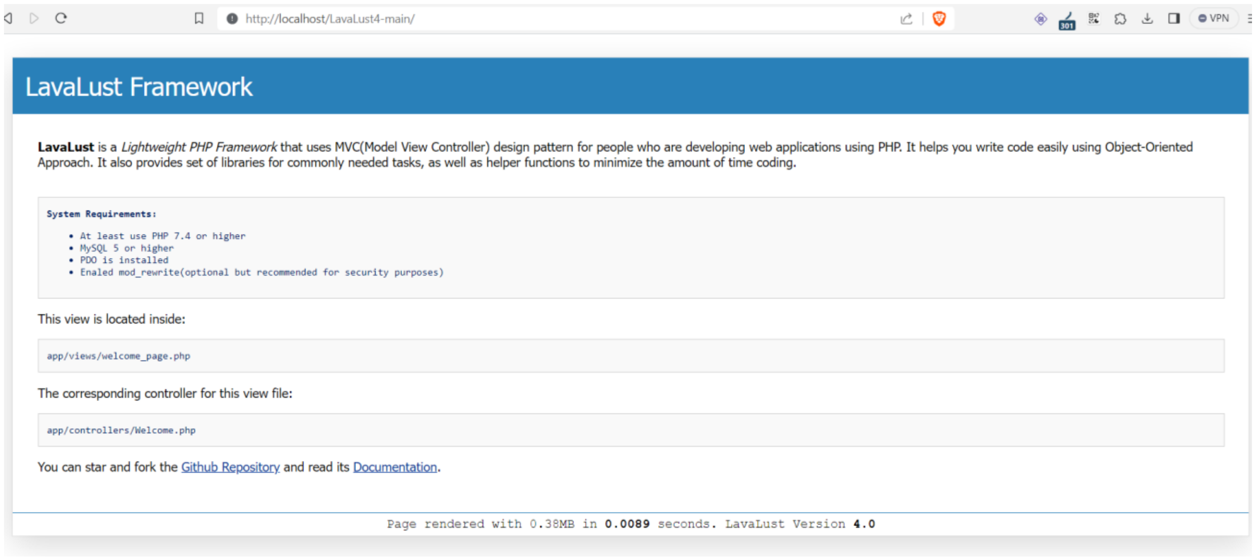
- ✓ Open the **app/config/config.php**
- ✓ Go to **$config['base_url'] = '';** and add your local project url



```
|
|   http://example.com/
|
| WARNING: You MUST set this value!
|
*/
$config['base_url']                 = 'http://localhost/LavaLust4-main/';

/*
|--------------------------------------------------------------------------
| Index File
```

FIRST RUN

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



LavaLust Framework

LavaLust is a *Lightweight PHP Framework* that uses MVC(Model View Controller) design pattern for people who are developing web applications using PHP. It helps you write code easily using Object-Oriented Approach. It also provides set of libraries for commonly needed tasks, as well as helper functions to minimize the amount of time coding.

```
System Requirements:
  • At least use PHP 7.4 or higher
  • MySQL 5 or higher
  • PDO is installed
  • Enaled mod_rewrite(optional but recommended for security purposes)
```

This view is located inside:

```
app/views/welcome_page.php
```

The corresponding controller for this view file:

```
app/controllers/Welcome.php
```

You can star and fork the Github Repository and read its Documentation.

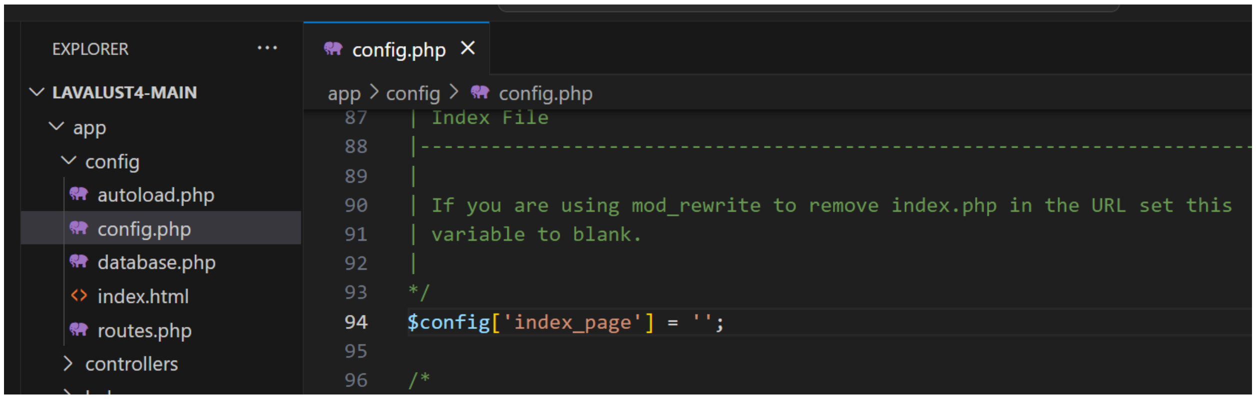
```
Page rendered with 0.38MB in 0.0089 seconds. LavaLust Version 4.0
```

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/ for more details

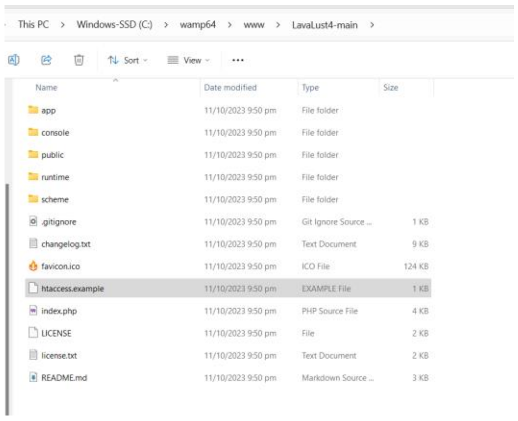


```
| Index File
|--------------------------------------------------------------------
|
| If you are using mod_rewrite to remove index.php in the URL set this
| variable to blank.
|
*/
$config['index_page'] = '';

/*
```

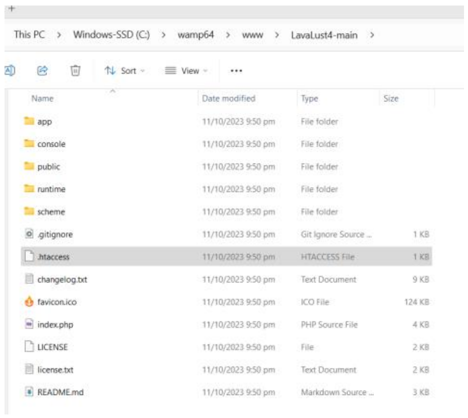
- ✓ Add/create **.htaccess** to your project root directory and add the following code.

```
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**


This PC > Windows-SSD (C:) > wamp64 > www > LavaLust4-main >

Name	Date modified	Type	Size
app	11/10/2023 9:50 pm	File folder	
console	11/10/2023 9:50 pm	File folder	
public	11/10/2023 9:50 pm	File folder	
runtime	11/10/2023 9:50 pm	File folder	
scheme	11/10/2023 9:50 pm	File folder	
.gitignore	11/10/2023 9:50 pm	Git Ignore Source ...	1 KB
changelog.txt	11/10/2023 9:50 pm	Text Document	9 KB
favicon.ico	11/10/2023 9:50 pm	ICO File	124 KB
htaccess.example	11/10/2023 9:50 pm	EXAMPLE File	1 KB
index.php	11/10/2023 9:50 pm	PHP Source File	4 KB
LICENSE	11/10/2023 9:50 pm	File	2 KB
license.txt	11/10/2023 9:50 pm	Text Document	2 KB
README.md	11/10/2023 9:50 pm	Markdown Source ...	3 KB


This PC > Windows-SSD (C:) > wamp64 > www > LavaLust4-main >

Name	Date modified	Type	Size
app	11/10/2023 9:50 pm	File folder	
console	11/10/2023 9:50 pm	File folder	
public	11/10/2023 9:50 pm	File folder	
runtime	11/10/2023 9:50 pm	File folder	
scheme	11/10/2023 9:50 pm	File folder	
.gitignore	11/10/2023 9:50 pm	Git Ignore Source ...	1 KB
.htaccess	11/10/2023 9:50 pm	HTACCESS File	1 KB
changelog.txt	11/10/2023 9:50 pm	Text Document	9 KB
favicon.ico	11/10/2023 9:50 pm	ICO File	124 KB
index.php	11/10/2023 9:50 pm	PHP Source File	4 KB
LICENSE	11/10/2023 9:50 pm	File	2 KB
license.txt	11/10/2023 9:50 pm	Text Document	2 KB
README.md	11/10/2023 9:50 pm	Markdown Source ...	3 KB

PARTS OF URL

URI Segments

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

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

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

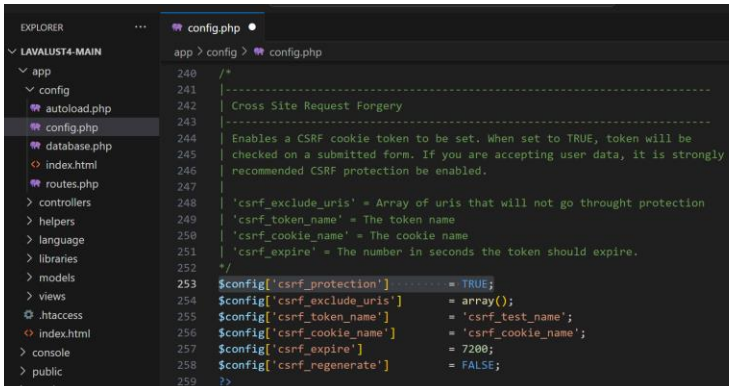
The segments in the URL, in following with the MVC approach, usually represent:

- 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**

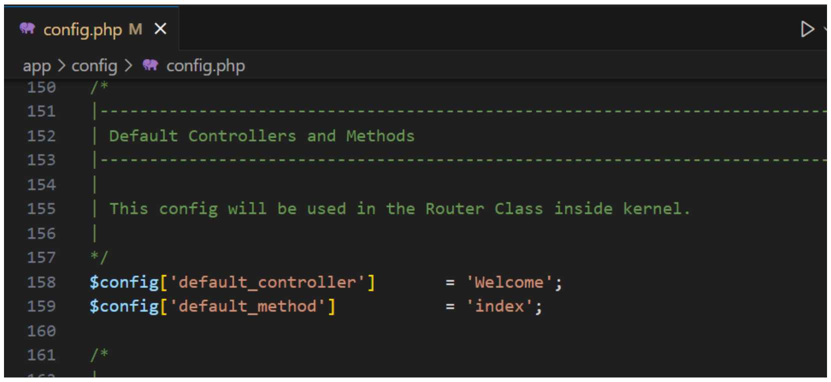


```php
/*
|--------------------------------------------------------------------------
| Cross Site Request Forgery
|--------------------------------------------------------------------------
| Enables a CSRF cookie token to be set. When set to TRUE, token will be
| checked on a submitted form. If you are accepting user data, it is strongly
| recommended CSRF protection be enabled.
|
| 'csrf_exclude_uris' = Array of uris that will not go throught protection
| 'csrf_token_name' = The token name
| 'csrf_cookie_name' = The cookie name
| 'csrf_expire' = The number in seconds the token should expire.
*/
$config['csrf_protection']          = TRUE;
$config['csrf_exclude_uris']        = array();
$config['csrf_token_name']          = 'csrf_test_name';
$config['csrf_cookie_name']         = 'csrf_cookie_name';
$config['csrf_expire']              = 7200;
$config['csrf_regenerate']          = FALSE;
?>
```

Default Controller & Method

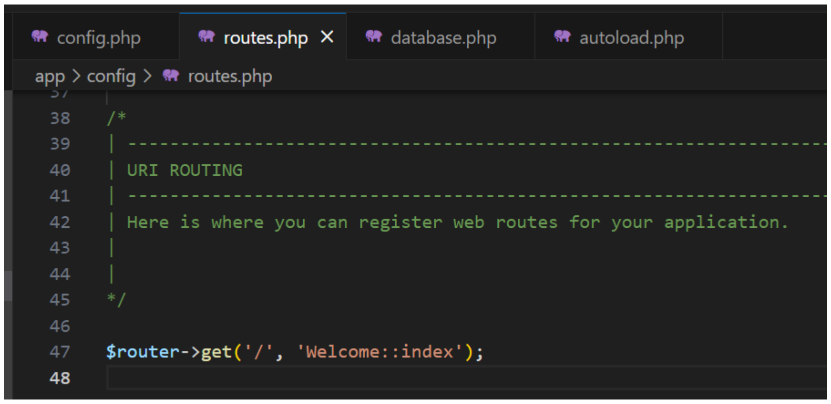
The default controller & method is the page that display/execute first when you access the base URL without specifying the controller & method.

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**



```php
/*
|--------------------------------------------------------------------------
| Default Controllers and Methods
|--------------------------------------------------------------------------
|
| This config will be used in the Router Class inside kernel.
|
*/
$config['default_controller']       = 'Welcome';
$config['default_method']           = 'index';

/*
```

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



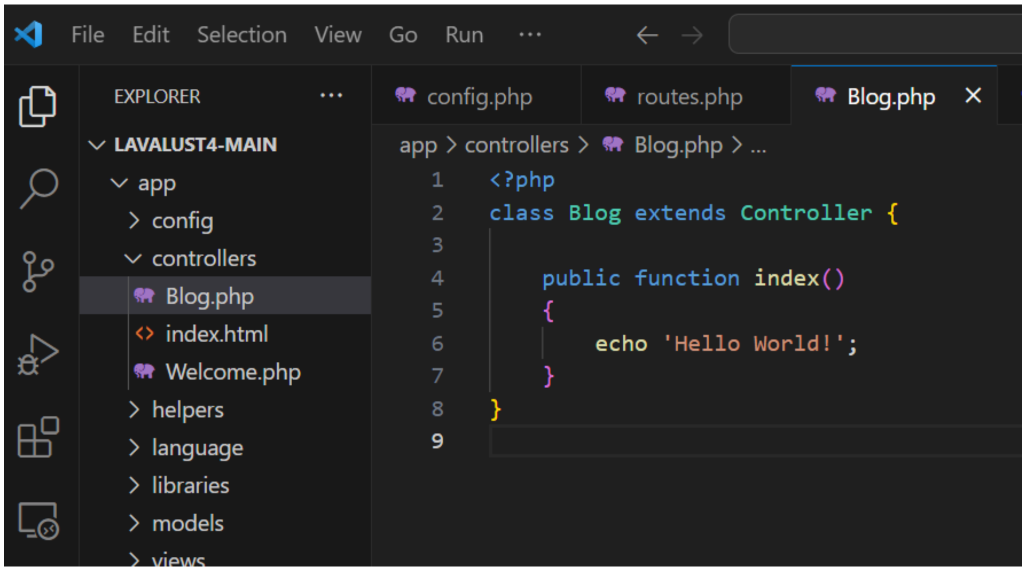
```php
/*
| -------------------------------------------------------------------
| URI ROUTING
| -------------------------------------------------------------------
| Here is where you can register web routes for your application.
|
|
*/

$router->get('/', 'Welcome::index');
```

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

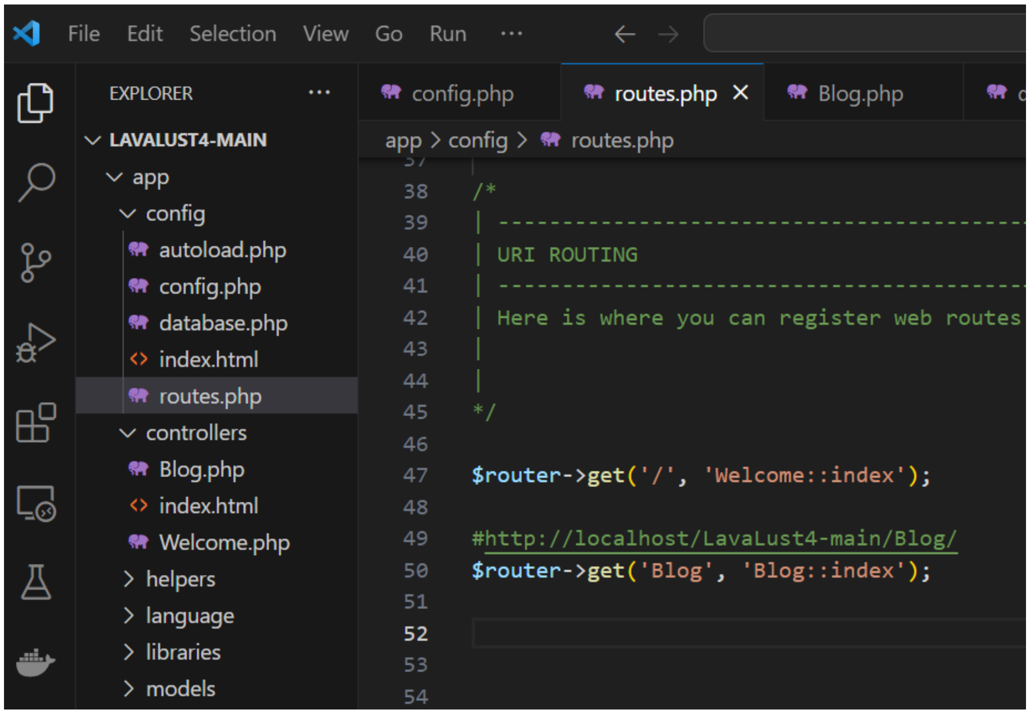
CONTROLLER

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.



```php
<?php
class Blog extends Controller {

    public function index()
    {
        echo 'Hello World!';
    }
}

```

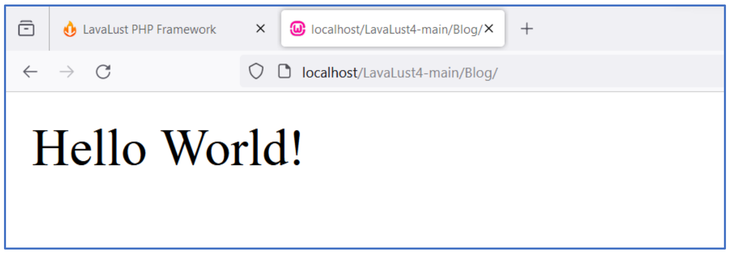
Next is to create a **route** for your **Blog** controller and **index()** method.



```php
/*
| -------------------------------------------------------
| URI ROUTING
| -------------------------------------------------------
| Here is where you can register web routes
|
|
*/

$router->get('/', 'Welcome::index');

#http://localhost/LavaLust4-main/Blog/
$router->get('Blog', 'Blog::index');



```

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:


LavaLust PHP Framework
localhost/LavaLust4-main/Blog/
localhost/LavaLust4-main/Blog/

Hello World!

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:

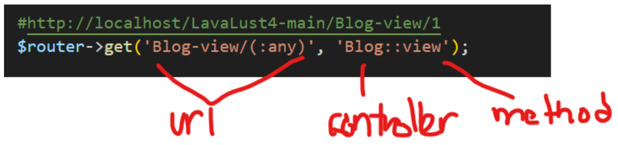
1. **get()** – used for Http Request method **Get**
2. **post()** – used for Http Request method **Post**
3. **match()** – used for both **Get** & **Post** or other Http Request method

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

a) The name used in URL
b) The Class/method from your Controller

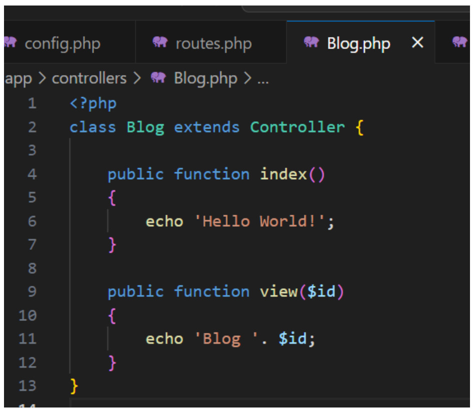
Example:

Route



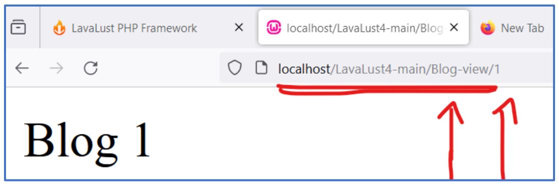
```php
#http://localhost/LavaLust4-main/Blog-view/1
$router->get('Blog-view/(:any)', 'Blog::view');
```

Controller



```php
<?php
class Blog extends Controller {

    public function index()
    {
        echo 'Hello World!';
    }

    public function view($id)
    {
        echo 'Blog '. $id;
    }
}
```

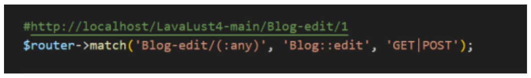
Output


LavaLust PHP Framework
localhost/LavaLust4-main/Blog
New Tab
localhost/LavaLust4-main/Blog-view/1

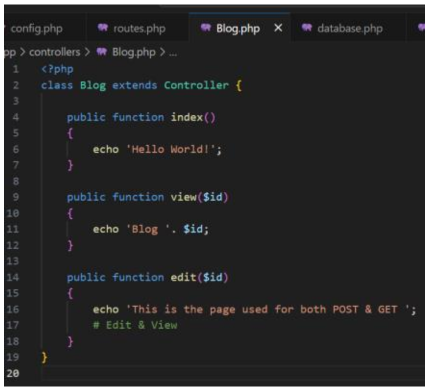
Blog 1

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

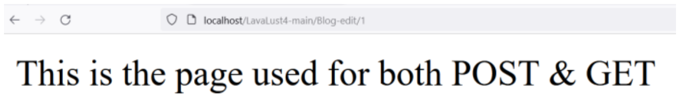
a) The name used in URL
b) The Class/method/ from your Controller
c) And the Http Request (e.g. POST, GET, POST|GET, etc.)



```
#http://localhost/LavaLust4-main/Blog-edit/1
$router->match('Blog-edit/(:any)', 'Blog::edit', 'GET|POST');
```



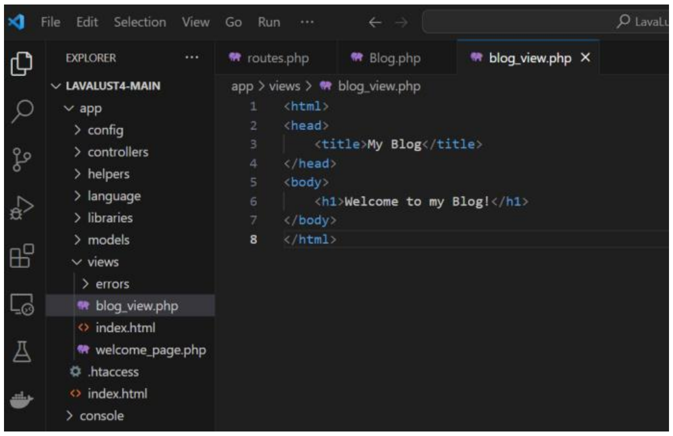
```
<?php
class Blog extends Controller {

    public function index()
    {
        echo 'Hello World!';
    }

    public function view($id)
    {
        echo 'Blog '. $id;
    }

    public function edit($id)
    {
        echo 'This is the page used for both POST & GET ';
        # Edit & View
    }
}
```



This is the page used for both POST & GET

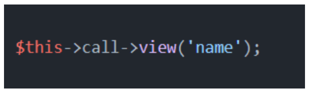
Views

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.



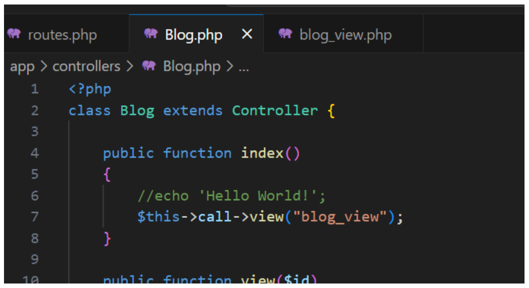
```
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <h1>Welcome to my Blog!</h1>
</body>
</html>
```

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



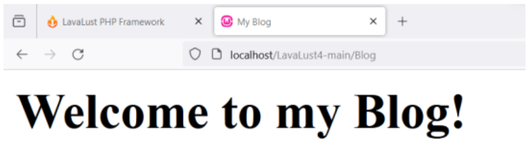
```
$this->call->view('name');
```

Let’s call it in our Controller.



```php
<?php
class Blog extends Controller {

    public function index()
    {
        //echo 'Hello World!';
        $this->call->view("blog_view");
    }

    public function view($id)
```

Output:



```
Welcome to my Blog!
```

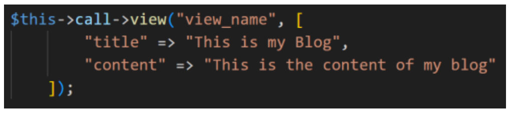
Passing data from Controller-View and View-Controller

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.

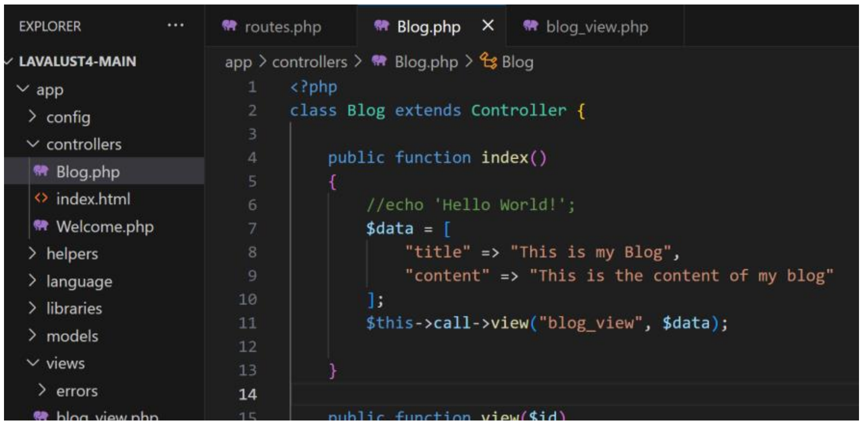
To pass data from controller to view, we put a second parameter in Associative Array format.

Example:



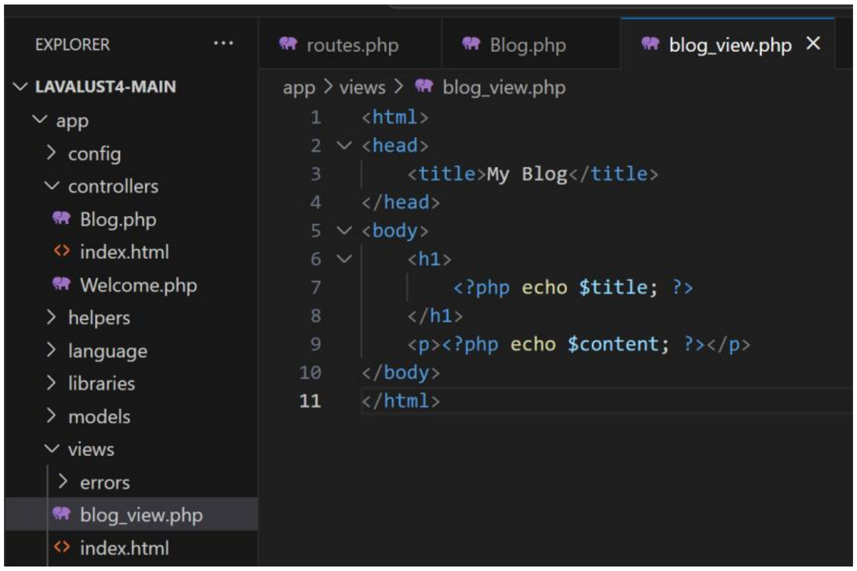
```php
$this->call->view("view_name", [
        "title" => "This is my Blog",
        "content" => "This is the content of my blog"
    ]);
```

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



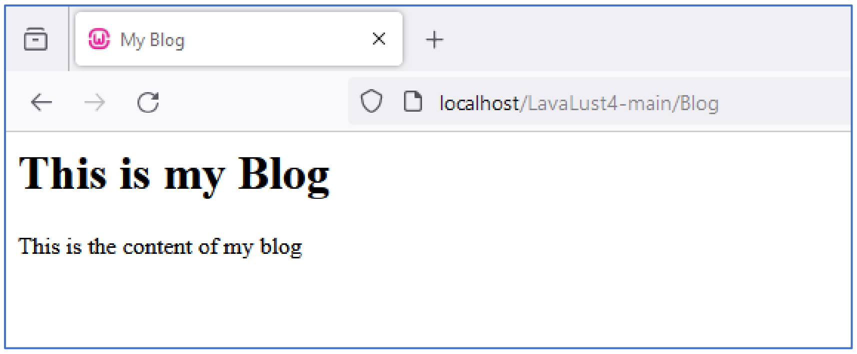
```php
<?php
class Blog extends Controller {

    public function index()
    {
        //echo 'Hello World!';
        $data = [
            "title" => "This is my Blog",
            "content" => "This is the content of my blog"
        ];
        $this->call->view("blog_view", $data);

    }

    public function view($id)
```

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



```
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <h1>
        <?php echo $title; ?>
    </h1>
    <p><?php echo $content; ?></p>
</body>
</html>
```

Output:



This is my Blog

This is the content of my blog

View to Controller

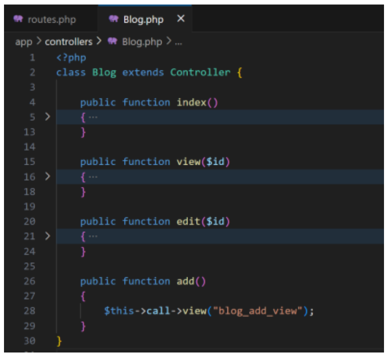
Now, let’s try to pass a data from our **View** file into **Controller**. This time we will used *http request* and *HTML Form*.

Example:

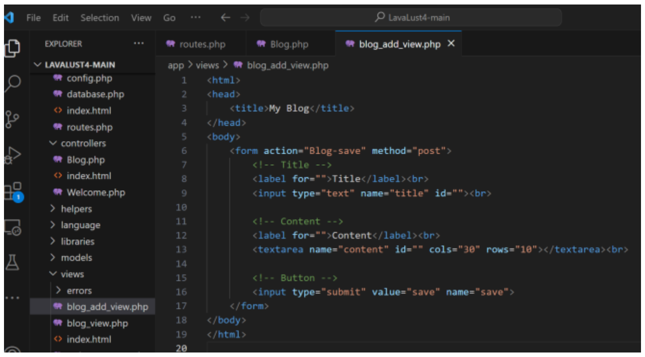
Let’s add first a **add()** method inside the **Blog** controller. Then create a **view** file **blog_add_view.php**, add *HTML Form* with *POST* method, *input tag* for Title and Content and a button.

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:

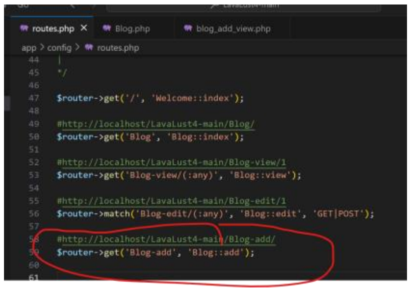


```php
<?php
class Blog extends Controller {

    public function index()
    { ···
    }

    public function view($id)
    { ···
    }

    public function edit($id)
    { ···
    }

    public function add()
    {
        $this->call->view("blog_add_view");
    }
}
```



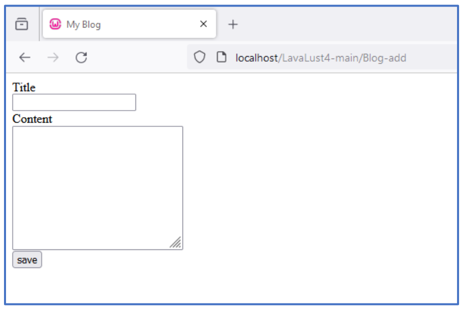
```html
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <form action="Blog-save" method="post">
        <!-- Title -->
        <label for="">Title</label><br>
        <input type="text" name="title" id=""><br>

        <!-- Content -->
        <label for="">Content</label><br>
        <textarea name="content" id="" cols="30" rows="10"></textarea><br>

        <!-- Button -->
        <input type="submit" value="save" name="save">
    </form>
</body>
</html>
```

Create our route!



```php
|
*/

$router->get('/', 'Welcome::index');

#http://localhost/LavaLust4-main/Blog/
$router->get('Blog', 'Blog::index');

#http://localhost/LavaLust4-main/Blog-view/1
$router->get('Blog-view/(:any)', 'Blog::view');

#http://localhost/LavaLust4-main/Blog-edit/1
$router->match('Blog-edit/(:any)', 'Blog::edit', 'GET|POST');

#http://localhost/LavaLust4-main/Blog-add/
$router->get('Blog-add', 'Blog::add');
```

Output:



localhost/LavaLust4-main/Blog-add

Title

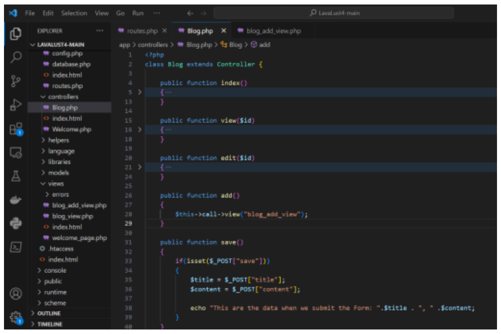
Content

save

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.

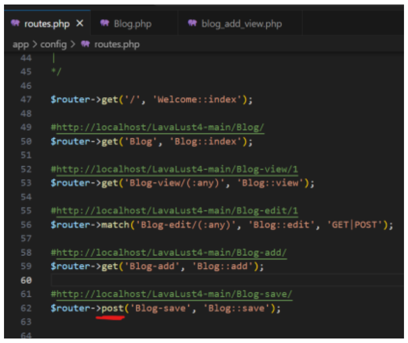
Controller:



```php
<?php
class Blog extends Controller {

    public function index()
    { ···
    }

    public function view($id)
    { ···
    }

    public function edit($id)
    { ···
    }

    public function add()
    {
        $this->call->view("blog_add_view");
    }

    public function save()
    {
        if(isset($_POST["save"]))
        {
            $title = $_POST["title"];
            $content = $_POST["content"];

            echo "This are the data when we submit the Form: ".$title . ", " .$content;
        }
    }
```

Route:

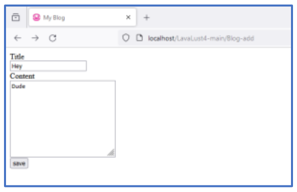
This time, we will used **POST** method for our route.

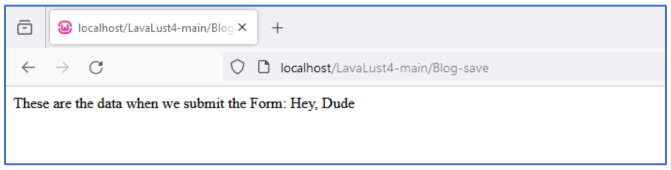


```php
|
*/

$router->get('/', 'Welcome::index');

#http://localhost/LavaLust4-main/Blog/
$router->get('Blog', 'Blog::index');

#http://localhost/LavaLust4-main/Blog-view/1
$router->get('Blog-view/(:any)', 'Blog::view');

#http://localhost/LavaLust4-main/Blog-edit/1
$router->match('Blog-edit/(:any)', 'Blog::edit', 'GET|POST');

#http://localhost/LavaLust4-main/Blog-add/
$router->get('Blog-add', 'Blog::add');

#http://localhost/LavaLust4-main/Blog-save/
$router->post('Blog-save', 'Blog::save');
```

Output:

Let's submit the form



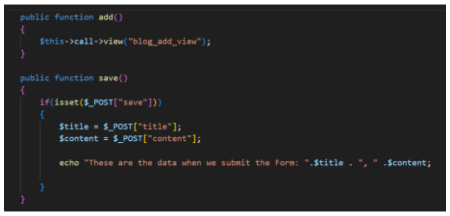

localhost/LavaLust4-main/Blog
localhost/LavaLust4-main/Blog-save

```
These are the data when we submit the Form: Hey, Dude
```

OPTIMIZING OUR CODE

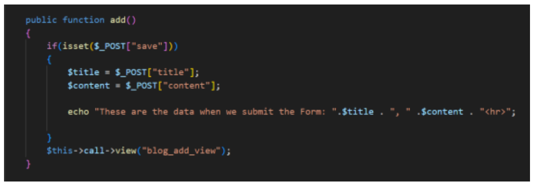
Let's merge our **Blog-add** page and **Blog-save** page for better organization of our code.

From this:



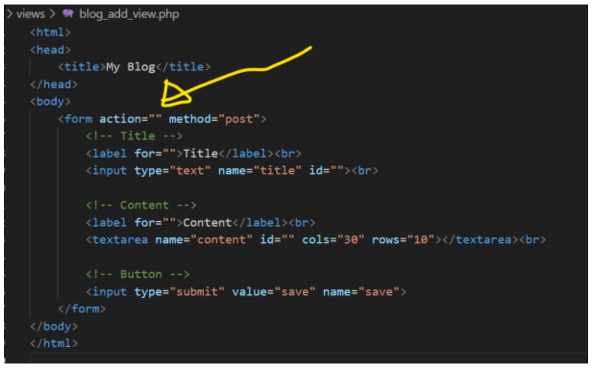
```php
public function add()
{
    $this->call->view("blog_add_view");
}

public function save()
{
    if(isset($_POST["save"]))
    {
        $title = $_POST["title"];
        $content = $_POST["content"];

        echo "These are the data when we submit the Form: ".$title . ", " .$content;

    }
}
```

To this:



```php
public function add()
{
    if(isset($_POST["save"]))
    {
        $title = $_POST["title"];
        $content = $_POST["content"];

        echo "These are the data when we submit the Form: ".$title . ", " .$content . "<hr>";

    }
    $this->call->view("blog_add_view");
}
```

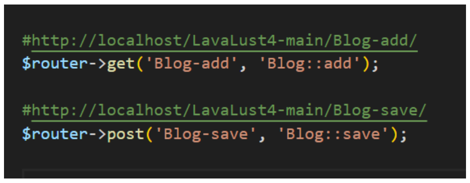
Now, empty the action attribute value, so it will redirect to the same page.



```
> views > blog_add_view.php
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <form action="" method="post">
        <!-- Title -->
        <label for="">Title</label><br>
        <input type="text" name="title" id=""><br>

        <!-- Content -->
        <label for="">Content</label><br>
        <textarea name="content" id="" cols="30" rows="10"></textarea><br>

        <!-- Button -->
        <input type="submit" value="save" name="save">
    </form>
</body>
</html>
```

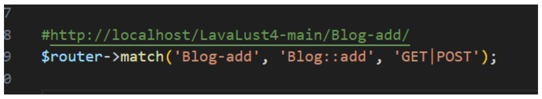
Let's modify the Routes

From this:



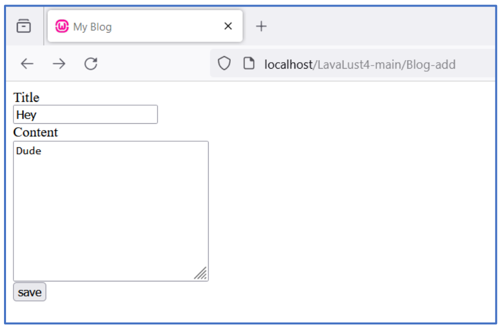
```
#http://localhost/LavaLust4-main/Blog-add/
$router->get('Blog-add', 'Blog::add');

#http://localhost/LavaLust4-main/Blog-save/
$router->post('Blog-save', 'Blog::save');
```

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



```
#http://localhost/LavaLust4-main/Blog-add/
$router->match('Blog-add', 'Blog::add', 'GET|POST');
```

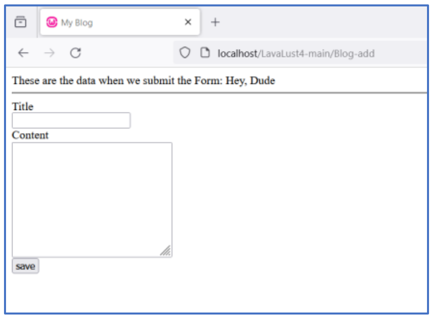
Now let's submit this form:



My Blog
localhost/LavaLust4-main/Blog-add

Title
Hey
Content
Dude
save

And this will be the output:



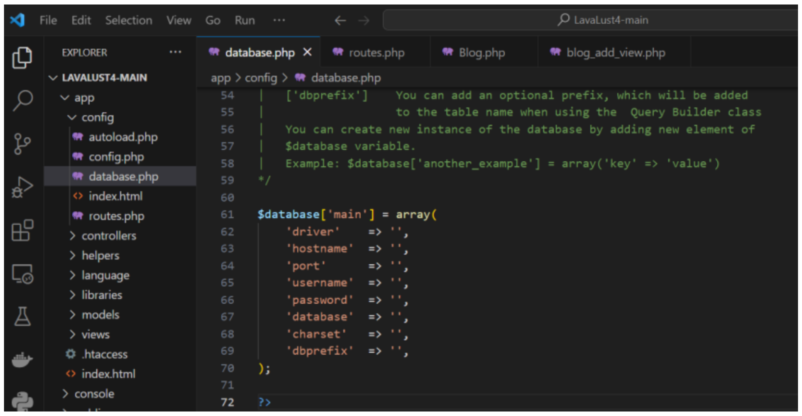
My Blog
localhost/LavaLust4-main/Blog-add

These are the data when we submit the Form: Hey, Dude

Title
Content
save

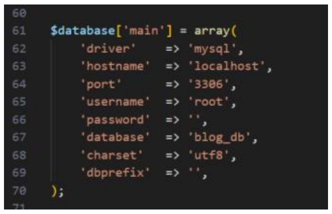
DATABASE CONFIGURATION

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



```
   |  ['dbprefix']    You can add an optional prefix, which will be added
   |                  to the table name when using the  Query Builder class
   |  You can create new instance of the database by adding new element of
   |  $database variable.
   |  Example: $database['another_example'] = array('key' => 'value')
   */

$database['main'] = array(
    'driver'    => '',
    'hostname'  => '',
    'port'      => '',
    'username'  => '',
    'password'  => '',
    'database'  => '',
    'charset'   => '',
    'dbprefix'  => '',
);

?>
```

Let's add our database configurations

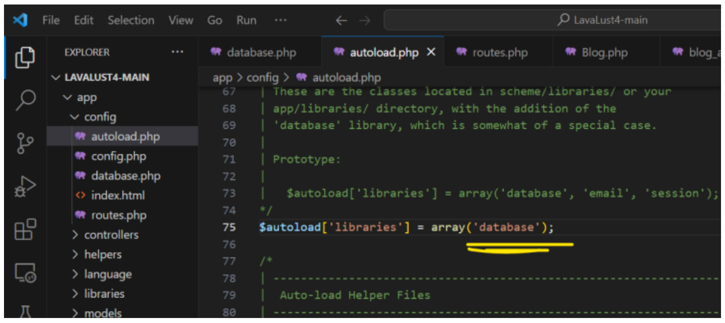


```
$database['main'] = array(
    'driver'    => 'mysql',
    'hostname'  => 'localhost',
    'port'      => '3306',
    'username'  => 'root',
    'password'  => '',
    'database'  => 'blog_db',
    'charset'   => 'utf8',
    'dbprefix'  => '',
);
```

Database Library

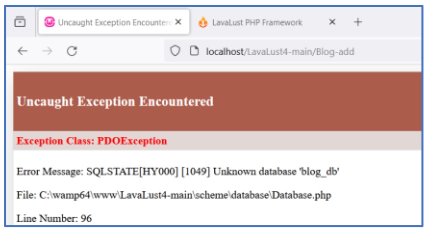
In an MVC Framework, we will use library for using database. To use that library, we need to load that library to our ***autolad.php*** file inside the ***config*** folder. The autoload.php file is used as configuration to load all the needed resources or initialized automatically every time the system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

Let's autoload the library called "database".



```
| These are the classes located in scheme/libraries/ or your
| app/libraries/ directory, with the addition of the
| 'database' library, which is somewhat of a special case.
|
| Prototype:
|
|   $autoload['libraries'] = array('database', 'email', 'session');
*/
$autoload['libraries'] = array('database');

/*
| -------------------------------------------------------------------
|  Auto-load Helper Files
| -------------------------------------------------------------------
```

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



Uncaught Exception Encountered

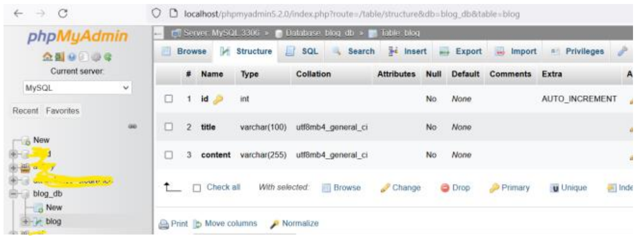
Exception Class: PDOException

Error Message: SQLSTATE[HY000] [1049] Unknown database 'blog_db'

File: C:\wamp64\www\LavaLust4-main\scheme\database\Database.php

Line Number: 96

Create the database.

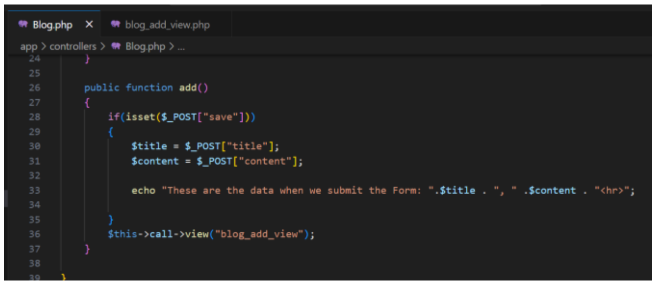


```
localhost/phpmyadmin5.2.0/index.php?route=/table/structure&db=blog_db&table=blog
Server: MySQL:3306 » Database: blog_db » Table: blog
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int			No	None		AUTO_INCREMENT
2	title	varchar(100)	utf8mb4_general_ci		No	None		
3	content	varchar(255)	utf8mb4_general_ci		No	None		

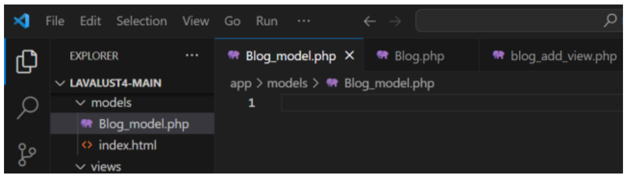
MODELS & QUERY BUILDER

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



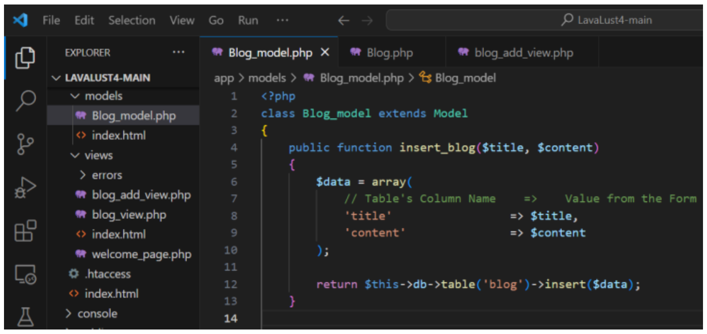
```php
    }

    public function add()
    {
        if(isset($_POST["save"]))
        {
            $title = $_POST["title"];
            $content = $_POST["content"];

            echo "These are the data when we submit the Form: ".$title . ", " .$content . "<hr>";

        }
        $this->call->view("blog_add_view");
    }

}
```

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*).



```php
<?php
class Blog_model extends Model
{
    public function insert_blog($title, $content)
    {
        $data = array(
            // Table's Column Name    =>    Value from the Form
            'title'                 => $title,
            'content'               => $content
        );

        return $this->db->table('blog')->insert($data);
    }
```

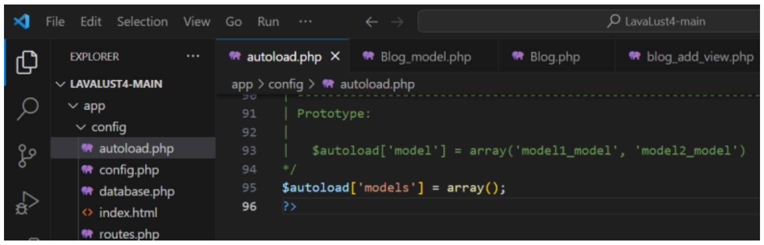
NOTE: *Visit https://lavalust.netlify.app/#item-6-3 about query builder.*

Loading the Model

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

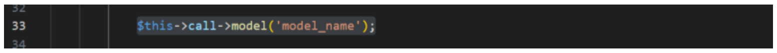
There are two ways of calling/loading/using the Model:

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



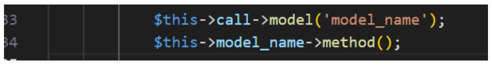
```
| Prototype:
|
|   $autoload['model'] = array('model1_model', 'model2_model')
*/
$autoload['models'] = array();
?>
```

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



```
$this->call->model('model_name');
```

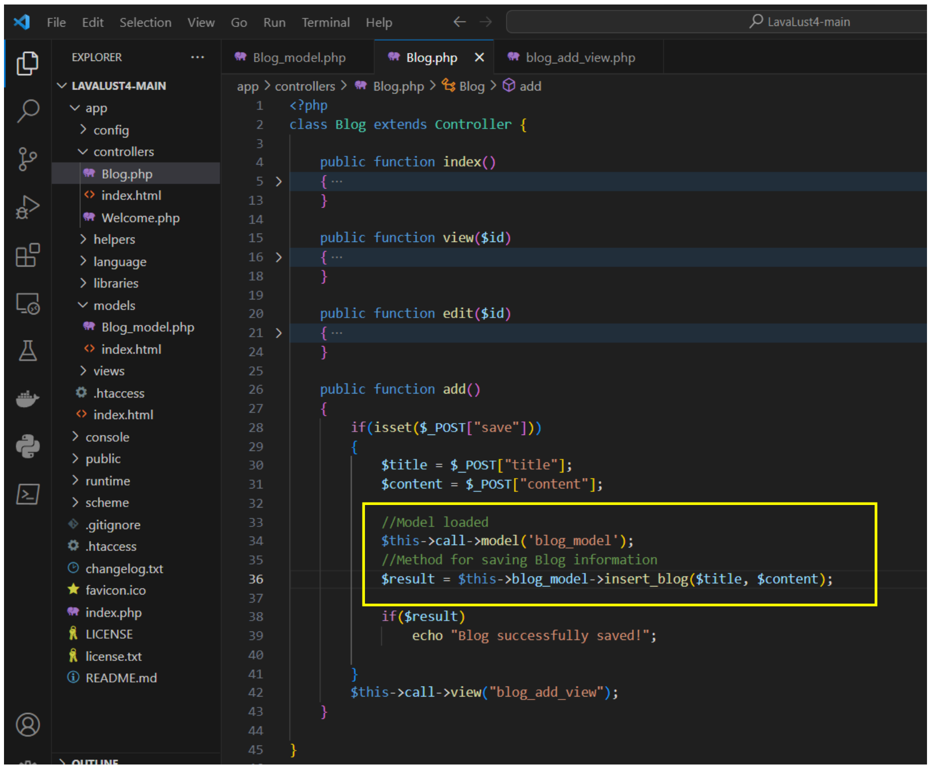
Once loaded, you will access your model methods using an object with the same name as your class:



```
$this->call->model('model_name');
$this->model_name->method();
```

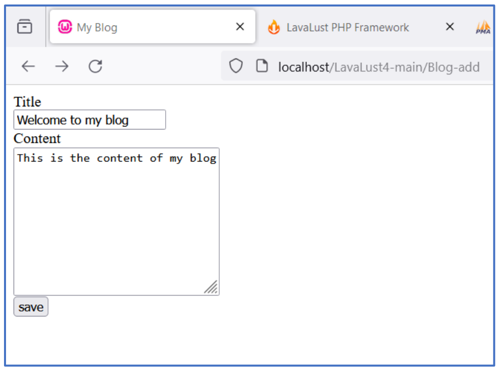
INSERT INTO (CRUD)

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



```
<?php
class Blog extends Controller {

    public function index()
    {…
    }

    public function view($id)
    {…
    }

    public function edit($id)
    {…
    }

    public function add()
    {
        if(isset($_POST["save"]))
        {
            $title = $_POST["title"];
            $content = $_POST["content"];

            //Model loaded
            $this->call->model('blog_model');
            //Method for saving Blog information
            $result = $this->blog_model->insert_blog($title, $content);

            if($result)
                echo "Blog successfully saved!";

        }
        $this->call->view("blog_add_view");
    }

}
```

Let’s try and submit some information:



Title

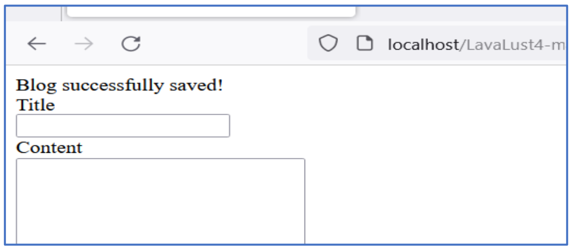
Welcome to my blog

Content

```
This is the content of my blog
```

save

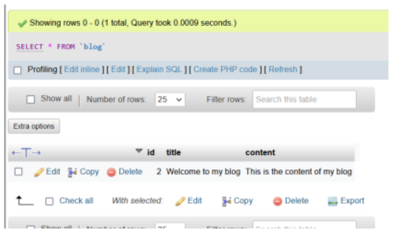
Output:



Blog successfully saved!

Title

Content

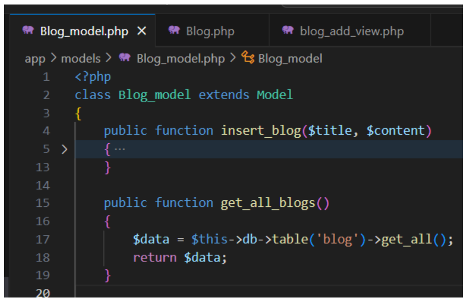


```
Showing rows 0 - 0 (1 total, Query took 0.0009 seconds.)

SELECT * FROM `blog`
```

id	title	content
2	Welcome to my blog	This is the content of my blog

SELECT STATEMENT (CRUD)

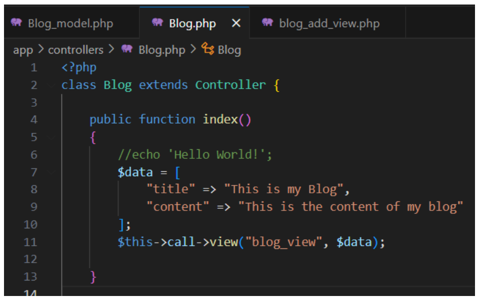
Model (select * from blog)



```php
<?php
class Blog_model extends Model
{
    public function insert_blog($title, $content)
    { ···
    }

    public function get_all_blogs()
    {
        $data = $this->db->table('blog')->get_all();
        return $data;
    }
```

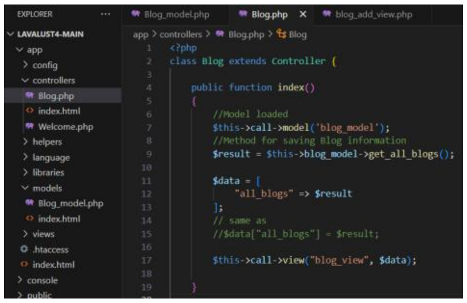
Controller (Blog/index)

From this:



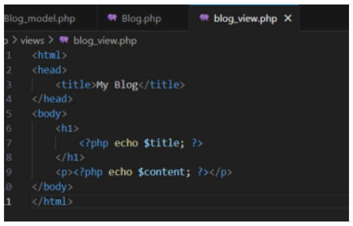
```php
<?php
class Blog extends Controller {

    public function index()
    {
        //echo 'Hello World!';
        $data = [
            "title" => "This is my Blog",
            "content" => "This is the content of my blog"
        ];
        $this->call->view("blog_view", $data);

    }
```

To this:

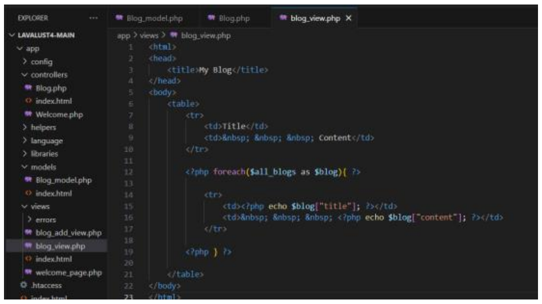


```php
<?php
class Blog extends Controller {

    public function index()
    {
        //Model loaded
        $this->call->model('blog_model');
        //Method for saving Blog information
        $result = $this->blog_model->get_all_blogs();

        $data = [
            "all_blogs" => $result
        ];
        // same as
        //$data["all_blogs"] = $result;

        $this->call->view("blog_view", $data);

    }
```

Views (Blog_view.php)

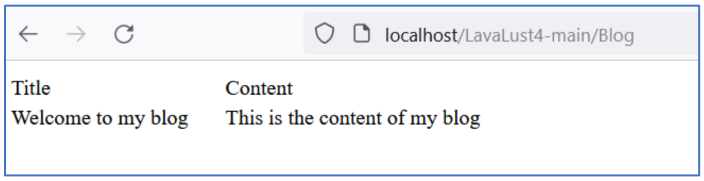
From this: 

```
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <h1>
        <?php echo $title; ?>
    </h1>
    <p><?php echo $content; ?></p>
</body>
</html>
```

To this: 

```
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <table>
        <tr>
            <td>Title</td>
            <td>&nbsp; &nbsp; &nbsp; Content</td>
        </tr>

        <?php foreach($all_blogs as $blog){ ?>

            <tr>
                <td><?php echo $blog["title"]; ?></td>
                <td>&nbsp; &nbsp; &nbsp; <?php echo $blog["content"]; ?></td>
            </tr>

        <?php } ?>

    </table>
</body>
</html>
```

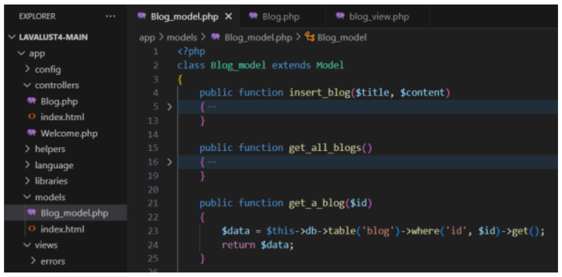
Output:



```
localhost/LavaLust4-main/Blog

Title               Content
Welcome to my blog  This is the content of my blog
```

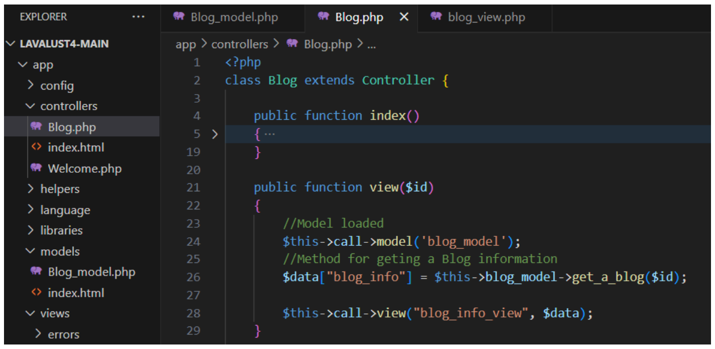
SELECT only one (1) row (CRUD)

Model (select * from blog where id=?)



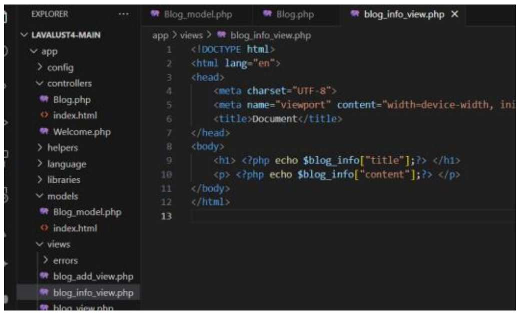
```
<?php
class Blog_model extends Model
{
    public function insert_blog($title, $content)
    {…
    }

    public function get_all_blogs()
    {…
    }

    public function get_a_blog($id)
    {
        $data = $this->db->table('blog')->where('id', $id)->get();
        return $data;
    }
```

Controller (Blog/view/$id)



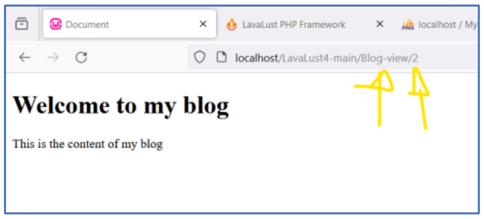
```
<?php
class Blog extends Controller {

    public function index()
    {…
    }

    public function view($id)
    {
        //Model loaded
        $this->call->model('blog_model');
        //Method for geting a Blog information
        $data["blog_info"] = $this->blog_model->get_a_blog($id);

        $this->call->view("blog_info_view", $data);
    }
```

View (blog_info_view.php)



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, ini
    <title>Document</title>
</head>
<body>
    <h1> <?php echo $blog_info["title"];?> </h1>
    <p> <?php echo $blog_info["content"];?> </p>
</body>
</html>
```

Output:



localhost/LavaLust4-main/Blog-view/2

Welcome to my blog

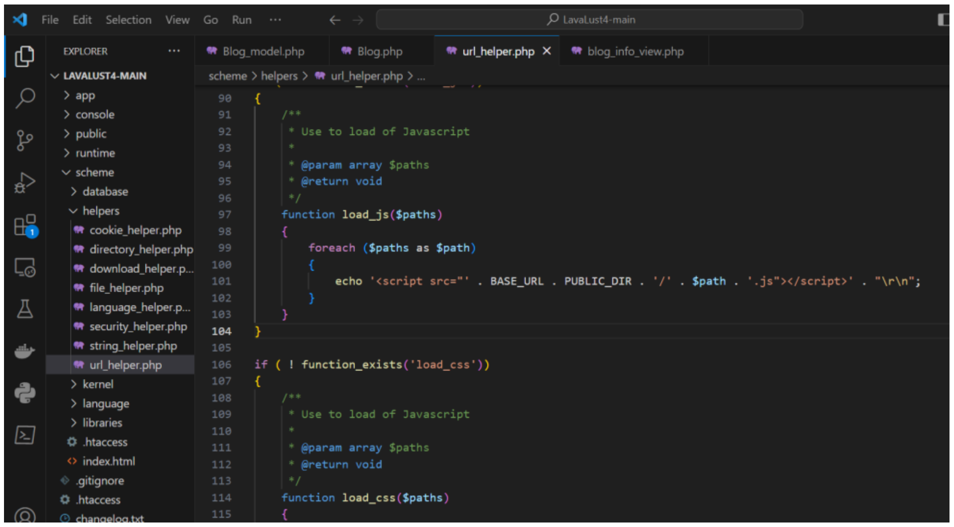
This is the content of my blog

*<span style="color:red">**YOUR TURN:** Apply the idea of the **Blog-view** & **Blog-add** page to create Edit/Update (CRUD).</span>*

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.



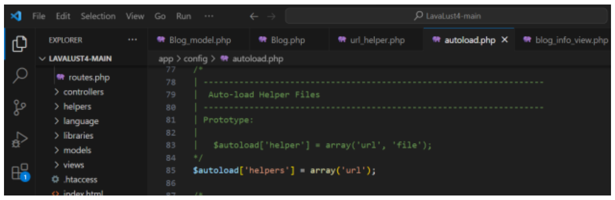
```
{
    /**
     * Use to load of Javascript
     *
     * @param array $paths
     * @return void
     */
    function load_js($paths)
    {
        foreach ($paths as $path)
        {
            echo '<script src="' . BASE_URL . PUBLIC_DIR . '/' . $path . '.js"></script>' . "\r\n";
        }
    }
}

if ( ! function_exists('load_css'))
{
    /**
     * Use to load of Javascript
     *
     * @param array $paths
     * @return void
     */
    function load_css($paths)
    {
```

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

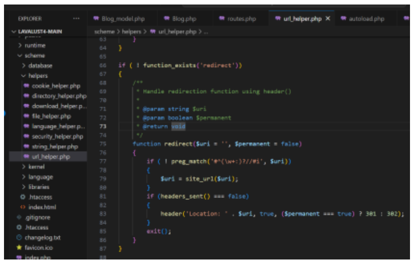
Load the helper in **autoload.php**

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



```php
/*
| -------------------------------------------------------------------
|  Auto-load Helper Files
| -------------------------------------------------------------------
| Prototype:
|
|   $autoload['helper'] = array('url', 'file');
*/
$autoload['helpers'] = array('url');
```

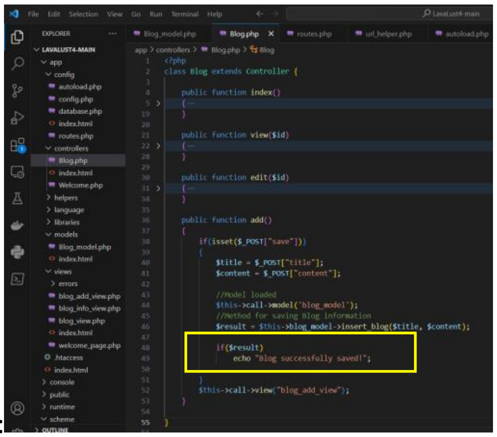
Now we can use the **redirect()** method from **url_helper** to redirect into another page.



```php
    }
}

if ( ! function_exists('redirect'))
{
    /**
     * Handle redirection function using header()
     *
     * @param string $uri
     * @param boolean $permanent
     * @return void
     */
    function redirect($uri = '', $permanent = false)
    {
        if ( ! preg_match('#^(\w+:)?//#i', $uri))
        {
            $uri = site_url($uri);
        }
        if (headers_sent() === false)
        {
            header('Location: ' . $uri, true, ($permanent === true) ? 301 : 302);
        }
        exit();
    }
}
```

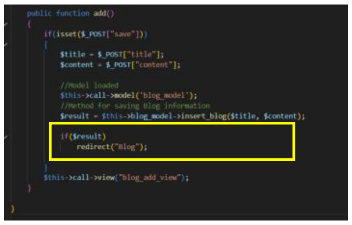
Let's try in **Blog** Controller:

Before:



```php
<?php
class Blog extends Controller {

    public function index()
    {…
    }

    public function view($id)
    {…
    }

    public function edit($id)
    {…
    }

    public function add()
    {
        if(isset($_POST["save"]))
        {
            $title = $_POST["title"];
            $content = $_POST["content"];

            //Model loaded
            $this->call->model('blog_model');
            //Method for saving Blog information
            $result = $this->blog_model->insert_blog($title, $content);

            if($result)
                echo "Blog successfully saved!";

        }
        $this->call->view("blog_add_view");
    }

}
```

After:



```php
    public function add()
    {
        if(isset($_POST["save"]))
        {
            $title = $_POST["title"];
            $content = $_POST["content"];

            //Model loaded
            $this->call->model('blog_model');
            //Method for saving Blog information
            $result = $this->blog_model->insert_blog($title, $content);

            if($result)
                redirect("Blog");

        }
        $this->call->view("blog_add_view");
    }

}
```

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.

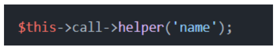
The following items can be loaded automatically:

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

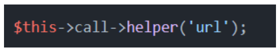
To autoload resources, open the **app/config/autoload.php**

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

Helper:

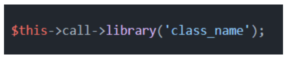


```
$this->call->helper('name');
```

Example: 

```
$this->call->helper('url');
```

Library:

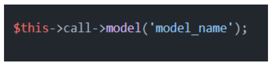


```
$this->call->library('class_name');
```

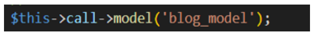
Example:

```
$this->call->library('Form_validation');
```

Model:



```
$this->call->model('model_name');
```

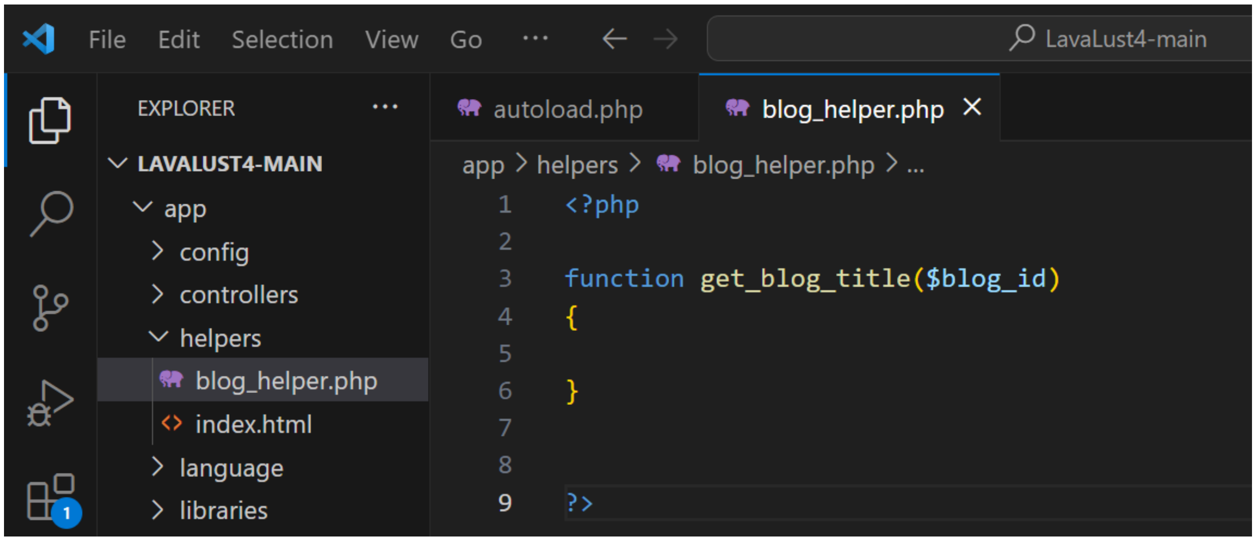
Example: 

```
$this->call->model('blog_model');
```

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.

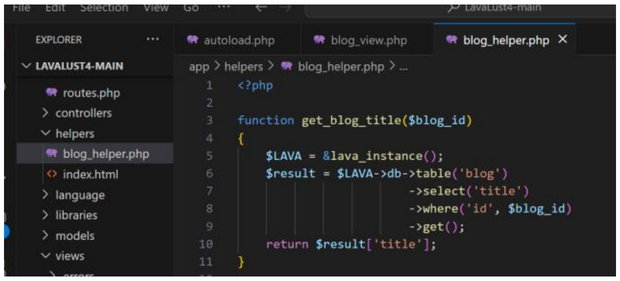
Example: blog_helper



```php
<?php

function get_blog_title($blog_id)
{

}


?>
```

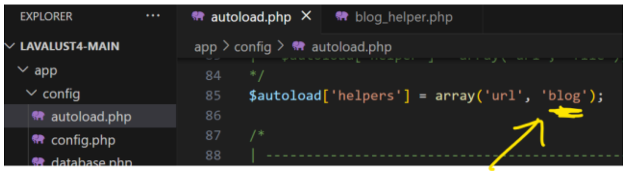
Now we will use the **get_blog_title()** function to get from the database the title of a specified **blog_id**.

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:



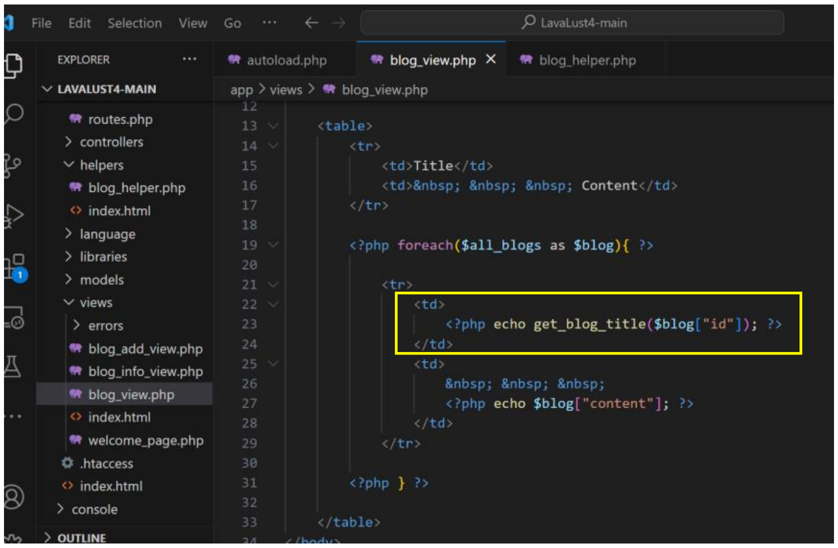
```php
<?php

function get_blog_title($blog_id)
{
    $LAVA = &lava_instance();
    $result = $LAVA->db->table('blog')
                        ->select('title')
                        ->where('id', $blog_id)
                        ->get();
    return $result['title'];
}
```

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



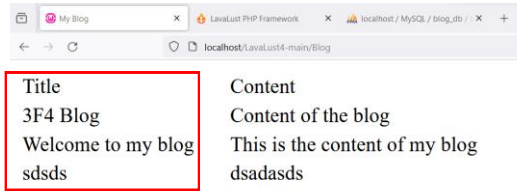
```php
*/
$autoload['helpers'] = array('url', 'blog');

/*
| ----------------------------------------
```

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.



```
    <table>
        <tr>
            <td>Title</td>
            <td>&nbsp; &nbsp; &nbsp; Content</td>
        </tr>

        <?php foreach($all_blogs as $blog){ ?>

            <tr>
                <td>
                    <?php echo get_blog_title($blog["id"]); ?>
                </td>
                <td>
                    &nbsp; &nbsp; &nbsp;
                    <?php echo $blog["content"]; ?>
                </td>
            </tr>

        <?php } ?>

    </table>
```

Output:

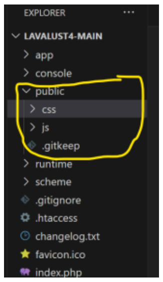


```
Title              Content
3F4 Blog           Content of the blog
Welcome to my blog This is the content of my blog
sdsds              dsadasds
```

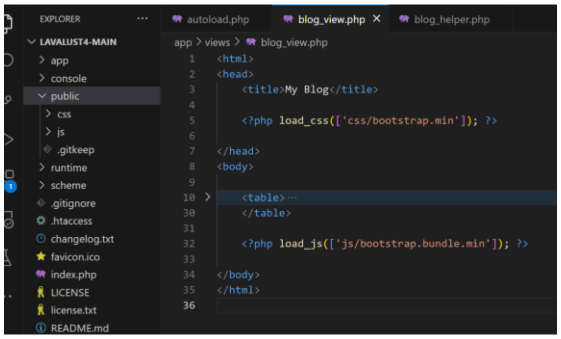
Links, and Public / Assets (css, js, images, etc) form Views

Public / Assets

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



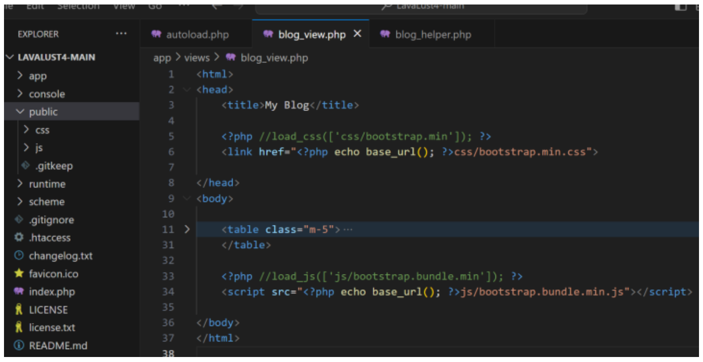
Now, use the load_css() and load_js() function from url_helper.



```
<html>
<head>
    <title>My Blog</title>

    <?php load_css(['css/bootstrap.min']); ?>

</head>
<body>

    <table>···
    </table>

    <?php load_js(['js/bootstrap.bundle.min']); ?>

</body>
</html>
```

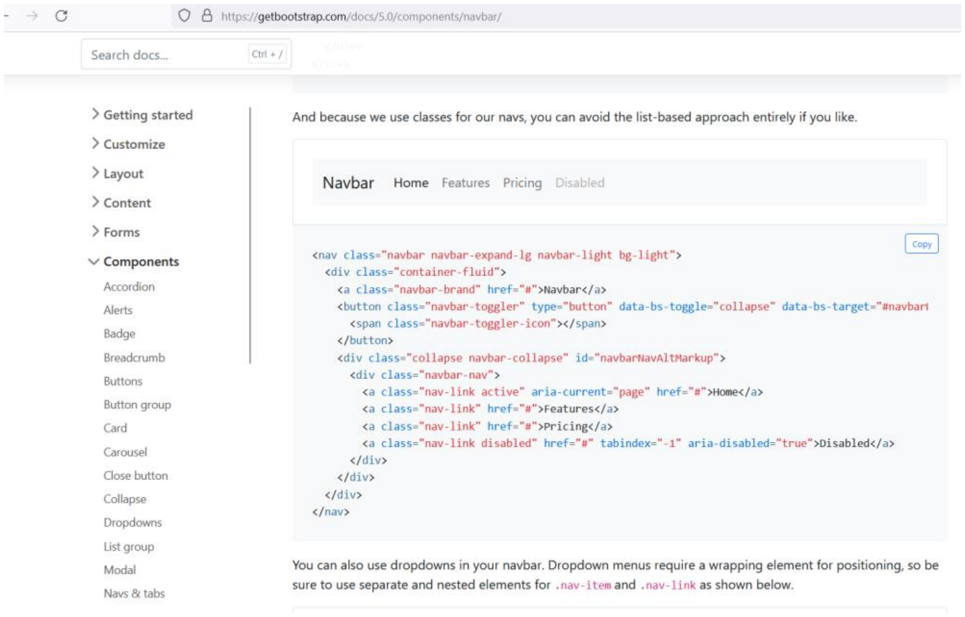
Note: Other way of accessing public assets is using the **base_url()**.

Example:



```
<html>
<head>
    <title>My Blog</title>

    <?php //load_css(['css/bootstrap.min']); ?>
    <link href="<?php echo base_url(); ?>css/bootstrap.min.css">

</head>
<body>

    <table class="m-5"> ···
    </table>

    <?php //load_js(['js/bootstrap.bundle.min']); ?>
    <script src="<?php echo base_url(); ?>js/bootstrap.bundle.min.js"></script>

</body>
</html>
```

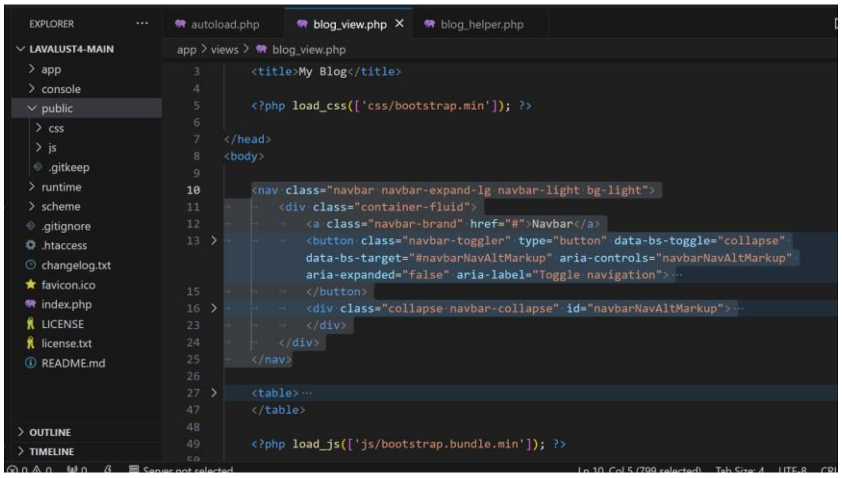
OK! Let's copy some code from Bootstrap5 documentation.



And because we use classes for our navs, you can avoid the list-based approach entirely if you like.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarl
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
      <div class="navbar-nav">
        <a class="nav-link active" aria-current="page" href="#">Home</a>
        <a class="nav-link" href="#">Features</a>
        <a class="nav-link" href="#">Pricing</a>
        <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
      </div>
    </div>
  </div>
</nav>
```

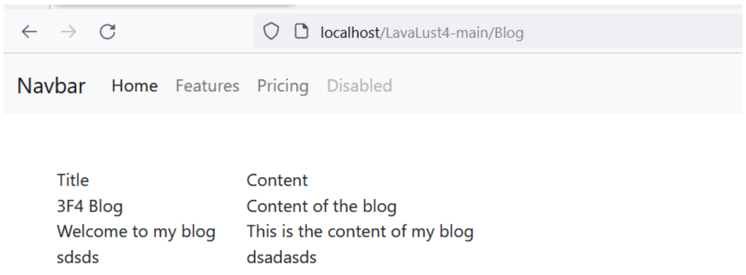
You can also use dropdowns in your navbar. Dropdown menus require a wrapping element for positioning, so be sure to use separate and nested elements for `.nav-item` and `.nav-link` as shown below.



```
    <title>My Blog</title>

    <?php load_css(['css/bootstrap.min']); ?>

</head>
<body>

    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Navbar</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
            data-bs-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup"
            aria-expanded="false" aria-label="Toggle navigation"> ···
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup"> ···
            </div>
        </div>
    </nav>

    <table> ···
    </table>

    <?php load_js(['js/bootstrap.bundle.min']); ?>
```

Output:

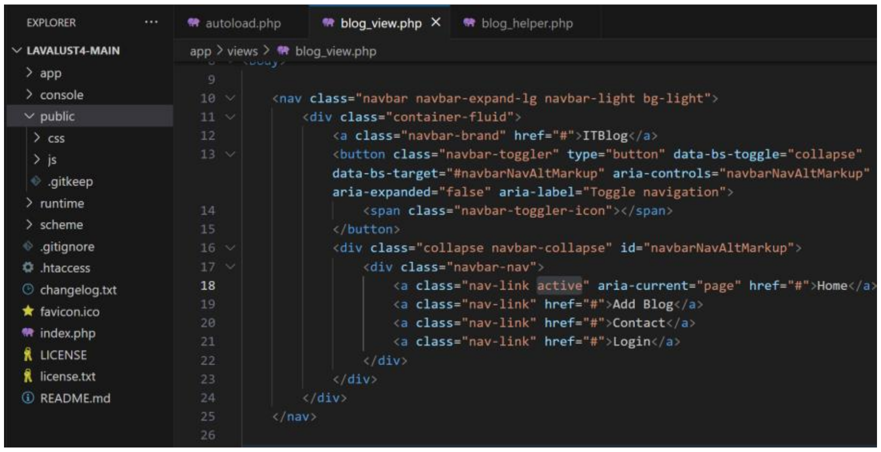
Now, we have a navigation bar. Let’s customized it!



```
localhost/LavaLust4-main/Blog

Navbar   Home   Features   Pricing   Disabled

Title                 Content
3F4 Blog              Content of the blog
Welcome to my blog    This is the content of my blog
sdsds                 dsadasds
```

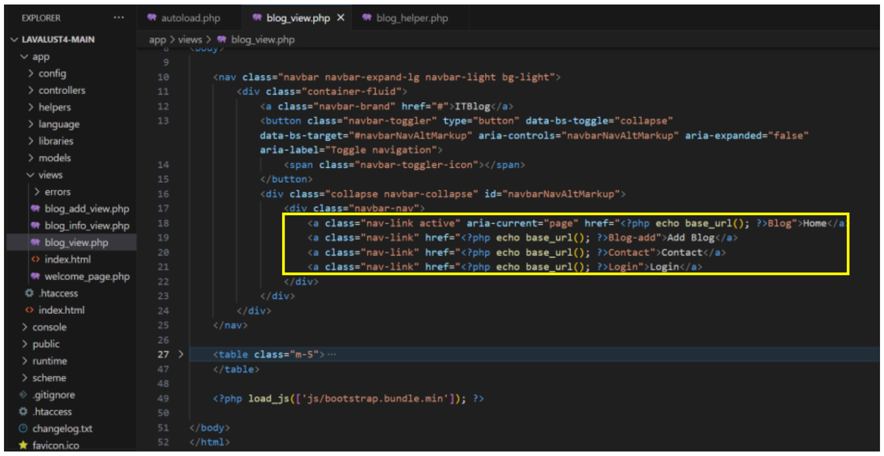
Links

Navigation Bar



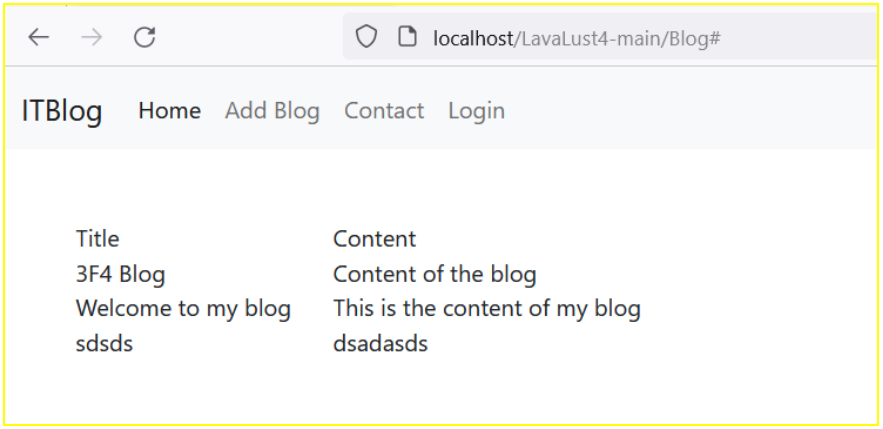
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">ITBlog</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
        data-bs-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup"
        aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
            <div class="navbar-nav">
                <a class="nav-link active" aria-current="page" href="#">Home</a>
                <a class="nav-link" href="#">Add Blog</a>
                <a class="nav-link" href="#">Contact</a>
                <a class="nav-link" href="#">Login</a>
            </div>
        </div>
    </div>
</nav>
```

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



```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">ITBlog</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
        data-bs-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false"
        aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
            <div class="navbar-nav">
                <a class="nav-link active" aria-current="page" href="<?php echo base_url(); ?>Blog">Home</a
                <a class="nav-link" href="<?php echo base_url(); ?>Blog-add">Add Blog</a>
                <a class="nav-link" href="<?php echo base_url(); ?>Contact">Contact</a>
                <a class="nav-link" href="<?php echo base_url(); ?>Login">Login</a>
            </div>
        </div>
    </div>
</nav>

<table class="m-5">···
</table>

<?php load_js(['js/bootstrap.bundle.min']); ?>

</body>
</html>
```

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

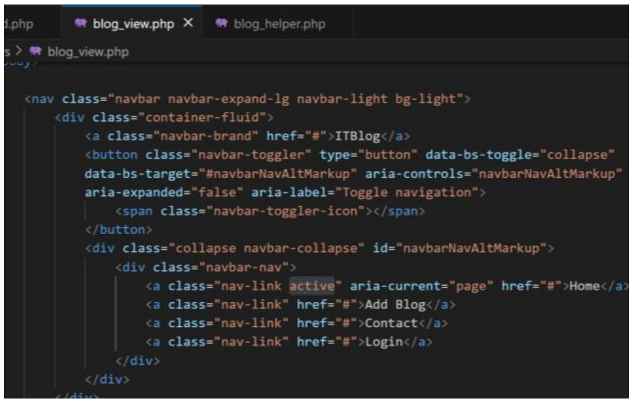

localhost/LavaLust4-main/Blog#
ITBlog Home Add Blog Contact Login

Title	Content
3F4 Blog	Content of the blog
Welcome to my blog	This is the content of my blog
sdsds	dsadasds

Now, copy the navigation bar to all view file.

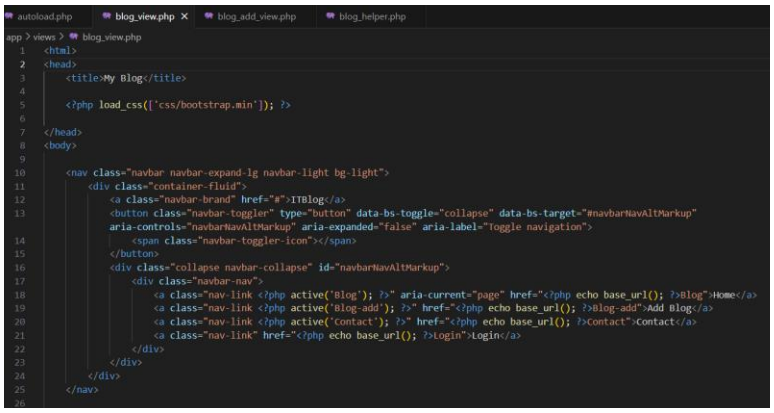
Active

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



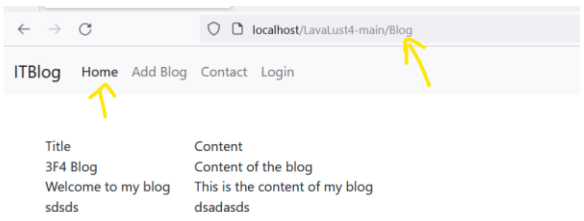
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">ITBlog</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
        data-bs-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup"
        aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
            <div class="navbar-nav">
                <a class="nav-link active" aria-current="page" href="#">Home</a>
                <a class="nav-link" href="#">Add Blog</a>
                <a class="nav-link" href="#">Contact</a>
                <a class="nav-link" href="#">Login</a>
            </div>
        </div>
```

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

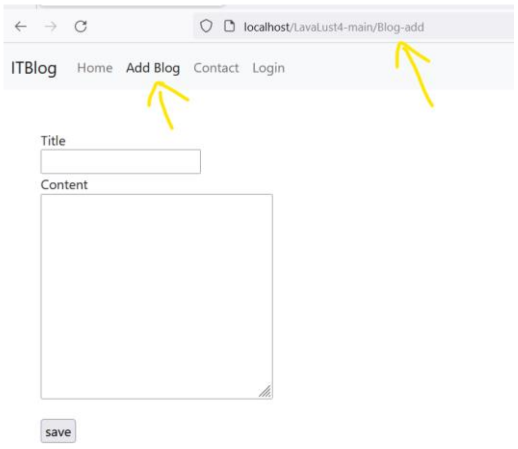


```
<html>
<head>
    <title>My Blog</title>

    <?php load_css(['css/bootstrap.min']); ?>

</head>
<body>

    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">ITBlog</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">
                    <a class="nav-link <?php active('Blog'); ?>" aria-current="page" href="<?php echo base_url(); ?>Blog">Home</a>
                    <a class="nav-link <?php active('Blog-add'); ?>" href="<?php echo base_url(); ?>Blog-add">Add Blog</a>
                    <a class="nav-link <?php active('Contact'); ?>" href="<?php echo base_url(); ?>Contact">Contact</a>
                    <a class="nav-link" href="<?php echo base_url(); ?>Login">Login</a>
                </div>
            </div>
        </div>
    </nav>

```

Output:



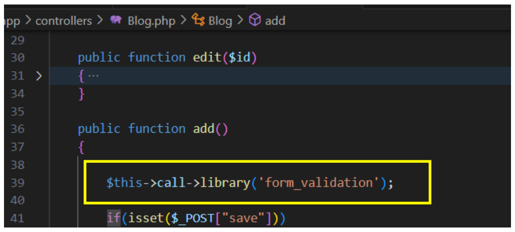
```
Title               Content
3F4 Blog            Content of the blog
Welcome to my blog  This is the content of my blog
sdsds               dsadasds
```



Library

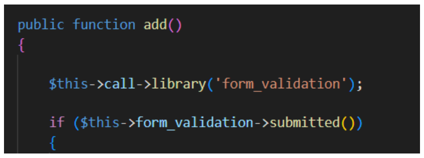
Form Validation Library

Load the form_validation library. This time, we load it manually inside controller.



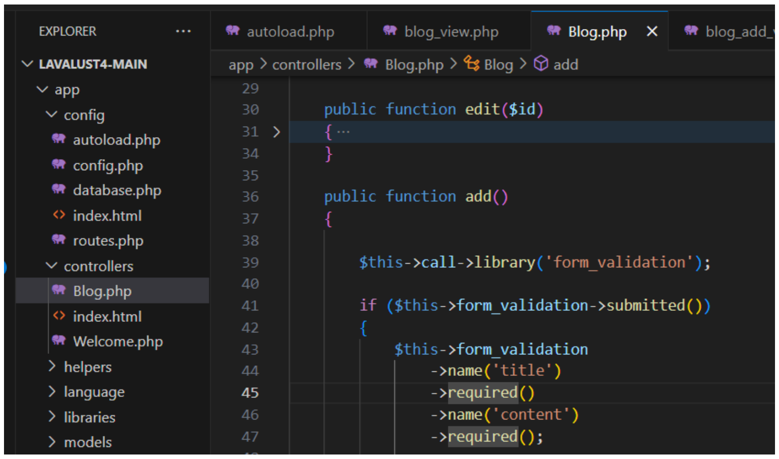
```
    public function edit($id)
    {…
    }

    public function add()
    {

        $this->call->library('form_validation');

        if(isset($_POST["save"]))
```

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



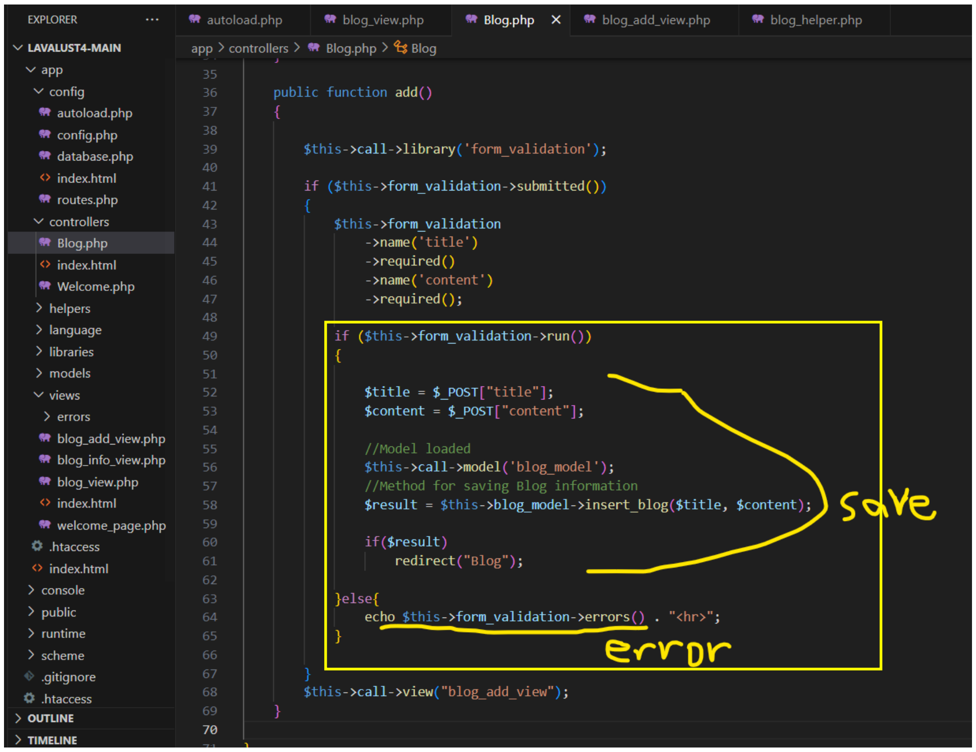
```
public function add()
{

    $this->call->library('form_validation');

    if ($this->form_validation->submitted())
    {
```

Let's validate our inputs (title & content)



```
    public function edit($id)
    {…
    }

    public function add()
    {

        $this->call->library('form_validation');

        if ($this->form_validation->submitted())
        {
            $this->form_validation
                ->name('title')
                ->required()
                ->name('content')
                ->required();
```

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.



```

    public function add()
    {

        $this->call->library('form_validation');

        if ($this->form_validation->submitted())
        {
            $this->form_validation
                ->name('title')
                ->required()
                ->name('content')
                ->required();

            if ($this->form_validation->run())
            {

                $title = $_POST["title"];
                $content = $_POST["content"];

                //Model loaded
                $this->call->model('blog_model');
                //Method for saving Blog information
                $result = $this->blog_model->insert_blog($title, $content);

                if($result)
                    redirect("Blog");

            }else{
                echo $this->form_validation->errors() . "<hr>";
            }

        }
        $this->call->view("blog_add_view");
    }

```

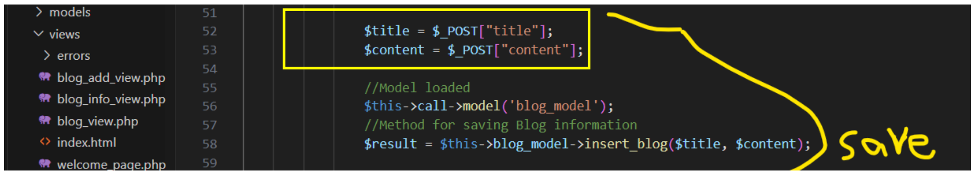
If the validation returns an error, it will display what kind of input error or else if no error, the data will save in database.

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

(IO) Input and Output Class

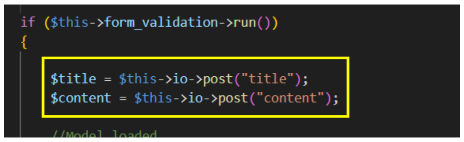
This class is initialized automatically by the system so there is no need to do it manually.

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



```
$title = $_POST["title"];
$content = $_POST["content"];

//Model loaded
$this->call->model('blog_model');
//Method for saving Blog information
$result = $this->blog_model->insert_blog($title, $content);
```

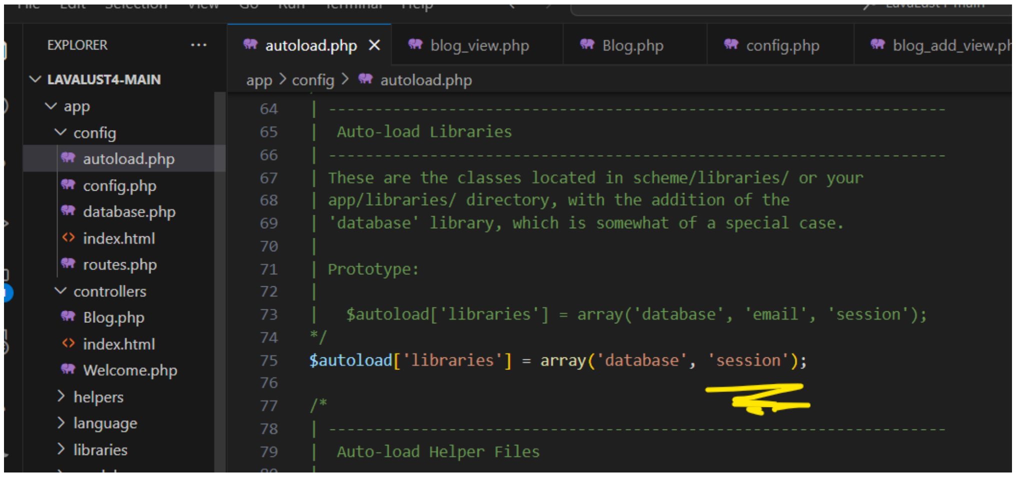
Now, let’s use the methods from the **io** class.



```
if ($this->form_validation->run())
{
    $title = $this->io->post("title");
    $content = $this->io->post("content");

    //Model loaded
```

Session Library

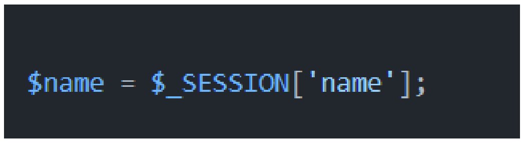
To use the session, load it first in ***autoload.php***



```
|--------------------------------------------------------------------------
|  Auto-load Libraries
|--------------------------------------------------------------------------
| These are the classes located in scheme/libraries/ or your
| app/libraries/ directory, with the addition of the
| 'database' library, which is somewhat of a special case.
|
| Prototype:
|
|   $autoload['libraries'] = array('database', 'email', 'session');
*/
$autoload['libraries'] = array('database', 'session');

/*
|--------------------------------------------------------------------------
|  Auto-load Helper Files
```

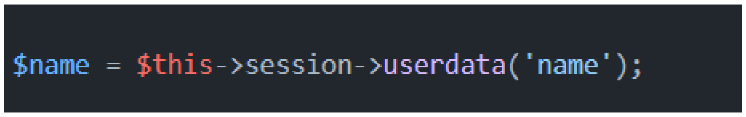
Retrieving Session Data

$_SESSION superglobal



```
$name = $_SESSION['name'];
```

Using **Session** library



```
$name = $this->session->userdata('name');
```

Adding Session Data

$_SESSION superglobal

```
$_SESSION["username"] = "juan23";
$_SESSION["user_role"] = "admin";
```

Using **Session** library

```
$this->session->set_userdata(array(
    "username" => "juan23",
    "user_role" => "admin"
));
```

Removing Session Data

$_SESSION superglobal

Just as with any other variable, unsetting a value in `$_SESSION` can be done through `unset()`:

```
unset($_SESSION['some_name']);

// or multiple values:

unset(
        $_SESSION['some_name'],
        $_SESSION['another_name']
);
```

Using **Session** Library

```
$array_items = array('username', 'email');

$this->session->unset_userdata($array_items);
```

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**