LavaLust

LavaLust is an application development framework, which can be used to develop websites, using PHP. It is an Open-Source framework. It has a very rich set of functionalities, which will increase the speed of website development work.

If you know PHP well, then LavaLust will make your task easier. It has a very rich set of libraries and helpers. By using LavaLust, you will save a lot of time, if you are developing a website from scratch. Not only that, a website built in LavaLust is secure too, as it has the ability to prevent various attacks that take place through websites.

It is very easy to install LavaLust. Just follow the steps given below −

Step-1 − Download the LavaLust from the link LavaLust

We can also go with GitHub and get all of the latest scripts.

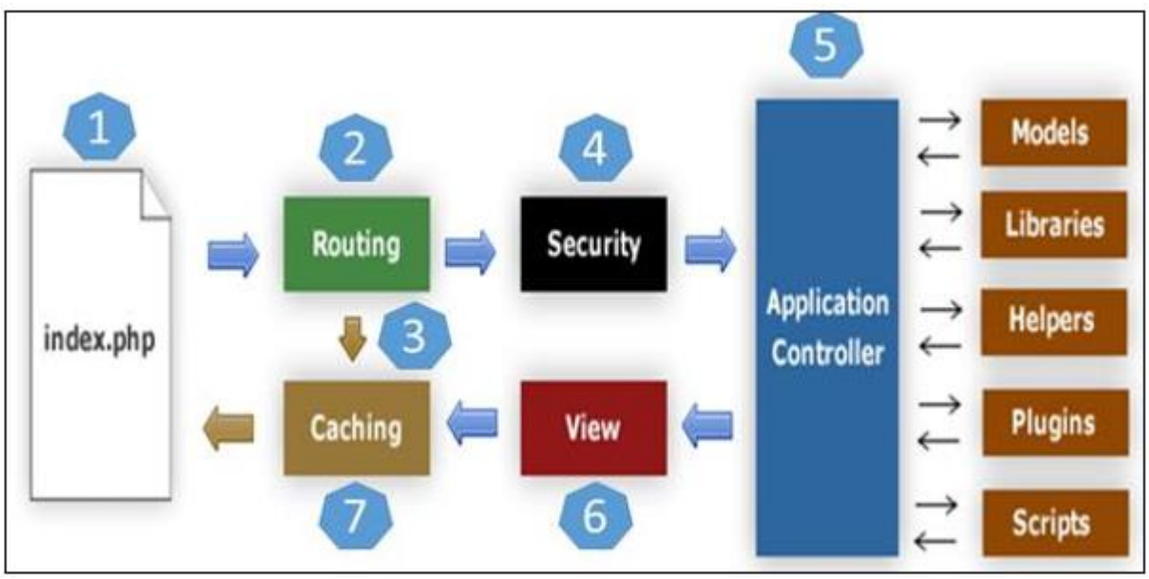
Step-2 − Unzip the folder.

Step-3 − Upload all files and folders to your server

Step-4 − After uploading all the files to your server, visit the URL of your server, e.g., www.domain-name.com.

On visiting the URL, you will see the following screen –

The architecture of LavaLust application is shown below.



- As shown in the figure, whenever a request comes to LavaLust, it will first go to **index.php** page.
- In the second step, **Routing** will decide whether to pass this request to step-3 for caching or to pass this request to step-4 for security check.
- If the requested page is already in **Caching**, then **Routing** will pass the request to step-3 and the response will go back to the user.
- If the requested page does not exist in **Caching**, then **Routing** will pass the requested page to step-4 for **Security** checks.
- Before passing the request to **Application Controller**, the **Security** of the submitted data is checked. After the **Security** check, the **Application Controller** loads necessary **Models, Libraries, Helpers** and **Scripts** and pass it on to **View**.
- The **View** will render the page with available data and pass it on for **Caching**. As the requested page was not cached before so this time it will be cached in **Caching**, to process this page quickly for future requests.

App

App or the Application folder contains all the code of your application that you are building. This is the folder where you will develop your project. The Application folder contains several other folders, which are explained below −

- **Cache** − This folder contains all the cached pages of your application. These cached pages will increase the overall speed of accessing the pages.
- **Config** − This folder contains various files to configure the application. With the help of **config.php** file, user can configure the application. Using **database.php** file, user can configure the database of the application.
- **Controllers** − This folder holds the controllers of your application. It is the basic part of your application.
- **Helpers** − In this folder, you can put helper class of your application.
- **Language** − This folder contains language related files.
- **Libraries** − This folder contains files of the libraries developed for your application.
- **Models** − The database login will be placed in this folder.
- **Views** − Application's HTML files will be placed in this folder.

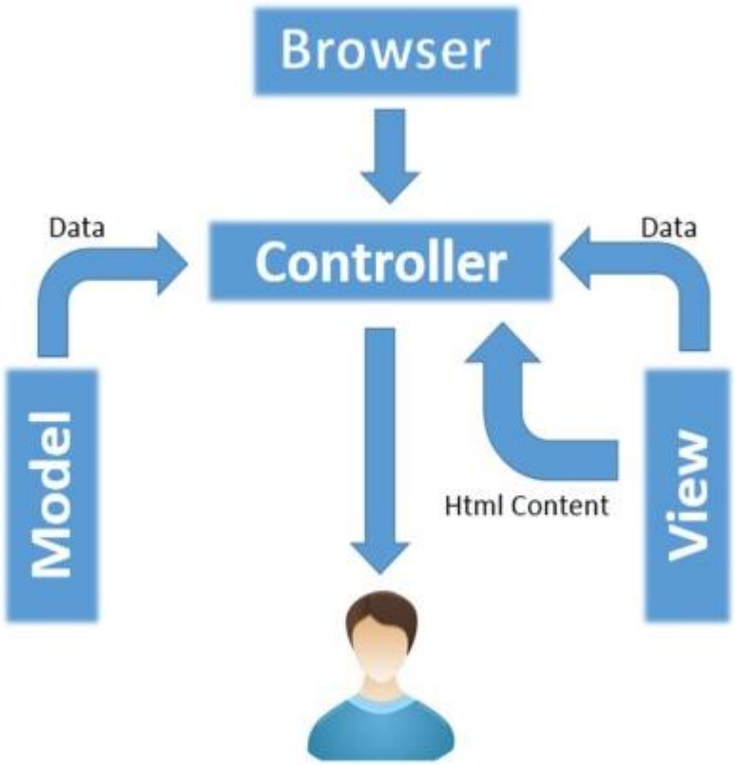
Scheme

This folder contains LavaLust kernel files, libraries, helpers and other files, which help make the coding easy. These libraries and helpers are loaded and used in web app development.

This folder contains all the LavaLust code of consequence, organized into various folders −

- **Kernel** − This folder contains LavaLust's core class. Do not modify anything here. All of your work will take place in the app or application folder.
- **Database** − The database folder contains the database class that contains SQL query builder functions.
- **Helpers** − The helpers folder contains standard LavaLust helpers (such as string, cookie, and URL helpers).
- **Language** − The language folder contains language files.
- **Libraries** − The libraries folder contains standard LavaLust libraries (to help you with e-mail, file downloads, and more). You can create your own libraries or extend (and even replace) standard ones, but those will be saved in the **app/libraries** directory to keep them separate from the standard LavaLust libraries saved in this particular folder.

LavaLust is based on the **Model-View-Controller (MVC) development pattern**. MVC is a software approach that separates application logic from presentation. In practice, it permits your web pages to contain minimal scripting since the presentation is separate from the PHP scripting.



- The **Model** represents your data structures. Typically, your model classes will contain functions that help you retrieve, insert and update information in your database.
- The **View** is information that is being presented to a user. A View will normally be a web page, but in LavaLust, a view can also be a page fragment like a header or footer. It can also be an RSS page, or any other type of "page".
- The **Controller** serves as an intermediary between the Model, the View, and any other resources needed to process the HTTP request and generate a web page.

Controllers

A controller is a simple class file. As the name suggests, it controls the whole application by URI.

➢ Creating a Controller

First, go to **app/controllers** folder. You will find two files there, **index.html** and **Welcome.php**. These files come with the LavaLust.

Keep these files as they are. Create a new file under the same path named "**Test.php**". Write the following code in that file –

Test.php U

lava > app > controllers > Test.php > ...

```php
<?php
defined('PREVENT_DIRECT_ACCESS') OR exit('No direct script access allowed');

class Test extends Controller {
    public function index() {
        echo 'Hello World';
    }
}
?>
```

The **Test** class extends an in-built class called **Controller**. This class must be extended whenever you want to make your own Controller class.

- Calling a Controller

The above controller can be called by URI as follows −

```
http://your-domain.com/index.php/controller-name/method-name
```

- Creating & Calling Constructor Method

Let us modify the above class and create another method named “hello”.

Test.php U

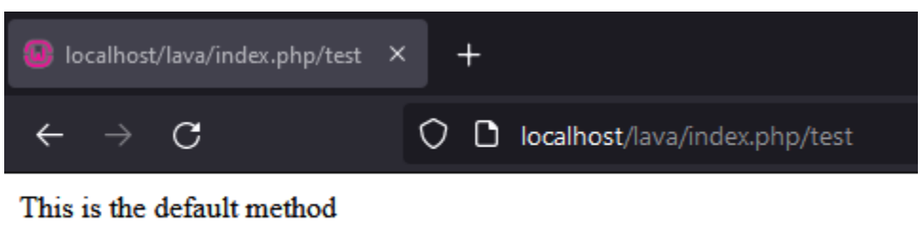
lava > app > controllers > Test.php > ...

```php
<?php
defined('PREVENT_DIRECT_ACCESS') OR exit('No direct script access allowed');

class Test extends Controller {

    public function index() {
        echo 'This is the default method';
    }

    public function hello() {
        echo 'This is a hello method';
    }

}
?>
```

We can execute the above controller in the following three ways −

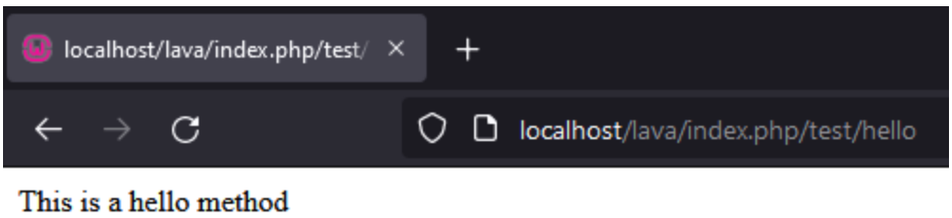
- http://www.your-domain.com/index.php/test
- http://www.your-domain.com/index.php/test/index
- http://www.your-domain.com/index.php/test/hello

After visiting the first URI in the browser, we get the output as shown in the picture given below. As you can see, we got the output of the method “**index**”, even though we did not pass the name of the method the URI. We have used only controller name in the URI. In such situations, the LavaLust calls the default method “**index**”.



Visiting the second URI in the browser, we get the same output as shown in the above picture. Here, we have passed method’s name after controller’s name in the URI. As the name of the method is “**index**”, we are getting the same output.

Visiting the third URI in the browser, we get the output as shown in picture given below. As you can see, we are getting the output of the method “**hello**” because we have passed “**hello**” as the method name, after the name of the controller “**test**” in the URI.

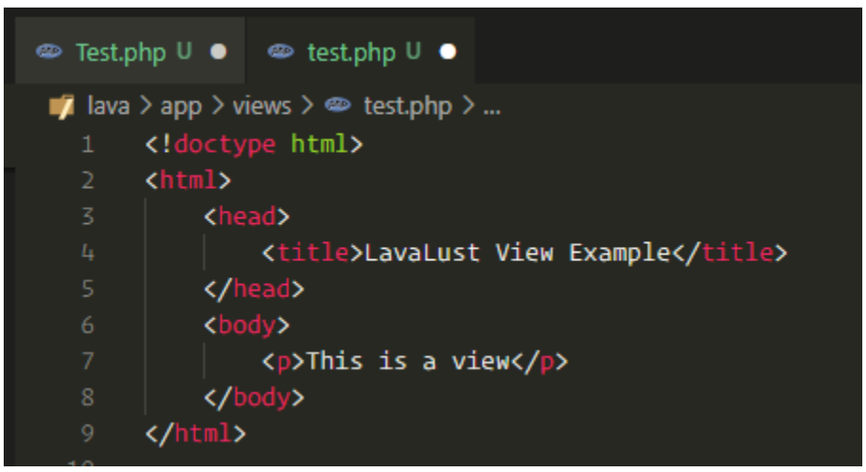


- Points to Remember

  - The name of the controller class must start with an uppercase letter.
  - The controller must be called with lowercase letter.
  - Do not use the same name of the method as your parent class, as it will override parent class's functionality.

Views

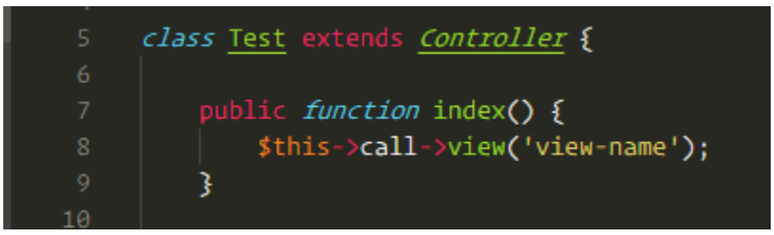
This can be a simple or complex webpage, which can be called by the controller. The webpage may contain header, footer, sidebar etc. View cannot be called directly. Let us create a simple view. Create a new file under **app/views** with name "**test.php**" and copy the below given code in that file.



Change the code of **app/controllers/test.php** file as shown in the below.

- Loading the View

The view can be loaded by the following syntax −



Where name is the view file, which is being rendered. If you have planned to store the view file in some directory then you can use the following syntax –

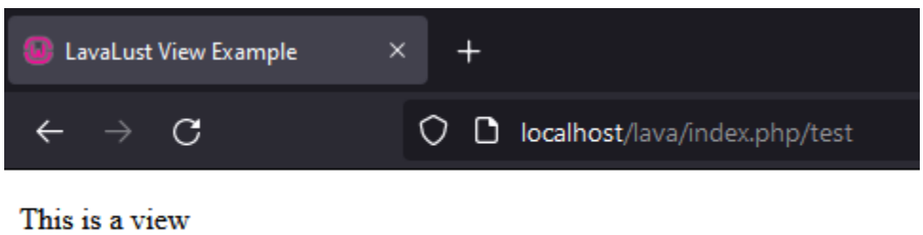
```php
class Test extends Controller {

    public function index() {
        $this->call->view('directory-name/view-name');
    }

```

It is not necessary to specify the extension as php, unless something other than .php is used.

The index() method is calling the view method and passing the “test” as argument to view() method because we have stored the html coding in “**test.php**” file under **app/views/test.php**.

```php
class Test extends Controller {

    public function index() {
        $this->call->view('test');
    }

}
?>
```

Here is the output of the above code –



This is a view

Models

Models classes are designed to work with information in the database. As an example, if you are using LavaLust to manage users in your application then you must have model class, which contains functions to insert, delete, update and retrieve your users’ data.

- Creating Model Class

Model classes are stored in **app/models** directory. Following code shows how to create model class in LavaLust.

```php
class Model_model extends Model {
    public function index() {

    }
}
?>
```

Where Model_name is the name of the model class that you want to give. Each model class must inherit the LavaLust’s Model class. The first letter of the model class must be in capital letter. Following is the code for users’ model class.

```php
<?php
defined('PREVENT_DIRECT_ACCESS') OR exit('No direct script access allowed');

class User_model extends Model {
    public function index() {

    }
}
?>
```

The above model class must be saved as User_model.php. The class name and file name must be same.

- Loading Model

Model can be called in controller. Following code can be used to load any model.

```php
$this->call->model('modelname_model');
```

Where modelname_model is the name of the model to be loaded. After loading the model you can simply call its method as shown below.

```php
$this->call->modelname_model->method();
```

- Auto-loading Models

There may be situations where you want some model class throughout your application. In such situations, it is better if we autoload it.

```php
/*
| -------------------------------------------------------------------
|  Auto-load Models
| -------------------------------------------------------------------
| Prototype:
|
|   $autoload['model'] = array('model1', 'model2')
*/
$autoload['models'] = array();
?>
```

As shown in the above figure, pass the name of the model in the array that you want to autoload and it will be autoloaded, while system is in initialization state and is accessible throughout the application.

Helpers

As the name suggests, it will help you build your system. It is divided into small functions to serve different functionality. A number of helpers are available in LavaLust, which are listed in the table below. We can build our own helpers too.

Helpers are typically stored in your **scheme/helpers**, or **app/helpers directory**. Custom helpers are stored in **app/helpers** directory and systems' helpers are stored in **scheme/helpers** directory. LavaLust will look first in your **app/helpers directory**. If the directory does not exist or the specified helper is not located, LavaLust will instead, look in your global ***scheme/helpers/* directory**. Each helper, whether it is custom or system helper, must be loaded before using it.

Cookie Helper

The Cookie Helper file contains functions that assist in working with cookies.

Directory Helper

The Directory Helper file contains functions that assist in working with directories.

Download Helper

The Download Helper lets you download data to your desktop.

File Helper

The File Helper file contains functions that assist in working with files.

Form Helper

The Form Helper file contains functions that assist in working with forms.

Language Helper

The Language Helper file contains functions that assist in working with language files.

Security Helper

The Security Helper file contains security related functions.

String Helper

The String Helper file contains functions that assist in working with strings.

URL Helper

The URL Helper file contains functions that assist in working with URLs.

- Loading a Helper

A helper can be loaded as shown below −

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

Where name is the name of the helper. For example, if you want to load the URL Helper, then it can be loaded as –

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

Routing

LavaLust has user-friendly URI routing system, so that you can easily re-route URL. Typically, there is a one-to-one relationship between a URL string and its corresponding controller class/method. The segments in a URI normally follow this pattern −

```
https://your-domain.com/index.php/class/method/id
```

- 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.

In some situations, you may want to change this default routing mechanism. LavaLust provides facility through which you can set your own routing rules.

- Customize Routing Rules

There is a particular file where you can handle all these. The file is located at ap/config/routes.php. You will find an array called $route in which you can customize your routing rules. The key in the $route array will decide what to route and the value will decide where to route. There are three reserved routes in LavaLust.

$route['default_controller']

This route indicates which controller class should be loaded, if the URI contains no data, which will be the case when people load your root URL. You are encouraged to have a default route otherwise a 404 page will appear, by default. We can set home page of website here so it will be loaded by default.

$route['translate_uri_dashes']

As evident by the Boolean value, this is not exactly a route. This option enables you to automatically replace dashes ('-') with underscores in the controller and method URI segments, thus saving you additional route entries if you need to do that. This is required because the dash is not a valid class or method-name character and will cause a fatal error, if you try to use it.

Routes can be customized by **wildcards** or by using **regular expressions** but keep in mind that these customized rules for routing must come after the reserved rules.

- Wildcards

We can use two wildcard characters as explained below −

- **(:num)** − It will match a segment containing only numbers.
- **(:any)** − It will match a segment containing any character.

Example

```
$route['product/:num']='catalog/product_lookup';
```

In the above example, if the literal word "product" is found in the first segment of the URL, and a number is found in the second segment, the "catalog" class and the "product_lookup" method are used instead.

- Regular Expressions

Like wildcards, we can also use regular expressions in **$route array key** part. If any URI matches with regular expression, then it will be routed to the value part set into $route array.

Example

```
$route['products/([a-z]+)/(\d+)']='$1/id_$2';
```

In the above example, a URI similar to products/shoes/123 would instead call the "**shoes**" controller class and the "**id_123**" method.

After setting up the site, the next thing that we should do is to configure the site. The app/config folder contains a group of files that set basic configuration of your site.

Configuring Base URL

The base URL of the site can be configured in app/config/config.php file. It is URL to your LavaLust root. Typically, this will be your base URL, with a trailing slash e.g.

```
http://example.com/
```

```
$config['base_url']                    = 'http://your-domain.com/';
```

Database Configuration

The database of the site can be configured in app/config/database.php file. Often we need to set up database for different environment like development and production. With the multidimensional array provided in the LavaLust, we can setup database for different environment. The configuration settings are stored in the array as shown below −

```
58  $database['driver']     = '';
59  $database['hostname']   = '';
60  $database['port']       = '';
61  $database['username']   = '';
62  $database['password']   = '';
63  $database['database']   = '';
64  $database['charset']    = '';
65  $database['dbprefix']   = '';
```

You can leave few options to their default values except hostname, username, password, database and driver.

- **hostname** − Specify location of your database here e.g. localhost or IP address
- **port** – Specify the port to connect.
- **username** − Set username of your database here.
- **password** − Set password of your database here.
- **database** − Set name of the database here.
- **driver** − Set type of database that you are using e.g. MySQL
- **charset** – The character set to be used.

Autoload Configuration

This file specifies, by default, which systems should be loaded. In order to keep the framework as light-weight as possible, only the absolute minimal resources are loaded by default. One should autoload the frequently used system, rather than loading it at local level, repeatedly. Following are the things you can load automatically −

- **Libraries** − It is a list of libraries, which should be auto loaded. Provide a list of libraries in an array as shown below to be autoloaded by LavaLust. In this example, we are auto loading database, email and session libraries.

```
$autoload['libraries'] = array('database', 'email', 'session');
```

- **Helper files** − It is a list of helper files, to be autoloaded. Provide a list of libraries in the array, as shown below, to be autoloaded by LavaLust. In the given example, we are autoloading URL and file helpers.

```
$autoload['helpers'] = array('url', 'file');
```

- **Models** − It is a list of models file, which should be autoloaded. Provide a list of models in an array as shown below to be autoloaded by LavaLust. Following is the example of how to auto load more than one models files.

```
$autoload['models'] = array('mode1_model', 'model2_model');
```

Like any other framework, we need to interact with the database very often and LavaLust makes this job easy for us. It provides rich set of functionalities to interact with database.

Connecting to a Database

We can connect to database in the following two way −

- **Automatic Connecting** − Automatic connection can be done by using the file app/config/autoload.php. Automatic connection will load the database for each and every page. We just need to add the database library as shown below −

```
$autoload['libraries'] = array('database');
```

- **Manual Connecting** − If you want database connectivity for only some of the pages, then we can go for manual connecting. We can connect to database manually by adding the following line in any class.

```
$this->call->database();
```

Here, we are not passing any argument because everything is set in the database config file app/config/database.php

For complete methods in Query Builder see

https://ronmarasigan.github.io

LavaLust Libraries

The essential part of a LavaLust framework is its libraries. It provides a rich set of libraries, which indirectly increase the speed of developing an application. The system library is located at system/libraries. All we need to do is to load the library that we want to use. The library can be loaded as shown below −

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

Where **class name** is the name of the library that we want to load. If we want to load multiple libraries, then we can simply pass an array as argument to **library()** function as shown below −

```
$this->call->library(array('email', 'file'));
```

Library Classes

The library classes are located in **system/libraries**. Each class has various functions to simplify the developing work. Following table shows the names of the library class and its description.

Given below are the most commonly used Library Classes.

Performance Class

Benchmarking class is always active, enabling the time difference between any two marked points to be calculated.

Caching Class

This class will cache the pages, to quickly access the page speed.

Email Class

This class provides email related functionality, like send or reply to email.

Form Validation Class

This class provides various functions to validate form.

IO Class

This class provides http request functionality

Security Class

This class contains security related functions like XSS Filtering, CSRF etc.

Session Library

This class provides functionalities to maintain session of your application.

Creating Libraries

LavaLust has rich set of libraries, which you can find in **scheme/libraries** folder but LavaLust is not just limited to system libraries, you can create your own libraries too, which can be stored in **app/libraries** folder. You can create libraries in three ways.

- Create new library
- Extend the native library
- Replace the native library

➢ Create New Library

While creating new library one should keep in mind, the following things −

- The name of the file must start with a capital letter e.g. Mylibrary.php
-

Mylibrary.php

```php
<?php
defined('PREVENT_DIRECT_ACCESS') OR exit('No direct script access allowed');

class Mylibrary {

    public function some_function() {

    }
}
```

Loading the Custom Library

The above library can be loaded by simply executing the following line in your controller.

```
$this->load->library('mylibrary');
```

mylibrary is the name of your library and you can write it in lowercase as well as uppercase letters. Use the name of the library without ".php" extension. After loading the library, you can also call the function of that class as shown below.

```
$this->mylibrary->some_function();
```

LavaLust Email Sending

Sending email in LavaLust is much easier. You also configure the preferences regarding email in LavaLust. LavaLust provides following features for sending emails –

- Multiple recipients
- HTML or Plaintext email
- Attachments
- Word wrapping

```php
$this->email->recipient('receiver@receiveremail.com');
$this->email->subject('This is an email');
$this->email->sender('sender@sendemail.com');
$this->email->reply_to('sender@sendemail.com');
$this->email->email_content($template, 'html');
$this->email->send();
```

Email class has the following functions to simplify the job of sending emails.

$sender (*string*) − "From" e-mail address
$subject (*string*) – Email subject
$email_content (*string*) – Plain or HTML
$recipient (string) – "Sender" of email
$attachment (file) – "link of your attachment file"

LavaLust Form Validation

Validation is an important process while building web application. It ensures that the data that we are getting is proper and valid to store or process. LavaLust has made this task very easy. Let us understand this process with a simple example.

```php
<?php
if($this->form_validation->submitted()) {
    $this->form_validation
        ->name('lastname')->required()
        ->name('firstname')->required()
        ->name('age')->numeric()
        ->name('address')->required();
            if($this->form_validation->run()) {
                $lastname = $this->io->post('lastname');
                $firstname = $this->io->post('firstname');
                $age = $this->io->post('age');
                $address = $this->io->post('address');
                } else {
                    echo
                    '<div class="alert alert-danger alert-dismissible fade show">
                        <button type="button" class="close" data-dismiss="alert">&times;</button>
                            '.validation_errors().'
                    </div>';
                }
}

```

Validation Rule Reference

The following is a list of all the native rules that are available to use −

Given below are the most commonly used list of native rules available to use.

Rule	Parameter	Description	Example
required	No	Returns FALSE if the form element is empty.	
matches	Yes	Returns FALSE if the form element does not match the one in the parameter.	matches[form_item]
differs	Yes	Returns FALSE if the form element does not differ from the one in the parameter.	differs[form_item]
is_unique	Yes	Returns FALSE if the form element is not unique to the table and field name in the parameter. Note − This rule requires *Query Builder* to be enabled in order to work.	is_unique[table.field]
min_length	Yes	Returns FALSE if the form element is shorter than the parameter value.	min_length[3]

max_length	Yes	Returns FALSE if the form element is longer than the parameter value.	max_length[12]
exact_length	Yes	Returns FALSE if the form element is not exactly the parameter value.	exact_length[8]
greater_than	Yes	Returns FALSE if the form element is less than or equal to the parameter value or not numeric.	greater_than[8]
greater_than_equal_to	Yes	Returns FALSE if the form element is less than the parameter value, or not numeric.	greater_than_equal_to[8]
less_than	Yes	Returns FALSE if the form element is greater than or equal to the parameter value or not numeric.	less_than[8]
less_than_equal_to	Yes	Returns FALSE if the form element is greater than the parameter value, or not numeric.	less_than_equal_to[8]
in_list	Yes	Returns FALSE if the form element is not within a predetermined list.	in_list[red,blue,green]
alpha	No	Returns FALSE if the form element contains anything other than alphabetical characters.	
alpha_numeric	No	Returns FALSE if the form element contains anything other than alphanumeric characters.	
alpha_numeric_space	No	Returns FALSE if the form element contains anything other than	

		alphanumeric characters or spaces. Should be used after trim to avoid spaces at the beginning or end	
alpha_dash	No	Returns FALSE if the form element contains anything other than alphanumeric characters, underscores or dashes.	
numeric	No	Returns FALSE if the form element contains anything other than numeric characters.	
integer	No	Returns FALSE if the form element contains anything other than an integer.	
decimal	No	Returns FALSE if the form element contains anything other than a decimal number.	
valid_email	No	Returns FALSE if the form element does not contain a valid email address.	

Check my video tutorials to see the different methods of form validation
https://youtube.com/ronmarasigan
or
LavaLust Docs
https://ronmarasigan.github.io

LavaLust Session

When building websites, we often need to track user's activity and state and for this purpose, we have to use **session**. LavaLust has session class for this purpose.

- **Initializing a Session**

Sessions data are available globally through the site but to use those data we first need to initialize the session. We can do that by executing the following line in constructor.

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

After loading the session library, you can simply use the session object as shown below.

```
$this->session;
```

- **Add Session Data**

In PHP, we simply use **$_SESSION** array to set any data in session as shown below.

```
$_SESSION['key'] = value;
```

Where '**key**' is the key of array and **value** is assigned on right side of equal to sign.

The same thing can be done in LavaLust as shown below.

```
$this->session->set_userdata('some_name', 'some_value');
```

set_userdata() function takes two arguments. The first argument, **some_name**, is the name of the session variable, under which, **some_value** will be stored.

set_userdata() function also supports another syntax in which you can pass array to store values as shown below.

```
$newdata = array(
   'username'  => 'johndoe',
   'email'     => 'johndoe@some-site.com',
   'logged_in' => TRUE
);

$this->session->set_userdata($newdata);
```

- **Remove Session Data**

In PHP, we can remove data stored in session using the **unset()** function as shown below.

```
$this->session->unset_userdata('some_name');
```

If you want to remove more values from session or to remove an entire array you can use the below version of **unset_userdata()** function.

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

- **Fetch Session Data**

After setting data in session, we can also retrieve that data as shown below. **Userdata()** function will be used for this purpose. This function will return **NULL** if the data you are trying to access is not available.

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

While building web application, we need to store some data for only one time and after that we want to remove that data. For example, to display some error message or information message. In PHP, we have to do it manually but LavaLust has made this job simple for us. In LavaLust, flashdata will only be available until the next request, and it will get deleted automatically.

- **Add Flashdata**

We can simply store flashdata as shown below.

```
$this->session->mark_as_flash('item');
```

- **mark_as_flash()** function is used for this purpose, which takes only one argument of the value to be stored. We can also pass an array to store multiple values.

- **set_flashdata()** function can also be used, which takes two arguments, name and value, as shown below. We can also pass an array.

```
$this->session->set_flashdata('item','value');
```

➢ **Retrieve Flashdata**

Flashdata can be retrieved using the flashdata() function which takes one argument of the item to be fetched as shown below. flashdata() function makes sure that you are getting only flash data and not any other data.

```
$this->session->flashdata('item');
```

If you do not pass any argument, then you can get an array with the same function.

LavaLust Cookie Management

Cookie is a small piece of data sent from web server to store on client's computer. LavaLust has one helper called "Cookie Helper" for cookie management.

Syntax	set_cookie(*$name*[, *$value* = *''*[, *$expire* = *''*[, *$domain* = *''*[, *$path* = *'/'*[, *$prefix* = *''*[, *$secure* = *FALSE[, $httponly* = *FALSE*]]]]]]]])
Parameters	• **$name** (*mixed*) − Cookie name or associative array of all of the parameters available to this function • **$value** (*string*) − Cookie value • **$expire** (*int*) − Number of seconds until expiration • **$domain** (*string*) − Cookie domain (usually: .yourdomain.com) • **$path** (*string*) − Cookie path • **$prefix** (*string*) − Cookie name prefix • **$secure** (*bool*) − Whether to only send the cookie through HTTPS • **$httponly** (*bool*) − Whether to hide the cookie from JavaScript
Return Type	void

In the **set_cookie()** function, we can pass all the values using two ways. In the first way, only array can be passed and in the second way, individual parameters can also be passed.

Syntax	get_cookie(*$index*[, *$xss_clean* = *NULL*]])
Parameters	• **$index** (*string*) − Cookie name • **$xss_clean** (*bool*) − Whether to apply XSS filtering to the returned value
Return	The cookie value or NULL if not found
Return Type	mixed

The **get_cookie()** function is used to get the cookie that has been set using the set_cookie() function.

Syntax	delete_cookie(*$name*[, *$domain* = *''*[, *$path* = *'/'*[, *$prefix* = *''*]]])
Parameters	• **$name** (*string*) − Cookie name • **$domain** (*string*) − Cookie domain (usually: .yourdomain.com) • **$path** (*string*) − Cookie path • **$prefix** (*string*) − Cookie name prefix
Return Type	void

The **delete_cookie()** function is used to delete the cookie().

LavaLust Cache

Caching a page will improve the page load speed. If the page is cached, then it will be stored in its fully rendered state. Next time, when the server gets a request for the cached page, it will be directly sent to the requested browser.

Cached files are stored in **app/cache** folder. Caching can be enabled on per page basis. While enabling the cache, we need to set the time, until which it needs to remain in cached folder and after that period, it will be deleted automatically.

```php
<?php
// Uncached model call
$this->blog_m->getPosts($category_id, 'live');

// cached model call
$this->cache->model('blog_m', 'getPosts', array($category_id, 'live'), 120); // keep for 2 minutes

// cached library call
$this->cache->library('some_library', 'calcualte_something', array($foo, $bar, $bla)); // keep for default time (0 = unlimited)

// cached array or object
$this->cache->write($data, 'cached-name');
$data = $this->cache->get('cached-name');

// Delete cache
$this->cache->delete('cached-name');

// Delete all cache
$this->cache->delete_all();
```

```php
// Delete all cache
$this->cache->delete_all();

// Delete cache group
$this->cache->write($data, 'nav_header');
$this->cache->write($data, 'nav_footer');
$this->cache->delete_group('nav_');

// Delete cache item
// Call like a normal library or model but give a negative $expire
$this->cache->model('blog_m', 'getPosts', array($category_id, 'live'), -1); // delete this specific cache file
```

LavaLust Performance Class

If you want to measure the time taken to execute a set of lines or memory usage, you can calculate it by using Performance points in LavaLust. There is a separate “**Perfomance**” class for this purpose in LavaLust.

This class is loaded automatically; you do not have to load it. It can be used anywhere in your controller, view, and model classes. All you have to do is to mark a start point and end point and then execute the **elapsed_time()** function between these two marked points and you can get the time it took to execute that code as shown below.

```php
$this->performance->tag('tag_name');

//Some codes here

$this->performance->tag('tagname');

echo $this->performance->elapsed_time('tagname');
```

To display the memory usage, use the function **memory_usage()** as shown in the following code.

```php
$this->performance->memory_usage();
```