CodeIgniter is a powerful PHP framework with a very small footprint, built for developers who need a simple and elegant toolkit to create full-featured web applications. CodeIgniter was created by EllisLab, and is now a project of the British Columbia Institute of Technology.

CodeIgniter Features

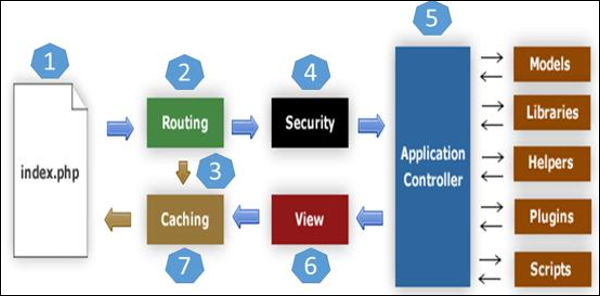
Some of the important features of CodeIgniter are listed below −

* Model-View-Controller Based System
* Extremely Light Weight
* Full Featured database classes with support for several platforms.
* Query Builder Database Support
* Form and Data Validation
* Security and XSS Filtering
* Session Management
* Email Sending Class. Supports Attachments, HTML/Text email, multiple protocols (sendmail, SMTP, and Mail) and more.
* Image Manipulation Library (cropping, resizing, rotating, etc.). Supports GD, ImageMagick, and NetPBM
* File Uploading Class
* FTP Class
* Localization
* Pagination
* Data Encryption
* Benchmarking
* Full Page Caching
* Error Logging
* Application Profiling
* Calendaring Class
* User Agent Class
* Zip Encoding Class
* Template Engine Class
* Trackback Class
* XML-RPC Library
* Unit Testing Class
* Search-engine Friendly URLs
* Flexible URI Routing
* Support for Hooks and Class Extensions
* Large library of “helper” functions

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

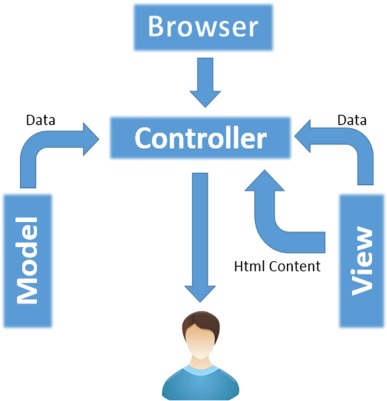
* **Step-1** − Download the CodeIgniter from the link [CodeIgniter](http://www.codeigniter.com/download" \t "_blank)
  + There are two different options legacy and latest. The names itself are self descriptive. legacy has version less than 2.x and latest has 3.0 version.
  + We can also go with GitHub 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.

The architecture of CodeIgniter application is shown below.



* As shown in the figure, whenever a request comes to CodeIgniter, 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, Plugins** 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.

CodeIgniter 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 CodeIgniter, 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 **application/controllers** folder. You will find two files there, **index.html** and **Welcome.php**. These files come with the CodeIgniter.

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

<?php

class Test extends CI\_Controller {

public function index() {

echo "Hello World!";

}

}

?>

The **Test** class extends an in-built class called **CI\_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://www.your-domain.com/index.php/test

Notice the word “**test**” in the above URI after index.php. This indicates the class name of controller. As we have given the name of the controller “**Test**”, we are writing “**test**” after the index.php. The class name must start with **uppercase letter** but we need to write **lowercase letter** when we call that controller by URI. The general syntax for calling the controller is as follows −

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

### **Creating & Calling Constructor Method**

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

<?php

class Test extends CI\_Controller {

public function index() {

echo "This is default function.";

}

public function hello() {

echo "This is hello function.";

}

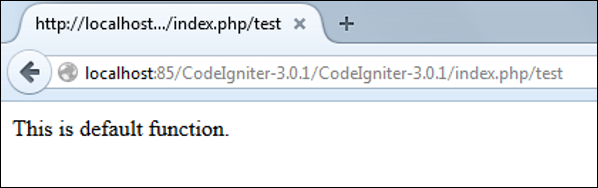
}

?>

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

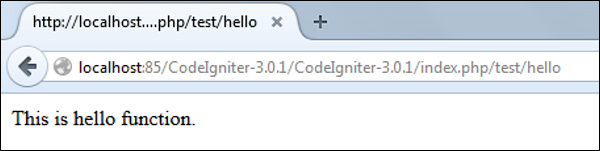
* <http://www.your-domain.com/index.php/test>
* <http://www.your-domain.com/index.php/test/index>
* <http://www.your-domain.com/index.php/test/hello>

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 CodeIgniter 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 **application/views** with name “**test.php**” and copy the below given code in that file.

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "utf-8">

<title>CodeIgniter View Example</title>

</head>

<body>

CodeIgniter View Example

</body>

</html>

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

### **Loading the View**

The view can be loaded by the following syntax −

$this->load->view('name');

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 −

$this->load->view('directory-name/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 **application/views/test.php**.

<?php

class Test extends CI\_Controller {

public function index() {

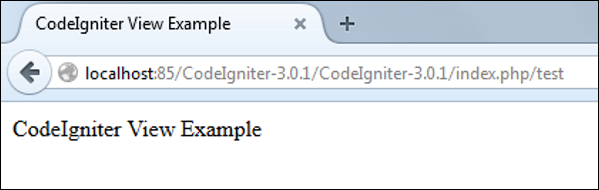
$this->load->view('test');

}

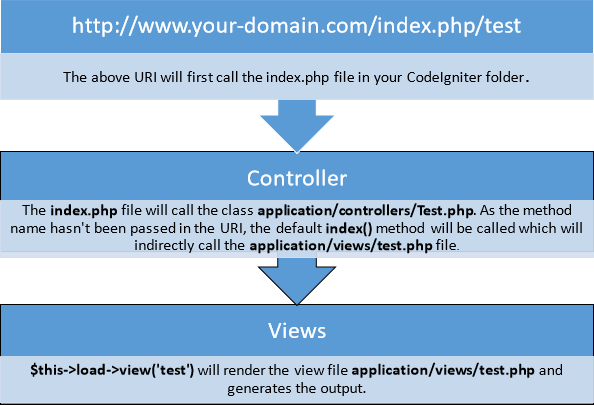
}

?>

Here is the output of the above code −



The following flowchart illustrates of how everything works −



## Models

Models classes are designed to work with information in the database. As an example, if you are using CodeIgniter 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 **application/models** directory. Following code shows how to create model class in CodeIgniter.

<?php

Class Model\_name extends CI\_Model {

Public function \_\_construct() {

parent::\_\_construct();

}

}

?>

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

<?php

Class User\_model extends CI\_Model {

Public function \_\_construct() {

parent::\_\_construct();

}

}

?>

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.

$this->load->model('model\_name');

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

$this->model\_name->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.

/\*

| ---------------------------------------------------------------

| Auto-Load Models

| ---------------------------------------------------------------

| Prototype:

|

| $autoload['model'] = array('first\_model', 'second\_model');

|

| You can also supply an alternative model name to be assigned

| in the controller:

|

| $autoload['model'] = array('first\_model' => 'first');

\*/

$autoload['model'] = 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 CodeIgniter, which are listed in the table below. We can build our own helpers too.

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

Given below are the most commonly used Helpers.

### **Loading a Helper**

A helper can be loaded as shown below −

$this->load->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->load->helper('url');

## Routing

CodeIgniter 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 −

your-domain.com/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. CodeIgniter 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 application/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 CodeIgniter.

|  |  |
| --- | --- |
| **S.N.** | **Reserved Routes & Description** |
| 1 | **$route['default\_controller']**  This route indicates which controller class should be loaded, if the URI contains no data, which will be the case when people load your root URL. You are encouraged to have a default route otherwise a 404 page will appear, by default. We can set home page of website here so it will be loaded by default. |
| 2 | **$route['404\_override']**  This route indicates which controller class should be loaded if the requested controller is not found. It will override the default 404 error page. It won’t affect to the **show\_404()** function, which will continue loading the default ***error\_404.php*** file in ***application/views/errors/error\_404.php***. |
| 3 | **$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.