CodeIgniter - MVC Framework

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.

# CodeIgniter - Basic Concepts

## 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.

# CodeIgniter - Configuration

After setting up the site, the next thing that we should do is to configure the site. The application/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 application/config/config.php file. It is URL to your CodeIgniter root. Typically, this will be your base URL, with a trailing slash e.g.

http://example.com/

If this is not set, then CodeIgniter will try to guess the protocol, domain and path to your installation. However, you should always configure this explicitly and never rely on autoguessing, especially in production environments. You can configure the base URL in the $config array with key “base\_url” as shown below −

$config['base\_url'] = 'http://your-domain.com';

## Database Configuration

The database of the site can be configured in application/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 CodeIgniter, we can setup database for different environment. The configuration settings are stored in the array as shown below −

$db['default'] = array(

'dsn' => '',

'hostname' => 'localhost',

'username' => 'root',

'password' => '',

'database' => 'database\_name',

'dbdriver' => 'mysqli',

'dbprefix' => '',

'pconnect' => TRUE,

'db\_debug' => TRUE,

'cache\_on' => FALSE,

'cachedir' => '',

'char\_set' => 'utf8',

'dbcollat' => 'utf8\_general\_ci',

'swap\_pre' => '',

'encrypt' => FALSE,

'compress' => FALSE,

'stricton' => FALSE,

'failover' => array()

);

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

* **hostname** − Specify location of your database here e.g. localhost or IP address
* **username** − Set username of your database here.
* **password** − Set password of your database here.
* **database** − Set name of the database here.
* **dbdriver** − Set type of database that you are using e.g. MySQL, MySQLi, Postgre SQL, ODBC, and MS SQL.

By changing the key of the array **$db**, you can set other configuration of database as shown below. Here, we have set the key to **‘test’** to set the database for testing environment, by keeping the other database environment as it is.

$db['test'] = array(

'dsn' => '',

'hostname' => 'localhost',

'username' => 'root',

'password' => '',

'database' => 'database\_name',

'dbdriver' => 'mysqli',

'dbprefix' => '',

'pconnect' => TRUE,

'db\_debug' => TRUE,

'cache\_on' => FALSE,

'cachedir' => '',

'char\_set' => 'utf8',

'dbcollat' => 'utf8\_general\_ci',

'swap\_pre' => '',

'encrypt' => FALSE,

'compress' => FALSE,

'stricton' => FALSE,

'failover' => array()

);

You can simply switch to different environment by changing the value of a variable as shown below −

**$active\_group = ‘default’; //This will set the default environment**

**$active\_group = ‘test’; //This will set the test environment**

## 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 CodeIgniter. In this example, we are auto loading database, email and session libraries.

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

* **Drivers** − These classes are located in system/libraries/ or in your application/libraries/ directory, but are also placed inside their own subdirectory and they extend the CI\_Driver\_Library class. They offer multiple interchangeable driver options. Following is an example to autoload cache drivers.

$autoload['drivers'] = array('cache');

* **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 CodeIgniter. In the given example, we are autoloading URL and file helpers.

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

* **Custom config files** − These files are intended for use, only if you have created custom config files. Otherwise, leave it blank. Following is an example of how to autoload more than one config files.

$autoload['language'] = array('lang1', 'lang2');

* **Language files** − It is a list of language files, which should be auto loaded. Look at the example given below. Provide a list of languages in an array as shown below to be auto loaded by CodeIgniter. Keep in mind that do not include the "\_lang" part of your file. For example, "codeigniter\_lang.php" would be referenced as array('codeigniter');
* **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 CodeIgniter. Following is the example of how to auto load more than one models files.

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

CodeIgniter - Working with Database

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

In this section, we will understand how the CRUD (Create, Read, Update, Delete) functions work with CodeIgniter. We will use **stud**table to select, update, delete, and insert the data in **stud** table.

|  |  |
| --- | --- |
| **Table Name: stud** | |
| roll\_no | int(11) |
| Name | varchar(30) |

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 application/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->load->database();

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

Inserting a Record

To insert a record in the database, the insert() function is used as shown in the following table −

|  |  |
| --- | --- |
| **Syntax** | insert([*$table = ''*[, *$set = NULL*[, *$escape = NULL*]]]) |
| **Parameters** | * **$table** (*string*) − Table name * **$set** (*array*) − An associative array of field/value pairs * **$escape** (*bool*) − Whether to escape values and identifiers |
| **Returns** | TRUE on success, FALSE on failure |
| **Return Type** | bool |

The following example shows how to insert a record in **stud** table. The $data is an array in which we have set the data and to insert this data to the table **stud**, we just need to pass this array to the insert function in the 2nd argument.

$data = array(

'roll\_no' => ‘1’,

'name' => ‘Virat’

);

$this->db->insert("stud", $data);

Updating a Record

To update a record in the database, the **update()** function is used along with **set()** and **where()** functions as shown in the tables below. The **set()** function will set the data to be updated.

|  |  |
| --- | --- |
| **Syntax** | set(*$key*[, *$value = ''*[, *$escape = NULL*]]) |
| **Parameters** | * **$key** (*mixed*) − Field name, or an array of field/value pairs * **$value** (*string*) − Field value, if $key is a single field * **$escape** (*bool*) − Whether to escape values and identifiers |
| **Returns** | CI\_DB\_query\_builder instance (method chaining) |
| **Return Type** | CI\_DB\_query\_builder |

The **where()** function will decide which record to update.

|  |  |
| --- | --- |
| **Syntax** | where(*$key*[, *$value = NULL*[, *$escape = NULL*]]) |
| **Parameters** | * **$key** (*mixed*) − Name of field to compare, or associative array * **$value** (*mixed*) − If a single key, compared to this value * **$escape** (*bool*) − Whether to escape values and identifiers |
| **Returns** | DB\_query\_builder instance |
| **Return Type** | object |

Finally, the **update()** function will update data in the database.

|  |  |
| --- | --- |
| **Syntax** | update([*$table = ''*[, *$set = NULL*[, *$where = NULL[, $limit = NULL*]]]]) |
| **Parameters** | * **$table** (*string*) − Table name * **$set** (*array*) − An associative array of field/value pairs * **$where** (*string*) − The WHERE clause * **$limit** (*int*) − The LIMIT clause |
| **Returns** | TRUE on success, FALSE on failure |
| **Return Type** | bool |

$data = array(

'roll\_no' => ‘1’,

'name' => ‘Virat’

);

$this->db->set($data);

$this->db->where("roll\_no", ‘1’);

$this->db->update("stud", $data);

Deleting a Record

To delete a record in the database, the delete() function is used as shown in the following table −

|  |  |
| --- | --- |
| **Syntax** | delete([*$table = ''*[, *$where = ''*[, *$limit = NULL[, $reset\_data = TRUE*]]]]) |
| **Parameters** | * **$table** (*mixed*) − The table(s) to delete from; string or array * **$where** (*string*) − The WHERE clause * **$limit** (*int*) − The LIMIT clause * **$reset\_data** (*bool*) − TRUE to reset the query “write” clause |
| **Returns** | CI\_DB\_query\_builder instance (method chaining) or FALSE on failure |
| **Return Type** | mixed |

Use the following code to to delete a record in the **stud** table. The first argument indicates the name of the table to delete record and the second argument decides which record to delete.

$this->db->delete("stud", "roll\_no = 1");

Selecting a Record

To select a record in the database, the **get** function is used, as shown in the following table −

|  |  |
| --- | --- |
| **Syntax** | get([*$table = ''*[, *$limit = NULL*[, *$offset = NULL*]]]) |
| **Parameters** | * **$table** (*string*) − The table to query array * **$limit** (*int*) − The LIMIT clause * **$offset** (*int*) − The OFFSET clause |
| **Returns** | CI\_DB\_result instance (method chaining) |
| **Return Type** | CI\_DB\_result |

Use the following code to get all the records from the database. The first statement fetches all the records from “stud” table and returns the object, which will be stored in $query object. The second statement calls the **result()** function with $query object to get all the records as array.

$query = $this->db->get("stud");

$data['records'] = $query->result();

Closing a Connection

Database connection can be closed manually, by executing the following code −

$this->db->close();

Example

Create a controller class called **Stud\_controller.php** and save it at **application/controller/Stud\_controller.php**

Here is a complete example, wherein all of the above-mentioned operations are performed. Before executing the following example, create a database and table as instructed at the starting of this chapter and make necessary changes in the database config file stored at **application/config/database.php**

<?php

class Stud\_controller extends CI\_Controller {

function \_\_construct() {

parent::\_\_construct();

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

$this->load->database();

}

public function index() {

$query = $this->db->get("stud");

$data['records'] = $query->result();

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

$this->load->view('Stud\_view',$data);

}

public function add\_student\_view() {

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

$this->load->view('Stud\_add');

}

public function add\_student() {

$this->load->model('Stud\_Model');

$data = array(

'roll\_no' => $this->input->post('roll\_no'),

'name' => $this->input->post('name')

);

$this->Stud\_Model->insert($data);

$query = $this->db->get("stud");

$data['records'] = $query->result();

$this->load->view('Stud\_view',$data);

}

public function update\_student\_view() {

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

$roll\_no = $this->uri->segment('3');

$query = $this->db->get\_where("stud",array("roll\_no"=>$roll\_no));

$data['records'] = $query->result();

$data['old\_roll\_no'] = $roll\_no;

$this->load->view('Stud\_edit',$data);

}

public function update\_student(){

$this->load->model('Stud\_Model');

$data = array(

'roll\_no' => $this->input->post('roll\_no'),

'name' => $this->input->post('name')

);

$old\_roll\_no = $this->input->post('old\_roll\_no');

$this->Stud\_Model->update($data,$old\_roll\_no);

$query = $this->db->get("stud");

$data['records'] = $query->result();

$this->load->view('Stud\_view',$data);

}

public function delete\_student() {

$this->load->model('Stud\_Model');

$roll\_no = $this->uri->segment('3');

$this->Stud\_Model->delete($roll\_no);

$query = $this->db->get("stud");

$data['records'] = $query->result();

$this->load->view('Stud\_view',$data);

}

}

?>

Create a model class called **Stud\_Model.php** and save it in**application/models/Stud\_Model.php**

<?php

class Stud\_Model extends CI\_Model {

function \_\_construct() {

parent::\_\_construct();

}

public function insert($data) {

if ($this->db->insert("stud", $data)) {

return true;

}

}

public function delete($roll\_no) {

if ($this->db->delete("stud", "roll\_no = ".$roll\_no)) {

return true;

}

}

public function update($data,$old\_roll\_no) {

$this->db->set($data);

$this->db->where("roll\_no", $old\_roll\_no);

$this->db->update("stud", $data);

}

}

?>

Create a view file called **Stud\_add.php** and save it in**application/views/Stud\_add.php**

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "utf-8">

<title>Students Example</title>

</head>

<body>

<?php

echo form\_open('Stud\_controller/add\_student');

echo form\_label('Roll No.');

echo form\_input(array('id'=>'roll\_no','name'=>'roll\_no'));

echo "<br/>";

echo form\_label('Name');

echo form\_input(array('id'=>'name','name'=>'name'));

echo "<br/>";

echo form\_submit(array('id'=>'submit','value'=>'Add'));

echo form\_close();

?>

</body>

</html>

Create a view file called **Stud\_edit.php** and save it in**application/views/Stud\_edit.php**

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "utf-8">

<title>Students Example</title>

</head>

<body>

<form method = "" action = "">

<?php

echo form\_open('Stud\_controller/update\_student');

echo form\_hidden('old\_roll\_no',$old\_roll\_no);

echo form\_label('Roll No.');

echo form\_input(array('id'⇒'roll\_no',

'name'⇒'roll\_no','value'⇒$records[0]→roll\_no));

echo "

";

echo form\_label('Name');

echo form\_input(array('id'⇒'name','name'⇒'name',

'value'⇒$records[0]→name));

echo "

";

echo form\_submit(array('id'⇒'sub mit','value'⇒'Edit'));

echo form\_close();

?>

</form>

</body>

</html>

Create a view file called **Stud\_view.php** and save it in**application/views/Stud\_view.php**

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "utf-8">

<title>Students Example</title>

</head>

<body>

<a href = "<?php echo base\_url(); ?>

index.php/stud/add\_view">Add</a>

<table border = "1">

<?php

$i = 1;

echo "<tr>";

echo "<td>Sr#</td>";

echo "<td>Roll No.</td>";

echo "<td>Name</td>";

echo "<td>Edit</td>";

echo "<td>Delete</td>";

echo "<tr>";

foreach($records as $r) {

echo "<tr>";

echo "<td>".$i++."</td>";

echo "<td>".$r->roll\_no."</td>";

echo "<td>".$r->name."</td>";

echo "<td><a href = '".base\_url()."index.php/stud/edit/"

.$r->roll\_no."'>Edit</a></td>";

echo "<td><a href = '".base\_url()."index.php/stud/delete/"

.$r->roll\_no."'>Delete</a></td>";

echo "<tr>";

}

?>

</table>

</body>

</html>

Make the following change in the route file at**application/config/routes.php** and add the following line at the end of file.

$route['stud'] = "Stud\_controller";

$route['stud/add'] = 'Stud\_controller/add\_student';

$route['stud/add\_view'] = 'Stud\_controller/add\_student\_view';

$route['stud/edit/(\d+)'] = 'Stud\_controller/update\_student\_view/$1';

$route['stud/delete/(\d+)'] = 'Stud\_controller/delete\_student/$1';

Now, let us execute this example by visiting the following URL in the browser. Replace the yoursite.com with your URL.

http://yoursite.com/index.php/stud

# CodeIgniter - Page Redirection

While building web application, we often need to redirect the user from one page to another page. CodeIgniter makes this job easy for us. The **redirect()** function is used for this purpose.

|  |  |
| --- | --- |
| **Syntax** | redirect(*$uri = '', $method = 'auto', $code = NULL*) |
| **Parameters** | * **$uri** (*string*) − URI string * **$method** (*string*) − Redirect method (‘auto’, ‘location’ or ‘refresh’) * **$code** (*string*) − HTTP Response code (usually 302 or 303) |
| **Return type** | void |

The first argument can have two types of URI. We can pass full site URL or URI segments to the controller you want to direct.

The second optional parameter can have any of the three values from auto, location or refresh. The default is auto.

The third optional parameter is only available with location redirects and it allows you to send specific HTTP response code.

Example

Create a controller called **Redirect\_controller.php** and save it in**application/controller/Redirect\_controller.php**

<?php

class Redirect\_controller extends CI\_Controller {

public function index() {

/\*Load the URL helper\*/

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

/\*Redirect the user to some site\*/

redirect('http://www.tutorialspoint.com');

}

public function computer\_graphics() {

/\*Load the URL helper\*/

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

redirect('http://www.tutorialspoint.com/computer\_graphics/index.htm');

}

public function version2() {

/\*Load the URL helper\*/

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

/\*Redirect the user to some internal controller’s method\*/

redirect('redirect/computer\_graphics');

}

}

?>

Change the **routes.php** file in **application/config/routes.php** to add route for the above controller and add the following line at the end of the file.

$route['redirect'] = 'Redirect\_controller';

$route['redirect/version2'] = 'Redirect\_controller/version2';

$route['redirect/computer\_graphics'] = 'Redirect\_controller/computer\_graphics';

Type the following URL in the browser, to execute the example.

http://yoursite.com/index.php/redirect

The above URL will redirect you to the tutorialspoint.com website and if you visit the following URL, then it will redirect you to the computer graphics tutorial at tutorialspoint.com.

http://yoursite.com/index.php/redirect/computer\_graphics

CodeIgniter Tutorial for Beginners

By: CodexWorldIn: [CodeIgniter](http://www.codexworld.com/tutorials/codeigniter/)Last Updated: Apr 22, 2016

[ShareTweet](javascript:void(0))

MVC stands for Model View Controller. The model view controller pattern is the most used pattern for today’s world web applications. At present, there is more than a dozen PHP framework based on MVC pattern. The MVC pattern separates an application in 3 modules: Model, View, Controller. The Model is responsible for managing the data. The View is responsible for displaying the data provided by the model in a specific format. The Controller handles the model and view layers to work together.

CodeIgniter is one of the most popular PHP frameworks and it follows the MVC pattern. Web Developer can be able to build full-featured web applications with CodeIgniter.

Today we will describe the CodeIgniter setup process for the beginners. In this tutorial, beginners would be able to learn the CodeIgniter development process from the scratch. Our step by step tutorial helps web developers to learn CodeIgniter quickly and easily.

Also, you can download the demo CodeIgniter project which we have been created at the time of describing this tutorial. You only need to extract the zip file and follow the instructions given in the ReadMe.txt file. If you wish to download the demo Codeigniter project, download it from the *Download Source Code* link at the end of the tutorial.

**CodeIgniter Basic**

* Download the latest version of CodeIgniter from here: <http://www.codeigniter.com/>
* Extract the zip file and rename the folder. For example codeigniter/.
* Move the folder at the localhost.
* If you try to open the home URL (http://localhost/codeigniter/), you’ll see the following screen at the browser.
* Open the application/config/config.php file and follow the below instruction.
  + Set the $config['base\_url'] variable value with your project base URL(http://localhost/codeigniter/)
* If you want to use database table, then open the application/config/database.php file and follow the below instruction.
  + Set $db['default']['hostname'] variable value with your database host name.
  + Set $db['default']['username'] variable value with your database username.
  + Set $db['default']['password'] variable value with your database password.
  + Set $db['default']['database'] variable value with your database name.
* Open the application/config/routes.php file and follow the below instruction.
  + Change $route['default\_controller'] variable value from "welcome" to "home". Or which you want to load the controller by default.
* Go to the application/controllers/ directory and create a PHP file called Home.php. Open this Home.php file and follow the below instruction.
  + At the beginning add the following codes for preventing the direct script access.

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

* + Create a class named **Home** and extends this class from **CI\_Controller**. Also, keep in mind that the class name and file name must be same.

class Home extends CI\_Controller {  
  
}

* Go to the application/views/ directory and create a view file (home\_view.php) for **Home** controller. Open this home\_view.php file and write some HTML code for testing purpose.
* Open the **Home** controller (application/controllers/Home.php) and add the following changes.
  + Create a index() function.
  + Load the view file into the index() function. The $this->load->view() statement is used for render the view.

$this->load->view('home\_view');

* Refresh the homepage (http://localhost/codeigniter/) at the browser, you’ll see the view page as per the given HTML.

**CodeIgniter with Database**

Now we will use the database with the CodeIgniter application. In this tutorial we will fetch the images from the database and display the images into CodeIgniter. Please follow the below steps.

* Create a database at the PHP MyAdmin. Like the database name is "test". Once the database is created, then create a table named "images". Also you can copy the below table SQL and run this SQL to the database.
* CREATE TABLE `images` (
* `id` *int*(**11**) NOT NULL AUTO\_INCREMENT,
* `image` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,
* `created` datetime NOT NULL,
* `modified` datetime NOT NULL,
* *PRIMARY KEY* (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;

* Create a directory(uploads/) into the CodeIgniter root directory and insert some images into this directory. Now insert the images name at the "images" table to the database.
* For database access we have to define the database hostname, database username, database password and database name in the database.php file. Open the application/config/database.php file and set the following values.

$db['default']['hostname'] = 'localhost';  
$db['default']['username'] = 'root';  
$db['default']['password'] = '';  
$db['default']['database'] = 'test';

* For use the database function we need load the CodeIgniter database library. We need to tell Codeigniter to auto-load the database library so we don’t have to ask for it every time we want to do a database query. Open the file application/config/autoload.php, go to the 55 line and make the below changes.
  + Find $autoload['libraries'] variable and set array('database'); value at this variable.

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

* Create a model into the models directory. Follow the below steps:
  + Go to the application/models/ directory and create the file Home\_model.php.
  + Create a class and extend this class from CI\_Model class. The class name should be the same of the file name, means the class name would be Home\_model.
  + Create a function for retrieve the data from the database. For this tutorial purpose we declare the get\_images() function, this function is used for fetch the images from the database.
  + The whole Home\_model.php file codes are given below.

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');  
  
class Home\_model extends CI\_Model {  
    function get\_images()  
    {  
        $query = $this->db->get('images');  
        if($query->num\_rows() > 0){  
            $result = $query->result\_array();  
            return $result;  
        }else{  
            return false;  
        }  
    }  
}

* Open the previously created application/controllers/Home.php file and make the following changes.
  + Load the model: $this->load->model('home\_model')
  + Get images data from the model: $this->home\_model->get\_images()
  + Store the images data into an associative array: $data[‘images’] = $this->home\_model->get\_images();
  + Pass the images data to the view: $this->load->view('home\_view', $data)
  + The whole Home.php file codes are given below.

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');  
  
class Home extends CI\_Controller {  
    public function index()  
    {  
        //load model  
        $this->load->model('home\_model');  
          
        //get data from the database  
        $data['images'] = $this->home\_model->get\_images();  
          
        //load view and pass the data  
        $this->load->view('home\_view', $data);  
    }  
}

* Open the previously created application/views/home\_view.php file and write some codes for display all images which are passed from the controller.

<?php if(!empty($images)): foreach($images as $img): ?>  
 <li><img src="uploads/<?php echo $img['image']; ?>" alt=""></li>  
<?php endforeach; endif; ?>

* The whole home\_view.php file codes are given below.
* <!DOCTYPE html>
* <html lang="en">
* <head>
* <meta charset="utf-8">
* <title>Welcome to my first CodeIgniter application.</title>
* <style type="text/css">
* body {background-color: **#fff**;margin: **40px**;font: **13px**/**20px** normal Helvetica, Arial, sans-serif;color: **#4F5155**;}
* h1 {color: **#fff**;background-color: **#FB4314**;border-bottom: **1px** solid **#D0D0D0**;font-size: **19px**;font-weight: normal;margin: **0** **0** **14px** **0**;padding: **14px** **15px** **10px** **15px**;}
* #container{margin: **10px**;border: **1px** solid **#D0D0D0**;-webkit-box-shadow: **0** **0** **8px** **#D0D0D0**;}
* .gallery{ width:**100%**; float:left; }
* .gallery ul{ margin:**0**; padding:**0**; list-style-type:none;}
* .gallery ul li{ padding:**7px**; border:**2px** solid **#ccc**; float:left; margin:**10px** **7px**; background:none; width:auto; height:auto;}
* </style>
* </head>
* <body>
* <div id="container">
* <h1>Welcome to First CodeIgniter Application.</h1>
* <div id="body">
* <div class="gallery">
* <ul>
* <?php if(!empty($images)): foreach($images as $img): ?>  
                 <li><img src="uploads/<?php echo $img['image']; ?>" alt=""></li>  
              <?php endforeach; endif; ?>
* </ul>
* </div>
* </div>
* </div>
* </body>
* </html>
* Refresh your browser again, homepage URL will show the following screen.