K.T.S.P.Madal's

Hutatma Rajguru Mahavidyalaya, Rajgurunagar

Tal-Khed, Dist.-Pune 410505.

TY.BSc (Computer Science)

Semester-VI

Subject- Web Technologies II

According to new CBCS syllabus w.e.f.2019-2020

Prof.S.V.Patole

Department of Computer Science

Hutatma Rajguru Mahavidyalaya,

Rajgurunagar.

Chapter 5. PHP Framework CodeIgniter

CodeIgniter- Overview, Installing CodeIgniter

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:

- **Free to use**

  It is licensed under MIT license, so it is free to use.

- **Follows MVC Pattern**

  It uses Model-View-Controller which basically separates logic and presentation parts. Request comes to controller, database action is performed through model and output is displayed through views.

  But in normal PHP scripting, every page represents MVC which increases complexity.

- **Light weight**

  It is extremely light-weighted. CodeIgniter core system requires very small library, other libraries may be added upon dynamic request based upon your needs. That is why it is quite fast and light weighted.

- **Generate SEO friendly URLs**

  URLs generated by CodeIgniter are search-engine friendly and clean. It uses a segment based approach rather than standard query based approach.

CodeIgniter Installation

Follow given steps to install CodeIgniter:

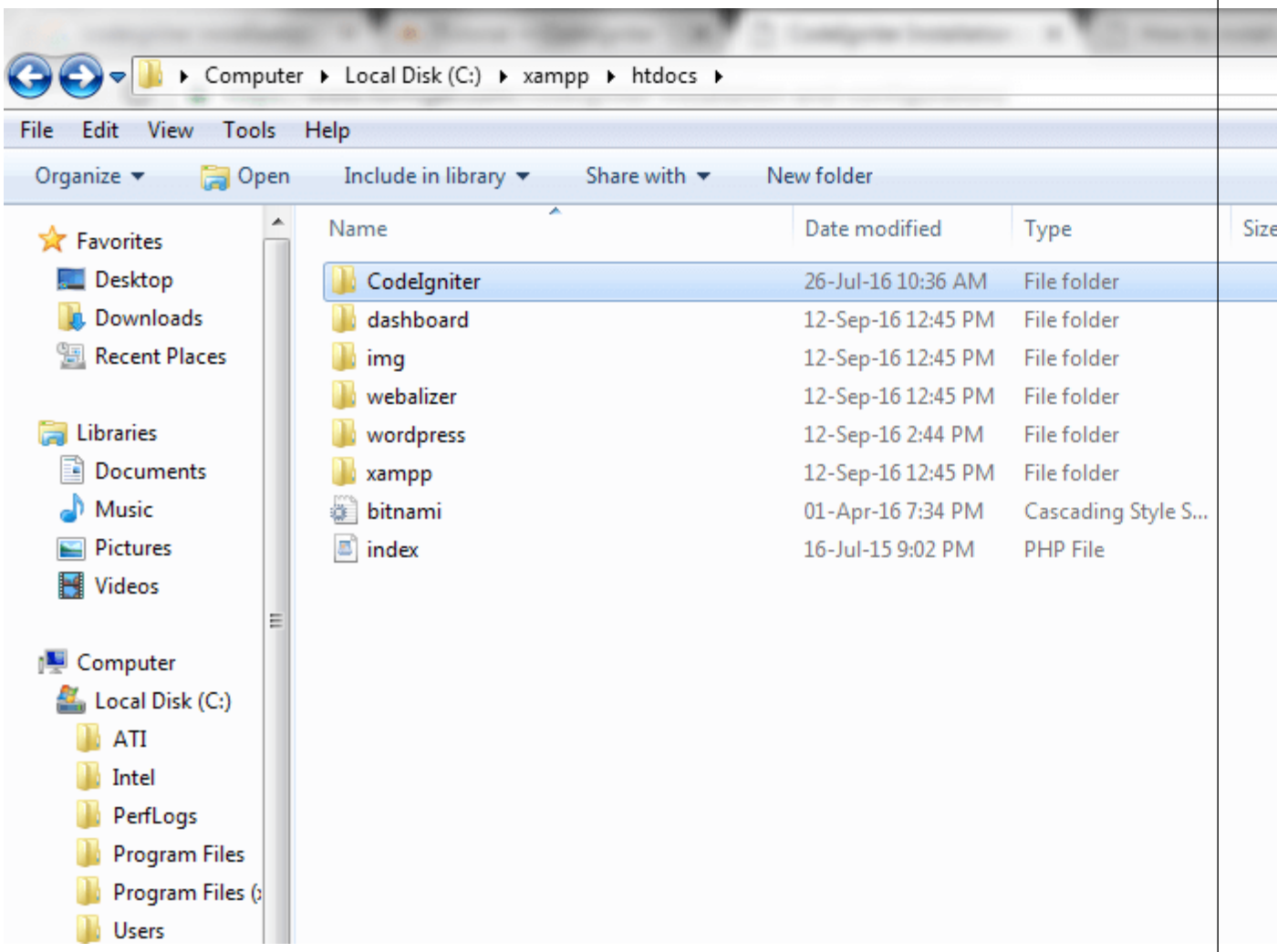
1) Download CodeIgniter from its official website.

Download current version of CodeIgniter from its official website

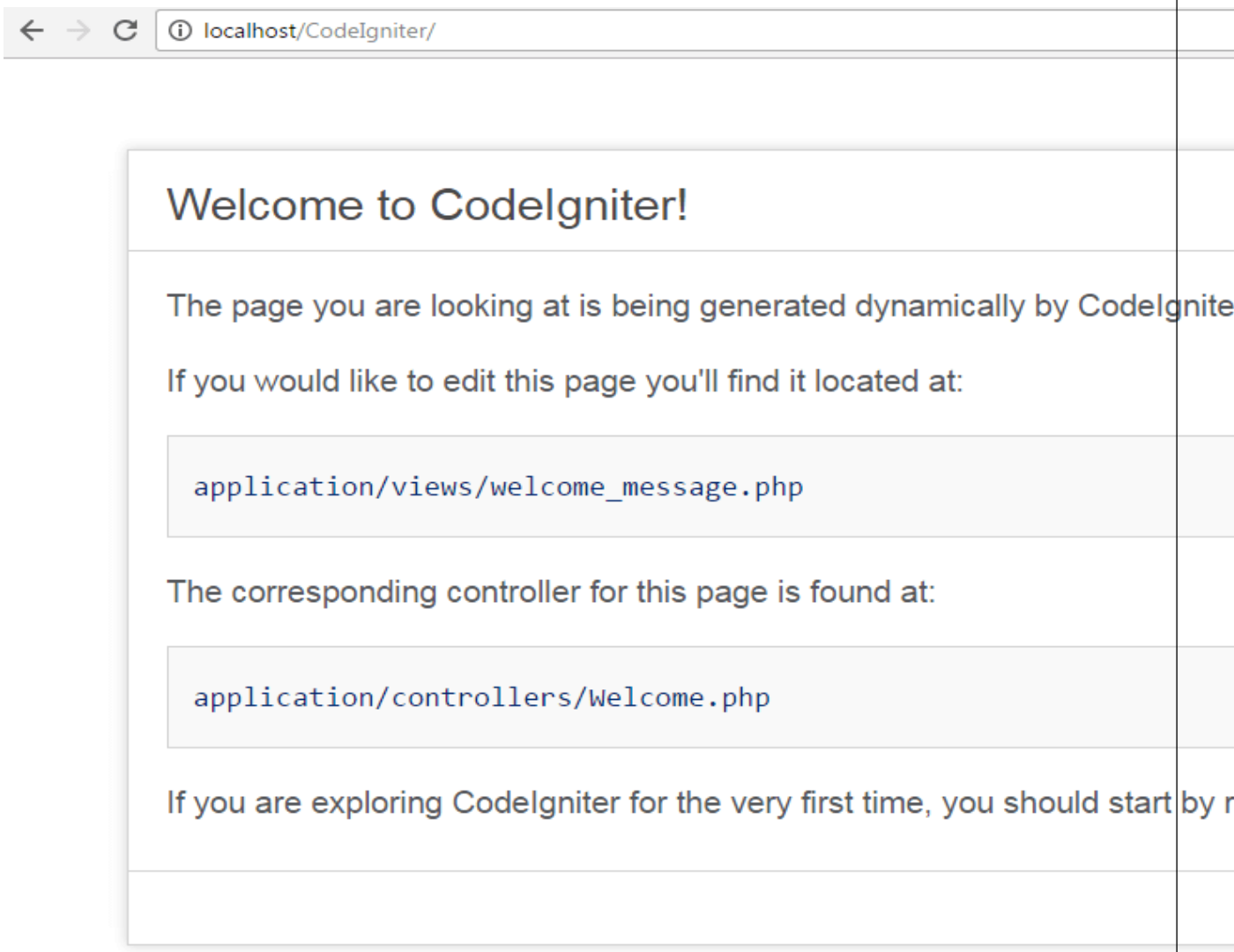
https://www.codeigniter.com

2) Unzip CodeIgniter package.

Downloaded CodeIgniter will be in zip format. Copy it and place it in your htdocs folder. Unzip and rename it. We are naming it as **CodeIgniter.**

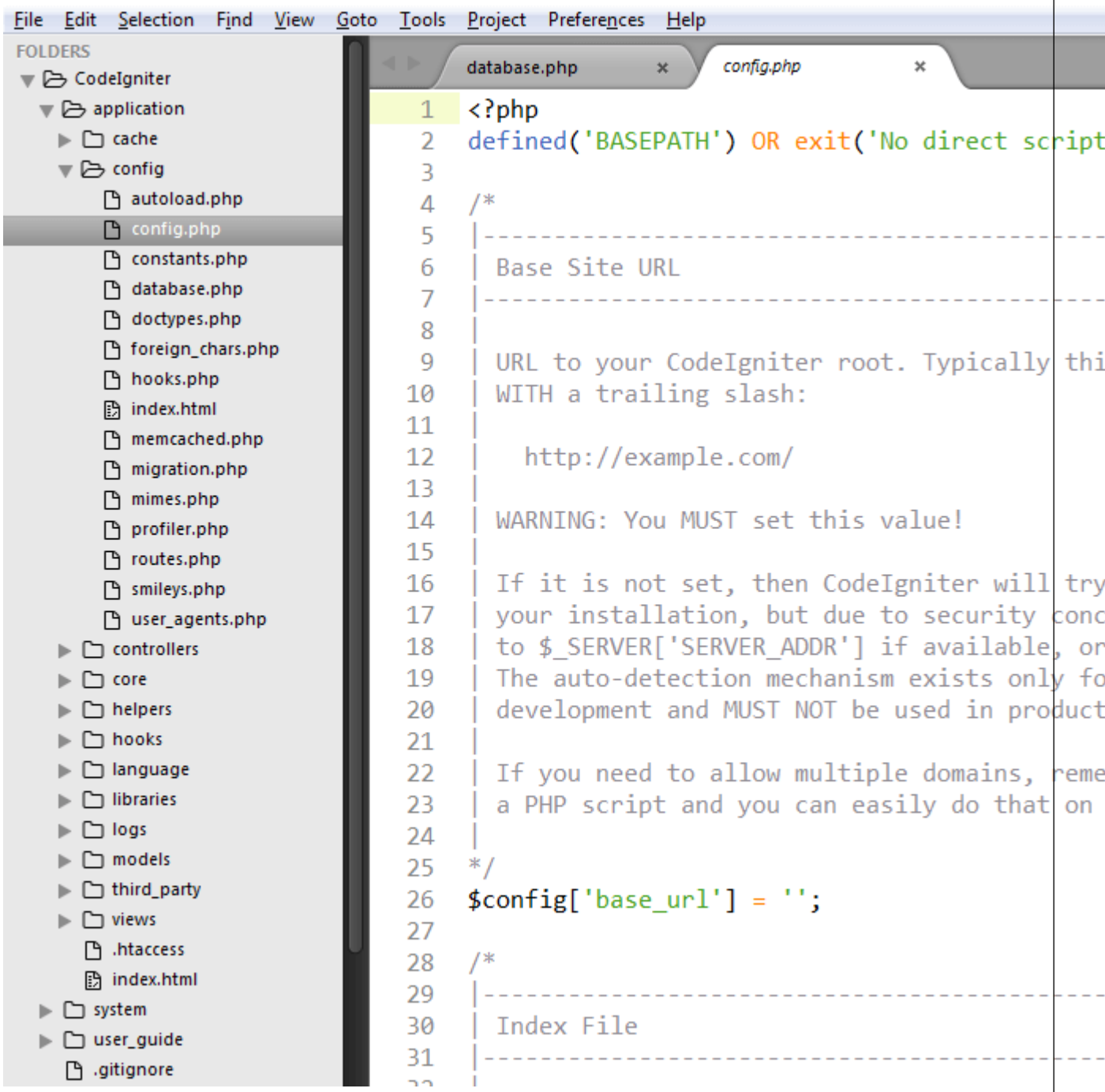


3) CodeIgniter user guide

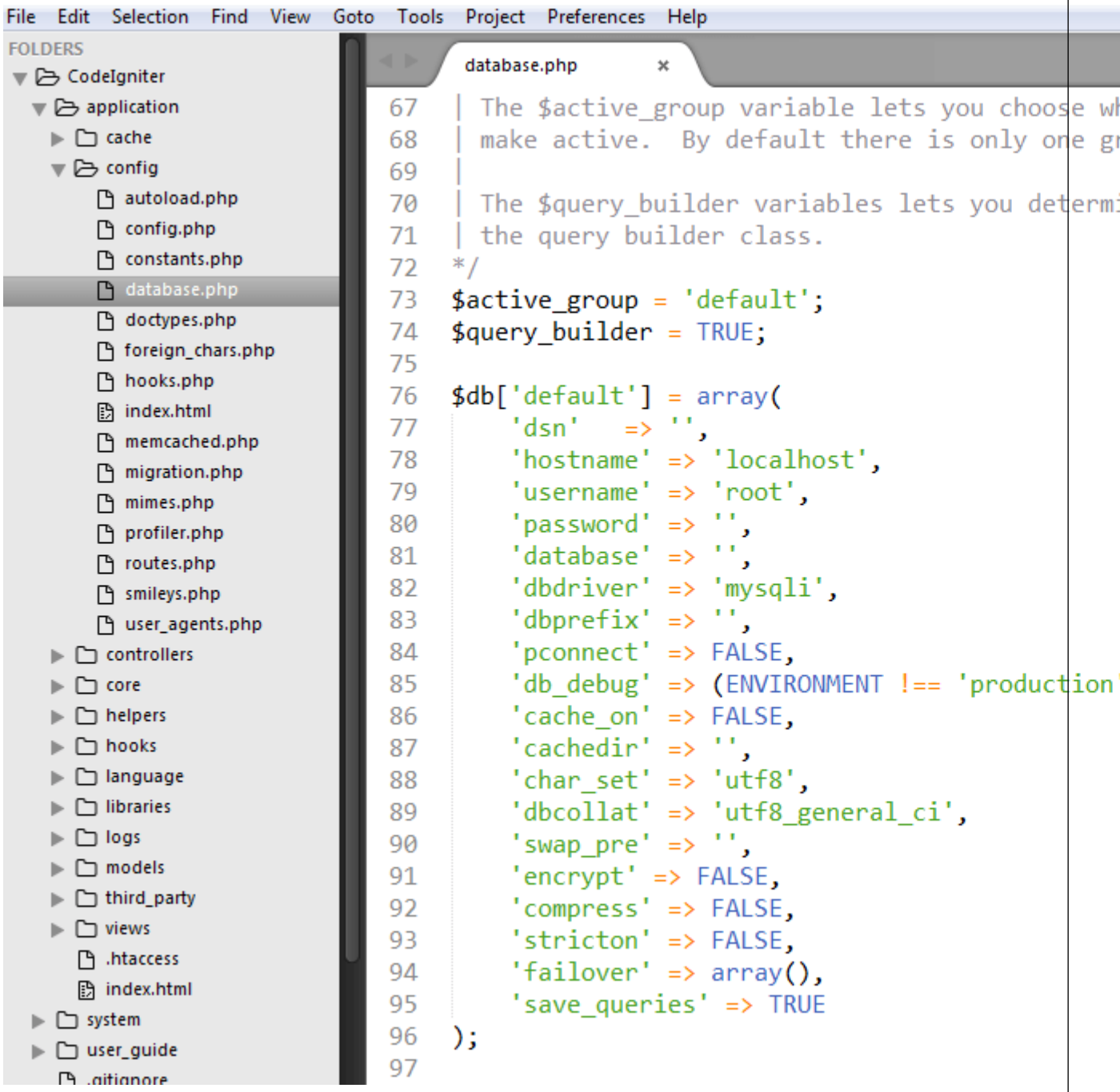


On browser type **localhost/CodeIgniter/** (after localhost type name of your unzipped folder). If the above snapshot page appears then it means your file is successfully installed.

4) Set the base URL in application/config/config.php file with any text editor.



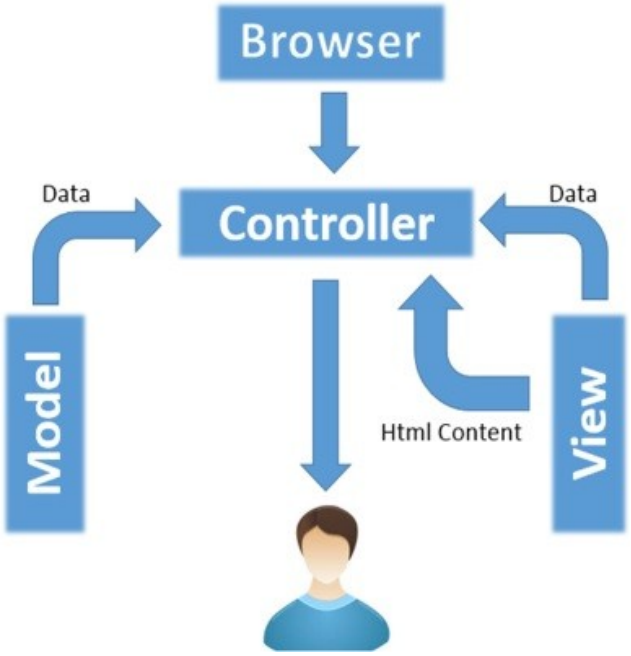
5) You need to establish the connectivity to your database. Go to the path application/config/database.php file.



Look at the above snapshot, fill the details about your database like hostname, username, password and database name.

codeigniter mvc architecture

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.

Basic concept of codeIgniter Libraries

The essential part of a CodeIgniter 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->load->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->load->library(array('email', 'table'));
```

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.

Creating Libraries

CodeIgniter has rich set of libraries, which you can find in **system/libraries** folder but CodeIgniter is not just limited to system libraries, you can create your own libraries too, which can be stored in **application/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
- The class name must start with a capital letter e.g. class Mylibrary
- The name of the class and name of the file must match.

Mylibrary.php

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

  class Mylibrary {

    public function some_function() {
    }
  }

/* End of file Mylibrary.php */
```

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();

Extend the Native Library

Sometimes, you may need to add your own functionality to the library provided by CodeIgniter. CodeIgniter provides facility by which you can extend the native library and add your own functions. To achieve this, you must extend the class of native library class. For example if you want to extend the Email library then it can be done as shown below −

```
Class MY_Email extends CI_Email {
}
```

Here, in the above example, MY_Email class is extending the native library's email class CI_Email. This library can be loaded by the standard way of loading email library. Save the above code in file My_Email.php

Replace the Native Library

In some situations, you do not want to use the native library the way it works and want to replace it with your own way. This can be done by replacing the native library. To achieve this, you just need to give the same class name as it is named in native library. For example, if you want to replace the **Email class**, then use the code as shown below. Save your file name with **Email.php** and give a class name to **CI_Email**.

Email.php

```
Class CI_Email {
}
```

Helper Function:

Helpers are the reusable code in codeignitor like libraries. The only difference is that libraries are collection of classes whereas helper is defined as individual independent set of functions. Helper functions need to be loaded before using it. We can find all the helpers in codeignitor documentation Codeignitor Helpers and that can be used depends on kind of requirement.

Create a controller **users.php** and then we can use below code to use helper.

```
<?php

defined('BASEPATH') OR exit('No direct script access allowed');

class Users extends CI_Controller {

        public function index() {


                // Load form helper

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

        }

}

?>
```

If we need to load multiple helpers, we can create an array and then we can define all the helper's name in that array.

```
$this->load->helper(array('form', 'email', 'url'));
```

Custom Helpers: Codeignitor has already a lot of built-in helpers but if we need to create a function which is not in the helper then we can create our own custom helper and use it in the same way like inbuilt helpers. Inbuilt helpers are available in system folder but custom helper needs to be created in application/helpers folder. Create a file **abc_helper.php** in application/helpers folder. Below is the example of creating functionality in our custom helper.

```
<?php
    function test() {
      echo "Custom helper for codeignitor";
    }
?>
```

Custom helpers can be used just like inbuilt helpers in the controller. So in our **users.php** controller use the code below to check this.

Controller: users.php

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Users extends CI_Controller {
    public function index() {

        // Load custom helper
        $this->load->helper('abc');
        test();
       }
  }
```

?

Output:
Custom helper for codeignitor

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.

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
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
```

```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**

```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';
```

Cookies_management in CodeIgniter

Cookie is a small piece of data sent from web server to store on client's computer. CodeIgniter 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().

Example

Create a controller called **Cookie_controller.php** and save it at **application/controller/Cookie_controller.php**

```
<?php
  class Cookie_controller extends CI_Controller {

    function __construct() {
      parent::__construct();
```

```
        $this->load->helper(array('cookie', 'url'));
    }

    public function index() {
        set_cookie('cookie_name','cookie_value','3600');
        $this->load->view('Cookie_view');
    }

    public function display_cookie() {
        echo get_cookie('cookie_name');
        $this->load->view('Cookie_view');
    }

    public function deletecookie() {
        delete_cookie('cookie_name');
        redirect('cookie/display');
    }

  }
?>
```

Create a view file called **Cookie_view.php** and save it at **application/views/Cookie_view.php**

```
<!DOCTYPE html>
<html lang = "en">

  <head>
    <meta charset = "utf-8">
    <title>CodeIgniter View Example</title>
  </head>

  <body>
    <a href = 'display'>Click Here</a> to view the cookie.<br>
    <a href = 'delete'>Click Here</a> to delete the cookie.
  </body>

</html>
```

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['cookie'] = "Cookie_controller";
$route['cookie/display'] = "Cookie_controller/display_cookie";
$route['cookie/delete'] = "Cookie_controller/deletecookie";
```

After that, you can execute the following URL in the browser to execute the example.

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

It will produce an output as shown in the following screenshot.