WEB DEVELOPMENT FRAMEWORK

(P22A2WDF)

UNIT 2: Overview of Codeigniter

1. Overview of Codeigniter

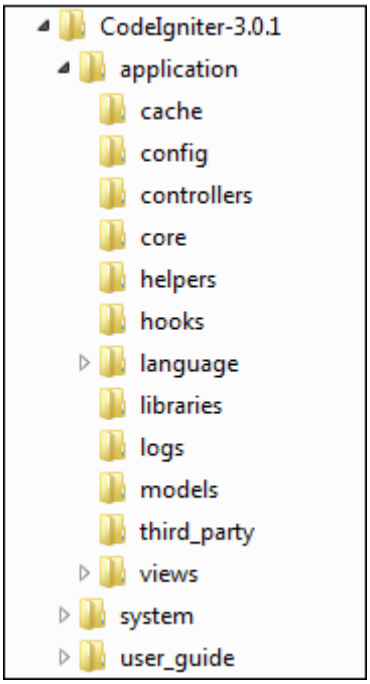
- CodeIgniter is an application development framework with small footprints which makes it much faster than other frameworks.
- It is one of the oldest frameworks with faster and high performance.
- We can develop projects much faster than a scratch, as it provides a large set of libraries, simple interface, and logical structure to access these libraries.

Key Features of CodeIgniter

- **Lightweight & Fast** – One of the fastest PHP frameworks due to its minimal core system.
- **MVC Architecture** – Separates logic, presentation, and data, making code more maintainable.
- **Built-in Security Features** – Protection against XSS, SQL Injection, and more.
- **Easy Database Management** – Supports multiple databases with an Active Record pattern.
- **Flexible and Extendable** – Can be customized with libraries and helpers.
- **Minimal Configuration** – Works with almost no configuration; just install and start coding.
- **Error Handling** – Provides a simple user-friendly error logging system.
- **SEO-Friendly URLs** – Uses clean and human-readable URLs.
- **Supports Third-Party Libraries** – Easily integrates external libraries and APIs.
- **Excellent Documentation** – Comes with clear and well-structured official documentation.

2. Codeigniter Directory structure

The image given below shows the directory structure of the CodeIgniter.



CodeIgniter directory structure is divided into 3 folders –

- Application
- System
- User_guide

Application

As the name indicates 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.
- **Core** – This folder will contain base class of your application.

- **Helpers** – In this folder, you can put helper class of your application.
- **Hooks** – The files in this folder provide a means to tap into and modify the inner workings of the framework without hacking the core files.
- **Language** – This folder contains language related files.
- **Libraries** – This folder contains files of the libraries developed for your application.
- **Logs** – This folder contains files related to the log of the system.
- **Models** – The database login will be placed in this folder.
- **Third_party** – In this folder, you can place any plugins, which will be used for your application.
- **Views** – Application's HTML files will be placed in this folder.

System

This folder contains CodeIgniter core codes, 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 CodeIgniter code of consequence, organized into various folders –

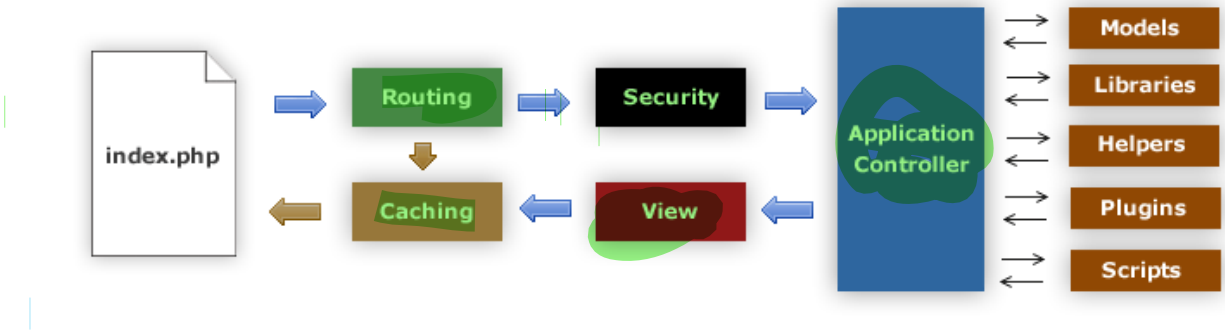
- **Core** – This folder contains CodeIgniter's core class. Do not modify anything here. All of your work will take place in the application folder. Even if your intent is to extend the CodeIgniter core, you have to do it with hooks, and hooks live in the application folder.
- **Database** – The database folder contains core database drivers and other database utilities.
- **Fonts** – The fonts folder contains font related information and utilities.
- **Helpers** – The helpers folder contains standard CodeIgniter helpers (such as date, cookie, and URL helpers).
- **Language** – The language folder contains language files. You can ignore it for now.
- **Libraries** – The libraries folder contains standard CodeIgniter libraries (to help you with e-mail, calendars, file uploads, and more). You can create your own libraries or extend (and even replace) standard ones, but those will be saved in the **application/libraries** directory to keep them separate from the standard CodeIgniter libraries saved in this particular folder.

User_guide

This is your user guide to CodeIgniter. It is basically, the offline version of user guide on CodeIgniter website. Using this, one can learn the functions of various libraries, helpers and classes. It is recommended to go through this user guide before building your first web app in CodeIgniter.

Beside these three folders, there is one more important file named "**index.php**". In this file, we can set the **application environment and error level** and we can define system and application folder name. It is recommended, not to edit these settings if you do not have enough knowledge about what you are going to do.

3. Application flow of Codeigniter



- 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.

4. Database Configuration

CodeIgniter has a config file that lets you store your database connection values (username, password, database name, etc.). The config file is located at application/config/database.php. You can also set database connection values for specific environments by placing database.php in the respective environment config folder.

database.php (codeigniter) - Sublime Text (UNREGISTERED)

Project Preferences Help

HomeController.php | database.php

```php
| the query builder class.
*/
$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'database_ci',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

❖ $active_group

- Specifies which database connection settings to use.
- You can define multiple database configurations and switch between them.

❖ **$query_builder**

$query_builder = TRUE;

- Enables Query Builder (Active Record), which makes it easier to construct queries without writing raw SQL.

❖ **dsn (Data Source Name)**

'dsn' => '',

- Used for PDO-based connections (not commonly used in MySQL).
- If empty, CodeIgniter uses traditional MySQL connections.

❖ **hostname**

'hostname' => 'localhost',

- Specifies the database server location.

Common values:

- 'localhost' → Local database
- '127.0.0.1' → Alternative local IP
- 'db.example.com' → Remote database server

❖ **username & password**

- The database login credentials.

❖ **database**

'database' => 'my_database',

- The database name to connect to.

❖ **dbdriver**

' dbdriver' => 'mysqli',

- The database type.

Available options:

- 'mysqli' → MySQLi (recommended)

- 'pdo' → PHP Data Objects
- 'postgre' → PostgreSQL
- 'sqlite3' → SQLite 3
- 'sqlsrv' → Microsoft SQL Server

❖ **dbprefix (Table Prefix)**

'dbprefix' => '',

- Adds a prefix to all table names (useful for shared databases).
- Example:

  'dbprefix' => 'ci_',

- Table users becomes ci_users.

❖ **pconnect (Persistent Connection)**

'pconnect' => FALSE,

- If TRUE, reuses the same database connection for multiple requests.
- Recommended: FALSE to prevent connection issues.

❖ **db_debug (Enable Debugging)**

'db_debug' => (ENVIRONMENT !== 'production'),

- Shows database errors when debugging.
- Disabled in production mode for security.

❖ **cache_on (Enable Query Caching)**

'cache_on' => FALSE,

- If TRUE, caches queries to improve performance.

❖ **cachedir (Query Cache Directory)**

'cachedir' => '',

- Specifies the directory for query caching.
- Example:

'cachedir' => 'application/cache/db',

❖ **char_set & dbcollat (Character Set & Collation)**

'char_set' => 'utf8',

'dbcollat' => 'utf8_general_ci',

- Defines encoding for storing data.
- Recommended for multilingual applications:
- **Collation Description**

utf8_general_ci Case-insensitive, fast, less accurate

utf8_unicode_ci Case-insensitive, better for multilingual support

utf8_bin Case-sensitive (e.g., A ≠ a)

❖ **swap_pre (Table Prefix Swapping)**

'swap_pre' => '',

- Allows switching table prefixes dynamically (rarely used).

❖ **encrypt (SSL Encryption)**

'encrypt' => FALSE,

- If TRUE, encrypts the connection for security.
- Used for secure remote database connections.

❖ **compress (Compressed Connections)**

'compress' => FALSE,

- If TRUE, enables compressed MySQL connections to reduce network load.

❖ **stricton (Strict Mode for MySQL)**

'stricton' => FALSE,

- If TRUE, enforces strict SQL mode, improving data integrity.

❖ **failover (Alternative Database Connections)**

```
'failover' => array(),
```

- Defines backup database connections if the primary database fails.
- Example:

```
'failover' => array(
  array(
 'hostname' => 'backup-db.example.com',
 'username' => 'backup_user',
 'password' => 'backup_password',
 'database' => 'backup_database',
 'dbdriver' => 'mysqli',
  )
),
```

❖ **save_queries (Store Executed Queries)**

```
'save_queries' => TRUE,
```

- If **TRUE**, stores the last executed SQL queries.
- Use for debugging:

```
print_r($this->db->last_query());
```

5. Configuration Files

Some of the key configuration files in **application/config/** are:

1. **config.php**
   - Main configuration file that contains base settings such as:
     - $config['base_url'] → Set the base URL of your application.
     - $config['index_page'] → Name of the index file (e.g., index.php).
     - $config['encryption_key'] → Secret key for encryption.
     - Other system settings (hooks, logging, session, etc.).
2. **database.php**
   - Stores database connection settings such as:

```
$db['default'] = array(
   'dsn'   => '',
   'hostname' => 'localhost',
   'username' => 'root',
   'password' => '',
   'database' => 'your_database',
   'dbdriver' => 'mysqli',
   'dbprefix' => '',
   'pconnect' => FALSE,
   'db_debug' => (ENVIRONMENT !== 'production'),
);
```

3. **routes.php**
   - Defines the URL routing:

```
$route['default_controller'] = 'welcome';
$route['404_override'] = '';
$route['translate_uri_dashes'] = FALSE;
```

4. **autoload.php**
   - Specifies libraries, helpers, and models to be loaded automatically:

```
$autoload['libraries'] = array('database', 'session',
'form_validation');
$autoload['helper'] = array('url', 'form');
```

5. **constants.php**
   - Used to define global constants:

     define('SITE_NAME', 'My Website');
     define('UPLOAD_PATH', '/uploads/');

6. **Perform basic CRUD (Create, Read, Update, Delete) operations using the Query Builder class of codeigniter. Provide examples for each operation.**

Crudcont.php

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

class Crudcont extends CI_Controller{

    public function index()
    {

    }

    public function add_user() {
        $this->load->model('crudmodel');

        $data = array(
            'name' => 'John Doe',
            'email' => 'john@example.com',
            'phone' => '1234567890'
        );

        if ($this->crudmodel->insert_user($data)) {
            echo "User added successfully!";
        } else {
            echo "Failed to add user.";
        }
    }


    public function list_users() {
        $this->load->model('Crudmodel');
        $users = $this->Crudmodel->get_users();

        foreach ($users as $user) {
            echo "<br>ID: " . $user->id ;
            echo "<br> Name: " . $user->name;
```

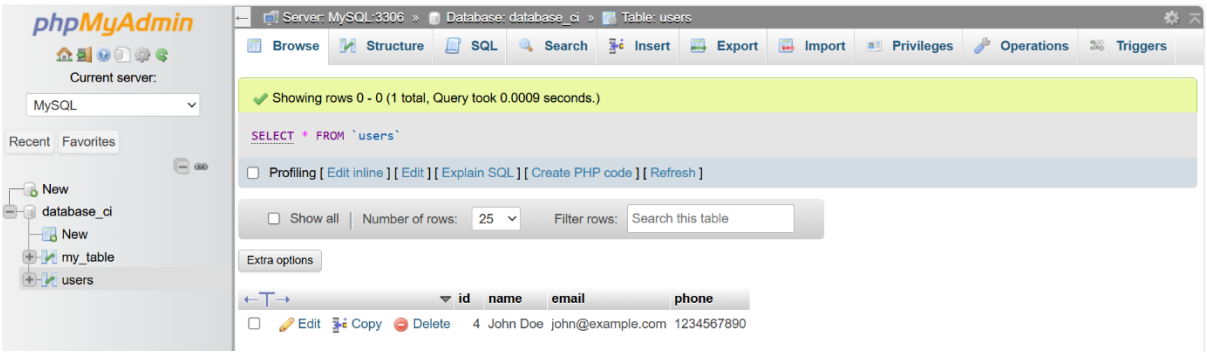
```
        }
    }

    public function edit_user($id) {
        $this->load->model('Crudmodel');

        $data = array(
            'name' => 'Jane Doe',
            'email' => 'jane@example.com'
        );

        if ($this->Crudmodel->update_user($id, $data)) {
            echo "User updated successfully!";
        } else {
            echo "Update failed.";
        }
    }

    public function remove_user($id) {
        $this->load->model('Crudmodel');

        if ($this->Crudmodel->delete_user($id)) {
            echo "User deleted successfully!";
        } else {
            echo "Delete failed.";
        }
    }


}
```

Crudmodel.php

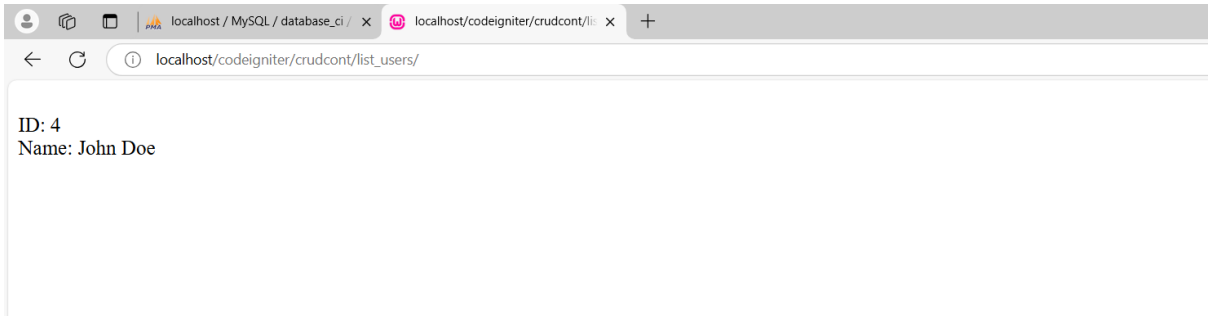
```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');


class Crudmodel extends CI_Model{

    public function insert_user($data)
    {
        return $this->db->insert('users', $data);
    }

    public function get_users() {
        return $this->db->get('users')->result();
    }
```

```
    public function update_user($id, $data)
    {
        $this->db->where('id', $id);
        return $this->db->update('users', $data);
    }

        public function delete_user($id)
        {
    $this->db->where('id', $id);
    return $this->db->delete('users');
}


}
```

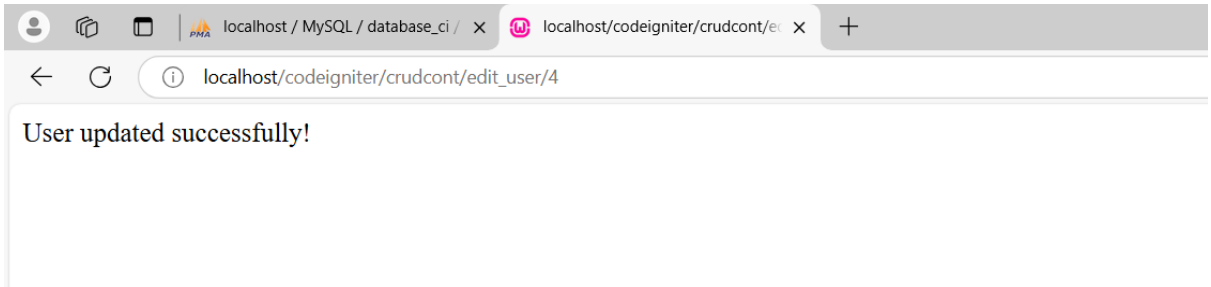
Adding user detail into the database

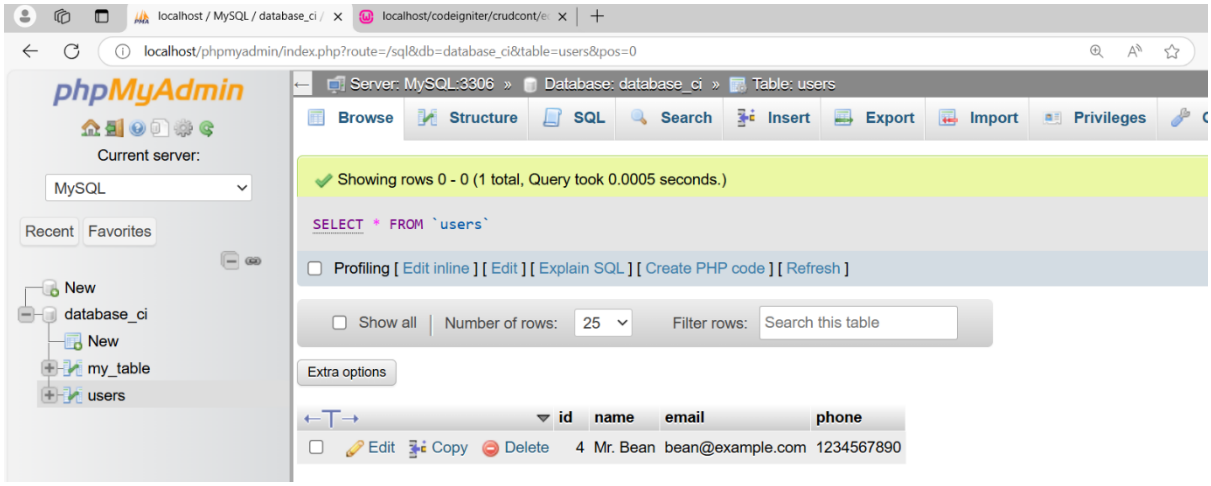


Reading user detail from the database

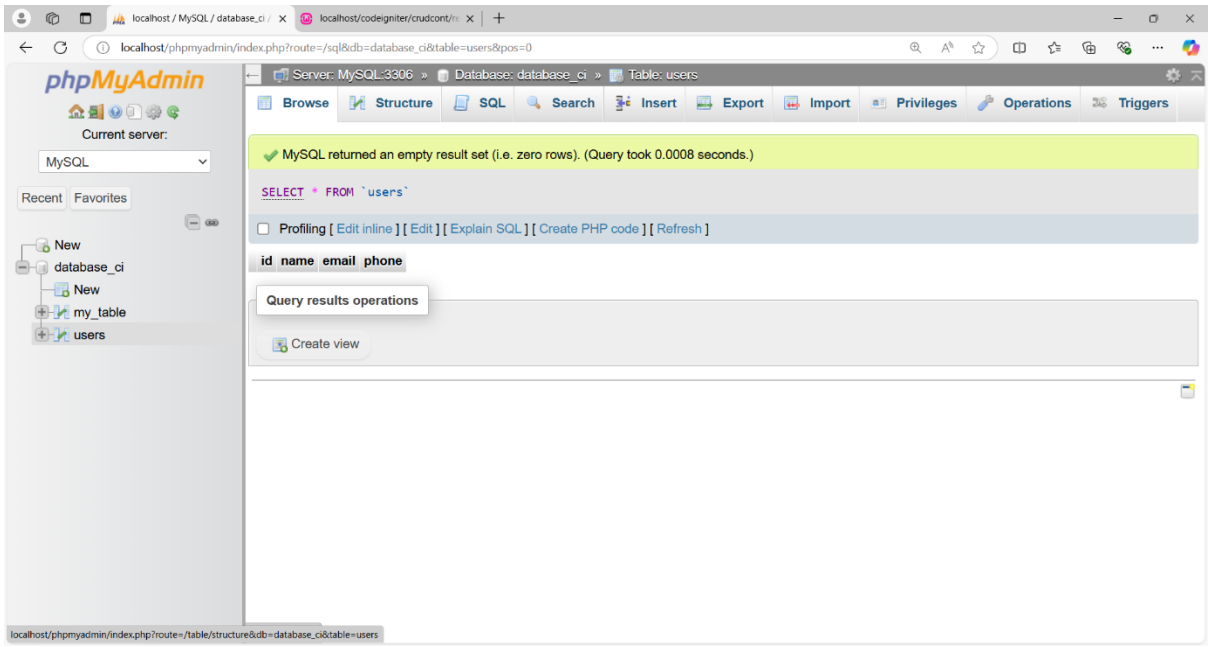


updating user detail into the database





Deleting user detail from the database



Operation	Method	Description
Create	$this->db->insert()	Insert data into the database
Read	$this->db->get()	Retrieve data from the database
Update	$this->db->update()	Modify existing records
Delete	$this->db->delete()	Remove records from the database

7. Simple program to work with MVC

Step 1: Model - User_model.php (Location: application/models/User_model.php)

```
<?php
class User_model extends CI_Model {

    // Fetch all users from the database
  public function get_users() {
    $query = $this->db->get('users');
    return $query->result();
  }
}
?>
```

Step 2: Controller - Users.php (Location: application/controllers/Users.php)

```
<?php
class Users extends CI_Controller {

    public function index() {
    $data['users'] = $this->User_model->get_users();
    $this->load->view('user_list', $data);
  }
}
?>
```

Step 3: View - user_list.php (Location: application/views/user_list.php)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>User List</title>
</head>
<body>
  <h1>List of Users</h1>
  <ul>
    <?php foreach ($users as $user): ?>
      <li><?php echo $user->name; ?> - <?php echo $user->email; ?></li>
    <?php endforeach; ?>
  </ul>
</body>
</html>
```