3.1 PHP with MySQL

- PHP is a powerful server-side scripting language, and MySQL is a popular open-source relational database.
- Together, they are widely used to build dynamic and data-driven websites.

What is MySQL?

- **MySQL** is a **Relational Database Management System (RDBMS)**.
- It stores data in **tables** consisting of **rows** and **columns**.
- Commonly used with PHP to store, retrieve, and manage data.

PHP and MySQL

PHP can **connect to MySQL databases**, **execute SQL queries**, and **interact with data** (insert, read, update, delete).

The **mysqli** (MySQL Improved) extension is used for this purpose. It supports.

- Procedural and Object-Oriented style
- Security features like **prepared statements**

Commonly Used mysqli Functions – PHP + MySQL

Function	Description	Syntax	Example
mysqli_connect()	Connect to MySQL database server	mysqli_connect(host, user, password, database)	$conn = mysqli_connect ("localhost", "root", "", "studentDB");
mysqli_select_db()	Selects a database after connecting	mysqli_select_db(connection, db_name)	mysqli_select_db($conn, "studentDB");
mysqli_query()	Executes SQL queries	mysqli_query(connection, sql)	$result = mysqli_query($conn,

	(SELECT, INSERT, etc.)		"SELECT * FROM students");
mysqli_fetch_a assoc()	Fetches result row as associative array	mysqli_fetch_assoc(result)	$row=mysqli_fetch_assoc($result);
mysqli_num_r rows()	Returns number of rows in result set	mysqli_num_rows(result)	if(mysqli_num_rows($result) > 0) { ... }
mysqli_error()	Returns last error from connection	mysqli_error(connection)	echo mysqli_error($conn);
mysqli_close()	Closes the database connection	mysqli_close(connection)	mysqli_close($conn);

3.1.1 Connecting to Databases using mysqli

This involves establishing a connection between your PHP script and the MySQL database using mysqli_connect() function.
You need:

- **Hostname (usually "localhost")**
- **Username (usually "root")**
- **Password (default is blank in local server)**
- **Database name**

3.1.2 Creating Databases and Tables

- Database: A container that holds tables and data.
- Table: Structure to store data in rows and columns.
  Using SQL commands like CREATE DATABASE and CREATE TABLE, you can define where and how data will be stored.

3.1.3 Executing CRUD Operations

CRUD stands for:

- Create → INSERT data into a table
- Read → SELECT data from a table
- Update → UPDATE existing data
- Delete → DELETE data from a table
- PHP uses SQL queries to perform these operations with the help of mysqli_connect("hostname", "username", "password", "database_name");mysqli_query().

3.1.4 Using Clauses: WHERE, ORDER BY, LIMIT

These are SQL clauses that help refine data retrieval:

- WHERE: Filter records based on condition
- ORDER BY: Sort records (ascending/descending)
- LIMIT: Restrict number of rows returned

Practical Implementations:

Connecting to Databases Using mysqli

To use a MySQL database with PHP, we must connect to it using the mysqli_connect() function.

Syntax:

```
mysqli_connect("hostname", "username", "password", "database_name");
```

- **hostname:** Usually "localhost" when using XAMPP/WAMP
- **username:** "root" by default
- **password:** Usually blank ("") for local server
- **database_name:** The name of your database

Example:

```
<?php
$conn = mysqli_connect("localhost", "root", "", "studentDB");

if (!$conn) {
```

```
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully!";
?>
```

INSERT Example:

```
<?php
$conn = mysqli_connect("localhost", "root", "", "studentDB");

$sql = "INSERT INTO students (name, email, age)
        VALUES ('Rahul Mehta', 'rahul@gmail.com', 21)";
if (mysqli_query($conn, $sql)) {
    echo "Record inserted successfully!";
} else {
    echo "Error: " . mysqli_error($conn);
}
?>
```

SELECT Example:

```
<?php
$conn = mysqli_connect("localhost", "root", "", "studentDB");


$sql = "SELECT * FROM students";
$result = mysqli_query($conn, $sql);


while($row = mysqli_fetch_assoc($result)) {
    echo "Name: " . $row["name"] . " - Email: " . $row["email"] . "<br>";
}
?>
```

UPDATE Example:

```
<?php
$conn = mysqli_connect("localhost", "root", "", "studentDB");
```

```
$sql = "UPDATE students SET age = 22 WHERE name = 'Rahul Mehta'";
if (mysqli_query($conn, $sql)) {
    echo "Record updated successfully!";
} else {
    echo "Error updating record: " . mysqli_error($conn);
}
?>
```

4. DELETE Example:

```
<?php
$conn = mysqli_connect("localhost", "root", "", "studentDB");

$sql = "DELETE FROM students WHERE name = 'Rahul Mehta'";
if (mysqli_query($conn, $sql)) {
    echo "Record deleted successfully!";
} else {
    echo "Error deleting record: " . mysqli_error($conn);
}
?>
```

3.1.4 Using Clauses: WHERE, ORDER BY, LIMIT

1. WHERE Clause

Used to filter records based on a condition.

Example:

```
$sql = "SELECT * FROM students WHERE age > 20";
```

2. ORDER BY Clause

Used to sort results either ascending (ASC) or descending (DESC).

Example:

```
$sql = "SELECT * FROM students ORDER BY age DESC";
```

3. LIMIT Clause

Used to limit the number of records returned.

Example:

```
$sql = "SELECT * FROM students LIMIT 3";
```

Combined Example:

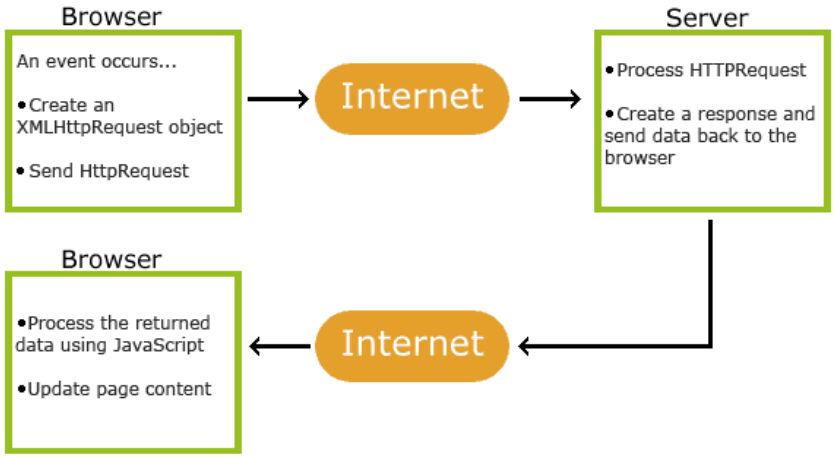
```
$sql = "SELECT * FROM students WHERE age > 18 ORDER BY age DESC LIMIT 2";
```

This query will return top 2 students whose age is more than 18, sorted from highest to lowest age.

3. 2 AJAX for Backend Integration

- AJAX (Asynchronous JavaScript and XML)
- AJAX is not a programming language.
- It allows web pages to send/receive data from the server in the background without reloading the entire page.
- AJAX is not a programming language—it's a technique that uses:
  - JavaScript → to send/receive data.
  - XMLHttpRequest / Fetch API → to communicate with the server.
  - PHP (or any backend language) → to process the request and return a response.
- Used in:
  - Search suggestions (like Google search box).
  - Submitting forms without refreshing.
  - Loading comments, likes, etc. dynamically.
- AJAX just uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)
- AJAX facilitates backend integration by enabling web pages to exchange data with a server asynchronously, without requiring a full page reload.
- This improves user experience by allowing for dynamic content updates and enhanced interactivity.

- AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.



- Synchronous vs Asynchronous
  - Synchronous: Request → Wait for server → Page reloads.
  - Asynchronous (AJAX): Request → Page doesn't reload → Only response updates specific part of the page.
- Basic AJAX Workflow
  - User action (like button click).
  - JavaScript sends AJAX request to PHP file.
  - PHP processes data and sends back a response.
  - JavaScript updates webpage dynamically.

  Flow for below code –
  1. An event occurs in a web page (the page is loaded, a button is clicked)
  2. An XMLHttpRequest object is created by JavaScript
  3. The XMLHttpRequest object sends a request to a web server
  4. The server processes the request
  5. The server sends a response back to the web page
  6. The response is read by JavaScript
  7. Proper action (like page update) is performed by JavaScript

Index.php

```
<!DOCTYPE html>
        <html>
        <script>
        function loadDoc() {
          var xhttp = new XMLHttpRequest();
          xhttp.onreadystatechange = function() {
```

```
            if (this.readyState == 4 && this.status == 200) {
             document.getElementById("demo").innerHTML = this.responseText;
            }
           };
           xhttp.open("GET", "Hello.php", true);
           xhttp.send();
         }
         </script>
         <body>

         <div id= “demo">
           <h2>AJAX button click event to load data from function</h2>
           <button type="button" onclick="loadDoc()">Submit</button>
         </div>

         </body>
</html>
```

Here,

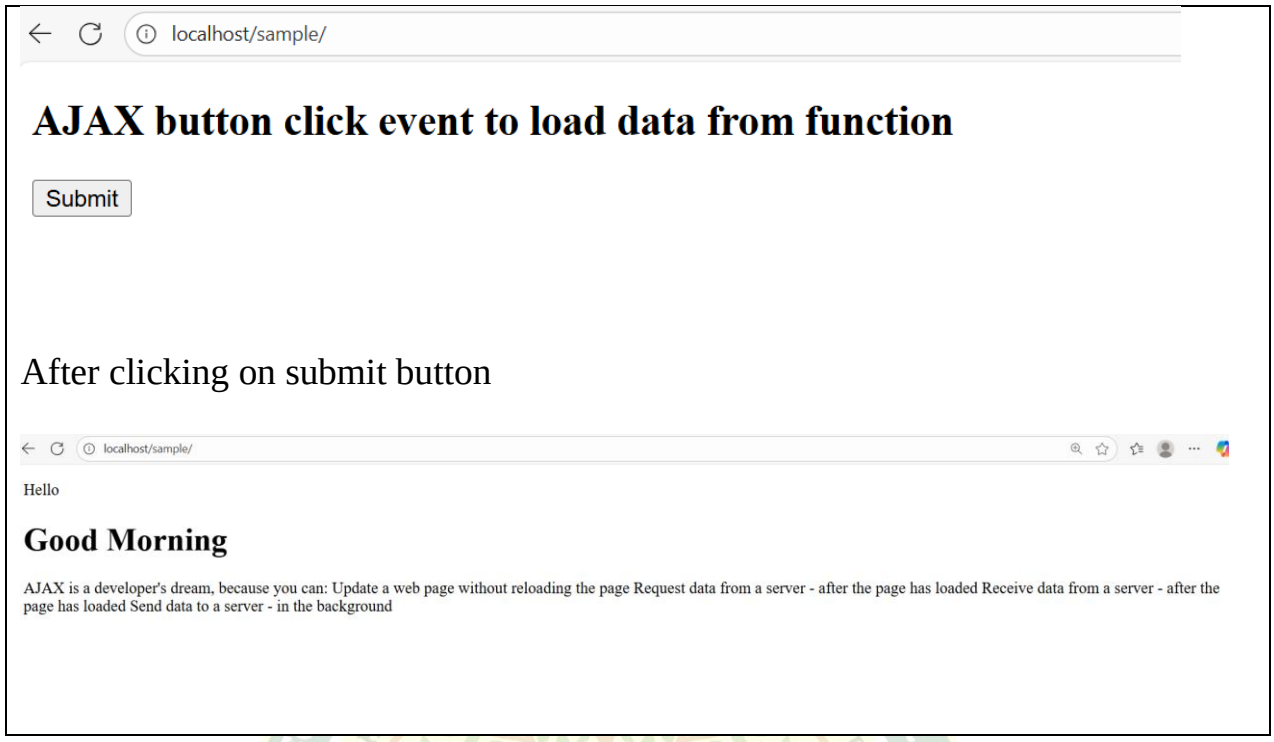
- section is used to display information from a server.
- <button> calls a function (if it is clicked)

Hello.php

```
         <p>Hello</p>
         <h1>Good Morning</h1>
         <p>
         AJAX is a developer's dream, because you can:

         Update a web page without reloading the page
         Request data from a server - after the page has loaded
         Receive data from a server - after the page has loaded
         Send data to a server - in the background
</p>
```

Output



The XMLHttpRequest Object

- The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
  var xhttp = new XMLHttpRequest();
- AJAX makes web apps more interactive and faster.
- It works using JavaScript + PHP (or any backend).
  - Always check:
  - readyState == 4 → request finished.
  - status == 200 → success.
  - Use GET for fetching data and POST for sending form data.

AJAX - Send a Request to a Server

- The XMLHttpRequest object is used to exchange data with a server.
- To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object-
  xhttp.open("GET", "hello.txt", true);
  xhttp.send();

Method	Description
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the server (file) location

	async: true (asynchronous) or false (synchronous)
Send()	Sends the request to the server (used for GET)
send(*string*)	Sends the request to the server (used for POST)

Method

- GET is simpler and faster than POST, and can be used in most cases.
- Condition in which post is more suitable –
    - A cached file is not an option (update a file or database on the server).
    - Sending a large amount of data to the server (POST has no size limitations).
    - Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

Simple GET request

```
xhttp.open("GET", "demo_get.php", true);
xhttp.send();
```

Add unique ID to the URL (Without it you may get a cached result)

```
xhttp.open("GET", "demo_get.asp?t=" + Math.random(), true);
xhttp.send();
```

If you want to send information with the GET method, add the information to the URL:

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford
", true);
xhttp.send();
```

3.2.3 Real-time search functionality

- Real-time search means as the user types in a search box, results appear instantly without reloading the page.

- Example: Google search suggestions.

- Why use AJAX for Search?
  - Without AJAX → You must submit the form and reload the page every time.
  - With AJAX → The search box automatically fetches results from the server as you type.
  - Benefits: Fast, interactive, better user experience.
- **How It Works (Flow)**
  - User types in search box.
  - JavaScript (AJAX) sends the typed text to a PHP file.
  - PHP searches the database (MySQL) for matching records.
  - PHP returns results.
  - AJAX updates the page with results instantly.

Connect_db.php

```
<?php
$ser = "localhost";
$host = "root";
$pwd = "";

$con = mysqli_connect($ser,$host,$pwd);

if ($con)
{
   // echo "Database connected successfully!!!";
}
else
{
   die("Cannot connect" . $con->connect_error);
}

mysqli_select_db($con,"student_db");

//CREATE DATABASE db_name
/*$query = "CREATE DATABASE stud_db";
if (mysqli_query($con,$query) == true)
{
   echo "Database created successfully!!!";
}
else
{
```

```
    die("Cannot create database" . $con->connect_error);
}

mysqli_select_db($con,"stud_db");
$query = "create table if not exists tbl_name (id int primary key, name varchar(30));";
if (mysqli_query($con,$query) == true)
{
    echo "Table created successfully!!!";
}
else
{
    die("Cannot create table" . $con->connect_error);
}
mysqli_close($con);
*/

?>
```

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>AJAX Live Search</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
    <h2>Search Users</h2>
    <input type="text" id="search" placeholder="Type a name...">
    <div id="result"></div>

    <script>
        $(document).ready(function(){
            $("#search").keyup(function(){
                query = $('#search').val();
                if(query!='')
                {
                    $.ajax({
                        url: 'search.php',
                        method: 'POST',
                        data: {query:query},
```

```
                success: function(data){
                    if(data=="No data found!!"){
                        $('#result').html("No data found!!!");
                    }
                    else{
                        $('#result').html(data);
                    }
                }
            })
        }
        else
        {
            $('#result').html("");
        }
    })
})
</script>

</body>
</html>
```

Search.php

```
<?php

require_once('connect_db.php');

if (isset($_POST['query']))
{
    $data = $con->real_escape_string($_POST['query']);

    $sql = "SELECT name,email FROM stud_data WHERE name LIKE '%$data%' OR
email LIKE '%$data%';";

    $result = mysqli_query($con,$sql);

    if($result->num_rows>0){
        echo "<ul type='circle'>";
        while($row = $result->fetch_assoc()){
            echo "<li>" . $row['name'] . " - " . $row['email'] . "</li>";
        }
```

```
        echo "</ul>";
    }
    else{
        echo "No data found!!";
    }
}

$con->close();

?>
```

JSON

- JSON = JavaScript Object Notation.
- It is a lightweight format for storing and transferring data between server and client.
- Data is stored as key–value pairs (similar to objects in JavaScript).
- Format:

{

"name": "Riya",

"age": 22,

"city": "Surat"

}

- Why JSON for AJAX?
    - Easy for both JavaScript and PHP to understand.
    - More structured than plain text.
    - Often used in APIs and AJAX responses.

Sending JSON from PHP to JavaScript

HTML + JavaScript (json_example.html)

```
<!DOCTYPE html>

<html>

<head>

  <title>AJAX JSON Example</title>
```

```
<script>
  function loadData() {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "data.php", true);
    xhr.onreadystatechange = function() {
      if (xhr.readyState == 4 && xhr.status == 200) {
        var response = JSON.parse(xhr.responseText); // convert JSON to JS object
        document.getElementById("result").innerHTML =
          "Name: " + response.name + "<br>" +
          "Age: " + response.age + "<br>" +
          "City: " + response.city;
      }
    };
    xhr.send();
  }
</script>
</head>
<body>
<h2>AJAX JSON Example</h2>
<button onclick="loadData()">Load JSON Data</button>
<div id="result"></div>
</body>
</html>
```

PHP File (data.php)

```
<?php
```

```
// create an array

$data = array(

  "name" => "NaishaL",

  "age" => 23,

  "city" => "Surat"

);



// convert array to JSON and send it

echo json_encode($data);

?>
```

O/p
Clicking the button shows:
Name: NaishaL
Age: 23
City: Surat

3.1 PHP with MySQL/MongoDB

3.1.1 Connecting to databases using mysqli or PDO

3.1.2 Creating databases and tables

3.1.3 Executing CRUD operations: INSERT, SELECT, UPDATE, DELETE

3.1.4 Using clauses: WHERE, ORDER BY, LIMIT

3.2 AJAX for Backend Integration

3.2.1 Introduction to AJAX and asynchronous requests

3.2.2 Sending AJAX requests to PHP

3.2.3 Real-time search functionality

3.2.4 JSON data exchange with JavaScript and PHP

3.3 CodeIgniter Introduction

3.3.1 Installing and configuring CodeIgniter (CI4)

3.3.2 Understanding MVC architecture in CodeIgniter

3.3.3 Creating models, views, and controllers

3.3.4 URL routing and default controller setup

3.4 Core Features in CodeIgniter

3.4.1 Form validation using CI validation library

3.4.2 Session management and flashdata

CodeIgniter Introduction

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

Prerequisite for CodeIgniter

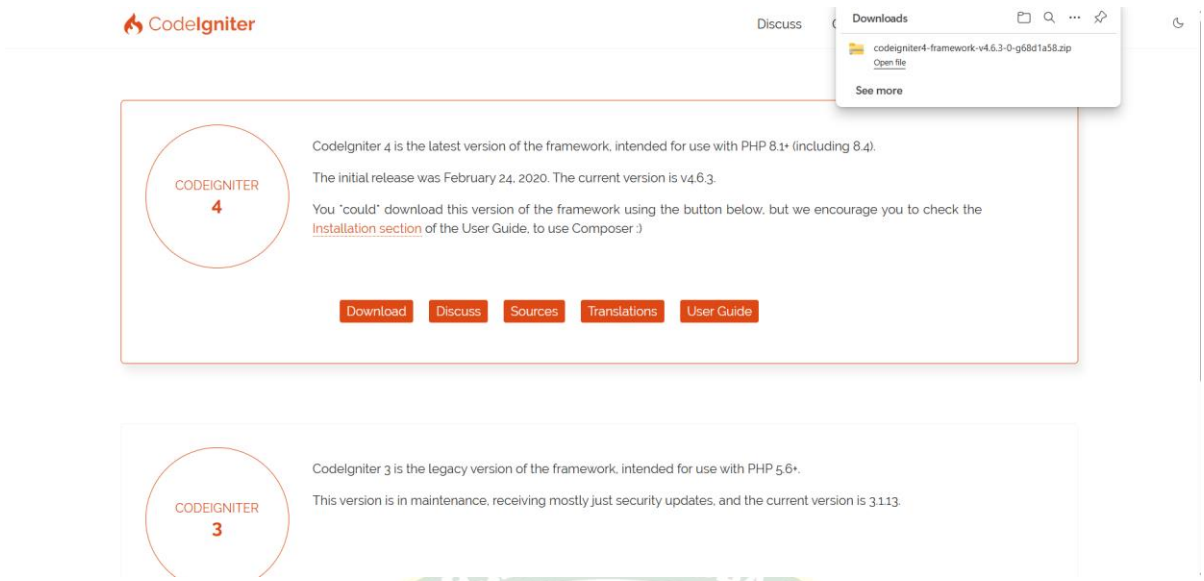
- Knowledge of PHP.

Advantage of using CodeIgniter

- It has a very rich set of libraries and helpers.
- save a lot of time, if you are developing a website from scratch.
- website built in CodeIgniter is secure too, as it has the ability to prevent various attacks that take place through websites.

Installation of CodeIgniter & Configuration of CodeIgniter

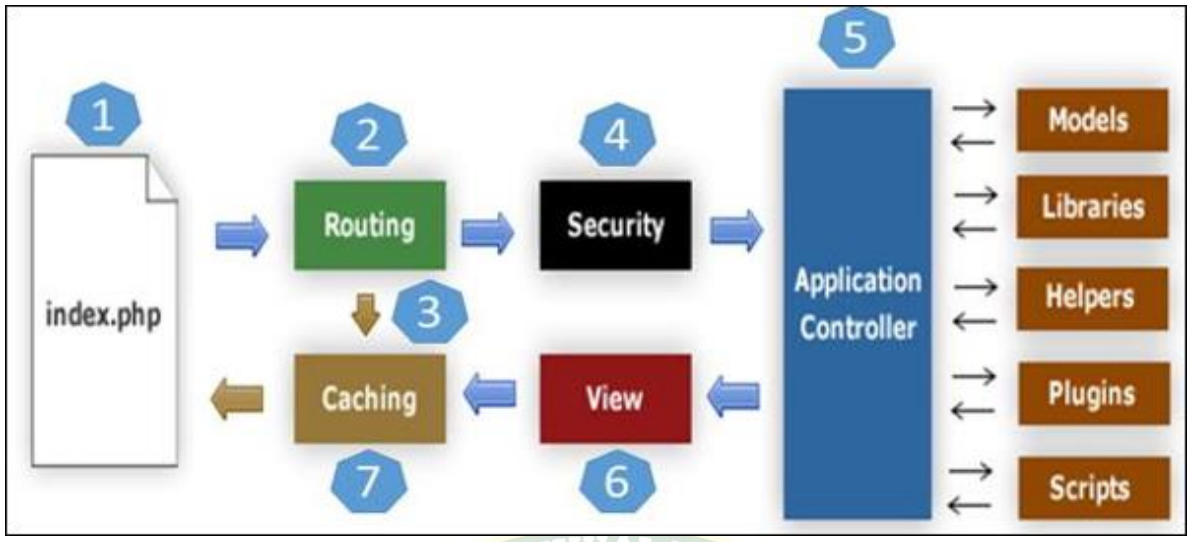
Step 1: Download the CodeIgniter from the link CodeIgniter



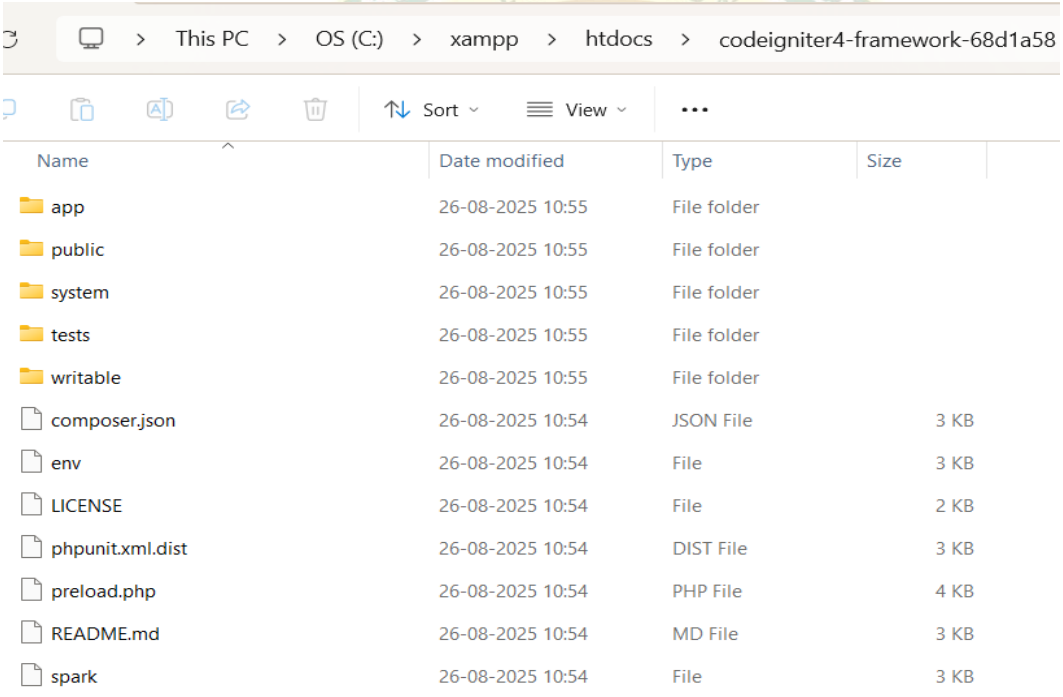
Step 2: Unzip folder
Step3: Upload all files and folders to your server.
Step4: After uploading all the files to your server, visit the URL of your server, e.g., www.domain-name.com.

Architecture of CodeIgniter application

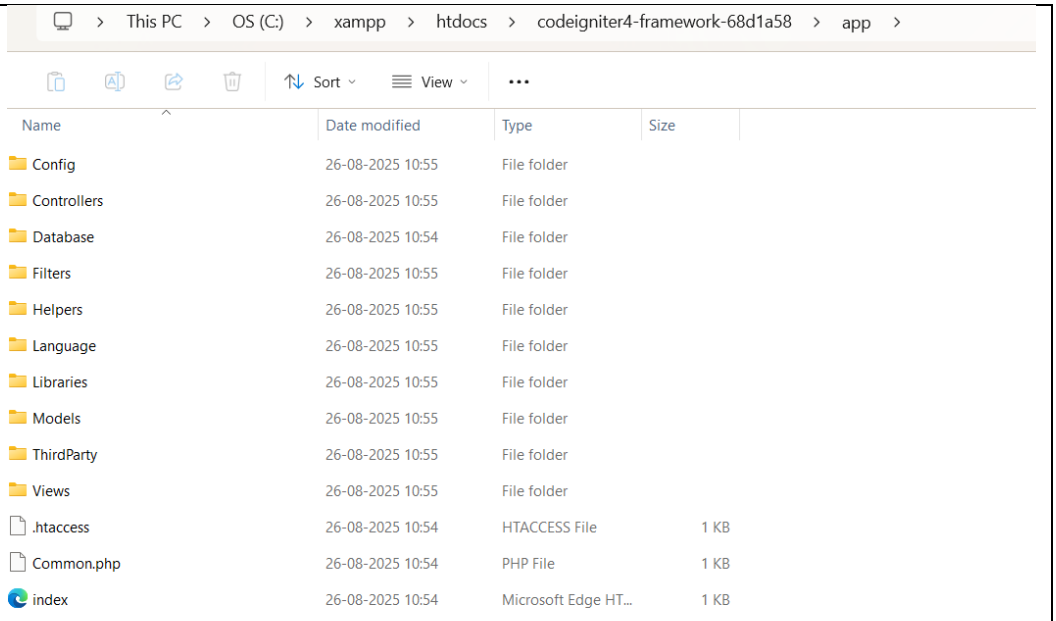
- whenever a request comes to CodeIgniter, it will first go to **index.php** page.
- In the second step, Routing will decide whether to pass this request to step-3 for caching or to pass this request to step-4 for security check.
- If the requested page is already in Caching, then Routing will pass the 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.



Directory Structure



Application	Application folder contains all the code of application that you are building. This is the folder where you will develop your project.



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.
Controller	This folder holds the controllers of your application. It is the basic part of your application.
Database	The database folder contains core database drivers and other database utilities.
Helper	In this folder, you can put helper class of your application.
Language	This folder contains language related files.
Libraries	This folder contains files of the libraries developed for your application.
Models	The database login will be placed in this folder.
Third-party	In this folder, you can place any plugins, which will be used for your application.
Views	Applications HTML files will be placed in this folder.

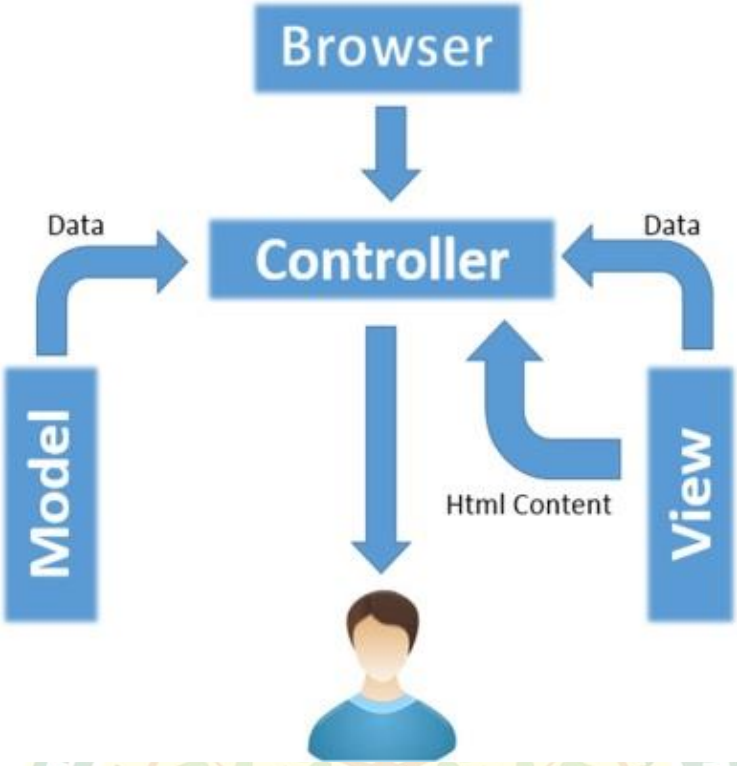
System	This folder contains CodeIgniter core codes, libraries, helpers and other files, which help make the coding easy. These libraries and helpers are loaded and used in web app development.

Core	This folder contains CodeIgniters 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.

3.3.2 Understanding MVC architecture in CodeIgniter

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.



Model	Model represents your data structures. Typically, your model classes will contain functions that help you retrieve, insert and update information in your database.
View	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.
Controller	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.

3.3.3 Creating Models, Views, and Controllers

1. Controller Example

Create file: /app/Controllers/Hello.php

```
<?php
namespace App\Controllers;

class Hello extends BaseController
{
    public function index()
```

```
    {
        return view('welcome_message'); // loads a view
    }

    public function greet($name = 'Guest')
    {
        return "Hello, " . $name;
    }
}
```

2. View Example

Create file: /app/Views/welcome_message.php

```
<!DOCTYPE html>
<html>
<head>
    <title>CodeIgniter 4</title>
</head>
<body>
    <h1>Welcome to CodeIgniter 4!</h1>
    <p>This is a simple view file.</p>
</body>
</html>
```

3. Model Example

Create file: /app/Models/UserModel.php

```
<?php
namespace App\Models;
use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table = 'users';   // Database table
    protected $primaryKey = 'id';
    protected $allowedFields = ['name', 'email', 'password'];
}
```

4. Using Model in Controller

```
<?php
namespace App\Controllers;
```

```
use App\Models\UserModel;

class User extends BaseController
{
    public function index()
    {
        $model = new UserModel();
        $data['users'] = $model->findAll(); // Fetch all records
        return view('user_list', $data);
    }
}
```

5. View for User List

```
/app/Views/user_list.php
<h2>User List</h2>
<ul>
<?php foreach ($users as $user): ?>
    <li><?= $user['name'] ?> - <?= $user['email'] ?></li>
<?php endforeach; ?>
</ul>
```

3.3.4 URL Routing and Default Controller Setup

URL Structure in CI4

Default format:

http://localhost:8080/controller/method/parameters

Example:

http://localhost:8080/hello/greet/John → Calls Hello::greet("John")

Route Configuration

Located in /app/Config/Routes.php

Example:

```
$routes->get('/', 'Home::index');              // Default homepage
```

```
$routes->get('about', 'Pages::about');          // Custom route
$routes->get('user/(:num)', 'User::profile/$1'); // Passing parameter
```

Default Controller Setup

In /app/Config/Routes.php:

```
$routes->setDefaultController('Home');
$routes->setDefaultMethod('index');
```

This means when you open http://localhost:8080/, it will load Home::index().

3.4 Core Features in CodeIgniter

3.4.1 Form Validation using CI Validation Library

Purpose

- Ensures user input is valid before processing (e.g., email format, required fields).
- CI4 has a built-in **Validation library**.

Example: Registration Form

Controller: app/Controllers/Register.php

```
<?php
namespace App\Controllers;

class Register extends BaseController
{
    public function index()
    {
        helper(['form']); // load form helper
        return view('register_form');
    }

    public function submit()
    {
        helper(['form']);
        $validation = \Config\Services::validation();
```

```
        $rules = [
            'username' => 'required|min_length[3]|max_length[20]',
            'email'    => 'required|valid_email',
            'password' => 'required|min_length[6]'
        ];

        if (!$this->validate($rules)) {
            return view('register_form', [
                'validation' => $this->validator
            ]);
        } else {
            return "Form submitted successfully!";
        }
    }
}
```

View: app/Views/register_form.php

```
<form action="/register/submit" method="post">
    <input type="text" name="username" placeholder="Username"><br>
    <input type="text" name="email" placeholder="Email"><br>
    <input type="password" name="password" placeholder="Password"><br>
    <button type="submit">Register</button>
</form>

<?php if(isset($validation)): ?>
    <div style="color:red;">
        <?= $validation->listErrors() ?>
    </div>
<?php endif; ?>
```

Route: app/Config/Routes.php

```
$routes->get('register', 'Register::index');
$routes->post('register/submit', 'Register::submit');
```

Output:

If fields are empty → shows validation errors.
If valid input → "Form submitted successfully!"

3.4.2 Session Management and Flashdata

Purpose

- **Sessions** store data across multiple requests (like user login info).
- **Flashdata** → temporary session data (available only for the next request).

Example: Login

Controller: app/Controllers/Auth.php

```
<?php
namespace App\Controllers;

class Auth extends BaseController
{
    public function login()
    {
        session()->set(['username' => 'JohnDoe']);
        session()->setFlashdata('message', 'You are logged in!');
        return redirect()->to('/auth/dashboard');
    }

    public function dashboard()
    {
        $username = session()->get('username');
        $message  = session()->getFlashdata('message');
        return view('dashboard', ['username' => $username, 'message' => $message]);
    }

    public function logout()
    {
        session()->destroy();
        return "Logged out!";
    }
}
```

View: app/Views/dashboard.php

```
<h2>Dashboard</h2>
<p>Welcome, <?= esc($username) ?>!</p>

<?php if ($message): ?>
```

```
    <p style="color:green;"><?= esc($message) ?></p>
<?php endif; ?>
```

Routes:

```
$routes->get('auth/login', 'Auth::login');
$routes->get('auth/dashboard', 'Auth::dashboard');
$routes->get('auth/logout', 'Auth::logout');
```

Output:

- Open /auth/login → redirects to dashboard.
- Dashboard shows: "Welcome, JohnDoe! You are logged in!" (flashdata disappears on refresh).
- /auth/logout → "Logged out!".

3.4.3 Handling File Uploads

Purpose

- CI4 provides an easy API for handling file uploads safely.

Example: Upload Profile Picture

Controller: app/Controllers/Upload.php

```
<?php
namespace App\Controllers;

class Upload extends BaseController
{
    public function index()
    {
        helper(['form']);
        return view('upload_form');
    }

    public function store()
    {
        $file = $this->request->getFile('userfile');

        if ($file->isValid() && !$file->hasMoved()) {
```

```
            $newName = $file->getRandomName();
            $file->move(WRITEPATH . 'uploads', $newName);
            return "File uploaded successfully: " . $newName;
        } else {
            return "File upload failed!";
        }
    }
}
```

View: app/Views/upload_form.php

```
<form action="/upload/store" method="post" enctype="multipart/form-data">
    <input type="file" name="userfile"><br>
    <button type="submit">Upload</button>
</form>
```

Routes:

```
$routes->get('upload', 'Upload::index');
$routes->post('upload/store', 'Upload::store');
```

Output:

- Uploads file into /writable/uploads/.
- Shows: "File uploaded successfully: randomname.jpg".

3.4.4 Loading Helpers and Libraries

Helpers

- Simple functions (e.g., form, url, text).
- Load them in controller or globally.

Example:

```
helper(['url', 'text']);

echo base_url();  // prints http://localhost:8080/
echo word_limiter("This is a very long sentence", 4);
```

Libraries (Services)

- Classes that provide advanced features (e.g., email, validation, session).
- Load using \Config\Services.

Example: Send Email

```
$email = \Config\Services::email();
$email->setFrom('you@example.com', 'Your Name');
$email->setTo('friend@example.com');
$email->setSubject('Test Email');
$email->setMessage('Hello from CI4 email library!');

if ($email->send()) {
    echo "Email sent!";
} else {
    echo $email->printDebugger();
}
```