**Project Structure**

**ecommerce\_project/**

**│**

**├── db/**

**│ └── db.php # Database connection file**

**│**

**├── includes/**

**│ ├── header.php # Common header file**

**│ ├── footer.php # Common footer file**

**│ └── sidebar.php # Sidebar file**

**│**

**├── assets/**

**│ ├── css/**

**│ │ ├── styles.css # Main styles for the project**

**│ │ └── user\_styles.css # User-specific styles**

**│ ├── js/**

**│ │ └── scripts.js # JavaScript files**

**│ └── images/ # Static images used across the project**

**│**

**├── uploads/ # Directory for uploaded images (e.g., profile images)**

**│**

**├── user/**

**│ ├── actions/**

**│ │ ├── register\_submit.php # Handles registration form submissions**

**│ │ └── login\_submit.php # Handles login form submissions**

**│ ├── register.php # User registration page**

**│ ├── login.php # User login page**

**│ └── profile.php # User profile page**

**│**

**├── cart/**

**│ ├── view\_cart.php # View cart page**

**│ └── checkout.php # Checkout page**

**│**

**├── product/**

**│ ├── list.php # Product listing page**

**│ ├── details.php # Product details page**

**│ └── add\_product.php # Page to add new products (admin)**

**│**

**├── admin/**

**│ ├── dashboard.php # Admin dashboard**

**│ └── manage\_users.php # Manage users page (admin)**

**│**

**├── index.php # Homepage**

**└── .htaccess # Apache configuration file (if using Apache)**

Add other file according to their requirement place.

**Key Components**

* **db/:** Contains the database connection file. You may include other database-related files like configuration settings or migration scripts.
* **includes/:** Common includes such as the header, footer, and sidebar to keep your HTML DRY (Don't Repeat Yourself).
* **assets/:** Holds all static files such as stylesheets, JavaScript, and images.
* **uploads/:** Stores files uploaded by users, such as profile images. Ensure this directory is secured and configured to prevent execution of scripts.
* **user/:** Handles user-specific functionalities such as registration, login, and profile management. The actions directory within it contains scripts that process form submissions and other user-related actions.
* **cart/:** Manages shopping cart functionalities, including viewing the cart and the checkout process.
* **product/:** Handles product-related functionalities, such as listing products, viewing product details, and adding new products (if there is an admin role).
* **admin/:** Contains admin-specific pages and functionalities like managing users and overseeing the dashboard.
* **index.php:** The main entry point for your website, typically the homepage.

**Security Considerations**

* **Secure Uploads:**
  + Ensure the uploads/ directory is configured to not allow execution of scripts (e.g., PHP files).
  + Validate uploaded files for correct types and sizes.
* **Input Validation:**
  + Always validate and sanitize user input to prevent SQL injection and XSS attacks.
  + Use prepared statements for database queries.
* **Session Management:**
  + Implement proper session management to handle user authentication and authorization securely.
* **File Permissions:**
  + Set appropriate file permissions for your directories and files to prevent unauthorized access or modifications.

**Step 1:**

**First Create a floder named 'ecommerce\_project’ inside htdocs of your XAMPP folder**

**Step 2:**

**Create a database connectivity file and write the code of connectivity for connection of PHP and Mysql and write code to create dynamically the database and dynamically create all tables required for project**

**db\_connect.php/db.php/congig.php**

|  |
| --- |
| <?php  // Database connection parameters  $host = 'localhost';     // Your MySQL host (e.g., 'localhost')  $username = 'root';      // Your MySQL username  $password = '';          // Your MySQL password  $database = 'ecommerce\_project'; // The database name you want to create  // Create a connection to the MySQL server  $conn = new mysqli($host, $username, $password);  // Check the connection  if ($conn->connect\_error) {      die("Connection failed: " . $conn->connect\_error);  }  // Create the database  $sql = "CREATE DATABASE IF NOT EXISTS $database";  if ($conn->query($sql) === TRUE) {    //  echo "Database '$database' created successfully or already exists.<br>";  } else {      die("Error creating database: " . $conn->error);  }  // Select the database  if ($conn->select\_db($database) === TRUE) {  //    echo "Successfully selected database '$database'.<br>";  } else {      die("Error selecting database: " . $conn->error);  }  $conn = new mysqli($host, $username, $password, $database);  // Check the connection  if ($conn->connect\_error) {      die("Connection failed: " . $conn->connect\_error);  }  $sql\_admin = "  CREATE TABLE Admin (      AdminID INT AUTO\_INCREMENT PRIMARY KEY,      Username VARCHAR(100) NOT NULL UNIQUE,      Password VARCHAR(255) NOT NULL  )";  // Execute the query  if ($conn->query($sql\_admin) === TRUE) {     // echo "Table User created successfully";  } else {  //echo "Error creating table: " . $conn->error;  }  $sql\_user = "  CREATE TABLE IF NOT EXISTS User (      UserID INT AUTO\_INCREMENT PRIMARY KEY,      UserName VARCHAR(100) NOT NULL,      Email VARCHAR(100) NOT NULL UNIQUE,      Password VARCHAR(255) NOT NULL,      Address TEXT,      Gender ENUM('Male', 'Female', 'Other'),      ProfileImage VARCHAR(255),      RegistrationDate DATETIME DEFAULT CURRENT\_TIMESTAMP  )";  // Execute the query  if ($conn->query($sql\_user) === TRUE) {     // echo "Table User created successfully";  } else {    //  echo "Error creating table: " . $conn->error;  }  $sql\_category = "  CREATE TABLE IF NOT EXISTS Category (      CategoryID INT AUTO\_INCREMENT PRIMARY KEY,      CategoryName VARCHAR(100) NOT NULL  )";  // Execute the query  if ($conn->query($sql\_category) === TRUE) {     // echo "Table User created successfully";  } else {     // echo "Error creating table: " . $conn->error;  }  $sql\_product = "  CREATE TABLE IF NOT EXISTS Product (      ProductID INT AUTO\_INCREMENT PRIMARY KEY,      ProductName VARCHAR(100) NOT NULL,      ProductDescription TEXT,      ProductPrice DECIMAL(10, 2) NOT NULL,      CategoryID INT,      ProductImage VARCHAR(255),      FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)  )";  // Execute the query  if ($conn->query($sql\_product) === TRUE) {     // echo "Table User created successfully";  } else {     // echo "Error creating table: " . $conn->error;  }  $sql\_order = "  CREATE TABLE `Order` (      OrderID INT AUTO\_INCREMENT PRIMARY KEY,      OrderDate DATETIME DEFAULT CURRENT\_TIMESTAMP,      UserID INT,      TotalAmount DECIMAL(10, 2) NOT NULL,      FOREIGN KEY (UserID) REFERENCES User(UserID)  )  ";  // Execute the query  if ($conn->query($sql\_order) === TRUE) {     // echo "Table User created successfully";  } else {  //echo "Error creating table: " . $conn->error;  }  $sql\_orderdetails = "  CREATE TABLE OrderDetails (      OrderDetailID INT AUTO\_INCREMENT PRIMARY KEY,      OrderID INT NOT NULL,      ProductName VARCHAR(100) NOT NULL,      ProductPrice DECIMAL(10, 2) NOT NULL,      Quantity INT NOT NULL,      Total DECIMAL(10, 2) NOT NULL,      Date DATETIME DEFAULT CURRENT\_TIMESTAMP,      FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID) ON DELETE CASCADE  )";  // Execute the query  if ($conn->query($sql\_orderdetails) === TRUE) {     // echo "Table User created successfully";  } else {    //  echo "Error creating table: " . $conn->error;  }  $sql\_payment = "  CREATE TABLE Payment (      PaymentID INT AUTO\_INCREMENT PRIMARY KEY,      OrderID INT,      PaymentDate DATETIME DEFAULT CURRENT\_TIMESTAMP,      PaymentAmount DECIMAL(10, 2) NOT NULL,      PaymentMethod VARCHAR(50),      PaymentStatus VARCHAR(50),      FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID)  )  ";  // Execute the query  if ($conn->query($sql\_payment) === TRUE) {     // echo "Table User created successfully";  } else {    //  echo "Error creating table: " . $conn->error;  }  $sql\_review = "  CREATE TABLE Review (      ReviewID INT AUTO\_INCREMENT PRIMARY KEY,      ProductID INT,      UserID INT,      Rating TINYINT NOT NULL CHECK (Rating BETWEEN 1 AND 5),      ReviewText TEXT,      FOREIGN KEY (ProductID) REFERENCES Product(ProductID),      FOREIGN KEY (UserID) REFERENCES User(UserID)  )  ";  // Execute the query  if ($conn->query($sql\_review) === TRUE) {     // echo "Table User created successfully";  } else {     // echo "Error creating table: " . $conn->error;  }  ?> |

**Register.php**

|  |
| --- |
| <?php  // Include database connection  include '../db/db.php';  // Initialize variables to store user input and error messages  $username = $email = $password = $confirmPassword = $address = $gender = $profileImage = '';  $errors = [];  ?>  <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>User Registration</title>      <link rel="stylesheet" href="../assets/css/styles.css">  </head>  <body>      <div class="container">          <h2>User Registration</h2>          <?php          if (!empty($errors)) {              echo '<div class="errors">';              foreach ($errors as $error) {                  echo '<p>' . htmlspecialchars($error) . '</p>';              }              echo '</div>';          }          ?>          <form action="actions/register\_submit.php" method="post" enctype="multipart/form-data">              <div>                  <label for="username">Username:</label>                  <input type="text" name="username" id="username" value="<?php echo htmlspecialchars($username); ?>" required>              </div>              <div>                  <label for="email">Email:</label>                  <input type="email" name="email" id="email" value="<?php echo htmlspecialchars($email); ?>" required>              </div>              <div>                  <label for="password">Password:</label>                  <input type="password" name="password" id="password" required>              </div>              <div>                  <label for="confirm\_password">Confirm Password:</label>                  <input type="password" name="confirm\_password" id="confirm\_password" required>              </div>              <div>                  <label for="address">Address:</label>                  <textarea name="address" id="address"><?php echo htmlspecialchars($address); ?></textarea>              </div>              <div>                  <label for="gender">Gender:</label>                  <select name="gender" id="gender" required>                      <option value="">Select Gender</option>                      <option value="Male" <?php if ($gender === 'Male') echo 'selected'; ?>>Male</option>                      <option value="Female" <?php if ($gender === 'Female') echo 'selected'; ?>>Female</option>                      <option value="Other" <?php if ($gender === 'Other') echo 'selected'; ?>>Other</option>                  </select>              </div>              <div>                  <label for="profile\_image">Profile Image:</label>                  <input type="file" name="profile\_image" id="profile\_image" accept="image/\*">              </div>              <div>                  <button type="submit">Register</button>              </div>          </form>      </div>  </body>  </html> |

**Code Explanation:**

**PHP Section**

1. **Include Database Connection**:

include '../db/db.php';

* + This line includes the database connection file (db.php), which should establish a connection to your database so that you can perform queries later.

1. **Initialize Variables**:

$username = $email = $password = $confirmPassword = $address = $gender = $profileImage = '';

$errors = [];

* + This initializes variables to hold user inputs and error messages. They are initially set to empty strings or an empty array ($errors).

**HTML Section**

1. **HTML Document Setup**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Registration</title>

<link rel="stylesheet" href="../assets/css/styles.css">

</head>

<body>

<div class="container">

<h2>User Registration</h2>

* + This sets up the HTML document with metadata and links to a CSS stylesheet for styling.

1. **Error Display**:

<?php

if (!empty($errors)) {

echo '<div class="errors">';

foreach ($errors as $error) {

echo '<p>' . htmlspecialchars($error) . '</p>';

}

echo '</div>';

}

?>

* + This PHP block checks if there are any errors in the $errors array and displays them to the user in a <div> with the class errors. It uses htmlspecialchars() to prevent XSS attacks by escaping special characters in error messages.

1. **Registration Form**:

<form action="actions/register\_submit.php" method="post" enctype="multipart/form-data">

* + **action="actions/register\_submit.php"**: Specifies that the form will be submitted to register\_submit.php located in the actions directory.
  + **method="post"**: Uses the POST method to send data, which is more secure than GET for sending sensitive information like passwords.
  + **enctype="multipart/form-data"**: Necessary for file uploads. This tells the form to allow file data to be submitted in addition to regular form data.

1. **Form Fields**:

<div>

<label for="username">Username:</label>

<input type="text" name="username" id="username" value="<?php echo htmlspecialchars($username); ?>" required>

</div>

* + **<label> and <input> elements**: Used to create form fields for user input. Each input field is linked to a corresponding variable for data persistence across form submissions.
  + **value="<?php echo htmlspecialchars($username); ?>"**: Pre-fills the input field with the previously entered value, making it easier for users to correct errors without re-entering everything.

1. **Gender Selection**:

<div>

<label for="gender">Gender:</label>

<select name="gender" id="gender" required>

<option value="">Select Gender</option>

<option value="Male" <?php if ($gender === 'Male') echo 'selected'; ?>>Male</option>

<option value="Female" <?php if ($gender === 'Female') echo 'selected'; ?>>Female</option>

<option value="Other" <?php if ($gender === 'Other') echo 'selected'; ?>>Other</option>

</select>

</div>

* + This creates a dropdown selection for gender with pre-selection logic. It automatically selects the gender option based on the previously submitted value.

1. **Profile Image Upload**:

<div>

<label for="profile\_image">Profile Image:</label>

<input type="file" name="profile\_image" id="profile\_image" accept="image/\*">

</div>

* + This allows users to upload a profile image. The accept="image/\*" attribute restricts file uploads to image types only.

1. **Submit Button**:

<div>

<button type="submit">Register</button>

</div>

* + A button to submit the form data for processing.

**Overall Workflow**

1. **User fills out the form** with their details, including uploading a profile image.
2. **Form submission** is handled by register\_submit.php, where validation and database operations should occur.
3. **Errors are displayed** on the same page if validation fails.
4. **Input persistence** is provided so users don't have to re-enter all data if they make a mistake.

**Important Considerations**

* **Validation**: Ensure register\_submit.php performs thorough validation of user input to prevent SQL injection and other security issues.
* **Security**: Passwords should be hashed before storing them in the database, using a secure hashing algorithm like bcrypt (which is done with password\_hash()).
* **File Uploads**: Validate and sanitize the uploaded file to ensure only valid images are accepted, preventing potential security vulnerabilities from uploading malicious files.

**Register\_submit.php**

|  |
| --- |
| <?php  // Include database connection  include \_\_DIR\_\_ . '/../../db/db.php'; // Adjust this based on the actual path  // Start session to store errors  session\_start();  $errors = [];  // Process the registration form submission  if ($\_SERVER["REQUEST\_METHOD"] == "POST") {      // Get user input      $username = trim($\_POST['username']);      $email = trim($\_POST['email']);      $password = $\_POST['password'];      $confirmPassword = $\_POST['confirm\_password'];      $address = trim($\_POST['address']);      $gender = $\_POST['gender'];      // Validate input      if (empty($username) || empty($email) || empty($password) || empty($confirmPassword) || empty($gender)) {          $errors[] = "All fields are required.";      }      if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {          $errors[] = "Invalid email format.";      }      if ($password !== $confirmPassword) {          $errors[] = "Passwords do not match.";      }      // Check for errors before proceeding      if (empty($errors)) {          // Hash the password for security          $hashedPassword = password\_hash($password, PASSWORD\_BCRYPT);          // Handle profile image upload          $profileImage = null; // Initialize as null          if (isset($\_FILES['profile\_image']) && $\_FILES['profile\_image']['error'] == UPLOAD\_ERR\_OK) {              $uploadDir = '../uploads/';              // Create the uploads directory if it doesn't exist              if (!is\_dir($uploadDir)) {                  mkdir($uploadDir, 0777, true);              }              $profileImage = basename($\_FILES['profile\_image']['name']);              $targetFilePath = $uploadDir . $profileImage;                // Move uploaded file to the target directory              if (!move\_uploaded\_file($\_FILES['profile\_image']['tmp\_name'], $targetFilePath)) {                  $errors[] = "Failed to upload profile image.";              }          }          // Prepare the SQL query to insert the new user into the database          $sql = "INSERT INTO User (UserName, Email, Password, Address, Gender, ProfileImage, RegistrationDate)                  VALUES ('$username', '$email', '$hashedPassword', '$address', '$gender', '$profileImage', NOW())";          // Execute the query          if ($conn->query($sql) === TRUE) {              // Successful registration, redirect to login page              header("Location: ../login.php");              exit();          } else {              $errors[] = "Error: " . $conn->error;              header("Location: ../register.php");              exit();            }      }      // Store errors in session to display on the registration form      $\_SESSION['errors'] = $errors;     // echo $\_SESSION['errors'];  }  ?> |

**Code Explanation:**

**PHP Script Breakdown**

1. **Database Connection**:

include \_\_DIR\_\_ . '/../../db/db.php'; // Adjust this based on the actual path

* + This line includes a file to establish a database connection. The \_\_DIR\_\_ constant provides the current directory of the file, making the path to db.php relative to the script's location.

1. **Session Start**:

session\_start();

$errors = [];

* + Starts a session to store errors, allowing them to be passed between different pages. An empty array $errors is initialized to store any validation errors encountered during the process.

1. **Check Request Method**:

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

...

}

* + Ensures the script only processes the form when it is submitted using the POST method. This is a common practice for handling form submissions securely.

1. **Get User Input**:

$username = trim($\_POST['username']);

$email = trim($\_POST['email']);

$password = $\_POST['password'];

$confirmPassword = $\_POST['confirm\_password'];

$address = trim($\_POST['address']);

$gender = $\_POST['gender'];

* + Retrieves and trims the input data from the form. Trimming removes any whitespace from the beginning and end of the input, which helps prevent accidental input errors and security issues.

1. **Input Validation**:

if (empty($username) || empty($email) || empty($password) || empty($confirmPassword) || empty($gender)) {

$errors[] = "All fields are required.";

}

if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {

$errors[] = "Invalid email format.";

}

if ($password !== $confirmPassword) {

$errors[] = "Passwords do not match.";

}

* + Checks if all required fields are filled.
  + Validates the email format using filter\_var().
  + Confirms that the password and confirmation password match.

1. **Hash Password**:

$hashedPassword = password\_hash($password, PASSWORD\_BCRYPT);

* + Hashes the password using bcrypt, a secure hashing algorithm. This is crucial for storing passwords securely in the database.

1. **Profile Image Upload**:

$profileImage = null; // Initialize as null

if (isset($\_FILES['profile\_image']) && $\_FILES['profile\_image']['error'] == UPLOAD\_ERR\_OK) {

$uploadDir = '../uploads/';

if (!is\_dir($uploadDir)) {

mkdir($uploadDir, 0777, true);

}

$profileImage = basename($\_FILES['profile\_image']['name']);

$targetFilePath = $uploadDir . $profileImage;

if (!move\_uploaded\_file($\_FILES['profile\_image']['tmp\_name'], $targetFilePath)) {

$errors[] = "Failed to upload profile image.";

}

}

* + Checks if a profile image has been uploaded and if there are no errors during the upload.
  + Creates an uploads directory if it doesn't exist to store the uploaded files.
  + Moves the uploaded file to the designated directory and updates the $profileImage variable with the file name. If moving the file fails, an error is added to the errors array.

1. **SQL Query Preparation**:

$sql = "INSERT INTO User (UserName, Email, Password, Address, Gender, ProfileImage, RegistrationDate)

VALUES ('$username', '$email', '$hashedPassword', '$address', '$gender', '$profileImage', NOW())";

* + Prepares an SQL query to insert the user data into the User table. The NOW() function is used to record the current timestamp for RegistrationDate.

1. **Execute Query**:

if ($conn->query($sql) === TRUE) {

header("Location: ../login.php");

exit();

} else {

$errors[] = "Error: " . $conn->error;

header("Location: ../register.php");

exit();

}

* + Attempts to execute the query. If successful, the user is redirected to the login page.
  + If there is an error with the query, it captures the error and redirects back to the registration page.

1. **Store Errors in Session**:

$\_SESSION['errors'] = $errors;

* Stores the $errors array in the session so that the registration page can access and display them to the user upon redirection.

**Key Considerations**

* **Security**:
  + Make sure to sanitize user inputs to prevent SQL injection. Using prepared statements or parameterized queries is highly recommended.
  + Ensure file uploads are validated to prevent the upload of malicious files. This includes checking file size, type, and possibly using a library to verify image content.
* **Error Handling**:
  + Proper error handling and user feedback are implemented to improve user experience. Errors are stored in a session and displayed to the user.
* **Directory Structure**:
  + Verify the correctness of the paths used for file inclusion and uploads to avoid issues during execution.

**Explain it with more Clarity:**

**How to add file:**

**Explanation of the Code**

1. **Check if File is Uploaded Successfully**:

if (isset($\_FILES['profile\_image']) && $\_FILES['profile\_image']['error'] == UPLOAD\_ERR\_OK) {

* + **$\_FILES['profile\_image']**: This is a superglobal array in PHP that contains information about the uploaded file. The profile\_image key corresponds to the name attribute of the file input in your HTML form.
  + **$\_FILES['profile\_image']['error']**: This checks for any error codes related to the file upload. UPLOAD\_ERR\_OK indicates that the file was uploaded successfully without errors.

1. **Set the Upload Directory**:

$uploadDir = '../uploads/';

* + This line sets the directory where the uploaded file will be stored. In this case, it’s set to ../uploads/, which is one level up from the current directory in a folder named uploads.

1. **Create the Uploads Directory if it Doesn’t Exist**:

if (!is\_dir($uploadDir)) {

mkdir($uploadDir, 0777, true);

}

* + **is\_dir($uploadDir)**: This function checks if the specified directory exists.
  + **mkdir($uploadDir, 0777, true)**: If the directory doesn’t exist, this line creates it with permissions 0777 (read, write, and execute for everyone). The true parameter allows the creation of nested directories.

1. **Prepare the Target File Path**:

$profileImage = basename($\_FILES['profile\_image']['name']);

$targetFilePath = $uploadDir . $profileImage;

* + **basename($\_FILES['profile\_image']['name'])**: This extracts the base name of the uploaded file, which is just the filename without the path.
  + **$targetFilePath**: This combines the upload directory and the filename to create the full path where the file will be saved.

1. **Move the Uploaded File**:

if (!move\_uploaded\_file($\_FILES['profile\_image']['tmp\_name'], $targetFilePath)) {

$errors[] = "Failed to upload profile image.";

}

* + **move\_uploaded\_file()**: This function moves the uploaded file from its temporary location (where PHP initially stores it) to the target location specified by $targetFilePath.
  + **$\_FILES['profile\_image']['tmp\_name']**: This contains the temporary filename of the uploaded file on the server.
  + If the function fails to move the file, it adds an error message to the $errors array.

**Key Points**

* **File Upload Safety**: This code does not currently handle the validation of file types or potential security risks (e.g., executing malicious files). For production use, you should check file types and implement other security measures.
* **File Overwriting**: If a file with the same name already exists in the uploads directory, this code will overwrite it. Consider using a unique filename (e.g., appending a timestamp or generating a unique ID) to prevent overwriting.
* **Directory Permissions**: Ensure the web server has appropriate permissions to create directories and write files in the specified location.