Lab 1

Objective: To Create CV Using HTML and CSS

Theory

HTML is the standard language used to define the structure and content of web pages. CSS is used to control the appearance and layout of the HTML elements. By combining HTML and CSS, you can create a visually appealing and well-structured web page, such as a personal CV.

Code Sample:

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>My CV</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="cv-container">
        <header>
            <h1>Person Person</h1>
            <p>Web Developer</p>
        </header>
            <section class="personal-info">
            <h2>Personal Information</h2>
            <ul>
                <li>Email: person@example.com</li>
                <li>Phone: (123) 456-7890</li>
                <li>Location: Pokhara, Nepal</li>
            </ul>
        </section>
        <section class="education">
            <h2>Education</h2>
            <p>Bachelor of Computer Application, PN University, 2020-2024</p>
        </section>
        <section class="experience">
```

```
            <h2>Work Experience</h2>
            <p>Web Developer at ABC Company, 2023-present</p>
        </section>
          <section class="skills">
            <h2>Skills</h2>
            <ul>
                <li>HTML, CSS, JavaScript</li>
                <li>PHP, MySQL</li>
                <li>React, Node.js</li>
            </ul>
        </section>
        <section class="contact">
            <h2>Contact</h2>
            <p>Email me at: person@example.com</p>
        </section>
    </div>
</body>
</html>
```

CSS:

styles.css > ...

```
body {font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
}v-container {
    width: 80%;
    margin: auto;
    padding: 20px;
    background-color: white;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}header {
    text-align: center;
}h1 {
    margin: 0;
}h2 {
    color: #333;
}ection {
    margin-bottom: 20px;
}ul {
    list-style-type: none;
    padding: 0;
}ul li {
    margin-bottom: 5px;
}
```

CSS is written in Separate file and saved with .css extension and linked with HTML within <head> section.

Demonstration:

Conclusion

This lab demonstrated the fundamental principles of HTML and CSS by creating a structured and styled personal CV. It provided hands-on experience in combining structure (HTML) with presentation (CSS), which are essential skills in web development.

LAB 2

Objective: To create a user registration form using HTML AND CSS

Theory

A user registration form is an essential component of many web applications. It typically includes input fields such as name, email, password, and other personal information. In HTML, forms are created using the <form> tag, and input elements are used to capture user data.

Code sample:

HTML

lab2.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>User Registration</title>
    <link rel="stylesheet" href="form.css">
</head>
<body>
    <div class="form-container">
        <h2>User Registration</h2>
        <form action="#" method="POST">
            <label for="name">Full Name:</label>
            <input type="text" id="name" name="name" required>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>
            <label for="confirm-password">Confirm Password:</label>
            <input type="password" id="confirm-password" name="confirm-password" required>
            <button type="submit">Register</button>
        </form>
    </div>
</body>
</html>
```

CSS

form.css

```css
body {
    font-family: Arial, sans-serif;
    background-color: #e9ecef;
}
.form-container {
    width: 50%;
    margin: auto;
    padding: 20px;
    background-color: white;
    border-radius: 10px;
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
}
h2 {
    text-align: center;
}
form {
    display: flex;
```

```
    flex-direction: column;
}
label {
    margin: 10px 0 5px;
}
input {
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
button {
    padding: 10px;
    background-color: #007BFF;
    color: white;
    border: none;
    border-radius: 5px;
}

```

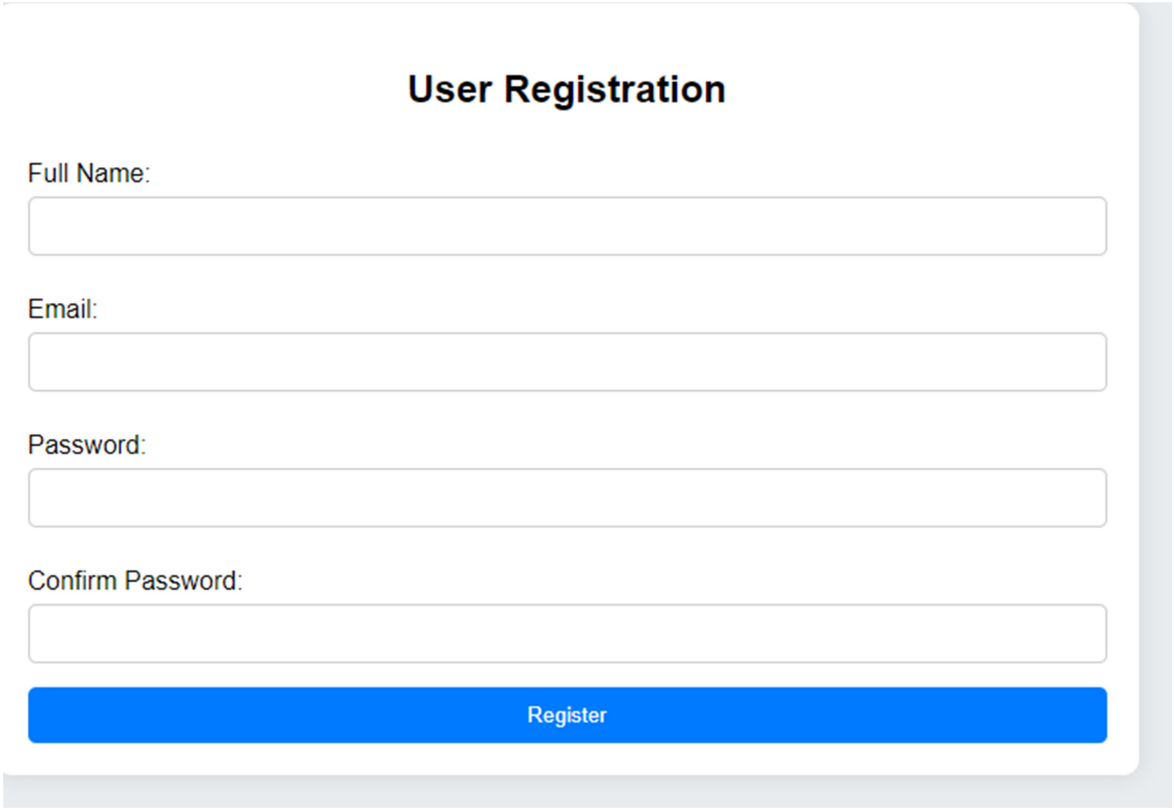
Implementation

1. Created a form with name, email, password, and submit fields.

2. Styled the form using CSS for alignment, background colors, and input styling.

3. Used basic validation in HTML (e.g., required attribute) to ensure users provide necessary information.

Output:



Conclusion

This task helped to build a simple yet effective user registration form. It reinforced the importance of proper HTML form elements and CSS for styling, as well as form validation for user input. This foundational knowledge is crucial for any web application that requires user data input.

LAB 3

Objective: To create a simple image slider using HTML CSS and JavaScript

Theory

An image slider is a common feature on websites, allowing users to view multiple images in a single space. It uses JavaScript to handle the interactivity and image transitions.

- **HTML** provides the structure by defining the images in <img> tags.
- **CSS** handles the positioning and styling of the slider, ensuring that images are displayed properly.
- **JavaScript** is used to change the displayed image when the user clicks navigation buttons.

Code Sample

HTML

lab3.html > html

```
<!DOCTYPE html>
<html>
<head>
    <title>Image Slider</title>
    <link rel="stylesheet" href="slider.css">
</head>
<body>
    <div class="slider-container">
        <div class="slider">
            <img src="image1.jpg" alt="Image 1" class="slider-image">
            <img src="image2.jpg" alt="Image 2" class="slider-image">
            <img src="image3.jpg" alt="Image 3" class="slider-image">
        </div>
        <button id="prev" class="nav-btn">&#10094;</button>
        <button id="next" class="nav-btn">&#10095;</button>
    </div>
    <script src="slider.js"></script>
</body>
</html>
```

CSS

slider.css > ...

```css
.slider-container {
    position: relative;
    width: 80%;
    margin: auto;
    overflow: hidden;
}
.slider {
    display: flex;
    transition: transform 0.5s ease;
}
.slider-image {
    width: 100%;
    height: auto;
}
.nav-btn {
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
    background-color: rgba(0, 0, 0, 0.5);
    color: white;
    padding: 10px;
    border: none;
    font-size: 20px;
    cursor: pointer;
}
#prev {
    left: 10px;
}
#next {
    right: 10px;
}
```

JavaScript

JS slider.js > ...

```javascript
let index = 0;

function showSlide(i) {
    const slides = document.querySelectorAll(".slider-image");
    index = (i + slides.length) % slides.length;
    document.querySelector(".slider").style.transform = `translateX(${-index * 100}%)`;
}
document.getElementById("next").addEventListener("click", () => showSlide(index + 1));
document.getElementById("prev").addEventListener("click", () => showSlide(index - 1));

setInterval(() => showSlide(index + 1), 3000); // Auto-slide every 3 seconds
```

Implementation

- Created a container with multiple images inside elements.
- Used CSS to hide images and display them in a row using display: none for inactive images.
- Implemented JavaScript to handle button clicks for image navigation (next and previous).

Output:

Conclusion:

This lab helped reinforce the integration of HTML, CSS, and JavaScript to create interactive web components. This task demonstrated how to combine basic web technologies to achieve dynamic behavior, enhancing the interactivity of web pages.

LAB 4

Objective: To develop simple calculator using front end stack

Theory

A calculator is an interactive tool that takes user input and performs arithmetic operations. JavaScript is an ideal choice for implementing the logic behind such a tool as it can handle mathematical operations and capture user events like button clicks.

- **HTML** provides the layout for buttons and display screen.
- **CSS** is used to style the calculator, adjusting button sizes and positions.
- **JavaScript** performs the actual calculations and updates the display based on user input.

Code sample

HTML:

lab4.html > html > body

```
<!DOCTYPE html>
<html>
<head>
    <title>Calculator</title>
    <link rel="stylesheet" href="clc.css">
</head>
<body>
    <div class="calculator">
        <input type="text" id="display" class="display" disabled />
        <div class="buttons">
            <button class="btn" onclick="appendToDisplay('7')">7</button>
            <button class="btn" onclick="appendToDisplay('8')">8</button>
            <button class="btn" onclick="appendToDisplay('9')">9</button>
            <button class="btn operator" onclick="appendToDisplay('+')">+</button>
            <button class="btn" onclick="appendToDisplay('4')">4</button>
            <button class="btn" onclick="appendToDisplay('5')">5</button>
            <button class="btn" onclick="appendToDisplay('6')">6</button>
            <button class="btn operator" onclick="appendToDisplay('-')">-</button>
            <button class="btn" onclick="appendToDisplay('1')">1</button>
            <button class="btn" onclick="appendToDisplay('2')">2</button>
            <button class="btn" onclick="appendToDisplay('3')">3</button>
            <button class="btn operator" onclick="appendToDisplay('*')">*</button>
            <button class="btn" onclick="appendToDisplay('0')">0</button>
            <button class="btn" onclick="appendToDisplay('.')">.</button>
            <button class="btn equal" onclick="calculate()">=</button>
            <button class="btn operator" onclick="appendToDisplay('/')">/</button>
            <button class="btn clear" onclick="clearDisplay()">C</button>
        </div>
    </div>
    <script src="clc.js"></script>
</body>
</html>
```

CSS:

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
.calculator {
    background-color: white;
    border-radius: 10px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
    padding: 20px;
    width: 260px;
}
.display {
    width: 100%;
    height: 50px;
    font-size: 24px;
    text-align: right;
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
.buttons {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    gap: 10px;
}
```

```css
button {
    font-size: 18px;
    padding: 20px;
    background-color: #f0f0f0;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}
button:hover {
    background-color: #ddd;
}
.operator {
    background-color: #ff9500;
    color: white;
}
.operator:hover {
    background-color: #ff7f00;
}
.equal {
    background-color: #4CAF50;
    color: white;
    grid-column: span 2;
}
.equal:hover {
    background-color: #45a049;
}
.clear {
    background-color: #f44336;
    color: white;
    grid-column: span 2;
}
.clear:hover {
    background-color: #e53935;
}
```

JavaScript:

```javascript
function appendToDisplay(value) {
    document.getElementById('display').value += value;
}
function clearDisplay() {
    document.getElementById('display').value = '';
}
function calculate() {
    const display = document.getElementById('display');
    try {
        display.value = eval(display.value);
    } catch (error) {
        display.value = 'Error';
    }
}
```

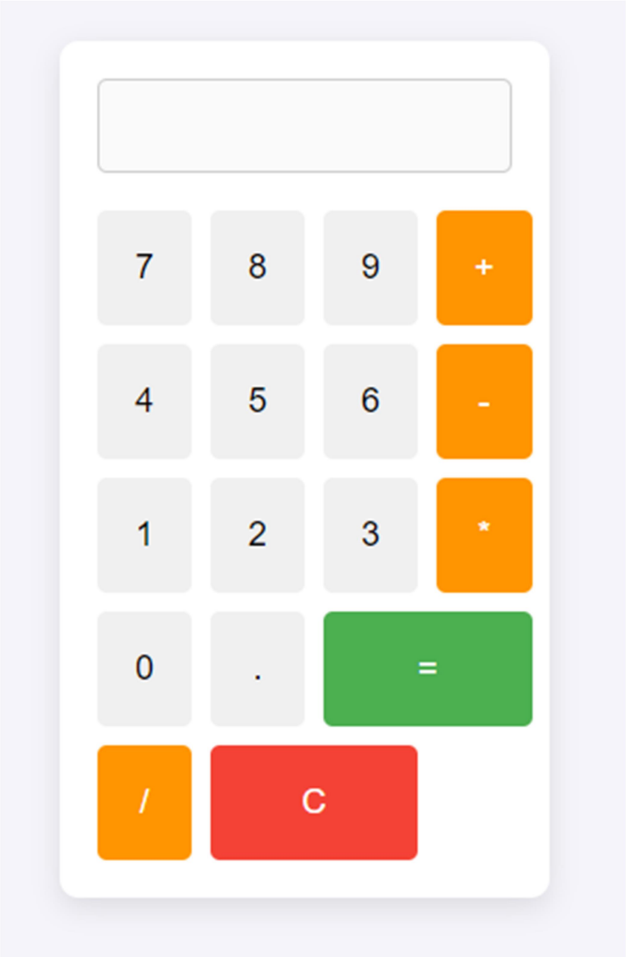
Implementation :

1. Designed the calculator layout using HTML with buttons for digits and operations.

2. Styled the calculator interface using CSS, creating a grid layout for the buttons.

3. Implemented JavaScript to handle button clicks and perform the appropriate mathematical operations.

Output:



Conclusion:

This lab provided a practical understanding of how JavaScript can interact with the DOM to create functional web applications. The task also reinforced the concepts of event handling and dynamic content updates, which are fundamental skills for interactive web development.

Lab 5

Objective: To create a file uploading system using PHP

Theory

File uploading is a common feature in many web applications. PHP can handle file uploads via forms, where users select files to upload, and PHP processes them on the server-side.

HTML provides the form and input elements for selecting files.

PHP handles the server-side logic for processing the uploaded files, checking file types, and saving them to the server.

Code sample:

HTML

lab5.html > ...

```html
<!DOCTYPE html>
<html>
<head>
    <title>File Upload</title>
</head>
<body>
    <h1>Upload a File</h1>
    <form action="upload.php" method="POST" enctype="multipart/form-data">
        <label for="file">Select file to upload:</label>
        <input type="file" name="file" id="file" required>
        <button type="submit" name="submit">Upload</button>
    </form>
</body>
</html>
```

Php

```php
<?php
$targetDir = "uploads/";
if (!is_dir($targetDir)) {
    mkdir($targetDir, 0777, true);
}

$targetFile = $targetDir . basename($_FILES["file"]["name"]);
$uploadOk = 1;
$fileType = strtolower(pathinfo($targetFile, PATHINFO_EXTENSION));
if (isset($_POST["submit"])) {
    if (isset($_FILES["file"])) {
        if ($_FILES["file"]["error"] != 0) {
            echo "Sorry, there was an error uploading your file.";
            $uploadOk = 0;
        }
    }
}
if ($_FILES["file"]["size"] > 5000000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
$allowedFileTypes = array("jpg", "jpeg", "png", "gif", "pdf", "txt");
if (!in_array($fileType, $allowedFileTypes)) {
    echo "Sorry, only JPG, JPEG, PNG, GIF, PDF, and TXT files are allowed.";
```

```php
    $uploadOk = 0;
}
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
} else {
    if (move_uploaded_file($_FILES["file"]["tmp_name"], $targetFile)) {
        echo "The file " . htmlspecialchars(basename($_FILES["file"]["name"])) . " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
```

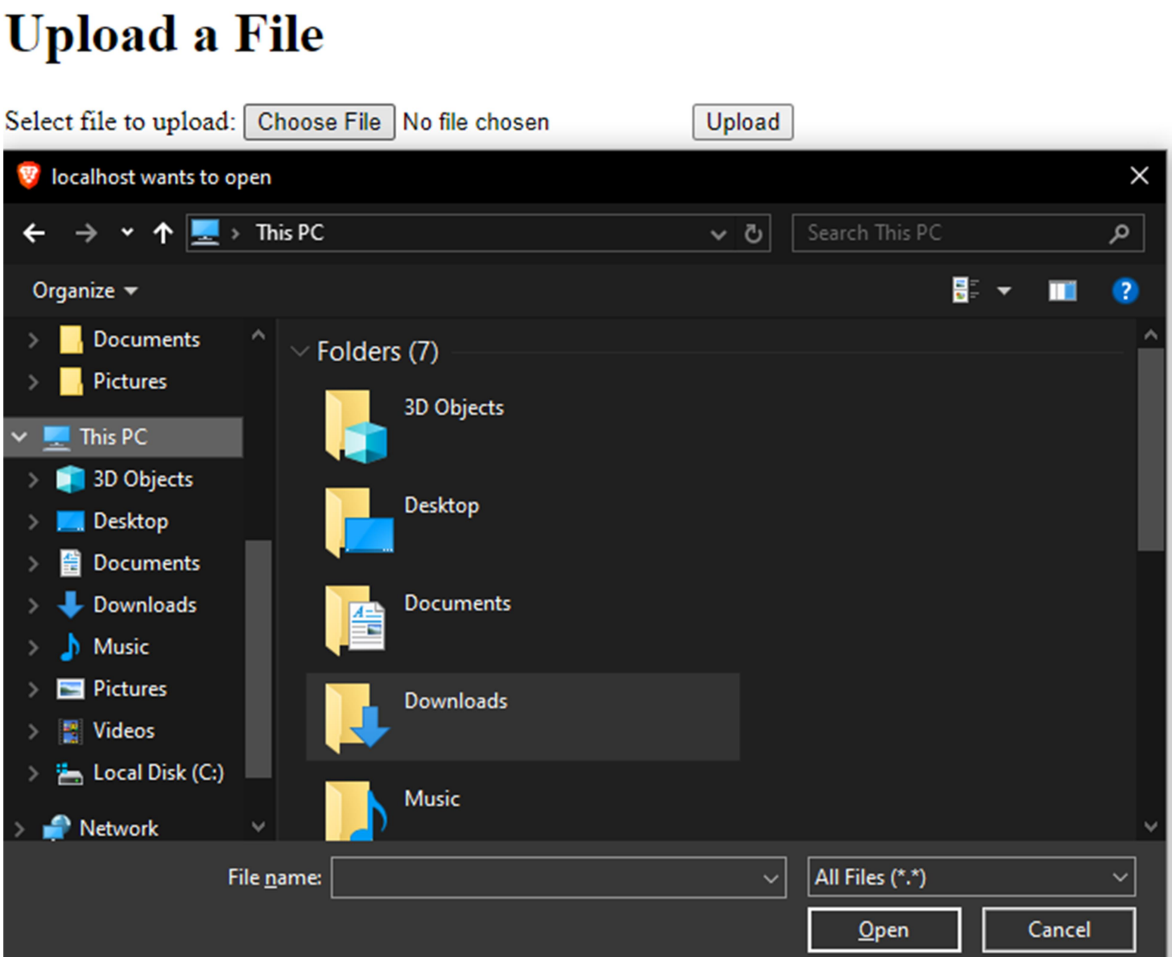
Implementation:

1. Created an HTML form with a file input field that allows users to select a file.

2. Used PHP to handle the form submission, process the file, and store it in a specific directory on the server.

3. Implemented basic validation to ensure only specific file types (e.g., images) are allowed for upload.

Output:



Conclusion:

This lab introduced server-side file handling in PHP, which is crucial for many web applications that need to manage user-generated content. It reinforced PHP's role in file processing and the importance of security when handling user uploads.

Lab 6

Objective: To create a simple webapplication to demonstrate CRUD operation using php and mysql

Theory

CRUD operations are the foundational tasks for managing data in databases. MySQL is a relational database management system, and PHP is often used to interact with MySQL to store, retrieve, and manipulate data.

PHP handles database connections and operations (CRUD).

MySQL stores the data in tables and processes the SQL queries.

Code sample

SQL commands

```
CREATE DATABASE crud_demo;

USE crud_demo;

CREATE TABLE users (

    id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(100) NOT NULL,

    email VARCHAR(100) NOT NULL

);
```

Insert values in the table

```
INSERT INTO users (name, email) VALUES ('Ram', 'ram@example.com');

INSERT INTO users (name, email) VALUES ('Rama', 'rama@example.com');
```

db.php

db.php

```php
<?php
$host = 'localhost';
$dbname = 'crud_demo';
$username = 'root';
$password = '';
try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Could not connect to the database: " . $e->getMessage());
}
?>
```

index.php

```php
<?php
include('db.php');
$sql = "SELECT * FROM users";
$stmt = $pdo->query($sql);
$users = $stmt->fetchAll();
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Users List</title>
</head>
<body>
    <h1>Users List</h1>
    <a href="create.php">Create New User</a>
<table border="1">
<thead>
 <tr>
    <th>Name</th>
    <th>Email</th>
    <th>Actions</th>
</tr>
</thead>
<tbody>
  <?php foreach ($users as $user): ?>
    <tr>
    <td><?php echo htmlspecialchars($user['name']); ?></td>
    <td><?php echo htmlspecialchars($user['email']); ?></td>
        <td>
        <a href="edit.php?id=<?php echo $user['id']; ?>">Edit</a>
        <a href="delete.php?id=<?php echo $user['id']; ?>"
        onclick="return confirm('Are you sure?')">Delete</a>
        </td>
    </tr>
    <?php endforeach; ?>
</tbody>
</table>
</body>
</html>
```

create.php

```php
<?php
include('db.php');
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $name = $_POST['name'];
    $email = $_POST['email'];

    $sql = "INSERT INTO users (name, email) VALUES (:name, :email)";
    $stmt = $pdo->prepare($sql);
    $stmt->execute(['name' => $name, 'email' => $email]);

    header('Location: index.php');
    exit;
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Create User</title>
</head>
<body>
    <h1>Create User</h1>
    <form action="create.php" method="POST">
        <label for="name">Name:</label><br>
        <input type="text" name="name" required><br><br>

        <label for="email">Email:</label><br>
        <input type="email" name="email" required><br><br>

        <input type="submit" value="Create User">
    </form>
    <a href="index.php">Back to List</a>
</body>
</html>
```

Update (edit.php)

edit.php

```php
<?php
include('db.php');
if (isset($_GET['id'])) {
    $id = $_GET['id'];
    $sql = "SELECT * FROM users WHERE id = :id";
    $stmt = $pdo->prepare($sql);
    $stmt->execute(['id' => $id]);
    $user = $stmt->fetch();
    if (!$user) {
        die('User not found.');
    }
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $name = $_POST['name'];
        $email = $_POST['email'];
        $sql = "UPDATE users SET name = :name, email = :email WHERE id = :id";
        $stmt = $pdo->prepare($sql);
        $stmt->execute(['name' => $name, 'email' => $email, 'id' => $id]);
        header('Location: index.php');
        exit;
    }
} else {
    die('Invalid request.');
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Edit User</title>
</head>
<body>
    <h1>Edit User</h1>
    <form action="edit.php?id=<?php echo $user['id']; ?>" method="POST">
        <label for="name">Name:</label><br>
        <input type="text" name="name" value="<?php echo htmlspecialchars($user['name']); ?>" required><br><br>

        <label for="email">Email:</label><br>
        <input type="email" name="email" value="<?php echo htmlspecialchars($user['email']); ?>" required><br><br>

        <input type="submit" value="Update User">
    </form>

    <a href="index.php">Back to List</a>
</body>
</html>
```

delete.php

delete.php

```php
<?php
include('db.php');
if (isset($_GET['id'])) {
    $id = $_GET['id'];
    $sql = "DELETE FROM users WHERE id = :id'
    $stmt = $pdo->prepare($sql);
    $stmt->execute(['id' => $id]);
    header('Location: index.php');
    exit;
} else {
    die('Invalid request.');
}
?>
```

Implementation

1. Set up a MySQL database with a users table for storing user information.

2. Created PHP scripts for inserting new records, retrieving existing records, updating data, and deleting records from the users table.

3. Used SQL queries to execute CRUD operations, with error handling in PHP.

Output:

Users List

Create New User

Name	Email	Actions
Ram	ram@example.com	Edit | Delete
Rama	rama@example.com	Edit | Delete

Create User

Name:
Person

Email:
person@example.com

Create User
Back to List

Users List

Create New User

Name	Email	Actions
Ram	ram@example.com	Edit | Delete
Rama	rama@example.com	Edit | Delete
Person	person@example.com	Edit | Delete

Click edit to update information

Edit User

Name:
Person1

Email:
person@example.com

Update User
Back to List

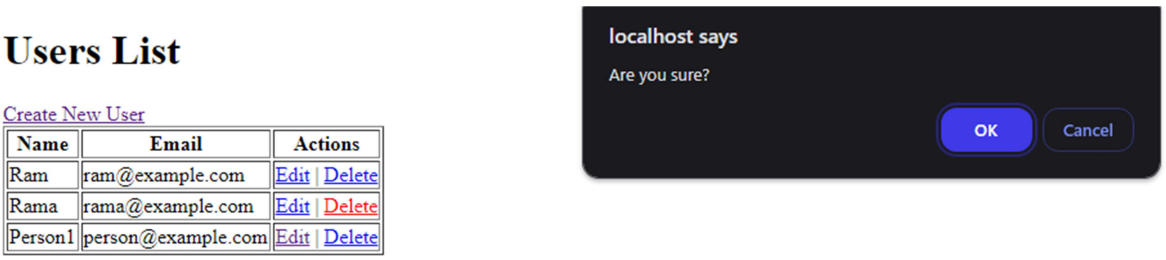
Updated table

Users List

Create New User

Name	Email	Actions
Ram	ram@example.com	Edit | Delete
Rama	rama@example.com	Edit | Delete
Person1	person@example.com	Edit | Delete

Click delete to delete data from table



Updated table

Conclusion

This simple CRUD application demonstrates basic functionality of creating, reading, updating, and deleting records in a MySQL database using PHP. This lab reinforced the fundamental concepts of database interaction in web development

Lab 7

Objective: To demonstrate the given concepts of OOP:

a. Classes and Objects

We define a class Animal with properties $name and $age, and a method makeSound(). The object $dog is created using the new keyword, and its details are accessed via the getDetails() method.

Code

```php
<?php
// A simple class definition with properties and methods.
class Animal {
    public $name;
    public $age;
    // Constructor method
    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
    public function makeSound() {
        return "Some generic sound!";
    }
    public function getDetails() {
        return "Name: {$this->name}, Age: {$this->age}";
    }
}
// Creating an object of class Animal
$dog = new Animal("Dog", 5);
echo $dog->getDetails(); // Output: Name: Dog, Age: 5
echo "<br>";
echo $dog->makeSound(); // Output: Some generic sound!
echo "<br>";
```

Output :

Name: Dog, Age: 5
Some generic sound!

b. Overloading

Overloading refers to defining multiple methods or properties with the same name but different parameters. Overloading allows us to create dynamic methods.

Code:

```php
<?php
class Car {
    private $properties = [];
    public function __set($name, $value) {
        $this->properties[$name] = $value;
    }

    public function __get($name) {
        return isset($this->properties[$name]) ? $this->properties[$name] : null;
    }

    public function __call($name, $arguments) {
        return "Method {$name} does not exist!";
    }
}

$car = new Car();
$car->color = "Red";
echo $car->color;
echo "<br>";
echo $car->getModel();
```

Output:

Red
Method getModel does not exist!

---

c. Overriding

Overriding means changing or redefining a method in a subclass that was already defined in its parent class. It's changing the implementation of a parent class method in the child class.

Code :

```php
<?php

class Dog extends Animal {
    public function makeSound() {
        return "Bark!";
    }
}
$dog = new Dog("Bulldog", 3);
echo $dog->getDetails();
echo "<br>";
echo $dog->makeSound();
```

Output

Name: Bulldog, Age: 3
Bark!

---

d. Inheritance:

When a class derives from another class i.e., parent-child relationship. Inheritance allows a class (child class) to inherit methods and properties from another class (parent class).

Code:

```php
class Cat extends Animal {
    public function makeSound() {
        return "Meow!";
    }
}

$cat = new Cat("Kitty", 2);
echo $cat->getDetails();
echo "<br>";
echo $cat->makeSound();
```

Output

Name: Kitty, Age: 2
Meow!

---

e. Static Keyword

Static keyword is used to declare class properties or methods that can be accessed without creating an instance of the class.

Code:

```php
class MathUtility {
    public static $pi = 3.14159;
       public static function calculateCircleArea($radius) {
        return self::$pi * $radius * $radius;
    }
}

echo MathUtility::$pi;
echo "<br>";
echo MathUtility::calculateCircleArea(5);
echo "<br>";
?>
```

Output:

3.14159
78.53975

Conclusion

This task reinforced my understanding of OOP principles in PHP. It showcased how OOP helps structure code more efficiently and makes it easier to scale and maintain web applications.

Lab 8

Objective: To demonstrate the concept of access modifiers in OOP

Theory

In OOP, access modifiers are used to set the visibility of class properties and methods. The three main access modifiers in PHP are:

Public: Members declared as public can be accessed from anywhere, both inside and outside the class.

Protected: Members declared as protected can be accessed within the class and by classes derived from it (subclasses), but not from outside the class.

Private: Members declared as private can only be accessed within the class itself and not from outside or from subclasses.

Php code:

```
<?php

// Base class

class Person {

    public $name;

    protected $age;

    private $password;

    public function __construct($name, $age, $password) {

        $this->name = $name;

        $this->age = $age;

        $this->password = $password;

    }

    public function showInfo() {

        echo "Name: " . $this->name . "<br>";

        echo "Age: " . $this->age . "<br>";

    }

    private function showPassword() {

        echo "Password: " . $this->password . "<br>";

    }

}
```

```
// Derived class
class Employee extends Person {
    public function showGreeting() {
        echo "Hello, " . $this->name . "!<br>";
    }
}
$person = new Person("Ram", 30, "password123");
echo $person->name . "<br>";
$person->showInfo();

echo "<br>";

// Creating an object of Employee class
$employee = new Employee("Shyam", 25, "password");
$employee->showGreeting();
?>
```

DEMONSTRATION:

In this program:

The Person class has public ($name), protected ($age), and private ($password) properties.

The Employee class extends Person and demonstrates how protected properties and methods are accessible in a derived class, but private ones are not.

Conclusion

This lab exercise provided a clear understanding of access modifiers in PHP and their role in OOP. The three types of access modifiers control the visibility and accessibility of class properties and methods

Lab 9

Objective: To create a user registration form with client-side and server-side validation

Theory

Client-Side Validation: Performed on the user's browser before the data is sent to the server. It improves user experience but should not be the only layer of validation since it can be bypassed.

Server-Side Validation: Performed on the server to ensure the data is valid and secure before processing it, regardless of what the client sends. This is crucial for security.

Code Sample

HTML:

```html
<!DOCTYPE html>
<html>
<head>
    <title>User Registration</title>
    <script>
        function validateForm() {
            var name = document.getElementById("name").value;
            var email = document.getElementById("email").value;
            var password = document.getElementById("password").value;
            var gender = document.querySelector('input[name="gender"]:checked');
            var errorMessage = "";

            // Client-side validation
            if (name == "") {
                errorMessage += "Name must be filled out\n";
            }
            if (email == "" || !email.includes("@")) {
                errorMessage += "Please enter a valid email\n";
            }
            if (password == "") {
                errorMessage += "Password must be filled out\n";
            } else if (password.length < 6) {
                errorMessage += "Password must be at least 6 characters long\n";
            }
            if (!gender) {
                errorMessage += "Please select your gender\n";
            }
            if (errorMessage != "") {
                alert(errorMessage);
                return false; // Prevent form submission if validation fails
            }
            return true;
        }
    </script>
</head>
<body>
    <h2>User Registration Form</h2>
    <form method="post" action="register.php" onsubmit="return validateForm()">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br><br>
```

```
        <label for="email">Email:</label>
        <input type="email" id="email" name="email"><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password"><br><br>

        <label>Gender:</label>
        <input type="radio" id="male" name="gender" value="male">
        <label for="male">Male</label>
        <input type="radio" id="female" name="gender" value="female">
        <label for="female">Female</label><br><br>

        <input type="submit" value="Register">
    </form>
</body>
</html>
```

Php:

```
<?php
// Server-side validation and processing

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['name'];
    $email = $_POST['email'];
    $password = $_POST['password'];
    $gender = $_POST['gender'];

    $errors = [];
    if (empty($name)) {
        $errors[] = "Name is required.";
    }
    if (empty($email) || !filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $errors[] = "Please enter a valid email.";
    }
    if (empty($password)) {
        $errors[] = "Password is required.";
    } elseif (strlen($password) < 6) {
        $errors[] = "Password must be at least 6 characters long.";
    }
    if (empty($gender)) {
        $errors[] = "Gender is required.";
    }
    if (!empty($errors)) {
        foreach ($errors as $error) {
            echo "<p style='color:red;'>$error</p>";
        }
    } else {
        echo "<p style='color:green;'>Registration successful!</p>";
    }
}
?>
```

Observation:

Client-side verification

- When the user submits the form, JavaScript checks if the **name**, **email**, **password**, and **gender** fields are filled out correctly.
- If any field is invalid, an alert is displayed, and the form is not submitted.

Server-side verification:

- After the form passes client-side validation, it is submitted to register.php. register.php checks the form data on the server
- If any validation fails, appropriate error messages are displayed. If the validation passes, a success message is shown.

Conclusion

This lab demonstrated the creation of a user registration form with both client-side validation and server-side validation. Client-side validation enhances user experience by providing immediate feedback, while server-side validation ensures data integrity and security

Lab 10

Objective: To create database schema and write queries

Use MySQL to create following database schema and write queries.

users(uid, name, email, contact, gender, age)

products(pid, pname, description, price, stock)

sales(sid, uid, pid, quantity, total_price, date)

Write MySQL queries to:

a. Write a query to find users aged between 18 and 30.

```
SELECT *
FROM users
WHERE age BETWEEN 18 AND 30;
```

b. Write a query to find the product ID, name, and stock for products where the stock is less than 10.

```
SELECT pid, pname, stock
FROM products
WHERE stock < 10;
```

c. Write a query to fetch all sales related to the user with uid = 2.

```
SELECT *
FROM sales
WHERE uid = 2;
```

d. Write a query to display the product ID, product name and the total quantity sold for each product.

```
SELECT p.pid, p.pname, SUM(s.quantity) AS total_quantity_sold
FROM sales s
JOIN products p ON s.pid = p.pid
```

GROUP BY p.pid, p.pname;

e. Write a query to calculate the total revenue generated from all sales.

SELECT SUM(total_price) AS total_revenue

FROM sales;

f. Write a query to count how many male and female users are in the users table.

SELECT gender, COUNT(*) AS count

FROM users

GROUP BY gender;

g. Write a query to fetch the user name, product name, quantity, and total price for each sale using joins.

SELECT u.name AS user_name, p.pname AS product_name, s.quantity, s.total_price

FROM sales s

JOIN users u ON s.uid = u.uid

JOIN products p ON s.pid = p.pid;

h. Write a query to find products that have not been sold yet.

SELECT p.pid, p.pname

FROM products p

LEFT JOIN sales s ON p.pid = s.pid

WHERE s.pid IS NULL;

i. Write a query to find the product ID and name of the product with the highest quantity sold.

SELECT p.pid, p.pname

FROM products p

JOIN sales s ON p.pid = s.pid

GROUP BY p.pid, p.pname

ORDER BY SUM(s.quantity) DESC

LIMIT 1;

j. Write a query to find all users who have not made any purchases.

SELECT u.uid, u.name, u.email

FROM users u

LEFT JOIN sales s ON u.uid = s.uid

WHERE s.uid IS NULL;

k. Write a query to display each user's ID, name, and total amount spent on purchases.

SELECT u.uid, u.name, u.email

FROM users u

LEFT JOIN sales s ON u.uid = s.uid

WHERE s.uid IS NULL;