

LAB MANUAL

FOR

WEB DESIGN LAB

5TH Semester

Diploma in Computer Science & Engineering



C. V. Raman Polytechnic

Bidya Nagar, Mahura, Janla, Bhubaneswar

CONTENTS

SL NO.	List of practical's	PG NO.
1	Design PHP based web pages using correct PHP, CSS and XHTML syntax Structure.	1-2
2	Create Web Form and pages that properly use HTTP GET and POST Protocol as appropriate	3-5
3	SQL language within MySQL and PHP to access and manipulate database.	6-8
4	Install and configure both PHP and MySQL.	9-12
5	Create PHP code that utilizes the commonly used API library function built in to PHP.	13-14
6	Design and Create a complete web site that demonstrates good PHP/MySQLClient/Server design.	15-23
7	To store a cookies using PHP on client	24-27
8	To save the user session on server side	28-31
9	. Design website.	32-33

Experiment-1

AIM:Design PHP based web pages using correct PHP,CSS andXHTML,syntax,structure

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="description" content="just a test website for learning html, css
and php">
<title>My great website </title>
</head>
<body>
<header>
<nav id="main-navigation">
<ul>
<li><a href="index.php">home</a></li>
<li><a href="index.php">page 2</a></li>
<li><a href="index.php">page 3</a></li>
</ul>
</nav>
</header>
<div id="main-contents">
This is page 1,the home page .
</div>
```

<footer>

</footer>

</body>

</html>

Experiment-2

AIM: Creating a web form that utilizes HTTP GET and POST methods is a fundamental part of web development. Here's a simple HTML form that demonstrates the use of both GET and POST methods and how they send data to a server:

HTML Form for GET and POST:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>HTTP GET and POST Form</title>
</head>
<body>

  <h2>HTTP GET Form</h2>
  <form action="get_handler.php" method="GET">
    <label for="get_name">Name:</label>
    <input type="text" id="get_name" name="name">
    <input type="submit" value="Submit GET">
  </form>

  <h2>HTTP POST Form</h2>
  <form action="post_handler.php" method="POST">
    <label for="post_name">Name:</label>

    <input type="text" id="post_name" name="name">
    <input type="submit" value="Submit POST">
```

```
</form>
</body>
</html>
```

Explanation:

The first form uses the GET method, where form data is sent in the URL's query parameters.

The second form uses the POST method, where form data is sent in the HTTP request body.

For the server-side handling, you'd need scripts (get_handler.php and post_handler.php in this case) to process the form data accordingly.

PHP Example for Handling GET:

```
<!-- get_handler.php -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>GET Handler</title>
</head>
<body>
<?php
if(isset($_GET['name'])) {
  $name = $_GET['name'];
  echo "<p>Hello, $name! You submitted this form using GET method.</p>";
}
```

```
?>
```

```
</body>
```

```
</html>
```

PHP Example for Handling POST:

```
<!-- post_handler.php -->
```

```
<!DOCTYPE html>
```

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

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>POST Handler</title>
```

```
</head>
```

```
<body>
```

```
<?php
```

```
if(isset($_POST['name'])) {
```

```
  $name = $_POST['name'];
```

```
  echo "<p>Hello, $name! You submitted this form using POST method.</p>";
```

```
}
```

```
?>
```

```
</body>
```

```
</html>
```

Experiment-3

Aim:SQL language within MySQL and PHP to access and manipulate database.

Certainly! An experiment involving SQL, MySQL, and PHP to access and manipulate a database can be a great learning experience. Here's a general outline of how you might approach such an experiment:

Experiment Overview:

Objective: To create a simple web application using PHP to interact with a MySQL database using SQL queries.

Steps:

1.Setup:

Install and set up a local development environment with MySQL (database server), PHP, and a web server (like Apache or Nginx).

Create a database in MySQL using the command line or a tool like phpMyAdmin. For instance:

Sql:

```
CREATE DATABASE mydatabase;
```

2.Connect PHP to MySQL:

Write a PHP script to connect to the MySQL database using the mysqli extension.

Establish a connection to the MySQL database:

```
$servername = "localhost";
```



```
$username = "your_username";
```

```
$password = "your_password";
```

```
$dbname = "mydatabase";
```

```
$conn = new mysqli($servername, $username, $password, $dbname);
```

```
if ($conn->connect_error) {
```

```
    die("Connection failed: " . $conn->connect_error);
```

```
}
```

3. Create a Table and Insert Data:

- Use PHP to Create a table in the database and insert sample data .For example

```
$sql = "CREATE TABLE users (
```

```
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
```

```
    firstname VARCHAR(30) NOT NULL,
```

```
    lastname VARCHAR(30) NOT NULL,
```

```
    email VARCHAR(50),
```

```
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
    CURRENT_TIMESTAMP
```

```
);
```

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

```
    echo "Table created successfully";
```

```
} else {
```

```
    echo "Error creating table: " . $conn->error;
```

```
}
```

4. Perform SQL Operation :

Use PHP to Perform SQL operation like SELECT , INSERT,UPDATE, and DELETE:

- Select Data from the database.

```
$sql = "SELECT id, firstname, lastname FROM users";
```

```
$result = $conn->query($sql);
```

```
if ($result->num_rows > 0) {
```

```
    while ($row = $result->fetch_assoc()) {
```

```
        echo "ID: " . $row["id"] . " - Name: " . $row["firstname"] . " " . $row["lastname"]  
        . "<br>";
```

```
    }
```

```
} else {
```

```
    echo "0 results";
```

```
}
```

5. Close Connection :

- Close The MySQL connection When done.

```
$conn->close();
```

Experiment-4

Aim: Install and configure both PHP and MySQL

Installing PHP:

UpdateSystemRepositories

```
bash
```

```
sudo apt update
```

2.install PHP

```
bash
```

```
sudo apt install php
```

3.Verify PHP Installation:

```
Bash
```

```
php -version
```

installing MySQL:

INSTALL MySQL Server:

```
Bash
```

```
sudo apt install mysql-server
```

2.Secure MySQL Installation (Recommended) : After Installation , You can secure your MySQL Installation by running :

```
Bash:
```

```
sudo mysql_secure_installation
```

this process will guide you through steps like setting a root password , removing anonymous users, disallowing remote root login,etc

3.Start/Enable MySQLService

```
bash
```

```
sudo systemctl start mysql
```

```
sudo systemctl enable mysql
```

4.Access MySQL: you can MySQL by typing sudo mysql

```
sudo mysql
```

You'll log in using the root user and password you set during the secure installation

Configuring MySQL and PHP

Install PHP Extensions for MySQL:

```
bash
```

```
sudo apt install php-mysql
```

This installs the necessary PHP extension to interact with MySQL databases.

2.Configure PHP.ini:

Locate and modify your php.ini file (often found in /etc/php/{version}/apache2/php.ini or /etc/php/{version}/cli/php.ini). Ensure these lines are uncommented or added:

```
extension=mysqli
```

```
extension=pdo_mysql
```

3.Restart Apache (if using it with PHP):

If you're using PHP with Apache, restart Apache for the changes to take effect:

```
bash
sudo systemctl restart apache2
```

Testing PHP and MySQL:

1. Create a PHP File to Test:

Create a file named `phpinfo.php` in your web server's root directory (often `/var/www/html/` for Apache) with the following content:

```
php
<?php
phpinfo();
?>
```

Access this file from your web browser (e.g., `http://your_server_ip/phpinfo.php`) to ensure PHP is configured correctly.

2. Test MySQL Connection via PHP:

Create a PHP file (e.g., `test_mysql.php`) with code to connect and test MySQL:

```
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
```

```
$conn = new mysqli($servername, $username, $password);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

echo "Connected successfully";

$conn->close();

?>
```

Replace your_username and your_password with your MySQL credentials.
Access this file via your browser to test the MySQL connection.

These steps will help you install, configure, and test both PHP and MySQL on your system. Adjust paths and configurations according to your specific setup if needed.

Experiment-5

Aim: Create PHP code that utilizes the commonly used API library function built in to PHP.

PHP provides various built-in functions to interact with APIs. Let's create an example that utilizes the `file_get_contents()` function to fetch data from a public API.

For this demonstration, let's use the Chuck Norris Jokes API to retrieve a random joke.

PHP Code Using `file_get_contents()` to Fetch API Data:

```
php
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Chuck Norris Joke</title>
</head>
<body>
  <h2>Chuck Norris Joke</h2>

  <?php
  // API Endpoint URL
  $apiUrl = 'https://api.chucknorris.io/jokes/random';
  // Fetch data from the API
```

```

$jokeData = file_get_contents($apiUrl);

// Decode JSON response to an associative array
$joke = json_decode($jokeData, true);

// Display the retrieved joke
if ($joke && isset($joke['value'])) {
    echo "<p>{$joke['value']}</p>";
} else {
    echo "<p>Failed to retrieve a joke. Try again later.</p>";
}
?>
</body>
</html>

```

This PHP script fetches a random Chuck Norris joke from the Chuck Norris Jokes API (<https://api.chucknorris.io/jokes/random>) using `file_get_contents()`. It then decodes the JSON response into an associative array and displays the joke on a web page.

Ensure that your server allows outgoing connections to external URLs. Some hosting environments might have restrictions on this.

The `file_get_contents()` method is just one way to interact with APIs in PHP. For more complex interactions or handling different HTTP methods (GET, POST, PUT, etc.), you might consider using libraries like `cURL` or PHP's `HttpRequest` class.

Remember to run this PHP file in a server environment with PHP enabled. When you access this file through a browser, it will display a Chuck Norris joke fetched from the API.

Experiment-6.

Design and Create a complete web site that demonstrates good PHP/MySQLClient/Server design.

Designing a complete website that showcases good PHP/MySQL client-server architecture involves creating multiple components. Here's a simplified example demonstrating the concept of a user authentication system using PHP and MySQL.

Directory Structure:

markdown

- /your_website_folder

 - /css

 - style.css

 - /includes

 - db.php

 - functions.php

 - /pages

 - login.php

 - register.php

 - dashboard.php

 - index.php

 - logout.php

Database Setup:

Assuming a users table with columns id, username, email, and password.

PHP Files:

1. db.php (Database Connection File)

php

<?php

// Database connection parameters

\$host = "localhost";

\$username = "your_username";

\$password = "your_password";

\$dbname = "your_database_name";

// Create connection

\$conn = new mysqli(\$host, \$username, \$password, \$dbname);

// Check connection

if (\$conn->connect_error) {

 die("Connection failed: " . \$conn->connect_error);

}

?>

2. functions.php (Functions for User Authentication)

php

Copy code

<?php

// Include the database connection

include_once 'db.php';

```
// Function to register a new user
```

```
function registerUser($username, $email, $password) {  
    global $conn;  
  
    $hashedPassword = password_hash($password, PASSWORD_DEFAULT);  
    $sql = "INSERT INTO users (username, email, password) VALUES  
($username', '$email', '$hashedPassword)";  
    $result = $conn->query($sql);  
  
    return $result;  
}
```

```
// Function to check user credentials
```

```
function loginUser($username, $password) {  
    global $conn;  
  
    $sql = "SELECT * FROM users WHERE username = '$username'";  
    $result = $conn->query($sql);  
    if ($result->num_rows == 1) {  
        $user = $result->fetch_assoc();  
        if (password_verify($password, $user['password'])) {  
            return $user;  
        }  
    }  
}
```

```

    return false;
}

?>

3. login.php (Login Page)

php
<?php
session_start();
include_once 'includes/functions.php';

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    $user = loginUser($username, $password);
    if ($user) {
        $_SESSION['user_id'] = $user['id'];
        header('Location: dashboard.php');
        exit();
    } else {
        $error = "Invalid username or password";
    }
}
}

```

```

?>
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
  <div class="container">
    <h2>Login</h2>
    <?php if(isset($error)) { ?>
      <p><?php echo $error; ?></p>
    <?php } ?>
    <form action="" method="post">
      <input type="text" name="username" placeholder="Username" required>
      <input type="password" name="password" placeholder="Password"
required>
      <input type="submit" value="Login">
    </form>
    <p>Don't have an account? <a href="register.php">Register here</a></p>
  </div>
</body>
</html>

```

4. register.php (Registration Page)

Php

```
<?php
```

```

session_start();

include_once 'includes/functions.php';

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $email = $_POST['email'];
    $password = $_POST['password'];
    $result = registerUser($username, $email, $password);

    if ($result) {
        header('Location: login.php');
        exit();
    } else {
        $error = "Registration failed. Please try again.";
    }
}

?>

<!DOCTYPE html>

<html>

<head>

    <title>Register</title>

    <link rel="stylesheet" href="css/style.css">

</head>

<body>

    <div class="container">

```

```

<h2>Register</h2>
<?php if(isset($error)) { ?>
    <p><?php echo $error; ?></p>

<?php } ?>
    <form action="" method="post">
        <input type="text" name="username" placeholder="Username" required>
        <input type="email" name="email" placeholder="Email" required>
        <input type="password" name="password" placeholder="Password"
required>
        <input type="submit" value="Register">
    </form>
    <p>Already have an account? <a href="login.php">Login here</a></p>
</div>
</body>
</html>

```

5. dashboard.php (Dashboard Page)

php

```
<?php
```

```
session_start();
```

```

if (!isset($_SESSION['user_id'])) {
    header('Location: login.php');
    exit();
}

```

```
}  
// Other dashboard content...  
?>  
<!DOCTYPE html>  
<html>  
<head>  
  <title>Dashboard</title>  
  <link rel="stylesheet" href="css/style.css">  
</head>  
<body>  
  <div class="container">  
    <h2>Welcome to Dashboard</h2>  
    <p>Hello, <?php echo $_SESSION['user_id']; ?>! This is your  
dashboard.</p>  
    <a href="logout.php">Logout</a>  
  </div>  
</body>  
</html>
```

6. logout.php (Logout Page)

php

```
<?php  
session_start();  
session_unset();  
session_destroy();  
header('Location: login.php');
```



```
exit();
```

```
?>
```

7. index.php (Main Entry Point)

This file can include some introductory content or redirect users to the login page (`header('Location: login.php');`).

CSS:

You can create a `style.css` file within the `css` folder to define styles for your website components.

This setup demonstrates a basic user authentication system using PHP and MySQL. Users can register, log in, access a dashboard upon successful login, and log out. It's essential to further enhance security by implementing features like password hashing, validation, error handling, and session management based on your application's requirements.

Experiment 7.

Storing cookies on the client-side using PHP involves setting specific headers in your PHP script. Here's an example of how you can create a cookie using PHP:

PHP Code to Set a Cookie:

```
php
```

```
<?php
```

```
// Set a cookie with name 'user' and value 'John Doe' that expires in 1 hour (3600 seconds)
```

```
setcookie('user', 'John Doe', time() + 3600, '/'); // '/' makes the cookie available across the entire domain
```

```
echo "Cookie 'user' is set.";
```

```
?>
```

This code uses the `setcookie()` function in PHP to create a cookie named 'user' with the value 'John Doe'. The cookie will expire in 1 hour (3600 seconds) and will be available across the entire domain due to the '/' path.

Retrieving and Using the Stored Cookie:

You can retrieve and use the stored cookie in subsequent requests using `$_COOKIE` superglobal array in PHP.

For example:

```
php
```

```
<?php
```

```
// Check if the cookie is set and retrieve its value
```

```
if(isset($_COOKIE['user'])) {  
    $username = $_COOKIE['user'];  
    echo "Welcome back, $username!";  
} else {  
    echo "Cookie 'user' is not set.";  
}  
?>
```

This code checks if the 'user' cookie is set and if so, retrieves its value from the `$_COOKIE` array. If the cookie is set, it greets the user; otherwise, it notifies that the cookie is not set.

Remember:

Cookies set via PHP are sent to the client's browser in the response headers.

The browser then automatically includes these cookies in subsequent requests to the same domain.

Cookies can store limited amounts of data, and sensitive information should not be stored in them for security reasons.

Ensure sensitive data is encrypted or handled securely on the server-side.

When executing this PHP code in a web server environment, the 'user' cookie will be set in the client's browser and can be accessed and utilized on subsequent requests.

Storing cookies on the client-side using PHP involves setting specific headers in your PHP script. Here's an example of how you can create a cookie using PHP:

PHP Code to Set a Cookie

```
php
<?php
// Set a cookie with name 'user' and value 'John Doe' that expires in 1 hour (3600
seconds)

setcookie('user', 'John Doe', time() + 3600, '/'); // '/' makes the cookie available
across the entire domain

echo "Cookie 'user' is set.";

?>
```

This code uses the `setcookie()` function in PHP to create a cookie named 'user' with the value 'John Doe'. The cookie will expire in 1 hour (3600 seconds) and will be available across the entire domain due to the '/' path.

Retrieving and Using the Stored Cookie:

You can retrieve and use the stored cookie in subsequent requests using `$_COOKIE` superglobal array in PHP.

For example:

```
php
<?php
// Check if the cookie is set and retrieve its value
if(isset($_COOKIE['user'])) {
    $username = $_COOKIE['user'];
    echo "Welcome back, $username!";
}
```

```
}  
else {  
    echo "Cookie 'user' is not set.";  
}  
?>
```

This code checks if the 'user' cookie is set and if so, retrieves its value from the `$_COOKIE` array. If the cookie is set, it greets the user; otherwise, it notifies that the cookie is not set.

Remember:

Cookies set via PHP are sent to the client's browser in the response headers.

The browser then automatically includes these cookies in subsequent requests to the same domain.

Cookies can store limited amounts of data, and sensitive information should not be stored in them for security reasons.

Ensure sensitive data is encrypted or handled securely on the server-side.

When executing this PHP code in a web server environment, the 'user' cookie will be set in the client's browser and can be accessed and utilized on subsequent requests.

Experiment-8

Aim: To save the user session on server side.

Managing user sessions on the server-side using PHP involves utilizing session handling functions to store and retrieve session data. Here's an example of how you can save user session data on the server:

PHP Code to Save User Session:

Starting a Session:

At the beginning of each PHP file where you want to work with sessions, you should start the session using `session_start()`.

For example:

php

Copy code

```
<?php
```

```
session_start();
```

```
?>
```

Storing Data in Session Variables:

You can store user-related data in session variables using the `$_SESSION` superglobal array.

For instance, after a user logs in and you authenticate them, you might want to store their user ID in a session variable:

Php

```
<?php
// Assume $user_id is the authenticated user's ID
session_start();

// Storing user ID in a session variable
$_SESSION['user_id'] = $user_id;

// Redirect the user to their dashboard or another authenticated page
header('Location: dashboard.php');
exit();
?>
```

Accessing Session Data:

Once the session is started and data is stored, you can access this data throughout the user's session using `$_SESSION`.

For example, in the `dashboard.php` page:

```
php
<?php
session_start();

// Checking if the user is logged in
if(isset($_SESSION['user_id'])) {
    $user_id = $_SESSION['user_id'];
```

```
echo "Welcome! Your user ID is: $user_id";  
} else {  
    // Redirect to login page or handle unauthorized access  
    header('Location: login.php');  
    exit();  
}  
?>
```

Destroying the Session:

When the user logs out or when the session needs to be destroyed for any reason, you can use `session_destroy()` to clear the session data.

For instance, in the `logout.php` page:

```
php  
<?php  
session_start();  
  
// Unset all session variables  
session_unset();  
  
// Destroy the session  
session_destroy();  
  
// Redirect to the login page or any other desired page after logout  
header('Location: login.php');
```



```
exit();
```

```
?>
```

This way, PHP manages the session data on the server-side, providing a way to store and retrieve user-specific information throughout the user's visit to your website. Always ensure to use sessions securely and handle sensitive data appropriately.

Experiment 9

Designing a website involves several steps, from planning and wireframing to actual implementation using HTML, CSS, and potentially other languages like JavaScript for interactivity. Here's a simplified walkthrough of the process:

1. Planning and Research:

Define Purpose: Determine the purpose of the website (e.g., portfolio, blog, e-commerce, informational).

Target Audience: Identify the audience and their needs to tailor the design and content.

Content Structure: Plan the site's structure, including pages, navigation, and hierarchy.

2. Wireframing and Mockups:

Create rough sketches or digital wireframes of the site's layout and structure using tools like Figma, Sketch, or even pen and paper.

Develop mockups and prototypes to visualize the site's look and feel.

3. Designing with HTML/CSS:

HTML Structure: Create the basic structure of your site using HTML, defining the layout with semantic elements like `<header>`, `<nav>`, `<main>`, `<footer>`, etc.

CSS Styling: Apply styles using CSS to make your site visually appealing. Use CSS frameworks like Bootstrap or custom CSS for design consistency.

4. Responsive Design:

Ensure your website is responsive and works well on various devices (desktops, tablets, mobile phones) using media queries and responsive design practices.

5. Adding Interactivity (Optional):

JavaScript: Implement JavaScript for interactive elements, animations, form validation, etc.

Frameworks/Libraries: Use libraries like jQuery, React, or Vue.js for more complex functionality if needed.

6. Testing:

Test your website across different browsers and devices to ensure compatibility and functionality.

Check for usability, responsiveness, and performance. Debug any issues that arise.

7. Deployment:

Choose a web hosting service and upload your files (HTML, CSS, JavaScript) to make your website live.

Register a domain name if necessary and configure your hosting settings.

8. Ongoing Maintenance:

Regularly update content, fix bugs, and improve user experience based on feedback and analytics.

Consider SEO practices to improve the site's visibility on search engines.

This is a high-level overview, and each step might involve more detailed tasks and considerations based on the specific requirements of your website. The process often involves a mix of creativity, technical skills, and attention to user experience.