**ADVANCED SERVER-SIDE CONCEPTS**

**(Skill Advanced Course-I)**

**WEB APPLICATION DEVELOPMENT**

**SDC MANUAL**

**AIMLDS**

**(R20)**

**III B-TECH – I SEM**

**(2024‐2025)**



**QIS COLLEGE OF ENGINEERING & TECHNOLOGY**

**(Autonomous Institution – UGC, Govt. of India)**

Recognized under 2(f) and 12 (B) of UGC ACT 1956

Affiliated to JNTUK, KAKINADA, Approved by AICTE ‐ Accredited by NAAC – ‘A’ Grade ‐ ISO 9001:2015 Certified) Pondur Road, Vengamukka Palem,

ONGOLE – 523 272, Andhra Pradesh State, India

# DEPARTMENT OF AIMLDS

**Name of the Student: ……………………………………………………………….………………………….**

**Class: ………………………………….……….. Roll Number: ………………………………………….…….**

**Year……………… ………………………………………Semester………………………………………….………..**

**Branch & Section: .............. ………………………………………………………………………………………..**

**Stream:…………………………………………………………………………………………………………………………**

**During the Academic Year 2024- 2025 .**

**Staff in-charge SDC Co-ordinator**

**Hod ExternalExaminer**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **III B.Tech.** | **ADVANCED SERVER-SIDE CONCEPTS**  **(Skill Advanced Course-I)** | **L** | **T** | **P** | **C** |
| **I Semester** | **1** | **0** | **2** | **2** |

**COURSE OBJECTIVES**

1. Understand how server-side programming works on the web.
2. How to receive and process form submission data.

**COURSE OUTCOMES**

Student will be able to

1. Write PHP scripts to handle HTML forms.
2. Write regular expressions including modifiers, operators, and Meta characters.
3. Create PHP programs that use various PHP library functions, and that manipulate files and directories.
4. Analyze and solve common Web application tasks by writing PHP programs.

**EXPERIMENTS**

1. Explain the technologies involved in AJAX.
2. Explain about JQUERY events and selectors.
3. Write a PHP program to display a digital clock which displays the current time of the server
4. Write a PHP program to keep track of the number of visitors visiting the web page

and to display this count of visitors, with proper headings

1. Write the PHP programs to do the following:

a. Implement simple calculator operations.

b. Find the transpose of a matrix.

c. Multiplication of two matrices

1. Write a PHP Program to implement file upload and download operations.
2. Write a PHP program to perform following operations in cookies
3. Adding a cookie
4. Deleting a cookie

8. Write a PHP program to implement php form handling.

**TEXTBOOKS**

1. The Joy of PHP Programming: A Beginner's Guide – by Alan Forbes.
2. PHP & MySQL Novice to Ninja – by Kevin Yank.
3. Headfirst PHP & MySQL – by Lynn Beighley & Michael Morrison.

**Advanced Server-side Concepts**

### Basics of HTML & CSS

#### HTML Basics

**Introduction to HTML** HTML (HyperText Markup Language) is the standard language for creating web pages and web applications. It describes the structure of a web page using markup.

**HTML Document Structure**

html

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

</head>

<body>

<h1>This is a Heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

* <!DOCTYPE html>: Declares the document type and HTML version.
* <html>: Root element of an HTML page.
* <head>: Contains meta-information about the document.
* <body>: Contains the visible page content.

**HTML Elements and Tags** HTML elements are defined by tags, which can have attributes providing additional information.

* <h1> to <h6>: Heading tags
* <p>: Paragraph tag
* <a href="URL">: Anchor tag for links
* <img src="image.jpg" alt="description">: Image tag

**Creating Links and Images**

* Links: <a href="http://example.com">This is a link</a>
* Images: <img src="path/to/image.jpg" alt="Description">

**Tables and Lists**

* Tables:

html

<table>

<tr>

<th>Header 1</th>

<th>Header 2</th>

</tr>

<tr>

<td>Data 1</td>

<td>Data 2</td>

</tr>

</table>

* Lists:
  + Ordered List:

html

<ol>

<li>Item 1</li>

<li>Item 2</li>

</ol>

* + Unordered List:

html

<ul>

<li>Item 1</li>

<li>Item 2</li>

</ul>

**Forms and Input Elements** Forms are used to collect user input:

html

<form action="submit.php" method="post">

<label for="name">Name:</label>

<input type="text" id="name" name="name">

<input type="submit" value="Submit">

</form>

* Form elements include <input>, <textarea>, <button>, <select>, and others.

#### CSS Basics

**Introduction to CSS** CSS (Cascading Style Sheets) is used to style and layout web pages — for example, to alter the font, color, size, and spacing of your content.

**CSS Syntax and Selectors** CSS rules are composed of selectors and declaration blocks.

css

selector {

property: value;

}

* Selectors target HTML elements.
* Declaration blocks contain one or more declarations separated by semicolons.

**CSS Box Model** The box model describes the structure of an element, including margins, borders, padding, and the actual content.

* margin: Outside space.
* border: Surrounds padding and content.
* padding: Space around content.
* content: The actual content.

**Positioning and Layout** CSS positioning allows you to specify the position of an element in a document.

* static: Default positioning.
* relative: Positioned relative to its normal position.
* absolute: Positioned relative to the nearest positioned ancestor.
* fixed: Positioned relative to the browser window.
* float: Floats an element to the left or right.

**Styling Text and Fonts**

css

p {

color: blue;

font-family: Arial, sans-serif;

font-size: 16px;

}

* color: Text color.
* font-family: Font type.
* font-size: Size of the font.

### Introduction to PHP Basics

#### a. Overview of PHP (What is PHP, why use it?)

**What is PHP?**

* PHP (Hypertext Preprocessor) is a server-side scripting language designed for web development but also used as a general-purpose programming language.
* PHP scripts are executed on the server, and the result is returned to the browser as plain HTML.

**Why Use PHP?**

* **Open Source**: PHP is free to use and widely supported.
* **Easy to Learn**: PHP syntax is easy to understand for new programmers.
* **Versatile**: PHP can be embedded directly into HTML, making it ideal for web development.
* **Database Integration**: PHP supports a wide range of databases like MySQL, PostgreSQL, and SQLite.

#### b. Setting up a Local Development Environment

**Installing PHP**

* You can install PHP on your local machine by downloading packages like XAMPP, WAMP, or MAMP, which include PHP, Apache server, and MySQL.

**Example: Setting Up XAMPP**

1. Download XAMPP from [apachefriends.org](https://www.apachefriends.org).
2. Install and run the XAMPP control panel.
3. Start the Apache server and MySQL.

**Verifying the Installation**

* Create a file named info.php in the htdocs directory with the following content:

php

<?php

phpinfo();

?>

* Open a browser and navigate to http://localhost/info.php. You should see the PHP information page.

#### c. Writing Your First PHP Script

**Basic PHP Syntax**

* PHP scripts are enclosed within <?php and ?> tags.

php

<?php

echo "Hello, World!";

?>

**Running a PHP Script**

* Save the script as hello.php in the htdocs directory and navigate to http://localhost/hello.php.

#### d. PHP Syntax Basics (Variables, Data Types, Operators)

**Variables**

* Variables in PHP start with a $ sign followed by the name of the variable.

php

<?php

$txt = "Hello World!";

$number = 5;

$decimal = 5.5;

echo $txt;

?>

**Data Types**

* **String**: A sequence of characters.
* **Integer**: A whole number.
* **Float**: A number with a decimal point.
* **Boolean**: Represents two possible states: TRUE or FALSE.

php

<?php

$str = "Hello";

$int = 123;

$float = 123.45;

$bool = true;

?>

**Operators**

* **Arithmetic Operators**: Used to perform common arithmetic operations.

php

<?php

$x = 10;

$y = 6;

echo $x + $y; // 16

echo $x - $y; // 4

echo $x \* $y; // 60

echo $x / $y; // 1.66666666667

echo $x % $y; // 4

?>

### Control Structures and Functions

#### a. Conditional Statements (if, else, elseif)

**if Statement**

* Executes some code if a condition is true.

php

<?php

$t = date("H");

if ($t < "20") {

echo "Have a good day!";

}

?>

**if...else Statement**

* Executes one block of code if the condition is true and another if it is false.

php

<?php

$t = date("H");

if ($t < "20") {

echo "Have a good day!";

} else {

echo "Have a good night!";

}

?>

**if...elseif...else Statement**

* Executes different blocks of code for multiple conditions.

php

<?php

$t = date("H");

if ($t < "10") {

echo "Have a good morning!";

} elseif ($t < "20") {

echo "Have a good day!";

} else {

echo "Have a good night!";

}

?>

#### b. Loops (for, while, do-while, foreach)

**for Loop**

* Loops through a block of code a specified number of times.

php

<?php

for ($x = 0; $x <= 10; $x++) {

echo "The number is: $x <br>";

}

?>

**while Loop**

* Loops through a block of code as long as the specified condition is true.

php

<?php

$x = 1;

while ($x <= 5) {

echo "The number is: $x <br>";

$x++;

}

?>

**do...while Loop**

* Loops through a block of code once, and then repeats the loop as long as the specified condition is true.

php

<?php

$x = 1;

do {

echo "The number is: $x <br>";

$x++;

} while ($x <= 5);

?>

**foreach Loop**

* Loops through a block of code for each element in an array.

php

<?php

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {

echo "$value <br>";

}

?>

#### c. Functions in PHP (defining functions, passing arguments, returning values)

**Defining a Function**

* A function is a block of statements that can be used repeatedly in a program.

php

<?php

function writeMsg() {

echo "Hello world!";

}

writeMsg(); // Outputs: Hello world!

?>

**Function Arguments**

* Functions can take arguments, which are values that are passed to the function.

php

<?php

function familyName($fname) {

echo "$fname Refsnes.<br>";

}

familyName("Jani");

familyName("Hege");

familyName("Stale");

?>

**Returning Values**

* Functions can return values using the return statement.

php

<?php

function sum($x, $y) {

$z = $x + $y;

return $z;

}

echo "5 + 10 = " . sum(5, 10);

?>

**Variable Scope and Global Variables**

* Variables declared outside a function have global scope, while those inside a function have local scope. Use the global keyword to access global variables inside functions.

php

<?php

$x = 5;

function test() {

global $x;

echo $x;

}

test(); // Outputs: 5

?>

**Anonymous Functions and Closures**

php

<?php

$greet = function($name) {

echo "Hello, $name";

};

$greet("World"); // Outputs: Hello, World

?>

### PHP and HTML Integration

#### a. Embedding PHP in HTML (PHP within HTML Tags, HTML within PHP Blocks)

**Basic Embedding Techniques**

* PHP code can be embedded within HTML using <?php ... ?> tags.

html

<!DOCTYPE html>

<html>

<head>

<title>My First PHP Page</title>

</head>

<body>

<?php echo "<h1>Welcome to my page!</h1>"; ?>

</body>

</html>

**HTML within PHP**

* HTML tags can also be embedded within PHP echo statements.

php

<?php

echo "<h1>Welcome to my page!</h1>";

echo "<p>This is a paragraph.</p>";

?>

**Example: Dynamic Content**

php

<!DOCTYPE html>

<html>

<body>

<h1>My First PHP Page</h1>

<?php

$txt = "PHP";

echo "I love $txt!";

?>

</body>

</html>

#### b. Handling Form Input and Validation with Regular Expressions (Basic Input Validation Techniques)

**Form Input Handling**

* PHP can collect form data using the $\_GET or $\_POST superglobals.

html

<!DOCTYPE html>

<html>

<body>

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">

Name: <input type="text" name="fname">

<input type="submit">

</form>

<?php

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

$name = htmlspecialchars($\_POST['fname']);

echo $name;

}

?>

</body>

</html>

**Basic Input Validation**

* Use filter\_var() and regular expressions for input validation to prevent malicious data.

php

<?php

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

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

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

echo "Valid email format";

} else {

echo "Invalid email format";

}

}

?>

**Regular Expressions**

* Regular expressions are patterns used to match character combinations in strings. PHP provides functions like preg\_match() to perform regex operations.

php

<?php

$pattern = "/^[a-zA-Z ]\*$/";

if (preg\_match($pattern, $name)) {

echo "Name is valid";

} else {

echo "Name is invalid";

}

?>

#### c. Reading and Writing Files in PHP (File Handling Functions)

**Opening and Reading Files**

* Use fopen(), fread(), and fclose() to open and read files.

php

<?php

$file = fopen("webdictionary.txt", "r") or die("Unable to open file!");

echo fread($file, filesize("webdictionary.txt"));

fclose($file);

?>

**Writing to Files**

* Use fopen(), fwrite(), and fclose() to write to files.

php

<?php

$file = fopen("newfile.txt", "w") or die("Unable to open file!");

$txt = "John Doe\n";

fwrite($file, $txt);

$txt = "Jane Doe\n";

fwrite($file, $txt);

fclose($file);

?>

**File Handling Functions**

* fgets(): Reads a single line from a file.
* fgetcsv(): Reads a line from a CSV file.
* file\_get\_contents(): Reads an entire file into a string.
* file\_put\_contents(): Writes data to a file.

php

<?php

$content = file\_get\_contents("webdictionary.txt");

echo $content;

file\_put\_contents("newfile.txt", "Hello, World!");

?>

### PHP Strings and Arrays

#### a. String Manipulation Functions in PHP (Concatenation, Substring, Searching, Replacing)

**Concatenation**

* Use the . operator to concatenate strings.

php

<?php

$txt1 = "Hello";

$txt2 = "World";

echo $txt1 . " " . $txt2;

?>

**Substring**

* Use substr() to extract a substring from a string.

php

<?php

$str = "Hello, World!";

echo substr($str, 0, 5); // Outputs: Hello

?>

**Searching**

* Use strpos() to find the position of the first occurrence of a substring.

php

<?php

$str = "Hello, World!";

echo strpos($str, "World"); // Outputs: 7

?>

**Replacing**

* Use str\_replace() to replace all occurrences of a substring.

php

<?php

$str = "Hello, World!";

echo str\_replace("World", "Dolly", $str); // Outputs: Hello, Dolly!

?>

**String Functions**

* strlen(): Returns the length of a string.
* strtoupper(): Converts a string to uppercase.
* strtolower(): Converts a string to lowercase.
* ucfirst(): Capitalizes the first character of a string.
* lcfirst(): Converts the first character of a string to lowercase.

php

<?php

$str = "hello";

echo strtoupper($str); // Outputs: HELLO

?>

#### b. Array Basics (Indexed Arrays, Associative Arrays, Multidimensional Arrays)

**Indexed Arrays**

* Arrays with numeric indexes.

php

<?php

$cars = array("Volvo", "BMW", "Toyota");

echo $cars[0]; // Outputs: Volvo

?>

**Associative Arrays**

* Arrays with named keys.

php

<?php

$age = array("Peter" => "35", "Ben" => "37", "Joe" => "43");

echo "Peter is " . $age['Peter'] . " years old."; // Outputs: Peter is 35 years old.

?>

**Multidimensional Arrays**

* Arrays containing one or more arrays.

php

<?php

$cars = array(

array("Volvo", 22, 18),

array("BMW", 15, 13),

array("Saab", 5, 2),

array("Land Rover", 17, 15)

);

echo $cars[0][0] . ": In stock: " . $cars[0][1] . ", sold: " . $cars[0][2] . ".<br>";

?>

#### c. Array Functions in PHP (push, pop, merge, sort)

**Array Functions**

* array\_push(): Adds one or more elements to the end of an array.
* array\_pop(): Removes the last element of an array.
* array\_merge(): Merges one or more arrays.
* sort(): Sorts an indexed array in ascending order.
* rsort(): Sorts an indexed array in descending order.
* asort(): Sorts an associative array in ascending order, according to the value.
* ksort(): Sorts an associative array in ascending order, according to the key.

php

<?php

$fruits = array("Apple", "Banana", "Cherry");

array\_push($fruits, "Date");

print\_r($fruits); // Outputs: Array ( [0] => Apple [1] => Banana [2] => Cherry [3] => Date )

array\_pop($fruits);

print\_r($fruits); // Outputs: Array ( [0] => Apple [1] => Banana [2] => Cherry )

$arr1 = array("a" => "red", "b" => "green");

$arr2 = array("c" => "blue", "d" => "yellow");

$result = array\_merge($arr1, $arr2);

print\_r($result); // Outputs: Array ( [a] => red [b] => green [c] => blue [d] => yellow )

sort($fruits);

print\_r($fruits); // Outputs: Array ( [0] => Apple [1] => Banana [2] => Cherry )

?>

### Introduction to MySQL

#### a. Introduction to MySQL and Database Basics

**What is MySQL?**

* MySQL is an open-source relational database management system (RDBMS) that uses SQL (Structured Query Language) for database management.

**Relational Database Concepts**

* **Tables**: Collections of related data entries, consisting of rows and columns.
* **Rows**: Individual records in a table.
* **Columns**: Fields in a table, representing a specific attribute of the record.
* **Primary Key**: A unique identifier for each record in a table.
* **Foreign Key**: A field (or collection of fields) in one table that uniquely identifies a row of another table.

**Common MySQL Commands**

* CREATE DATABASE: Creates a new database.

sql

CREATE DATABASE database\_name;

* USE: Selects a database.

sql

USE database\_name;

* CREATE TABLE: Creates a new table.

sql

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

...

);

* INSERT INTO: Inserts new data into a table.

sql

INSERT INTO table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

* SELECT: Retrieves data from one or more tables.

sql

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* UPDATE: Modifies existing data in a table.

sql

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

* DELETE: Deletes data from a table.

sql

DELETE FROM table\_name

WHERE condition;

#### b. Connecting to MySQL Database Using PHP (mysqli Extension)

**Connecting to MySQL**

* Use the mysqli extension to connect to a MySQL database.

php

<?php

$servername = "localhost";

$username = "username";

$password = "password";

$dbname = "database";

// Create connection

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

// Check connection

if ($conn->connect\_error) {

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

}

echo "Connected successfully";

?>

#### c. Executing SQL Queries from PHP (SELECT, INSERT, UPDATE, DELETE)

**Executing SELECT Queries**

* Fetch data from a database using the mysqli\_query() and mysqli\_fetch\_assoc() functions.

php

<?php

$sql = "SELECT id, firstname, lastname FROM MyGuests";

$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";

}

?>

**Executing INSERT Queries**

* Insert data into a database using the mysqli\_query() function.

php

<?php

$sql = "INSERT INTO MyGuests (firstname, lastname, email)

VALUES ('John', 'Doe', 'john@example.com')";

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

echo "New record created successfully";

} else {

echo "Error: " . $sql . "<br>" . $conn->error;

}

?>

**Executing UPDATE Queries**

* Update existing data in a database using the mysqli\_query() function.

php

<?php

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

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

echo "Record updated successfully";

} else {

echo "Error updating record: " . $conn->error;

}

?>

**Executing DELETE Queries**

* Delete data from a database using the mysqli\_query() function.

php

<?php

$sql = "DELETE FROM MyGuests WHERE id=3";

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

echo "Record deleted successfully";

} else {

echo "Error deleting record: " . $conn->error;

}

?>

#### d. Form Handling with PHP (GET and POST Methods)

**Form Handling with GET Method**

* Use the $\_GET superglobal to collect form data sent with the GET method.

html

<!DOCTYPE html>

<html>

<body>

<form method="get" action="<?php echo $\_SERVER['PHP\_SELF'];?>">

Name: <input type="text" name="fname">

<input type="submit">

</form>

<?php

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

$name = htmlspecialchars($\_GET['fname']);

echo $name;

}

?>

</body>

</html>

**Form Handling with POST Method**

* Use the $\_POST superglobal to collect form data sent with the POST method.

html

<!DOCTYPE html>

<html>

<body>

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">

Name: <input type="text" name="fname">

<input type="submit">

</form>

<?php

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

$name = htmlspecialchars($\_POST['fname']);

echo $name;

}

?>

</body>

</html>

**Advanced Server-Side Concepts**

**LAB EXPERIMENTS**

### 1. Explain the technologies involved in AJAX

AJAX (Asynchronous JavaScript and XML) is a web development technique used to create asynchronous web applications. It allows web pages to update content dynamically by making requests to a web server in the background, without reloading the entire page. The technologies involved in AJAX include:

* **HTML & CSS**: Provide the structure and styling for the web page.
* **JavaScript (JS)**: Essential for implementing AJAX functionality. It makes asynchronous requests to the server and handles responses dynamically.
* **XMLHttpRequest (XHR)**: A built-in JavaScript object that performs asynchronous communication with a web server. It allows sending and receiving data in various formats, not limited to XML.
* **JSON (JavaScript Object Notation)**: Although AJAX can handle XML responses, JSON has become more popular due to its lightweight nature and ease of use with JavaScript.

#### Example Use Case of AJAX:

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>AJAX Example</title>

<script>

function fetchData() {

var xhr = new XMLHttpRequest();

xhr.open('GET', 'data.json', true); // Replace 'data.json' with your server endpoint

xhr.onload = function() {

if (xhr.status >= 200 && xhr.status < 400) {

var data = JSON.parse(xhr.responseText);

document.getElementById('output').innerHTML = data.message;

} else {

console.error('Request failed with status:', xhr.status);

}

};

xhr.onerror = function() {

console.error('Request failed');

};

xhr.send();

}

</script>

</head>

<body>

<button onclick="fetchData()">Fetch Data</button>

<div id="output"></div>

</body>

</html>

#### Explanation:

* **HTML Structure**: Defines a button (<button>) that triggers the AJAX request and a <div> (<div id="output">) where the fetched data will be displayed.
* **JavaScript Function (fetchData)**: Initiates an AJAX GET request using XMLHttpRequest() to fetch data from data.json. Upon successful response (xhr.onload), it parses the JSON data (xhr.responseText) and updates the <div> content dynamically.

### 2. Explain about jQuery events and selectors

jQuery simplifies JavaScript programming by providing a set of methods for DOM manipulation, event handling, animations, and AJAX interactions.

#### Selectors:

jQuery selectors allow you to find and manipulate HTML elements using CSS-like syntax ($(selector)).

Example:

javascript

$(document).ready(function() {

// Selects all <p> elements and changes their text color

$("p").css("color", "blue");

});

#### Events:

jQuery provides methods to handle various events such as click, hover, submit, etc. ($(selector).on(event, handler)).

Example:

javascript

$(document).ready(function() {

// Click event handler for button with id 'btnSubmit'

$('#btnSubmit').click(function() {

alert('Button clicked!');

});

});

jQuery simplifies the process of attaching event handlers and manipulating DOM elements compared to plain JavaScript, making it popular for web development.

### 3. PHP program to display a digital clock

php

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Digital Clock</title>

<script>

function updateClock() {

var now = new Date();

var hours = now.getHours();

var minutes = now.getMinutes();

var seconds = now.getSeconds();

document.getElementById('clock').textContent = hours + ':' + minutes + ':' + seconds;

}

setInterval(updateClock, 1000); // Update every second

</script>

</head>

<body>

<h1>Digital Clock</h1>

<div id="clock"></div>

</body>

</html>

#### Explanation:

* **HTML Structure**: Contains a <div> (<div id="clock">) where the current time is dynamically displayed.
* **JavaScript Function (updateClock)**: Gets the current time using Date() object and updates the <div> content (document.getElementById('clock').textContent) every second (setInterval(updateClock, 1000)).

### 4. PHP program to keep track of visitors

php

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Visitor Counter</title>

</head>

<body>

<h1>Visitor Counter</h1>

<?php

$visits = 0;

if (isset($\_COOKIE['visits'])) {

$visits = $\_COOKIE['visits'];

}

$visits++;

setcookie('visits', $visits, time() + 3600 \* 24 \* 30); // Cookie set for 30 days

echo "<p>Total visits: $visits</p>";

?>

</body>

</html>

#### Explanation:

* **PHP Code**:
  + Initializes $visits to 0 and increments it each time the page is visited.
  + Uses cookies ($\_COOKIE['visits']) to store and retrieve the visit count.
  + Updates the cookie with setcookie() to persist the visit count for 30 days (time() + 3600 \* 24 \* 30).

### 5. PHP programs

#### a. Simple calculator operations

php

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Simple Calculator</title>

</head>

<body>

<h1>Simple Calculator</h1>

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF']; ?>">

<input type="number" name="num1" required>

<select name="operator" required>

<option value="add">+</option>

<option value="subtract">-</option>

<option value="multiply">\*</option>

<option value="divide">/</option>

</select>

<input type="number" name="num2" required>

<button type="submit" name="calculate">Calculate</button>

</form>

<?php

if (isset($\_POST['calculate'])) {

$num1 = $\_POST['num1'];

$num2 = $\_POST['num2'];

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

switch ($operator) {

case 'add':

$result = $num1 + $num2;

break;

case 'subtract':

$result = $num1 - $num2;

break;

case 'multiply':

$result = $num1 \* $num2;

break;

case 'divide':

if ($num2 != 0) {

$result = $num1 / $num2;

} else {

echo '<p style="color: red;">Error: Division by zero</p>';

}

break;

default:

echo '<p style="color: red;">Invalid operator</p>';

break;

}

echo "<p>Result: $result</p>";

}

?>

</body>

</html>

#### Explanation:

* **HTML Form**: Accepts two numbers (<input type="number" name="num1", <input type="number" name="num2") and an operator (<select name="operator">).
* **PHP Code**:
  + Retrieves form inputs ($\_POST['num1'], $\_POST['num2'], $\_POST['operator']).
  + Performs calculations based on the selected operator (switch statement) and displays the result (echo "<p>Result: $result</p>").

#### b. Transpose of a matrix

php

<?php

function transposeMatrix($matrix) {

$rows = count($matrix);

$cols = count($matrix[0]);

$transposed = [];

for ($i = 0; $i < $cols; $i++) {

for ($j = 0; $j < $rows; $j++) {

$transposed[$i][$j] = $matrix[$j][$i];

}

}

return $transposed;

}

$matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

$transposedMatrix = transposeMatrix($matrix);

echo '<pre>';

print\_r($transposedMatrix);

echo '</pre>';

?>

#### Explanation:

* **PHP Function (transposeMatrix)**:
  + Accepts a matrix as input and calculates its transpose.
  + Iterates through the rows and columns of the matrix to swap elements ($transposed[$i][$j] = $matrix[$j][$i]).
  + Returns the transposed matrix.

#### c. Multiplication of two matrices

php

<?php

function multiplyMatrices($matrix1, $matrix2) {

$rows1 = count($matrix1);

$cols1 = count($matrix1[0]);

$cols2 = count($matrix2[0]);

$result = array\_fill(0, $rows1, array\_fill(0, $cols2, 0));

for ($i = 0; $i < $rows1; $i++) {

for ($j = 0; $j < $cols2; $j++) {

for ($k = 0; $k < $cols1; $k++) {

$result[$i][$j] += $matrix1[$i][$k] \* $matrix2[$k][$j];

}

}

}

return $result;

}

$matrix1 = [

[1, 2],

[3, 4],

[5, 6]

];

$matrix2 = [

[7, 8, 9],

[10, 11, 12]

];

$resultMatrix = multiplyMatrices($matrix1, $matrix2);

echo '<pre>';

print\_r($resultMatrix);

echo '</pre>';

?>

#### Explanation:

* **PHP Function (multiplyMatrices)**:
  + Accepts two matrices as input and calculates their product.
  + Checks dimensions to ensure compatibility ($cols1 == $rows2).
  + Uses nested loops to perform matrix multiplication ($result[$i][$j] += $matrix1[$i][$k] \* $matrix2[$k][$j]).
  + Returns the resulting matrix.

### 6. PHP Program to implement file upload and download operations

#### File Upload (upload.php)

php

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>File Upload</title>

</head>

<body>

<h1>Upload File</h1>

<form action="upload.php" method="post" enctype="multipart/form-data">

<input type="file" name="fileToUpload" id="fileToUpload">

<input type="submit" value="Upload File" name="submit">

</form>

<?php

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

$targetDir = "uploads/";

$targetFile = $targetDir . basename($\_FILES["fileToUpload"]["name"]);

$uploadOk = 1;

$imageFileType = strtolower(pathinfo($targetFile, PATHINFO\_EXTENSION));

// Check if file already exists

if (file\_exists($targetFile)) {

echo "Sorry, file already exists.";

$uploadOk = 0;

}

// Check file size

if ($\_FILES["fileToUpload"]["size"] > 500000) {

echo "Sorry, your file is too large.";

$uploadOk = 0;

}

// Allow certain file formats

if ($imageFileType != "txt" && $imageFileType != "pdf" && $imageFileType != "doc") {

echo "Sorry, only TXT, PDF, and DOC files are allowed.";

$uploadOk = 0;

}

// Check if $uploadOk is set to 0 by an error

if ($uploadOk == 0) {

echo "Sorry, your file was not uploaded.";

} else {

if (move\_uploaded\_file($\_FILES["fileToUpload"]["tmp\_name"], $targetFile)) {

echo "The file ". htmlspecialchars(basename( $\_FILES["fileToUpload"]["name"])). " has been uploaded.";

} else {

echo "Sorry, there was an error uploading your file.";

}

}

}

?>

</body>

</html>

#### File Download (download.php)

php

<?php

$filePath = 'path/to/your/file.ext'; // Replace with your file path

$fileName = basename($filePath);

if (file\_exists($filePath)) {

header('Content-Description: File Transfer');

header('Content-Type: application/octet-stream');

header('Content-Disposition: attachment; filename="' . $fileName . '"');

header('Expires: 0');

header('Cache-Control: must-revalidate');

header('Pragma: public');

header('Content-Length: ' . filesize($filePath));

readfile($filePath);

exit;

} else {

echo "File not found.";

}

?>

#### Explanation:

* **File Upload (upload.php)**:
  + Provides a form (<form action="upload.php" method="post" enctype="multipart/form-data">) to upload a file.
  + Checks file size, format, and existence before uploading (move\_uploaded\_file()).
  + Displays appropriate messages based on upload success or failure.
* **File Download (download.php)**:
  + Uses PHP headers (header()) to force download of a specified file ($filePath).
  + Validates file existence (file\_exists()) before initiating the download.

### 7. PHP program to perform operations in cookies

#### Adding a Cookie

php

<?php

$cookieName = "user";

$cookieValue = "John Doe";

$expiration = time() + 3600 \* 24 \* 30; // 30 days

setcookie($cookieName, $cookieValue, $expiration);

echo "Cookie '$cookieName' is set.";

?>

#### Deleting a Cookie

php

<?php

$cookieName = "user";

unset($\_COOKIE[$cookieName]);

setcookie($cookieName, "", time() - 3600); // Set expiration in the past to delete cookie

echo "Cookie '$cookieName' is deleted.";

?>

#### Explanation:

* **Adding a Cookie**:
  + Uses setcookie() to create a cookie ($cookieName) with a value ($cookieValue) and expiration time ($expiration).
* **Deleting a Cookie**:
  + Unsets the cookie (unset($\_COOKIE[$cookieName])) and sets its expiration time in the past (setcookie($cookieName, "", time() - 3600)) to delete it.

### 8. PHP program to implement PHP form handling

php

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>PHP Form Handling</title>

</head>

<body>

<h1>PHP Form Handling</h1>

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF']; ?>">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required><br><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required><br><br>

<label for="message">Message:</label><br>

<textarea id="message" name="message" rows="4" cols="50" required></textarea><br><br>

<button type="submit" name="submit">Submit</button>

</form>

<?php

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

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

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

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

echo "<h2>Form Data:</h2>";

echo "<p>Name: $name</p>";

echo "<p>Email: $email</p>";

echo "<p>Message: $message</p>";

}

?>

</body>

</html>

#### Explanation:

* **HTML Form**:
  + Collects user input (<input>, <textarea>) for name, email, and message.
  + Submits data to the same page (action="<?php echo $\_SERVER['PHP\_SELF']; ?>") using POST method.
* **PHP Code**:
  + Checks if the form is submitted (if ($\_SERVER["REQUEST\_METHOD"] == "POST")).
  + Retrieves form data ($\_POST['name'], $\_POST['email'], $\_POST['message']).
  + Displays submitted data (echo "<p>Name: $name</p>", etc.).

**CASE STUDIES**

1. **Step-by-Step Guide: Creating an Online Bookstore with PHP Backend**

**Step 1: Objective**

Create a simple online bookstore where users can browse books, view book details, and add books to their shopping cart. The application should have a responsive design and include features such as book categories, search functionality, and a shopping cart summary.

**Step 2: Introduction**

Building an online bookstore involves creating a user-friendly platform where users can explore various books, read detailed descriptions, and purchase their selected books. This project emphasizes creating a seamless shopping experience through a responsive design.

**Step 3: Background**

An online bookstore requires a well-structured database to store book information and a user interface that provides easy navigation. The combination of these elements creates a shopping experience that is both efficient and enjoyable for the user.

**Step 4: Methodology**

**4.1 Data Collection**

* **Compile Book Data:**
  + Gather a list of books with details such as titles, authors, categories, descriptions, and prices.
  + Store this data in a CSV file or directly in a database.

**4.2 Design**

* **Create Wireframes:**
  + **Homepage:** Display categories and featured books.
  + **Book Details Page:** Show detailed information about the selected book.
  + **Shopping Cart:** Display the list of books added to the cart along with a summary.
* **Ensure Responsive Design:**
  + Make sure the wireframes are adaptable to both desktop and mobile devices.

**4.3 Development**

* **Front-End Development:**
  + **HTML:** Structure the content of the web pages.
  + **CSS:** Style the web pages for a consistent look and feel.
  + **JavaScript:** Add interactivity to the web pages.

html

<!-- Example: Book List Display -->

<div class="book-list">

<div class="book-item">

<img src="book-cover.jpg" alt="Book Cover">

<h3>Book Title</h3>

<p>Author: Author Name</p>

<p>Category: Fiction</p>

<p>Price: $20</p>

<button onclick="addToCart('Book Title')">Add to Cart</button>

</div>

<!-- Repeat for other books -->

</div>

css

/\* Example: Basic Styling for Book Items \*/

.book-list {

display: flex;

flex-wrap: wrap;

}

.book-item {

border: 1px solid #ddd;

margin: 10px;

padding: 10px;

width: 200px;

text-align: center;

}

.book-item img {

max-width: 100%;

height: auto;

}

.book-item h3 {

font-size: 1.2em;

}

javascript

// Example: Adding a Book to the Cart

function addToCart(bookTitle) {

alert(bookTitle + ' added to cart!');

}

* **Back-End Development:**
  + **PHP:** Handle server-side logic, including routing, database interactions, and user sessions.

php

<?php

// Example: Basic Route in PHP

$servername = "localhost";

$username = "username";

$password = "password";

$dbname = "online\_bookstore";

// Create connection

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

// Check connection

if ($conn->connect\_error) {

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

}

$sql = "SELECT title, author, category, price, cover FROM books";

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

if ($result->num\_rows > 0) {

while($row = $result->fetch\_assoc()) {

echo "<div class='book-item'>";

echo "<img src='" . $row["cover"] . "' alt='Book Cover'>";

echo "<h3>" . $row["title"] . "</h3>";

echo "<p>Author: " . $row["author"] . "</p>";

echo "<p>Category: " . $row["category"] . "</p>";

echo "<p>Price: $" . $row["price"] . "</p>";

echo "<button onclick=\"addToCart('" . $row["title"] . "')\">Add to Cart</button>";

echo "</div>";

}

} else {

echo "0 results";

}

$conn->close();

?>

**4.4 Testing**

* **Usability Testing:**
  + Test the application to ensure users can navigate and use it seamlessly.
  + Perform tests on different devices and browsers to ensure compatibility and responsiveness.

**4.5 Deployment**

* **Deploy the Application:**
  + Use a web hosting service to deploy the application. Make sure to upload all necessary files (HTML, CSS, JavaScript, PHP, and database) to the server.

**Step 5: Results**

The online bookstore allows users to browse books, view details, and manage their shopping carts. The responsive design ensures that the application works well on both desktop and mobile devices.

**Step 6: Discussion**

The implemented online bookstore provides a user-friendly platform for browsing and purchasing books. The responsive design improves accessibility, and the categorization and search functionalities enhance the user experience.

**Step 7: Conclusion**

The project successfully developed an online bookstore with essential features such as book browsing, detailed views, and a shopping cart. The responsive design ensures usability across different devices.

**Step 8: Future Work**

* **User Reviews and Ratings:**
  + Add functionality for users to leave reviews and rate books.
* **Recommendation System:**
  + Implement a system to recommend books based on user preferences.
* **Payment Integration:**
  + Integrate with payment gateways for a seamless checkout process.

1. **Step-by-Step Guide: Creating a User Registration and Authentication System**

**Step 1: Objective**

Create a secure user registration and authentication system using PHP, MySQL, and sessions. Implement password hashing and salting for security.

**Step 2: Introduction**

A robust user authentication system is crucial for protecting user data and maintaining the security of web applications. This project focuses on creating a secure and efficient authentication system.

**Step 3: Background**

User authentication systems are essential for any application requiring user data protection. By implementing password hashing and session management, the security of user accounts can be significantly enhanced.

**Step 4: Methodology**

**4.1 User Registration**

* **Create a Registration Form:**
  + Develop an HTML form to collect user information such as username and password.
  + Validate user input to ensure the data is in the correct format.

html

<!-- Example: User Registration Form -->

<form action="register.php" method="POST">

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

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

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

</form>

* **Store User Data Securely:**
  + Use PHP to handle form submission and store user data in a MySQL database.
  + Hash passwords using PHP’s password\_hash() function before storing them.

php

<?php

// register.php

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

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

$password = password\_hash($\_POST['password'], PASSWORD\_BCRYPT);

// Database connection

$servername = "localhost";

$dbusername = "root";

$dbpassword = "";

$dbname = "user\_auth";

$conn = new mysqli($servername, $dbusername, $dbpassword, $dbname);

// Check connection

if ($conn->connect\_error) {

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

}

// Insert user data

$sql = "INSERT INTO users (username, password) VALUES ('$username', '$password')";

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

echo "New record created successfully";

} else {

echo "Error: " . $sql . "<br>" . $conn->error;

}

$conn->close();

}

?>

**4.2 Login System**

* **Create a Login Form:**
  + Develop an HTML form to collect the username and password for login.

html

<!-- Example: User Login Form -->

<form action="login.php" method="POST">

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

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

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

</form>

* **Session Management:**
  + Use PHP sessions to track logged-in users.
  + Verify the password using PHP’s password\_verify() function.

php

<?php

// login.php

session\_start();

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

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

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

// Database connection

$servername = "localhost";

$dbusername = "root";

$dbpassword = "";

$dbname = "user\_auth";

$conn = new mysqli($servername, $dbusername, $dbpassword, $dbname);

// Check connection

if ($conn->connect\_error) {

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

}

// Retrieve user data

$sql = "SELECT username, password FROM users WHERE username='$username'";

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

if ($result->num\_rows > 0) {

// Check password

$row = $result->fetch\_assoc();

if (password\_verify($password, $row['password'])) {

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

header("Location: secure\_page.php"); // Redirect to a secure page

} else {

echo "Invalid password";

}

} else {

echo "No user found";

}

$conn->close();

}

?>

**4.3 Password Security**

* **Implement Password Hashing:**
  + Use password\_hash() to hash passwords when storing them.
  + Use password\_verify() to check the password during login.

**4.4 Password Recovery**

* **Password Recovery Form:**
  + Develop an HTML form to collect the email address for password recovery.

html

<!-- Example: Password Recovery Form -->

<form action="recover\_password.php" method="POST">

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<button type="submit">Recover Password</button>

</form>

* **Email Verification:**
  + Implement a mechanism to send a password reset link to the user's email.
  + Use PHP’s mail() function to send the email.

php

<?php

// recover\_password.php

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

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

// Database connection

$servername = "localhost";

$dbusername = "root";

$dbpassword = "";

$dbname = "user\_auth";

$conn = new mysqli($servername, $dbusername, $dbpassword, $dbname);

// Check connection

if ($conn->connect\_error) {

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

}

// Check if email exists

$sql = "SELECT email FROM users WHERE email='$email'";

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

if ($result->num\_rows > 0) {

// Generate a unique token

$token = bin2hex(random\_bytes(50));

// Store token in the database with an expiration time

$sql = "UPDATE users SET reset\_token='$token', reset\_expiration=DATE\_ADD(NOW(), INTERVAL 1 HOUR) WHERE email='$email'";

$conn->query($sql);

// Send password reset email

$reset\_link = "http://yourwebsite.com/reset\_password.php?token=$token";

$subject = "Password Recovery";

$message = "Click the following link to reset your password: $reset\_link";

$headers = "From: no-reply@yourwebsite.com";

if (mail($email, $subject, $message, $headers)) {

echo "Password recovery email sent";

} else {

echo "Failed to send email";

}

} else {

echo "No user found with that email";

}

$conn->close();

}

?>

* **Reset Password Form:**
  + Develop an HTML form to reset the password using the token.

html

<!-- Example: Reset Password Form -->

<form action="reset\_password.php" method="POST">

<input type="hidden" name="token" value="<?php echo $\_GET['token']; ?>">

<label for="password">New Password:</label>

<input type="password" id="password" name="password" required>

<button type="submit">Reset Password</button>

</form>

* **Update Password in Database:**
  + Validate the token and update the password.

php

<?php

// reset\_password.php

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

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

$password = password\_hash($\_POST['password'], PASSWORD\_BCRYPT);

// Database connection

$servername = "localhost";

$dbusername = "root";

$dbpassword = "";

$dbname = "user\_auth";

$conn = new mysqli($servername, $dbusername, $dbpassword, $dbname);

// Check connection

if ($conn->connect\_error) {

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

}

// Validate token and update password

$sql = "SELECT email FROM users WHERE reset\_token='$token' AND reset\_expiration > NOW()";

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

if ($result->num\_rows > 0) {

$sql = "UPDATE users SET password='$password', reset\_token=NULL, reset\_expiration=NULL WHERE reset\_token='$token'";

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

echo "Password reset successfully";

} else {

echo "Error: " . $conn->error;

}

} else {

echo "Invalid or expired token";

}

$conn->close();

}

?>

**Step 5: Results**

The system allows users to register, log in, and recover passwords securely. Passwords are hashed, ensuring that even if the database is compromised, user passwords remain protected.

**Step 6: Discussion**

The user authentication system effectively secures user data and ensures that only authenticated users can access certain parts of the application. The use of password hashing adds an additional layer of security.

**Step 7: Conclusion**

The project successfully developed a secure user registration and authentication system. The use of password hashing and session management ensures that user data is well-protected.

**Step 8: Future Work**

* **Implementing Two-Factor Authentication:**
  + Enhance security by adding an extra layer of authentication.
* **User Role Management:**
  + Develop a system to manage different user roles (e.g., admin, user).
* **Improving the User Interface:**
  + Enhance the user interface for registration and login forms to improve user experience.

1. **Step-by-Step Guide: Developing an Online Shopping Cart Application**

**Step 1: Objective**

Develop an online shopping cart application with PHP and MySQL. Ensure secure handling of user data and transactions.

**Step 2: Introduction**

An online shopping cart is a core component of e-commerce platforms, allowing users to manage their selected products before checkout. This project emphasizes secure data handling and transaction processing.

**Step 3: Background**

The shopping cart functionality is crucial for e-commerce websites as it allows users to select products, view their selections, and proceed to checkout. Proper session management and data handling are essential for a seamless user experience.

**Step 4: Methodology**

**4.1 Product Catalog**

* **Create a Product Database:**
  + Develop a MySQL database to store product information such as names, prices, categories, and descriptions.

sql

-- Example: Create Products Table

CREATE TABLE products (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

price DECIMAL(10, 2) NOT NULL,

category VARCHAR(255),

description TEXT,

image VARCHAR(255)

);

* **Populate the Product Database:**
  + Insert sample data into the products table for testing purposes.

sql

-- Example: Insert Sample Products

INSERT INTO products (name, price, category, description, image)

VALUES

('Product 1', 19.99, 'Category 1', 'Description of Product 1', 'image1.jpg'),

('Product 2', 29.99, 'Category 2', 'Description of Product 2', 'image2.jpg');

**4.2 User Authentication**

* **Create User Authentication System:**
  + Develop a registration and login system using PHP and MySQL to manage user sessions.

php

// register.php

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

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

$password = password\_hash($\_POST['password'], PASSWORD\_BCRYPT);

// Database connection

$conn = new mysqli('localhost', 'root', '', 'shopping\_cart');

// Insert user data

$sql = "INSERT INTO users (username, password) VALUES ('$username', '$password')";

$conn->query($sql);

$conn->close();

}

php

// login.php

session\_start();

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

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

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

// Database connection

$conn = new mysqli('localhost', 'root', '', 'shopping\_cart');

// Retrieve user data

$sql = "SELECT password FROM users WHERE username='$username'";

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

$row = $result->fetch\_assoc();

if (password\_verify($password, $row['password'])) {

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

header("Location: index.php");

} else {

echo "Invalid credentials";

}

$conn->close();

}

**4.3 Shopping Cart Management**

* **Add to Cart:**
  + Implement functionality to add products to the shopping cart using PHP sessions.

php

// add\_to\_cart.php

session\_start();

$product\_id = $\_POST['product\_id'];

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

if (!isset($\_SESSION['cart'])) {

$\_SESSION['cart'] = array();

}

$\_SESSION['cart'][$product\_id] = $quantity;

header("Location: cart.php");

* **Display Cart:**
  + Develop a script to display the contents of the shopping cart by retrieving product details from the database.

php

// cart.php

session\_start();

if (isset($\_SESSION['cart'])) {

$conn = new mysqli('localhost', 'root', '', 'shopping\_cart');

foreach ($\_SESSION['cart'] as $product\_id => $quantity) {

$sql = "SELECT name, price FROM products WHERE id='$product\_id'";

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

$row = $result->fetch\_assoc();

echo "Product: " . $row['name'] . ", Quantity: " . $quantity . ", Price: " . $row['price'] \* $quantity . "<br>";

}

$conn->close();

} else {

echo "Your cart is empty";

}

**4.4 Checkout Process**

* **Integrate Payment Gateway:**
  + Integrate a payment gateway like PayPal or Stripe for secure payment processing. (Note: Detailed integration steps can vary based on the chosen gateway and are typically provided in their documentation.)

php

// checkout.php

session\_start();

// Ensure the user is logged in

if (!isset($\_SESSION['username'])) {

header("Location: login.php");

exit();

}

// Process the payment (simplified example)

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

$total\_amount = $\_POST['total\_amount'];

// Redirect to PayPal or Stripe for payment processing

// Example: Redirect to PayPal

header("Location: https://www.paypal.com/cgi-bin/webscr?cmd=\_xclick&business=merchant@example.com&amount=$total\_amount");

exit();

}

// Calculate total amount from cart

$total\_amount = 0;

if (isset($\_SESSION['cart'])) {

$conn = new mysqli('localhost', 'root', '', 'shopping\_cart');

foreach ($\_SESSION['cart'] as $product\_id => $quantity) {

$sql = "SELECT price FROM products WHERE id='$product\_id'";

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

$row = $result->fetch\_assoc();

$total\_amount += $row['price'] \* $quantity;

}

$conn->close();

}

echo "Total Amount: $" . $total\_amount;

**Step 5: Results**

The application allows users to browse products, add them to their cart, and proceed to checkout securely. User sessions are managed to keep track of cart contents.

**Step 6: Discussion**

The online shopping cart application provides a seamless shopping experience with secure data handling. The integration of a secure payment gateway ensures safe transactions.

**Step 7: Conclusion**

The project successfully developed an online shopping cart application with essential e-commerce functionalities. Secure data handling and transaction processing are ensured.

**Step 8: Future Work**

* **Adding Product Recommendations:**
  + Implement a recommendation system to suggest products based on user behavior.
* **Implementing Discount Codes and Promotions:**
  + Develop functionality to allow users to apply discount codes during checkout.
* **Enhancing the User Interface:**
  + Improve the user interface to provide a better shopping experience.

1. **Step-by-Step Guide: Developing a Task Management System**

**Step 1: Objective**

Build a web-based task management system where users can create, update, and delete tasks. Focus on error handling and database interactions.

**Step 2: Introduction**

A task management system helps users organize and prioritize their tasks. This project involves building a CRUD application with robust error handling to ensure smooth and reliable operations.

**Step 3: Background**

Task management systems are essential for productivity, allowing users to keep track of their tasks and deadlines. This project requires implementing basic CRUD operations (Create, Read, Update, Delete) and ensuring the application is user-friendly and secure.

**Step 4: Methodology**

**4.1 Task CRUD Operations**

* **Create Database and Tasks Table:**
  + Develop a MySQL database to store task information such as task name, description, and user ID.

sql

-- Example: Create Tasks Table

CREATE TABLE tasks (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

name VARCHAR(255) NOT NULL,

description TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

**4.2 Error Handling**

* **Utilize Try-Catch Blocks:**
  + Implement try-catch blocks for database operations to handle errors gracefully and ensure the application does not crash unexpectedly.

php

// db.php (Database Connection and Error Handling)

try {

$db = new mysqli('localhost', 'root', '', 'task\_management');

} catch (Exception $e) {

echo "Connection failed: " . $e->getMessage();

exit();

}

**4.3 User Authentication**

* **Create User Authentication System:**
  + Develop a registration and login system using PHP and MySQL to manage user sessions securely.

php

// register.php

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

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

$password = password\_hash($\_POST['password'], PASSWORD\_BCRYPT);

try {

$conn = new mysqli('localhost', 'root', '', 'task\_management');

$sql = "INSERT INTO users (username, password) VALUES ('$username', '$password')";

$conn->query($sql);

} catch (Exception $e) {

echo "Error: " . $e->getMessage();

}

$conn->close();

}

php

// login.php

session\_start();

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

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

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

try {

$conn = new mysqli('localhost', 'root', '', 'task\_management');

$sql = "SELECT password FROM users WHERE username='$username'";

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

$row = $result->fetch\_assoc();

if (password\_verify($password, $row['password'])) {

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

header("Location: index.php");

} else {

echo "Invalid credentials";

}

} catch (Exception $e) {

echo "Error: " . $e->getMessage();

}

$conn->close();

}

**4.4 Responsive Design**

* **Ensure Usability Across Devices:**
  + Develop the front-end using HTML, CSS, and JavaScript to ensure the application is responsive and works well on different devices.

html

<!-- Example: Task Management System Front-End -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Task Management System</title>

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

</head>

<body>

<div class="container">

<h1>Task Management System</h1>

<form action="create\_task.php" method="post">

<input type="text" name="task\_name" placeholder="Task Name" required>

<textarea name="task\_description" placeholder="Task Description"></textarea>

<button type="submit">Create Task</button>

</form>

<div class="task-list">

<?php

// Fetch and display tasks

session\_start();

$user\_id = $\_SESSION['user\_id'];

try {

$conn = new mysqli('localhost', 'root', '', 'task\_management');

$sql = "SELECT \* FROM tasks WHERE user\_id='$user\_id'";

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

while ($task = $result->fetch\_assoc()) {

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

echo "<h3>" . $task['name'] . "</h3>";

echo "<p>" . $task['description'] . "</p>";

echo "</div>";

}

} catch (Exception $e) {

echo "Error: " . $e->getMessage();

}

$conn->close();

?>

</div>

</div>

</body>

</html>

**Step 5: Coding**

**5.1 Create Task**

* **Insert Task into Database:**

php

// create\_task.php

session\_start();

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

$task\_name = $\_POST['task\_name'];

$task\_description = $\_POST['task\_description'];

$user\_id = $\_SESSION['user\_id'];

try {

$conn = new mysqli('localhost', 'root', '', 'task\_management');

$sql = "INSERT INTO tasks (user\_id, name, description) VALUES ('$user\_id', '$task\_name', '$task\_description')";

$conn->query($sql);

} catch (Exception $e) {

echo "Error: " . $e->getMessage();

}

$conn->close();

header("Location: index.php");

}

**5.2 Display Tasks**

* **Retrieve and Display Tasks from Database:**

php

// index.php

session\_start();

$user\_id = $\_SESSION['user\_id'];

try {

$conn = new mysqli('localhost', 'root', '', 'task\_management');

$sql = "SELECT \* FROM tasks WHERE user\_id='$user\_id'";

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

while ($task = $result->fetch\_assoc()) {

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

echo "<h3>" . $task['name'] . "</h3>";

echo "<p>" . $task['description'] . "</p>";

echo "</div>";

}

} catch (Exception $e) {

echo "Error: " . $e->getMessage();

}

$conn->close();

**Step 6: Results**

The task management system allows users to create, view, update, and delete tasks efficiently. Error handling ensures the application remains robust and user-friendly.

**Step 7: Discussion**

The task management system enhances productivity by providing users with a tool to manage their tasks effectively. Error handling ensures that the application can recover from unexpected issues without crashing.

**Step 8: Conclusion**

The project successfully developed a task management system with essential CRUD operations and robust error handling. The responsive design ensures usability across different devices.

**Step 9: Future Work**

* **Implementing Task Prioritization and Categorization:**
  + Develop functionality to prioritize and categorize tasks for better organization.
* **Adding Notifications and Reminders:**
  + Implement a notification system to remind users of upcoming deadlines.
* **Enhancing the User Interface:**
  + Improve the user interface to provide a better user experience.

1. **Step-by-Step Guide: Developing a Secure File Upload and Management System**

**Step 1: Objective**

Develop a secure file upload and management system using PHP. Implement measures to prevent file upload vulnerabilities.

**Step 2: Introduction**

File upload systems are common in web applications, but they pose significant security risks. This project aims to create a secure system for handling file uploads, ensuring that user-uploaded files are processed safely and stored securely.

**Step 3: Background**

File upload vulnerabilities can lead to serious security issues such as unauthorized access and malware uploads. Proper validation, secure handling, and storage of uploaded files are crucial to prevent such risks and maintain the integrity of the web application.

**Step 4: Methodology**

**4.1 File Upload Form**

* **Create a File Upload Form:**
  + Develop an HTML form for users to upload files. The form should include validation checks for file type and size.

html

Copy code

<!-- Example: File Upload Form -->

<form action="upload.php" method="post" enctype="multipart/form-data">

<label for="file">Choose file to upload:</label>

<input type="file" name="file" id="file" required>

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

</form>

**4.2 Secure File Handling**

* **Validate and Securely Handle Files:**
  + Use PHP functions like move\_uploaded\_file() to securely handle files. Validate the file type and size to prevent malicious files from being uploaded.

php

Copy code

// upload.php

if ($\_SERVER['REQUEST\_METHOD'] == 'POST' && isset($\_FILES['file'])) {

$file = $\_FILES['file'];

$allowed\_types = ['image/jpeg', 'image/png', 'application/pdf'];

$max\_size = 5000000; // 5MB

if (in\_array($file['type'], $allowed\_types) && $file['size'] <= $max\_size) {

$upload\_dir = 'uploads/';

$file\_path = $upload\_dir . basename($file['name']);

if (move\_uploaded\_file($file['tmp\_name'], $file\_path)) {

echo "File uploaded successfully.";

} else {

echo "Failed to upload file.";

}

} else {

echo "Invalid file type or size.";

}

}

**4.3 File Management**

* **Display and Manage Uploaded Files:**
  + Display a list of uploaded files with download links. Provide options to manage these files (e.g., delete).

php

Copy code

// display\_files.php

$files = scandir('uploads');

echo "<h2>Uploaded Files:</h2>";

foreach ($files as $file) {

if ($file != '.' && $file != '..') {

echo "<a href='uploads/$file'>$file</a><br>";

}

}

**4.4 Admin Panel**

* **Develop an Admin Panel:**
  + Create an admin panel to manage uploaded files. This includes functionalities to delete or update files.

php

Copy code

// admin.php

session\_start();

if (!isset($\_SESSION['admin\_logged\_in'])) {

header("Location: login.php");

exit();

}

$files = scandir('uploads');

echo "<h2>Admin Panel - Manage Files:</h2>";

foreach ($files as $file) {

if ($file != '.' && $file != '..') {

echo "<a href='uploads/$file'>$file</a> ";

echo "<a href='delete\_file.php?file=$file'>Delete</a><br>";

}

}

**Step 5: Coding**

**5.1 File Upload**

* **Securely Handle File Uploads:**

php

Copy code

// upload.php

if ($\_SERVER['REQUEST\_METHOD'] == 'POST' && isset($\_FILES['file'])) {

$file = $\_FILES['file'];

$allowed\_types = ['image/jpeg', 'image/png', 'application/pdf'];

$max\_size = 5000000; // 5MB

if (in\_array($file['type'], $allowed\_types) && $file['size'] <= $max\_size) {

$upload\_dir = 'uploads/';

$file\_path = $upload\_dir . basename($file['name']);

if (move\_uploaded\_file($file['tmp\_name'], $file\_path)) {

echo "File uploaded successfully.";

} else {

echo "Failed to upload file.";

}

} else {

echo "Invalid file type or size.";

}

}

**5.2 Display Uploaded Files**

* **List and Provide Download Links for Uploaded Files:**

php

Copy code

// display\_files.php

$files = scandir('uploads');

echo "<h2>Uploaded Files:</h2>";

foreach ($files as $file) {

if ($file != '.' && $file != '..') {

echo "<a href='uploads/$file'>$file</a><br>";

}

}

**5.3 Delete File (Admin Functionality)**

* **Admin Panel Functionality to Delete Files:**

php

Copy code

// delete\_file.php

session\_start();

if (!isset($\_SESSION['admin\_logged\_in'])) {

header("Location: login.php");

exit();

}

if (isset($\_GET['file'])) {

$file = $\_GET['file'];

$file\_path = 'uploads/' . $file;

if (file\_exists($file\_path)) {

unlink($file\_path);

echo "File deleted successfully.";

} else {

echo "File not found.";

}

}

**Step 6: Results**

The system allows users to upload and manage files securely. Validation checks ensure that only allowed file types and sizes are uploaded, significantly reducing the risk of file upload vulnerabilities.

**Step 7: Discussion**

The secure file upload system effectively handles user-uploaded files while preventing common vulnerabilities. The admin panel allows for easy management of uploaded files, providing an additional layer of control over the uploaded content.

**Step 8: Conclusion**

The project successfully developed a secure file upload and management system with validation for file type and size. Secure handling ensures the integrity of the application, making it safe for users to upload and manage their files.

**Step 9: Future Work**

* **Implementing File Version Control:**
  + Develop functionality to maintain different versions of uploaded files, allowing users to track changes and revert to previous versions if necessary.
* **Adding User Access Controls for Uploaded Files:**
  + Implement user-specific access controls to ensure that only authorized users can view or manage certain files.
* **Enhancing the User Interface for the Admin Panel:**
  + Improve the user interface of the admin panel to provide a more intuitive and user-friendly experience.

Case Study 6: Employee Management System

Case Study 7: Job Board Platform

Case Study 8: Appointment Booking System

Case Study 9: Music Playlist Manager

Case Study 10: Weather Forecast Application

**Project List**

**Project 1: Online Learning Management System (LMS)**

**Project Description:** Develop an Online Learning Management System (LMS) for managing courses, students, and instructors, leveraging PHP, MySQL, HTML, CSS, JavaScript, and possibly a PHP framework like Laravel for efficient development.

**Technologies Used:**

* **HTML:** Structure web pages.
* **CSS:** Style and layout.
* **JavaScript:** Client-side interactions and validation.
* **PHP:** Server-side scripting and business logic.
* **MySQL:** Database management and storage of course, student, and instructor data.

**Features:**

* **Course Management:**
  + Add new courses with details (title, description, instructor).
  + List all courses with search and filter options.
  + Delete existing courses.
* **Student Management:**
  + Register new students with basic details (name, email, enrolled courses).
  + View and manage student records.
  + Enroll students in courses.
* **Instructor Management:**
  + Add new instructors with details (name, contact information).
  + View and manage instructor records.
  + Assign instructors to courses.
* **User Authentication and Authorization:**
  + Secure login system for administrators.
  + Role-based access control (admin, instructor, student).
* **Responsive Design:**
  + Ensure usability across devices with responsive design principles.

**Database Schema:**

* **Course Table:**
  + CourseID (Primary Key)
  + CourseTitle
  + Description
  + InstructorID (Foreign Key)
* **Student Table:**
  + StudentID (Primary Key)
  + StudentName
  + Email
  + EnrolledCourses (Serialized or linked to CourseID)
* **Instructor Table:**
  + InstructorID (Primary Key)
  + InstructorName
  + ContactInfo

**Pages and Functionalities:**

* **Login Page:**
  + Admin login form with username and password.
* **Dashboard:**
  + Overview of courses, students, and instructors.
  + Quick links to course, student, and instructor management.
* **Course Management Page:**
  + Add Course Form
  + List of Courses
  + Delete Course functionality
* **Student Management Page:**
  + Add Student Form
  + List of Students
  + Delete Student functionality
* **Instructor Management Page:**
  + Add Instructor Form
  + List of Instructors
  + Delete Instructor functionality
* **Logout Functionality:**
  + Secure logout mechanism for administrators.

**Project 2: Online Marketplace for Freelancers**

**Project Description:** Create an online marketplace platform where freelancers can showcase their services, clients can browse and hire freelancers, and administrators can manage the platform. This project will utilize PHP, MySQL, HTML, CSS, JavaScript, and may incorporate a PHP framework like Symfony for robust application development.

**Technologies Used:**

* **HTML:** Structure web pages.
* **CSS:** Styling and layout.
* **JavaScript:** Client-side interactions and dynamic content.
* **PHP:** Server-side scripting and business logic.
* **MySQL:** Database management and storage of user, service, and transaction data.

**Features:**

* **User Registration and Authentication:**
  + Freelancer and client registration with profile management.
  + Secure login system with role-based access (freelancer, client, admin).
* **Service Listing and Search:**
  + Freelancers can list their services with descriptions, pricing, and categories.
  + Clients can search for services based on categories, keywords, or location.
* **Booking and Payment System:**
  + Clients can book services offered by freelancers.
  + Secure payment integration (e.g., PayPal, Stripe) for transactions.
  + Handling of booking confirmations and cancellations.
* **Messaging and Communication:**
  + Integrated messaging system for communication between clients and freelancers.
  + Notification system for new messages and booking updates.
* **Admin Dashboard:**
  + Manage user accounts, services, and transactions.
  + View and moderate user-generated content (service listings, reviews).

**Database Schema:**

* **User Table:**
  + UserID (Primary Key)
  + Username
  + Password
  + Email
  + Role (Freelancer, Client, Admin)
* **Service Table:**
  + ServiceID (Primary Key)
  + Title
  + Description
  + Price
  + CategoryID (Foreign Key)
  + FreelancerID (Foreign Key)
* **Booking Table:**
  + BookingID (Primary Key)
  + ClientID (Foreign Key)
  + FreelancerID (Foreign Key)
  + ServiceID (Foreign Key)
  + BookingDate
  + Status (Confirmed, Pending, Canceled)

**Pages and Functionalities:**

* **Home Page:**
  + Overview of featured freelancers and services.
* **Registration/Login Page:**
  + User registration and login form.
* **Freelancer Profile Page:**
  + Display freelancer's profile, services offered, and reviews.
* **Service Listing and Search Page:**
  + Search and filter services based on categories, keywords, or location.
* **Booking and Payment Page:**
  + Book services and process payments securely.
* **Admin Panel:**
  + Manage users, services, bookings, and transactions.

Important Question

1. PHP Basics and Control Structures

2. What is PHP? How does it differ from client-side scripting languages like JavaScript?

3. Explain the process of setting up a local PHP development environment. Why is it important?

4. Describe PHP variables. What are the different data types supported in PHP?

5. Compare and contrast if, else, elseif statements in PHP. Provide an example of each.

6. What are loops in PHP? Explain the differences between for, while, do-while, and foreach loops with suitable examples.

7. Functions in PHP

8. What is a function in PHP? How do you define and call a function?

9. Explain the concept of variable scope in PHP. How are global and local variables managed within functions?

10. Why is it important to use functions in PHP development? Provide examples where functions could simplify code structure.

11. PHP Strings and Arrays

12. How do you manipulate strings in PHP? Provide examples of common string operations such as concatenation, substring, and searching.

13. What are arrays in PHP? Explain indexed arrays, associative arrays, and multidimensional arrays with examples.

14. Describe array functions in PHP. Give examples of functions like array\_push, array\_pop, array\_merge, and array\_sort.

15. PHP and HTML Integration, Form Handling

16. How do you embed PHP code within HTML files? Provide an example of mixing PHP and HTML to display dynamic content.

17. Explain the concept of form handling in PHP. What are the HTTP methods used for form submission?

18. Discuss form validation techniques in PHP. Why is input validation important in web development?

19. File Handling and MySQL Integration

20. What are the common file handling functions in PHP? Provide examples of reading from and writing to files using PHP.

21. What is MySQL? How do you connect PHP to MySQL databases using mysqli or PDO extensions?

22. Explain how to execute SQL queries from PHP to perform CRUD operations (Create, Read, Update, Delete) on a MySQL database.

23. Discuss the importance of project-based learning in PHP development. What are the key phases of developing a PHP project, and why are they important?

24. How can sessions and cookies be managed in PHP applications? What are their practical uses?

25. Explain the concept of sanitization and validation in PHP data handling. Why is it crucial to sanitize user inputs?