JSPM's
RAJARSHI SHAHU COLLEGE OF ENGINEERING
TATHAWADE, PUNE-33
(An Autonomous Institute Affiliated to Savitribai Phule Pune University, Pune)

S. Y. B. Tech (Computer Engineering)
Academic Year -2025-2026 (Semester –III)
Skill Course (VSEC)
[CS2213L]:- Web Technologies

Teaching Scheme:	**Credit**	**Examination Scheme:**
PR: - 2 Hours/Week	**PR: 1**	**Laboratory:** **ISCE : 30 Marks** **End Sem. Exam : 20 Marks**

Course Objective:

- To understand core web technologies and development methodologies.
- To develop static and responsive web pages using HTML, CSS, and JavaScript.
- To implement basic dynamic functionality using client-side and server-side scripting.
- To provide hands-on experience in web application deployment and content management systems**.**

Course Outcome:
After successful completion of the course, students will able to:
CO1: Apply HTML, CSS, and Bootstrap to create structured and styled responsive web pages.
CO2: Develop dynamic web pages using JavaScript, including DOM manipulation and form validation.
CO3: Build basic server-side applications with PHP or Node.js and perform database interactions.
CO4: Deploy a simple web application, ensuring responsive design and user-centric functionality.

UNIT-I	**Introduction to Web Technologies**	**04 Hours**
Internet, World Wide Web (WWW), and Web Browsers, HTML Basics: Tags, attributes, headings, lists, tables, images, forms, links. CSS Fundamentals: Inline, internal, and external CSS; basic styling (colors, fonts, layout); box model; positioning; flexbox/grid layout, Bootstrap, Responsive Design Basics**.**		
UNIT-II	**JavaScript and DOM Manipulation**	**03 Hours**
JavaScript Basics: Variables, functions, loops, arrays, objects, and events, DOM Manipulation: Accessing and modifying HTML elements dynamically using JavaScript, Basic Form Validation: Writing client-side validation scripts for forms using JavaScript.		
UNIT-III	**Introduction to Server-Side Technologies**	**04 Hours**
Server-Side Programming: Introduction to server-side languages, Database Interaction: Introduction to database concepts, Basic Dynamic Web Applications: Using PHP/Node.js/Reactjs for creating simple dynamic pages.		
UNIT-IV	**Web Application Deployment and Hosting**	**04 Hours**
Web Hosting Basics: Introduction to domain names, web hosting, and file deployment, Introduction to Content Management Systems (CMS): Overview of WordPress for content management.		

[illegible]	[illegible]	JSPM's Rajarshi Shahu College of Engineering Pune University I.D. No. PU/PN/Engg./173/(2001) Tathawade, Pune-33.	[illegible]
Dr. S. V. Kedar H.O.D, Computer	Dr. A. M. Badadhe Dean Academics		Dr. S. P. Bhosle Director RSCOE, Pune

Lab Contents

Guidelines for Assessment

1) Continuous assessment shall be based on experiments performed, submission of results of program in the form of report/journal, timely completion, attendance, understanding, efficient codes, punctuality and neatness.

2)Practical/Oral examination shall be based on the practical's performed in the lab.

3)Lab assessment marks shall be based on continuous assessment and performance in Practical/Oral examination

List of Laboratory Assignments/Experiments

1	**HTML, CSS and Bootstrap** Design a static webpage for a personal portfolio or a small business (e.g., a restaurant) with the following elements: • A header with a navigation menu • A section with information about the business or individual • A footer with contact information and social media links Use both inline and external CSS for styling.
2	**JavaScript and DOM Manipulation** Implement a simple "To-Do List" application using HTML, CSS, and JavaScript. The application should: • Allow users to add new tasks. • Mark tasks as completed. • Remove tasks from the list. Use JavaScript for DOM manipulation to update the task list dynamically.
3	**Form Validations** Create a contact form that includes the following fields: • Name, email, phone number, and message. • Implement JavaScript validation to check that all fields are filled correctly: o Ensure the email is in a valid format. o Ensure the phone number consists of only numbers and is 10 digits long. Display appropriate error messages if any validation fails.
4	**Server-Side Technologies (PHP)** Create a simple feedback form that: • Takes user input (name, email, feedback message). • Saves the feedback to a file or database (e.g., using MySQL). • Displays a confirmation message after submission. Implement basic error handling and form validation on the server side.
5	**Node.js / React js / Server-Side JavaScript** Create a basic server-side web application that performs the following: • Handles form submission from the client side • Processes and stores submitted data temporarily (JSON file). • Responds with a confirmation page or message after form submission. • Includes basic error handling for invalid routes (404 page).
6	**Web Application Deployment** • Deploy web application to a cloud platform (like GitHub Pages for static sites). • Ensure the application is accessible via a public URL.

[illegible]	[illegible]	JSPM's Rajarshi Shahu College of Engineering, Pune University I.D. No. PU/PN/Engg./173/(2001) Tathawade, Pune-33	[illegible]
Dr. S. V. Kedar H.O.D, Computer	Dr. A. M. Badadhe Dean Academics		Dr. S. P. Bhosle Director RSCOE, Pune

	• Implement basic error handling (404 pages, server errors). Test the functionality in a live environment.
7	**Mini Project**: Develop a web application for a real-world business scenario, such as an E-Commerce Store, Online Library Management System, or Event Booking System

Text Books:

T1: Jon Duckett, *"HTML and CSS: Design and Build Websites,"* Wiley, ISBN-13: 978-1118008188.
T2: Ethan Brown, *"Web Development with Node and Express: Leveraging the JavaScript Stack,"* O'Reilly Media, ISBN-13: 978-1492053507.
T3: Robin Nixon, *"Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5,"* O'Reilly Media, ISBN-13: 978-1492093824.

Reference Books:

R1: Simon Holmes, Clive Harber, *"Getting MEAN with Mongo, Express, Angular, and Node,"* Manning Publications, ISBN-13: 978-1617294754.
R2: Brad Dayley, Brendan Dayley, Caleb Dayley, *"Node.js, MongoDB, and Angular Web Development,"* Addison-Wesley Professional, ISBN-13: 978-0134655536.
R3: Robin Wieruch, *"The Road to React,"* ISBN-13: 978-1730853938.

MOOC's:

https://archive.nptel.ac.in/courses/106/106/106106156/
https://www.edx.org/certificates/professional-certificate/w3cx-front-end-web-developer

[illegible]	[illegible]	J.S.P.M.'s Rajarshi Shahu College of Engineering Pune University I.D. No. PU/PN/Engg./173/(2001) Tathawade, Pune-33.	[illegible]
Dr. S. V. Kedar H.O.D, Computer	Dr. A. M. Badadhe Dean Academics		Dr. S. P. Bhosle Director RSCOE, Pune

EXPERIMENT No. 1

Design a static webpage for a personal portfolio or a small business (e.g., a restaurant) with the following elements:

- A header with a navigation menu
- A section with information about the business or individual
- A footer with contact information and social media links

Use both inline and external CSS for styling.

Experiment No. 1

AIM: Study of HTML, CSS and Bootstrap

PROBLEM STATEMENT: Design a static webpage for a personal portfolio or a small business (e.g., a restaurant) with the following elements:

- A header with a navigation menu
- A section with information about the business or individual
- A footer with contact information and social media links

Use both inline and external CSS for styling.

REQUIREMENT: Text Editor ,Web browser

OPERATING SYSTEM: Windows/ Linux/Unix

THEORY:

1. HTML (HyperText Markup Language)

HTMLis used to design web pages using a markup language. A markup language is used to define the text document within the tag which defines the structure of web pages. HTML describes the structure of a Web page. HTML consists of a series of elements.HTML elements tell the browser how to display the content

Example: Creating a simple web page

1. Start Notepad.
2. Type in the following text:

```
<html>
    <head> <title>Title of page</title> </head>
    <body>
            This is a very basic webpage. <b>This text will be displayed in bold</b>
    </body>
</html>
```

3. Save the file as "firstpage.html".
4. Double-click the saved file the browser will display the page.

HTML TAGS

1. HTML tags are used to mark-up HTML elements

2. HTML tags are surrounded by the two characters < and >, called angle brackets.

3. HTML tags normally come in pairs like <h1> and </h1>**.** The first tag in a pair is the start tag, the second tag is the end tag

4. The text between the start and end tags is the element content

5. HTML tags are not case sensitive, **means <b> the same as <B>.**

6. Tags can have attributes. Attributes can provide additional information about the HTML elements:

with an added bgcolor attribute to <body>, you can tell the browser that the background color of your page should be red, like this: <body bgcolor="red" >, attributes always come in name/value pairs like this: name="value". Attributes are always added to the start tag of an HTML element.

HEADINGS

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading. <h6> defines the smallest heading.HTML automatically adds an extra blank line before and after a heading.

PARAGRAPHS

Paragraphs are defined with the <p> tag. <p>This is a paragraph</p>

HTML automatically adds an extra blank line before and after a paragraph.

LINE BREAKS

The <br> tag is used when you want to end a line, but don't want to start a new paragraph. The

<br> tag forces a line break wherever you place it.

Example: <p>This <br> is a para<br>graph with line breaks</p>

The <br> tag is an empty tag. It has no closing tag.

COMMENTS IN HTML

The comment tag is used to insert a comment in the HTML source code. A comment will be ignored by the browser. You can use comments to explain your code, which can help you when you edit the source code at a later date.

<!-- This is a comment --

TEXT FORMATTING TAGS

Tag	Description
<b>	Defines bold text
<big>	Defines big text
<em>	Defines emphasized text
<i>	Defines italic text
<small>	Defines small text
<strong>	Defines strong text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text

THE ANCHOR TAG AND THE HREF ATTRIBUTE

HTML uses the (anchor) tag to create a link to another document. An anchor can point to any resource on the Web: an HTML page, an image, a sound file, a movie, etc.
The syntax of creating an anchor: <a href="url">Text to be displayed</a>
<a href="https://www.google.com">Visit Google Page</a>

HTML LIST

HTML supports ordered, unordered and definition lists.
UNORDERED LISTS: An unordered list is a list of items. The list items are marked with bullets (typically small black circles).
An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.

```
<ul> <li>Coffee</li> <li>Milk</li> </ul>
```

ORDERED LISTS

An ordered list is also a list of items. The list items are marked with numbers. An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.

```
<ol> <li>Coffee</li> <li>Milk</li> </ol>
```

DEFINITION LISTS

A definition list is not a list of items. This is a list of terms and explanation of the terms.
A definition list starts with the <dl> tag. Each definition-list term starts with the <dt> tag. Each definition-list definition starts with the <dd> tag.

```
<dl> <dt>Coffee</dt> <dd>Black hot drink</dd>
     <dt>Milk</dt> <dd>White cold drink</dd>
</dl>
```

LIST TAGS

Tag	Description
<ol>	Defines an ordered list
<ul>	Defines an unordered list
<li>	Defines a list item
<dl>	Defines a definition list
<dt>	Defines a definition term
<dd>	Defines a definition description

What is CSS?

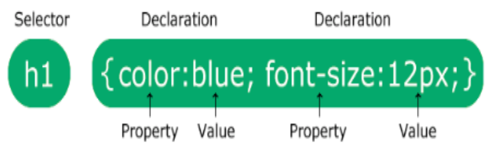
Cascading Style Sheets, referred to as CSS, is intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.

ADVANTAGES OF CSS

- CSS saves time
- Pages load faster
- Easy maintenance.
- Superior styles to HTML
- Multiple Device Compatibility
- Global web standards
- Offline Browsing
- Platform Independence

CSS Rule:

A CSS rule consists of a selector and a declaration block:



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

Example: `p {color: red; text-align: center;}`

- p is a selector in CSS.
- color is a property, and red is the property value
- text-align is a property, and center is the property value

There are three ways of inserting a style sheet:

- External CSS
- Internal CSS
- Inline CSS

Internal CSS

An internal style sheet may be used if one single HTML page has a unique style.

The internal style is defined inside the <style> element, inside the head section.

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: pink;}
h1 {
  color: maroon;
  margin-left: 40px;
   }
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Inline CSS

An inline style may be used to apply a unique style for a single element.To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

Example:

```
<!DOCTYPE html>

<html>

<body>

<h1 style="color:blue; text-align:center;">This is a heading</h1>

<p style="color:red;">This is a paragraph.</p>

</body>

</html>
```

External CSS

With an external style sheet, you can change the look of an entire website by changing just one file. Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="mystyle.css">
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

An external style sheet can be written in any text editor, and must be saved with a .css extension. The external .css file should not contain any HTML tags.

Here is the "mystyle.css" file :

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
  margin-left: 20px;
}
```

What is Bootstrap?

- Bootstrap is a free front-end framework for faster and easier web development
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins
- Bootstrap also gives you the ability to easily create responsive designs.
- Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites

What is Responsive Web Design?

Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.

Bootstrap Examples:

Example 1:

```
<div class="jumbotron text-center">
  <h1>My First Bootstrap Page</h1>
  <p>Resize this responsive page to see the effect!</p>
</div>

<div class="container">
  <div class="row">
    <div class="col-sm-4">
      <h3>Column 1</h3>
      <p>Lorem ipsum dolor..</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 2</h3>
      <p>Lorem ipsum dolor..</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 3</h3>
      <p>Lorem ipsum dolor..</p>
    </div>
  </div>
</div>
```

A jumbotron indicates a big box for calling extra attention to some special content or information. A jumbotron is displayed as a grey box with rounded corners. It also enlarges the font sizes of the text inside it.

Example 2:

Bootstrap different styles of buttons:

```
<button type="button" class="btn">Basic</button>
<button type="button" class="btn btn-default">Default</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-link">Link</button>
```

Conclusion:

EXPERIMENT No. 2

Implement a simple "To-Do List" application using HTML, CSS, and JavaScript. The application should:

- Allow users to add new tasks.
- Mark tasks as completed.
- Remove tasks from the list.

Use JavaScript for DOM manipulation to update the task list dynamically.

Experiment No. 2

AIM: Study of JavaScript and DOM Manipulation.

PROBLEM STATEMENT: Implement a simple "To-Do List" application using HTML, CSS, and JavaScript. The application should:

- Allow users to add new tasks.
- Mark tasks as completed.
- Remove tasks from the list.

Use JavaScript for DOM manipulation to update the task list dynamically.

REQUIREMENT: Text Editor, Web browser

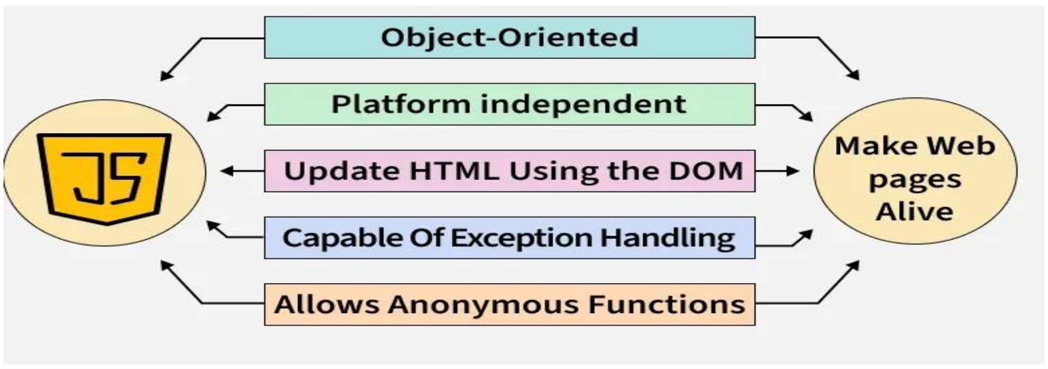
OPERATING SYSTEM: Windows/ Linux/Unix

THEORY:

JavaScript:

Introduction to JavaScript

Javascript, also known as JS, is a scripting language used for the web development. It is used to make HTML pages more interactive, and dynamic. It is used for building interactive web applications, supports both client-side and server-side development, and integrates seamlessly with HTML, CSS, and a rich standard library. The HTML <script> tag is used to define a client-side script (JavaScript). Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.


Object-Oriented
Platform independent
Update HTML Using the DOM
Capable Of Exception Handling
Allows Anonymous Functions
JS
Make Web pages Alive

Example: A "Hello, World!" program.

```
<html>
<head></head>
<body>
   <h1>Check the console for the message!</h1>
   <script>
      // This is our first JavaScript program
      console.log("Hello, World!");
   </script>
</body>
</html>
```

What is DOM?

DOM (Document Object Model) is a programming interface that transforms a web page into an object model that the browser can understand. When the browser loads a web page, it analyzes the content of the page and creates this content in memory in the form of a DOM tree. This tree structure contains all the elements of the page (e.g., title, paragraph, image, links), and these elements are related to each other.

Why is it important?

DOM offers web developers the ability to create dynamic and interactive content for user interactions. When the user clicks a button or submits a form, page content can be dynamically updated using the JavaScript DOM. This forms the basis of modern websites and applications.

DOM Elements

DOM methods are used to access and manipulate HTML elements on web pages using JavaScript. These methods are used to select and make changes to certain types of elements.

getElementById():

This method selects based on the id attribute of an element on the page.

```
var element = document.getElementById("myElementId");
```

getElementsByClassName():

This method is used to select all elements belonging to a particular class.

```
var elements = document.getElementsByClassName("myClassName");
```

getElementsByTagName():

This method is used to select all elements with a specific tag.

```
var elements = document.getElementsByTagName("div");
```

querySelector() and querySelectorAll():

querySelector() selects the first element that matches a given CSS selector, while querySelectorAll() selects all matching elements.

```
var element = document.querySelector("#myElementId");
var elements = document.querySelectorAll(".myClassName");
```

DOM Manipulation:

Web developers can perform DOM manipulation using JavaScript. This includes dynamically changing page content, adding new elements, removing existing elements, and updating style properties.

Changing Element Content:

We can use innerHTML or textContent properties to change the text content of an element within the DOM.

```
var element = document.getElementById("myElementId");
element.innerHTML = "New content"; // or element.textContent = "New content";
```

Creating a New Element:

We can add it to the page by creating a new HTML element.

```
var newElement = document.createElement("div");
newElement.innerHTML = "New Item";
document.body.appendChild(newElement);
```

Removing Elements:

To remove an existing element, we must find the parent element of the element and then use the removeChild() method.

```
var elementToRemove = document.getElementById("myElementId");
elementToRemove.parentNode.removeChild(elementToRemove);
```

Changing Element Styles and Properties:

We can change the CSS style properties and other properties of the elements.

```
var element = document.getElementById("myElementId");
element.style.color = "red";
```

DOM Events

Interactions on web pages are carried out using events to enable users to interact with the page. Events represent interactions such as a user clicking a mouse, pressing a key, submitting a form, etc. Using JavaScript, we can listen for these events and react to user interactions.

Event Listeners:

We can use the addEventListener() function to listen for a specific event. For example, to listen for a button click event:

```
var button = document.getElementById("myButton");
button.addEventListener("click", function() {
   // Actions to be taken when a click event occurs
});
```

Common Events:

- **click:** For click on an item.

- **mouseover and mouseout:** For hover over and move away from an element.
- **keydown and keyup:** For pressing a key and for it to work when you unclick it.
- **submit:** For submit a form.

Conclusion:

Experiment No: 3

Create a contact form that includes the following Fields:

- Name,email,phone number,and message
- Implement JavaScript validations to check that all fields are filled correctly:

    Ensure the email is in a valid format

    Ensure the phone number consists of only numbers and is 10 digits long.

Display appropriate error messages if any validation fails

Experiment No:3

Aim: To implement form validation using JavaScript.

PROBLEM STATEMENT:
Create a contact form that includes the following Fields:

- Name,email,phone number,and message
- Implement JavaScript validations to check that all fields are filled correctly:

    Ensure the email is in a valid format

    Ensure the phone number consists of only numbers and is 10 digits long.

Display appropriate error messages if any validation fails

Requirements:

Text Editor (Notepad, VS Code, Sublime Text etc.)

Web Browser (Chrome, Firefox, Edge etc.)

Operating System:
Windows / Linux / Unix

Theory:
HTML Forms:

HTML forms are used to collect user data.

The <form> tag contains form elements such as <input>, <textarea>, and <button>.

Example:

```
<form>

  <input type="text" placeholder="Enter Name">

  <input type="email" placeholder="Enter Email">

</form>
```

Form Validation:

Validation ensures user inputs are correct before submission.

Types of validation:

- Client-side Validation (JavaScript): Performed on the browser before submission.
- Server-side Validation: Performed on the server after submission.

JavaScript in Form Validation:

- JavaScript is widely used for real-time form validation.
- Common checks include:
- Empty Fields: Ensuring no field is left blank.

Email Format Validation: Using Regular Expressions (RegEx) to match patterns like abc@xyz.com.

Phone Number Validation: Checking for numeric values with fixed length (10 digits).

Example of Phone Number Validation:

```
var phonePattern = /^[0-9]{10}$/;

if (!phonePattern.test(phone)) {

   alert("Phone number must be 10 digits long");

}
```

Error Messages:

- Immediate feedback improves user experience.
- Helps prevent submission of incorrect or incomplete data.

Importance in Web Development:

- Ensures data integrity.
- Reduces server load by filtering invalid entries early.
- Improves usability of web applications.

Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Contact Form</title>
</head>
<body>
  <h2>Contact Form</h2>
  <form onsubmit="return validateForm()">
    Name: <input type="text" id="name"><br><br>
    Email: <input type="text" id="email"><br><br>
    Phone: <input type="text" id="phone"><br><br>
    Message: <textarea id="message"></textarea><br><br>
    <input type="submit" value="Submit">
  </form>
  <p id="error" style="color:red;"></p>
  <script>
    function validateForm() {
      let name = document.getElementById("name").value.trim();
      let email = document.getElementById("email").value.trim();
      let phone = document.getElementById("phone").value.trim();
      let msg = document.getElementById("message").value.trim();
      let error = "";
```

```
 if (!name) error = "Name is required";
      else if (!/^[^ ]+@[^ ]+\.[a-z]{2,}$/.test(email)) error = "Invalid email";
      else if (!/^[0-9]{10}$/.test(phone)) error = "Phone must be 10 digits";
      else if (!msg) error = "Message is required";
      document.getElementById("error").innerText = error;
      return error === "";
    }
  </script>
</body>
</html>
```

Conclusion:

Experiment No: 4

Create a simple feedback form that :

- Takes User Input(name,email,feedback message).
- Saves the Feedback to the file or Database.
- Display a confirmation message after submission.

Implement basic error handling and form validation on server side.

Experiment No:4

Aim: To study Server-Side Technologies using PHP

PROBLEM STATEMENT:
Create a simple feedback form that :

- Takes User Input(name,email,feedback message).
- Saves the Feedback to the file or Database.
- Display a confirmation message after submission.

Implement basic error handling and form validation on server side.

Requirements:
Text Editor (Notepad, VS Code, Sublime Text, etc.)

Web Browser (Chrome, Firefox, Edge, etc.)

XAMPP / WAMP Server (for running PHP and MySQL)

MySQL Database (optional, if using database storage)

Operating System:
Windows / Linux / Unix

Theory:
Introduction to Server-Side Programming:

Unlike client-side scripting (JavaScript), server-side scripting is executed on the web server.

PHP is a widely used server-side scripting language for building dynamic web applications.

PHP in Web Development:

PHP stands for Hypertext Preprocessor.

It is embedded within HTML and executed on the server to generate dynamic content.

PHP scripts can interact with databases, handle form submissions, and manage sessions.

Form Handling in PHP:

Forms use the POST or GET method to send data to the server.

PHP retrieves this data using $_POST[] or $_GET[] superglobals.

Example:

$name = $_POST['name'];

$email = $_POST['email'];

$message = $_POST['message'];

Validation in PHP:

Ensures correctness and completeness of submitted data.

Examples:

Empty Check: empty() function ensures fields are not blank.

Email Check: filter_var($email, FILTER_VALIDATE_EMAIL) ensures a valid email format.

Storing Feedback:

File Storage: Feedback can be saved in a text file using fopen(), fwrite(), and fclose().

Database Storage (MySQL): Feedback can be inserted into a database using SQL queries with MySQL_query().

Error Handling:

Handles invalid inputs and server errors.

Example: Display a custom error message if database connection fails.

Confirmation Message:

After successful submission, the server responds with a confirmation (e.g., “Thank you for your feedback”).

Sample Code with MySQL Storage

HTML Form (feedback.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>Feedback Form</title>
</head>
<body>
    <h2>Feedback Form</h2>
    <form action="feedback.php" method="post">
        Name: <input type="text" name="name" required><br><br>
        Email: <input type="email" name="email" required><br><br>
        Message:<br>
        <textarea name="message" rows="4" cols="30" required></textarea><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

PHP Script with MySQL (feedback.php):

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Get form data
    $name = trim($_POST['name']);
    $email = trim($_POST['email']);
    $message = trim($_POST['message']);
    // Basic validation
    if (empty($name) || empty($email) || empty($message)) {
        echo "All fields are required!";
    } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "Invalid email format!";
    } else {
        // Database connection
        $servername = "localhost";   // Change if needed
        $username   = "root";        // Default XAMPP/WAMP user
        $password   = "";            // Default is empty
        $dbname     = "feedback_db"; // Database name
        $conn = new mysqli($servername, $username, $password, $dbname);
        // Check connection
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }
        // Insert query
        $sql = "INSERT INTO feedback (name, email, message)
                VALUES ('$name', '$email', '$message')";
        if ($conn->query($sql) === TRUE) {
            echo "Thank you, $name. Your feedback has been recorded!";
        } else {
            echo "Error: " . $sql . "<br>" . $conn->error;
        }
        $conn->close();
    }
}
?>
```

SQL Query to Create Table (Run in phpMyAdmin or MySQL CLI):

```
CREATE DATABASE feedback_db;

USE feedback_db;

CREATE TABLE feedback (

    id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(100) NOT NULL,

    email VARCHAR(100) NOT NULL,

    message TEXT NOT NULL,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);
```

✅**This version will:**

Store the feedback in a MySQL database (feedback_db).

Provide server-side validation for empty fields and email format.

Display a confirmation message after successful submission.

Conclusion:

Experiment No: 5

Create a basic server-side web application that performs the following:

- Handles form submission from the client side
- Processes and stores submitted data temporarily (JSON file).
- Responds with a confirmation page or message after form submission. □
- Includes basic error handling for invalid routes (404 page).

Experiment No:5

Aim: To develop Node.js/React.js/Server-Side web application using JavaScript.

PROBLEM STATEMENT:
Create a server-side web application that accepts user input through a form, stores the data temporarily, and displays confirmation. Include error handling for invalid routes.

REQUIREMENT:
Python, Flask library, Text Editor, Web Browser

OPERATING SYSTEM:
Windows / Linux / Unix

Theory:
Server-side web applications handle requests from the client, process data, interact with storage, and respond with HTML or other content. Flask is a lightweight web framework in Python, ideal for learning and building small applications.

1. Core concepts (short)
- Request / Response: Client → server (method, headers, body) → server responds (status, headers, body).

- Statelessness: HTTP is stateless; use persistent storage or sessions to keep data between requests.

- Routes: URL + method mapped to a handler (e.g., GET / shows form; POST /submit processes it).

2. Minimal form flow (logical steps)
1. Show HTML form (GET /).

2. User fills and posts (POST /submit).

3. Server validates input (required fields, email format, lengths).

4. If invalid → return error page or show messages. If valid → store (temp) and show confirmation.

5. Optional: view current submissions (GET /submissions).

3. Security & good practice (brief)
- Server-side validation: Always validate on the server (client-side is UX only).

- Escaping: Escape user data in templates to prevent XSS.

- HTTPS & CSRF: In production, use HTTPS and CSRF tokens for POSTs.

- Secrets: Don't store secrets in code — use environment variables.

4. Temporary vs persistent storage (short)
- In-memory list: easiest, lost on restart — good for demos.

- JSON file / SQLite: next step for persistence. Use atomic writes or transactions to avoid corruption.

- DBMS (Postgres/Mongo): for scaling and multi-user apps.

5. Error handling (compact)
Return 400 for validation errors, 404 for invalid routes, 500 for server issues. Provide friendly error pages and log details server-side.

6. Small Flask example

```
from flask import Flask, request, render_template_string, redirect, url_for
app = Flask(__name__)
submissions = []

@app.route('/', methods=['GET'])
def form():
   return render_template_string('<form method="post" action="/submit">Name:
<input name="name"><button>Send</button></form>')

@app.route('/submit', methods=['POST'])
def submit():
   name = (request.form.get('name') or '').strip()
   if len(name) < 2:
      return "Name too short", 400
   submissions.append({'name': name})
   return f"Thanks, {name}!"

@app.errorhandler(404)
def not_found(e):
   return "404 — not found", 404
```

7. How to expand (quick roadmap)
- Add server-side templates (Jinja2) and CSS.

- Replace in-memory storage with SQLite (or JSON file with atomic write).

- Add form validation library (WTForms) and CSRF protection.

- Add authentication if submissions are user-specific.

- Add tests (Flask test_client() / pytest).

- Swap Flask dev server for Gunicorn and add Dockerfile for deployment.

8. Expected Output Steps
- Start the server: Run the app (e.g., `node express_form_app.js` or `python app.py`). Console should show:
Server running: http://localhost:3000

- GET / — Form page (200 OK): Browser shows the form page with fields: Name, Email, Message, and a Submit button. Look for the heading 'Send us a message'.

- POST /submit — valid input (200 OK): On successful submission you should see a confirmation page echoing the sanitized values and a timestamp. Example: 'Thank you, Swapnali! Your message was received and stored temporarily.'

- POST /submit — invalid input (400 Bad Request): If validation fails (short name, invalid email, short message) you should see a friendly error page listing the problems and a link back to the form.

- GET /submissions — list stored items (200 OK): Shows the current in-memory submissions with ID, name, email, timestamp, and message. Useful for demo verification.

- Unknown route — 404 Not Found: Any unknown URL returns a 404 page that mentions the requested path and offers a link back to home.

- Server error — 500 Internal Server Error: On unhandled exceptions the server returns a generic 500 page and logs the error on the server console.

- Quick curl tests: Examples:
1) GET the form:
```
curl -i http://localhost:3000/
```

2) POST valid data:
```
curl -i -X POST http://localhost:3000/submit -d
```

```
"name=Swapnali&email=swapnali@example.com&message=Hello"
```

3) POST invalid data:

```
   curl -i -X POST http://localhost:3000/submit -d "name=S&email=bad&message=hi"
```

4) Request unknown route:

```
   curl -i http://localhost:3000/invalid-route
```

Quick verification checklist

• Server prints startup message with localhost URL.

• Opening / shows the form with Name, Email, Message fields.

• Valid POST returns confirmation with sanitized values and timestamp.

• Invalid POST returns 400 with error messages.

• /submissions lists stored entries.

• Unknown route returns a 404 page.

• Server logs errors and 500 page is user-friendly.

Conclusion:

Experiement No:6

Web Application Deployment

- Deploy web application to a cloud platform
- Ensure the application is accessible via a public URL
- Implement basic error handling(404 pages,server errrors)

Experiment No:6

Aim: To deploy a basic web application to a cloud platform (e.g., GitHub Pages or Python Anywhere), ensure it is accessible via a public URL, and implement basic error handling (404 pages and server errors).

PROBLEM STATEMENT:

- Deploying a web application to a cloud platform(like GitHub Pages for static sites)
- Ensure the application is accesible via a public URL.
- Implement basic error handling(404 pages,server errors).

Requirements: A computer with internet, code editor, Git/Python, and a web browser.

Operating System: Windows, Linux, or macOS.

Theory:

Web application deployment is the process of transferring a web project from a development environment (local machine) to a production environment (web server or cloud platform) so that it can be accessed globally through the internet. The process ensures that applications are stable, scalable, and maintain proper error handling mechanisms. Deployment can be broadly categorized into static hosting and dynamic hosting, depending on the nature of the application.

1. Static Hosting

- Definition: Static websites consist of fixed content like HTML, CSS, JavaScript, and images. They do not require a backend server for processing.
- Examples of platforms: GitHub Pages, Netlify, Vercel.
- Use cases: Personal portfolios, documentation sites, landing pages.
- Example: A simple portfolio website with index.html, style.css, and script.js files can be deployed on GitHub Pages. If a user types a wrong URL, GitHub Pages can serve a custom 404.html page.

2. Dynamic Hosting
- Definition: Dynamic applications include server-side logic, databases, and frameworks like Flask, Django, or Node.js. These require a hosting environment capable of running backend code.
- Examples of platforms: Python Anywhere, Heroku, Render, AWS, Azure.
- Use cases: E-commerce applications, blogs with user authentication, data-driven dashboards.
- Example: A Flask-based blog application that fetches articles from a SQLite or MySQL database can be deployed on Python Anywhere. The app dynamically generates pages depending on user requests.

3. Key Deployment Concepts
- Build and Release: Converting source code into a deployable form (e.g., minified CSS/JS or packaged Python apps).
- Continuous Integration/Continuous Deployment (CI/CD): Automates testing and deployment using tools like GitHub Actions or Jenkins.
- Scalability: Cloud platforms allow applications to handle increased traffic by scaling horizontally (adding more servers) or vertically (upgrading resources).

4. Error Handling in Deployment
Error handling ensures a smooth user experience and provides meaningful feedback in case of issues:
- 404 Not Found: Occurs when a requested resource/page does not exist. Example: On GitHub Pages, creating a 404.html file allows developers to display a custom error message like “Oops! Page not found.”
- 500 Internal Server Error: Occurs due to issues in server-side code or misconfiguration. Example: In Flask, error handling can be added as:

```
@app.errorhandler(500)
def server_error(e):
    return "An internal error occurred. Please try again later.", 500
```

- Best Practice: Log errors (using logging libraries) to identify and fix problems quickly.

5. Advantages of Cloud Deployment
- Accessibility: Applications can be accessed globally through a public URL.
- Cost-Effectiveness: Many platforms (GitHub Pages, Render, Heroku free tier) offer free hosting for small projects.
- Security: SSL certificates can be enabled to serve applications over HTTPS.
- Flexibility: Developers can use custom domains for branding.

Web deployment is the process of publishing a website or web application from a local environment to an online server so that users can access it through the internet.

There are two main types:

- Static Hosting (e.g., GitHub Pages): Used for HTML, CSS, JS files.

- Dynamic Hosting (e.g., PythonAnywhere, Render): Used for backend apps like Flask, Django, Node.js, etc.

Implementation Steps:

A. For Static Sites using GitHub Pages:

1. Create a GitHub repository (e.g., my-web-app).

2. Add index.html and optionally 404.html files.

3. Commit and push changes to the main branch.

4. Go to Settings → Pages and select main branch as source.

5. GitHub generates a public URL like https://username.github.io/my-web-app/.

6. Visit the link to check your site.

B. For Dynamic Flask App using Python Anywhere:

1. Create a Python Anywhere account.

2. Upload app.py and required files.

3. Set up a web app → Choose Flask → Python version.

4. Update wsgi.py to point to your Flask app.

5. Reload the app and visit the generated public URL.

6. Add 404 error handler in Flask using:

```
@app.errorhandler(404)
def page_not_found(e):
   return "Page Not Found", 404
```

Expected Output:
- The application is successfully hosted on a public URL.

- Static/dynamic content loads as expected.

- Visiting a wrong route shows a custom 404 error page.

- The deployed URL is accessible from any device.

Enhancements (Optional):
- Use a custom domain.

- Add SSL certificate (HTTPS).

- Set up CI/CD using GitHub Actions.

- Log server errors for debugging.

Conclusion: