**WEB TECHNOLOGIES LAB MANUAL**

**Experiment 1: HTML Page**

**a) Create a webpage with HTML describing your department. Use paragraph and list tags.**

* **Aim**: To create a basic HTML webpage describing the department using paragraph and list tags.
* **Software Required**: Text Editor (VS Code, Sublime Text, Notepad++), Web Browser (Chrome, Firefox, Edge).
* **Description**: This task involves creating a simple webpage with paragraphs describing the department and using list tags to display additional information.
* **Theory**: The <p> tag is used to define paragraphs, while <ul>, <ol>, and <li> tags are used to create unordered and ordered lists.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Department of Computer Science</title>

</head>

<body>

<h1>Department of Computer Science</h1>

<p>The Department of Computer Science focuses on providing high-quality education and research in various areas of computing.</p>

<p>Our programs include:</p>

<ul>

<li>Bachelor of Computer Science</li>

<li>Master of Computer Science</li>

<li>PhD in Computer Science</li>

</ul>

<p>We aim to equip students with the skills necessary for the modern technological landscape.</p>

</body>

</html>

* **Output**: A webpage with a title, paragraphs, and an unordered list displaying the department’s programs.
* **Result**: The webpage displays information about the department, including educational programs.

**b) Apply various colors to suitably distinguish key words. Also apply font styling like italics, underline, and two other fonts to words you find appropriate. Also use header tags.**

* **Aim**: To apply colors and font styles to various elements in the webpage.
* **Software Required**: Same as above.
* **Description**: This task involves using color and font styling to distinguish keywords and adding various header tags.
* **Theory**: CSS properties like color, font-style, text-decoration, and font-family are used to style text elements.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Department of Computer Science</title>

</head>

<body>

<h1 style="color: blue; font-family: Arial, sans-serif;">Department of Computer Science</h1>

<p>The <i style="color: green;">Department</i> of Computer Science provides high-quality education and research opportunities.</p>

<p><u>Key areas of study include:</u></p>

<ul>

<li style="color: red;">Software Engineering</li>

<li style="font-weight: bold;">Artificial Intelligence</li>

<li style="font-family: 'Times New Roman', serif;">Data Science</li>

</ul>

<h2 style="color: purple;">Programs Offered:</h2>

<ul>

<li>Bachelor's in Computer Science</li>

<li>Master's in Computer Science</li>

<li>PhD in Computer Science</li>

</ul>

</body>

</html>

* **Output**: The webpage will show various styles applied to text, with color, italic, underline, and custom fonts.
* **Result**: Styling was successfully applied to distinguish keywords and make text more visually appealing.

**c) Create links on the words e.g. "Wi-Fi" and "LAN" to link them to Wikipedia pages.**

* **Aim**: To add hyperlinks to specific words, linking to external Wikipedia pages.
* **Software Required**: Same as above.
* **Description**: This task requires creating hyperlinks that navigate to Wikipedia pages when clicked.
* **Theory**: The <a> tag is used to create hyperlinks in HTML.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Department of Computer Science</title>

</head>

<body>

<h1>Department of Computer Science</h1>

<p>The department offers advanced research in networking and communication technologies such as <a href="https://en.wikipedia.org/wiki/Wi-Fi" target="\_blank">Wi-Fi</a> and <a href="https://en.wikipedia.org/wiki/Local\_area\_network" target="\_blank">LAN</a>.</p>

</body>

</html>

* **Output**: Clicking the "Wi-Fi" and "LAN" links will open their respective Wikipedia pages in a new tab.
* **Result**: Hyperlinks were successfully created, linking to external Wikipedia pages.

**d) Insert an image and create a link such that clicking on the image takes the user to another page.**

* **Aim**: To insert an image and make it clickable, redirecting to another page.
* **Software Required**: Same as above.
* **Description**: This task requires embedding an image in the webpage and using the <a> tag to make it clickable.
* **Theory**: The <img> tag is used to embed images, and the <a> tag is used to make them clickable.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Department of Computer Science</title>

</head>

<body>

<h1>Welcome to the Department of Computer Science</h1>

<a href="https://www.example.com">

<img src="department\_image.jpg" alt="Department Image" width="300" height="200">

</a>

</body>

</html>

* **Output**: Clicking the image will take the user to "<https://www.example.com>".
* **Result**: The image is successfully linked to a new webpage.

**e) Change the background color of the page. At the bottom create a link to take the user to the top of the page.**

* **Aim**: To change the background color of the webpage and create a link to return to the top of the page.
* **Software Required**: Same as above.
* **Description**: This task involves setting a background color for the page and creating a "Back to Top" link.
* **Theory**: The background-color property is used to change the background color, and the href="#top" creates an anchor link.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Department of Computer Science</title>

<style>

body { background-color: lightblue; }

</style>

</head>

<body>

<h1 id="top">Department of Computer Science</h1>

<p>The department focuses on cutting-edge research and development.</p>

<a href="#top">Back to Top</a>

</body>

</html>

* **Output**: The webpage background will be light blue, and there will be a link at the bottom to return to the top.
* **Result**: The background color was changed, and the "Back to Top" link works correctly.

**Experiment 2: Tables**

**a) Create a table to show your class timetable.**

* **Aim**: To create a table displaying the class timetable.
* **Software Required**: Same as above.
* **Description**: This task involves creating a table to display the class timetable.
* **Theory**: Tables use <table>, <tr>, <th>, and <td> tags to organize data into rows and columns.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Class Timetable</title>

</head>

<body>

<h1>Class Timetable</h1>

<table border="1">

<tr>

<th>Day</th>

<th>Subject</th>

<th>Time</th>

</tr>

<tr>

<td>Monday</td>

<td>Math</td>

<td>9:00 AM - 10:30 AM</td>

</tr>

<tr>

<td>Tuesday</td>

<td>Physics</td>

<td>11:00 AM - 12:30 PM</td>

</tr>

</table>

</body>

</html>

* **Output**: A table displaying the class timetable with days, subjects, and time slots.
* **Result**: Table structure was successfully implemented for the timetable.

**b) Use tables to provide layout to your HTML page describing your university infrastructure.**

* **Aim**: To use tables for layout instead of divs or CSS grids.
* **Software Required**: Same as above.
* **Description**: This task involves using a table to layout the content of the webpage describing university infrastructure.
* **Theory**: Tables can also be used for page layout by organizing content into rows and columns.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>University Infrastructure</title>

</head>

<body>

<h1>University Infrastructure</h1>

<table border="1">

<tr>

<th>Building</th>

<th>Description</th>

</tr>

<tr>

<td>Main Building</td>

<td>Housing classrooms and offices.</td>

</tr>

<tr>

<td>Library</td>

<td>A spacious library with various academic resources.</td>

</tr>

</table>

</body>

</html>

* **Output**: The infrastructure information is displayed in a table format.
* **Result**: Layout achieved using tables for organizing infrastructure details.

**c) Use <span> and <div> tags to provide a layout to the above page instead of a table layout.**

* **Aim**: To use <span> and <div> tags for layout.
* **Software Required**: Same as above.
* **Description**: This task involves using block-level and inline elements like <div> and <span> for page layout.
* **Theory**: The <div> tag is a block-level element used for grouping content, while <span> is an inline element used for styling text.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>University Infrastructure</title>

<style>

.building {

margin-bottom: 10px;

padding: 10px;

border: 1px solid #ccc;

}

</style>

</head>

<body>

<h1>University Infrastructure</h1>

<div class="building">

<h2>Main Building</h2>

<p>Housing classrooms and offices.</p>

</div>

<div class="building">

<h2>Library</h2>

<p>A spacious library with various academic resources.</p>

</div>

</body>

</html>

* **Output**: The content is organized using <div> elements.
* **Result**: The layout was achieved using <span> and <div> for grouping content.

**d) Use frames such that the page is divided into 3 frames: 20% on left for content, 60% in the center for the body, and 20% on the right for remarks.**

* **Aim**: To create a page layout using frames.
* **Software Required**: Same as above.
* **Description**: This task involves dividing the webpage into 3 frames using the <frameset> tag.
* **Theory**: The <frameset> element is used to create frames, and the cols attribute specifies the layout proportions.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Frames Layout</title>

</head>

<frameset cols="20%, 60%, 20%">

<frame src="left.html">

<frame src="center.html">

<frame src="right.html">

</frameset>

</html>

* **Output**: The page will be divided into 3 frames: left, center, and right.
* **Result**: The frame layout was successfully implemented.

**e) Embed audio and video into your HTML web page.**

* **Aim**: To embed audio and video files into a webpage.
* **Software Required**: Same as above.
* **Description**: This task involves embedding audio and video files using the <audio> and <video> tags.
* **Theory**: The <audio> and <video> tags are used to embed media elements in HTML.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Media Embedding</title>

</head>

<body>

<h1>Media Embedding Example</h1>

<audio controls>

<source src="audio\_sample.mp3" type="audio/mp3">

</audio>

<video width="320" height="240" controls>

<source src="video\_sample.mp4" type="video/mp4">

</video>

</body>

</html>

* **Output**: Audio and video players will appear on the webpage.
* **Result**: Media elements were successfully embedded in the HTML page.

**Experiment 3: CSS (Cascading Style Sheets)**

**a) Apply in-line CSS to change colors of certain text portions, bold, underline, and italics certain words in your HTML webpage. Also change the background color of each paragraph using in-line CSS.**

* **Aim**: To apply in-line CSS to style text portions in HTML.
* **Software Required**: Same as above.
* **Description**: This task involves using in-line CSS to apply various text styles.
* **Theory**: In-line CSS is applied directly within HTML tags using the style attribute.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Styled Webpage</title>

</head>

<body>

<h1 style="color: blue;">Welcome to the Department of Computer Science</h1>

<p style="background-color: lightyellow; color: green; font-style: italic;">The department offers a variety of programs to suit your interests.</p>

<p style="background-color: lightgray; font-weight: bold;">Research areas include Artificial Intelligence and Data Science.</p>

<p style="background-color: lightblue; text-decoration: underline;">Apply now and join us in the world of technology!</p>

</body>

</html>

* **Output**: The webpage will display text with different styles: background colors for each paragraph, and styles like italic, bold, and underlined for specific words.
* **Result**: Successfully applied in-line CSS for text styling and background color changes.

**b) Write all the above styling in CSS in a different file (.css) and link it to your webpage such that changes made in the CSS file are immediately reflected on the page. Group paragraphs into a single class and add styling information to the class in CSS.**

* **Aim**: To link an external CSS file to your HTML page and apply styles to group elements.
* **Software Required**: Same as above.
* **Description**: This task involves creating a separate CSS file and linking it to the HTML file. You will also group paragraphs into a class.
* **Theory**: External CSS is linked to HTML using the <link> tag. CSS classes are defined to apply styling to a group of elements.
* **Code**:

**HTML File** (index.html):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Styled Webpage</title>

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

</head>

<body>

<h1>Welcome to the Department of Computer Science</h1>

<p class="styled-text">The department offers a variety of programs to suit your interests.</p>

<p class="styled-text">Research areas include Artificial Intelligence and Data Science.</p>

<p class="styled-text">Apply now and join us in the world of technology!</p>

</body>

</html>

**CSS File** (styles.css):

.styled-text {

background-color: lightyellow;

font-style: italic;

color: green;

}

h1 {

color: blue;

font-family: Arial, sans-serif;

}

* **Output**: The styles defined in the CSS file will be applied to the webpage.
* **Result**: External CSS file was successfully linked and applied to the webpage.

**c) Create a simple form to submit user input like name, age, address, favorite subject, movie, and singer.**

* **Aim**: To create a simple form for collecting user information.
* **Software Required**: Same as above.
* **Description**: This task involves creating a form with input fields for name, age, address, favorite subject, movie, and singer.
* **Theory**: Forms in HTML use the <form>, <input>, <textarea>, and <select> elements to collect user input.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>User Information Form</title>

</head>

<body>

<h1>User Information Form</h1>

<form>

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

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

<label for="age">Age:</label>

<input type="number" id="age" name="age"><br><br>

<label for="address">Address:</label>

<textarea id="address" name="address"></textarea><br><br>

<label for="subject">Favorite Subject:</label>

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

<label for="movie">Favorite Movie:</label>

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

<label for="singer">Favorite Singer:</label>

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

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

</form>

</body>

</html>

* **Output**: A form that collects the user’s name, age, address, and other details.
* **Result**: The form successfully collects data from the user.

**d) Add a few form elements such as radio buttons, checkboxes, and a password field. Add a submit button at last.**

* **Aim**: To add additional form elements like radio buttons, checkboxes, and a password field.
* **Software Required**: Same as above.
* **Description**: This task involves enhancing the form to include radio buttons, checkboxes, and a password field for user input.
* **Theory**: Radio buttons allow a user to choose only one option from a set, checkboxes allow multiple selections, and the <input type="password"> field is used for password input.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>User Information Form</title>

</head>

<body>

<h1>User Information Form</h1>

<form>

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

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

<label for="age">Age:</label>

<input type="number" id="age" name="age"><br><br>

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

<input type="radio" id="male" name="gender" value="Male"> Male

<input type="radio" id="female" name="gender" value="Female"> Female<br><br>

<label for="interests">Interests:</label><br>

<input type="checkbox" id="sports" name="interests" value="Sports"> Sports<br>

<input type="checkbox" id="music" name="interests" value="Music"> Music<br><br>

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

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

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

</form>

</body>

</html>

* **Output**: The form will have radio buttons, checkboxes, a password field, and a submit button.
* **Result**: The form allows users to select gender, interests, and enter a password.

**Experiment 4: JavaScript**

**a) Create a form similar to the one in the previous experiment. Put validation checks on values entered by the user using JavaScript (such as age should be a value between 1 and 150).**

* **Aim**: To add JavaScript validation to form inputs.
* **Software Required**: Same as above.
* **Description**: This task involves validating the form using JavaScript to ensure that the age is between 1 and 150.
* **Theory**: JavaScript can be used to validate form input by checking the values entered by the user before submission.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Form with Validation</title>

<script>

function validateForm() {

var age = document.getElementById("age").value;

if (age < 1 || age > 150) {

alert("Age must be between 1 and 150.");

return false;

}

return true;

}

</script>

</head>

<body>

<h1>User Information Form</h1>

<form onsubmit="return validateForm()">

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

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

<label for="age">Age:</label>

<input type="number" id="age" name="age"><br><br>

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

</form>

</body>

</html>

* **Output**: The form will display an alert if the age is not between 1 and 150.
* **Result**: JavaScript validation is successfully implemented to check the age input.

**b) Write a JavaScript program to display an information box as soon as the page loads.**

* **Aim**: To display an information box when the page loads using JavaScript.
* **Software Required**: Same as above.
* **Description**: This task involves displaying an information box (using alert()) when the page loads.
* **Theory**: JavaScript's alert() function can be used to display a simple message when the page loads.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Information Box</title>

<script>

window.onload = function() {

alert("Welcome to the Department of Computer Science!");

}

</script>

</head>

<body>

<h1>Department of Computer Science</h1>

</body>

</html>

* **Output**: An alert box will pop up when the page loads with the message "Welcome to the Department of Computer Science!".
* **Result**: Information box successfully displayed when the page loaded.

**c) Write a JavaScript program to change the background color after 5 seconds of page load.**

* **Aim**: To change the background color after a delay using JavaScript.
* **Software Required**: Same as above.
* **Description**: This task involves using setTimeout() to change the background color of the page after a 5-second delay.
* **Theory**: The setTimeout() function allows you to execute a function after a specified amount of time.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Change Background Color</title>

<script>

window.onload = function() {

setTimeout(function() {

document.body.style.backgroundColor = "lightblue";

}, 5000); // 5000ms = 5 seconds

}

</script>

</head>

<body>

<h1>Welcome to the Department of Computer Science</h1>

</body>

</html>

* **Output**: The background color will change to light blue after 5 seconds.
* **Result**: The background color was successfully changed after a delay.

**d) Write a JavaScript program to dynamically bold, italic, and underline words and phrases based on user actions.**

* **Aim**: To dynamically change text styles based on user interaction.
* **Software Required**: Same as above.
* **Description**: This task involves adding JavaScript functions to bold, italicize, and underline text when buttons are clicked.
* **Theory**: JavaScript can manipulate DOM elements to apply styles dynamically.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Dynamic Text Style</title>

<script>

function boldText() {

document.getElementById("text").style.fontWeight = "bold";

}

function italicText() {

document.getElementById("text").style.fontStyle = "italic";

}

function underlineText() {

document.getElementById("text").style.textDecoration = "underline";

}

</script>

</head>

<body>

<h1>Dynamic Text Style</h1>

<p id="text">This is some text that can be styled.</p>

<button onclick="boldText()">Bold</button>

<button onclick="italicText()">Italic</button>

<button onclick="underlineText()">Underline</button>

</body>

</html>

* **Output**: Buttons will apply bold, italic, or underline styles to the text dynamically.
* **Result**: The text style was successfully changed based on user actions.

**e) Write a JavaScript program to display a hidden div (e.g., showing stats of a player when the user clicks on his name).**

* **Aim**: To reveal hidden content using JavaScript when a user clicks on a button.
* **Software Required**: Same as above.
* **Description**: This task involves showing hidden content (a div) when the user clicks a button.
* **Theory**: JavaScript can change the display property of a div to show or hide it.
* **Code**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Show Stats</title>

<script>

function showStats() {

document.getElementById("stats").style.display = "block";

}

</script>

</head>

<body>

<h1>Click the button to see player stats</h1>

<button onclick="showStats()">Show Player Stats</button>

<div id="stats" style="display:none;">

<h2>Player Stats</h2>

<p>Goals: 25</p>

<p>Assists: 10</p>

<p>Matches Played: 30</p>

</div>

</body>

</html>

* **Output**: The hidden stats will be revealed when the user clicks the button.
* **Result**: Hidden content successfully displayed on user action.

**Experiment 5: Servlets**

**a) Create a user registration web application.**

* **Aim**: To create a user registration system using servlets.
* **Software Required**: Java, Apache Tomcat, Eclipse IDE (or any Java IDE).
* **Description**: This task involves creating a web application for user registration where users can input their details.
* **Theory**: Servlets are Java programs that run on a server to handle client requests and send responses. They are used to handle HTTP requests and responses for web applications.
* **Code**:

**User Registration Form (HTML)** (register.html):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>User Registration</title>

</head>

<body>

<h1>User Registration</h1>

<form action="registerServlet" method="POST">

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

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

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

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

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

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

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

</form>

</body>

</html>

**Servlet Code (Java)** (RegisterServlet.java):

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class RegisterServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String username = request.getParameter("username");

String password = request.getParameter("password");

String email = request.getParameter("email");

// For demonstration, we are just displaying the data

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<h1>User Registration Success</h1>");

out.println("<p>Username: " + username + "</p>");

out.println("<p>Password: " + password + "</p>");

out.println("<p>Email: " + email + "</p>");

}

}

* **Output**: Upon submitting the registration form, the servlet will process the data and show the entered details.
* **Result**: The registration system was successfully implemented using a servlet.

**b) At the server end, write code to retrieve contents of the request object and show them to the user. Match user input password with predefined password and show “Valid User” or “Invalid User”.**

* **Aim**: To retrieve user input from the request object and validate it against a predefined password.
* **Software Required**: Same as above.
* **Description**: This task involves retrieving user input from the request object and performing validation.
* **Theory**: The request object in servlets holds data sent by the client (user) via HTTP methods. Servlets can retrieve parameters using getParameter() and validate the data as needed.
* **Code**:

**HTML Form (login.html)**

html

<!DOCTYPE html>

<html>

<head>

<title>Login Page</title>

</head>

<body>

<h1>Login</h1>

<form action="LoginServlet" method="POST">

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

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

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

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

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

</form>

</body>

</html>

**Servlet Code (Java)** (LoginServlet.java):

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class LoginServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String username = request.getParameter("username");

String password = request.getParameter("password");

String predefinedPassword = "admin123";

response.setContentType("text/html");

PrintWriter out = response.getWriter();

if (password.equals(predefinedPassword)) {

out.println("<h1>Valid User</h1>");

out.println("<p>Welcome " + username + "</p>");

} else {

out.println("<h1>Invalid User</h1>");

out.println("<p>Incorrect password.</p>");

}

}

}

* **Output**: Depending on the user’s input, the servlet will display either "Valid User" or "Invalid User".
* **Result**: User input was successfully validated against a predefined password.

**c) A web application name as input, and on submit, it should show hello <name>. It shows the start time at the right top corner of the page and provides a logout button. On clicking the logout button, it should go to the logout page that shows "Thank you <name>" with duration of usage.**

* **Aim**: To create a session-based user interaction where the user's name and session duration are displayed.
* **Software Required**: Same as above.
* **Description**: This task involves creating a session to keep track of user details and session duration.
* **Theory**: The HttpSession object is used in servlets to track a user's session, and the System.currentTimeMillis() function is used to calculate session duration.
* **Code**:

**Session Page (HTML)** (session.html):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Login</title>

</head>

<body>

<h1>Login Page</h1>

<form action="SessionServlet" method="POST">

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

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

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

</form>

</body>

</html>

**Servlet Code (Java)** (SessionServlet.java):

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class SessionServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String name = request.getParameter("name");

HttpSession session = request.getSession();

session.setAttribute("name", name);

session.setAttribute("startTime", System.currentTimeMillis());

response.sendRedirect("welcome.jsp");

}

}

**Welcome Page (JSP)** (welcome.jsp):

<%

HttpSession session = request.getSession();

String name = (String) session.getAttribute("name");

long startTime = (long) session.getAttribute("startTime");

long duration = (System.currentTimeMillis() - startTime) / 1000; // in seconds

%>

<html>

<head>

<title>Welcome page</title>

</head>

<body>

<h1>Hello <%= name %>!</h1>

<p>Session started at: <%= new java.util.Date(startTime) %></p>

<p>Session duration: <%= duration %> seconds.</p>

<a href="logout.jsp">Logout</a>

</body>

</html>

**Logout Page (JSP)** (logout.jsp):

<%

HttpSession sss = request.getSession(false);

String name = null;

if (sss != null) {

// Retrieve the name attribute before invalidating the session

name = (String) sss.getAttribute("name");

sss.invalidate(); // Invalidate the session

}

%>

<html>

<head>

<title>Logout</title>

</head>

<body>

<h1>Thank you <%= name != null ? name : "Guest" %> for visiting!</h1>

</body>

</html>

* **Output**: Displays a welcome message with the session duration, and allows logout.
* **Result**: Session tracking, user greeting, and duration calculation were successfully implemented.

**d) A web application that takes name and age from the HTML page. If age is less than 18, it should show "Hello <name>, You are not authorized to visit the site." Otherwise, it should show "Welcome <name> to the website."**

* **Aim**: To create an application that validates user age before granting access to the website.
* **Software Required**: Same as above.
* **Description**: This task involves validating the user's age input to decide whether they can access the website.
* **Theory**: A simple validation based on age allows or denies access to a website.
* **Code**:

**HTML Form** (ageValidation.html):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Age Validation</title>

</head>

<body>

<h1>Enter your details</h1>

<form action="AgeValidationServlet" method="POST">

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

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

<label for="age">Age:</label>

<input type="number" id="age" name="age"><br><br>

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

</form>

</body>

</html>

**Servlet Code (Java)** (AgeValidationServlet.java):

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class AgeValidationServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String name = request.getParameter("name");

int age = Integer.parseInt(request.getParameter("age"));

response.setContentType("text/html");

PrintWriter out = response.getWriter();

if (age < 18) {

out.println("<h1>Hello " + name + ", You are not authorized to visit the site.</h1>");

} else {

out.println("<h1>Welcome " + name + " to the website!</h1>");

}

}

}

* **Output**: Depending on the age entered, the servlet will display either an authorization message or a welcome message.
* **Result**: Age validation was successfully implemented for user authorization.

**Experiment 6: JSP**

**a) Create a simple JSP page, preferably for the form in experiment 3. Embed JSP in HTML page itself.**

* **Aim**: To embed JSP code in an HTML page to process form data.
* **Software Required**: Apache Tomcat, Java.
* **Description**: This task involves embedding JSP code directly in an HTML page for form processing.
* **Theory**: JSP (Java Server Pages) allows dynamic content generation and is often used for creating web pages that interact with databases or other data sources.
* **Code**:

**JSP Page with Embedded Code** (formWithJSP.jsp):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Form with JSP</title>

</head>

<body>

<h1>User Information</h1>

<form action="" method="POST">

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

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

<label for="age">Age:</label>

<input type="number" id="age" name="age"><br><br>

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

</form>

<%

String name = request.getParameter("name");

String age = request.getParameter("age");

if (name != null && age != null) {

out.println("<h3>Name: " + name + "</h3>");

out.println("<h3>Age: " + age + "</h3>");

}

%>

</body>

</html>

* **Output**: The form data entered by the user will be displayed on the same page.
* **Result**: JSP embedded inside HTML successfully processed and displayed form data.

**b) Separate the JSP and HTML coding in different files and link them together. Add data to the request object.**

* **Aim**: To separate JSP and HTML code into different files and transfer data using the request object.
* **Software Required**: Apache Tomcat, Java.
* **Description**: This task involves separating HTML and JSP content, then using the request object to pass data between them.
* **Theory**: The request object can store attributes that can be shared between different pages or components of a web application. The request.setAttribute() method is used to set data, which can be retrieved by other JSPs using request.getAttribute().
* **Code**:

**HTML Form (index.html)**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Form Submission</title>

</head>

<body>

<h1>Submit Your Details</h1>

<form action="submit.jsp" method="POST">

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

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

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

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

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

</form>

</body>

</html>

**JSP (submit.jsp)**:

<%

String name = request.getParameter("name");

String email = request.getParameter("email");

// Set data in request object

request.setAttribute("userName", name);

request.setAttribute("userEmail", email);

%>

<html>

<head>

<title>Form Submission Result</title>

</head>

<body>

<h1>Submission Result</h1>

<p>Name: <%= request.getAttribute("userName") %></p>

<p>Email: <%= request.getAttribute("userEmail") %></p>

</body>

</html>

* **Output**: After submitting the form, the user's name and email will be displayed on the submit.jsp page.
* **Result**: JSP successfully separated from HTML and data passed using the request object.

**c) At the server end, write code to retrieve contents of request object and show them to the user. Match user input password with predefined password and show “Valid User” or “Invalid User”.**

* **Aim**: To retrieve data from the request object and perform validation.
* **Software Required**: Same as above.
* **Description**: This task involves validating the user's password from the request object and showing appropriate messages.
* **Theory**: In JSP, the request.getParameter() method is used to retrieve form data, and conditionals can be used to validate this data.
* **Code**:

**Login Form (login.html)**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Login</title>

</head>

<body>

<h1>Login</h1>

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

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

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

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

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

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

</form>

</body>

</html>

**Login Validation (login.jsp)**:

<%

String username = request.getParameter("username");

String password = request.getParameter("password");

String predefinedPassword = "admin123"; // predefined password

if (password.equals(predefinedPassword)) {

%>

<h1>Valid User: <%= username %></h1>

<%

} else {

%>

<h1>Invalid User: Incorrect password.</h1>

<%

}

%>

* **Output**: Based on the entered password, the page will show either "Valid User" or "Invalid User."
* **Result**: Password validation using request parameters was successfully implemented.

**d) Modify the above program to use XML file instead of a database.**

* **Aim**: To use XML files for storing user data and retrieve the data using JSP.
* **Software Required**: Same as above.
* **Description**: This task involves using an XML file to store user details instead of a database, and retrieving this data in JSP for validation.
* **Theory**: An XML file can be used to store data in a structured format, and the DocumentBuilderFactory can be used to read the XML file in JSP.
* **Code**:

**XML File (users.xml)**:

<?xml version="1.0" encoding="UTF-8"?>

<users>

<user>

<username>admin</username>

<password>admin123</password>

</user>

<user>

<username>guest</username>

<password>guest123</password>

</user>

</users>

**JSP (login.jsp)**:

<%@ page import="org.w3c.dom.\*, javax.xml.parsers.\*, java.io.\*" %>

<%

String username = request.getParameter("username");

String password = request.getParameter("password");

boolean validUser = false;

// Load the XML file

try {

// Resolve the path to the XML file dynamically

String filePath = application.getRealPath("WEB-INF/users.xml");

File file = new File(filePath);

// Parse the XML file

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document document = builder.parse(file);

// Get all user elements

NodeList userList = document.getElementsByTagName("user");

for (int i = 0; i < userList.getLength(); i++) {

Node user = userList.item(i);

NodeList userDetails = user.getChildNodes();

String storedUsername = "";

String storedPassword = "";

for (int j = 0; j < userDetails.getLength(); j++) {

Node detail = userDetails.item(j);

if (detail.getNodeName().equals("username")) {

storedUsername = detail.getTextContent().trim();

} else if (detail.getNodeName().equals("password")) {

storedPassword = detail.getTextContent().trim();

}

}

// Compare with input data

if (storedUsername.equals(username) && storedPassword.equals(password)) {

validUser = true;

break;

}

}

} catch (Exception e) {

e.printStackTrace();

}

if (validUser) {

%>

<h1>Valid User: <%= username %></h1>

<%

} else {

%>

<h1>Invalid User: Incorrect username or password.</h1>

<%

}

%>

* **Output**: The program will validate the entered username and password using data from the XML file.
* **Result**: XML-based user validation was successfully implemented.

**e) Modify the above program to show the result on the same page using AJAX.**

* **Aim**: To use AJAX to fetch user details based on a userId and display the result without refreshing the page.
* **Software Required**: Apache Tomcat, Java, JavaScript.
* **Description**: This task involves using AJAX to asynchronously send the user ID to a server-side servlet or JSP and fetch the user details, which are displayed on the same page.
* **Theory**: AJAX allows for asynchronous communication between the client and server, making the user experience faster and more interactive. It uses JavaScript to send requests to the server and receive the response without reloading the page.
* **Code**:

**HTML and JavaScript (ajax\_user\_search.html)**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AJAX User Search</title>

<script>

function searchUser() {

var userId = document.getElementById("userId").value;

var xhr = new XMLHttpRequest();

xhr.open("GET", "fetch\_user.jsp?userId=" + userId, true);

xhr.onreadystatechange = function() {

if (xhr.readyState == 4 && xhr.status == 200) {

document.getElementById("result").innerHTML = xhr.responseText;

}

};

xhr.send();

}

</script>

</head>

<body>

<h1>Search User</h1>

<label for="userId">Enter User ID:</label>

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

<button onclick="searchUser()">Search</button>

<div id="result"></div>

</body>

</html>

**JSP (fetch\_user.jsp)**:

<%

String userId = request.getParameter("userId");

// In a real application, this data could come from a database or an XML file.

String userData = "";

if ("1".equals(userId)) {

userData = "User ID: 1<br>Username: user1<br>Email: user1@example.com";

} else if ("2".equals(userId)) {

userData = "User ID: 2<br>Username: user2<br>Email: user2@example.com";

} else {

userData = "User not found!";

}

out.println(userData);

%>

* **Output**: Upon entering a user ID and clicking the "Search" button, the corresponding user data is displayed on the same page without refreshing.
* **Result**: AJAX was successfully used to asynchronously fetch and display user data.

**Experiment 7: JSP & DOM**

**a) A web application name as input and on submit, it should show hello <name>. It shows the start time at the right top corner of the page and provides a logout button. On clicking the logout button, it should go to the logout page that shows "Thank you <name>" with duration of usage.**

* **Aim**: To create a web application that greets the user and tracks the session duration.
* **Software Required**: Same as above.
* **Description**: This task involves creating a session-based greeting with the time of login and logout.
* **Theory**: The HttpSession object stores data between requests, such as the user's name and login time.
* **Code**:

**Login Page (login.jsp)**:

<%

String name = request.getParameter("name");

HttpSession session = request.getSession(true);

session.setAttribute("name", name);

session.setAttribute("loginTime", System.currentTimeMillis());

%>

<html>

<head>

<title>Welcome</title>

</head>

<body>

<h1>Hello, <%= name %></h1>

<p>Session started at: <%= new java.util.Date(session.getAttribute("loginTime")) %></p>

<p><a href="logout.jsp">Logout</a></p>

</body>

</html>

**Logout Page (logout.jsp)**:

<%

HttpSession session = request.getSession(false);

if (session != null) {

String name = (String) session.getAttribute("name");

long loginTime = (long) session.getAttribute("loginTime");

long duration = (System.currentTimeMillis() - loginTime) / 1000;

session.invalidate();

%>

<h1>Thank you, <%= name %></h1>

<p>You spent <%= duration %> seconds on the site.</p>

<%

} else {

%>

<h1>Session expired.</h1>

<%

}

%>

* **Output**: The user is greeted with their name, and upon logout, the duration of their session is displayed.
* **Result**: The application successfully tracks the user's session time and logs them out properly.

**b) A web application that takes name and age from an HTML page. If age is less than 18, it should show “Hello <name>, You are not authorized to visit the site.” Otherwise, it should show “Welcome <name> to the website.”**

* **Aim**: To validate user input (name and age) and show different messages based on the age entered.
* **Software Required**: Apache Tomcat, Java.
* **Description**: This task involves validating the age entered by the user on an HTML form and displaying appropriate messages using JSP.
* **Theory**: Using JSP, we can retrieve the form data, validate it, and dynamically show results based on the entered age.
* **Code**:

**HTML Form (age\_validation.html)**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Age Validation</title>

</head>

<body>

<h1>Age Validation</h1>

<form action="age\_validation.jsp" method="POST">

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

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

<label for="age">Age:</label>

<input type="number" id="age" name="age"><br><br>

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

</form>

</body>

</html>

**JSP (age\_validation.jsp)**:

<%

String name = request.getParameter("name");

int age = Integer.parseInt(request.getParameter("age"));

if (age < 18) {

%>

<h1>Hello <%= name %>, You are not authorized to visit the site.</h1>

<%

} else {

%>

<h1>Welcome <%= name %> to the website!</h1>

<%

}

%>

* **Output**: The JSP page will display either "You are not authorized" or "Welcome" based on the age entered by the user.
* **Result**: Age validation was successfully implemented using JSP and the user was greeted appropriately.

**a) Create an XML Document containing 10 users' information. Write a Java program that takes userId as input and returns user details from XML file by using DOM Parser.**

* **Aim**: To retrieve user details from an XML file based on user input using the DOM parser.
* **Software Required**: Java, Apache Tomcat.
* **Description**: This task involves using the DOM parser to read XML data and return details based on the user’s ID.
* **Theory**: DOM (Document Object Model) is an in-memory representation of the XML document, which allows navigation and manipulation of XML elements.
* **Code**:

**XML File (users.xml)**:

<?xml version="1.0" encoding="UTF-8"?>

<users>

<user id="1">

<username>user1</username>

<email>user1@example.com</email>

</user>

<!-- Add 9 more users here -->

</users>

**Java Program (DOMParserExample.java)**:

import org.w3c.dom.\*;

import javax.xml.parsers.\*;

import java.io.\*;

public class DOMParserExample {

public static void main(String[] args) throws Exception {

File file = new File("users.xml");

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(file);

String userId = "1"; // Assume this ID is entered by the user

NodeList userList = doc.getElementsByTagName("user");

for (int i = 0; i < userList.getLength(); i++) {

Node user = userList.item(i);

Element element = (Element) user;

String id = element.getAttribute("id");

if (id.equals(userId)) {

String username = element.getElementsByTagName("username").item(0).getTextContent();

String email = element.getElementsByTagName("email").item(0).getTextContent();

System.out.println("User ID: " + userId);

System.out.println("Username: " + username);

System.out.println("Email: " + email);

}

}

}

}

* **Output**: The program will display the user details based on the entered user ID.
* **Result**: User details were successfully retrieved from the XML file using the DOM parser.

**b) Create an XML Document containing 10 users' information. Write a Java program that takes userId as input and returns user details from the XML file by using SAX Parser.**

* **Aim**: To retrieve user details using the SAX parser from an XML file based on user input.
* **Software Required**: Same as above.
* **Description**: This task involves using the SAX parser, an event-driven approach to reading XML data, to fetch user information.
* **Theory**: SAX (Simple API for XML) is an event-driven XML parser. Unlike DOM, it doesn't load the entire document into memory but rather triggers events as it processes the document.
* **Code**:

**Java Program (SAXParserExample.java)**:

import org.xml.sax.\*;

import org.xml.sax.helpers.\*;

import javax.xml.parsers.\*;

import java.io.\*;

public class SAXParserExample extends DefaultHandler {

boolean usernameFlag = false;

boolean emailFlag = false;

String userId = "";

String username = "";

String email = "";

public static void main(String[] args) throws Exception {

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

SAXParserExample handler = new SAXParserExample();

saxParser.parse(new File("users.xml"), handler);

}

@Override

public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {

if (qName.equalsIgnoreCase("user")) {

userId = attributes.getValue("id");

}

if (qName.equalsIgnoreCase("username")) {

usernameFlag = true;

}

if (qName.equalsIgnoreCase("email")) {

emailFlag = true;

}

}

@Override

public void characters(char[] ch, int start, int length) throws SAXException {

if (usernameFlag) {

username = new String(ch, start, length);

usernameFlag = false;

}

if (emailFlag) {

email = new String(ch, start, length);

emailFlag = false;

}

}

@Override

public void endElement(String uri, String localName, String qName) throws SAXException {

if (qName.equalsIgnoreCase("user")) {

if (userId.equals("1")) { // Assume userID 1 is queried

System.out.println("User ID: " + userId);

System.out.println("Username: " + username);

System.out.println("Email: " + email);

}

}

}

}

* **Output**: The program will display the user details based on the entered user ID using SAX parsing.
* **Result**: SAX parser successfully retrieved user details from the XML file.