Exercise: 1(i)

Create a simple HTML file to demonstrate the use of different tags

---

Aim:

To create a simple HTML file demonstrating the use of different HTML tags such as headings, paragraphs, lists, images, links, tables, and formatting tags.

Algorithm:

Step1: Start the HTML document with < !DOCTYPE html > declaration.

Step2: Open the <html> tag and define the <head> section with a <title>.

Step3: In the <body> section, demonstrate different HTML tags:

- Use <h1> to <h6> for headings.
- Use <p> for paragraphs.
- Use <b>, <i>, <u> for text formatting.
- Use <a> for hyperlinks.
- Use <img> to insert an image.
- Use <ul> and <ol> for unordered and ordered lists.
- Use <table> with rows and columns.

Step4: Save the file as demo.html.

Step5: Open the file in a web browser to view the output.

Program:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Tags Demonstration</title>
</head>
<body>
<!-- Headings -->
<h1>This is Heading 1</h1>
<h2>This is Heading 2</h2>
<h3>This is Heading 3</h3>

<!-- Paragraph with formatting -->
<p>
This is a <b>bold</b> word,
this is an <i>italic</i> word,
and this is an <u>underlined</u> word.
</p>
```

```
<!-- Link -->
<p>Visit <a href="https://www.google.com" target="_blank">Google</a></p>

<!-- Image -->
<img src="https://via.placeholder.com/150" alt="Sample Image">

<!-- Lists -->
<h3>Unordered List</h3>
<ul>
<li>HTML</li>
<li>CSS</li>
<li>JavaScript</li>
</ul>

<h3>Ordered List</h3>
<ol>
<li>First Step</li>
<li>Second Step</li>
<li>Third Step</li>
</ol>

<!-- Table -->
<h3>Sample Table</h3>
<table border="1" cellpadding="5" cellspacing="0">
<tr>
<th>Name</th>
<th>Course</th>
<th>Marks</th>
</tr>
<tr>
<td>Hari</td>
<td>HTML</td>
<td>90</td>
</tr>
<tr>
<td>John</td>
<td>CSS</td>
<td>85</td>
</tr>
</table>
</body>
</html>
```

OUTPUT:

This is Heading 1

This is Heading 2

This is Heading 3

This is a **bold** word, this is an *italic* word, and this is an underlined word.

Visit Google

Unordered List

- HTML
- CSS
- JavaScript

Ordered List

1. First Step
2. Second Step
3. Third Step

Sample Table

Name	Course	Marks
Hari	HTML	90
John	CSS	85

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exersice: 1(ii)

Write an HTML page that contains a selection box with a list of 5 countries. When the user selects a country, its capital should be printed next to the list. Add CSS to customize the properties of the font of the capital.

Aim:

To create an HTML page with a selection box of 5 countries. When a user selects a country, its capital city is displayed next to the list, with customized font styling using CSS.

Algorithm:

Step1: Start the HTML document with < !DOCTYPE html >.

Step2: Create a <select> element containing 5 countries as < option >.

Step3: Add an empty or <p> element where the capital will be displayed.

Step4: Write a JavaScript function that detects the selected country and displays its corresponding capital.

Step5: Apply CSS styles to the capital (font size, color, style).

Step6: Save the file as country-capital.html and run it in a browser.

Program:

```
<!DOCTYPE html>
<html>
<head>
<title>Country and Capital</title>
<style>
/* CSS for capital text */
#capital {
font-size: 20px;
font-weight: bold;
color: darkblue;
font-family: Arial, sans-serif;
margin-left: 20px;
}
</style>
</head>
<body>
<h2>Select a Country to See Its Capital</h2>
```

```
<!-- Dropdown List -->
<select id="country" onchange="showCapital()">
<option value="">--Select a Country--</option>
<option value="India">India</option>
<option value="USA">USA</option>
<option value="Japan">Japan</option>
<option value="France">France</option>
<option value="Australia">Australia</option>
</select>

<!-- Display Capital -->
<span id="capital"></span>

<!-- JavaScript -->
<script>
function showCapital() {
var country = document.getElementById("country").value;
var capital = "";

switch (country) {
case "India":
capital = "New Delhi";
break;
case "USA":
capital = "Washington, D.C.";
break;
case "Japan":
capital = "Tokyo";
break;
case "France":
capital = "Paris";
break;
case "Australia":
capital = "Canberra";
break;
}

document.getElementById("capital").textContent = capital;
}
</script>
</body>
</html>
```

.

OUTPUT:

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 2(a)

Program using XML Schema.

Aim:

To design an XML document with student information and validate it using XML Schema (XSD). Then, create an HTML page to display the data.

Algorithm:

Step1: Create an XML document with student details such as Roll No, Name, and Department.

Step2: Create an XML Schema (XSD) to define the structure and data types of the XML.

Step3: Link the XML file with the schema using xsi: no Namespace Schema Location.

Step4: Create an HTML page to display the student data in a readable format.

Step5: Open the HTML page in a browser.

Program:

Step1:

student.xml

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

<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="student.xsd">

<rollno>101</rollno>

<name>Hari Haran</name>

<department>Computer Science</department>

</student>
```

Step2: student.xsd

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="student">

<xs:complexType>

<xs:sequence>

<xs:element name="rollno" type="xs:integer"/>

<xs:element name="name" type="xs:string"/>

<xs:element name="department" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>
```

Step3: Index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Student Information</title>
<style>
body {
font-family: Arial, sans-serif;
}
table {
border-collapse: collapse;
width: 50%;
margin: 20px auto;
}
th, td {
border: 1px solid black;
padding: 8px;
text-align: center;
}
th {
background-color: lightblue;
```

```
}
</style>
</head>
<body>
<h2 style="text-align:center;">Student Information from XML</h2>
<table>
<tr>
<th>Roll No</th>
<th>Name</th>
<th>Department</th>
</tr>
<tr>
<td>101</td>
<td>Hari Haran</td>
<td>Computer Science</td>
</tr>
</table>
</body>
</html>
```

OUTPUT:

Student Information from XML

Roll No	Name	Course	Department
122012012797	Hari Haran	Web Technology	Computer Science

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 2(b)

Write a program using XSLT/XSL and ALAX

Aim:

To write a program using XSLT/XSL and AJAX

Algorithm:

Step1: Start.

Step2: Create an XML file (students.xml) containing structured data (e.g., student name, roll number, marks).

Step3: Create an XSL file (students.xsl) that defines how XML data should be displayed (e.g., table format).

Step4: Create an HTML page (index.html) with a button to trigger data loading.

Step5: Write JavaScript using AJAX to:

- Request both XML and XSL files from the server.

Step6: Apply the XSL transformation using the browser's XSLTProcessor.

Step7: Display the transformed HTML inside a element.

Step8: Test by running the HTML file in a browser.

Step9: End.

Program:

Step1: Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>AJAX XSLT Example</title>
<style>
/* Simple styles for the transformed content */
table {
width: 100%;
border-collapse: collapse;
}
```

```
table, th, td {
border: 1px solid black;
}

th, td {
padding: 8px;
text-align: left;
}
</style>
</head>
<body>
<h2>AJAX with XSLT Example</h2>
<div id="output"></div>

<script>
// Function to load the XML and XSL files, and perform the XSLT transformation
function loadXMLDoc(filename) {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
if (this.readyState === 4 && this.status === 200) {
if (filename.endsWith(".xml")) {
var xmlDoc = this.responseXML;
// Load the XSLT file and transform the XML document
loadXSLT(xmlDoc);
}
}
};
xhttp.open("GET", filename, true);
xhttp.send();
}

// Function to load the XSLT file and transform the XML data
function loadXSLT(xml) {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
if (this.readyState === 4 && this.status === 200) {
var xslDoc = this.responseXML;
transformXML(xml, xslDoc);
}
};
xhttp.open("GET", "style.xsl", true);
xhttp.send();
}

// Function to perform the XSLT transformation and display the result
function transformXML(xml, xsl) {
// Check for browser support for XSLT
if (window.ActiveXObject || "ActiveXObject" in window) {
// For IE
```

```
var ex = xml.transformNode(xsl);
document.getElementById("output").innerHTML = ex;
} else if (document.implementation && document.implementation.createDocument) {
// For modern browsers (Chrome, Firefox, etc.)
var xsltProcessor = new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
var resultDocument = xsltProcessor.transformToFragment(xml, document);
document.getElementById("output").appendChild(resultDocument);
}
}

// Load XML data on page load
window.onload = function () {
loadXMLDoc("data.xml");
};
</script>
</body>
</html>
```

Step2: Data.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
<book>
<title>Learning XSLT</title>
<author>John Doe</author>
<year>2020</year>
<price>39.99</price>
</book>
<book>
<title>JavaScript Fundamentals</title>
<author>Jane Smith</author>
<year>2021</year>
<price>29.99</price>
</book>
<book>
<title>Advanced CSS</title>
<author>Robert Brown</author>
<year>2019</year>
<price>49.99</price>
</book>
</catalog>
```

Step3:Data.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>Book Catalog</h2>
```

```
<table>
<tr>
<th>Title</th>
<th>Author</th>
<th>Year</th>
<th>Price</th>
</tr>
<xsl:for-each select="catalog/book">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="author"/></td>
<td><xsl:value-of select="year"/></td>
<td><xsl:value-of select="price"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

OUTPUT:

AJAX with XSLT Example

Book Catalog

Title	Author	Year	Price
Learning XSLT	John Doe	2020	39.99
JavaScript Fundamentals	Jane Smith	2021	29.99
Advanced CSS	Robert Brown	2019	49.99

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 2(c)

Develop Angular js

Aim:

To create a simple AngularJS application that takes user input and dynamically displays a greeting message using data binding.

Algorithm:

Step1: Start the program.

Step2: Include Angular JS library in the HTML file.

Step3: Define an Angular JS application module named "my App".

Step4: Create a controller "my Ctrl" for the application.

Step5: Initialize a variable name in the controller's scope.

Step6: Bind an input field with the model name using ng-model.

Step7: Display the value of name dynamically using Angular JS expression {{name}}.

Step8: When the user types in the input field, the greeting message updates automatically.

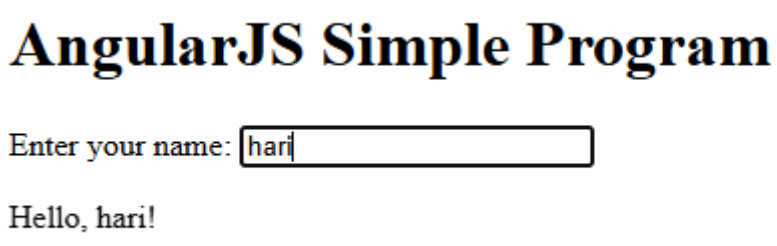
Step9: End the program.

Program:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
<title>AngularJS Example</title>
</head>
<body ng-controller="myCtrl">

<h1>AngularJS Simple Program</h1>
<p>Enter your name: <input type="text" ng-model="name"></p>
<p>Hello, {{name}}!</p>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
$scope.name = "";
```

```
});
</script>

</body>
</html>
```

OUTPUT:



Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 3(a)

Write an HTML page including JavaScript that takes a set of integer numbers from the user, sorts them in descending order.

Aim:

To create an HTML page with JavaScript that takes a set of integer numbers entered by the user, sorts them in descending order, and displays the sorted list.

Algorithm:

Step1: Start

Step2: Display a text input box to the user to enter integer numbers separated by commas.

Step3: When the user clicks the "Sort" button:
a. Read the input string from the text box.
b. Split the input string by commas to get an array of strings.
c. Trim each string and convert it to an integer.
d. Check if all inputs are valid integers; if not, show an error message and stop.
e. Sort the array of integers in descending order (largest to smallest).

Step4: Display the sorted array on the webpage.

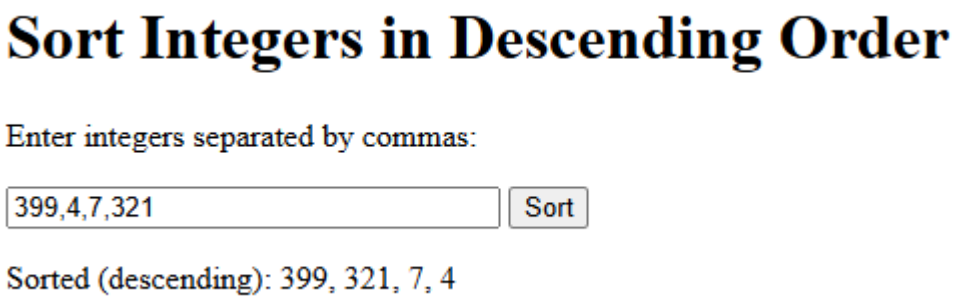
Step5: End

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Sort Integers in Descending Order</title>
<script>
function sortDescending() {
// Get the input string
let input = document.getElementById("numbers").value;

// Convert input string into an array of integers
let numArray = input.split(',').map(num => parseInt(num.trim(), 10));

// Check for invalid inputs
if (numArray.some(isNaN)) {
alert("Please enter a valid set of integers separated by commas.");
```

```
return;
}

// Sort array in descending order
numArray.sort((a, b) => b - a);

// Display the sorted numbers
document.getElementById("result").textContent = "Sorted (descending): " + numArray.join(',
');
}
</script>
</head>
<body>
<h1>Sort Integers in Descending Order</h1>
<p>Enter integers separated by commas:</p>
<input type="text" id="numbers" placeholder="e.g. 5, 3, 8, 1" size="30" />
<button onclick="sortDescending()">Sort</button>
<p id="result"></p>
</body>
</html>
```

OUTPUT:


Sort Integers in Descending Order
Enter integers separated by commas:
399,4,7,321
Sort
Sorted (descending): 399, 321, 7, 4

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 3(b)

Client side scripts for validating web form controls and creating events using Java Script

---

Aim:

To create a web form that validates user input on the client side using JavaScript, ensuring the form fields are correctly filled before submission, and to handle form submission events.

Algorithm:

Step1: Start

Step2: Create an HTML form with input fields (e.g., Name and Email) and a submit button.

Step3: Attach a JavaScript function to the form's `onsubmit` event.

Step4: When the form is submitted, the JavaScript function is triggered.

Step5: Inside the function, prevent the default form submission to validate inputs first.

Step6: Retrieve the values entered in the form fields.

Step7: Check if the "Name" field is empty.

- If empty, alert the user to fill in the name and stop submission.

Step8: Validate the "Email" field using a regular expression pattern to check its format.

- If the email is invalid, alert the user and stop submission.

Step9: If all validations pass, alert the user that the form is submitted successfully.

Step10: Allow form submission or handle the data as needed.

Step11: End.

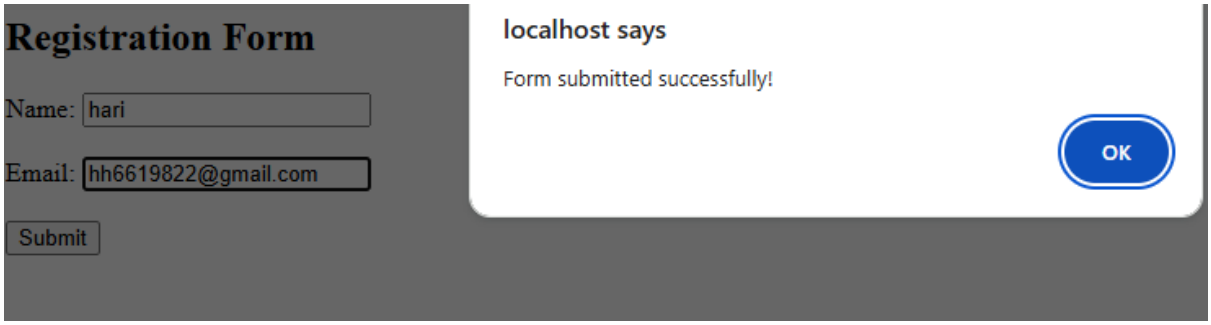
Program:

```
<!DOCTYPE html>

<html>

<head>
```

```
<title>Form Validation Example</title>
<script>
function validateForm(event) {
event.preventDefault(); // Prevent form from submitting
const name = document.forms["myForm"]["name"].value;
const email = document.forms["myForm"]["email"].value;
if (name === "") {
alert("Name must be filled out");
return false;
}
const emailPattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;
if (!email.match(emailPattern)) {
alert("Please enter a valid email address");
return false;
}
alert("Form submitted successfully!");
// You can submit the form here if needed using AJAX or:
// document.forms["myForm"].submit();
}
</script>
</head>
<body>
<h2>Registration Form</h2>
<form name="myForm" onsubmit="validateForm(event)">
Name: <input type="text" name="name"><br><br>
```

```
Email: <input type="text" name="email"><br><br>

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

</form>

</body>

</html>
```

OUTPUT:


Registration Form
Name: hari
Email: hh6619822@gmail.com
Submit
localhost says
Form submitted successfully!
OK

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 4(a)

Write programs in Java Servlet to do the following: a Set the URL of another server

---

Aim:

To write programs in Java Servlet to Set the URL of another server

Algorithm:

Step1: Start the program.

Step2: Initialize servlet response content type to "text/html".

Step3: Create a Print Writer object to send output to the client.

Step4: Print initial HTML and heading indicating servlet start and purpose.

Step5: Fetch the integer value of the URL parameter named "calledCount" from the request:

Step6: Display the value of "calledCount" received in the request:

Step7: Increment the "called Count" by 1.

Step8: Construct a new URL to the same servlet, appending the updated "calledCount" as a query parameter and encode it using response.encode URL() for session handling.

Step9: Print a hyperlink labeled "Click to reload" which points to the new URL with the updated parameter.

Step10: End

Program:

Index html:

```
<!DOCTYPE html>
<html>
<head>
<title>Set Server URL</title>
</head>
<body>
<h2>Enter Server URL</h2>
<form action="SetServerURL" method="get">
<label for="url">Server URL:</label>
```

```
<input type="text" id="url" name="url" placeholder="https://www.google.com" required>
<input type="submit" value="Go">
</form>
</body>
</html>
```

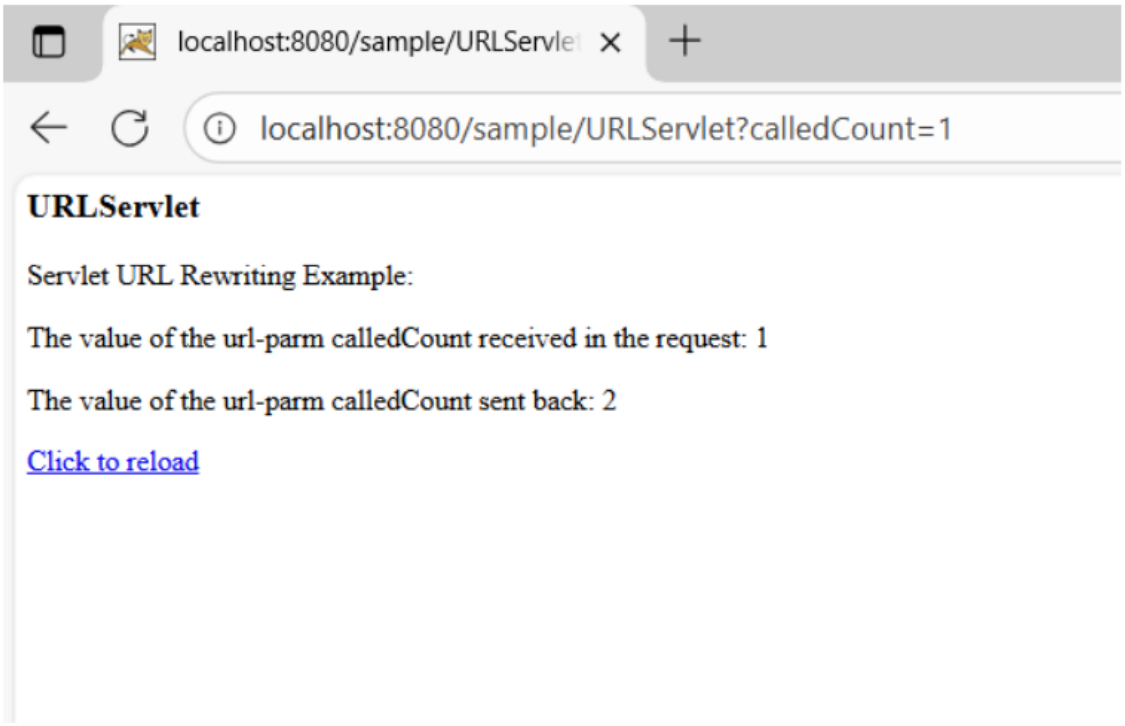
Java Servlet (SetServerURL.java):

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SetServerURL extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

// Get the URL entered by user
String serverURL = request.getParameter("url");

if (serverURL != null && !serverURL.trim().isEmpty()) {
// Redirect to the given URL
response.sendRedirect(serverURL);
} else {
// If no URL provided, show an error message
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><body>");
out.println("<h3 style='color:red;'>Invalid URL! Please enter a valid one.</h3>");
out.println("</body></html>");
}
}
}
```

web.xml Configuration (inside WEB-INF folder:

```
<web-app>
<servlet>
<servlet-name>SetServerURL</servlet-name>
<servlet-class>SetServerURL</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>SetServerURL</servlet-name>
<url-pattern>/SetServerURL</url-pattern>
</servlet-mapping>
</web-app>
```

OUTPUT:


localhost:8080/sample/URLServle
localhost:8080/sample/URLServlet?calledCount=1
URLServlet
Servlet URL Rewriting Example:
The value of the url-parm calledCount received in the request: 1
The value of the url-parm calledCount sent back: 2
Click to reload

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 4(b)

Download the homepage of the server

Aim:

To write a servlet program to Download the homepage of the server.

Algorithm:

Step1: Start.

Step2: Initialize target URL with the webpage address whose homepage is to be downloaded. Example: http://www.gmail.com.

Step3: Create a URL object for the target URL.

Step4: Open a BufferedReader to read the webpage content from the URL input stream.

Step5: Set the response content type to "text/html" to indicate the response contains HTML content.

Step6: Read the webpage content line by line using the BufferedReader.

Step7: After reading all content, close the BufferedReader.

Step8: End.

Program:

Index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Download Server Homepage</title>
</head>
<body>
<h2>Download Homepage of Another Server</h2>
<form action="DownloadServerPage" method="get">
<label for="url">Enter Server URL:</label>
<input type="text" id="url" name="url" placeholder="https://www.google.com" size="50" required>
<input type="submit" value="Download">
```

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

Java Servlet.java

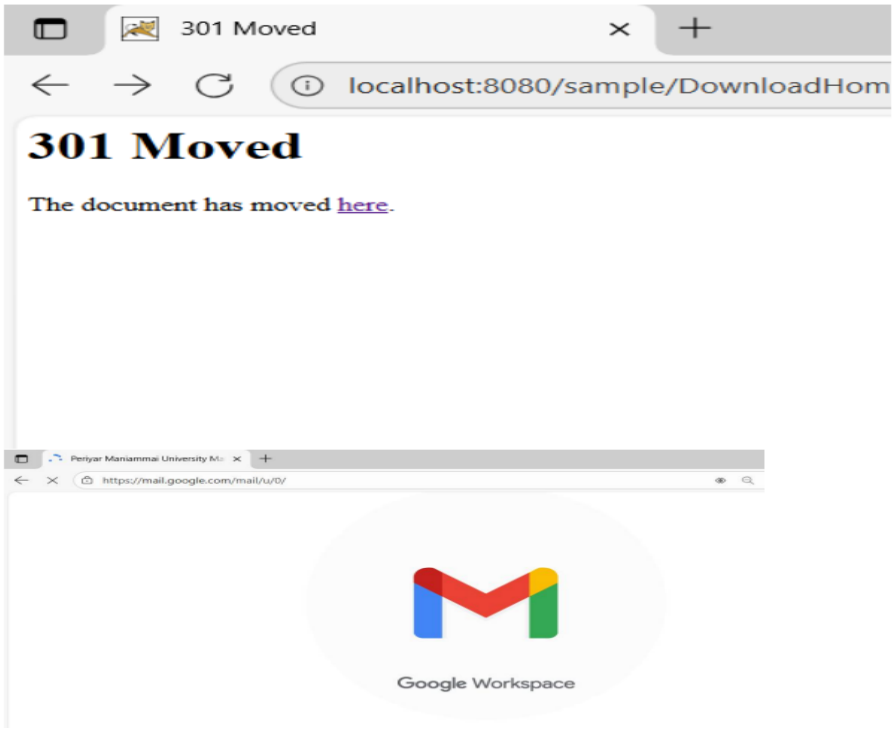
```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DownloadServerPage extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
String serverURL = request.getParameter("url");
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><body>");
out.println("<h2>Downloaded Homepage Content:</h2><hr>");
if (serverURL == null || serverURL.trim().isEmpty()) {
out.println("<p style='color:red;'>Please enter a valid URL.</p>");
} else {
try {
// Connect to the given server URL
URL url = new URL(serverURL);
BufferedReader br = new BufferedReader(new InputStreamReader(url.openStream()));
String line;
out.println("<pre>");
while ((line = br.readLine()) != null) {
out.println(line);
}
out.println("</pre>");
br.close();
} catch (Exception e) {
out.println("<p style='color:red;'>Error downloading page: " + e.getMessage() + "</p>");
}
}
out.println("</body></html>");
}
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>DownloadServerPage</servlet-name>
<servlet-class>DownloadServerPage</servlet-class>
```

```
</servlet>

<servlet-mapping>
<servlet-name>DownloadServerPage</servlet-name>
<url-pattern>/DownloadServerPage</url-pattern>
</servlet-mapping>
</web-app>
```

OUTPUT:


301 Moved
localhost:8080/sample/DownloadHom
301 Moved
The document has moved here.
https://mail.google.com/mail/u/0/
Google Workspace

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 4(c)

Display the contents of home page with date, content type, and expiration date. Last modified and length of the home page

---

Aim:

To write a servlet program to Display the contents of home page with date, content type, and Expiration date. Last modified and length of the home page.

Algorithm:

Step1: Start.

Step2: Define the target URL for which homepage content and metadata are to be fetched. Example: http://www.gmail.com.

Step3: Create a URL object to connect to the target URL.

Step4: Open a URL connection (URLConnection) to access HTTP headers and content.

Step5: Retrieve metadata from the connection:
- Date of the page (getDate())
- Expiration date (getExpiration())
- Last Modified date (getLastModified())
- Content Length (getContentLength())
- Content Type (getContentType())

Step6: Format the retrieved dates to a readable string format using Simple Date Format.

Step7: Set the servlet response content type to "text/html".

Step8: Write HTML output to the response to display metadata:
- Display Date, or "Not available" if missing.
- Display Content Type.
- Display Expiration Date, or "Not available" if missing.
- Display Last Modified Date, or "Not available" if missing.
- Display Content Length in bytes.

Step9: Read the homepage content line by line from the connection input stream using Buffered Reader.

Step10: Write each line of homepage content to the servlet response output.

Step11: End.

Program:

Index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Website Info Viewer</title>
</head>
<body>
<h2>Get Website Details and Homepage Content</h2>
<form action="WebsiteInfoServlet" method="get">
<label for="url">Enter Website URL:</label>
<input type="text" id="url" name="url" placeholder="https://www.google.com" size="50" required>
<input type="submit" value="View Details">
</form>
</body>
</html>
```

WebsiteInfoServlet.java

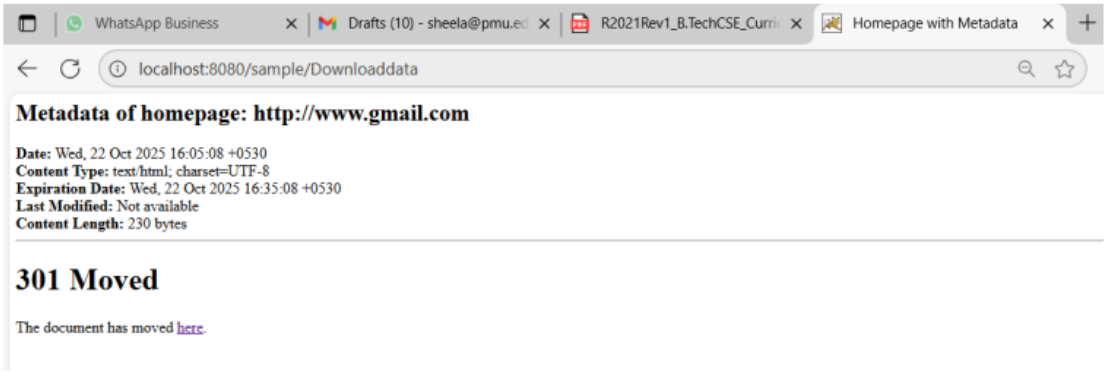
```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.text.SimpleDateFormat;
import java.util.Date;
public class WebsiteInfoServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
String serverURL = request.getParameter("url");
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><body>");
out.println("<h2>Website Information</h2><hr>");
if (serverURL == null || serverURL.trim().isEmpty()) {
out.println("<p style='color:red;'>Please enter a valid URL.</p>");
} else {
try {
URL url = new URL(serverURL);
URLConnection connection = url.openConnection();
// Retrieve header details
long date = connection.getDate();
long expiration = connection.getExpiration();
long lastModified = connection.getLastModified();
String contentType = connection.getContentType();
int contentLength = connection.getContentLength();
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
out.println("<table border='1' cellpadding='5'>");
```

```
out.println("<tr><th>Property</th><th>Value</th></tr>");
out.println("<tr><td>Date</td><td>" + (date == 0 ? "N/A" : sdf.format(new Date(date))) +
"</td></tr>");
out.println("<tr><td>Content Type</td><td>" + contentType + "</td></tr>");
out.println("<tr><td>Expiration Date</td><td>" + (expiration == 0 ? "N/A" : sdf.format(new
Date(expiration))) + "</td></tr>");
out.println("<tr><td>Last Modified</td><td>" + (lastModified == 0 ? "N/A" : sdf.format(new
Date(lastModified))) + "</td></tr>");
out.println("<tr><td>Content Length</td><td>" + (contentLength == -1 ? "Unknown" :
contentLength + " bytes") + "</td></tr>");
out.println("</table><hr>");
// Display homepage content
out.println("<h3>Homepage Content:</h3>");
BufferedReader br = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
String line;
out.println("<pre>");
while ((line = br.readLine()) != null) {
out.println(line);
}
out.println("</pre>");
br.close();
} catch (Exception e) {
out.println("<p style='color:red;'>Error: " + e.getMessage() + "</p>");
}
}
out.println("</body></html>");
}
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>WebsiteInfoServlet</servlet-name>
<servlet-class>WebsiteInfoServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>WebsiteInfoServlet</servlet-name>
<url-pattern>/WebsiteInfoServlet</url-pattern>
</servlet-mapping>
</web-app>
```

OUTPUT:


WhatsApp Business
Drafts (10) - sheela@pmu.ec
R2021Rev1_B.TechCSE_Curri
Homepage with Metadata
localhost:8080/sample/Downloaddata
Metadata of homepage: http://www.gmail.com
Date: Wed, 22 Oct 2025 16:05:08 +0530
Content Type: text/html; charset=UTF-8
Expiration Date: Wed, 22 Oct 2025 16:35:08 +0530
Last Modified: Not available
Content Length: 230 bytes
301 Moved
The document has moved here.

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Expt. No. 4.4
Date 23/10/25

Web application Development Using Node.js

Page No. 105

Aim:-

To write a web application developing using node.js

Algorithm:-

Step 1 - Install the node.js and verify installation node.-version and npm.-version

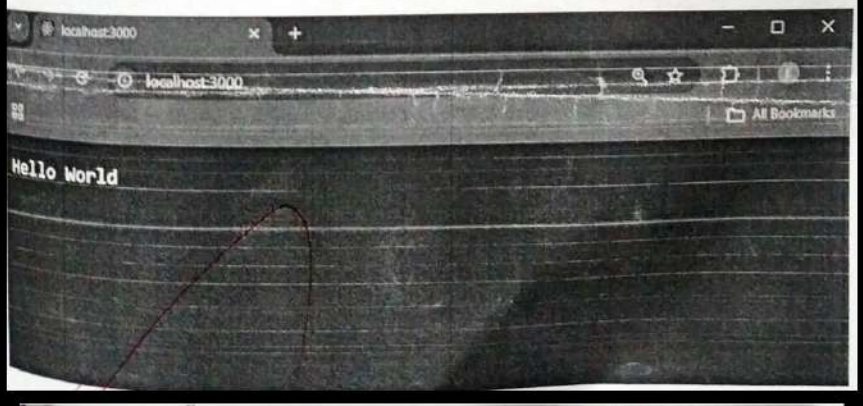
Step 2 - Create a project

Step 3:- Initialize project (create package.json) with default

Step 4:- Track file server.js and download HTTP server

Step 5:- Run the application and verify in the browser port : localhost : 3000.

```
Microsoft Windows [Version 10.0.26200.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\eshwa>cd C:\Users\eshwa\Desktop\Simplewebapp

C:\Users\eshwa\Desktop\Simplewebapp>node -v
v24.11.0

C:\Users\eshwa\Desktop\Simplewebapp>npm init -y
Wrote to C:\Users\eshwa\Desktop\Simplewebapp\package.json:

{
  "name": "simplewebapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}


C:\Users\eshwa\Desktop\Simplewebapp>npm install express

added 68 packages, and audited 69 packages in 5s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\eshwa\Desktop\Simplewebapp>node index.js
Server running at http://localhost:3000/
```



Page No. 107

Install node.js

1) open command prompt

2) After installation, set you folder path by change directory Command
→
mkdir Simple webApp
cd Simple Web app
C:\user\ADMIN\ Simple app >

3) Initialize your project with NPm
→ npm init

4) Install Express in your project
npm install express - save
After running this command, node_modules folder is created in your project.

5) Create a inder.js and run the program.

Page No. 109

Program

```
Coast http = require ('http')
Coast Server = http.Create server ( (req.rs) => {
        rewrite head (200, {Counter_type = "text/plain"});
    3);
Server : step (3000, local host ' :1) => {
    console.log ("Server running at http:// Local host:
                3000 \n");
```

Rubrics :-

Criteria	Max marks	mark awarded
Aim/algorithm	20	20
program	20	20
Execution	20	20
Output	20	10
Viva voce	20	10
Total	100	80

Result :-

Thus the creation of web application development using node.js have been executed and verified successfully

Exercise: 5(i)

A JSP program for simple user authentication (username, password)

Aim:

To create a simple user authentication program using JSP. The program will validate a user's credentials (username and password) against predefined, hardcoded values.

Algorithm:

Step1: Create index.html: Build a login form with username and password fields. Set the form method to post and the action to login Action.jsp.

Step2: Create loginAction.jsp.

Step3: Get username and password from the request using request.getParameter().

Step4: Define the correct login details (e.g., admin, pass123).

Step5: Use an if statement (.equals()) to check if the submitted details match the correct ones.

Step6: Display a "Welcome" message on success.

Step7: Display an "Invalid credentials" error and a "try again" link on failure.

Program:

Index.html (The Login Form):

```
<html>
<head>
<title>Login Page</title>
</head>
<body>
<h1 align="center">Login</h1>
<form name="form1" action="login.jsp" method="post">
<table>
<tr>
<td><b>UserName:</b></td>
<td><input type="text" name="username"></td>
</tr>
<tr>
<td><b>Password:</b></td>
<td><input type="password" name="password"></td>
</tr>
<tr>
<td><input type="submit" value="Submit"></td>
<td><input type="reset" value="Cancel"></td>
</tr>
```

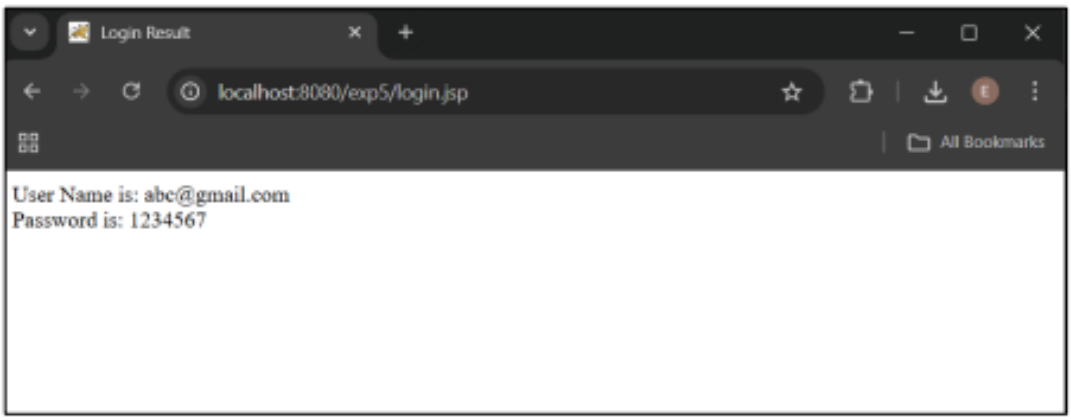
```
</table>
</form>
</body>
</html>
```

LoginAction.jsp (The Authentication Logic):

```
<%@ page import="javax.io.*"%>
<%@ page import="javax.servlet.*"%>
<%@ page import="javax.servlet.http.*"%>
<%@ page import="javax.servlet.jsp.*"%>
<%@ page import="javax.io.*"%>
<%@ page import="javax.servlet.*"%>
<%@ page import="javax.servlet.http.*"%>
<%@ page import="javax.servlet.jsp.*"%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head><title>Login Result</title></head>
<body>
User Name is: <%= request.getParameter("t1") %><br>
Password is: <%= request.getParameter("t2") %>
</body>
</html>
```

OUTPUT:


Login Page
localhost:8080/exp5/index.html
All Bookmarks
Login
UserName:
abc@gmail.com
Password:
Submit
Cancel


Login Result
localhost:8080/exp5/login.jsp
All Bookmarks
User Name is: abc@gmail.com
Password is: 1234567

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 5(ii)

A **w**eb application using JSP with JDBC.

Aim:

To develop a dynamic web application using JSP and JDBC that performs CRUD operations on a student database.

Algorithm:

Step1: Start the server using Apache Tomcat.

Step2: Design the database with a table named students containing fields: id, name, email, and course.

Step3: Establish JDBC connection using Driver Manager and appropriate JDBC driver.

Step4: Create JSP pages for:

- Adding a student
- Viewing all students
- Updating student details
- Deleting a student

Step5: Implement Java classes/Servlets to handle backend logic and database operations.

Step6: Deploy the application on Tomcat and test each functionality.

Step7: Display appropriate messages for success or failure of operations.

Program:

DBUtil.java:

```
Import javapackagecom.example.util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBUtil {
private static final String URL = "jdbc:mysql://localhost:3306/companydb";
private static final String USER = "root"; // XAMPP default
private static final String PASSWORD = ""; // XAMPP default

// Static block to load the MySQL driver once
```

```
static {
try {
// Register the JDBC Driver (optional for modern JDBC 4.0+)
Class.forName("com.mysql.cj.jdbc.Driver");
} catch (ClassNotFoundException e) {
e.printStackTrace();
throw new RuntimeException("Failed to load MySQL JDBC Driver.");
}
}

public static Connection getConnection() throws SQLException {
return DriverManager.getConnection(URL, USER, PASSWORD);
}
}
```

Employee.java (Model):

Create a simple model class to hold employee data.

```
package com.example.servlet;

public class Employee {
private int id;
private String name;
private String email;

public Employee(int id, String name, String email) {
this.id = id;
this.name = name;
this.email = email;
}

// Getters and Setters (omitted for brevity, but needed in a full project)
public int getId() { return id; }
public String getName() { return name; }
public String getEmail() { return email; }
// ... other methods
}
```

EmployeeServlet.java (Controller)

Java

```
package com.example.servlet;

import com.example.util.DBUtil;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
```

```
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

// NOTE: Use @WebServlet annotation OR configure in web.xml. We'll use web.xml.

public class EmployeeServlet extends HttpServlet {
private static final long serialVersionUID = 1L;

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

List<Employee> employeeList = new ArrayList<>();

try (Connection connection = DBUtil.getConnection();
PreparedStatement preparedStatement = connection.prepareStatement("SELECT id, name, email
FROM Employee");
ResultSet rs = preparedStatement.executeQuery()) {

while (rs.next()) {
int id = rs.getInt("id");
String name = rs.getString("name");
String email = rs.getString("email");
employeeList.add(new Employee(id, name, email));
}

// Set the list as a request attribute
request.setAttribute("employees", employeeList);

// Forward to the JSP page
request.getRequestDispatcher("/employees.jsp").forward(request, response);

} catch (SQLException e) {
e.printStackTrace();
response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "Database error:
" + e.getMessage());
}
}
}
```

Employees.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<%@ page import="java.util.List" %>
<%@ page import="com.example.servlet.Employee" %>
<!DOCTYPE html>
<html>
<head>
<title>Employee List</title>
<style>
table, th, td { border: 1px solid black; border-collapse: collapse; padding: 8px; }
</style>
```

```
</head>
<body>
<h1>Employee List</h1>
<%
// Retrieve the employee list from the request attribute set by the servlet
List<Employee> employeeList = (List<Employee>) request.getAttribute("employees");

if (employeeList == null || employeeList.isEmpty()) {
out.println("<p>No employees found.</p>");
} else {
%>
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<%
// Iterate through the list and display each employee
for (Employee employee : employeeList) {
%>
<tr>
<td><%= employee.getId() %></td>
<td><%= employee.getName() %></td>
<td><%= employee.getEmail() %></td>
</tr>
<%
}
%>
</tbody>
</table>
<%
}
%>
</body>
</html>
```

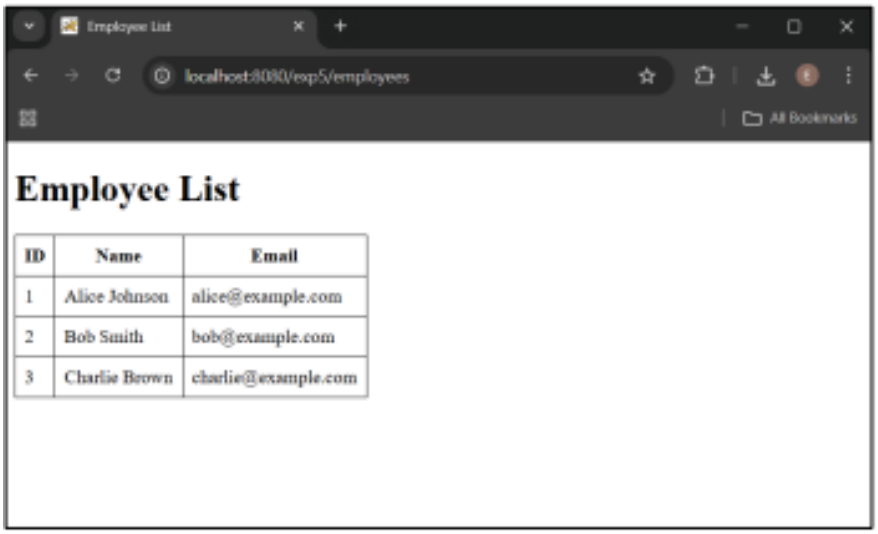
Web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
version="5.0">

<display-name>EmployeeApp</display-name>

<servlet>
```

```
<servlet-name>EmployeeList</servlet-name>


<servlet-class>com.example.servlet.EmployeeServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>EmployeeList</servlet-name>
<url-pattern>/employees</url-pattern>
</servlet-mapping>

<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

OUTPUT:

```
C:\>cd C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps\exp5\WEB-INF\classes

C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps\exp5\WEB-INF\classes>javac com\example\util\DBUtil.java

C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps\exp5\WEB-INF\classes>javac com\example\servlet\Employee.java

C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps\exp5\WEB-INF\classes>javac com\example\servlet\EmployeeServlet.java
C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps\exp5\WEB-INF\classes>
```



Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.

Exercise: 5(iii)

A Java Servlets program for an email registration form with JavaScript validation

---

Aim:

To create a web application with a user registration form. The application will use JavaScript for client-side validation (to check the form before it's submitted) and a Java Servlet for server-side processing (to receive the data).

Algorithm:

Step1: Create index.html: Build a registration form with fields (username, email, password, confirm password). Set the form to method="post", action="register", and onsubmit="return validateForm()".

Step2: Add JavaScript (validateForm()): In the HTML's <script> tag, create the validateForm() function.

- Get the values from all form fields.
- Check if fields are empty, if the email format is valid, and if passwords match.
- If any check fails, alert() an error and return false.
- If all checks pass, return true.

Step3: Create RegisterServlet.java:

- Make a Java class that extends HttpServlet.
- Override the doPost() method.
- Inside doPost(), use request.getParameter() to get the form data.
- Use response.getWriter() to send an HTML success message (e.g., "Welcome, [username]!") back to the user.

Step4: Configure web.xml: Map the URL pattern /register to the RegisterServlet class so the server knows which servlet to run for that URL.

Program:

```
<!DOCTYPE html>
<html>
<head>
<title>Email Registration</title>
<script>
function validateForm() {
var email = document.getElementById("email").value;
var password = document.getElementById("password").value;
var confirmPassword = document.getElementById("confirmPassword").value;
```

```
if (email === "") {
alert("Email is required.");
return false;
}
if (!email.includes("@") || !email.includes(".")) {
alert("Please enter a valid email address.");
return false;
}
if (password === "") {
alert("Password is required.");
return false;
}
if (password.length < 6) {
alert("Password must be at least 6 characters long.");
return false;
}
if (password !== confirmPassword) {
alert("Passwords do not match.");
return false;
}
return true;
}
</script>
</head>
<body>
<h2>Email Registration</h2>
<form action="RegisterServlet" method="post" onsubmit="return validateForm()">
<label>Email:</label>
<input type="text" id="email" name="email"><br><br>
<label>Password:</label>
<input type="password" id="password" name="password"><br><br>
<label>Confirm Password:</label>
<input type="password" id="confirmPassword" name="confirmPassword"><br><br>
<input type="submit" value="Register">
</form>
</body>
</html>
```

RegisterServlet.java:

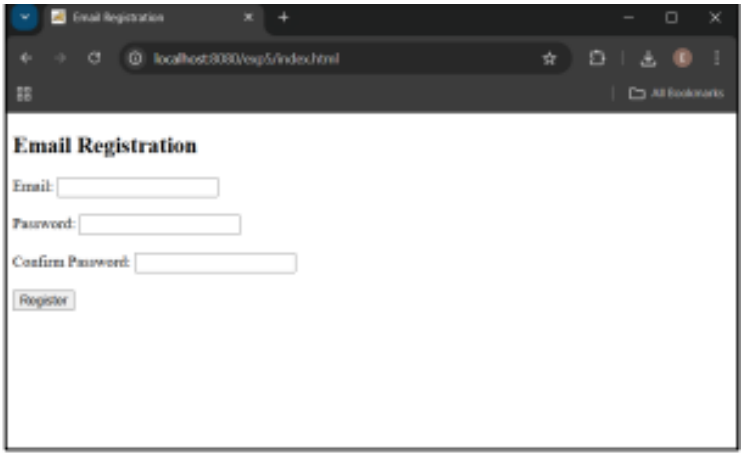
```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class RegisterServlet extends HttpServlet {
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String email = request.getParameter("email");
String password = request.getParameter("password");
// Simple server-side validation (optional, but recommended)
if (email == null || email.isEmpty() || password == null || password.isEmpty()) {
```

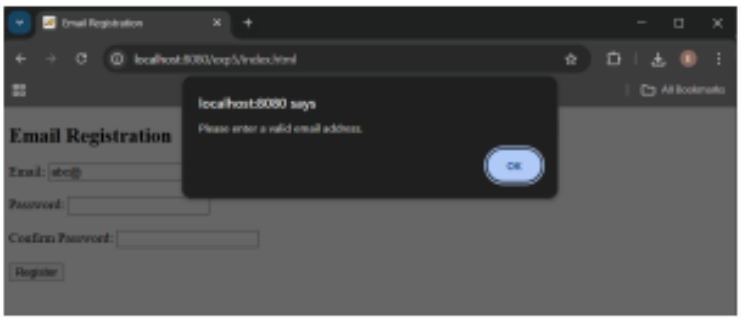
```
out.println("<h2>Registration failed: Missing required fields.</h2>");

return;
}
out.println("<h2>Registration successful!</h2>");
out.println("<p>Email: " + email + "</p>");
}
}
```

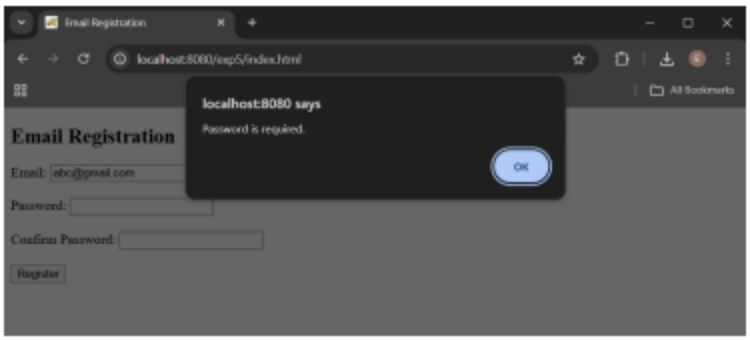
Web.xml:

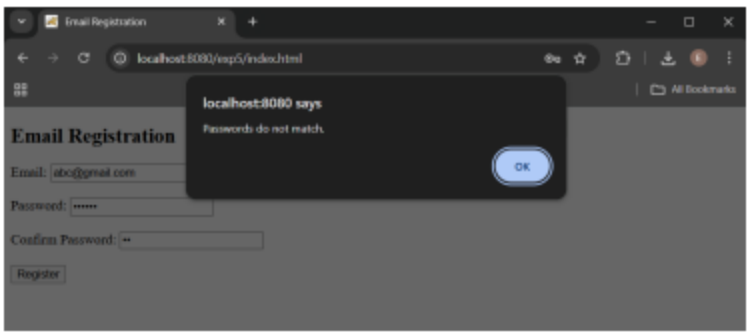
```
<web-app>
<servlet>
<servlet-name>RegisterServlet</servlet-name>
<servlet-class>RegisterServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RegisterServlet</servlet-name>
<url-pattern>/RegisterServlet</url-pattern>
</servlet-mapping>
</web-app>
```

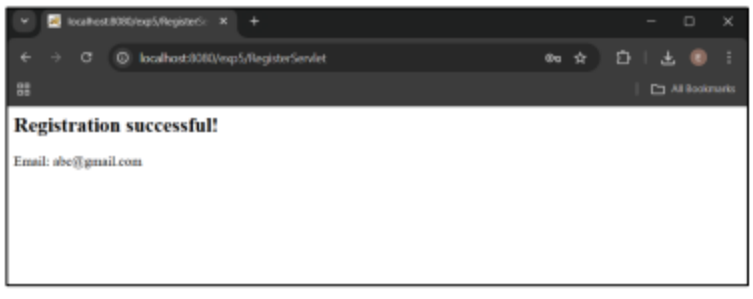
OUTPUT:


Email Registration
localhost:8080/exp5/index.html
Email Registration
Email:
Password:
Confirm Password:
Register


localhost:8080 says
Please enter a valid email address.
OK
Email Registration
Email: abc@
Password:
Confirm Password:
Register


localhost:8080 says
Password is required.
OK
Email Registration
Email: abc@gmail.com
Password:
Confirm Password:
Register


localhost:8080 says
Passwords do not match.
OK
Email Registration
Email: abc@gmail.com
Password:
Confirm Password:
Register


localhost:8080/exp5/RegisterServlet
Registration successful!
Email: abc@gmail.com

Rubrics:

Content	Max Marks	Marks Awarded
Aim	20	
Algorithm	20	
Program Execution	30	
Result	20	
Viva voice	10	
Total	100	

Result:

Thus the program was executed and verified successfully.