

Ex No:
Date:

1. Creating web page using HTML

Aim:

Algorithm:

1. Start the Program
2. Open Notepad and type the HTML code for the image map. .
3. Select the image and decide the hotspot regions. .
4. Add and <map> tags.
5. Use <area> tags to specify shapes and coordinates for the clickable region.
6. Use href or onclick to define actions for hotspots.
7. Create sections to show information when a hotspot is clicked.
8. Stop the program.

Program coding:

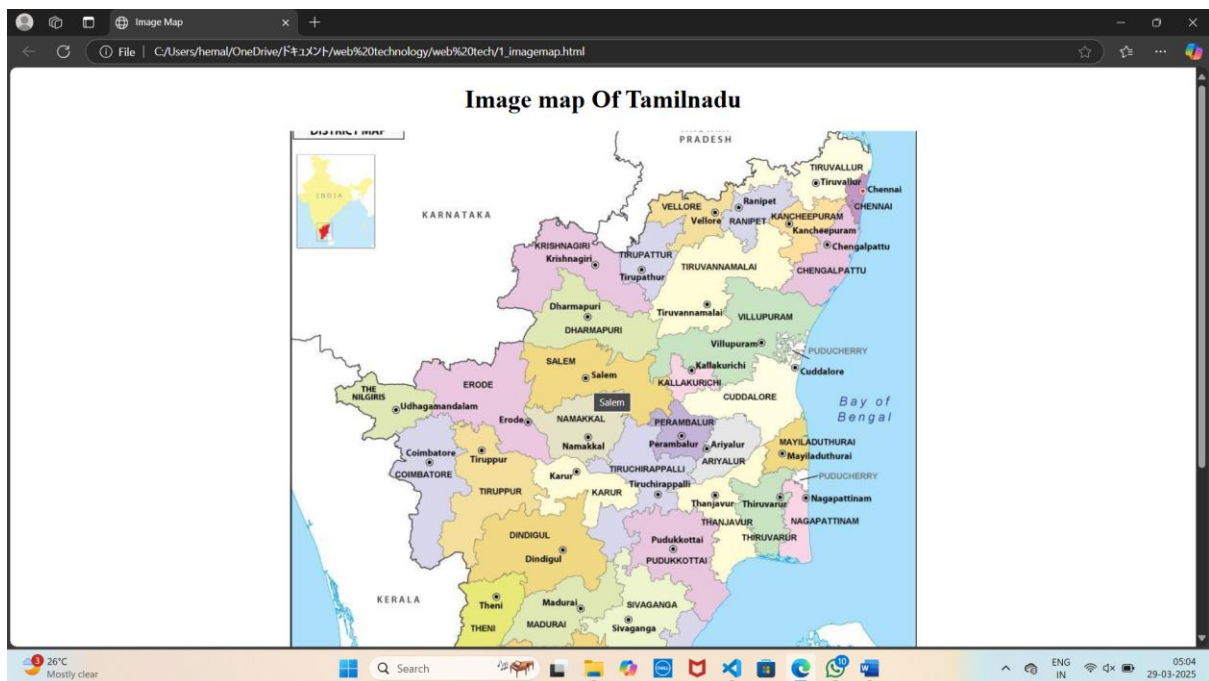
imagemap. html

```
<!DOCTYPE html>
<html>
<head>
  <title>Image Map</title>
  <style>
    body {
      text-align: center;
    }
    img {
      display: block;
      margin: auto;
    }
  </style>
</head>
<body>
  <h1>Image map Of Tamilnadu</h1>
```

```

<map name="dmap">
  <area shape="rect" coords="406,410,336,357" title="Namakkal">
  <area shape="rect" coords="325,332,421,299" title="Salem">
  <area shape="rect" coords="717,68,735,89" title="Chennai">
</map>
</body>
</html>
```

OUTPUT:



Result:

Ex No:
Date:

2. Creating Webpage using all types of Cascading style sheets

Aim:

Algorithm:

1. Start the program.
2. Use the specific HTML tags to create and display the formatted web page that contains Inline, Internal and external cascading style sheet settings.
3. Using <link> tag to make a link from current web page to external cascading style sheet and apply web page formatting settings.
4. Open the web pages to execute the program.
5. Stop the program.

Program coding:

Inline_css. html

```
<html>
<head>
<title>inline sheet</title>
</head>
<body>
  <h1> <center>INLINE STYLE SHEETS </h1>
  <center><p style="font-family: Showcard Gothic">Welcome user. . </p>
  <p style="font-size:24pt; font-family: Microsoft sans serif">Gmail</p>
  <p style="font-size:20pt; colour: red; font-family: arial">The world largest spam free Email
  provider</p>
  <p style="font-size:15pt; colour: blue; font-family: Kristen ITC">Login here. . . </p>
  <h4 style="font-family: Lucida Handwriting">
  User Name:<input type="text" value=""><br><br>
  Password:<input type="password" value=""></h4>
  <input type="submit" value="login">
  <input type="reset" value="cancel"><br><br>
```

</center>

</body>

</html>

Internal_css. html

<html>

<head>

<title> Embedded style sheet </title>

<style type="text/css">

h1

{

font-family: times new roman;

color: green;

}

h2

{

font-family: arial;

color: red;

left:20px;

}

h3

{

font-family: agency fb;

color: blue;

}

p

{

font-size:14pt;

font-family: consolas;

text-align: center;

```
}  
</style>  
</head>  
<body>  
<h1> <center>EMBEDDED STYLE SHEET </h1>  
<h2> CSE </h2>  
<h3>Computer Science and Engineering</h3>  
<center></center><p>  
Computer Science & Engineering (CSE) is an academic program  
at many universities . which comprises scientific and engineering  
aspects of computing.  
</p>  
</center>  
</body>  
</html>
```

External_css. html

```
<html>  
<head>  
<link rel="stylesheet" type="text/css" href="Styles. css"/>  
</head>  
<body> <center>  
<h1> EXTERNAL STYLE SHEETS</h1></center>  
<h2><center>ANNA UNIVERSITY</center></h2>  
<h3>Our Institution consists of various departments:</h3>  
<h4>  
1. CSE<br>  
2. ECE<br>  
3. MECH<br>  
4. EEE<br>  
5. MECHT<br>  
6. CIVIL<br>
```

7. IT
 </h4>

</body>

</html>

Styles. css

h1

{

font-family:times new roman;

color:red;

}

h2

{

font-family:consolas;

color:pink;

font-size: 35;

}

h3

{

font-family:agency fb;

font-size: 30;

color:palevioletred;

text-align: center;

}

h4

{

font-family: agency fb;

font-size: 30;

color: blueviolet;

text-align:center;

}

OUTPUT:

INLINE STYLE SHEETS
WELCOME USER..
Gmail
The worlds largest spam free Email provider
[Login here...](#)
User Name:
Password:

Internal STYLE SHEET
CSE
[Computer Science and Engineering](#)
Computer Science & Engineering (CSE) is an academic program at many universities .

EXTERNAL STYLE SHEETS
ANNA UNIVERSITY
Our Institution consists of various departments:
1. CSE
2. ECE
3. MECH
4. EEE
5. MECHT
6. CIVIL
7. IT

Result:

Ex No:

Date:

3. Create a Website for CRM using Cascading Style sheets and perform validation using Java Script

Aim:

Algorithm:

1. Start the Program.
2. Plan website structure (Home, Login, Dashboard, Reports).
3. Create HTML layout using <div>, <form>, and <input>.
4. Style with CSS for layout, colors, fonts, and responsiveness.
5. Add JavaScript for form validation (email, password, required fields) CSS to design the layout the form
6. Test & Debug in a browser, fix errors, and refine UI.
7. Save & Deploy after final adjustments.
8. Stop the program.

Program coding:

index. html

```
<!DOCTYPE html>
<html>
<head>
  <title>CRM System</title>
  <link rel="stylesheet" href="styles. css">
</head>
<body>
  <div class="container">
    <h2>Customer Relationship Management</h2>
    <form id="customerForm">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name">

      <label for="email">Email:</label>
      <input type="text" id="email" name="email">

      <label for="phone">Phone:</label>
      <input type="text" id="phone" name="phone">
```



```
<label for="message">Message:</label>
<textarea id="message" name="message"></textarea>

<button type="submit">Add Customer</button>
</form>

<h3>Customer List</h3>
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Email</th>
      <th>Phone</th>
      <th>Message</th>
    </tr>
  </thead>
  <tbody id="customerTable">
  </tbody>
</table>
</div>
<script src="script. js"></script>
</body>
</html>
```

Styles. css

```
body {
  font-family: Arial, sans-serif;
  background: url('. /cmr1. jpg') no-repeat center center fixed;
  background-size: cover;
  margin: 0;
  padding: 20px;
}

.container {
  max-width: 600px;
  background: rgba(255,255,255, 0. 8);
  padding: 20px;
  margin: auto;
  box-shadow: 0px 0px 10px rgba(0, 0, 0, 0. 1);
}
```

```
    border-radius: 8px;
}
h2, h3 {
    text-align: center;
}
form {
    display: flex;
    flex-direction: column;
}
label {
    margin-top: 10px;
}
input, textarea {
    padding: 8px;
    margin-top: 5px;
    border: 1px solid black;
    border-radius: 5px;
}
button {
    margin-top: 15px;
    padding: 10px;
    background-color: #2896a7;
    color: white;
    border: none;
    cursor: pointer;
    border-radius: 5px;
}
button:hover {
    background-color: #2881a7;
}
table {
    width: 100%;
```

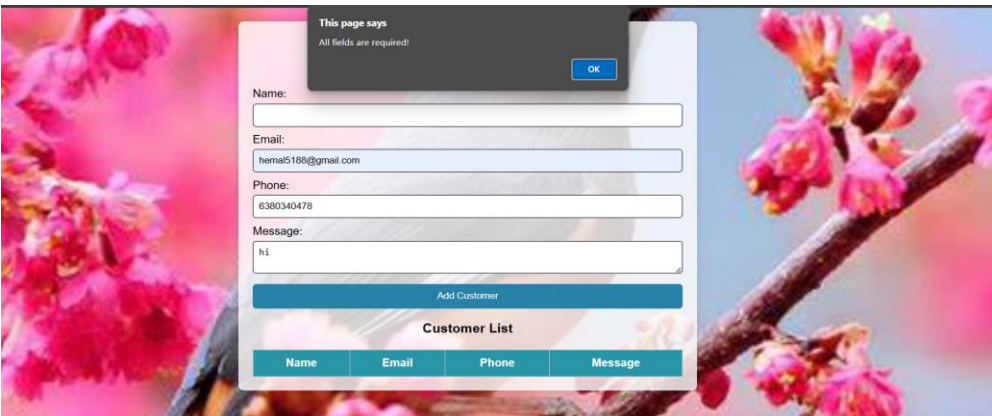
```
margin-top: 20px;
border-collapse: collapse;
}
th, td {
padding: 10px;
border: 1px solid #ccc;
text-align: center;
}
th {
background-color: #2896a7;
color: white;
}
```

Script. js

```
document. getElementById("customerForm"). addEventListener("submit",
function(event) {
event. preventDefault();
let name = document. getElementById("name"). value. trim();
let email = document. getElementById("email"). value. trim();
let phone = document. getElementById("phone"). value. trim();
let message = document. getElementById("message"). value. trim();
if (!validateForm(name, email, phone)) {
return; }
addCustomerToTable(name, email, phone, message);
clearForm();
});
function validateForm(name, email, phone) {
let emailPattern = /^[^ ]+@[^ ]+\.[ a-z]{2,3}$/;
let phonePattern = /^\d{10}$/;
if (name === "" || email === "" || phone === "") {
alert("All fields are required!");
return false;
}
```

```
if (!email. match(emailPattern)) {  
    alert("Enter a valid email!");  
    return false;  
}  
  
if (!phone. match(phonePattern)) {  
    alert("Enter a valid 10-digit phone number!");  
    return false;  
}  
  
return true;  
}  
  
function addCustomerToTable(name, email, phone, message) {  
    let table = document. getElementById("customerTable");  
    let row = table. insertRow();  
    row. innerHTML = `  
        <td>${name}</td>  
        <td>${email}</td>  
        <td>${phone}</td>  
        <td>${message}</td>`;   
    }  
  
function clearForm() {  
    document. getElementById("customerForm"). reset();  
}
```

OUTPUT:



The screenshot shows a web form titled "Customer Form" with a validation error message: "This page says: All fields are required!". The form fields are: Name (empty), Email (filled with "hema1518@gmail.com"), Phone (filled with "6380340478"), and Message (filled with "hi"). Below the form is a blue "Add Customer" button. Below the button is a table titled "Customer List" with columns: Name, Email, Phone, and Message. The table is currently empty.

Name	Email	Phone	Message
------	-------	-------	---------

This page says
Enter a valid email!

OK

Name:
Hema

Email:
hema15188@gmai

Phone:
6380340478

Message:
hi

Add Customer

Customer List

Name	Email	Phone	Message
------	-------	-------	---------

The screenshot shows a web form with a validation error. A dark grey alert box at the top says "This page says Enter a valid email!" with an "OK" button. The form fields are: Name (Hema), Email (hema15188@gmai), Phone (6380340478), and Message (hi). Below the fields is a blue "Add Customer" button. Underneath is a section titled "Customer List" with a table header showing columns for Name, Email, Phone, and Message.

This page says
Enter a valid 10-digit phone number!

OK

Name:
Hema

Email:
hema15188@gmail.com

Phone:
63803404

Message:
hi

Add Customer

Customer List

Name	Email	Phone	Message
------	-------	-------	---------

The screenshot shows the same web form as above, but with a different validation error. The alert box now says "Enter a valid 10-digit phone number!". The form fields are: Name (Hema), Email (hema15188@gmail.com), Phone (63803404), and Message (hi). The "Add Customer" button and "Customer List" table header are also present.

Customer Relationship Management

Name:
Hema

Email:
hema15188@gmail.com

Phone:
6380340478

Message:
hi

Add Customer

Customer List

Name	Email	Phone	Message
Hema	hema15188@gmail.com	6380340478	hi

The screenshot shows the web form with the title "Customer Relationship Management". The form fields are filled with the same data as before. The "Add Customer" button is present. Below it, the "Customer List" section shows a table with one row of data: Name (Hema), Email (hema15188@gmail.com), Phone (6380340478), and Message (hi).

Result:

Ex No:

Date:

4. Creation of Servlet based web Application with JDBC

Aim:

Algorithm:

1. Start the program.
2. Setup the Environment.
3. Create the database and table.
4. Establish database connection using JDBC.
5. Create Servlet for user login.
6. Create JSP page for user login.
7. Evaluate HTTP Request and Response process and which have proper session management.
8. Deploy and test the web application.
9. Stop the program.

PROGRAM:

LoginServlet. java

```
package cscorner;
```

```
import java. io. IOException;
import java. io. PrintWriter;
import java. sql. Connection;
import java. sql. DriverManager;
import java. sql. PreparedStatement;
import java. sql. ResultSet;
import java. sql. SQLException;

import javax. servlet. RequestDispatcher;
import javax. servlet. ServletException;
import javax. servlet. annotation..WebServlet;
import javax. servlet. http. HttpServlet;
import javax. servlet. http. HttpServletRequest;
import javax. servlet. http. HttpServletResponse;
```

```
/**
```

```
 * Servlet implementation class LoginServlet
```

```
 */
```

```
@WebServlet("/LoginServlet")
```

```
public class LoginServlet extends HttpServlet {
```

```
private static final long serialVersionUID = 1L;
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    try{
        PrintWriter out=response. getWriter();
        out. println("Login Successful");
        Class.forName("com. mysql. cj. jdbc. Driver");
        Connection con = DriverManager.
getConnection("jdbc:mysql://localhost:3306/Hema","root","cscorner@1234");
        String n=request. getParameter("txtName");
        String p=request. getParameter("txtpwd");
        PreparedStatement ps=con. prepareStatement("select uname from login where
uname=? and password=?");
        ps. setString(1,n);
        ps. setString(2,p);
        ResultSet rs=ps. executeQuery();
        if(rs. next())
        {
            RequestDispatcher rd=request. getRequestDispatcher("welcome. jsp");
            rd. forward(request, response);
        }
        else
        {
            out. println("<font color=red size-18>Login Failed!!<br>");
            out. println("<a href=login. jsp>Try Again!!</a>");
        }
    } catch(ClassNotFoundException e) {
        e. printStackTrace();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e. printStackTrace();
    };
}
```

Login. jsp

```
<% @ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>Login</title>
</head>
<body bgcolor="pink">
<div align=center>
<h1>User login</h1>
</div>
<form action=LoginServlet method=post>
<center>
<table>
<tr><td>Enter name: </td><td><input type=text name=txtName></td></tr><br>
<tr><td>Enter password: </td><td><input type=password name=txtName></td></tr><br>
<tr><td><input type=submit value=login></td><td><input type=reset></td></tr>
</table>
</center>
</form>
</body>
</html>
```

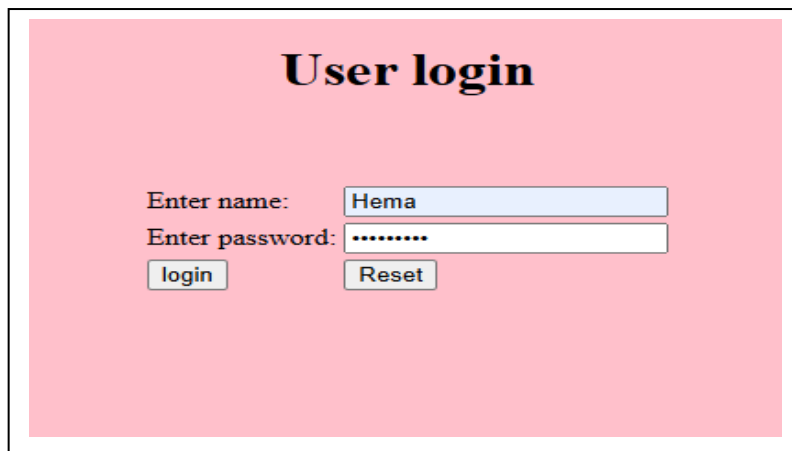
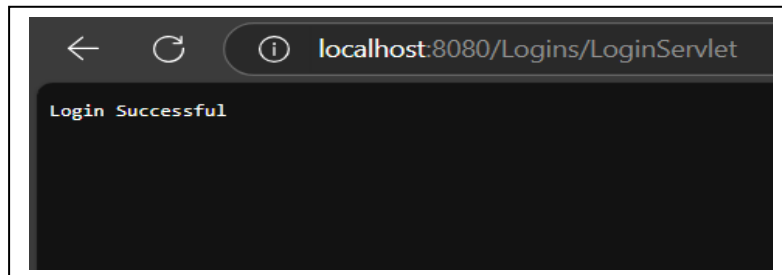
Welcome. jsp

```
<% @ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<style>
body{
    background: linear-gradient(to right, #00ffff, #8E44AD);
```



```
}</style>
<body>
<div align=center>
<h1>Login Successfull !!</h1>
</div>
</body>
</html>
```

OUTPUT:

A screenshot of a web application's login page. The page has a light pink background. At the top center, the text "User login" is displayed in a bold, black, serif font. Below this, there are two input fields. The first is labeled "Enter name:" and contains the text "Hema". The second is labeled "Enter password:" and contains a series of dots. Below the input fields, there are two buttons: "login" and "Reset".

RESULT:

Ex No:

Date:

5. Create three-tier applications using JSP and Databases

Aim:

Algorithm:

1. Start the program.
2. Create tables from database for store on-line exam question, answer and final mark details of students.
3. Create a JSP for Home page to have hyperlink with new student registration page and old student login page to start on-line exam for specific subject according to the department.
4. Create a JSP for Registration page to get personal and login information of the student.
5. Create a JSP for login page to start on-line examination to assessing the student knowledge.
6. Create a JSP to display student mark list at the end of on-line examination.
7. Make proper hyperlink with each web pages.
8. Stop the program.

PROGRAM:

index.html:

```
<html>

<head>

<title>exam portal</title>

<meta charset="UTF-8">

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

</head>

<body>

<div style="text-align: center">

<h3>ONLINE EXAM PORTAL - Login </h3>

<hr>

<form method = "get" action="acceptuser.jsp">

Username <input type="text" name="uname" value=""><br><br>

Password <input type="password" name="pass" value=""><br><br>

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

</form>
```

```
</div>
</body>
</html>
```

acceptuser.jsp

```
<% @ page import="java.sql.*"%>
<% @ page import="java.util.*"%>
<%!
Connection con;
PreparedStatement ps1, ps2;
public void jspInit(){
try {
    Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
    con = DriverManager.getConnection("jdbc:derby://localhost:1527/iplab","root","root");
    ps1 = con.prepareStatement("select count(*) from USERS where username = ? and password=?");
    ps2 = con.prepareStatement("select * from USERS");
}
catch(Exception ex){
    ex.printStackTrace();
}
}
%>
<%
String param = request.getParameter("s1");
if(param=="link"){
    ResultSet rs = ps2.executeQuery();
    out.println("<table>");
    while(rs.next()){
        out.println("<tr>");
        out.println("<td>"+rs.getString(1)+"</td>");
        out.println("<td>"+rs.getString(2)+"</td>");
        out.println("</tr>");
    }
}
```

```
out.println("</table>");
rs.close();
}
else{
String user = request.getParameter("uname");
String pass = request.getParameter("pass");
ps1.setString(1,user);
ps1.setString(2,pass);
ResultSet rs = ps1.executeQuery();
int cnt = 0;
if (rs.next())
    cnt = rs.getInt(1); // index 1 is the "''" column
if(cnt == 0)
    out.println("<b><i><font color=red>Invalid credential</font></i></b>");
else{
out.println("<form><center><fieldset style= width:55% height:60%;>");
out.println("<b><font color=red>Valid Credential..</font></b><br>");
out.println("<b><a href=examclient.html><font size=6 color=blue>Click Here to take Online  
Exam</font></a></b>");
out.println("</fieldset></form>");
}
}
%>
<%!
public void jspDestroy(){
try{
ps1.close();
ps2.close();
con.close();
}
catch(Exception ex){
ex.printStackTrace();
```

```
}  
}  
%>
```

examclient.html:

```
<html>  
<head>  
<title>Online Exam Client</title>  
</head>  
<body>  
<h2 style="text-align:center">ONLINE EXAMINATION</h2>  
<h3>Answer the following questions (5 marks for each correct answer)</h3>  
<hr/>  
<form name="examForm" method="post" action="ExamServer.jsp">  
1.Who is called as the father of computer?<br/>  
<input type="radio" name="ans1" value="Sachin">Sachin  
<input type="radio" name="ans1" value="Stuart">Stuart  
<input type="radio" name="ans1" value="Charles Babbage">Charles Babbage  
<input type="radio" name="ans1" value="Napier">Napier  
<br/><br/>  
2.Python was developed by?<br/>  
<input type="radio" name="ans2" value="Dennis Ritchie">Dennis Ritchie  
<input type="radio" name="ans2" value="Guido van Rossum"> Guido van Rossum  
<input type="radio" name="ans2" value="David Ritchie">David Ritchie  
<input type="radio" name="ans2" value="John">John  
<br/><br/>  
3.C was developed by?<br/>  
<input type="radio" name="ans3" value="Dennis Ritchie">Dennis Ritchie  
<input type="radio" name="ans3" value="Stroustrup">Stroustrup  
<input type="radio" name="ans3" value="David Ritchie">David Ritchie  
<input type="radio" name="ans3" value="Charles Babbage">Charles Babbage  
<br/><br/>  
<input type="submit" value="Check Your Result"/>
```

</form>

</body>

</html>

ExamServer.jsp:

```
<% @page contentType="text/html" language="java" import="java.sql.*"%>
```

```
<html>
```

```
<head>
```

```
<title>Online Exam Server</title>
```

```
<style type="text/css">
```

```
body{ background-color:white;font-family:courier;color:blue}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2 style="text-align:center">ONLINE EXAMINATION</h2>
```

```
<p>
```

```
<a href="examclient.html">Back To Main Page</a>
```

```
</p>
```

```
<hr/>
```

```
<%
```

```
String str1=request.getParameter("ans1");
```

```
String str2=request.getParameter("ans2");
```

```
String str3=request.getParameter("ans3");
```

```
int mark=0;
```

```
Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
```

```
Connection con=DriverManager.getConnection("jdbc:derby://localhost:1527/iplab","root","root");
```

```
Statement stmt=con.createStatement();
```

```
ResultSet rs=stmt.executeQuery("SELECT * FROM examTab");
```

```
while(rs.next())
```

```
{
```

```
String dbans1=rs.getString(1);
```

```
String dbans2=rs.getString(2);
```

```
String dbans3=rs.getString(3);
```

```
if(str1.equals(dbans1)){
    mark=mark+5;
}
if(str2.equals(dbans2)){
    mark=mark+5;
}
if(str3.equals(dbans3)){
    mark=mark+5;
}
}
if(mark>=10){
    out.println("<h4>Your Mark Is : "+mark+"</h4>");
    out.println("<h3>Congratulations....! You Are Eligible For The Next Round...</h3>");
}
else{
    out.println("<h4>Your Mark is : "+mark+"</h4>");
    out.println("<h3>Sorry.....!! You Are Not Eligible For The Next Round...</h3>");
}
%>
</form>
</body>
</html>
```

OUTPUT:

ONLINE EXAM PORTAL - Login	
Username	<input type="text"/>
Password	<input type="password"/>
<input type="button" value="LOGIN"/>	

ONLINE EXAMINATION

Answer the following questions (5 marks for each correct answer)

I

1. Who is called as the father of computer?

☐ Sachin ☐ Stuart ☐ Charles Babbage ☐ Napier

2. Python was developed by?

☐ Dennis Ritchie ☐ Guido van Rossum ☐ David Ritchie ☐ John

3. C was developed by?

☐ Dennis Ritchie ☐ Stroustrup ☐ David Ritchie ☐ Charles Babbage

[Check Your Result](#)

ONLINE EXAMINATION

[Back To Exam](#)

I

Your Mark Is : 15

Congratulations....! You Are Eligible For The Next Round...

RESULT:

Ex No:

Date:

6. Creation of XML File using DTD and XML Schema

Aim

Algorithm:

1. Start program.
2. Create web document structure using XML Schema for employee details.
3. Store the employee' details in the created unstructured database location.
4. Display XML document content using supporting browser to parse XML tags and show only raw data in browser.
5. Translate the XML document content into other markup language type with proper style by using DTD.
6. Before proceeding with XML DTD, must check the validation.
7. A well-formed and valid XML document is one which has been validated against DTD.
8. Stop program

PROGRAM:

Employee.xml

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

```
<!DOCTYPE library SYSTEM "library.dtd">
```

```
<library>
```

```
  <book isbn="978-3-16-148410-0" publishedYear="2020">
```

```
    <title>Introduction to Algorithms</title>
```

```
    <author>
```

```
      <firstname>Thomas</firstname>
```

```
      <lastname>Cormen</lastname>
```

```
    </author>
```

```
    <publisher>MIT Press</publisher>
```

```
    <price>59.99</price>
```

```
  </book>
```

```
  <book isbn="978-0-321-49805-2" publishedYear="2015">
```

```
    <title>Artificial Intelligence: A Modern Approach</title>
```

```
<author>
  <firstname>Stuart</firstname>
  <lastname>Russell</lastname>
</author>
<publisher>Pearson</publisher>
<price>89.99</price>
</book>
<book isbn="978-1-491-94600-8" publishedYear="2017">
  <title>Python for Data Analysis</title>
  <author>
    <firstname>Wes</firstname>
    <lastname>McKinney</lastname>
  </author>
  <publisher>O'Reilly</publisher>
  <price>39.99</price>
</book>
</library>
```

library.dtd

```
<!ELEMENT library (book+)>
<!ELEMENT book (title, author, publisher, price)>
<!ATTLIST book
  isbn CDATA #REQUIRED
  publishedYear CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
```

<!ELEMENT price (#PCDATA)>

OUTPUT:

```
▼<library>
  ▼<book isbn="978-3-16-148410-0" publishedYear="2020">
    <title>Introduction to Algorithms</title>
    ▼<author>
      <firstname>Thomas</firstname>
      <lastname>Cormen</lastname>
    </author>
    <publisher>MIT Press</publisher>
    <price>59.99</price>
  </book>
  ▼<book isbn="978-0-321-49805-2" publishedYear="2015">
    <title>Artificial Intelligence: A Modern Approach</title>
    ▼<author>
      <firstname>Stuart</firstname>
      <lastname>Russell</lastname>
    </author>
    <publisher>Pearson</publisher>
    <price>89.99</price>
  </book>
  ▼<book isbn="978-1-491-94600-8" publishedYear="2017">
    <title>Python for Data Analysis</title>
    ▼<author>
      <firstname>Wes</firstname>
      <lastname>McKinney</lastname>
    </author>
    <publisher>O'Reilly</publisher>
    <price>39.99</price>
  </book>
</library>
```

RESULT:

Ex No:

Date:

7. Develop a new Web Services for Calculator

Aim

.

Algorithm:

1. Create a JSP form for user input (two numbers and an operation).
2. Submit the form data to a Servlet for processing.
3. Retrieve input values in the Servlet and validate them.
4. Perform the selected arithmetic operation.
5. Forward the result to a JSP page for display.
6. Show the result or an error message, with a "Back" button for a new calculation.

PROGRAM:

index.jsp

```
<% @ page language="java" contentType="text/html; charset=UTF-8" %>

<!DOCTYPE html>

<html>

<head>

    <title>Web Calculator</title>

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

</head>

<center>

<body>

    <h2>Calculator</h2>

    <form action="calculate" method="post">

        <input type="number" name="num1" required placeholder="Enter first number"><br>
```

```
<select name="operation">

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

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

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

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

</select><br>

<input type="number" name="num2" required placeholder="Enter second number">

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

</form>

</body>

</center>

</html>
```

result.jsp

```
<% @ page language="java" contentType="text/html; charset=UTF-8" %>

<!DOCTYPE html>

<html>

<head>

    <title>Calculation Result</title>

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

</head>

<body>

    <div class="result-container">

        <h2>Calculation Result</h2><br><br>

        <%

            if (request.getAttribute("error") != null) {
```

```
%>

<p class="error"><%= request.getAttribute("error") %></p>

<%

    } else {

%>

    <p>Result: <%= request.getAttribute("result") %></p>

<% }

%>

<a href="index.jsp">Back</a>

</div>

</body>

</html>
```

CalculatorServlet.java

```
package com.calculator;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/calculate")

public class CalculatorServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {
```

```
double num1 = Double.parseDouble(request.getParameter("num1"));

double num2 = Double.parseDouble(request.getParameter("num2"));

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

double result = 0;

switch (operation) {

    case "add":

        result = num1 + num2;

        break;

    case "subtract":

        result = num1 - num2;

        break;

    case "multiply":

        result = num1 * num2;

        break;

    case "divide":

        if (num2 != 0) {

            result = num1 / num2;

        } else {

            request.setAttribute("error", "Cannot divide by zero!");

        }

        break;

    default:

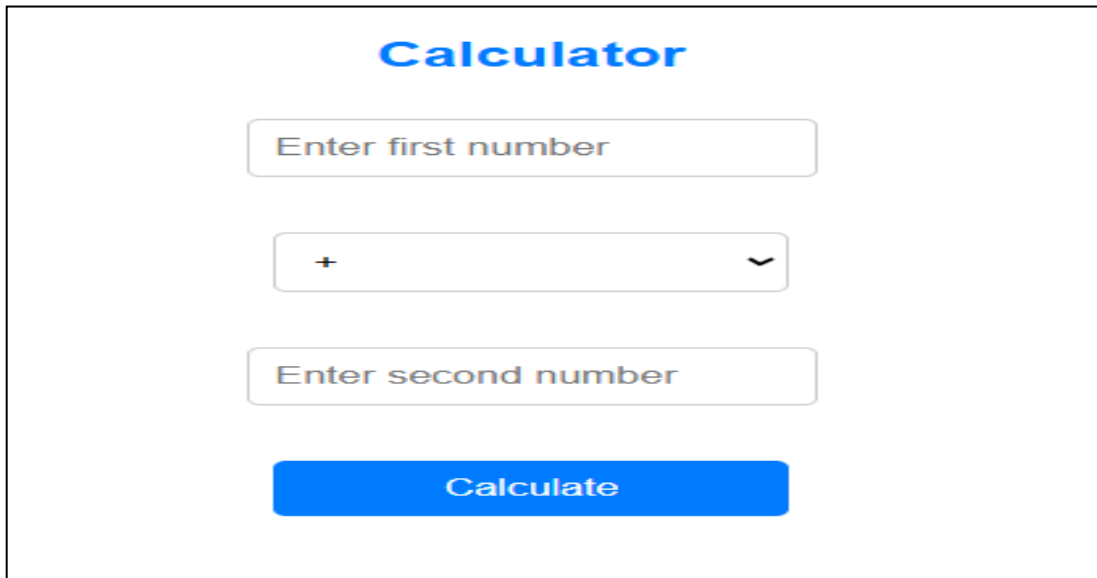
        request.setAttribute("error", "Invalid operation!");

        break;

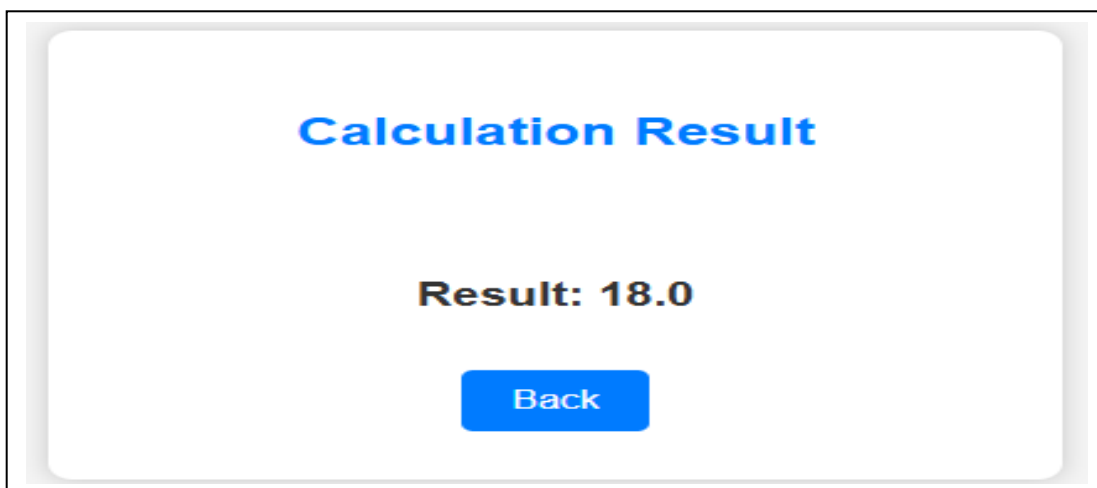
}
```

```
request.setAttribute("result", result);  
  
request.getRequestDispatcher("result.jsp").forward(request, response);  
  
}  
  
}
```

OUTPUT:



The image shows a web-based calculator interface. At the top, the word "Calculator" is displayed in a bold, blue font. Below this, there are two input fields with the placeholder text "Enter first number" and "Enter second number". Between these fields is a dropdown menu showing a plus sign (+) and a downward arrow. At the bottom of the interface is a blue button with the text "Calculate" in white.



The image shows a web-based interface for displaying calculation results. It features a light gray box with a subtle drop shadow. Inside the box, the text "Calculation Result" is centered at the top in a bold, blue font. Below this, the text "Result: 18.0" is centered in a bold, black font. At the bottom of the box is a blue button with the text "Back" in white.

Result:

Ex No:

Date:

8. Working with DOM and SAX parsers

Aim

Algorithm:

1. Start the program
2. Import necessary packages which are support for this implementation.
3. Define class and declare objects for DocumentBuilderFactory, DocumentBuilder and Document classes.
4. Define a constructor to create instances for the above classes.
5. Get the XML file path for XML content to be parse.
6. Pass the FileInputStream with filepath as argument to DocumentBuilder function called 'parse' to parse XML document elements and return the result as document to store in Document object.
7. Split the elements as Root, Sibling, Parent, Child and etc. , to display in output window.
8. Stop the program.

PROGRAM:

DOM Parser

```
package com.madhu.xml;
import java.io.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;

public class SimpleWalker {
    protected DocumentBuilder docBuilder;
    protected Element root;

    public SimpleWalker() throws Exception {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        docBuilder = dbf.newDocumentBuilder();
        DOMImplementation domImp = docBuilder.getDOMImplementation();
        if (domImp.hasFeature("XML", "2.0")) {
            System.out.println("Parser supports extended interfaces");
        }
    }

    public void parse(String fileName) throws Exception {
        Document doc = docBuilder.parse(new FileInputStream(fileName));
```

```
    root = doc.getDocumentElement();
    System.out.println("Root element is " + root.getNodeName());
}
public void printAllElements() {
    if (root != null) {
        printElement("", (Node) root);
    } else {
        System.out.println("No elements to print.XML file may be empty.");
    }
}
public void printElement(String indent, Node node) {
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        System.out.println(indent + "<" + node.getNodeName() + ">");
        Node child = node.getFirstChild();
        while (child != null) {
            printElement(indent + " ", child);
            child = child.getNextSibling();
        }
        System.out.println(indent + "</" + node.getNodeName() + ">");
    }
    else if (node.getNodeType() == Node.TEXT_NODE) {
        String text = node.getTextContent().trim();
        if (!text.isEmpty()) {
            System.out.println(indent + text);
        }
    }
}
public static void main(String args[]) throws Exception {
    if (args.length == 0) {
        System.out.println("Usage: java com.madhu.xml.SimpleWalker <XML file path>");
        return;
    }
    SimpleWalker sw = new SimpleWalker();
    sw.parse(args[0]);
}
```

```
        sw.printAllElements();
    }
}
```

Parse XML Using SAX Parser

```
package com.madhu.xml;
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;

public class SAXDemo extends DefaultHandler {

    public void startDocument() {
        System.out.println("*** Start of Document ***");
    }

    public void endDocument() {
        System.out.println("*** End of Document ***");
    }

    public void startElement(String uri, String localName, String qName, Attributes attributes) {
        System.out.print("<" + qName);
        int n = attributes.getLength();
        for (int i = 0; i < n; i++) {
            System.out.print(" " + attributes.getQName(i) + "=" + attributes.getValue(i) + "");
        }
        System.out.println(">");
    }

    public void characters(char[] ch, int start, int length) {
        String text = new String(ch, start, length).trim();
        if (!text.isEmpty()) {
            System.out.println(text);
        }
    }

    public void endElement(String namespaceURI, String localName, String qName) throws
        SAXException {
        System.out.println("</" + qName + ">");
    }
}
```

```
}  
public static void main(String[] args) {  
    if (args.length == 0) {  
        System.out.println("Usage: java com.madhu.xml.SAXDemo <XML file path>");  
        return;  
    }  
    String xmlFilePath = args[0]; // Get XML file path from command line  
    try {  
        SAXParserFactory factory = SAXParserFactory.newInstance();  
        SAXParser saxParser = factory.newSAXParser();  
        SAXDemo handler = new SAXDemo();  
        saxParser.parse(new File(xmlFilePath), handler);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Library.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<library>  
    <book>  
        <title>Wings of Fire</title>  
        <author>A.P.J.Abdul Kalam</author>  
    </book>  
    <book>  
        <title>The Alchemist</title>  
        <author>Paulo Coelho</author>  
    </book>  
</library>
```

OUTPUT:

SimpleWalker.java	SAXDemo.java
<pre>D:\XMLParsierProject\src>java -cp . Parser supports extended interfaces Root element is library <library> <book> <title> Wings of Fire </title> <author> A. P. J. Abdul Kalam </author> </book> <book> <title> The Alchemist </title> <author> Paulo Coelho </author> </book> </library></pre>	<pre>D:\XMLParsierProject\src>java *** Start of Document *** <library> <book> <title> Wings of Fire </title> <author> A. P. J. Abdul Kalam </author> </book> <book> <title> The Alchemist </title> <author> Paulo Coelho </author> </book> </library> *** End of Document *** D:\XMLParsierProject\src></pre>

RESULT:

Ex No:

Date:

9. Creation of AJAX based application

Aim

Algorithm:

1. Start the program.
2. Create dynamic web page using HTML tags.
3. Embed java script with XML HTTP Request object.
4. To check browser supports using XML HTTP Request object.
5. Create <div> for display file content using AJAX.
6. Make necessary status and code checkup and everything to create link with external file to access file content and display in <div> location.
7. To change file content dynamically, this will refresh only <div> location instead of refresh the complete web page.

PROGRAM:

ajax.jsp

```
<% @ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>ajax</title>
<script type="text/javascript">
function fun()
{
if (window.XMLHttpRequest)
xmlhttp=new XMLHttpRequest();
else
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
xmlhttp.onreadystatechange=function()
{
```

```
if (xmlhttp.readyState==4 &&xmlhttp.status==200)

{
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
}
}

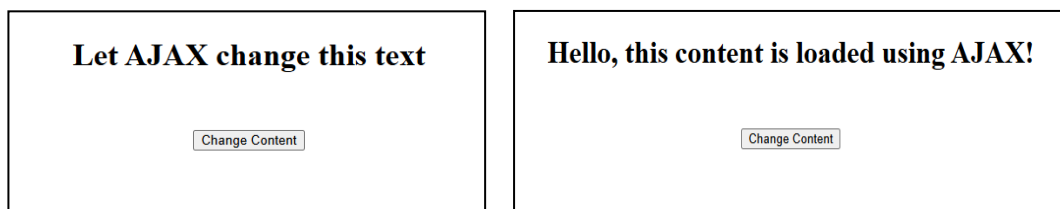
xmlhttp.open("GET","change.txt",true);
xmlhttp.send();
}

</script>
</head>
<center>
<body>
<br><br>
<div id="myDiv"><h1>Let AJAX change this text</h1></div><br><br>
<button type="button" onclick="fun()">Change Content</button>
</body>
</center>
</html>
```

Change.txt:

<h1>Hello, this content is loaded using AJAX!</h1>

OUTPUT:



RESULT:

.

Advanced Experiments

Ex No:
Date:



10. Programs using WEB Service for Authentication

Aim

Algorithm:

1. Create a client ID and client secret

To create a client ID and client secret, create a Google API Console project, set up an OAuth client ID, and register your JavaScript origins:

1. Go to the Google API Console .
2. From the project drop-down, select an existing project , or create a new one by selecting **Create a new project**.

Note: Use a single project to hold all platform instances of your app (Android, iOS, web, etc.), each with a different Client ID.

3. In the sidebar under "APIs & Services", select **Credentials**, then select the **OAuth consent screen** tab.
 - a. Choose an **Email Address**, specify a **Product Name**, and press **Save**.
4. In the **Credentials** tab, select the **Create credentials** drop-down list, and choose **OAuth client ID**.
5. Under **Application type**, select **Web application**. Register the origins from which your app is allowed to access the Google APIs, as follows. An origin is a unique combination of protocol, hostname, and port.
 - a. In the **Authorized JavaScript origins** field, enter the origin for your app. You can enter multiple origins to allow for your app to run on different protocols, domains, or subdomains. You cannot use wildcards. In the example below, the second URL could be a production URL.

```
http://localhost:8080
https://myproductionurl. example. com
```

- b. The **Authorized redirect URI** field does not require a value. Redirect URIs are not used with JavaScript APIs.
 - c. Press the **Create** button.
6. From the resulting **OAuth client dialog box**, copy the **Client ID** . The Client ID lets your app access enabled Google APIs.

Step 2: Include the Google platform library on your page

Include the following scripts that demonstrate an anonymous function that inserts a script into the DOM of this `index.html` web page.

```
<!-- The top of file index.html -->
<htmlitemscopeitemtype="http://schema.org/Article">
<head>
  <!-- BEGIN Pre-requisites -->
  <scriptsrc="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">
  </script>
  <scriptsrc="https://apis.google.com/js/client:platform.js?onload=start"asyncdefer>
  </script>
  <!-- END Pre-requisites -->
```

Step 3: Initialize the GoogleAuth object

Load the `auth2` library and call `gapi.auth2.init()` to initialize the `GoogleAuth` object. Specify your client ID and the scopes you want to request when you call `init()`.

```
<!-- Continuing the <head> section -->
<script>
  function start(){
    gapi.load('auth2',function(){
      auth2 = gapi.auth2.init({
        client_id:'YOUR_CLIENT_ID.apps.googleusercontent.com',
        // Scopes to request in addition to 'profile' and 'email'
        //scope: 'additional_scope'
      });
    });
  }
</script>
</head>
<body>
  <!-- . . . -->
</body>
</html>
```

Step 4: Add the sign-in button to your page

Add the sign-in button to your web page, and attach a click handler to call `grantOfflineAccess()` to start the one-time-code flow.

```
<!-- Add where you want your sign-in button to render -->
<!-- Use an image that follows the branding guidelines in a real app -->
<buttonid="signinButton">Sign in with Google</button>
<script>
  $('#signinButton'). click(function(){
    // signInCallback defined in step 6.
    auth2. grantOfflineAccess(). then(signInCallback);
  });
</script>
```

Step 5: Sign in the user

The user clicks the sign-in button and grants your app access to the permissions that you requested. Then, the callback function that you specified in the `grantOfflineAccess().then()` method is passed a JSON object with an authorization code. For example:

```
{ "code": "4/yU4cQZTMnnMtetyFcIWNItG32eKxxxgXXX-Z4yyJJJo. 4qHskT-
UtugceFc0ZRONyF4z7U4UmAI" }
```

Step 6: Send the authorization code to the server

The code is your one-time code that your server can exchange for its own access token and refresh token. You can only obtain a refresh token after the user has been presented an authorization dialog requesting offline access. You must store the refresh token that you retrieve for later use because subsequent exchanges will return null for the refresh token. This flow provides increased security over your standard OAuth 2.0 flow.

Access tokens are always returned with the exchange of a valid authorization code.

The following script defines a callback function for the sign-in button. When a sign-in is successful, the function stores the access token for client-side use and sends the one-time code to your server on the same domain.

```
<!-- Last part of BODY element in file index. html -->
<script>
functionsignInCallback(authResult){
  if(authResult['code']){

    // Hide the sign-in button now that the user is authorized, for example:
    $('#signinButton'). attr('style','display: none');

    // Send the code to the server
    $. ajax({
      type:'POST',
      url:'http://example. com/storeauthcode',
      // Always include an `X-Requested-With` header in every AJAX request,
      // to protect against CSRF attacks.
```

```
headers:{
  'X-Requested-With':'XMLHttpRequest'
},
contentType:'application/octet-stream; charset=utf-8',
success:function(result){
  // Handle or verify the server response.
},
processData:false,
data:authResult['code']
});
}else{
  // There was an error.
}
}
</script>
```

Step 7: Exchange the authorization code for an access token

On the server, exchange the auth code for access and refresh tokens. Use the access token to call Google APIs on behalf of the user and, optionally, store the refresh token to acquire a new access token when the access token expires.

If you requested profile access, you also get an ID token that contains basic profile information for the user.

For example:

```
// (Receive authCode via HTTPS POST)

if(request.getHeader('X-Requested-With')==null){
  // Without the `X-Requested-With` header, this request could be forged.  Aborts.
}

// Set path to the Web application client_secret*. json file you downloaded from the
// Google API Console: https://console.developers.google.com/apis/credentials
// You can also find your Web application client ID and client secret from the
// console and specify them directly when you create the
GoogleAuthorizationCodeTokenRequest
// object.
String CLIENT_SECRET_FILE = "/path/to/client_secret.json";

// Exchange auth code for access token
GoogleClientSecretsclientSecrets=
  GoogleClientSecrets.load(
    JacksonFactory.getDefaultInstance(),newFileReader(CLIENT_SECRET_FILE));
GoogleTokenResponsetokenResponse=
```

```
newGoogleAuthorizationCodeTokenRequest(
    newNetHttpTransport(),
    JacksonFactory. getDefaultInstance(),
    "https://www. googleapis. com/oauth2/v4/token",
    clientSecrets. getDetails(). getClientId(),
    clientSecrets. getDetails(). getClientSecret(),
    authCode,
    REDIRECT_URI) // Specify the same redirect URI that you use with your web
                  // app. If you don't have a web version of your app, you can
                  // specify an empty string.
    . execute();

StringaccessToken=tokenResponse. getAccessToken();

// Use access token to call API
GoogleCredential credential =newGoogleCredential(). setAccessToken(accessToken);
Drive drive=
    newDrive. Builder(newNetHttpTransport(),JacksonFactory. getDefaultInstance(),
credential)
        . setApplicationName("Auth Code Exchange Demo")
        . build();
File file=drive. files(). get("appfolder"). execute();

// Get profile info from ID token
GoogleIdToken idToken=tokenResponse. parseIdToken();
GoogleIdToken.Payload payload =idToken. getPayload();
String userId=payload. getSubject(); // Use this value as a key to identify a user.
String email =payload. getEmail();
boolean emailVerified=Boolean. valueOf(payload. getEmailVerified());
String name =(String)payload. get("name");
String pictureUrl=(String)payload. get("picture");
String locale =(String)payload. get("locale");
String familyName=(String)payload. get("family_name");
String givenName=(String)payload. get("given_name");
```

Result:

Advanced Experiments

Ex No:

Date:

11. Programs using Web Service for Data storing and Retrieving

Aim:

Algorithm:

Google Cloud Storage Setup

Google Cloud Storage offers online storage tailored to an individual application's needs based on location, the frequency of access, and cost. Unlike Amazon Web Services, Google Cloud Storage uses a single API for high, medium, and low-frequency access.

Maven Dependency

We need to add a single dependency to our *pom.xml*:

```
<dependency>  
  <groupId>com. google. cloud</groupId>  
  <artifactId>google-cloud-storage</artifactId>  
  <version>1. 17. 0</version>  
</dependency>
```

Maven Central has the latest version of the library.

Create Authentication Key

Before we can connect to Google Cloud, we need to configure authentication. Google Cloud Platform (GCP) applications load a private key and configuration information from a JSON configuration file. We generate this file via the GCP console. Access to the console requires a valid Google Cloud Platform Account.

We create our configuration by:

1. Going to the Google Cloud Platform Console
2. If we haven't yet defined a GCP project, we click the *create* button and enter a project name, such as “*baeldung-cloud-tutorial*”
3. Select “*new service account*” from the drop-down list
4. Add a name such as “*baeldung-cloud-storage*” into the account name field.
5. Under “*role*” select Project, and then Owner in the submenu.
6. Select create, and the console downloads a private key file.

The role in step #6 authorizes the account to access project resources. For the sake of simplicity, we gave this account complete access to all project resources.

Install the Authentication Key

Next, we copy the file downloaded from GCP console to a convenient location and point the `GOOGLE_APPLICATION_CREDENTIALS` environment variable at it. This is the easiest way to load the credentials, although we'll look at another possibility below.

For Linux or Mac:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/file"
```

For Windows:

```
set GOOGLE_APPLICATION_CREDENTIALS="C:\path\to\file"
```

Install Cloud Tools

Google provides several tools for managing their cloud platform. We're going to use *gsutil* during this tutorial to read and write data alongside the API.

We can do this in two easy steps:

1. Install the Cloud SDK from the instructions [here](#) for our platform.
2. Follow the Quickstart for our platform [here](#). In step 4 of *Initialize the SDK*, we select the project name in step 4 of section 2. 2 above ("*baeldung-cloud-storage*" or whichever name you used).

gsutil is now installed and configured to read data from our cloud project.

Connecting to Storage and Creating a Bucket

Connect to Storage

Before we can use Google Cloud storage, we have to create a service object. If we've already set up the `GOOGLE_APPLICATION_CREDENTIALS` environment variable, we can use the default instance:

```
Storage storage = StorageOptions.getDefaultInstance().getService();
```

If we don't want to use the environment variable, we have to create a Credentials instance and pass it to Storage with the project name:

```
Credentials credentials = GoogleCredentials.fromStream(new
```

```
FileInputStream("path/to/file"));
```

```
Storage storage = StorageOptions.newBuilder().setCredentials(credentials)
    .setProjectId("baeldung-cloud-tutorial").
build().getService();
```

Creating a *Bucket*

Now that we're connected and authenticated, we can create a bucket. Buckets are containers that hold objects. They can be used to organize and control data access. There is no limit to the number of objects in a bucket. GCP limits the numbers of operations on buckets and **encourages application designers to emphasize operations on objects rather than on buckets.**

Creating a bucket requires a *BucketInfo*:

```
Bucket bucket = storage. create(BucketInfo. of("baeldung-bucket"));
```

For this simple example, we a bucket name and accept the default properties. Bucket names must be globally unique. If we choose a name that is already used, create() will fail.

Examining a *Bucket* with *gsutil*

Since we have a bucket now, we can take a examine it with *gsutil*. Let's open a command prompt and take a look:

```
$ gsutil ls -L -b gs://baeldung-1-bucket/
gs://baeldung-1-bucket/ :
Storage class:      STANDARD
Location constraint: US
Versioning enabled:  None
Logging configuration:  None
Website configuration:  None
CORS configuration:   None
Lifecycle configuration:  None
Requester Pays enabled:  None
Labels:             None
Time created:        Sun, 11 Feb 2018 21:09:15 GMT
Time updated:        Sun, 11 Feb 2018 21:09:15 GMT
Metageneration:      1
ACL:
[
  {
    "entity": "project-owners-385323156907",
    "projectTeam": {
      "projectNumber": "385323156907",
      "team": "owners"
    },
    "role": "OWNER"
  },
]
```

```
. . .
]
Default ACL:
[
  {
    "entity": "project-owners-385323156907",
    "projectTeam": {
      "projectNumber": "385323156907",
      "team": "owners"
    },
    "role": "OWNER"
  },
  . . .
]
```

Reading, Writing and Updating Data

In Google Cloud Storage, objects are stored in *Blobs*; *Blob* names can contain any Unicode character, limited to 1024 characters.

Writing Data

Let's save a *String* to our bucket:

```
1 String value = "Hello, World!";
2 byte[] bytes = value. getBytes(UTF_8);
3 Blob blob = bucket. create("my-first-blob", bytes);
```

As you can see, objects are simply arrays of *bytes* in the bucket, so we store a *String* by simply operating with its raw bytes.

Reading Data with *gsutil*

Now that we have a bucket with an object in it, let's take a look at *gsutil*.

Let's start by listing the contents of our bucket:

```
1 $ gsutil ls gs://baeldung-1-bucket/
2 gs://baeldung-1-bucket/my-first-blob
```

We passed *gsutil* the *ls* option again but omitted *-b* and *-L*, so we asked for a brief listing of objects. We receive a list of URIs for each object, which is one in our case.

Let's examine the object:

```
1 $ gsutil cat gs://baeldung-1-bucket/my-first-blob
```


2 Hello World!

Cat concatenates the contents of the object to standard output. We see the *String* we wrote to the *Blob*.

Reading Data

Blobs are assigned a *BlobId* upon creation.

The easiest way to retrieve a *Blob* is with the *BlobId*:

```
1 Blob blob = storage. get(blobId);
2 String value = new String(blob. getContent());
```

We pass the id to *Storage* and get the *Blob* in return, and *getContent()* returns the bytes.

If we don't have the *BlobId*, we can search the *Bucket* by name:

```
1 Page<Blob> blobs = bucket. list();
2 for (Blob blob: blobs. getValues()) {
3     if (name. equals(blob. getName())) {
4         return new String(blob. getContent());
5     }
6 }
```

Updating Data

We can update a *Blob* by retrieving it and then accessing its *WritableByteChannel*:

```
1 String newString = "Bye now!";
2 Blob blob = storage. get(blobId);
3 WritableByteChannel channel = blob. writer();
4 channel. write(ByteBuffer. wrap(newString. getBytes(UTF_8)));
5 channel. close();
```

Let's examine the updated object:

```
1 $ gsutil cat gs://baeldung-1-bucket/my-first-blob
2 Bye now!
```

Save an Object to File, Then Delete

Let's save the updated the object to a file:

```
1 $ gsutil copy gs://baeldung-1-bucket/my-first-blob my-first-blob
2 Copying gs://baeldung-1-bucket/my-first-blob. . .
3 / [1 files][ 9. 0 B/ 9. 0 B]
4 Operation completed over 1 objects/9. 0 B.
5 Grovers-Mill:~ egoebelbecker$ cat my-first-blob
6 Bye now!
```

As expected, the *copy* option copies the object to the filename specified on the command line.

gsutil can copy any object from Google Cloud Storage to the local file system, assuming there is enough space to store it.

We'll finish by cleaning up:

```
1 $ gsutilrm gs://baeldung-1-bucket/my-first-blob
2 Removing gs://baeldung-1-bucket/my-first-blob. . .
3 / [1 objects]
4 Operation completed over 1 objects.
5 $ gsutil ls gs://baeldung-1-bucket/
6 $
```

rm (*del* works too) deletes the specified object

Result:

Design Experiment

Ex No:

Date:

12. Programs using JSP with Database Connectivity for Banking App

Aim:

Algorithm:

1. Start program.
2. Create index and welcome page for start application
3. Create registration page for new customer registration.
4. Login page for account accessing for make transaction.
5. Statement page for process based statement generation.
6. Stop the process with successful logout.

Program:

Index Page

```
<%@ include file="logininc. jsp" %>
<html>
<head>
</head>
<body>
<br><br><br>
<table width="500px" align="center" style="background-color:ffeeff;">
<tr>
<td>&nbsp;</td>
</tr>
<tr>
<td style="font-weight:bold;font-size:20pt;" align="center">Welcome To Bank
Application</td>
</tr>
```

```
<tr>
<td>&nbsp;</td>
</tr>
<tr>
<td>&nbsp;</td>
</tr>
<%
if(session. getAttribute("userid")==null)
{
%>
<tr>
<td style="font-weight:bold;font-size:20pt;" align="center"><a href="login. jsp"
>Login</a></td>
</tr>
<tr>
<td style="font-weight:bold;font-size:20pt;" align="center"><a href="saveuser.
jsp">Register</a></td>
</tr>
<%
}
%>
</table>
</body>
</html>
```

Login Page:

```
<html>
<head>
<script src="valid. js" type="text/javascript"></script>
<style>
A:hover {
text-decoration: none;
font-family:arial;
font-size:12px;
```

```
color: #000000;
BORDER: none;
}
A {
text-decoration: underline;
font-family:arial;
font-size:12px;
color: #000000;
BORDER: none;
}
</style>
</head>
<body>
<table width="800px" border=0 align="center" >
<tr><td align="right"><a href="index. jsp" class="1"
style="color:#000000;">Home</a></td></tr>
</table>
<br><br><br>
<form name="loginform" method="post" action="loginmid. jsp" onsubmit="return
validLogin();">
<table width="250px" border=0 align="center" style="background-color:ffeeff;">
<tr>
<td colspan=2 align="center" style="font-weight:bold;font-size:20pt;"
align="center">User Login</td>
</tr>
<tr>
<td colspan=2>&nbsp;  </td>
</tr>
<tr>
<td style="font-size:12pt;" align="center">Login Name</td>
<td><input type="text" name="userName" value=""></td>
</tr>
<tr>
<td style="font-size:12pt;" align="center">Password</td>
```

```
        <td><input type="password" name="password" value=""></td>
    </tr>
    <tr>
        <td></td>
        <td><input type="submit" name="Submit" value="Login"></td>
    </tr>
    <tr>
        <td></td>
        <td align="right"><a href="saveuser. jsp">New User?</a></td>
    </tr>
</table>
</form>
</body>
</html>
```

Welcome Page:

```
<% @ include file="logininc. jsp" %>
<html>
<head>
<style>
A:hover {
text-decoration: none;font-family:arial;font-size:14px;
color: #000000;BORDER: none;
}
A {
text-decoration: underline;
font-family:arial;font-size:14px;
color: #000000;BORDER: none;
}
</style>
</head>
<body>
<br><br><br>
<table border=0 width="300px" align="center" style="background-color:ffeeff;">
```

```
<%  
if(session. getAttribute("userid")!=null)  
{  
String user = session. getAttribute("userid"). toString();  
%>  
<tr><td style="font-weight:bold;font-size:20pt;" align="center">Welcome  
<%= user%></td></tr><tr><td align="right">  
&nbsp;</td></tr><tr><td align="center">  
<a href="saveuser. jsp?flag=edit">Update profile</a>  
</td></tr>  
<tr><td align="right">&nbsp;</td></tr>  
<tr><td align="center">  
<a href="cash. jsp">Cash Transaction</a>  
</td></tr>  
<tr><td align="center">&nbsp;</td></tr><tr><td align="center">  
<a href="viewstatement. jsp">View Transaction</a>  
</td></tr><tr><td align="center">  
&nbsp;</td></tr>  
<% } %>  
</table>  
</body>  
</html>
```

Result:

Open Ended Experiment

Ex No:

Date:

13.Study of AWS Technology

Aim:

1. Simple Website Hosting

Simple websites typically consist of a single web server which runs either a Content Management System (CMS), such as Word Press, an e Commerce application, such as Magento, or a development stack, like LAMP. The software makes it easy to build, update, manage, and serve the content of your website.

Simple websites are best for low to medium trafficked sites with multiple authors and more frequent content changes, such as marketing websites, content websites or blogs. They provide a simple starting point for website which might grow in the future. While typically low cost, these sites require IT administration of the web server and are not built to be highly available or scalable beyond a few servers.

Best for:

- Websites built on common applications like Word Press, Joomla, Drupal, Magento
- Websites built on popular development stacks like LAMP, LEMP, MEAN, Node. Js
- Websites that are unlikely to scale beyond 5 servers
- Customers who want to manage their own web server and resources

Customers who want one console to manage their web server, DNS, and networking Amazon Lightsail is the easiest way to launch and manage a Web server using AWS. Lightsail includes everything you need to jumpstart your Website – a virtual machine, SSD-based storage, data transfer, DNS management, and a static IP – for a low, predictable price.

Use Amazon Lightsail:

You can get started using Lightsail for your website with just a few clicks. Choose the operating system or application template that's best for your website, and your virtual private server is ready in less than a minute. You can easily manage your web server, DNS, and IP addresses directly from the Lightsail console.

1. Single Page Web App Hosting

Static web apps that require only a single load in a web browser are referred to as Single page web apps. All subsequent actions by the user are made available through HTML, JavaScript, and CSS that are pre-loaded in the browser. Backend data is accessed via GraphQL or REST APIs that fetch content from a data store and update the UI without requiring a page reload.

Single page web apps offer native or desktop app-like performance. They offer all the static website benefits (low cost, high levels of reliability, no server administration, and scalability to handle enterprise-level traffic) with dynamic functionality and blazing fast performance.

Best for:

- Websites built with Single page app frameworks such as React JS, Vue JS, Angular JS, and Nuxt.
- Websites built with static site generators such as Gatsby JS, React-static, Jekyll, and Hugo.
- Progressive web apps or PWAs
- Websites that do not contain server-side scripting, like PHP or ASP. NET
- Websites that have serverless back ends.

Use AWS Amplify Console:

The AWS Amplify Console provides a complete workflow for developing, deploying, and hosting single page web apps or static sites with serverless backends. You can add dynamic functionality to your app with the Amplify Framework, and then deploy it to your end users instantly with the Amplify Console. The Amplify Console offers a number of features:

1. Continuous deployment allows you to deploy updates to your web app on every code commit.
2. Deploy your app to a global audience using our CDN, Amazon CloudFront.
3. Set up your custom domain with HTTPS automatically enabled in a single click.
4. Work on new features without impacting production users with feature branch deployments.

1. Simple Static Website Hosting

Static websites deliver HTML, JavaScript, images, video and other files to your website visitors and contain no server-side application code, like PHP or ASP. NET. They typically are used to deliver personal or marketing sites.

Static websites are very low cost, provide high-levels of reliability, require no server administration, and scale to handle enterprise-level traffic with no additional work.

Best for:

- Websites that do not contain server-side scripting, like PHP or ASP. NET
- Websites that change infrequently with few authors
- Websites need to scale for occasional intervals of high traffic
- Customers who do not want to manage infrastructure

Amazon S3 is object storage with a simple web service interface to store and retrieve any amount of data from anywhere on the web. It is designed to deliver 99. 999999999% durability and scale past trillions of objects worldwide.

To use S3 for a static website, you simply upload files to an S3 bucket and configure your S3 bucket for web hosting.

1. Enterprise Web Hosting

Enterprise websites include very popular marketing and media sites, as well as social, travel, and other application-heavy websites. For example, Lamborghini, Coursera, and Nordstrom use AWS to host their websites. Enterprise websites need to dynamically scale resources and be highly available to support the most demanding and highly trafficked websites.

Enterprise websites use multiple AWS services and often span multiple data centers (called Availability Zones). Enterprise websites built on AWS provide high levels of availability, scalability, and performance, but require higher amounts of management and administration than static or simple websites.

Best for:

- Websites that use multiple web servers across at least two data centers
- Websites that need to scale using load balancing, autoscaling, or external databases
- Websites that require sustained high CPU utilization
- Customers who need maximum control and flexibility for their web server configuration and administration.

Use Amazon Elastic Cloud Computing (Amazon EC2):

Amazon EC2 provides resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers and allows maximum scalability and availability for websites and web applications. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use.

To use EC2 to host your website, you will need to create and configure an EC2 instance in the AWS Management Console.

Result:

Open Ended Experiment:

Ex No:
Date:

14.Study of Simple Secure Storage in AWS Technology

Aim:

General S3 FAQs

Q: What is Amazon S3?

Amazon S3 is object storage built to store and retrieve any amount of data from anywhere on the Internet. It's a simple storage service that offers an extremely durable, highly available, and infinitely scalable data storage infrastructure at very low costs.

Q: What can I do with Amazon S3?

Amazon S3 provides a simple web service interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. Using this web service, you can easily build applications that make use of Internet storage. Since Amazon S3 is highly scalable and you only pay for what you use, you can start small and grow your application as you wish, with no compromise on performance or reliability.

Q: How can I get started using Amazon S3?

To sign up for Amazon S3, click [this link](#). You must have an Amazon Web Services account to access this service; if you do not already have one, you will be prompted to create one when you begin the Amazon S3 sign-up process. After signing up, please refer to the Amazon S3 documentation and sample code in the [Resource Center](#) to begin using Amazon S3.

Q: What can developers do with Amazon S3 that they could not do with an on-premises solution?

Amazon S3 enables any developer to leverage Amazon's own benefits of massive scale with no up-front investment or performance compromises. Developers are now free to innovate knowing that no matter how successful their businesses become, it will be inexpensive and simple to ensure their data is quickly accessible, always available, and secure.

Q: What kind of data can I store in Amazon S3?

You can store virtually any kind of data in any format. Please refer to the Amazon Web Services Licensing Agreement for details.

Q: How much data can I store in Amazon S3?

The total volume of data and number of objects you can store are unlimited. Individual Amazon S3 objects can range in size from a minimum of 0 bytes to a maximum of 5 terabytes. The largest object that can be uploaded in a single PUT is 5 gigabytes. For objects larger than 100 megabytes, customers should consider using the [Multipart Upload](#) capability.

Q: What storage classes does Amazon S3 offer?

Amazon S3 offers a range of storage classes designed for different use cases. These include S3 Standard for general-purpose storage of frequently accessed data; S3 Intelligent-

Tiering for data with unknown or changing access patterns; S3 Standard-Infrequent Access (S3 Standard-IA) and S3 One Zone-Infrequent Access (S3 One Zone-IA) for long-lived, but less frequently accessed data; and Amazon S3 Glacier (S3 Glacier) and Amazon S3 Glacier Deep Archive (S3 Glacier Deep Archive) for long-term archive and digital preservation. You can learn more about these storage classes on the [Amazon S3 Storage Classes page](#).

Q: What does Amazon do with my data in Amazon S3?

Amazon will store your data and track its associated usage for billing purposes. Amazon will not otherwise access your data for any purpose outside of the Amazon S3 offering, except when required to do so by law. Please refer to the [Amazon Web Services Licensing Agreement](#) for details.

Q: Does Amazon store its own data in Amazon S3?

Yes. Developers within Amazon use Amazon S3 for a wide variety of projects. Many of these projects use Amazon S3 as their authoritative data store and rely on it for business-critical operations.

Q: How is Amazon S3 data organized?

Amazon S3 is a simple key-based object store. When you store data, you assign a unique object key that can later be used to retrieve the data. Keys can be any string, and they can be constructed to mimic hierarchical attributes. Alternatively, you can use S3 Object Tagging to organize your data across all of your S3 buckets and/or prefixes.

Q: How do I interface with Amazon S3?

Amazon S3 provides a simple, standards-based REST web services interface that is designed to work with any Internet-development toolkit. The operations are intentionally made simple to make it easy to add new distribution protocols and functional layers.

Q: How reliable is Amazon S3?

Amazon S3 gives any developer access to the same highly scalable, highly available, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The S3 Standard storage class is designed for 99.99% availability, the S3 Standard-IA storage class is designed for 99.9% availability, and the S3 One Zone-IA storage class is designed for 99.5% availability. All of these storage classes are backed by the [Amazon S3 Service Level Agreement](#).

Q: How will Amazon S3 perform if traffic from my application suddenly spikes?

Amazon S3 was designed from the ground up to handle traffic for any Internet application. Pay-as-you-go pricing and unlimited capacity ensures that your incremental costs don't change and that your service is not interrupted. Amazon S3's massive scale enables us to spread load evenly, so that no individual application is affected by traffic spikes.

Q: Does Amazon S3 offer a Service Level Agreement (SLA)?

Yes. The [Amazon S3 SLA](#) provides for a service credit if a customer's monthly uptime percentage is below our service commitment in any billing cycle.

Result: