Module -I

Week 1: List Manipulation

1. Write Java Script code to add a new item to an existing HTML list.
2. Create a function that removes an item from a list when a button is clicked.

Week 2: Mouse Events

1. Create a java Script program that changes the background color of an element when the mouse hovers over it.
2. Develop an image Zoom feature that zooms in when the mouse is over an image and zooms out when the mouse leaves.

Week 3: DOM Manipulation

1. Build a “To-Do List” application that allows users to add, edit and delete tasks using DOM manipulation.

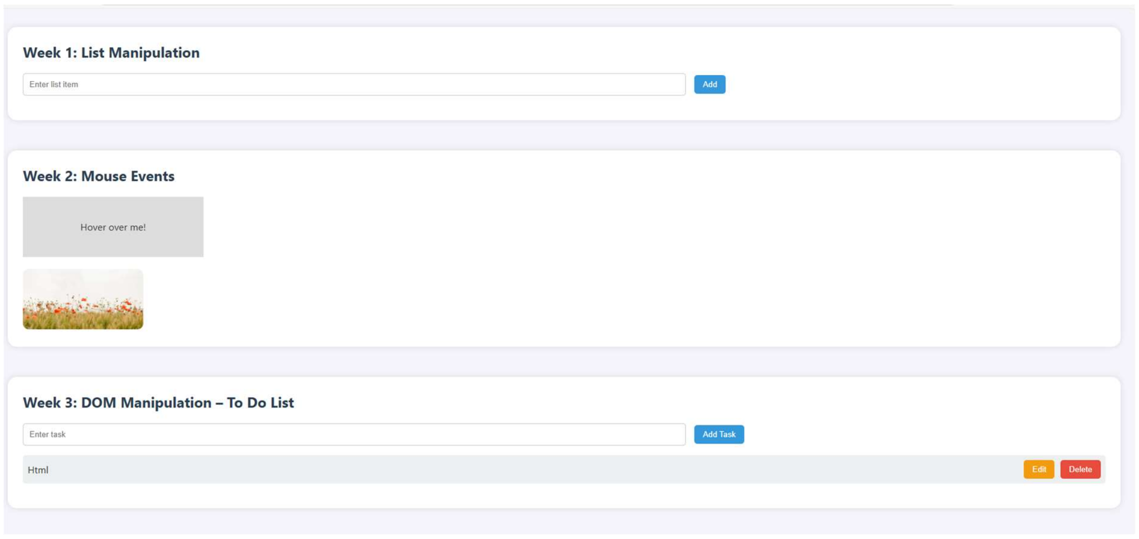
<u>program:</u>

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title> Full Stack Web Dev Labs</title>
 <style>
  body {
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
   margin: 30px;
   background: #f4f4f9;
   color: #333;
  }
  section {
   margin-bottom: 50px;
}

  padding: 25px;
  border-radius: 15px;
  background: #fff;
  box-shadow: 0 0 10px rgba(0,0,0,0.1);
}
h2 {
  color: #2c3e50;
  margin-top: 0;
}
input[type="text"] {
  padding: 10px;
  width: 60%;
  margin-right: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
}
button {
  padding: 8px 14px;
  border: none;
  border-radius: 5px;
  background-color: #3498db;
  color: white;
  cursor: pointer;
}
button:hover {
  background-color: #2980b9;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  padding: 8px;
  margin-bottom: 10px;
  background: #ecf0f1;
  border-radius: 5px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}
.zoom-image {
  transition: transform 0.3s ease-in-out;
  width: 200px;
  height: 100px;
  border-radius: 10px;
}
.zoom-container {
  width: 200px;
  overflow: hidden;
}
.zoom-container:hover .zoom-image {
  transform: scale(1.2); /* Zoom in */
}
.actions button {
  margin-left: 10px;
  background-color: #2ecc71;
```

```
    }
    .actions button.delete {
      background-color: #e74c3c;
    }
    .actions button.edit {
      background-color: #f39c12;
    }
  </style>
</head>
<body>

  <!-- Week 1: List Manipulation -->
  <section>
    <h2>Week 1: List Manipulation</h2>
    <input type="text" id="listInput" placeholder="Enter list item">
    <button onclick="addListItem()">Add</button>
    <ul id="listItems"></ul>
    <script>
      function addListItem() {
        const input = document.getElementById("listInput");
        if (input.value.trim()) {
          const li = document.createElement("li");

          const textSpan = document.createElement("span");
          textSpan.textContent = input.value;

          const delBtn = document.createElement("button");
          delBtn.textContent = "Delete";
          delBtn.className = "delete";

          delBtn.onclick = () => li.remove();


          li.appendChild(textSpan);

          li.appendChild(delBtn);

          document.getElementById("listItems").appendChild(li);

          input.value = "";

        }

      }

    </script>

  </section>


  <!-- Week 2: Mouse Events -->

  <section>

    <h2>Week 2: Mouse Events</h2>

    <div id="hoverBox" style="width:300px;height:100px;line-height:100px;text-
align:center;background:#ddd; margin-bottom: 20px;">

      Hover over me!

    </div>


    <div class="zoom-container">

        <img src="https://images.unsplash.com/photo-1465146344425-
f00d5f5c8f07?w=600&auto=format&fit=crop&q=60&ixlib=rb-
4.1.0&ixid=M3wxMjA3fDB8MHxzZWFyY2h8M3x8bmF0dXJlfGVufDB8fDB8fHww"
alt="Zoomable Image" class="zoom-image">

    </div>


    <script>

      const hoverBox = document.getElementById("hoverBox");
```

```
hoverBox.addEventListener("mouseover", () => hoverBox.style.background = "#a29bfe");
hoverBox.addEventListener("mouseout", () => hoverBox.style.background = "#ddd");
  </script>
 </section>

 <!-- Week 3: To-Do List with Full DOM Manipulation -->
 <section>
  <h2>Week 3: DOM Manipulation – To Do List</h2>
  <input type="text" id="taskInput" placeholder="Enter task">
  <button onclick="addTask()">Add Task</button>
  <ul id="taskList"></ul>
  <script>
   function addTask() {
    const taskInput = document.getElementById("taskInput");
    const taskText = taskInput.value.trim();
    if (taskText) {
     const li = document.createElement("li");
     const span = document.createElement("span");
     span.textContent = taskText;
     li.appendChild(span);

     const actions = document.createElement("span");
     actions.className = "actions";

     const editBtn = document.createElement("button");
     editBtn.textContent = "Edit";
     editBtn.className = "edit";
     editBtn.onclick = () => {
       const newTask = prompt("Edit task:", span.textContent);
       if (newTask !== null) span.textContent = newTask;
     };

     const deleteBtn = document.createElement("button");
     deleteBtn.textContent = "Delete";
     deleteBtn.className = "delete";
     deleteBtn.onclick = () => li.remove();

     actions.append(editBtn, deleteBtn);
     li.appendChild(actions);
     document.getElementById("taskList").appendChild(li);
     taskInput.value = "";
    }
   }
  </script>
 </section>

</body>
</html>
```

Output:



Module -II

WEEK :4

1. Create an XML document that represents a simple address book with names,phone numbers and email addresses for multiple contacts.

Example file — address-book.xml

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

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

        xsi:noNamespaceSchemaLocation="address-book.xsd"

        lastUpdated="2025-08-01">

 <Contact id="C001" type="personal">

  <Name>John Doe</Name>

  <Phones>

   <Phone countryCode="+1" type="mobile">+1-9876543210</Phone>

   <Phone countryCode="+1" type="home">+1-2125550123</Phone>

  </Phones>

  <Emails>

   <Email type="personal">john.doe@example.com</Email>

   <Email type="work">john.doe@acme.com</Email>

  </Emails>
```

```xml
  <Address>
   <Street>123 Main St</Street>
   <City>New York</City>
   <State>NY</State>
   <Zip>10001</Zip>
   <Country>USA</Country>
  </Address>
 </Contact>

 <Contact id="C002" type="work">
  <Name>Jane Smith</Name>
  <Phones>
   <Phone countryCode="+91" type="mobile">+91-9876543211</Phone>
  </Phones>
  <Emails>
   <Email type="work">jane.smith@company.com</Email>
  </Emails>
  <Address>
   <Street>456 Park Ave</Street>
   <City>Mumbai</City>
   <State>Maharashtra</State>
   <Zip>400001</Zip>
   <Country>India</Country>
  </Address>
 </Contact>
</AddressBook>
```

2. Add elements and attributes to represent different contact details.

Refer above lab (4.1)

Contact has attributes id (unique) and type (personal/work).

- Phones contains multiple Phone elements; each Phone has attributes countryCode and type.
- Emails contains Email elements with type.
- Address is a nested element grouping address fields.
- Root has lastUpdated metadata attribute.

This structure shows how attributes and nested elements serve different semantic roles:

- attributes = metadata/short flags (id, type)
- elements = structured content (address, multi-valued lists)

Lab 4.3 — Validation with XSD

Example XSD — address-book.xsd (enforces presence & basic structure):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="AddressBook">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Contact" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="Name" type="xs:string"/>
       <xs:element name="Phones" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Phone" maxOccurs="unbounded">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:string">
              <xs:attribute name="countryCode" type="xs:string" use="required"/>
              <xs:attribute name="type" type="xs:string" use="optional"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>

       <xs:element name="Emails" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Email" type="xs:string" maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>

       <xs:element name="Address" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Street" type="xs:string" minOccurs="0"/>
          <xs:element name="City" type="xs:string"/>
          <xs:element name="State" type="xs:string" minOccurs="0"/>
          <xs:element name="Zip" type="xs:string" minOccurs="0"/>
          <xs:element name="Country" type="xs:string" minOccurs="0"/>
         </xs:sequence>
        </xs:complexType>
```

```
       </xs:element>


      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="type" type="xs:string" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="lastUpdated" type="xs:date" use="optional"/>
  </xs:complexType>
 </xs:element>


</xs:schema>
```

How to create & validate (tools & commands)

- Use any text editor (VS Code, Notepad++, Sublime) or XML editors (Oxygen, XMLSpy).
- **Command-line validation (xmllint)**: xmllint --noout --schema address-book.xsd address-book.xml
    - Output address-book.xml validates on success.

Week 5

Lab 5.1 — Choose language & parser (recommendation)

- **Java** (recommended for your course): DOM + XPath for clarity, StAX for streaming (very large files), SAX for event-based parsing. Also Jackson XmlMapper and JAXB for object mapping.
- **Python**: xml.etree.ElementTree for small/medium files, lxml for full XPath & validation, xmltodict for quick mapping to dicts.

Lab 5.2 — Read XML and extract specific data (detailed solutions)

We'll demonstrate extraction of:

- all work emails,
- all phone numbers with countryCode="+91",
- contact names in City = Mumbai,
- retrieve contact XML by id.

A — Java: DOM + XPath (concise & powerful)

File: XPathExtract.java

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.xpath.*;
import org.w3c.dom.*;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class XPathExtract {
  private final Document doc;
  private final XPath xpath;

  public XPathExtract(File xmlFile) throws Exception {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(false);
    DocumentBuilder db = dbf.newDocumentBuilder();
    this.doc = db.parse(xmlFile);
    this.xpath = XPathFactory.newInstance().newXPath();
  }

  // 1. All work emails
  public List<String> getWorkEmails() throws XPathExpressionException {
        String expr = "/AddressBook/Contact[@type='work']/Emails/Email/text()";
        NodeList nl = (NodeList) xpath.evaluate(expr, doc, XPathConstants.NODESET);
        List<String> out = new ArrayList<>();
        for (int i = 0; i < nl.getLength(); i++) out.add(nl.item(i).getNodeValue());
        return out;
    }

    // 2. Phones with countryCode
    public List<String> getPhonesByCountry(String countryCode) throws
XPathExpressionException {
        String expr =
String.format("/AddressBook/Contact/Phones/Phone[@countryCode='%s']/text()",
countryCode);
        NodeList nl = (NodeList) xpath.evaluate(expr, doc, XPathConstants.NODESET);
        List<String> out = new ArrayList<>();
        for (int i = 0; i < nl.getLength(); i++) out.add(nl.item(i).getNodeValue());
        return out;
    }

    // 3. Contact names in a city
    public List<String> getContactsInCity(String city) throws XPathExpressionException {
        String expr =
String.format("/AddressBook/Contact[Address/City/text()='%s']/Name/text()", city);
        NodeList nl = (NodeList) xpath.evaluate(expr, doc, XPathConstants.NODESET);
        List<String> out = new ArrayList<>();
        for (int i = 0; i < nl.getLength(); i++) out.add(nl.item(i).getNodeValue());
        return out;
    }
```

```
  // 4. Get contact XML by id (as formatted string)

  public String getContactXmlById(String id) throws Exception {

    Node node = (Node)
xpath.evaluate(String.format("/AddressBook/Contact[@id='%s']", id), doc,
XPathConstants.NODE);

    if (node == null) return null;

    javax.xml.transform.Transformer t =
javax.xml.transform.TransformerFactory.newInstance().newTransformer();

    t.setOutputProperty(javax.xml.transform.OutputKeys.INDENT, "yes");

    java.io.StringWriter sw = new java.io.StringWriter();

    t.transform(new javax.xml.transform.dom.DOMSource(node), new
javax.xml.transform.stream.StreamResult(sw));

    return sw.toString();

  }


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

    XPathExtract ex = new XPathExtract(new File("address-book.xml"));

    System.out.println("Work emails: " + ex.getWorkEmails());

    System.out.println("Phones +91: " + ex.getPhonesByCountry("+91"));

    System.out.println("Contacts in Mumbai: " + ex.getContactsInCity("Mumbai"));

    System.out.println("Contact C001 xml:\n" + ex.getContactXmlById("C001"));

  }

}
```

Run: compile and run with Java 11+. Output will be lists printed in console. This is clear for lab demonstration.

Note:

- XPath expressions are concise and safe for queries.
- Document loads entire XML — suitable for small/medium-sized files.

B — Java: StAX streaming (for very large files)

Use when XML may be huge (memory-limited environment).

File: StaxPhoneExtractor.java

```
import javax.xml.stream.*;

import javax.xml.stream.events.*;

import java.io.FileInputStream;

import java.util.ArrayList;

import java.util.List;


public class StaxPhoneExtractor {

  public static List<String> phonesForCountry(String filePath, String countryCode)
throws Exception {

    List<String> phones = new ArrayList<>();

    XMLInputFactory factory = XMLInputFactory.newInstance();

    try (FileInputStream fis = new FileInputStream(filePath)) {

      XMLEventReader reader = factory.createXMLEventReader(fis);

      boolean inPhone = false;

      String currentCountry = null;

      while (reader.hasNext()) {

        XMLEvent ev = reader.nextEvent();

        if (ev.isStartElement()) {

          StartElement se = ev.asStartElement();

          if ("Phone".equals(se.getName().getLocalPart())) {

            inPhone = true;

            Attribute attr = se.getAttributeByName(new
javax.xml.namespace.QName("countryCode"));

            currentCountry = (attr == null) ? null : attr.getValue();

          }

        } else if (ev.isCharacters() && inPhone) {

          String text = ev.asCharacters().getData().trim();
```

```
            if (!text.isEmpty() &&
```

Python Program to Extract and Display Data:

Python

```
import xml.etree.ElementTree as ET

def extract_and_display_xml_data(xml_file_path):
  """
  Reads an XML file, extracts specific data (book titles and authors),
  and displays it in a structured format.


  Args:
    xml_file_path (str): The path to the XML file.
  """
  try:
    tree = ET.parse(xml_file_path)
    root = tree.getroot()

    print("Extracted Book Information:")
    print("--------------------------")

    for book in root.findall('book'):
      book_id = book.get('id')
      title = book.find('title').text
      author = book.find('author').text
      genre = book.find('genre').text
      price = book.find('price').text

      print(f"Book ID: {book_id}")
      print(f"  Title: {title}")
      print(f"  Author: {author}")
      print(f"  Genre: {genre}")
      print(f"  Price: ${price}\n")

  except FileNotFoundError:
    print(f"Error: XML file not found at '{xml_file_path}'")
  except Exception as e:
    print(f"An error occurred: {e}")

# Specify the path to your XML file
xml_file = 'data.xml'

# Call the function to extract and display data
extract_and_display_xml_data(xml_file)
```

Week 6:

1.Take an existing XML document and write a program to convert it into JSON format

Example XML Document (data.xml):

```
<bookstore>
 <book category="fiction">
  <title lang="en">The Great Gatsby</title>
  <author>F. Scott Fitzgerald</author>
  <year>1925</year>
  <price>10.99</price>
 </book>
 <book category="science">
  <title lang="en">Cosmos</title>
  <author>Carl Sagan</author>
  <year>1980</year>
  <price>15.50</price>
 </book>
</bookstore>
```

Python Program (xml_to_json.py):

```
import xmltodict

import json

def convert_xml_to_json(xml_file_path, json_file_path):
```

```
"""
Converts an XML file to a JSON file.
"""
try:
    with open(xml_file_path, 'r', encoding='utf-8') as xml_file:
        xml_content = xml_file.read()

    # Convert XML to Python dictionary
    ordered_dict = xmltodict.parse(xml_content)

    # Convert Python dictionary to JSON string
    json_content = json.dumps(ordered_dict, indent=4)

    with open(json_file_path, 'w', encoding='utf-8') as json_file:
        json_file.write(json_content)

    print(f"Successfully converted '{xml_file_path}' to '{json_file_path}'")

except FileNotFoundError:
    print(f"Error: The file '{xml_file_path}' was not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

2. Compare the XML and JSON representation of the data.

XML (Extensible Markup Language):

- Structure:

Uses a tag-based, hierarchical structure with opening and closing tags to define elements and attributes.

- Readability:

Can be verbose due to repetitive tags, potentially affecting readability for complex data.

- Data Types:

Does not inherently support distinct data types (e.g., numbers, booleans) and treats all content as character data.

- Parsing:

Requires dedicated XML parsers to process the data.

JSON (JavaScript Object Notation):

- Structure:

Uses a lightweight, key-value pair structure with objects ({}) and arrays ([]).

- Readability:

Generally more concise and easier to read, especially for nested data, due to its less verbose syntax.

- Data Types:

Natively supports various data types, including strings, numbers, booleans, arrays, and nested objects.

- Parsing:

Can be parsed directly by standard JavaScript functions and is widely supported in other programming languages.

Key Differences in Representation:

- Syntax: XML relies on tags, while JSON uses key-value pairs and array notation.
- Verbosity: XML is typically more verbose than JSON for the same data set.
- Data Type Handling: JSON explicitly supports data types, whereas XML treats data as text unless schema definitions are used.
- Arrays: JSON has native array support, while XML represents lists of items through repeated elements.
- Attributes vs. Elements: XML distinguishes between attributes and child elements, while JSON typically represents both as key-value pairs within an object.

Week 7:

1.Create an XML Document representing a collection of items (Ex: list of books, movies or products)

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

<books>

  <book id="B001">

    <title>The Great Gatsby</title>

    <author>F. Scott Fitzgerald</author>

    <genre>Classic</genre>

    <publication_year>1925</publication_year>

  </book>

  <book id="B002">

    <title>To Kill a Mockingbird</title>

    <author>Harper Lee</author>

    <genre>Fiction</genre>

    <publication_year>1960</publication_year>

  </book>

  <book id="B003">

    <title>1984</title>

    <author>George Orwell</author>

    <genre>Dystopian</genre>

    <publication_year>1949</publication_year>

  </book>

</books>
```

2. Write a program that allows users to perform CRUD(Create,Read,Update,Delete) operations on the items in XML document.

A program can be written in a language like Python using
the xml.etree.ElementTree module to perform CRUD operations on the XML document.

```
import xml.etree.ElementTree as ET
```

```
class XMLManager:
    def __init__(self, filename="books.xml"):
        self.filename = filename
        try:
            self.tree = ET.parse(self.filename)
            self.root = self.tree.getroot()
        except FileNotFoundError:
            self.root = ET.Element("books")
            self.tree = ET.ElementTree(self.root)
            self._save_xml()

    def _save_xml(self):
        self.tree.write(self.filename, encoding="UTF-8", xml_declaration=True)

    def create_book(self, book_id, title, author, genre, year):
        book = ET.SubElement(self.root, "book", id=book_id)
        ET.SubElement(book, "title").text = title
        ET.SubElement(book, "author").text = author
        ET.SubElement(book, "genre").text = genre
        ET.SubElement(book, "publication_year").text = year
        self._save_xml()
        print(f"Book '{title}' added.")

    def read_books(self):
        print("\n--- Current Books ---")
        for book in self.root.findall("book"):
            book_id = book.get("id")
            title = book.find("title").text
            author = book.find("author").text
            genre = book.find("genre").text
            year = book.find("publication_year").text
            print(f"ID: {book_id}, Title: {title}, Author: {author}, Genre: {genre}, Year: {year}")
        print("--------------------\n")

    def update_book(self, book_id, new_title=None, new_author=None,
new_genre=None, new_year=None):
        for book in self.root.findall("book"):
            if book.get("id") == book_id:
                if new_title:
                    book.find("title").text = new_title
                if new_author:
                    book.find("author").text = new_author
                if new_genre:
                    book.find("genre").text = new_genre
                if new_year:
                    book.find("publication_year").text = new_year
                self._save_xml()
                print(f"Book ID '{book_id}' updated.")
                return
        print(f"Book ID '{book_id}' not found.")

    def delete_book(self, book_id):
        for book in self.root.findall("book"):
            if book.get("id") == book_id:
                self.root.remove(book)
                self._save_xml()
            print(f"Book ID '{book_id}' deleted.")
            return
        print(f"Book ID '{book_id}' not found.")
# Example Usage:
if __name__ == "__main__":
    manager = XMLManager()
    # Create
    manager.create_book("B004", "Dune", "Frank Herbert", "Science Fiction", "1965")
    manager.create_book("B005", "Pride and Prejudice", "Jane Austen", "Romance",
"1813")
    # Read
    manager.read_books()

    # Update
    manager.update_book("B004", new_genre="Sci-Fi")
    # Read after update
    manager.read_books()
    # Delete
    manager.delete_book("B005")
    # Read after delete
    manager.read_books()
```

MODULE – III

WEEK 8 :

1.**Create a Servlet that connects to a MySQL database using JDBC and retrieves a list of employee names and their salaries from a table named employees. Implement the necessary steps for database connection and data retrieval, and explain the key components and methods used in your Servlet code.**

1. Database Setup

Create a table named employees in MySQL:

```
CREATE DATABASE companydb;

USE companydb;

CREATE TABLE employees (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL,
   salary DECIMAL(10,2) NOT NULL
);

INSERT INTO employees (name, salary) VALUES
('Alice', 55000.00),
('Bob', 60000.00),
('Charlie', 75000.00);
```

---

2. Directory Structure

```
FullStackLabs/
│
├── src/main/java/
│   ├── com.example.servlet/
│     └── EmployeeServlet.java
│
├── src/main/webapp/
│   ├── index.jsp
│   └── WEB-INF/web.xml
│
└── pom.xml
```

---

3. pom.xml (Maven Dependencies)

```
<dependencies>
  <!-- Servlet API -->
  <dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <version>6.0.0</version>
    <scope>provided</scope>
  </dependency>

  <!-- MySQL Connector -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
  </dependency>
```

```
</dependencies>
```

---

4. EmployeeServlet.java

```
package com.example.servlet;

import jakarta.servlet.*;
import jakarta.servlet.http.*;
import java.io.*;
import java.sql.*;
import java.util.*;

public class EmployeeServlet extends HttpServlet {

  private static final String JDBC_URL = "jdbc:mysql://localhost:3306/companydb";
  private static final String JDBC_USER = "root";
  private static final String JDBC_PASSWORD = "your_password";

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Employee List</title>");
    out.println("<style>table{border-collapse:collapse;} td,th{padding:8px;border:1px
solid #ccc;}</style>");
    out.println("</head><body>");
    out.println("<h2>Employee Names and Salaries</h2>");

    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
      try (Connection conn = DriverManager.getConnection(JDBC_URL, JDBC_USER,
JDBC_PASSWORD);
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT name, salary FROM employees")) {

        out.println("<table>");
        out.println("<tr><th>Name</th><th>Salary</th></tr>");
        while (rs.next()) {
          String name = rs.getString("name");
          double salary = rs.getDouble("salary");
          out.println("<tr><td>" + name + "</td><td>" + salary + "</td></tr>");
        }
        out.println("</table>");

      }
    } catch (Exception e) {
      out.println("<p style='color:red;'>Error: " + e.getMessage() + "</p>");
    }

    out.println("</body></html>");
  }
```

```
}
```

---

5. web.xml Configuration

```
<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_6_0.xsd"
     version="6.0">

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

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

---

6. Key Components Explained

✓ Class.forName("com.mysql.cj.jdbc.Driver") → Loads MySQL JDBC driver

✓ DriverManager.getConnection() → Establishes DB connection

✓ Statement and ResultSet → Executes SQL and retrieves results

✓ Servlet doGet() method → Handles HTTP GET requests and displays data in HTML

---

7. Output

When you access:

http://localhost:8080/FullStackLabs/employees

You’ll see a styled HTML table:

```
+---------+--------+
| Name    | Salary |
+---------+--------+
| Alice   | 55000  |
| Bob     | 60000  |
| Charlie | 75000  |
```

Week 9:

1.Write a Python program using a web framework (e.g., Flask or Django) that handles an HTTP GET request and responds with a simple "Hello, World!" message.

```
from flask import Flask
```

```
# Create a Flask application instance
app = Flask(__name__)


# Define a route for the root URL ("/") that handles GET requests
@app.route('/', methods=['GET'])
def hello_world():
    """
    This function is called when a GET request is made to the root URL.
    It returns the "Hello, World!" message.
    """
    return 'Hello, World!'


# Run the Flask application if the script is executed directly
if __name__ == '__main__':
    # Run the app in debug mode for development purposes.
    # This enables automatic reloading and provides a debugger.
    app.run(debug=True)
```

2. Implement a program that sends an HTTP POST request to a server with JSON data. Parse the JSON data on the server-side and respond with a success message.

Client-side (Python)

```
 import requests
import json


# Define the URL of the server endpoint
url = 'http://127.0.0.1:5000/data'


# Define the JSON data to send
data = {
  "name": "John Doe",
  "age": 30,
  "city": "New York"
}

# Set the headers to indicate JSON content
headers = {'Content-Type': 'application/json'}

try:
  # Send the POST request with JSON data
  response = requests.post(url, data=json.dumps(data), headers=headers)

  # Check if the request was successful (status code 200)
  if response.status_code == 200:
    print("POST request successful!")
    print("Server response:", response.json())
  else:
    print(f"POST request failed with status code: {response.status_code}")
    print("Server response:", response.text)

except requests.exceptions.ConnectionError as e:
  print(f"Error connecting to the server: {e}")
except Exception as e:
  print(f"An unexpected error occurred: {e}")
```

<u>Server-side (Python - Flask)</u>

```
from flask import Flask, request, jsonify
```

```
app = Flask(__name__)


@app.route('/data', methods=['POST'])
def receive_data():
  if request.is_json:
    data = request.get_json()
    print("Received JSON data:")
    print(f"Name: {data.get('name')}")
    print(f"Age: {data.get('age')}")
    print(f"City: {data.get('city')}")


    # Respond with a success message
    return jsonify({"message": "Data received and parsed successfully!"}), 200
  else:
    return jsonify({"error": "Request must be JSON"}), 400


if __name__ == '__main__':
  app.run(debug=True)
```

--o--

To run this example:

Install Flask and Requests.

Code

```
pip install Flask requests
```

- **Save the Server-Side Code:** Save the Flask code as a Python file (e.g., server.py).
- **Run the Server:** Execute python server.py in your terminal.
- **Save the Client-Side Code:** Save the client-side code as a Python file (e.g., client.py).

- **Run the Client:** Execute python client.py in a separate terminal.

The client will send the JSON data, and the server will receive, parse, and print it, then respond with a success message, which the client will then display.

MODULE – IV

Week 10:

1. **Develop a JSP Program to validate a particular user login based on the username password stored in the database and display a welcome page.**

   Prerequisites:

- A Java Development Kit (JDK) installed.
- An IDE like Eclipse or NetBeans.
- A database (e.g., MySQL, MariaDB) with a user table.
- A web server (e.g., Apache Tomcat).
- The JDBC driver for your database. You must place the .jar file in your project's WEB-INF/lib directory.

Step 1: Set up the database

First, create a table to store user credentials.

sql

```
CREATE DATABASE userdb;

USE userdb;

CREATE TABLE users (
  username VARCHAR(50) NOT NULL PRIMARY KEY,
  password VARCHAR(50) NOT NULL
);

INSERT INTO users (username, password) VALUES ('testuser', 'password123');
INSERT INTO users (username, password) VALUES ('admin', 'adminpass');
```

Use code with caution.

Step 2: Create the login form (login.jsp)

This page displays a form for the user to enter their credentials.

login.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Login Page</title>
<style>
  body { font-family: Arial, sans-serif; text-align: center; }
  .container { width: 300px; margin: 100px auto; padding: 20px; border: 1px solid
#ccc; border-radius: 5px; }
  input[type="text"], input[type="password"] { width: 90%; padding: 10px;
margin: 10px 0; border: 1px solid #ccc; }
  input[type="submit"] { background-color: #4CAF50; color: white; padding:
10px 15px; border: none; cursor: pointer; }
  .error { color: red; margin-top: 10px; }
</style>
</head>
<body>
  <div class="container">
    <h2>User Login</h2>
    <form action="ValidateUser.jsp" method="post">
      <p>
        <input type="text" name="username" placeholder="Username" required
/>
      </p>
      <p>
        <input type="password" name="password" placeholder="Password"
required />
      </p>
      <p>
        <input type="submit" value="Login" />
      </p>
    </form>
    <%
      // Display an error message if the login failed
      String error = request.getParameter("error");
      if ("invalid".equals(error)) {
        out.println("<p class='error'>Invalid username or password.</p>");
        }
      %>
   </div>
</body>
</html>
```

Step 3: Create the validation logic (ValidateUser.jsp)

This JSP page retrieves form data, connects to the database, and checks for a matching user. The code includes necessary imports for SQL operations, retrieves username and password parameters, establishes a database connection using JDBC, prepares a SQL statement to prevent injection, executes the query, and checks if a user is found. It also handles potential exceptions and closes database resources. Based on the validation result, it either sets a session attribute and redirects to the welcome page or redirects back to the login page with an error parameter.

The full code for ValidateUser.jsp can be found in the referenced web document.

Step 4: Create the welcome page (welcome.jsp)

This page is displayed after a successful login and shows a personalized welcome message. It checks for a username in the session and displays a welcome message if found. Otherwise, it redirects to the login page. It also includes a link to a logout page.

The full code for welcome.jsp can be found in the referenced web document.

Step 5: Implement the logout functionality (logout.jsp)

This page invalidates the user's session and redirects them to the login page.

The full code for logout.jsp can be found in the referenced web document.

Step 6: Deploy and run

1. Place the files: Put login.jsp, ValidateUser.jsp, welcome.jsp, and logout.jsp in your web server's application directory (e.g., webapps/your_project_name/).
2. Start the server: Start your Apache Tomcat or other web server.
3. Access the application: Open a web browser and navigate to http://localhost:8080/your_project_name/login.jsp.
   This will show the login form, and upon entering valid credentials, it will display the welcome page. Incorrect credentials will result in an error message.

2.Develop a JSP Program to validate a particular user login based on the username password stored in the database and display a welcome page.

Database Setup

Create a table users in MySQL:

```
CREATE DATABASE userdb;

USE userdb;

CREATE TABLE users (
   id INT PRIMARY KEY AUTO_INCREMENT,
   username VARCHAR(50) UNIQUE NOT NULL,
   password VARCHAR(255) NOT NULL  -- store hashed passwords ideally
);

-- Insert sample users (password stored as plain text for demo)
INSERT INTO users (username, password) VALUES ('admin', 'admin123'), ('john',
'john123');
```

Project Structure

```
WebApp/
 ├── index.jsp        (Login Form)
 ├── welcome.jsp      (Welcome Page)
 ├── error.jsp        (Error Page)
 ├── LoginServlet.java  (Handles validation)
 ├── DBConnection.java  (Database utility)
```

```
└── WEB-INF/
    └── web.xml      (Servlet configuration)
```

---

1. index.jsp (Login Form)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <title>User Login</title>
  <style>
    body { font-family: Arial; background: #f4f4f4; text-align: center; }
    form { margin-top: 100px; display: inline-block; background: #fff; padding: 20px;
border-radius: 8px; box-shadow: 0 0 10px #ccc; }
    input { display: block; margin: 10px auto; padding: 10px; width: 80%; }
    button { background: #28a745; color: #fff; border: none; padding: 10px; width: 80%;
cursor: pointer; }
  </style>
</head>
<body>
  <h2>Login Page</h2>
  <form action="LoginServlet" method="post">
    <input type="text" name="username" placeholder="Enter Username" required />
    <input type="password" name="password" placeholder="Enter Password" required
/>
```

```
    <button type="submit">Login</button>
  </form>
</body>
</html>
```

---

2. DBConnection.java (Database Utility Class)

```
import java.sql.Connection;
import java.sql.DriverManager;

public class DBConnection {
  private static final String URL = "jdbc:mysql://localhost:3306/userdb";
  private static final String USER = "root";
  private static final String PASSWORD = "yourpassword";

  public static Connection getConnection() throws Exception {
    Class.forName("com.mysql.cj.jdbc.Driver");
    return DriverManager.getConnection(URL, USER, PASSWORD);
  }
}
```

---

3. LoginServlet.java (Validation Logic)

```
import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;


@WebServlet("/LoginServlet")

public class LoginServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)

      throws ServletException, IOException {

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

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


    try (Connection conn = DBConnection.getConnection()) {

      String sql = "SELECT * FROM users WHERE username=? AND password=?";

      PreparedStatement stmt = conn.prepareStatement(sql);

      stmt.setString(1, username);

      stmt.setString(2, password);


      ResultSet rs = stmt.executeQuery();

      if (rs.next()) {

        request.setAttribute("username", username);

        request.getRequestDispatcher("welcome.jsp").forward(request, response);
```

```
            } else {

                response.sendRedirect("error.jsp");

            }

        } catch (Exception e) {

            e.printStackTrace();

            response.sendRedirect("error.jsp");

        }

    }

}
```

---

4. welcome.jsp (Welcome Page)

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

<!DOCTYPE html>

<html>

<head>

    <title>Welcome</title>

</head>

<body>

    <h2>Welcome, ${username}!</h2>

    <p>You have successfully logged in.</p>

</body>

</html>
```

---

5. error.jsp (Error Page)

```
<!DOCTYPE html>
<html>
<head>
  <title>Login Failed</title>
</head>
<body>
  <h3 style="color:red;">Invalid username or password!</h3>
  <a href="index.jsp">Try Again</a>
</body>
</html>
```

---

6. web.xml (Servlet Mapping)

```
<web-app>
  <servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
```

```
        <url-pattern>/LoginServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Week 11:

1.Write SQL Statements to create a table names “employees” with columns for ID, Name, Salary, and Department. Then, insert a few sample records into the table.

```
CREATE DATABASE companydb;
USE companydb;

CREATE TABLE employees (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Salary DECIMAL(10,2) NOT NULL,
    Department VARCHAR(50) NOT NULL
);

-- Insert sample records
INSERT INTO employees (Name, Salary, Department) VALUES
('Alice', 55000.00, 'IT'),
('Bob', 60000.00, 'Finance'),
('Charlie', 75000.00, 'HR'),

('David', 50000.00, 'IT'),

('Eva', 65000.00, 'HR');
```

2. **Write an SQL statement to update the salary of an employee with a specific ID in the Employees table.**

```
UPDATE employees

SET Salary = 80000.00

WHERE ID = 2;
```

✓ This updates the salary for the employee with ID 2 (Bob) to 80000.00.

3. Delete All Employees from the HR Department

```
DELETE FROM employees

WHERE Department = 'HR';
```

✓ This removes all employees whose department is HR.

4. Add a New Column Email to the employees Table

```
ALTER TABLE employees

ADD Email VARCHAR(100);


-- Update emails for existing employees

UPDATE employees

SET Email = CONCAT(LOWER(Name), '@company.com');
```

✓ Adds an email column and populates it with a pattern like alice@company.com.

Output After All Operations

```
+----+--------+---------+-----------+----------------------+

| ID | Name   | Salary  | Department| Email                |

+----+--------+---------+-----------+----------------------+

|  1 | Alice  | 55000.00| IT        | alice@company.com    |

|  2 | Bob    | 80000.00| Finance   | bob@company.com      |

|  4 | David  | 50000.00| IT        | david@company.com    |
+----+--------+---------+-----------+---------------------+
```

Week 12:

Create two tables, ‘products” and “orders”, and write an SQL query to perform an inner join to retrieve a list of products along with the order information.

```
  CREATE TABLE products (
   ProductID INT PRIMARY KEY AUTO_INCREMENT,
   ProductName VARCHAR(100) NOT NULL,
   Price DECIMAL(10,2) NOT NULL
);

CREATE TABLE orders (
   OrderID INT PRIMARY KEY AUTO_INCREMENT,
   ProductID INT,
   Quantity INT NOT NULL,
   FOREIGN KEY (ProductID) REFERENCES products(ProductID)
);

-- Insert products
INSERT INTO products (ProductName, Price) VALUES
('Laptop', 80000.00),
('Mouse', 1500.00),
('Keyboard', 2500.00);

-- Insert orders
INSERT INTO orders (ProductID, Quantity) VALUES
(1, 2),
(2, 5),
(3, 3);
```

```
SELECT p.ProductName, p.Price, o.Quantity, (p.Price * o.Quantity) AS TotalAmount

FROM products p

INNER JOIN orders o ON p.ProductID = o.ProductID;
```

✓ Output Example:

```
+------------+--------+----------+-------------+

| ProductName| Price  | Quantity | TotalAmount |

+------------+--------+----------+-------------+

| Laptop    | 80000  |   2    | 160000     |

| Mouse     | 1500   |   5    | 7500       |

| Keyboard   | 2500   |   3    | 7500       |

+------------+--------+----------+-------------+
```

2. Explain the differences between inner join, left join, and right join operations in SQL. Provide examples for each.

INNER JOIN

Returns matching rows from both tables.

```
SELECT p.ProductName, o.Quantity

FROM products p

INNER JOIN orders o ON p.ProductID = o.ProductID;
```

LEFT JOIN

Returns all rows from the left table, even if no match exists in the right table.

```
SELECT p.ProductName, o.Quantity

FROM products p

LEFT JOIN orders o ON p.ProductID = o.ProductID;
```

If a product has no order, Quantity will be NULL.

RIGHT JOIN

Returns all rows from the right table, even if no match exists in the left table.

```
SELECT p.ProductName, o.Quantity

FROM products p
```

```
RIGHT JOIN orders o ON p.ProductID = o.ProductID;
```