Department of Information Technology
BABU BANARASI DAS
Institute of Technology & Management, Lucknow
Recognized by AICTE, Govt.of India & Affiliated to Dr. A.P.J. Abdul Kalam Technical University,Lucknow,AKTU College Code=054

LAB MANUAL

OF

WEB TECHNOLOGY

BCS552

B.TECH, 3rd Year, Semester -V

Dr. A.P.J. Abdul Kalam Tech. University
Uttar Pradesh

2025-26

Department of Information Technology

Faculty Name: Mr. Arvind Goutam
(Assistant Professor)

Approved by: (Head of the Department)

MANUAL CONTENTS

This manual is intended for the 3rd year students of Information Technology in the subject of Web Technology BCS552. This manual typically contains practical sessions related to web designing covering various aspects related to the subject of enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

Preface

This practical manual will be helpful for students of Information Technology for understanding the course from the point of view of applied aspects. Though all the efforts have been made to make this manual error free, yet some errors might have crept in inadvertently. Suggestions from the readers for the improvement of the manual are most welcomed.

Vision of the Department

Vision of the department to transform the students to be skilled IT Innovative leaders, and environmentally receptive citizens.

Mission of the Department

M1: To develop and deliver quality academic programs in emerging & innovative fields of engineering to empower the students to meet industry standards.

M2: To provide students with the ethical and professional tools to become productive & become continuous Learners.

M3: To inculcate professional behavior, positive attitude, and communication skills.

Program Educational Objectives (PEOs)

1. **PE01:** To develop the ability among students to understand the concept of core Information Technology subjects that will facilitate understanding of new Technology.

2. **PE02:** To embed a strong foundation in the engineering fundamentals to solve, analyze and design real time engineering products.

2. **PE03:** To give exposures to emerging edge technologies, adequate training and opportunities to work as team on multidisciplinary projects with effective communication skills and leadership qualities

Program Specific Outcomes (PSOs)

PSO 1: Use and apply current technical concepts and practices in the core Information Technologies of human Machine Learning, Information management ,programming, Cloud computing.

PSO 2: Design, develop and test software systems for worldwide network of computers to provide solutions to real world problem.

Program Outcome (POs)

Engineering Graduates will be able to:

PO: 1 Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO: 2 Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO: 3 Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO: 4 Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO: 5 Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO: 6 The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO: 7 Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO: 8 Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO: 9 Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO: 10 Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO: 11 Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO: 12 Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Evaluation Scheme

S.No	Subject Code	Subject	Periods			Evaluation Scheme		End Semester	Total	Credit
			L	T	P	PS	Total	PE		
1	BCS552	Web Technology Lab	0	0	2	50	50	50	100	1

Course Objectives:

The teacher will explain:

1	Web Development Fundamentals: Gain proficiency in HTML and CSS to design and develop structured, responsive websites that effectively present information, such as departmental details for an institute.
2	Form Handling and Data Validation: Learn to create user-friendly entry forms for collecting data (e.g., student or employee information) and implement JavaScript for client-side validation to ensure data integrity.
3	Server-Side Programming and Database Interaction: Understand how to use Java Servlets and JSP to manage user sessions, authenticate users, and interact with databases for storing and retrieving user information.
4	JavaScript and Dynamic Content: Utilize JavaScript to enhance user experience by displaying browser information, validating input data, and creating interactive web applications.
5	E-Commerce and Session Management: Design and implement a simple shopping cart application with session tracking, understanding the principles of e-commerce and how to manage user sessions effectively.

Pre- requisite:

1. Basic Understanding of HTML and CSS: Familiarity with HTML and CSS for creating and styling web pages.
2. Familiarity with JavaScript: Basic knowledge of JavaScript for client-side validation and interactivity.
3. Introduction to Programming Concepts: Understanding fundamental programming concepts like variables, loops, and functions.
4. Basic Knowledge of Databases: Introductory knowledge of databases and SQL for data interaction.
5. Understanding of Web Technologies: Familiarity with web technologies and protocols (HTTP/HTTPS) for client-server interactions

Course Outcome of Practical

Course Outcome (CO) At the end of course, the student will be able to:		Bloom's Level (L)
CO 1	Understanding fundaments of website development and apply HTL and XML languages for development of websites	**L4**
CO 2	Applying CSS in designing and development of responsive website for compatibility of various devices.	**L5**
CO 3	Understand, analyze and design the role of JavaScript for dynamic web pages.	**L5**
CO 4	Design and deploy different components using Java Bean, Node.js and database tables using MongoDB and produce various results based on given query.	**L5**
CO 5	Design and deploy server-side java application called Servlet & JSP tools to catch form data sent from client, process it and store it on database.	**L4**

CO-PO-PSO Mapping

COURSE OUTCOME	COs	PROGRAM OUTCOMES(POs) & PROGRAM SPECIFIC OUTCOME(PSO)												
		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO 2
	CO1	3	2	2	-	-	-	-	-	-	-	-	3	2	1
	CO2	3	3	2	2	-	-	-	-	-	-	-	3	2	1
	CO3	3	3	3	2	-	-	-	-	-	-	-	3	3	-
	CO4	3	3	3	3	2	-	-	-	-	-	-	3	3	-
	CO5	3	3	3	3	2	-	-	-	-	-	-	3	3	-
CO (Average)		3	2.8	2.6	2.5	2		-	-	-	-	-	3	2.6	1.00

INDEX

S. No	Program Name	CO	BL
1	Write HTML program for designing your institute website. Display departmental information of your institute on the website.	CO1	L6
2	Write HTML program to design an entry form for student details/employee information/faculty details	CO1	L6
3	Write a program in XML for creation of DTD, which specifies set of rules. Create a style sheet in CSS/ XSL & display the document in internet explorer	CO2	L6
4	Develop a responsive website using CSS and HTML. Website may be for tutorial/blogs/commercial website etc.	CO2	L5
5	Write programs using Java script for Web Page to display browsers information.	CO3	L5
6	Write programs using HTML and Java Script for validation of input data validate to the fields (First Name. Last Name, Password, E-mail id, Mobile Number and Address) of the Registration page.	CO3	L5
7	Write a Java applet to display the Application Program screen i.e. calculator and other.	CO4	L5
8	Create a Java Bean for Employee information (EmpID, Name, Salary, Designation and Department).	CO4	L5
9	Build a command-line utility using Node.js that performs a specific task, such as converting text to uppercase, calculating the factorial of a number, or generating random passwords.	CO4	L5
10	Develop a script that uses MongoDB's aggregation framework to perform operations like grouping, filtering, and sorting. For instance, aggregate user data to find the average age of users in different cities	CO4	L5
11	Assume four users user1, user2, user3 and user4 having the passwords pwd1, pwd2, pwd3 and pwd4 respectively. Write a servlet for doing the following: 1. Create a Cookie and add these four user id's and passwords to this Cookie. 2. Read the user id and passwords entered in the Login form and authenticate with the values available in the cookies.	CO5	L4
12	Create a table which should contain at least the following fields: name, password, email-id, phone number Write Servlet/JSP to connect to that database and extract data from the tables and display them. Insert the details of the users who register with the web site, whenever a new user clicks the submit button in the registration page.	CO5	L4
	Experiments Beyond the Syllabus:		
S. No	**Experiment Name**	**CO**	**S. No**
13	Write a JSP which insert the details of the 3 or 4 users who register with the web site by using registration form. Authenticate the user when he submits the login form using the user name and password from the database.	CO5	L6
14	Design and implement a simple shopping cart example with session tracking API	CO5	L6

Softwares & Hardwares Required

1. Computer- Multi-core processor, Min 8 GB RAM, Min 256 GB SSD storage, monitor
2. Input Devices- keyboard, mouse and Internet Connection-Reliable high-speed internet
3. Development Environment- Operating System: Windows, macOS, or Linux
4. Development Tools:
   a) Text Editor/IDE: Visual Studio Code, Sublime Text, or Eclipse for writing HTML, CSS, JavaScript, and Java code.
   b) Web Browser: Google Chrome, Mozilla Firefox, or Microsoft Edge for testing and debugging web applications.
   c) Java Development Kit (JDK): JDK (version 8 or higher) for developing Java applications, including Servlets and Java Beans.
   d) Database Management System: MongoDB for managing databases and storing user information.
   e) Node.js: Node.js runtime for building command-line utilities and server-side applications.
   f) Apache Tomcat or GlassFish: A web server for deploying and running Java Servlets and JSP applications.
   g) Version Control System: Git for version control and collaboration on projects.

Practical No. 1

1. **Experiment Description:** To create a simple, static website for an institute that effectively displays departmental information. This exercise will help students understand the fundamental structure of HTML documents, the use of various HTML elements, and how to organize content for a web page.

Tools Required: A text editor (such as Visual Studio Code, Sublime Text, or Notepad++) for writing HTML code., A web browser (like Google Chrome, Mozilla Firefox, or Microsoft Edge) for testing and viewing the website.

Set Up the Project: Begin by creating a new folder on your computer to store all project files. This folder will serve as the workspace for your website development.

Create the HTML Document: Inside the project folder, create a new HTML file named index.html. This file will be the main page of your website.

Structure the HTML Document: Start by defining the basic structure of an HTML document, which includes the necessary elements such as the document type declaration, HTML tags, head section, and body section.

Add a Header Section: In the body of the HTML document, create a header that includes the name of the institute and a navigation menu. This menu should provide links to different sections of the website, such as "Departments" and "Contact Us."

Create a Department Section: Add a dedicated section to the webpage that outlines the various departments within the institute. This section should include headings and descriptive text for each department, providing visitors with an overview of the programs offered.

Save and Preview:

After completing the HTML structure, save the file and open it in a web browser to preview the website. Check that all sections are displayed correctly and that the navigation links function as intended.

2. **Objective:** Write HTML program for designing your institute website. Display departmental information of your institute on the website.
3. **Code and Procedure:**

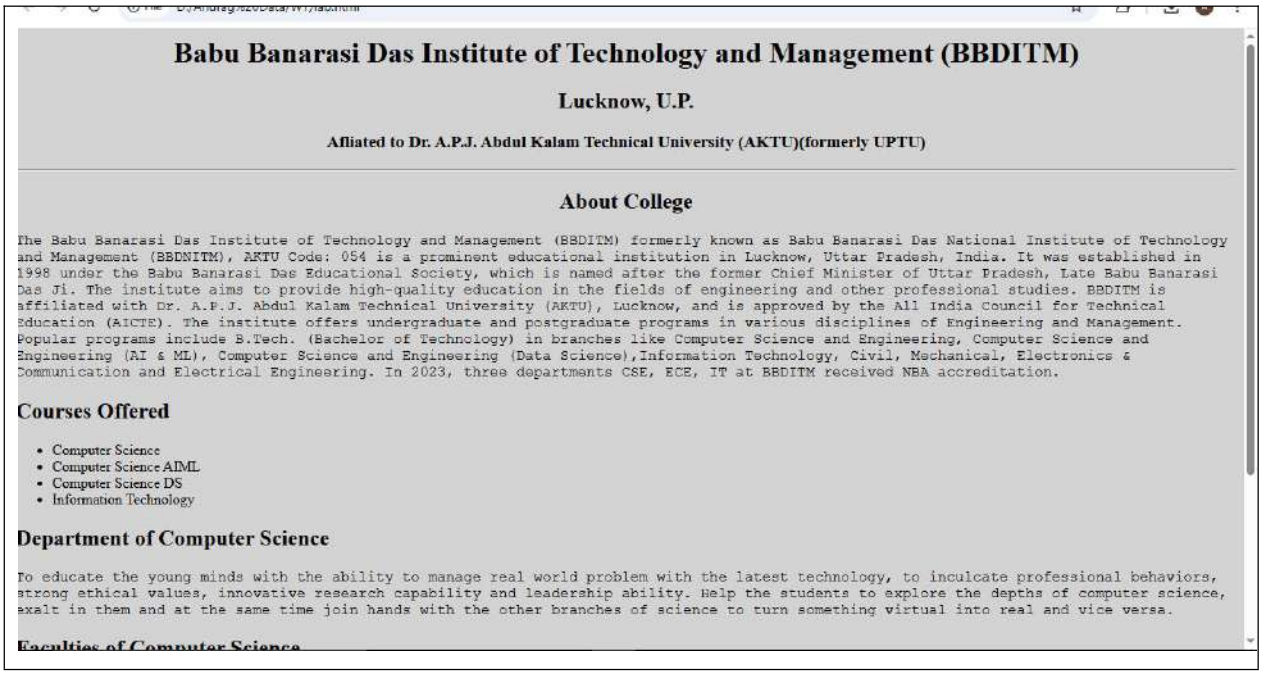
home.html:

```
<html><head><title>College Department Details</title></head>
<body bgcolor="lightgray">
<h1 align="center" style="color: black;"> Babu Banarasi Das Institute of Technology and Management (BBDITM)</h1>
<h2 align="center" style="color: black;">Lucknow, U.P.</h2>
<h3 align="center" style="color: black;">Afliated to Dr. A.P.J. Abdul Kalam Technical University (AKTU)(formerly UPTU)</h3>
<hr>
<h2 align="center" style="color: black;">About College </h2>
<p style="font-family:courier";>The Babu Banarasi Das Institute of Technology and Management (BBDITM) formerly known as Babu Banarasi Das National Institute of Technology and Management (BBDNITM), AKTU Code: 054 is a prominent educational institution in Lucknow, Uttar Pradesh, India. It was established in 1998 under the Babu Banarasi Das Educational Society, which is named after the former Chief Minister of Uttar Pradesh, Late Babu Banarasi Das Ji. The institute aims to provide high-quality education in the fields of engineering and other professional studies. BBDITM is affiliated with Dr. A.P.J. Abdul Kalam Technical University (AKTU), Lucknow, and is approved by the All India Council for Technical Education (AICTE). The institute offers undergraduate and postgraduate programs in various disciplines of Engineering and Management. Popular programs include B.Tech. (Bachelor of Technology) in branches
```

like Computer Science and Engineering, Computer Science and Engineering (AI & ML), Computer Science and Engineering (Data Science),Information Technology, Civil, Mechanical, Electronics & Communication and Electrical Engineering. In 2023, three departments CSE, ECE, IT at BBDITM received NBA accreditation.</p>

```
<h2><b>Courses Offered</b></h2>
<ul>
<li>Computer Science</li>
<li> Computer Science AIML</li>
<li> Computer Science DS</li>
<li>Information Technology</li>
</ul>
<h2>Department of Computer Science</h2>
<p style="font-family:courier">To educate the young
minds with the ability to manage real world problem with the latest technology,
to inculcate professional behaviors, strong ethical values, innovative research capability and leadership ability.
Help the students to explore the depths of computer science, exalt in them and at the same time join hands with the other
branches of science to turn something virtual into real and vice versa.</p>
<h2>Faculties of Computer Science</h2>
<ol>
<li>Mr. Anurag Tiwari (HOD) </li>
<li>Mr. Sandeep Kumar Mishra</li>
<li>Mr. Subha Mishra</li>
<li>Mr. Anurag Shukla </li>
<li>Ms. Jyoti Gaur </li>
</ol>
<h3>Condact as </h3>
<p>Phone Number: 00522 666333</p>
<p>Email:hodcs@bbdnitm.ac.in</p>
</body></html>
```

4. Output:



Output of Home Page

5. **Outcome of the Practical:** Students will have created a simple, static website that effectively displays departmental information for an institute. They will gain hands-on experience in structuring HTML documents, using various HTML elements, and organizing content for web presentation.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is the purpose of the <!DOCTYPE html> declaration in an HTML document?
   - Can you explain the structure of a basic HTML document?
   - What are semantic HTML elements, and why are they important?
   - How do you create a navigation menu in HTML?
   - What is the role of the <header> and <footer> tags in an HTML document?
   - How can you ensure that your website is responsive and looks good on different devices?
   - What are some best practices for writing clean and maintainable HTML code?
   - How can you add images to your HTML page, and what attributes are important for accessibility?
   - What is the significance of using the <section> tag in your webpage?
   - How would you test the functionality of your navigation links in the website?

Practical No. 2

1. **Experiment Description:** To create a user-friendly entry form using HTML that allows users to input and submit details for students, employees, or faculty members. This exercise will help students understand form elements, attributes, and the overall structure of HTML forms.

Tools Required:A text editor (such as Visual Studio Code, Sublime Text, or Notepad++) for writing HTML code.,A web browser (like Google Chrome, Mozilla Firefox, or Microsoft Edge) for testing and viewing the form.

Set Up the Project: Create a new folder on your computer to store the project files. This folder will serve as the workspace for your form development.

Create the HTML Document: Inside the project folder, create a new HTML file named entry_form.html. This file will contain the form for entering details.

Structure the HTML Document: Begin by defining the basic structure of an HTML document, which includes the necessary elements such as the document type declaration, HTML tags, head section, and body section.

Add a Form Element: In the body of the HTML document, create a <form> element that will contain all the input fields. This form will be used to collect user data.

- Include Input Fields: Add various input fields to the form, such as:
- Text fields for first name, last name, and email address.
- A password field for secure input.
- A telephone number field for contact information.
- A dropdown menu or radio buttons for selecting the role (student, employee, or faculty).
- A textarea for additional comments or address.

Labeling Input Fields: Ensure that each input field has an associated <label> element to improve accessibility and usability. Labels should clearly describe the purpose of each field.

Add a Submit Button: Include a submit button at the end of the form that users can click to send their entered information. This button will trigger the form submission process.

Save and Preview: After completing the HTML structure, save the file and open it in a web browser to preview the form. Check that all input fields are displayed correctly and that the form layout is user-friendly.

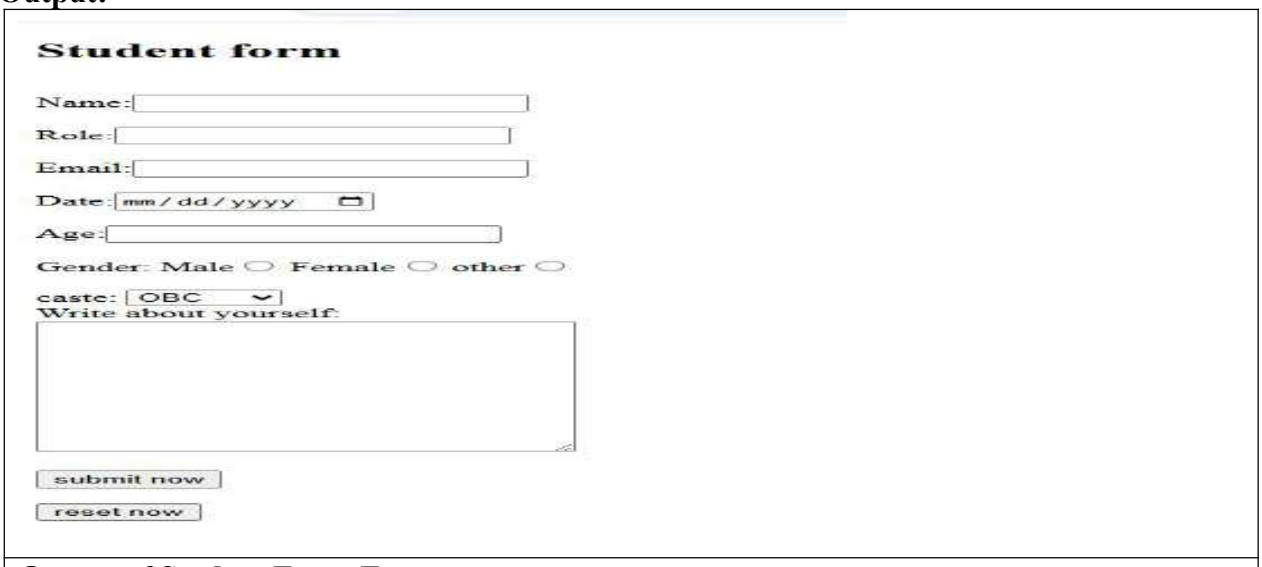
2. **Objective:** Write HTML program to design an entry form for student details/employee information/faculty details
3. **Code and Procedure:**

entryform.html

```
<html>  <head>   <title>Hello World!</title>
   <link rel="stylesheet" href="styles.css" />
 </head>
 <body>
    <h2> Student form </h2>
    <form action="#">
      <div>          Name:<input type ="text">  </div> <br>
      <div>           Role:<input type="text"> </div> <br>
      <div>           Email:<input type = "email" name="myemail"> </div> <br>
       <div>          Date:<input type ="date" name =" my date"> </div> <br>
          <div>           Age:<input type ="number"> </div> <br>
```

```
<div>          Gender: Male <input type ="radio" name ="my gender">
 Female<input type ="radio" name="my gender">
 other<input type ="radio" name ="my gender"> </div><br>
 <div>   <lebel for ="caste"> caste: </lebel>
 <select name="my caste" id="caste">
  <option value="obc"> OBC</option>
  <option value="general"> General</option>
  <option value="sc/st"> Sc/St</option>
  </div> <br>   </select>   <div>
 Write about yourself: <br> <textarea name="my text" cols="30" rows="10"></textarea>
 </div>  <br>
<div>  <input type ="submit" value="submit now">  </div>  <br>
  <div>    <input type="reset" value="reset now">  </div>  </form>  </body></html>
```

4. **Output:**



Output of Student Entry Form

5. **Outcome of the Practical:** Students learn to use essential HTML tags to create a functional login page, including the <form>, <input>, and <button> elements. They gain hands-on experience in structuring forms, applying input types for better user experience, and incorporating labels for accessibility. This exercise enhances their understanding of HTML form handling and prepares them for more advanced web development concepts.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What are the key components of a login page, and how did you structure them using HTML?
   - Can you explain the purpose of the <form> element in your login page?
   - What types of <input> elements did you use for the username and password fields, and why?
   - How did you ensure that the any input is masked when the user types it?
   - What attributes did you include in the <input> elements to enhance user experience (e.g., placeholders, required fields)?
   - How did you implement the submit button, and what type did you choose for it?

- What validation techniques are available in HTML to ensure that the user inputs are correct before submission?
- How did you test the functionality of the page to ensure it works as intended?
- How can you handle form submission in HTML, and what are the common methods used?
- What are some best practices for designing user-friendly forms in HTML?

Practical No. 3

1. **Experiment Description:** Designing a cart page that displays details of books added to a shopping cart, utilizing XML, DTD (Document Type Definition), HTML, and XSL (Extensible Stylesheet Language).
   XML (eXtensible Markup Language) is used to store the cart data in a structured format, allowing for the representation of complex data hierarchies. In this context, XML can include elements such as <book>, <title>, <author>, <price>, and <quantity>, encapsulating all relevant information about each book in the cart. The use of XML provides flexibility, as it can easily accommodate changes in data structure, such as adding new attributes or elements without disrupting the existing data. Additionally, XML is platform-independent, making it suitable for data interchange between different systems.
   DTD (Document Type Definition) is created to define the structure and rules for the XML file. It specifies the allowed elements, their attributes, and the relationships between them, ensuring that the XML document adheres to a consistent format. For example, a DTD for the cart might define that each <book> element must contain a <title>, <author>, <price>, and <quantity>. This validation helps maintain data integrity, ensuring that any XML file conforming to the DTD is well-formed and can be reliably parsed by applications. By enforcing these rules, DTDs help prevent errors and inconsistencies in the data.
   HTML (HyperText Markup Language) is employed to create the layout of the cart page. It structures the content, allowing for the display of book details in a user-friendly format. HTML elements such as <table>, <tr>, and <td> are used to organize the information clearly, presenting each book's details in rows and columns.
   XSL (Extensible Stylesheet Language) is utilized to transform the XML data into a presentable format for the web page. XSLT (XSL Transformations) allows developers to define how the XML content should be displayed, enabling the transformation of XML into HTML. This process involves applying styles and formatting to the XML data, ensuring that the cart page is visually appealing and easy to read. For instance, XSL can be used to format prices, apply styles to book titles, or create conditional formatting based on the quantity of books in the cart. By separating the data (XML) from its presentation (XSL), developers can maintain a clean and organized codebase, making future updates easier.
2. **Objective:** Write a program in XML for creation of DTD, which specifies set of rules. Create a style sheet in CSS/ XSL & display the document in internet explorer.
3. **Code and Procedure:**

cart.html

```
<html> <body >
<table   cellspacing="50" cellpadding="20">
<tr><td>Book name</td><td>price</td><td>quantity</td><td>amount</td></tr>
<tr><td>java2</td><td>$35.5</td><td>2</td><td>$70</td></tr>
<tr><td>XML bible</td><td>$40.5</td><td>1</td><td>$40.5</td></tr>
<tr><td></td><td></td> <td>Total amount </td> <td>130.5 </td></tr>
</body> </html>
```

Program for creating DTD for the XML file(dtd.xml

```
<?xml version="1.0" ?>
```

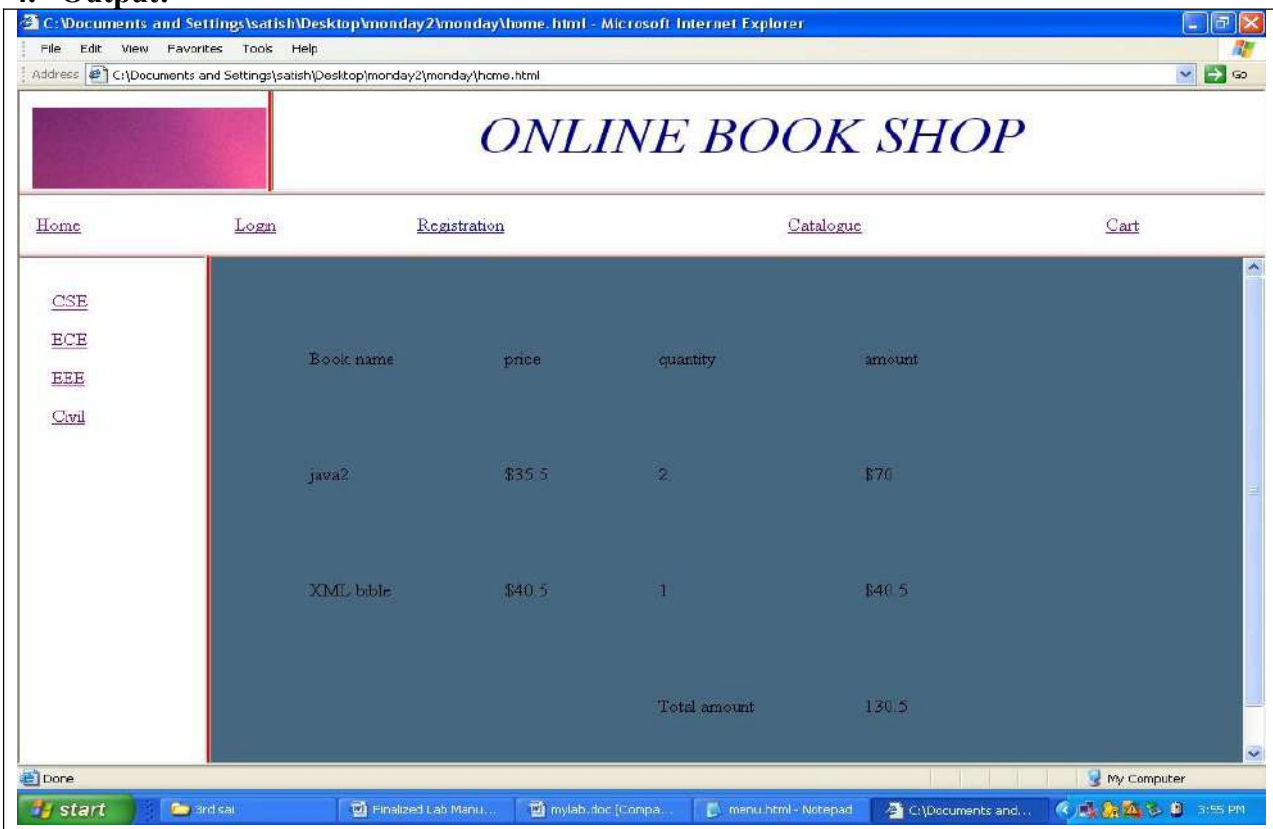
```
<?xml:stylesheet type="text/xsl" href="Style.xsl"?>
<!DOCTYPE BOOKS[
<!ELEMENT BOOKS (INFORMATION)>
<!ELEMENT INFORMATION (Book)>
<!ELEMENT Book (book_name,Author_name,ISBN_number,publisher,Edition,Price)>
<!ELEMENT book_name (#PCDATA)>
<!ELEMENT Author_name (#PCDATA)>
<!ELEMENT ISBN_number (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT Edition (#PCDATA)>
<!ELEMENT price (#PCDATA)>]>
<BOOKS>
<INFORMATION>
<Book>
<book_name>XML BIBLE</book_name>
<Author_name>RAJA</Author_name>
<ISBN_number>11530-115</ISBN_number>
<publisher>Pearson</publisher>
<Edition>I-Edition</Edition>
<Price>750</Price>
</Book>
<Book>
<book_name>HTML</book_name>
<Author_name>DIETL</Author_name>
<ISBN_number>11528-115</ISBN_number>
<publisher>wiley</publisher>
<Edition>III-Edition</Edition>
<Price>470</Price>
</Book>
<Book>
<book_name>JAVASCRIPT</book_name>
<Author_name>RANI</Author_name>
<ISBN_number>11525-115</ISBN_number>
<publisher>Techical</publisher><Edition>IV-Edition</Edition>
<Price>270</Price>
</Book>
<Book>
<book_name>WEB TECHNOLOGIES</book_name>
<Author_name>PUNTAMBEKAR</Author_name>
<ISBN_number>11530-115</ISBN_number>
<publisher>Technical</publisher>
<Edition>I-Edition</Edition>
<Price>340</Price>
</Book>
</INFORMATION>
</BOOKS>
```

style.xsl

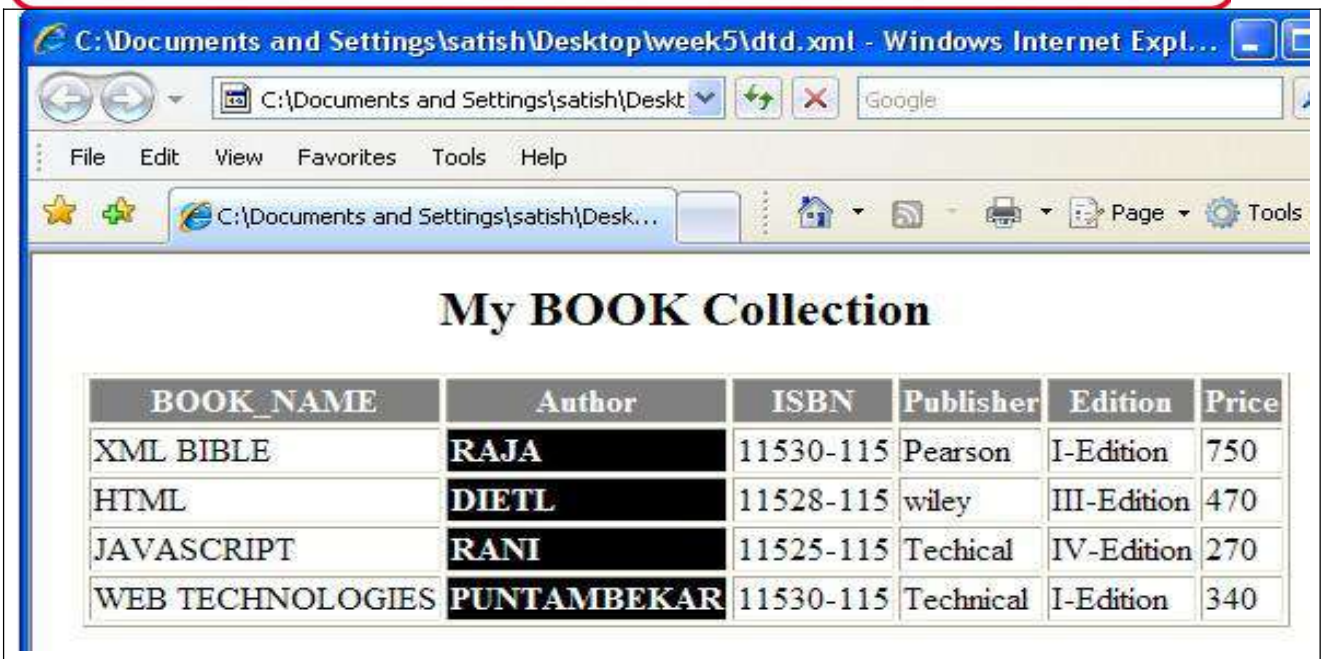
```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
```

```
<h2 align="center">My BOOK Collection</h2>
<table border="1" align="center">
<tr bgcolor="GRAY">
<th><font color="white"> BOOK_NAME</font></th>
<th><font color="white"> Author</font></th>
<th><font color="white"> ISBN</font></th>
<th><font color="white"> Publisher</font></th>
<th><font color="white"> Edition</font></th>
<th><font color="white"> Price</font></th>
</tr>
<xsl:for-each select="BOOKS/INFORMATION/Book">
<tr>
<td><xsl:value-of select="book_name"/></td>
<td bgcolor="black"><b><font color="white">
<xsl:value-of select="Author_name"/></font></b></td>
<td><xsl:value-of select="ISBN_number"/></td>
<td><xsl:value-of select="publisher"/></td>
<td><xsl:value-of select="Edition"/></td>
<td><xsl:value-of select="Price"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

4. Output:



1. **Output of Cart Page**



2. Output of dtd.xml file

5. **Outcome of the Practical:** The outcome of this practical is a comprehensive understanding of how to design and implement a cart page that effectively displays book details using XML, DTD, HTML, and XSL. Students gain hands-on experience in structuring data with XML and defining its rules through DTD, ensuring data integrity and consistency. They learn to create a user-friendly layout with HTML, organizing information in a clear and accessible manner.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. Viva Questions/Review Questions:

- What is the purpose of the cart page, and how does it enhance the user experience in an online bookstore?
- How is XML utilized to store the cart data, and what elements are included in the XML structure?
- Can you explain the role of DTD (Document Type Definition) in your XML file and how it ensures data integrity?
- What specific rules and relationships did you define in the DTD for the cart's XML structure?
- How did you use HTML to create the layout of the cart page, and what elements did you employ to display the book details?
- What is XSL, and how does it facilitate the transformation of XML data into a presentable format for the cart page?
- Can you describe how you applied XSLT (XSL Transformations) to format the XML data for display on the cart page?
- How does the separation of data (XML) and presentation (XSL) benefit the development and maintenance of the cart page?
- What styling techniques did you implement to ensure that the cart page is visually appealing and user-friendly?
- How did you test the functionality of the cart page to ensure that all book details are displayed correctly and that the data is accurately represented?

Practical No. 4

1. **Experiment Description:** To develop a responsive website using CSS and HTML Focusing on registration page. The registration page is designed to collect user information for account creation.
   HTML is used to create the structure and content of a web page. It consists of elements (tags) that define the content, such as headings, paragraphs, forms, images, and links. HTML defines the structure of input fields, labels, and buttons to collect user information.
   CSS is used to style the HTML elements, making the web page visually appealing. It controls aspects like layout, colors, fonts, spacing, and responsiveness. CSS allows designers to separate content (HTML) from presentation, improving maintainability and flexibility.
2. **Objective:** Develop a responsive website using CSS and HTML. Website may be for tutorial/blogs/commercial website etc.
3. **Code and Procedure:**

registration.html

```
<html >
<head> <title>Registration Page</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
 <div class="container">
  <h2>Registration Form</h2>
      <form action="#" method="POST">
        <label for="name">Name</label>
        <input type="text" id="name" name="name" required>
        <label for="password">Password</label>
        <input type="password" id="password" name="password" required>
        <label for="email">E-mail</label>
        <input type="email" id="email" name="email" required>
        <label for="phone">Phone Number</label>
        <input type="text" id="phone" name="phone" required>
        <label>Sex</label>
       <div class="radio-group">
          <input type="radio" id="male" name="sex" value="male">
          <label for="male">Male</label>
          <input type="radio" id="female" name="sex" value="female">
          <label for="female">Female</label>
          <input type="radio" id="other" name="sex" value="other">
          <label for="other">Other</label>
        </div>
    <label for="dob">Date of Birth</label>
  <div>
          <select id="day" name="day" required>
            <option value="" disabled selected>Day</option>
            <option value="1">1</option>
            <option value="2">2</option>
            <option value="3">3</option>
            <!-- Continue this up to 31 -->
          </select>
          <select id="month" name="month" required>
```

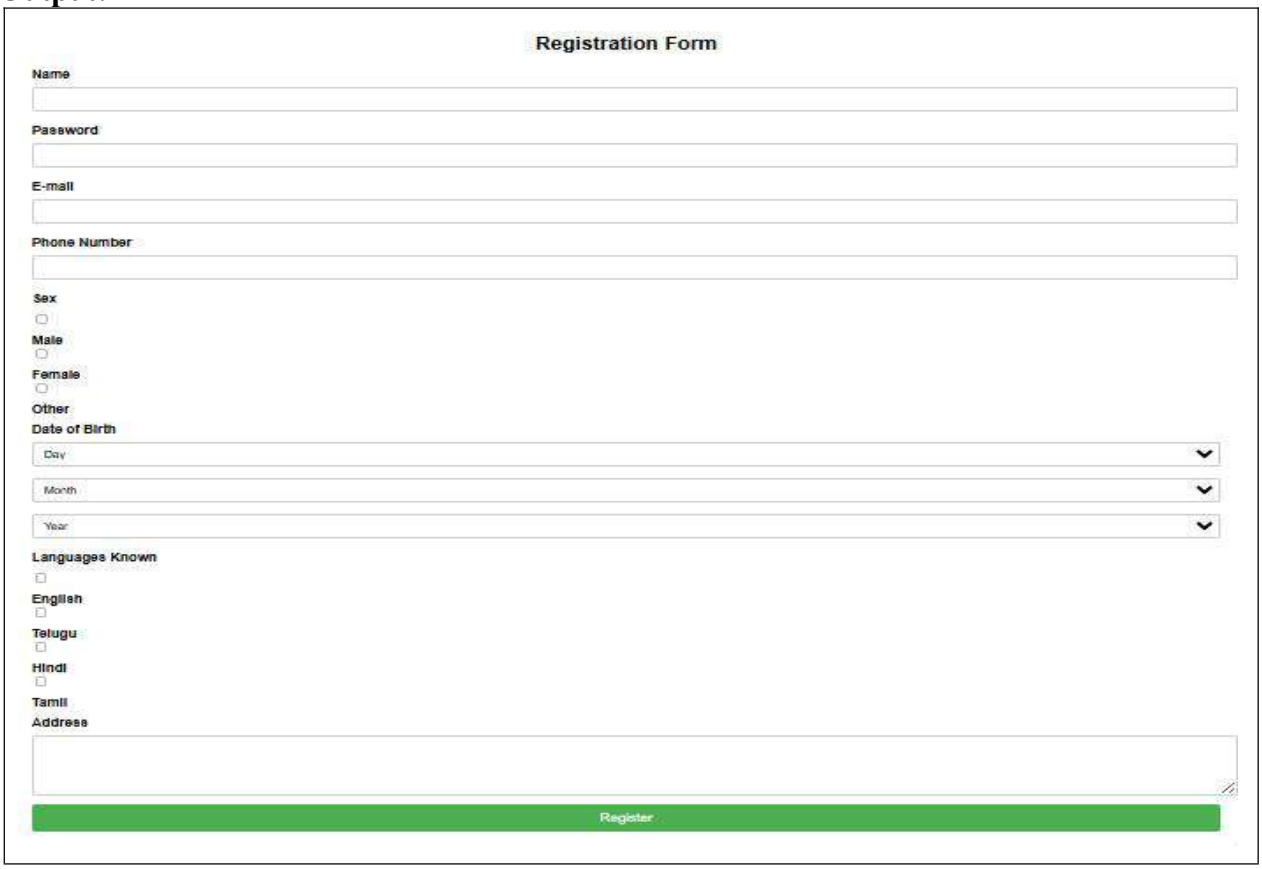
```
            <option value="" disabled selected>Month</option>
            <option value="1">January</option>
            <option value="2">February</option>
            <option value="3">March</option>
            <!-- Continue for all months -->
          </select>
          <select id="year" name="year" required>
            <option value="" disabled selected>Year</option>
            <option value="2000">2000</option>
            <option value="1999">1999</option>
            <!-- Continue for other years -->
          </select>
        </div>
        <label>Languages Known</label>
    <div class="checkbox-group">
          <input type="checkbox" id="english" name="languages" value="English">
          <label for="english">English</label>
          <input type="checkbox" id="telugu" name="languages" value="Telugu">
          <label for="telugu">Telugu</label>
          <input type="checkbox" id="hindi" name="languages" value="Hindi">
          <label for="hindi">Hindi</label>
          <input type="checkbox" id="tamil" name="languages" value="Tamil">
          <label for="tamil">Tamil</label>          </div>
        <label for="address">Address</label>
        <textarea id="address" name="address" rows="4" required></textarea>
        <button type="submit" class="submit-btn">Register</button>
      </form>
</div>
</body>
</html>
```

style.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f9;
}
.container {
    width: 50%;
    margin: 50px auto;
    padding: 20px;
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
h2 {
    text-align: center;
}
label {
    margin-top: 10px;
```

```
    display: block;
    font-weight: bold;
}
input[type="text"], input[type="password"], input[type="email"], select, textarea {
    width: 100%;
    padding: 8px;
    margin: 8px 0;
    border-radius: 4px;
    border: 1px solid #ccc;
}
.radio-group, .checkbox-group {
    margin: 10px 0;
}
input[type="radio"], input[type="checkbox"] {
    margin-right: 10px;
}
.submit-btn {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    border: none;
    cursor: pointer;
    border-radius: 4px;
    width: 100%;
    font-size: 16px;
}
```

4. Output:



Output of Registration Page for responsive website

5. **Outcome of the Practical:** The outcome of implementing the described registration page will significantly enhance user experience and facilitate comprehensive data collection. The intuitive design of the form, with clearly labeled fields and a logical flow, ensures that users can easily understand what information is required, making the registration process straightforward and user-friendly. Additionally, the form's responsive layout, achieved through effective CSS styling, guarantees that it is visually appealing and functions well across various devices, including desktops, tablets, and smartphones.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What specific HTML elements did you use to create the input fields for the registration form?
   - How did you implement the password masking feature in the password field?
   - What validation methods did you use to ensure that the email address entered is in the correct format?
   - How did you structure the radio buttons for gender selection, and what attributes did you include?
   - What CSS properties did you apply to enhance the layout and design of the registration form?
   - How did you ensure that the address field is large enough for users to enter their complete address?
   - What approach did you take to make the registration form responsive for different screen sizes?
   - How did you organize the select boxes for the date of birth, and what options did you provide for each?
   - What techniques did you use to style the checkboxes for language selection to improve user experience?
   - How did you test the functionality of the registration form to ensure all fields work as intended?

Practical No. 5

1. **Experiment Description:** To create a web page that utilizes JavaScript to retrieve and display information about the user's web browser. This exercise will help students understand how to interact with the browser's environment using the Document Object Model (DOM) and the navigator object in JavaScript.

Tools Required: A text editor (such as Visual Studio Code, Sublime Text, or Notepad++) for writing HTML and JavaScript code, A web browser (like Google Chrome, Mozilla Firefox, or Microsoft Edge) for testing and viewing the web page.

Set Up the Project: Create a new folder on your computer to store the project files. This folder will serve as the workspace for your browser information display project.

Create the HTML Document: Inside the project folder, create a new HTML file named browser_info.html. This file will contain the structure of the web page.

Structure the HTML Document: Begin by defining the basic structure of an HTML document, which includes the necessary elements such as the document type declaration, HTML tags, head section, and body section.

Add a Title and Heading: In the <head> section, set the title of the web page to something descriptive, such as "Browser Information". In the <body>, add a heading that indicates the purpose of the page.

Create a Section for Displaying Information: Add a or <section> element in the body where the browser information will be displayed. This section will be populated dynamically using JavaScript.

Write JavaScript Code: Inside a <script> tag in the HTML document, write JavaScript code to retrieve browser information using the navigator object. This object provides various properties and methods to access details about the browser, such as:

- navigator.userAgent: Returns the user agent string for the browser.
- navigator.appName: Returns the name of the browser.
- navigator.appVersion: Returns the version of the browser.
- navigator.platform: Returns the platform on which the browser is running.

Display the Information: Use JavaScript to manipulate the DOM and insert the retrieved browser information into the designated section of the web page. This can be done using methods like document.getElementById() or document.querySelector() to target the HTML elements.

Save and Preview: After completing the HTML structure and JavaScript code, save the file and open it in a web browser to preview the page. Check that the browser information is displayed correctly and updates dynamically based on the user's browser.

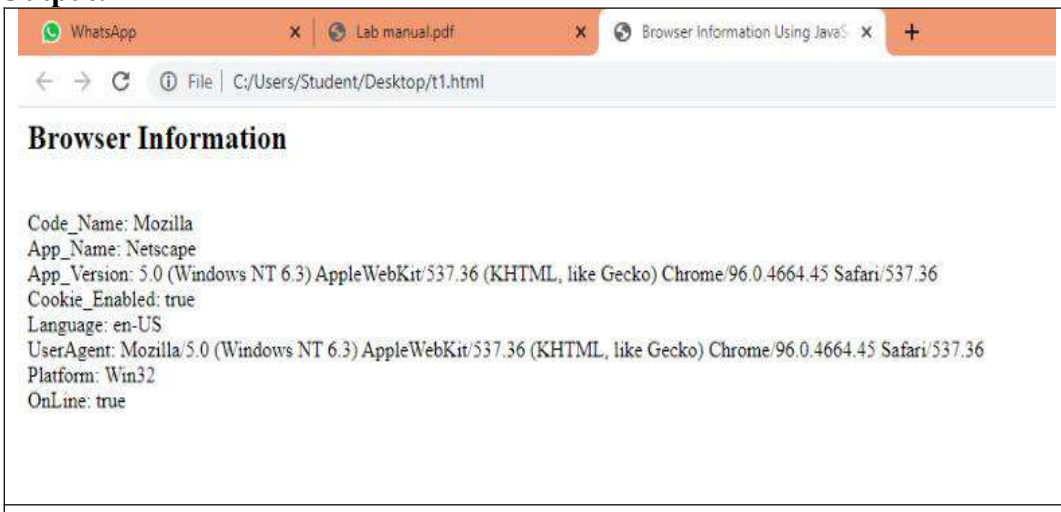
2. **Objective:** Write programs using Java script for Web Page to display browsers information.
3. **Code and Procedure:**

browser.html

```
<html>
<head>
<title>Browser Information Using JavaScript</title>
</head>
<body>
<h2>Browser Information</h2>
<script>
document.writeln("<br/>Code_Name: "+navigator.appCodeName);
document.writeln("<br/>App_Name: "+navigator.appName);
document.writeln("<br/>App_Version: "+navigator.appVersion);
document.writeln("<br/>Cookie_Enabled: "+navigator.cookieEnabled);
document.writeln("<br/>Language: "+navigator.language);
document.writeln("<br/>UserAgent: "+navigator.userAgent);
```

```
document.writeln("<br/>Platform: "+navigator.platform);
document.writeln("<br/>OnLine: "+navigator.onLine);
</script>
</body>
</html>
```

4. **Output:**



Output showing Browser Information

5. **Outcome of the Practical:** Students learned to create a web page that displays various pieces of information about the user's web browser. They will gain practical experience in using JavaScript to interact with the browser environment, manipulate the DOM, and dynamically update content on a web page.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is the purpose of the navigator object in JavaScript?
   - How can you retrieve the user agent string of the browser using JavaScript?
   - What information can you obtain from the navigator object? Can you name at least three properties?
   - How do you manipulate the DOM in JavaScript to display browser information on a web page?
   - What is the difference between document.getElementById() and document.querySelector()?
   - How can you ensure that your JavaScript code runs after the HTML document has fully loaded?
   - What are some potential security concerns when displaying browser information on a web page?
   - How would you handle cases where the browser does not support certain properties of the navigator object?
   - Can you explain how to create a button that refreshes the displayed browser information when clicked?
   - What are some practical applications of retrieving and displaying browser information in web development?

Practical No. 6

1. **Experiment Description:** The validation process involves creating a JavaScript function that will be triggered when the user submits the registration form. This function will check the following rules:
   Name Validation: The name must contain only alphabetic characters (both uppercase and lowercase).,The length of the name must be at least 6 characters.
   Password Validation: The password must be at least 6 characters long.
   Email ID Validation:The email must follow the standard format of name@domain.com.
   It should not contain any invalid characters, such as spaces or special characters that are not allowed in email addresses.
   Phone Number Validation:The phone number must consist of exactly 10 digits.
   It should not contain any letters, special characters, or spaces.
   HTML Structure: Ensure that the registration form includes input fields for the email ID and phone number, along with a submit button. The form should also have an area to display validation messages.
   JavaScript Function: Create a function that retrieves the values of the email and phone number fields when the form is submitted. This function will perform the following checks:
   For the email, use a regular expression to validate the format.
   For the phone number, check that it consists of exactly 10 digits.
   Display Validation Messages: If the validation fails, display appropriate error messages to the user, indicating what needs to be corrected. If both validations pass, allow the form to be submitted.
2. **Objective:** Write programs using HTML and Java Script for validation of input data validate to the fields (First Name. Last Name, Password, E-mail id, Mobile Number and Address) of the Registration page.
3. **Code and Procedure:**

validation.html

```
<html ><head>
   <title>Registration Form</title>
   <link rel="stylesheet" href="style.css">
</head>
<body>    <div class="container">
      <h2>Registration Form</h2>
      <form id="registrationForm">
         <div class="form-group">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>
            <span class="error" id="nameError"></span>          </div>
         <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
            <span class="error" id="emailError"></span>          </div>
         <div class="form-group">
            <label for="phone">Phone Number:</label>
            <input type="text" id="phone" name="phone" required>
            <span class="error" id="phoneError"></span>          </div>
         <div class="form-group">
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>
```

```
            <span class="error" id="passwordError"></span>          </div>
        <button type="submit">Register</button>       </form>    </div>
   <script src="script.js"></script></body></html>
```

style.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 20px;
}
.container {
    max-width: 400px;
    margin: auto;
    background: white;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
h2 {
    text-align: center;
}
.form-group {
    margin-bottom: 15px;
}
label {
    display: block;
    margin-bottom: 5px;
}
.error {
    color: red;
    font-size: 0.9em;
}
button {
    width: 100%;
    padding: 10px;
    background-color: #28a745;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
```

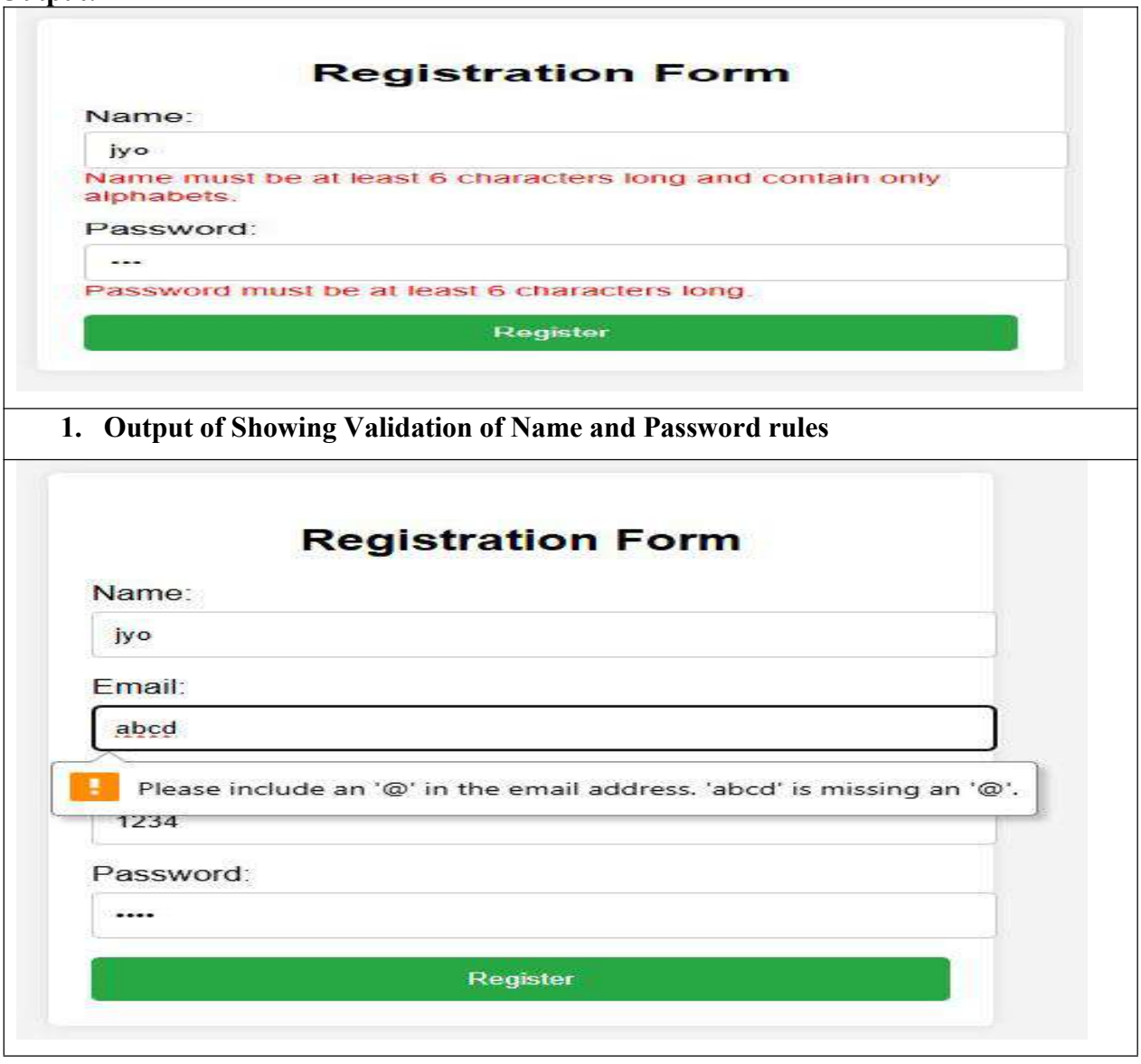
script.js

```
document.getElementById('registrationForm').addEventListener('submit', function(event) {
    event.preventDefault();
    document.getElementById('nameError').textContent = '';
    document.getElementById('emailError').textContent = '';
    document.getElementById('phoneError').textContent = '';
    document.getElementById('passwordError').textContent = '';
    const name = document.getElementById('name').value;
    const email = document.getElementById('email').value;
```

```
    const phone = document.getElementById('phone').value;
    const password = document.getElementById('password').value;

    let valid = true;
    // Validate Email
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; // Basic email validation
    if (!emailRegex.test(email)) {
        document.getElementById('emailError').textContent = 'Please enter a valid email address.';
        valid = false;
    }
    // Validate Phone Number
const phoneRegex = /^\d{10}$/; // Exactly 10 digits
     if (!phoneRegex.test(phone)) {
        document.getElementById('phoneError').textContent = 'Phone number must contain exactly 10 digits.';
        valid = false;
    }
```

4. Output:



1. Output of Showing Validation of Name and Password rules

2. Output of Showing Validation Email rules

5. **Outcome of the Practical:** The implementation of JavaScript validation for the name and password email and phone number fields significantly enhances user experience by providing immediate feedback on input errors, allowing users to correct mistakes before form submission. By enforcing rules that require that require the name to be at least 6 characters long and contain only alphabetic characters, as well as ensuring the password meets a minimum length, the email format, as well as ensuring the phone number meets a minimum length and no inclusion of other characters, the validation process helps maintain data integrity and security. Clear error messages guide users in correcting their entries, reducing frustration and confusion.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What specific regular expression did you use to validate the format of the email address?
   - How do you ensure that the email validation checks for the presence of both the "@" symbol and a valid domain?
   - What criteria did you establish to determine if the email contains invalid characters?
   - How do you retrieve the value of the phone number field when the form is submitted?
   - What method do you use to check that the phone number consists of exactly 10 digits?
   - How do you handle cases where the phone number input contains letters or special characters?
   - What feedback mechanism do you implement to inform users of specific errors in their email or phone number inputs?
   - How did you structure the HTML to include input fields for both the email ID and phone number, as well as an area for validation messages?
   - What steps do you take to prevent the form from submitting if either the email or phone number validation fails?
   - How do you test the validation function to ensure it works correctly with various valid and invalid email and phone number inputs?

Practical No. 7

1. **Experiment Description:** To create a simple calculator application using Java Applet. The applet will provide a graphical user interface (GUI) that allows users to perform basic arithmetic operations such as addition, subtraction, multiplication, and division.

Materials Required: A computer with Java Development Kit (JDK) installed.
A text editor or Integrated Development Environment (IDE) such as Eclipse or IntelliJ IDEA.
A web browser that supports Java Applets (Note: Most modern browsers have deprecated support for Java Applets).
The Java applet is implemented in the ApplicationProgram.java file. The applet consists of a display area for showing the input and results, and buttons for digits and operations.
Code Explanation:
Imports: The necessary classes from the java.applet, java.awt, and java.awt.event packages are imported to create the applet and handle events.
Class Declaration: The ApplicationProgram class extends Applet and implements ActionListener to handle button click events.
Variables:

- TextField display: A text field to show the current input and results.
- Button[] buttons: An array to hold the calculator buttons.
- String[] labels: An array of button labels representing digits and operations.
- double num1, num2, result: Variables to store the operands and result of calculations.
- char operator: A variable to store the current operator.

init() Method: This method initializes the applet:

- Sets the layout to BorderLayout.
- Creates a non-editable text field for display.
- Creates a panel with a GridLayout to arrange buttons in a 4x4 grid.
- Adds buttons to the panel and registers the action listener.

actionPerformed() Method: This method handles button click events:

- If a number button is clicked, it appends the number to the display.
- If the "C" button is clicked, it clears the display and resets the operands and result.
- If the "=" button is clicked, it performs the calculation based on the selected operator and displays the result.
- If an operator button is clicked, it stores the operator and the first operand for the calculation.

Procedure:
Open your text editor or IDE: Create a new Java file named ApplicationProgram.java.
Code Implementation: Code into the ApplicationProgram.java file and Save the file.

- Compile the Applet: Open a command prompt or terminal, Navigate to the directory where the ApplicationProgram.java file is located.
- Compile the applet using the command: javac ApplicationProgram.java
- Run the Applet: To run the applet, you can use the appletviewer tool provided by the JDK. Create an HTML file (e.g., AppletTest.html) with the following content:
- <html><applet code="ApplicationProgram.class" width="300" height="400"> </applet> </html>
- Save the HTML file in the same directory as the compiled class file.
- Run the applet using the command: appletviewer AppletTest.html

2. **Objective:** Write a Java applet to display the Application Program screen i.e. calculator and other.

3. Code and Procedure:

ApplicationProgram.java

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

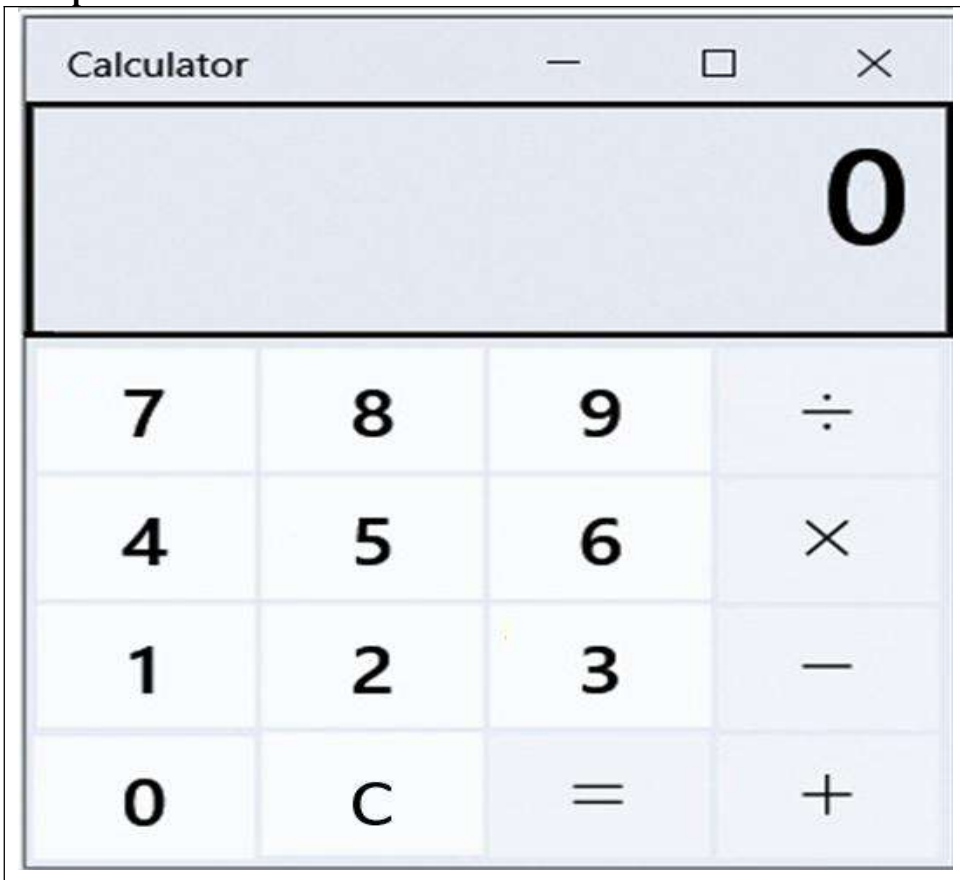
public class ApplicationProgram extends Applet implements ActionListener {
    TextField display;
    Button[] buttons;
    String[] labels = {"7", "8", "9", "*", "4", "5", "6", "*", "1", "2", "3", "-", "0", "C", "=", "+"};
    double num1 = 0, num2 = 0, result = 0;
    char operator;

    @Override
    public void init() {
        setLayout(new BorderLayout());
        display = new TextField();
        display.setEditable(false);
        add(display, BorderLayout.NORTH);
        Panel buttonPanel = new Panel();
        buttonPanel.setLayout(new GridLayout(4, 4));
        buttons = new Button[16];
        for (int i = 0; i < 16; i++) {
            buttons[i] = new Button(labels[i]);
            buttons[i].addActionListener(this);
            buttonPanel.add(buttons[i]);
        }
        add(buttonPanel, BorderLayout.CENTER);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (command.charAt(0) >= '0' && command.charAt(0) <= '9') {
            display.setText(display.getText() + command);
        } else if (command.equals("C")) {
            display.setText("");
            num1 = num2 = result = 0;
        } else if (command.equals("=")) {
            num2 = Double.parseDouble(display.getText());
            switch (operator) {
                case '+':
                    result = num1 + num2;
                    break;
                case '-':
                    result = num1 - num2;
                    break;
                case '*':
                    result = num1 * num2;
                    break;
```

```
            case '%':
                result = num1 % num2;
                break;
        }
        display.setText("" + result);
    } else {
        operator = command.charAt(0);
        num1 = Double.parseDouble(display.getText());
        display.setText("");
    }
  }
}
```

4 Output:



Output of Calculator

5 Outcome of the Practical: On completing this practical, students will gain hands-on experience in creating a simple graphical user interface (GUI) using Java Applet, which enhances their understanding of GUI programming concepts. They will learn how to handle events and respond to user interactions effectively, which is crucial for developing interactive applications. Additionally, students will implement basic arithmetic operations in a calculator application, reinforcing their knowledge of Java's built-in classes and methods.

6 Assessment Criteria:

Record	3 Marks
Execution	4 Marks

Viva	3 Marks

7 **Viva Questions/Review Questions:**

- What is a Java Applet, and how does it differ from a standard Java application?
- Explain the role of the init() method in a Java Applet. What is its purpose?
- How do you handle user interactions in a Java Applet? Can you describe the event handling mechanism?
- What are the different layout managers available in Java, and why did you choose BorderLayout and GridLayout for this calculator?
- How does the actionPerformed() method work in your applet? What is its significance in the context of the calculator?
- Can you explain how you implemented the arithmetic operations (addition, subtraction, multiplication, division) in your calculator?
- What is the purpose of the TextField in your applet, and how did you ensure it is non-editable?
- Why is it important to reset the operands and result when the "C" button is clicked? How did you implement this functionality?
- What challenges did you face while developing this applet, and how did you overcome them?
- Given that applets are largely deprecated in modern web development, what alternative technologies would you consider for creating similar applications today?

Practical No. 8

1. **Experiment Description**: To create a Java Bean that encapsulates employee information, including attributes such as Employee ID, Name, Salary, Designation, and Department. This exercise will help students understand the concept of Java Beans, their properties, and how to use them in Java applications.

Tools Required: Java Development Kit (JDK) for compiling and running Java programs.

A text editor or Integrated Development Environment (IDE) (such as Eclipse, IntelliJ IDEA, or NetBeans) for writing Java code.

Understanding Java Beans:A Java Bean is a reusable software component that follows specific conventions, including having a public no-argument constructor, providing getter and setter methods for accessing properties, and being serializable. Java Beans are commonly used in Java applications to encapsulate data.

Creating the Employee Java Bean:Create a new Java class named Employee that implements the Serializable interface. This class will contain private fields for the employee's attributes:

- empID (int)
- name (String)
- salary (double)
- designation (String)
- department (String)

Implement public getter and setter methods for each attribute to allow access and modification of the employee's information.

Implementing the Display Method: Add a method named displayInfo() to the Employee class that returns a formatted string containing the employee's details. This method will be useful for displaying the employee's information in a readable format.

Testing the Java Bean: Create a separate class named EmployeeTest with a main method to test the functionality of the Employee Java Bean. In this class:

Instantiate an Employee object and set its attributes using the setter methods.

Call the displayInfo() method to print the employee's information to the console.

Update the employee's salary and designation using the setter methods and display the updated information.

Compiling and Running the Application: Compile the Employee and EmployeeTest classes using the Java compiler.

Run the EmployeeTest class to verify that the Java Bean works as expected and that the employee information is displayed correctly.

2. **Objective:** Create a Java Bean for Employee information (EmpID, Name, Salary, Designation and Department).
3. **Code and Procedure:**

Employee.java (Creating java bean)

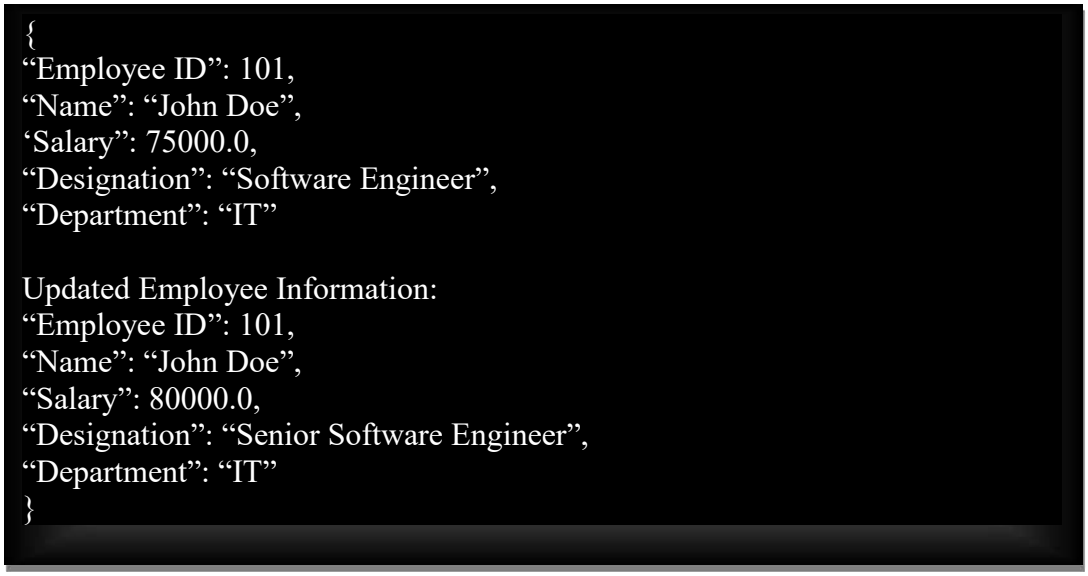
```
import java.io.Serializable;
public class Employee implements Serializable {
    private int empID;
    private String name;
    private double salary;
    private String designation;
    private String department;

    public Employee() {
            }
    public int getEmpID() {
        return empID;
    }
    public void setEmpID(int empID) {
        this.empID = empID;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public String getDesignation() {
        return designation;
    }
    public void setDesignation(String designation) {
        this.designation = designation;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
    public String displayInfo() {
        return "Employee ID: " + empID + "\n" +
               "Name: " + name + "\n" +
               "Salary: " + salary + "\n" +
               "Designation: " + designation + "\n" +
               "Department: " + department;
    }
}
```

EmployeeTest .java (To access java bean in java)

```
public class EmployeeTest {
   public static void main(String[] args) {
            Employee emp = new Employee(101, "John Doe", 75000.00, "Software Engineer", "IT");
        System.out.println(emp.displayInfo());
      emp.setSalary(80000.00);
      emp.setDesignation("Senior Software Engineer");
      System.out.println("\nUpdated Employee Information:");
      System.out.println(emp.displayInfo());
   }
}
```

4. **Output:**



Output of Java Bean

5. **Outcome of the Practical:** Students will learn and create a Java Bean for managing employee information. They will gain practical experience in object-oriented programming concepts, including encapsulation, and understand how to create and use Java Beans in Java applications.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**

- What is a Java Bean, and what are its key characteristics?
- Why is it important for a Java Bean to implement the Serializable interface?

- Can you explain the purpose of getter and setter methods in a Java Bean? How do they contribute to encapsulation?
- What is the role of the displayInfo() method in the Employee Java Bean? How does it enhance the functionality of the bean?
- How do you create an instance of a Java Bean, and how do you set its properties?
- What are some best practices for naming conventions when creating Java Beans?
- How can you validate input data in the setter methods of a Java Bean? Why is this important?
- What are the advantages of using Java Beans in Java applications?
- Can you describe a scenario where you would use a Java Bean in a web application?
- How would you enhance the Employee Java Bean to include additional features, such as calculating bonuses or handling promotions?

Practical No. 9

1. **Experiment Description:** To develop a command-line utility using Node.js that performs specific tasks, including converting text to uppercase, calculating the factorial of a number, and generating random passwords. This exercise aims to enhance understanding of Node.js, asynchronous programming, and user input handling in a command-line environment.

Materials Required: A computer with Node.js installed (version 12 or higher recommended).
A text editor or Integrated Development Environment (IDE) such as Visual Studio Code, Atom, or Sublime Text., Basic knowledge of JavaScript and Node.js. The utility is implemented in a single JavaScript file. The code utilizes the readline module for handling user input and the crypto module for generating random passwords. The utility presents a menu to the user, allowing them to choose from different functionalities.

Code Explanation:

- Imports:

const readline = require("readline");: Imports the readline module to facilitate reading input from the command line.
const crypto = require("crypto");: Imports the crypto module, although it is not used in the current implementation.

- Creating the Readline Interface:

const rl = readline.createInterface(...): Sets up the interface for reading input from the standard input (keyboard) and outputting to the standard output (console).

- Menu Definition: A string variable menu is defined to display the available options to the user.
- Function Definitions:

a) convertToUppercase(text): Takes a string input and returns it in uppercase.
b) calculateFactorial(num): Computes the factorial of a given number using a loop.
c) generateRandomPassword(length): Generates a random password of specified length using a set of alphanumeric characters.

- Main Function: The main() function displays the menu and prompts the user for their choice.

Based on the user's input, it calls the appropriate function and handles further input as needed.
The function uses a switch statement to determine which action to perform.

- User Interaction: The utility prompts the user for input and processes it asynchronously, allowing for a smooth user experience.

After completing an action, the utility returns to the main menu, allowing the user to perform additional tasks or exit.

Procedure:

- Create a new directory for the project and navigate into it.
- Create a new JavaScript file named utility.js.
- Code Implementation: code into utility.js.and Save the file.
- Run the Utility: Open a terminal or command prompt, Navigate to the directory where utility.js is located.
- Run the utility using the command: node utility.js
- Interacting with the Utility: The utility will display the menu. Enter a number corresponding to the desired action (1, 2, 3, or 4).

Follow the prompts to provide the necessary input for the selected action.
After each action, the utility will return to the main menu until the user chooses to exit by entering 4.
Testing Different Features:

- Test the text conversion feature by entering a string.
- Test the factorial calculation by entering a non-negative integer.
- Test the password generation feature by specifying a length for the password.
- Try entering invalid options to see how the utility handles errors.

2. **Objective:** Build a command-line utility using Node.js that performs a specific task, such as converting text to uppercase, calculating the factorial of a number, or generating random passwords.
3. **Code and Procedure:**

```
const readline = require("readline");
const crypto = require("crypto");
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
});
const menu = `
1. Convert text to Uppercase.
2. Calculate factorial of a number.
3. Generate random password.
4. Exit.
`;
const convertToUppercase = (text) => {
    return text.toUpperCase();
};
const calculateFactorial = (num) => {
    let result = 1;
    for (let i = 2; i <= num; i++) {
        result *= i;
    }
    return result;
};
const generateRandomPassword = (length) => {
    const characters =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
```

```
    let password = '';
    for (let i = 0; i < length; i++) {
        password += characters.charAt(Math.floor(Math.random() * characters.length));
    }
    return password;
};
const main = () => {
    console.log(menu);
    rl.question("Enter Your Choice (1/2/3/4): ", (answer) => {
        switch (answer) {
            case "1":
                rl.question("Enter text to convert to uppercase: ", (text) => {
                    console.log(convertToUppercase(text));
                    main();
                });
                break;
            case "2":
                rl.question("Enter a number to calculate factorial: ", (num) => {
                    console.log(calculateFactorial(parseInt(num)));
                    main();
                });
                break;
            case "3":
                rl.question("Enter password length: ", (length) => {
                    console.log(generateRandomPassword(parseInt(length)));
                    main();
                });
                break;
            case "4":
                console.log("Goodbye!");
                process.exit(0);
                break;
            default:
                console.log("Invalid choice, please try again.");
                main();
        }
    });
};
main();
```

4. Output:

```
Choose and option :
1: Convert text to Uppercase.
2: Calculate factorial of a number.
3: Generate random password.
4: Exit.
Enter Your Choice (1/2/3/4): 1
Enter text to convert to uppercase: hello world
Uppercase Text : HELLO WORLD

Choose and option :
1: Convert text to Uppercase.
2: Calculate factorial of a number.
3: Generate random password.
4: Exit.
Enter Your Choice (1/2/3/4): 2
Enter a number to calculate factorial: 5
```

Output of different cases for command-line utility using Node.js

5. **Outcome of the Practical:** Students will learn how to create a command-line utility using Node.js. They will gain experience in handling user input and output in a command-line environment. Students will understand how to implement basic algorithms (like factorial calculation) and string manipulation (like converting text to uppercase). They will also learn about generating random values and using built-in Node.js modules.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is Node.js, and how does it differ from traditional server-side technologies?
   - Explain the purpose of the readline module in your utility. How does it facilitate user input?
   - How does the asynchronous nature of Node.js affect the execution of your utility? Can you provide an example from your code?
   - Describe how you implemented the factorial calculation in your utility. What is the time complexity of your approach?
   - What are the potential limitations of your password generation function? How could you improve it?
   - How does the switch statement work in your utility? Why did you choose to use it instead of multiple if statements?
   - What would happen if a user inputs a negative number when calculating the factorial? How would you handle such cases?
   - Can you explain the significance of using parseInt() in your utility? What would happen if you didn't use it?
   - How would you modify your utility to include additional features, such as validating the length of the password or allowing for special characters?

- What are some best practices you followed while writing this utility, and what improvements would you suggest for future iterations?

Practical No. 10

1. **Experiment Description:** To develop a script that connects to a MongoDB database and utilizes the aggregation framework to analyze user data. Specifically, the script will filter users based on age, group them by city, calculate the average age of users in each city, and sort the results by average age in descending order.

Prerequisites:

- MongoDB: Have a MongoDB server running locally or remotely.
- MongoDB Collection: A collection named users in a database named userDB with documents containing at least the fields age and city.

Procedure:

Setup Environment: Ensure MongoDB is running and accessible.

- Create a database named userDB and a collection named users.
- Insert sample user documents into the users collection.
- Install Dependencies: Create a new Node.js project and install the MongoDB driver:
  a) npm init -y
  b) npm install mongodb
- Create the Script: Create a new JavaScript file (e.g., aggregateUsers.js) and copy the provided code into it.
- Run the Script: node aggregateUsers.js

2. **Objective:** Develop a script that uses MongoDB's aggregation framework to perform operations like grouping, filtering, and sorting. For instance, aggregate user data to find the average age of users in different cities

3. **Code and Procedure:**

aggregateUsers.js

```
const { MongoClient } = require("mongodb");
const uri = "mongodb://localhost:27017/";
const dbName = "userDB";
```

```
const collectionName = "users";
async function runAggregation() {
  const client = new MongoClient(uri);
  try {
    await client.connect();
    console.log("Connected to MongoDB");
    const db = client.db(dbName);
    const collection = db.collection(collectionName);
    const pipeline = [
      { $match: { age: { $gte: 18 } } },
      {
        $group: {
          _id: "$city",
          averageAge: { $avg: "$age" },
          userCount: { $sum: 1 },
        },
      },
      { $sort: { averageAge: -1 } },
    ];
    const result = await collection.aggregate(pipeline).toArray();
    console.log("Aggregation Results:");
    result.forEach((result) => {
      console.log(`City: ${result._id}, Average Age: ${result.averageAge.toFixed(2)}, User Count: ${result.userCount}`);
    });
  } catch (error) {
    console.error("Error:", error);
  } finally {
    await client.close();
  }
}
runAggregation();
```

4 Output:

```
Connected to MongoDB
Aggregation Results:
City: New York, Average Age: 30.50, User Count: 10
City: Los Angeles, Average Age: 28.75, User Count: 8
City: Chicago, Average Age: 25.00, User Count: 5
```

Output of mongodb for Aggregation result with grouping

5 **Outcome of the Practical:** Students will learn and develop a script that successfully connected to a MongoDB database and aggregated user data from the users collection. The script filtered out users under 18, grouped the remaining users by city, and calculated both the average age and total user count for each city, sorting the results in descending order by average age. The output was clearly formatted, displaying city names, average ages, and user counts. Participants gained hands-on experience with MongoDB's aggregation framework.

6 **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7 **Viva Questions/Review Questions:**

- What is MongoDB, and how does it differ from traditional relational databases?
- What are the key features of MongoDB?
- Explain the concept of a document in MongoDB. What is BSON?
- How does MongoDB handle data indexing, and why is indexing important?
- What is the aggregation framework in MongoDB, and how does it work?
- How does MongoDB ensure data consistency and durability?
- What are the differences between a primary and a secondary replica in a MongoDB replica set?
- How can you perform CRUD operations in MongoDB?
- What are some common use cases for MongoDB?
- How do you handle data migrations and schema changes in MongoDB?

Value Added Programs

Practical No. 11

1. **Experiment Description:** Assume four users user1, user2, user3 and user4 having the passwords pwd1, pwd2, pwd3 and pwd4 respectively. Write a servlet for doing the following. Create a Cookie and add these four user id's and passwords to this Cookie. 2.Read the user id and passwords entered in the Login form and authenticate with the values available in the cookies.

   Important steps to follow

   a. First install the tomcat into the system.
   b. Then make a subdirectly (eg., tr) in the \tomcat\webapps.
   c. Under tr create WEB-INF directory and also place the html files in this tr directory only.
   d. Next under WEB-INF create two subclasses lib, classes and web.xml
   e. Next place all the class files under the classes and jar files (servlet api.jar,classes12.jar etc…) under lib subdirectories.
   f. After this start tomcat by giving the following command at the instll_dir>tomcat>bin
   g. Catalina.bat run
   h. At the I.E(web browser) give the url as http;//localhost:8080//tr/htmlfile or servlet url pattern
   i. Portno 8080 is assigned for the tomcat.

Create a directory:

Create a directory "cookies", in that directory copy login.html file and create a directory a "WEB-INF". In that WEB-INF directory again create directory "classes" and copy web.xml file.

Compile the servlet:

Compile the servlet file then copy the class file of that servlet into the directory **C:\Program Files\Apache Software Foundation\Tomcat 6.0\cookies\WEB-INF\classes.**

Open the server:

1. Start tomcat by giving the following command at the instll_dir>tomcat>bin
   Catalina.bat run
2. At the I.E. (web browser) give the url as http://localhost:8080/ cookies /login.html

2. **Objective:** Write a servlet to create a Cookie and add four user id's and passwords to this Cookie.
3. **Code and Procedure:**

Login.html:

```
<html>
<body bgcolor="pink">
<form action="show" method="get">
<center>User Name: <input type="test" name="name"><br>
Password:<input type="password" name="pass"><br>
<input type="submit" name="b1">
<input type="Button" name="b2" value="Reset">
</center></form>
</body></html>
```

Login.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/** Example using servlet initialization. Here, the message
* to print and the number of times the message should be
* repeated is taken from the init parameters.
*/
public class login extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
String na=request.getParameter("name");
String pa=request.getParameter("pass");
PrintWriter out = response.getWriter();
Cookie nam1=new Cookie("user1","pace");
Cookie nam2=new Cookie("user2","phani");
Cookie nam3=new Cookie("user3","cse");
Cookie nam4=new Cookie("user4","ece");
Cookie pas1=new Cookie("pwd1","college");
Cookie pas2=new Cookie("pwd2","kumar");
Cookie pas3=new Cookie("pwd3","it");
Cookie pas4=new Cookie("pwd4","eee");
int flag=0;
String nam[]={nam1.getValue(),nam2.getValue(),nam3.getValue(),nam4.getValue()};
```

```
String pas[]={pas1.getValue(),pas2.getValue(),pas3.getValue(),pas4.getValue()};
for(int i=0;i<4;i++)
{
if(nam[i].equals(na)&&pas[i].equals(pa))
{
flag=1;
}
}
if(flag==1)
{
out.println("<title>The ShowMessage Servlet</title>");
out.println("<BODY BGCOLOR=\"#FDF5E6\">\n" +"<H1 ALIGN=CENTER>
WELCOME <br/>TO</br>" +na.toUpperCase() + "</H1>");
out.println("</BODY></HTML>");
}
else
{
out.println("<title>The ShowMessage Servlet</title>");
out.println("<BODY BGCOLOR=\"#FDF5E6\">\n" + "<H1 ALIGN=CENTER>
User is invalid </H1>");
out.println("</BODY></HTML>");
```

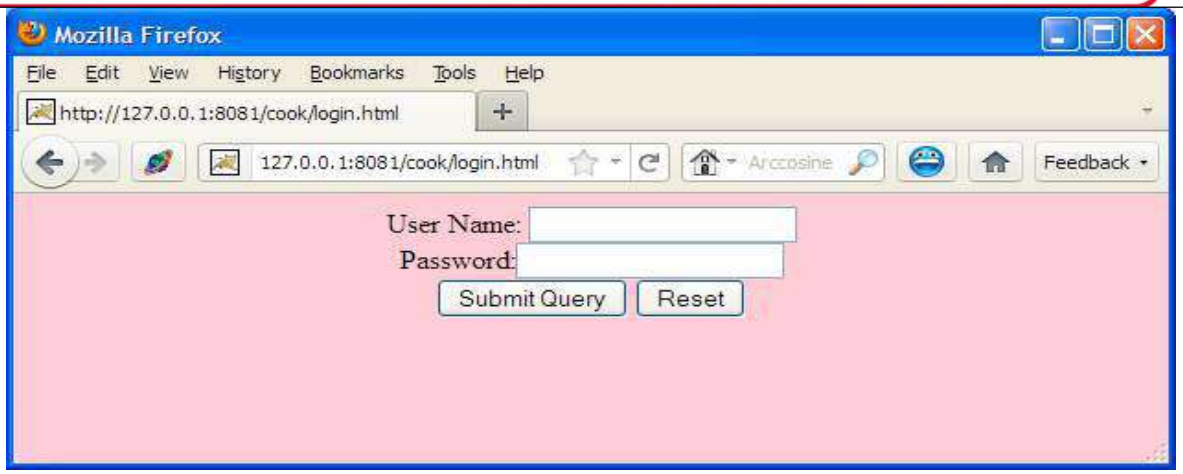


```
}
}
}
```

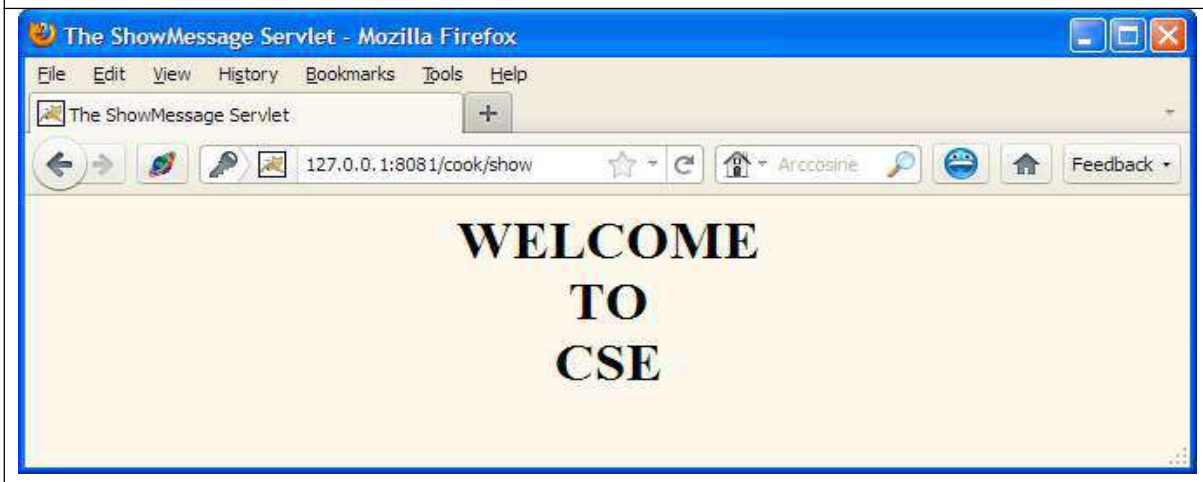
Web.xml:

```
<web-app>
<servlet>
<servlet-name>log </servlet-name>
<servlet-class>login </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>log </servlet-name>
<url-pattern> /show </url-pattern>
</servlet-mapping>
</web-app>
```

4. Output:



1.Output of login.html



2. Output of login.java

5. **Outcome of the Practical:** Students gained hands-on experience in servlet programming, handling HTTP requests, managing cookies, and configuring web applications in a Tomcat environment. This exercise reinforced the understanding of web technologies and the interaction between client-side and server-side components.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is the purpose of using cookies in web applications, and how do they work?
   - Can you explain the difference between GET and POST methods in HTTP requests? Why did you choose GET for the login form?
   - How does the servlet retrieve the parameters (username and password) from the login form?
   - What are the steps involved in creating and managing cookies in a Java servlet?
   - How does the servlet authenticate the user based on the cookies created? What logic is used to compare the entered credentials?
   - What is the role of the web.xml file in a Java web application, and how is it structured?

- How do you handle invalid login attempts in your servlet? What response is sent to the user?
- What are some security considerations when using cookies for authentication? How can you enhance the security of your application?
- How would you modify the servlet to store user credentials in a database instead of using cookies?
- What are the advantages and disadvantages of using cookies for session management in web applications?

Practical No. 12

1. **Experiment Description**: To develop a web application that allows users to register by providing their details (name, password, email ID, and phone number). The application will use a servlet or JSP to connect to a database, insert user data upon registration, and retrieve and display the registered users from the database.

Tools Required: Java Development Kit (JDK) for compiling and running Java programs.
Apache Tomcat server for deploying the web application.
A text editor or Integrated Development Environment (IDE) (such as Eclipse or IntelliJ IDEA) for writing Java code and JSP.
Important Steps:
Ensure that the database is accessible and that you have the necessary credentials to connect to it.
Create the Registration Form: Develop an HTML registration form (register.html) that allows users to input their name, password, email ID, and phone number. The form should have a submit button to send the data to the servlet or JSP for processing.
Write the Servlet/JSP: Create a Java servlet (e.g., RegistrationServlet.java) or a JSP file (e.g., register.jsp) that will handle the form submission. This component will:

Retrieve the user input from the request parameters.
Establish a connection to the database using JDBC (Java Database Connectivity).
Insert the user details into the Users table when the submit button is clicked.
Retrieve and display all registered users from the database.
Retrieve and Display Data: After inserting the user data, fetch all records from the Users table. Display the retrieved data in a user-friendly format (e.g., in an HTML table) on the same page or redirect to a new page that shows the list of registered users.
Compile and Deploy: Compile the servlet or JSP and deploy it on the Apache Tomcat server. Ensure that the necessary libraries are included in the project.
Testing the Application: Start the Tomcat server and access the registration form through a web browser. Fill in the form with user details and submit it. Verify that the data is correctly inserted into the database and that the list of registered users is displayed as expected.

2. Objective: Create a table which should contain at least the following fields: name, password, email-id, phone number Write a servlet/JSP to connect to that database and extract data from the tables and display them. Insert the details of the users who register with the web site, whenever a new user clicks the submit button in the registration page

3. Code and Procedure:

reg.html:

```
<html>
<head>
<title>Registration page</title>
</head>
<body bgcolor="#00FFFf">
<form METHOD="POST" ACTION="register">
<CENTER>
<table>
<center>
<tr><td> Username </td>
<td><input type="text" name="usr"></td></tr>
<tr><td> Password </td>
<td><input type="password" name="pwd"></td></tr>
<tr><td>Age</td>
<td><input type="text" name="age"></td></tr>
<tr><td>Address</td>
<td><input type="text" name="add"></td></tr>
<tr><td>email</td>
<td><input type="text" name="mail"></td></tr>
<tr><td>Phone</td>
<td><input type="text" name="phone"></td></tr>
<tr><td colspan=2 align=center><input type="submit" value="submit"></td></tr>
</center>
</table>
</form>
</body>
```

Login.html

```
<html>
<head>
<title>Registration page</title>
</head>
<body bgcolor=pink><center><table>
<form METHOD="POST" ACTION="authent">
```

```
<tr><td> Username </td>
<td><input type="text" name="usr"></td></tr>
<tr><td> Password </td>
<td><input type="password" name="pwd"></td></tr>
<tr><td align=center colspan="2"><input type="submit" value="submit"></td></tr>
</table></center>
</form>
</body>
</html>
```

Ini.java:

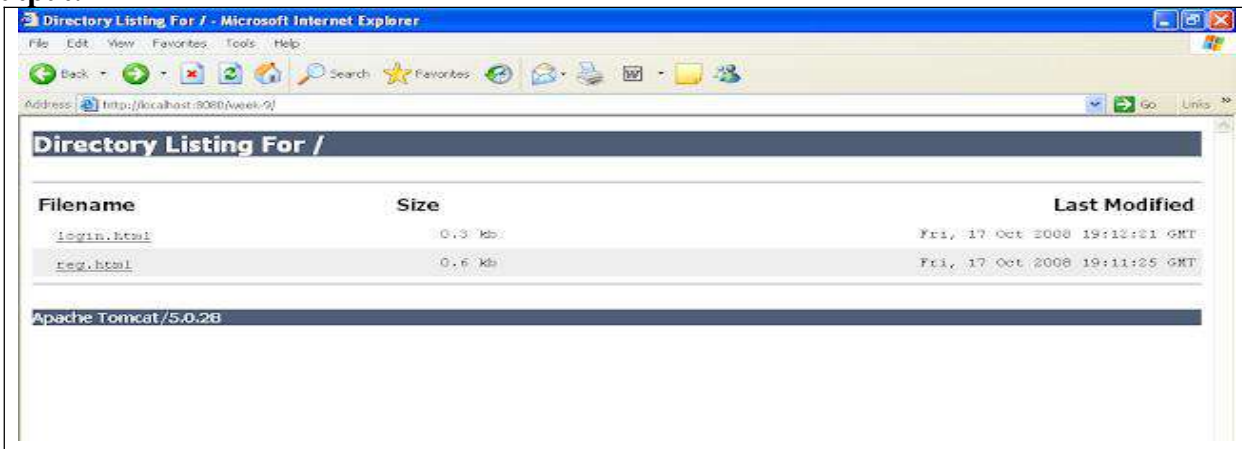
```
import javax.servlet.*;
import java.sql.*;
import java.io.*;
public class Ini extends GenericServlet
{
private String user1,pwd1,email1;
public void service(ServletRequest req,ServletResponse res) throws ServletException,IOException
{
user1=req.getParameter("user");
pwd1=req.getParameter("pwd");
email1=req.getParameter("email");
res.setContentType("text/html");
PrintWriter out=res.getWriter();
try
   {
    Class.forName("oracle.jdbc.driver.OracleDriver");
                                                                    Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@195.100.101.158:1521:cclab","scott","tige
r");
   PreparedStatement st=con.prepareStatement("insert into personal values(?,?,?,?,?,?)");
 st.setString(1,user1);
st.setString(2,pwd1);
st.setString(3,"25");
st.setString(4,"hyd");
st.setString(5,email1);
st.setString(6,"21234");
st.executeUpdate();
con.close();
  }
catch(SQLException s)
{  out.println("not found "+s);
}
catch(ClassNotFoundException c)
{
   out.println("not found "+c);
}
} }
```

web.xml:

```
<web-app>
<servlet>
```

```
<servlet-name>init1</servlet-name>
<servlet-class>Ini</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>init1</servlet-name>
<url-pattern>/regis</url-pattern>
</servlet-mapping>
</web-app>
```

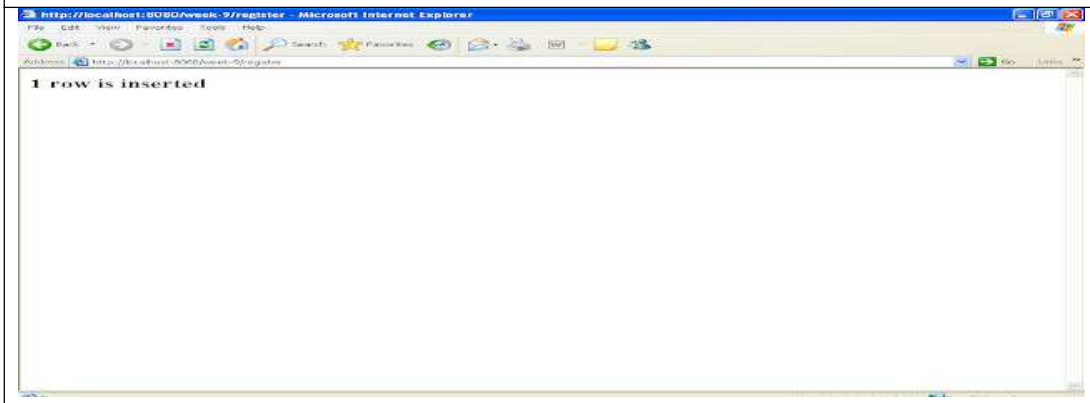
4.Output:



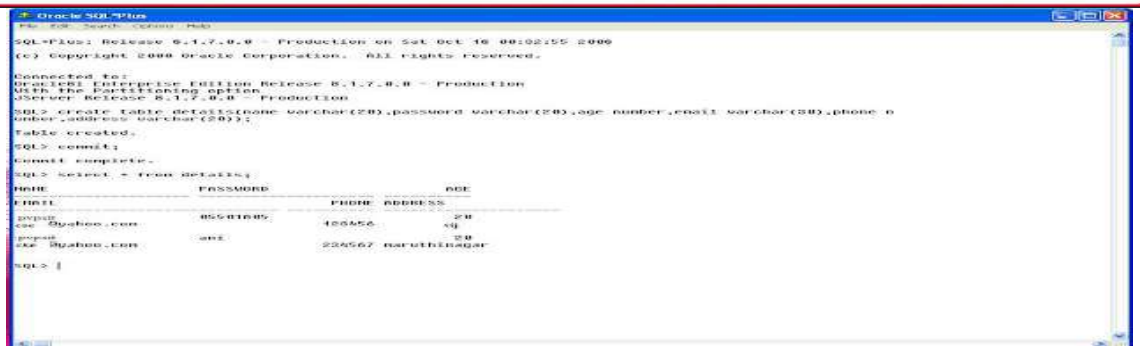
1.Output showing directory list



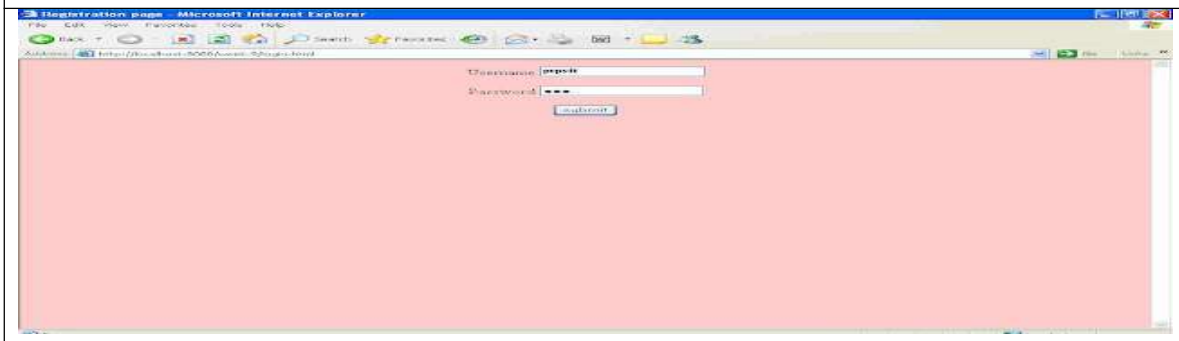
2.Output Registration Page



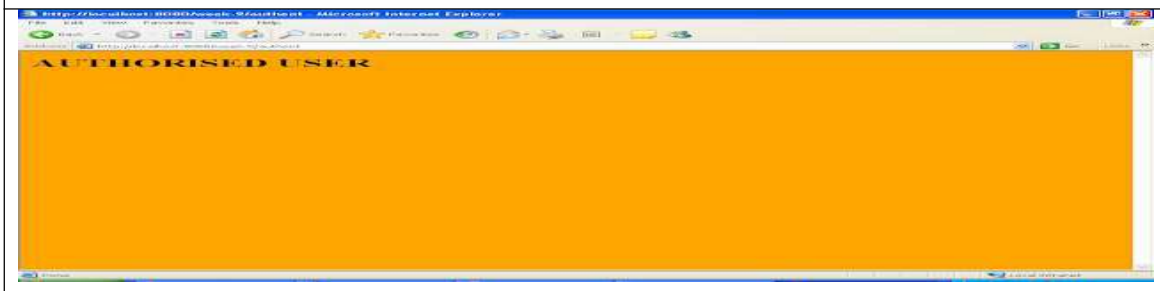
3. Output after inserting 1 Row in database



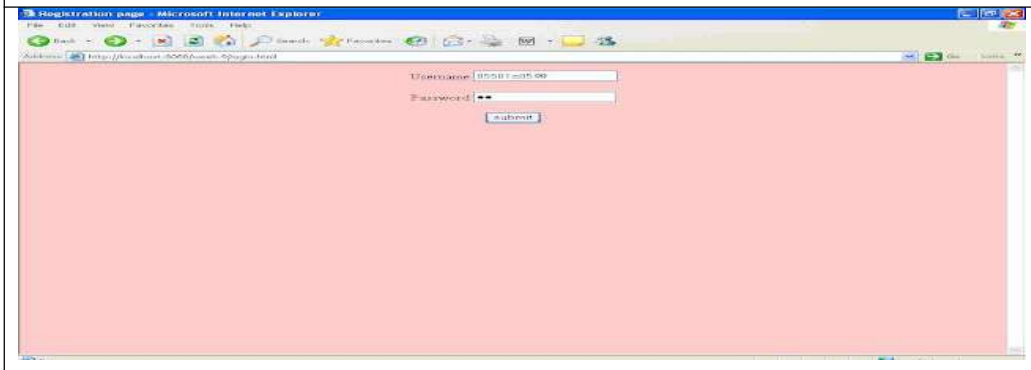
4. Output of database



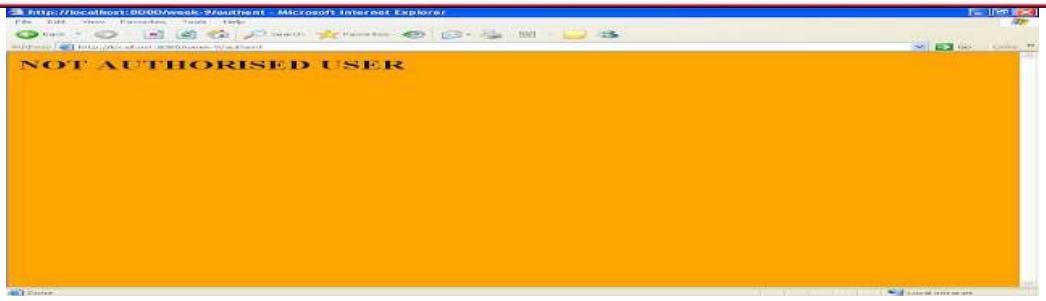
5. Login for authorised User



6. Output showing authorised User



7. Login for Unauthorised User



8. Output for Unauthorised User

5.Outcome of the Practical: Students learned and created a user registration system that allows users to register and view their details stored in a database. They will gain practical experience in servlet /JSP programming, database connectivity, and handling user input in web applications.

6.Assessment Criteria:

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7.Viva Questions/Review Questions:

- What is the role of a servlet in a Java web application? How does it differ from a traditional Java application?
- Can you explain the lifecycle of a servlet? What are the key methods involved?
- How do you retrieve form data submitted by a user in a servlet? What methods are used to access request parameters?
- What is the purpose of the web.xml file in a Java web application, and how is it structured?
- How do you handle user input validation in a registration form? What techniques can be used to ensure data integrity?
- What are the advantages of using JSP over servlets for generating dynamic web content?
- How can you manage session data in a web application? What methods are available for tracking user sessions?
- What is the difference between GET and POST methods in HTTP requests, and when would you use each in a registration form?
- How do you handle exceptions in a servlet or JSP? What strategies can be employed to provide user-friendly error messages?
- What security measures should be taken when handling user registration data, especially passwords

Practical No. 13

1. **Experiment Description:** To develop a web application using JavaServer Pages (JSP) that allows users to register by providing their details through a registration form. The application will store user information in a database and authenticate users when they submit a login form using their username and password.

Tools Required: Java Development Kit (JDK) for compiling and running Java programs.
Apache Tomcat server for deploying the web application.
A relational database management system for storing user data.
A text editor or Integrated Development Environment (IDE) (such as Eclipse or IntelliJ IDEA) for writing JSP code.

Steps:

Set Up the Database: Create a database (e.g., User DB) and a table (e.g., Users) with fields for user details, including a unique identifier, username, password, email, and phone number. Ensure that the database is accessible and that you have the necessary credentials to connect to it.

Create the Registration Form: Develop a user-friendly registration form that allows users to input their username, password, email, and phone number. The form should include a submit button that directs the user to the JSP page responsible for processing the registration.

Implement Registration Logic: In the JSP page that handles the registration, write the logic to process the form submission. This includes retrieving the user input, establishing a connection to the database, and inserting the user details into the database table.

Create the Login Form: Develop a login form that allows users to enter their username and password. This form should also include a submit button that directs the user to a JSP page for authentication.

Implement Authentication Logic: In the JSP page that handles the login, write the logic to process the login form submission. This includes retrieving the username and password, establishing a connection to the database, and checking the provided credentials against the stored records in the database.

Display Authentication Results: Based on the authentication result, provide feedback to the user. If the credentials are valid, display a welcome message. If the credentials are invalid, inform the user that the login attempt has failed.

Compile and Deploy: Compile the JSP files and deploy them on the Apache Tomcat server. Ensure that the necessary libraries, such as the JDBC driver, are included in the project setup.

Testing the Application: Start the Tomcat server and access the registration form through a web browser. Fill in the form with user details and submit it to verify that the data is correctly inserted into the database. Then, access the login form, enter the credentials, and check that the authentication process works as expected.

2.Objective: Write a JSP which insert the details of the 3 or 4 users who register with the web site by using registration form. Authenticate the user when he submits the login form using the user name and password from the database.

3.Code and Procedure:

Main.html:

```
<html>
<body>
<br /><br /><br /><br /><br />
<h1 align="center"><U>ONLINE BOOK STORAGE</U></h1><br /><br /><br />
<h2 align="center"><pre>
<b>Welcome to online book storage.
Press LOGIN if you are having id
otherwise press REGISTRATION
</b></pre></h2>
<br /><br /><pre>
<div align="center"><a href="login.html">LOGIN</a><a href="reg.html">
REGISTRATION</a></div></pre>
</body>
</html>
```

Login.html:

```
<html>
<body><br /><br /><br />
<form name="myform" method="post" action="login.jsp">
<div align="center"><pre>
LOGIN ID :<input type="text" name="id" /><br />
PASSWORD :<input type="password" name="pwd" /></pre><br /><br />
```

```
</div>
<br /><br />
<div align="center">
<input type="submit" value="ok" onclick="validate()" />
      <input type="reset" value="clear" />
</div>
</form>
</body>
</html>
```

Reg.html:

```
<html>
<body><br /><br />
<form name="myform" method="post" action="reg.jsp">
<table align="center">
<tr>
<td>NAME</td>
<td>:<input type="text" name="name" /></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>:<input type="text" name="addr" /></td>
</tr>
<tr>
<td>CONTACT NUMBER</td>
<td>:<input type="text" name="phno" /></td>
</tr>
<tr>
<td>LOGINID</td>
<td>:<input type="text" name="id" /></td>
</tr>
<tr>
<td>PASSWORD</td>
<td>:<input type="password" name="pwd" /></td>
</tr>
</table>
<br /><br />
<div align="center">
<input type="submit" value="ok" onclick="validate()" />
      <input type="reset" value="clear" />
</div>
</form>
</body>
</html>
```

Reg.jsp:

```
<%@page import="java.sql.*"%>
<%@page import="java.io.*"%>
<% response.setContentType("text/html");
out.println("<html><body>");
String name=request.getParameter("name");
String addr=request.getParameter("addr");
String phno=request.getParameter("phno");
```

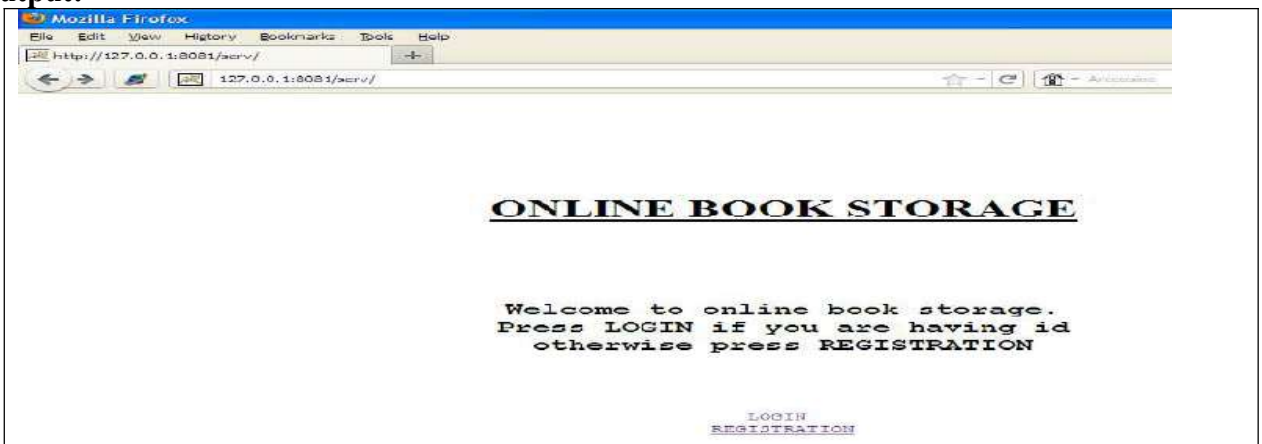
```
String id1=request.getParameter("id");
String pwd1=request.getParameter("pwd");
int no=Integer.parseInt(phno);
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","tiger")
;
Statement stmt=con.createStatement();
String sqlstmt="select id,pwd from login";
ResultSet rs=stmt.executeQuery(sqlstmt);
int flag=0;
while(rs.next())
{ if(id1.equals(rs.getString(1))&&pwd1.equals(rs.getString(2)))
{ flag=1;
}
}i
f(flag==1)
{ out.println("<br><br>SORRY INVALID ID ALREADY EXITS TRY AGAIN WITH
NEW
ID<br><br>");
out.println("<a href=\"reg.html\">press REGISTER to RETRY</a>");
} else
{ Statement stmt1=con.createStatement();
stmt1.executeUpdate("insert into login
values('"+name+"','"+addr+"','"+phno+"','"+id1+"','"+pwd1+"');");
out.println("<br><br>YOUR DETAILS ARE ENTERED<br><br>");
out.println("<a href=\"login.html\">press LOGIN to login</a>");
}
out.println("</body></html>");
con.close();
%>
```

Login.jsp:

```
<%@page import="java.sql.*"%>
<%@page import="java.io.*"%>
<html>
<body>
<% String id=request.getParameter("id");
String pwd=request.getParameter("pwd");
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","tiger")
;
Statement stmt=con.createStatement();
String sqlstmt="select id,pwd from login";
ResultSet rs=stmt.executeQuery(sqlstmt);
int flag=0;
while(rs.next())
{
if(id.equals(rs.getString(1))&&pwd.equals(rs.getString(2)))
{
flag=1;
```

```
}
}i
f(flag==0)
{
out.println("<br><br>SORRY INVALID ID TRY AGAIN ID<br><br>");
out.println("<a href=\"login.html\">press LOGIN to RETRY</a>");
} else
{ out.println("<br><br>VALID LOGIN ID<br><br>");
out.println("WELCOME <br>To<br>"+id);
} con.close();
%>
</body></html>
```

4.Output:



1.Output of main.html



2.Output of registration page



3.Output of Login Page

5.Outcome of the Practical: Students will learn and create a user registration and authentication system using JSP. They will gain practical experience in web application development, database connectivity, and secure handling of user input.

6.Assessment Criteria:

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7.Viva Questions/Review Questions:

- What is JSP, and how does it differ from servlets in terms of functionality and usage?
- Can you explain the lifecycle of a JSP page? What happens when a JSP page is requested by a client?
- How do you retrieve form data submitted by a user in a JSP page? What methods are typically used to access request parameters?
- What are the key components of a database table used for user registration? Why is it important to have a unique identifier for each user?
- How do you handle user input validation in the registration form? What techniques can be used to ensure that the data entered is correct and secure?
- What security measures should be implemented when storing user passwords in a database? Why is password encryption important?
- How can you manage user sessions in a JSP application? What methods are available for tracking user sessions?
- What is the purpose of the web.xml file in a Java web application, and how is it used to configure servlets and JSPs?
- How do you handle exceptions in JSP? What strategies can be employed to provide user-friendly error messages during registration and login?
- What are some best practices for developing secure web applications, particularly in the context of user registration and authentication?

Practical No. 14

1. Experiment Description: To design and implement a simple shopping cart application using Java Servlets and JSP (JavaServer Pages). The application will allow users to browse a list of products, select items to add to their shopping cart, and view the contents of their cart. The session tracking API will be utilized to maintain the cart's state across different pages, ensuring a seamless user experience.

Components Overview

The application consists of three main components:

Product Page (products.jsp): This page displays a list of products available for purchase. Each product is associated with a checkbox that allows users to select multiple items. A form is provided to submit the selected products to the server.

- Cart Servlet (CartServlet.java): This servlet processes the user's selections from the product page. It manages the shopping cart by adding selected products to the user's session. The servlet also handles the redirection to the cart view page after the products are added.
- View Cart Page (viewCart.jsp): This page displays the contents of the shopping cart. It retrieves the cart from the user's session and presents the selected products. If the cart is empty, a message is displayed to inform the user.
- Deployment Descriptor (web.xml): This XML file configures the servlet and its URL mapping, allowing the web server to route requests to the appropriate servlet.

Product Selection:

- Users navigate to the product page (products.jsp) where they can see a list of available products, each with a checkbox and a price.
- Users can select one or more products by checking the corresponding checkboxes.
- Upon clicking the "Add to Cart" button, the selected products are submitted to the server.

Adding to Cart:

- The form submission triggers the CartServlet, which processes the request.
- The servlet retrieves the current session and checks if a shopping cart already exists in the session.
- If the cart does not exist, a new cart is created. The servlet then adds the selected products to the cart.
- After updating the cart, the servlet redirects the user to the view cart page (viewCart.jsp).

Viewing the Cart:

- On the view cart page, the servlet retrieves the cart from the session.
- If the cart contains products, they are displayed in a list format. If the cart is empty, a message is shown to inform the user.
- Users have the option to return to the product page to continue shopping

.

Session Management: The application uses the session tracking API to maintain the state of the shopping cart. When a user adds products to the cart, the cart is stored in the user's session. This allows the application to remember the user's selections even as they navigate between different pages. The session is created when the user first accesses the application and is maintained until the user closes the browser or the session times out.

2.Objective: Design and implement a simple shopping cart example with session tracking API

3.Code and Procedure:

products.jsp (Product Page)

```
<% @ page import =”java.util.*” %>
<html>
<body>
```

```
<h2> Product List</h2>
<form action= “CartServlet” method = “post”>
<input type= “checkbox” name =” product” value= “Product1”> Product1-$100<br>
<input type= “checkbox” name =” product” value= “Product2”> Product2-$200<br>
<input type= “checkbox” name =” product” value= “Product3”> Product3-$300<br>
<input type= “Submit” value= “Add to Cart”> </form>
<a href=”viewCart.jsp”> View Cart</a>
</body> </html>
```

CartServlet.java (Servlet for adding to cart)

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

public class CartServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
IOException {
       HttpSession session = request.getSession();
       List<String> cart = (List<String>) session.getAttribute("cart");

       if (cart == null) {
          cart = new ArrayList<>();
       }

       String[] products = request.getParameterValues("product");

       if (products != null) {
          cart.addAll(Arrays.asList(products));
       }

       session.setAttribute("cart", cart);
       response.sendRedirect("viewCart.jsp");
   }
}
```

viewCart.jsp (View Cart Page)

```
<%@ page import="java.util.*" %>
<html>
<body>
<h2>Your Shopping Cart</h2>
<%
   HttpSession session = request.getSession();
   List<String> cart = (List<String>) session.getAttribute("cart");

   if (cart == null || cart.isEmpty()) {
      out.println("<p>Your cart is empty!</p>");
   } else {
      out.println("<ul>");
      for (String product : cart) {
         out.println("<li>" + product + "</li>");
      }
```

```
        out.println("</ul>");
    }
%>
<a href="products.jsp">Continue Shopping</a>
</body>
</html>
```

Web.xml (Deployment Descriptor)

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

    <servlet>
        <servlet-name>CartServlet</servlet-name>
        <servlet-class>CartServlet</servlet-class>
    </servlet>

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

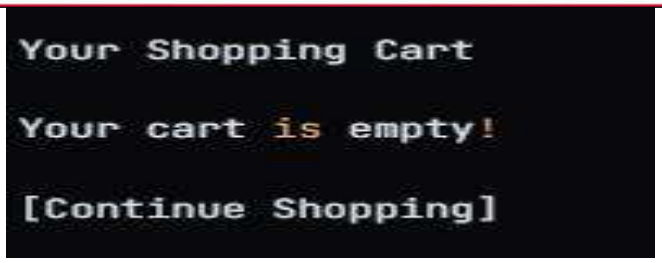
4. **Output:**



1.Output of product.jsp



2.After adding products to cart



3. **Output If cart is empty**



4. **Output of View Cart**

5. **Outcome of the Practical:** Students will learn and create a simple shopping cart application that utilizes session tracking to manage user interactions. They will gain practical experience in web application development using Java Servlets and JSP, as well as understanding session management in web applications.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is the purpose of session tracking in web applications, and why is it important for a shopping cart system?
   - How do you create a session in a Java web application using the HttpSession interface? What methods are commonly used?
   - Can you explain how to store and retrieve objects in a session? What types of objects can be stored in an HttpSession?
   - What is the difference between session tracking and cookie-based tracking? How does each method work?
   - How can you invalidate a session in a Java web application, and why is it important to do so?
   - What happens to the session data when a user closes their browser? How can you manage session persistence across multiple browser sessions?
   - How do you handle session timeouts in a web application? What configuration options are available to set session timeout values?
   - What are some potential security concerns related to session management, and how can you mitigate these risks?
   - How can you implement session attributes to store user-specific data, such as a shopping cart, and how do you access these attributes in different JSP pages?
   - Can you describe a scenario where you would need to use session tracking in a web application other than a shopping cart? What would be the use case?

Textbooks and References

Textbooks

Title	Author(s)	Description
HTML and CSS: Design and Build Websites	Jon Duckett	A comprehensive introduction to HTML and CSS, focusing on design aspects with a visually rich format.
JavaScript: The Good Parts	Douglas Crockford	Highlights the best features of JavaScript, providing insights into its strengths and weaknesses.
Head First Java	Kathy Sierra, Bert Bates	A beginner-friendly introduction to Java programming using a visually engaging format and interactive exercises.
Core Java Volume I: Fundamentals	Cay S. Horstmann	Covers the fundamentals of Java programming, including object-oriented concepts and GUI programming.
Java Servlet & JSP Cookbook	Bruce W. Perry	Provides solutions to common problems in Java Servlet and JSP development with practical examples.

Reference Books

Title	Author(s)	Description
Learning JavaScript Data Structures and Algorithms	Sammie Bae	Teaches data structures and algorithms using JavaScript, with practical examples for skill improvement.
Node.js Design Patterns	Mario Casciaro	Explores design patterns in Node.js for building scalable and maintainable applications.
MongoDB: The Definitive Guide	Shannon Bradshaw, et al.	A comprehensive guide to MongoDB covering features, architecture, and best practices for data modeling.
CSS: The Definitive Guide	Eric A. Meyer	A comprehensive reference for CSS, covering basic syntax to advanced layout techniques.
Web Development with Node and Express	Ethan Brown	A practical introduction to building web applications using Node.js and the Express framework.