AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय
Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

Department of Computer Engineering

"WT TERM WORK FILE"

Submitted By

Name of the Student: Piyusha Rajendra Supe

Roll No: 23CO315

Under the Guidance of

Dr. S. F. Sayyad

Sr. No.	Contents of the file
1	Write-up + Output of each practical
2	Mini-project report
3	Case Study
4	Theory assignment WT

ALL INDIA SHRI SHIVAJI MEMORIAL SOCIETY'S COLLEGE OF ENGINEERING PUNE-411001

Academic Year: 2024-25(Term-II)

Savitribai Phule Pune University

AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय
Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

WEB TECHNOLOGY INDEX

Expt. No	Title of the Experiment	Date
1	**Case study:** Before coding of the website, planning is important, students should visit different websites (Min. 5) for the different client projects and note down the evaluation results for these websites, either good website or bad website in following format: Sr. No. | Website URL | Purpose of Website | Things liked in the website | Things disliked in the website | Overall evaluation of the website (Good/Bad) From the evaluation, students should learn and conclude different website design issues, which should be considered while developing a website	20/01/25
2	Implement a web page **index.html** for any client website (e.g., a restaurant website project) using following: a. HTML syntax: heading tags, basic tags and attributes, frames, tables, images, lists, links for text and images, forms etc. b. Use of Internal CSS, Inline CSS, External CSS	27/01/25
3	Design the **XML document** to store the information of the employees of any business organization and demonstrate the use of: a) DTD b) XML Schema And display the content in (e.g., tabular format) by using CSS/XSL	03/02/25
4	Implement an application in **Java Script** using following: a) Design UI of application using HTML, CSS etc. b) Include Java script validation c) Use of prompt and alert window using Java Script e.g., Design and implement a simple calculator using Java Script for operations like addition, multiplication, subtraction, division, square of number etc. a) Design calculator interface like text field for input and output, buttons for numbers and operators etc. b) Validate input values c) Prompt/alerts for invalid values etc.	10/02/25
5	Implement the sample program demonstrating the use of **Servlet.**	03/03/25

	e.g., Create a database table ebookshop (book_id, book_title, book_author, book_price, quantity) using database like Oracle/MySQL etc. and display (use SQL select query) the table content using servlet	
6	Implement the program demonstrating the use of **JSP**. e.g., Create database table students_info (stud_id, stud_name, class, division, city) using database like Oracle/MySQL etc. and display (use SQL select query) the table content using JSP.	10/03/25
7	Build a dynamic web application using **PHP and MySQL**. a. Create database tables in MySQL and create connection with PHP. b. Create the add, update, delete and retrieve functions in the PHP web app interacting with MySQL database	17/03/25
8	Design a login page with entries for name, mobile number email id and login button. Use **struts** and perform following validations a. Validation for correct names b. Validation for mobile numbers c. Validation for email id d. Validation if no entered any value e. Re-display for wrongly entered values with message f. Congratulations and welcome page upon successful entries	24/03/25
9	Design an application using **Angular JS**. e.g., Design registration (first name, last name, username, password) and login page using Angular JS	07/04/25
10	Design and implement a business interface with necessary business logic for any web application using **EJB**. e.g., Design and implement the web application logic for deposit and withdraw amount transactions using EJB	14/04/25
11	**Mini Project**: Design and implement a dynamic web application for any business functionality by using web development technologies that you have learnt in the above given assignments.	**21/04/25**
12	**Case Study:** Transform the blogging application from a loose collection of various resources (servlets, HTML documents, etc.) to an integrated web application that follows the MVC paradigm	**18/03/25**
13	**Theory Assignment WT**	**07/04/25**

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Assignment 1

Statement - Case study:

Before coding of the website, planning is important, students should visit different websites (Min. 5) for the different client projects and note down the evaluation results for these websites, either good website or bad website in following format, From the evaluation, students should learn and conclude different website design issues, which should be considered while developing a website.

Sr. No.	Website URL	Purpose of Website	Things Liked in the Website	Things Disliked in the Website	Overall Evaluation
1	www.khanacademy.org	Educational	User-friendly interface, interactive content, responsive design	Limited color palette may seem dull to younger users	Good
2	www.espn.com	Sports	Real-time updates, multimedia integration, clear categorization	Too many ads, slow loading on mobile devices	Good
3	www.amazon.com	E-commerce	Wide variety of products, advanced filtering, and search options	Overwhelming homepage, too many pop-ups	Good
4	www.wikipedia.org	Informational	Simple interface, multilingual support, ad-free	No visual appeal, text-heavy	Good
5	www.nike.com	Brand/E-commerce	Clean design, high-quality images, interactive features	Slightly slow loading on image-heavy pages	Good
6	www.reddit.com	Community/Social	Community-driven content, minimalistic design	Cluttered layout, outdated design	Bad

7	www.etsy.com	E-commerce for Artisans	Beautiful product presentation, easy navigation	Limited payment options in some regions	Good
8	www.olx.in	E-commerce	Easy-to-use posting, localized search	High number of fake listings	Bad
9	www.codecademy.com	Educational	Interactive learning modules, responsive design	Limited free content	Good
10	www.iplay.com	Gaming	Engaging animations, vibrant design	Slow load time for large games	Bad
11	www.facebook.com	Social Media	Wide reach, easy sharing features	Privacy concerns, ad-heavy	Bad
12	www.flipkart.com	E-commerce	Personalized recommendations, robust payment options	Occasional technical glitches during sales	Good
13	www.architecturaldigest.com	Magazine	Stunning visual content, intuitive navigation	Text readability issues on mobile	Good
14	www.medium.com	Blogging	Simple layout, good readability, clean interface	Limited customization for writers	Good
15	www.cricbuzz.com	Sports	Real-time updates, clear stats	Slightly crowded layout	Good

Page No. : 1
Date. : / /

Experiment - 02

AIM: Implement a webpage index.htm for any client website (e.g. a restaurant website project) using the following -

a) HTML Syntax: heading tags, basic tags, and attributes, frames, tables, images, lists, links for text and images, forms, etc.

b) Use of internal css, Inline css, External css.

THEORY :

I] HTML and CSS -

- HTML (Hypertext Markup language) and CSS (Cascading Style Sheets) are the fundamental technologies for building and designing web pages.
- HTML is used to create structure of webpage by defining elements such as headings, paragraphs, images, and links. It acts as the back bone of a website.
- CSS is used to style and enhance the appearance of HTML elements by controlling colors, fonts, layouts and animations.
- Together, HTML and CSS allow developers to build visually appealing, structured and responsive webpages that improve user experience and engagement.

For Educational Use Only

(1) HTML Tags.

1. Heading Tags -
<h1> to <h6> - Define headings, with <h1> being the largest and <h6> the smallest, used for organizing content.

2. Basic Tags -
- <html> - The root element that contains all HTML Content.
- <head> - Contains meta information, title, and links to external resources.
- <title> - Sets the title of the webpage, displayed in the browser tab.
- <body> - Contains all visible content of the web page
- <p> Defines a paragraph for text content.
-
 - Inserts a line break within text.
- <hr> - Creates a horizontal lines to separate sections
- <b> - Makes text bold for emphasis.
- <i> - Italicizes text for emphasis or stylistic purposes.
- <u> - Underlines text for emphasis.
- <img> - Displays an image on the webpage.
- <a> - Creates hyperlinks to navigate between pages or external websites.

For Educational Use Only

(2). HTML Attributes –

- href (in <a>) – Specifies the URL of the link destination
- src (in <img>) – Defines the source file path of an image.
- alt (in <img>) – Provides alternative text for an image (useful for accessibility and SEO).
- width & height (in <img>) – Set the dimensions of an image.
- id – Assigns a unique identifier to an element for styling or scripting.
- class – Assign a reusable class name for CSS styling.
- Style – Allows inline CSS styling for individual elements.
- target (in <a>) – Defines how a link should open (-blank for a new tab, -self for the same tab).
- title – Provides additional information when the user hovers over an element.

These tags and attributes collectively help in structuring and enhancing the index.htm page for your website.

For Educational Use Only

* CSS (Cascading Style Sheets) is used to style HTML elements. Selectors help target specific elements for styling -

- Basic Selectors -
  - Universal selector that applies styles to all elements.
  - Following are syntaxes of each selector -

1. element (eg. p, h1) - Targets all elements of a specific type.
2. # id - Selects an element with a specific id attribute.
3. .class - Selects all elements with a specified class attribute.
4. Grouping and Combinators -
   - Selector 1, selector 2 - Applies the same style to multiple elements.
   - Selector 1 Selector 2 - Targets elements inside another element (descendent Selector)
   - Selector 1 > Selector 2 - Targets direct child elements.
   - Selector 1 + Selector 2 - Targets the adjacent sibling element.
   - Selector 1 ~ Selector 2 - Targets all sibling elements after the first one.

For Educational Use Only

(2). CSS Properties and values -

1. Text and Font styling -

- Color - sets the text color (eg - color : red;).
- font-size - Defines the size of the text (eg - font-size : 16px;)
- font-family - Specifies the font type (eg. font-family : Arial;)
- font-weight - Defines the thickness of the text (eg - font-weight : bold;)
- text-align - Aligns text (left, center, right)
- text-decoration - Adds or removes text decorations (underline, none).
- line-height - Sets the space between lines of text.

2. Background and Box styling -

- background-color - Sets the background color
- background-image - Sets an image as a background.
- border - Defines the border (width, style, color)
- border-radius - Rounds the corners of an element
- margin - Sets the outer space around an element.
- padding - sets the inner space inside an element.

3. Layout and Positioning -

- display - Specifies the display behaviour (block, inline, grid, flex).
- position - Defines the positioning (static, relative, absolute, fixed).

For Educational Use Only

- top - left, right, bottom - Positions an element when position is not static.

- z-index - Controls the stack order of elements
- overflow - Handles content overflow (hidden, scroll).

4. Flexbox and grid -

- display : flex; - Enables the flexbox layout.
- justify-content - Aligns items horizontally in a flex container (center, space-between).
- align-items - Align items vertically (flex-start, center).
- grid-template-columns - Defines columns structure in a grid layout.

5. Animations and Effects -

- Opacity - Controls transperency (0 for fully transperent, 1 for opaque).
- transitions - Adds smooth animations to property changes.

* **Conclusion** : Thus, an index.htm page was successfully created using the html tags and css attributes., properties.

9/2/25

* * * * * * *

For Educational Use Only

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Assignment 2

Implement a web page index.htm for any client website (e.g., a restaurant website project) using following:

a. HTML syntax: heading tags, basic tags and attributes, frames, tables, images, lists, links for text and images, forms etc

b. Use of Internal CSS, Inline CSS, External CSS

CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hospital Navbar & Footer</title>
    <style>
        h1{
            color: darkblue;
        }
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f2f2f2;
        }
        .navbar {
            display: flex;
            justify-content: center;
            background-color: #005bb5;
```

```
    padding: 10px;
}
.navbar a {
    color: white;
    text-decoration: none;
    padding: 10px 20px;
    margin: 0 10px;
    font-size: 18px;
}
.navbar a:hover {
    background-color: #003f80;
    border-radius: 5px;
}
.footer {
    background-color: #0073e6;
    color: white;
    text-align: center;
    padding: 15px;
    position: fixed;
    bottom: 0;
    width: 100%;
}
 body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f2f2f2;
}
.form-container {
    width: 80%;
    margin: auto;
    background: white;
```

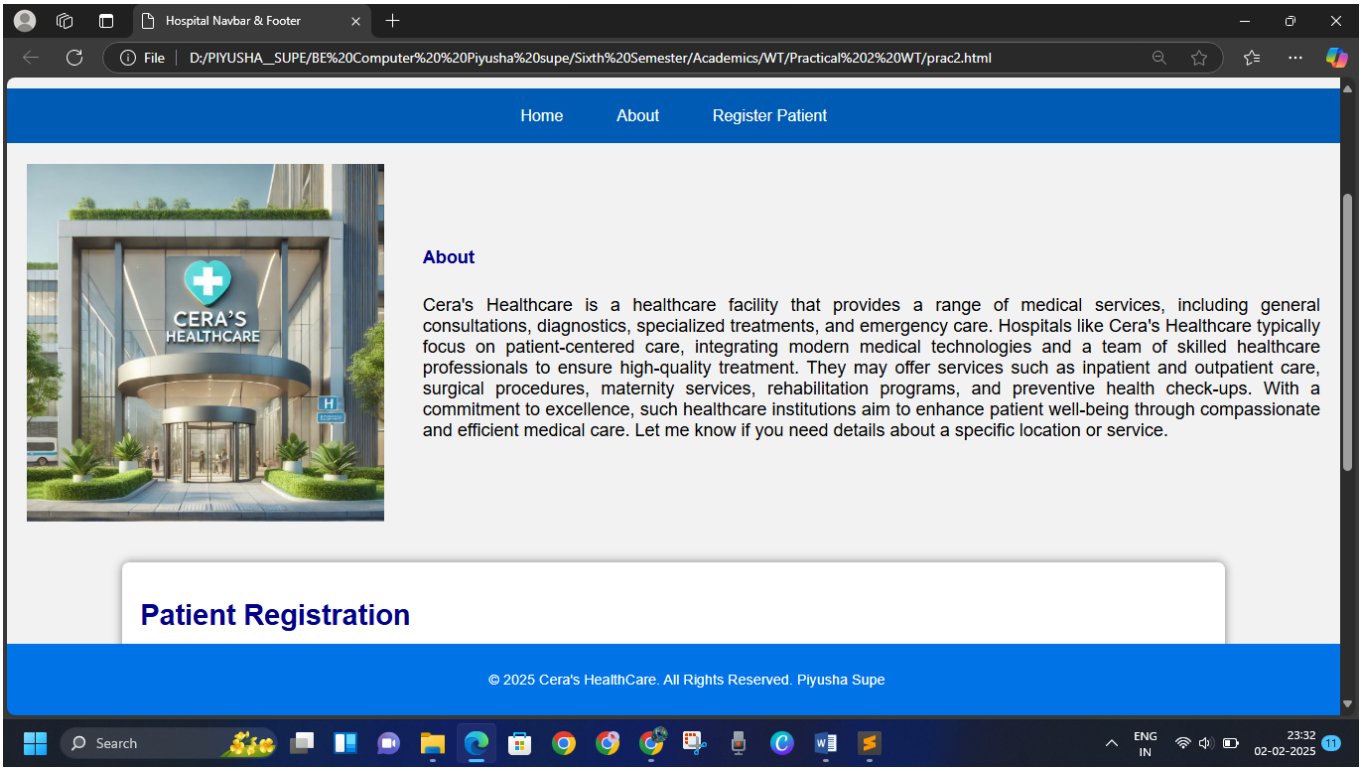
```
        padding: 20px;
        border-radius: 8px;
        box-shadow: 0px 0px 10px gray;
        margin-top: 20px;
    }
    input, select, textarea {
        width: 100%;
        padding: 8px;
        margin-top: 10px;
        border: 1px solid #ccc;
        border-radius: 4px;
    }
    button {
        width: 100%;
        padding: 10px;
        background-color: #0073e6;
        color: white;
        border: none;
        cursor: pointer;
        margin-top: 10px;
    }
    button:hover {
        background-color: #005bb5;
    }
    .box{
        padding: 10px;
        margin: 10px;
        font-size: 20px;
        text-align: justify;
    }

</style>
```

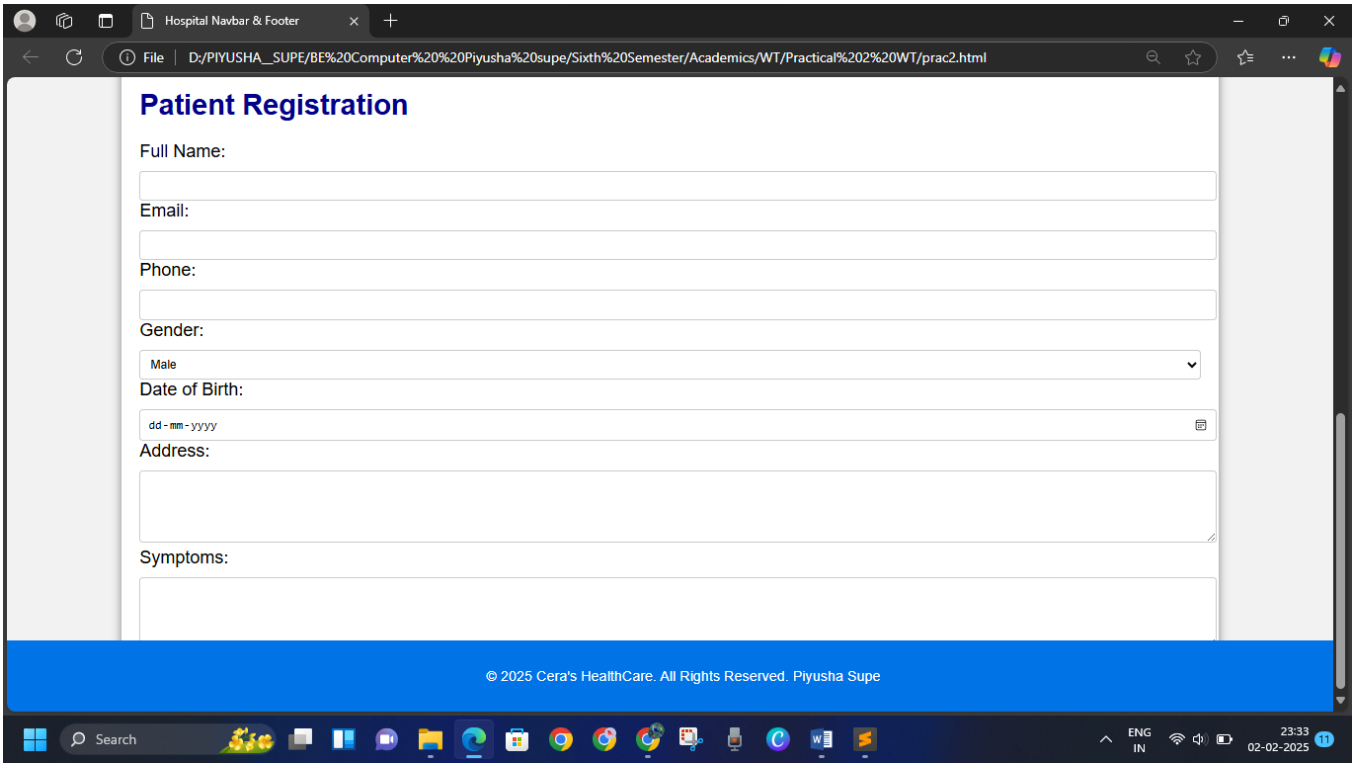
```
</head>
<body>
   <div>
      <center><img src="logo.jpg" height="150px" width="150px" style="border-radius: 60px;">
      <h1 style="color: darkblue; font-size: 40px; font-family: monospace; border-radius: ; ">Cera's Healthcare</h1>
      </center>
   </div>
   <div class="navbar">

      <a href="#home">Home</a>
      <a href="#about">About</a>
      <a href="#contact">Register Patient</a>

   </div>
   <div ><center>
      <table>
         <tr></tr>
      <tr> <td>
      <img src="hospital.jpg" height="400px" width="400px" class="box"></td><td><h1 id="about" class="box">About </h1>
      <p class="box">Cera's Healthcare is a healthcare facility that provides a range of medical services, including general consultations, diagnostics, specialized treatments, and emergency care. Hospitals like Cera's Healthcare typically focus on patient-centered care, integrating modern medical technologies and a team of skilled healthcare professionals to ensure high-quality treatment. They may offer services such as inpatient and outpatient care, surgical procedures, maternity services, rehabilitation programs, and preventive health check-ups. With a commitment to excellence, such healthcare institutions aim to enhance patient well-being through compassionate and efficient medical care. Let me know if you need details about a specific location or service.</p></td></tr>
   </table>  </center>
   </div>
   <div class="form-container" id="contact">
      <h1>Patient Registration</h1>
      <form style="font-size: 20px;">
         <label>Full Name:</label>
         <input type="text" required>
```

```
        <label>Email:</label>
        <input type="email" required>

        <label>Phone:</label>
        <input type="tel" required>

        <label>Gender:</label>
        <select>
            <option>Male</option>
            <option>Female</option>
            <option>Other</option>
        </select>

        <label>Date of Birth:</label>
        <input type="date">

        <label>Address:</label>
        <textarea rows="4"></textarea>

        <label>Symptoms:</label>
        <textarea rows="4"></textarea>

        <button type="submit">Submit</button>
    </form>
  </div>
  <div class="footer">
    <p>&copy; 2025 Cera's HealthCare. All Rights Reserved. Piyusha Supe</p>
  </div>
</body>
</html>
```

OUTPUT:

The output is as follows:





CONCLUSION:

Thus I have successfully created a website for Hospital portal using HTML and CSS tags

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

WT.

Experiment - 03.

- **Aim:** Design the XML document to store the information of employees of any business organisation and demonstrate use of:
  a) DTD.
  b) XML Schema.
  c) And display the content in (eg. tabular format) by using CSS/XSL.

- **Theory:**

(1) XML:

- XML stands for extensible markup language. It's a markup language much like HTML, but whereas HTML is used to display data, XML is used to store and transport data
- It's a flexible way to create common information formats and share structured data over the internet, making it a common tool in web development, data exchange and various other application Some key points of XML:
  1) Markup language
  2) Extensible.
  3) Platform independent.
  4) Human and machine readable.
  5) Heirarchical structure.
  6) Well defined structure.

13.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- Basic XML Syntax -

```
<? xml version = "1.0" encoding = "UTF-0-8"?>
<! DOCTYPE root SYSTEM "file. dtd" >
    <root>
        <element attribute = "value" > Content
        <!element>
    <!root>
```

(1)

(2). Structure for employee information -

- Root element : <employee>
- Child element : <employee> (Each employee's data).
- Attributes : <employee> has an 'id' attribute.
- Nested elements: <name>, <department>, <position>, <salary>, <contact>
- Contact information: <email> and <phone>

(3). Document type definition (DTD) -

- DTD is a set of rules that define the structure and allowed elements of an XML document. It ensures that an XML file follows a specific format and maintains consistency.
- Ensures data validity checking whether an XML document follows the specified format.
- Helps applications understand the expected hierarchy and relationships within an XML file.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- **Types of DTD -**

(1) **Internal:** Defined inside XML document using <! doctype >

Syntax:

```
<? xml version = "1.0" encoding = "UTF-8" ? >
<! DOCTYPE root [
        <! ELEMENT root (child1, child2) >
        <! ELEMENT child1 (#PCDATA) >
]>
<root>
        <child1> Data1 </child1>
        <child2> Data2 </child2>
</root>
```

#PCDATA - Means elements contain text, parsed character data.

<! ELEMENT > - Specifies elements and their content.

(2) **External DTD:**

```
<! DOCTYPE root SYSTEM "example.dtd" >
```

4] **XML Schema -**

- An XML Schema (XSD - XML Schema definition) is a powerful way to define the structure, data types and rules for an XML document.
- Supports datatypes like integers, decimals, dates, etc.
- Allows reusability of data definitions.
- Provides namespace support for XML document.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- **Basic syntax of XML Schema –**

```
<xs: schema xmlns:xs = "http://w3.org/2001/XMLSchema"
    <!-- Define elements -->
</xs: schema>
```

- **XML Schema components –**

1. Element – `<xs: element name = "elementName" type = "DT"/>`
2. Attribute declaration –

```
<xs: attribute name = "attributeName" type = "Datatype"
    use = "required/optional"/>
```

3. **Datatypes in XML Schema –**

Data type.	Description.
xs: string	Text content.
xs: integer.	Whole numbers
xs: decimal.	Decimal numbers
xs: boolean.	true or false.
xs: date.	Date format (YYYY-MM-DD)
xs: time.	Time format (HH:MM:SS).

Eg: `<xs: element name = "salary" type = "xs: decimal"/>`

4. You can also add constraints such as –
   - xs: minLength
   - xs: maxLength.
   - xs: restriction

- Advantages of Schema -

- Supports data types like integer, date, boolean, etc.
- Better validation with contraints like min/max value.
- Reusable components using complex types.
- Supports namespace for handling multiple Schema.
- More readable and scalable than DTD.

5) XSLT : Extensible Stylesheet language Transformations.

- Converts XML to readable formats.

Basic XSLT Syntax:

```
<xsl : stylesheet version = "1.0"  xmlns : xsl = "http://www.w3.org/1999/XSL/Transform "/>
    < xsl : template match = "/ " >
        <html>
            <body>
                <table>
                    < xsl : for-each select = "Root Element / ChildElement " >
                    <tr>
                        <td > <xsl : Value of Select = "element" /> </td
                    </tr>
                    </xsl : for each >
                </table>
            </body>
        </html>
    </xsl : template>
</xsl : stylesheet >
```

17.

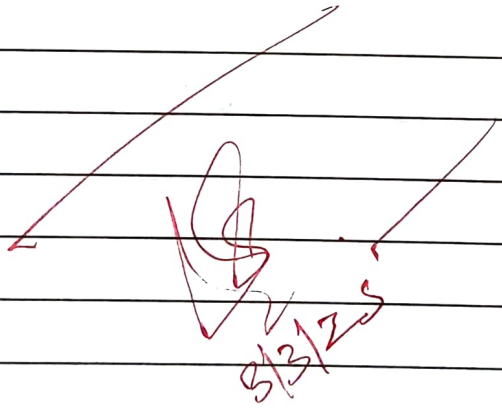
A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

* XSLT -

- <xsl: for-each select = "employees / employee ">
  - loops through all <employee> elements
- <td> <xsl: value-of select = "element" /> </td>
  - Extracts all and displays element values inside table cells.

- **Conclusion:** Thus we successfully studied and implemented xml, dtd and XML schema, Stylesheets.

* * * * * * *



Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Practical 3

Title: Design the XML document to store the information of the employees of any business organization and demonstrate the use of:

a) DTD

b) XML Schema

And display the content in (e.g., tabular format) by using CSS/XSL.

The files are as follows:

[1] Employee.xml – Normal xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="piyushastyle.xsl"?>
<!DOCTYPE PIU_Companyemployee SYSTEM "employee.dtd">
<PIU_Companyemployee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="employee.xsd">
   <emp>
      <name>Piyusha Supe</name>
      <age>20</age>
      <phone>1234567890</phone>
      <email>piyu@gmail.com</email>
      <salary>900000</salary>
      <location>Pune</location>
   </emp>
   <emp>
      <name>Swapnakirti</name>
      <age>28</age>
      <phone>7578378</phone>
      <email>sk@gmail.com</email>
      <salary>550002</salary>
      <location>Bangalore</location>
```

```
</emp>
<emp>
    <name>Brielle</name>
    <age>29</age>
    <phone>0987654321</phone>
    <email>bll@gmail.com</email>
    <salary>20000</salary>
    <location>Pune</location>
</emp>
<emp>
    <name>Cindy Ella</name>
    <age>39</age>
    <phone>7248278</phone>
    <email>cindy@gmail.com</email>
    <salary>80000</salary>
    <location>Pune</location>
</emp>
<emp>
    <name>Maxton Parker</name>
    <age>34</age>
    <phone>372478784</phone>
    <email>maxton@gmail.com</email>
    <salary>300000</salary>
    <location>Mumbai</location>
</emp>
<emp>
    <name>Piper Dcosta</name>
    <age>33</age>
    <phone>88987878</phone>
    <email>piper@gmail.com</email>
    <salary>940000</salary>
    <location>Pune</location>
```

```
   </emp>
</PIU_Companyemployee>
```

[2] Employee.dtd – DTD file document type definition

```
<!ELEMENT PIU_Companyemployee (emp+)>
<!ELEMENT emp (name, age, phone, email, salary, location)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
<!ELEMENT location (#PCDATA)>
```

[3] Piyushastyle.xsl – XSL for styling

```
<xsl:stylesheet version = "1.0"
xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "/">
        <html>
                <body>
                        <h2>PIU Company Employee details</h2>
                        <table border = "1">
                        <tr bgcolor = "lightpink">
                                <th>Name</th>
                                <th>Age</th>
                                <th>Phone</th>
                                <th>Email</th>
                                <th>Salary</th>
                                <th>Location</th>
                        </tr>
                        <xsl:for-each select="PIU_Companyemployee/emp">
                                <tr>
                                        <td><xsl:value-of select = "name"/></td>
```

```
                    <td><xsl:value-of select = "age"/></td>
                    <td><xsl:value-of select = "phone"/></td>
                    <td><xsl:value-of select = "email"/></td>

                    <td><xsl:value-of select = "salary"/></td>
                    <td><xsl:value-of select = "location"/></td>
                </tr>
            </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

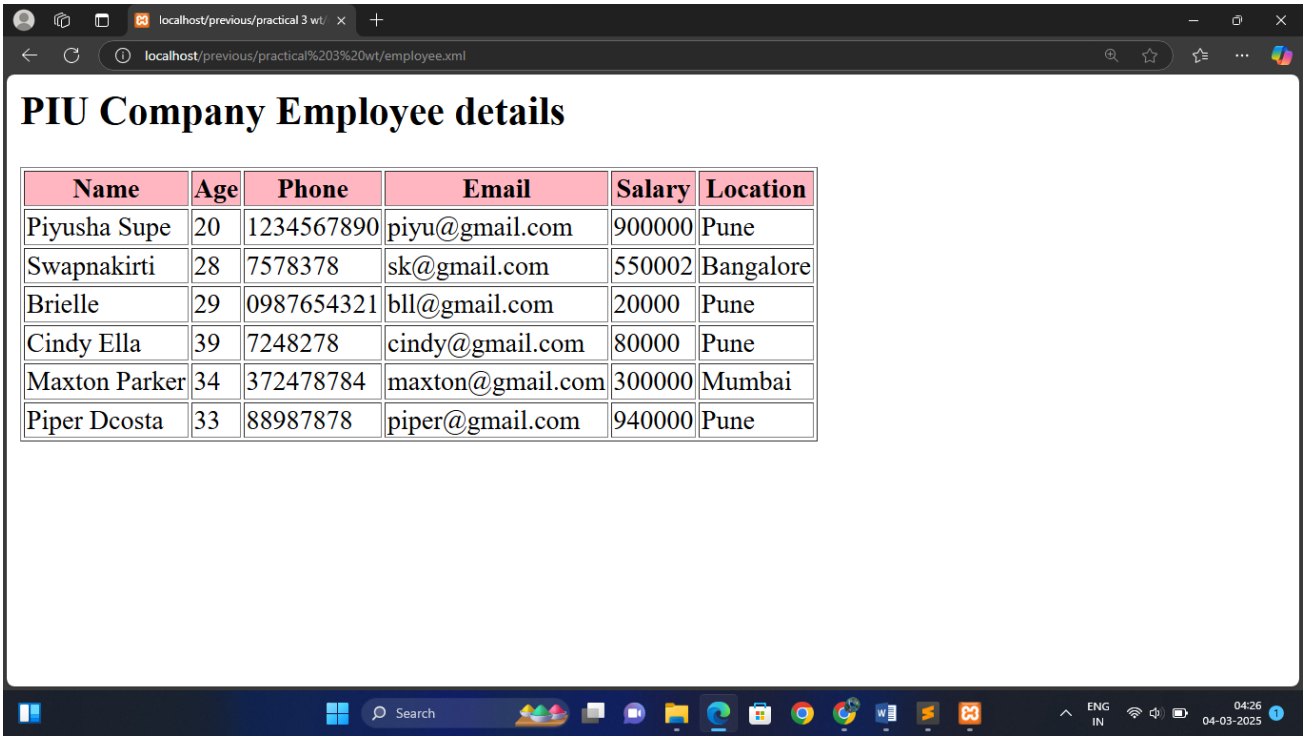
[4] Employee.xsd – Schema file

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="PIU_Companyemployee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="emp" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="age" type="xs:integer"/>
              <xs:element name="phone" type="xs:string"/>
              <xs:element name="email" type="xs:string"/>
              <xs:element name="salary" type="xs:integer"/>
              <xs:element name="location" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
```

```
        </xs:complexType>
    </xs:element>
</xs:schema>
```

OUTPUT:

The newer versions of browsers do not support running xsl sometimes so my solution was to run it with xampp. As xampp has apache server it interprets the xml and related files easily.



CONCLUSION:

Thus the XML, DTD, XSL, Schema were successfully executed as illustrated above.

Experiment-04

- **Aim :** Implement an application in javascript using following - (a) Design UI of application using HTML, CSS, etc.
  (b). Include Javascript validation.
  (c) Use of prompt and alert window using javascript.

eg - Design and implement a simple calculator using JS. for operations like addition, multiplication, subtraction, division, square of a number, etc.

a) Interface like text field for input, output, buttons for numbers and operators, etc.
b) Validate input values.
c) Prompts / alerts for invalid values, etc.

- **Theory :**

- A calculator application is a fundamental project that demonstrates the use of HTML, CSS and JS. in web development.
- This application enables users to perform basic arithmetic operations including addition, multiplication, etc.

HTML : Structures the calculators interface.
CSS : Enhances visual appeal and user experience.
JS : Adds interactivity, validation, dynamic calculations.

For Educational Use Only

I] UI design and Interface using HTML, CSS –

HTML for calculator structure.

- Input field – Displays user input and results.
- Buttons – Represent digits (0-9) operators (+, -, ×, ÷) and functions like clear (C), equal (=).

Example syntax –

```
<input type = "text" id = "display" readonly >
<button onclick = "appendNumber(1)" > 1 </button >
<button onclick = "appendOperator('+')" > + </button >
```

II] CSS for styling –

CSS improves the visual layout, making the calculator user friendly. It controls –

- Button size and alignment for better usability.
- Colors and backgrounds for a modern look.
- Input field styling – to make output readable.

CSS –

```
# display {
    width : 100%;
    font-size : 20px;
    text-align : right;
    padding : 5px;
}
```

For Educational Use Only

```
button {
    width : 50 px. ;
    height : 50 px. ;
    font-size : 18 px;
    margin : 5 px;
}
```

III] Javascript Implementation –

- JS is responsible for processing input, performing calculation and handling errors.

(a) Handling user input and operations :-
Javascript functions capture user input and perform calculations.
Example function to handle button clicks –

```
function appendNumber(num) {
    document.getElementById("display").value += num;
}
```

(b) Performing calculations –
Javascript executes calculation when user clicks the "=" button.

```
function calculateResult() {
    let expression = document.getElementById("display").value
    document.getElementById("display").value = eval(expression)
}
```

For Educational Use Only

eval() is used for simplicity but should be replaced with a safer alternative in production code.

(c) Input validation - It ensures users enter correct values and prevents invalid operations like division by zero.

```
function validateInput(value) {
    if( isNaN(value) || value === "") {
        alert("Invalid input!");
        return false;
    }
    return true;
}
```

(d). Using prompts and alerts for User Interaction.

- alert() - Displays error messages when invalid input is detected.
- prompt() - Accepts user input dynamically eg. for squaring a number.

```
function squareNumber() {
    let num = prompt("Enter a number to square:");
    if(validateInput(num)) {
        alert("The square is: " + (num * num));
    }
}
```

Flow is as follows -

For Educational Use Only

IV) Application flow -

(1). User enters numbers and operators using calculator buttons.
(2) Javascript captures the input and updates the display field.
(3) The equals (=) button triggers the calculation.
(4) Javascript validates the input to prevent errors
(5) Results are displayed in the text field.
(6). Alerts are shown if the user enters invalid values.
(7). The clear button resets the calculator for new calculations.

- **Conclusion :** This JS calculator demonstrates how HTML, CSS and JS work together to create an interactive web application.
  - HTML provides the structure.
  - CSS enhances design.
  - JS handles functionality.

This helped us understand DOM manipulation, event handling, user input validation and interactive UI design. Thus we successfully created a JS application.

* * * * * * *

10/2/25

For Educational Use Only

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Practical 4

Implement an application in Java Script using following:

a) Design UI of application using HTML, CSS etc.

b) Include Java script validation

c) Use of prompt and alert window using Java Script

Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scientific Calculator</title>
    <style>
        .calculator {
    width: 450px;
    background: navy;
    padding: 30px;
    border-radius: 20px;
    text-align: center;
    box-shadow: 20px 20px 20px lightskyblue;
}
#display {
    width: 100%;
```

```
    height: 50px;
    font-size: 1.5em;
    text-align: right;
    background: #1a2a42;
    color: white;
    padding: 10px;
    border: none;
    border-radius: 10px;
}
.buttons {
    display: grid;
    grid-template-columns: repeat(6, 1fr);
    gap: 10px;
    margin-top: 20px;
}
.btn {
    padding: 15px;
    font-size: 25px;
    border: none;
    background: #2b4a6f;
    color: white;
    cursor: pointer;
    border-radius: 10px;
    transition: 0.2s;
}
```

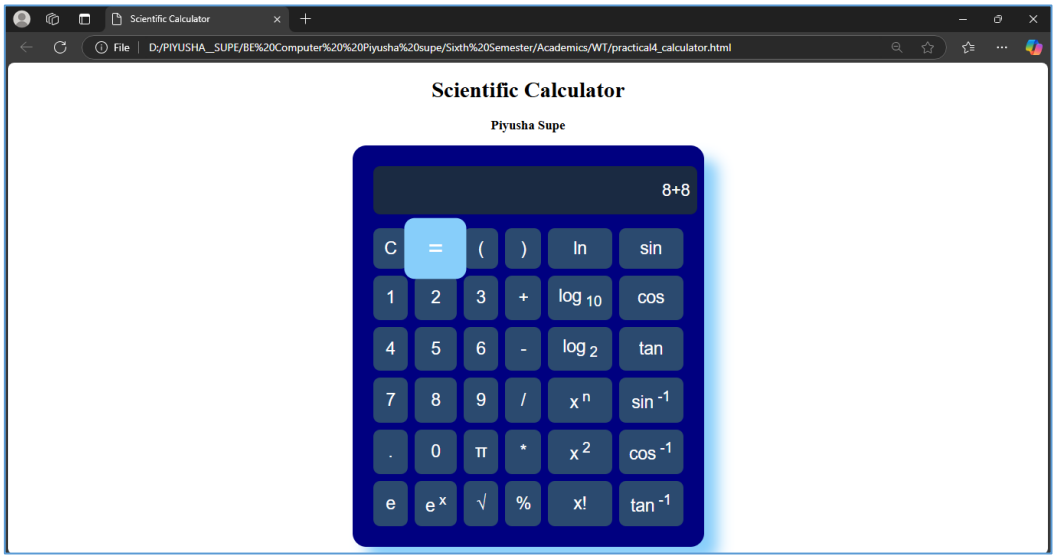
```
.btn:hover {
    background: lightskyblue;
    transform: scale(1.5);
}
    </style>
</head>
<body><center><h1>Scientific Calculator</h1>
    <h3>Piyusha Supe</h3>
    <div class="calculator">
        <input type="text" id="display" disabled>
        <div class="buttons">
            <button class="btn" onclick="clearDisplay()">C</button>
            <button class="btn" onclick="calculate()">=</button>
            <button class="btn" onclick="append('(')">(</button>
            <button class="btn" onclick="append(')')">)</button>
            <button class="btn" onclick="append('Math.log(')">ln</button>
            <button class="btn" onclick="append('Math.sin(toRadians(')">sin</button>
            <button class="btn" onclick="append('1')">1</button>
            <button class="btn" onclick="append('2')">2</button>
            <button class="btn" onclick="append('3')">3</button>
            <button class="btn" onclick="append('+')">+</button>
            <button class="btn"
onclick="append('Math.log10(')">log<sub> 10</sub></button>
            <button class="btn" onclick="append('Math.cos(toRadians(')">cos</button>
            <button class="btn" onclick="append('4')">4</button>
            <button class="btn" onclick="append('5')">5</button>
```

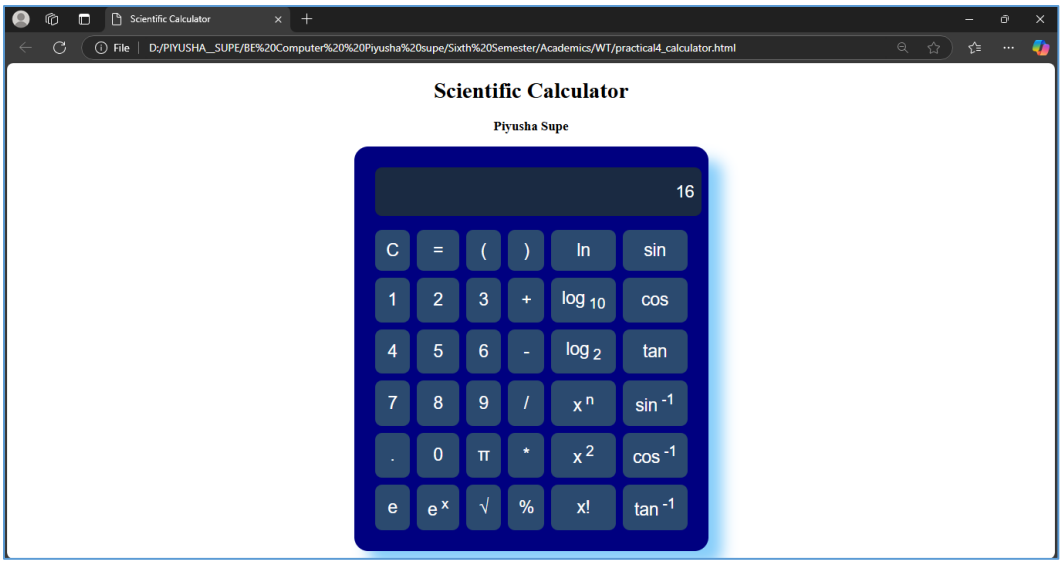
```
        <button class="btn" onclick="append('6')">6</button>

        <button class="btn" onclick="append('-')">-</button>

        <button class="btn"
onclick="append('Math.log2(')">log<sub> 2</sub></button>

        <button class="btn" onclick="append('Math.tan(toRadians(')">tan</button>

        <button class="btn" onclick="append('7')">7</button>

        <button class="btn" onclick="append('8')">8</button>

        <button class="btn" onclick="append('9')">9</button>

        <button class="btn" onclick="append('/')">/</button>

        <button class="btn" onclick="append('**')">x<sup> n</sup></button>

        <button class="btn" onclick="append('Math.asin(')">sin<sup> -
1</sup></button>

        <button class="btn" onclick="append('.')">.</button>

        <button class="btn" onclick="append('0')">0</button>

        <button class="btn" onclick="append('Math.PI')">π</button>

        <button class="btn" onclick="append('*')">*</button>

        <button class="btn" onclick="append('**2')">x<sup> 2</sup></button>

        <button class="btn" onclick="append('Math.acos(')">cos<sup> -
1</sup></button>

        <button class="btn" onclick="append('Math.E')">e</button>

        <button class="btn" onclick="append('Math.exp(')">e<sup> x</sup></button>

        <button class="btn" onclick="append('Math.sqrt(')">√</button>

        <button class="btn" onclick="append('%')">%</button>

        <button class="btn" onclick="append('factorial(')">x!</button>

        <button class="btn" onclick="append('Math.atan(')">tan<sup> -
1</sup></button>

    </div>
```

```
    </div>
    <script>
        function append(value) {
            document.getElementById("display").value += value;
        }
        function clearDisplay() {
            document.getElementById("display").value = "";
        }
        function calculate() {
            try {
                let expression = document.getElementById("display").value;
                let result = eval(expression.replace(/factorial\((\d+)\)/g, (_, num) =>
factorial(parseInt(num))));
                document.getElementById("display").value = result;
            } catch (error) {
                alert("Error! Maybe you missed a parentheses!");
            }
        }
        function toRadians(degrees) {
            return degrees * (Math.PI / 180);
        }
        function factorial(n) {
            if (n === 0 || n === 1) return 1;
            let result = 1;
            for (let i = 2; i <= n; i++) {
                result *= i;
```

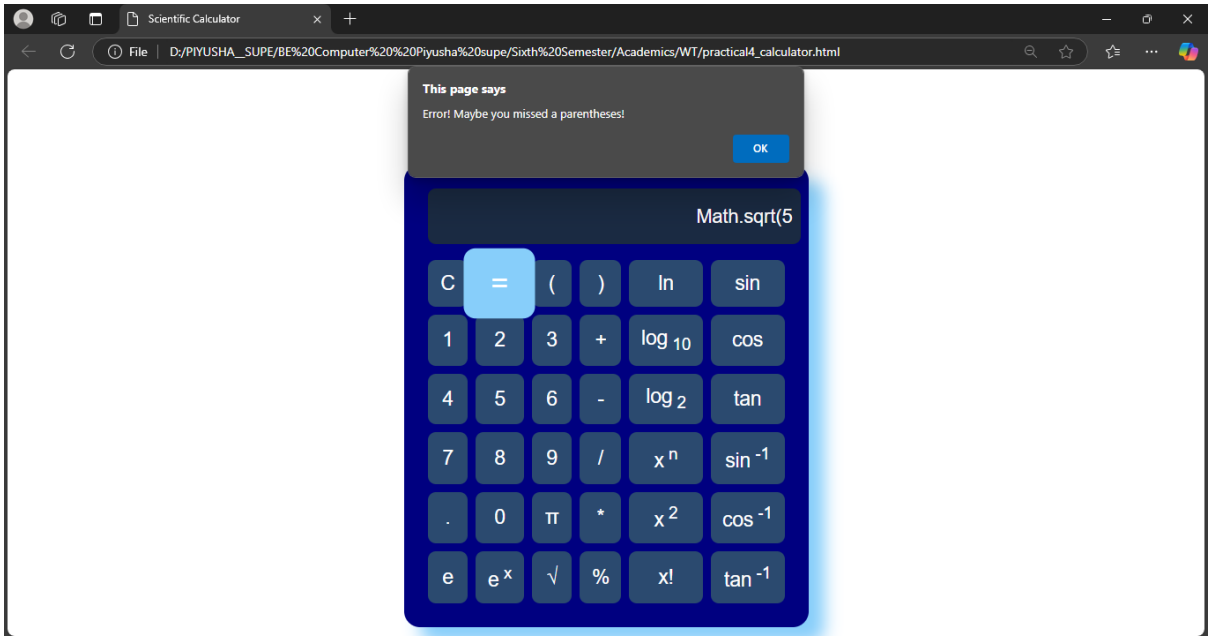
```
        }
        return result;
    }
  </script>
</center>
</body>
</html>
```

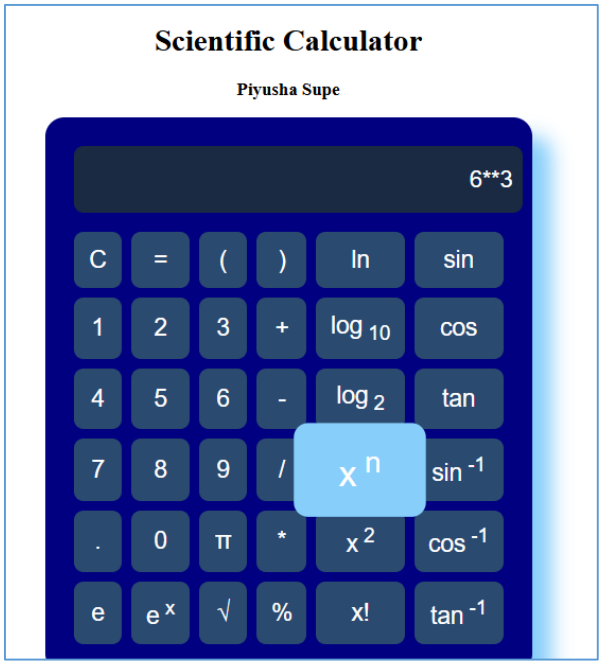
OUTPUT:

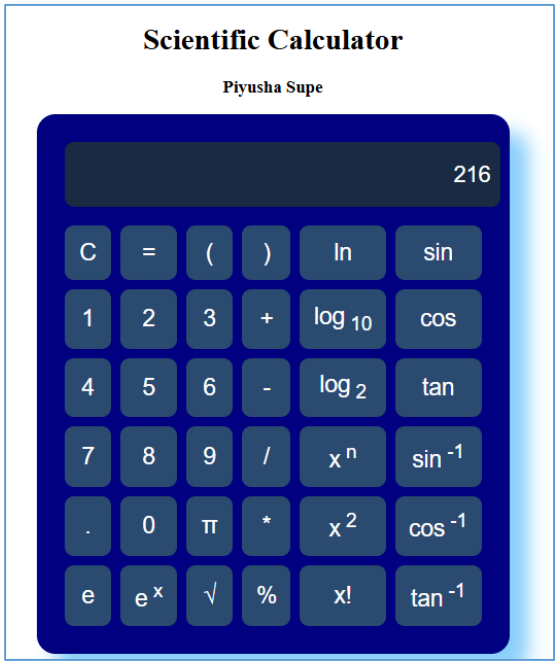




Error messages displayed using alert()







Experiment - 05.

- **Aim :** Implement the sample program demonstrating the use of servlet.

eg: Create a database table cbookshop (book-id, book-title, book-author, book-price, quantity) using database like Oracle / Mysql etc. and display (use sql select query) the table content using servelet.

- **Theory :** SERVELET.

I. Introduction to servelets.

- Servelets are java programs that run on a web server and handle HTTP requests and responses. They are an essential part of Java EE (Enterprise edition) and are used to build dynamic web applications.

- Servelets help in interacting with databases, processing user requests and dynamically generating web content.

II. Servelet Architecture -

- A servelet operates within a Java EE server, handling client requests (usually from a web browser) and responding with dynamically generated content.

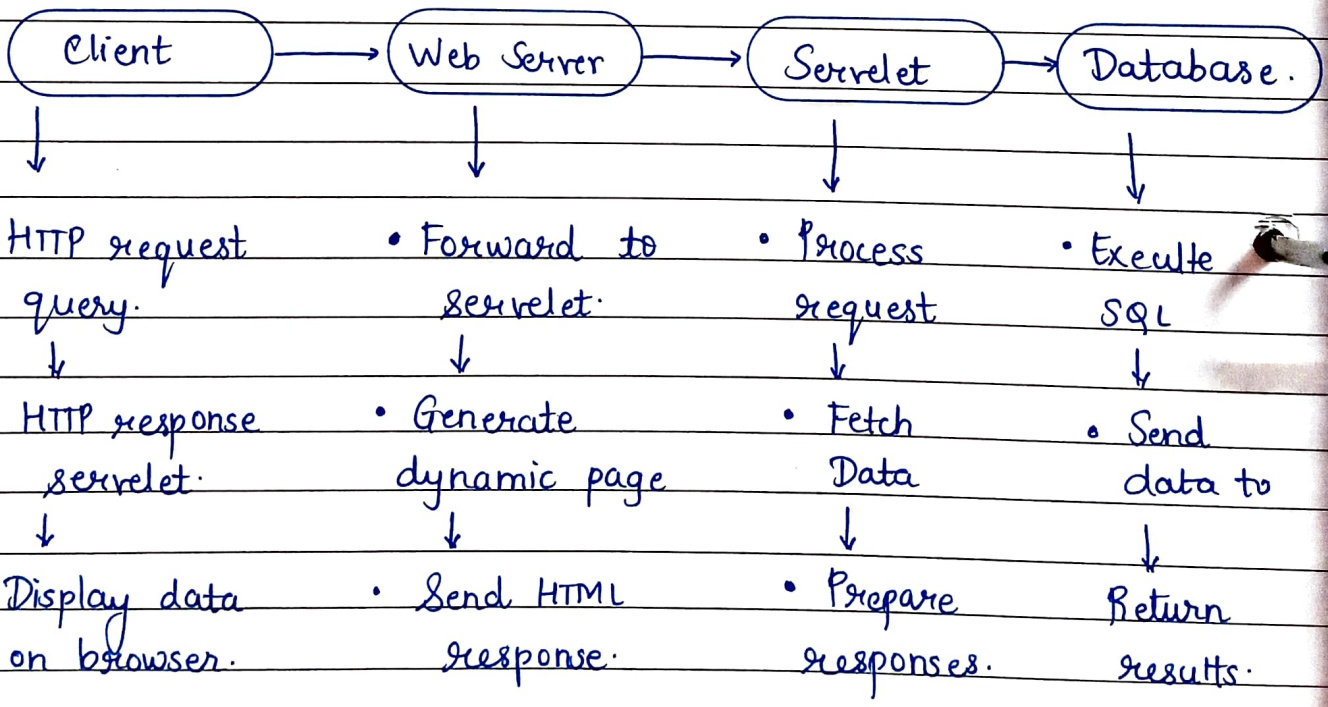
For Educational Use Only

Page No. :
Date. : / /

- **Basic Servelet Flow -**

1. Client sends an HTTP request (eg. clicking a button on a web page)
2. Web server receives the request and passes it to the appropriate Servelet.
3. Servelet processes the request, interacts with a database (if required) and prepares response.
4. Web server sends the response back to the client.

- **Block diagram of servelet processing -**



For Educational Use Only

III] Steps to implement Servelet for displaying database content.

STEP 1 : Set up the database (MySQL/ Oracle).

1. Choose database System.
   - Install mysql or Oracle and configure it.
   - Start the database server and ensure it is running.

2. Create database and table: ebookshop.
   eg - book-id (Primary-key, INT)
   book-title (VARCHAR)
   book-author (VARCHAR)
   book-price (FLOAT)
   book-quantity (INT)

3. Insert Sample data.

STEP 2 : Configure the java EE Environment.

1. Install JDK (Java development kit).
   - Download and install JDK 8 or later.
   - Set up the JAVA-HOME environment variable.

2. Install Tomcat Server.
   - Download and configure Apache Tomcat (or any Java EE Server)
   - Deploy applications inside the webapps folder.

For Educational Use Only

26.

Page No. :
Date. : / /

STEP 3 : Create a Java Servlet.

- Connect to database using JDBC.
- Execute and SQL select query to retrieve the book details.
- Display the results in an HTML table.

1. **Import required packages.**
   - java.io.* for handling I/O operations.
   - javax.servlet and http. for servlet.
   - java.sql.* for database operations.

2. **Extend HttpServelet Class.**
   Override doGet() method to process the HTTP GET request.

3. **Load JDBC driver** – Use Class.forName() to register the MySQL / Oracle driver.

4. **Estabilish Database connection** –
   - Use DriverManager.getConnection() with the correct database URL, username and password.

5. **Execute SQL Query** –
   Use a statement or prepared Statement. to retriere book records.

6. **Generate HTML outputs** –
   - Dynamically generate an HTML page with the book details in <HTML> table.

For Educational Use Only

STEP 4 : Configure web.xml deployment descriptor.

Every servelet application requires deployment configuration.

1. Define the servelet in web.xml.
   - Map the servelet to a specific URL pattern
2. Place web.xml inside WEB-INF folder.

STEP 5: Deploy and run the servelet:

1. Start tomcat server
   - Run startup.bat (Windows) or startup.sh (linux)
2. Deploy the servelet application
   - Copy the WAR file or project folder to webapps.
3. Access the servelet via web browser -
   Open a browser and navigate to.

   http://localhost:8080/YourProjectName/displayBooks.

IV] Handling errors and security Best practices -

- While implementing servelets, it is essential to handle errors properly and follow security guidelines.

Error Handling in servelets -

1. SQL Exceptions - Use try-catch blocks when executing queries.

For Educational Use Only

Page No. :
Date. : / /

2. Servelet Exceptions - Override doPost() or doGet() methods correctly.

3. NullPointerException - Check if request parameters are null before processing.

* Security Best Practices -

- Use prepared statements instead of Statement to prevent SQL injection.
- Use HTTPs instead of HTTP for secure data transmission.
- Use data validation to prevent malicious inputs.

- **Conclusion:** Thus we successfully, implemented the java servelet application for books.

* * * * * * * *.

For Educational Use Only

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

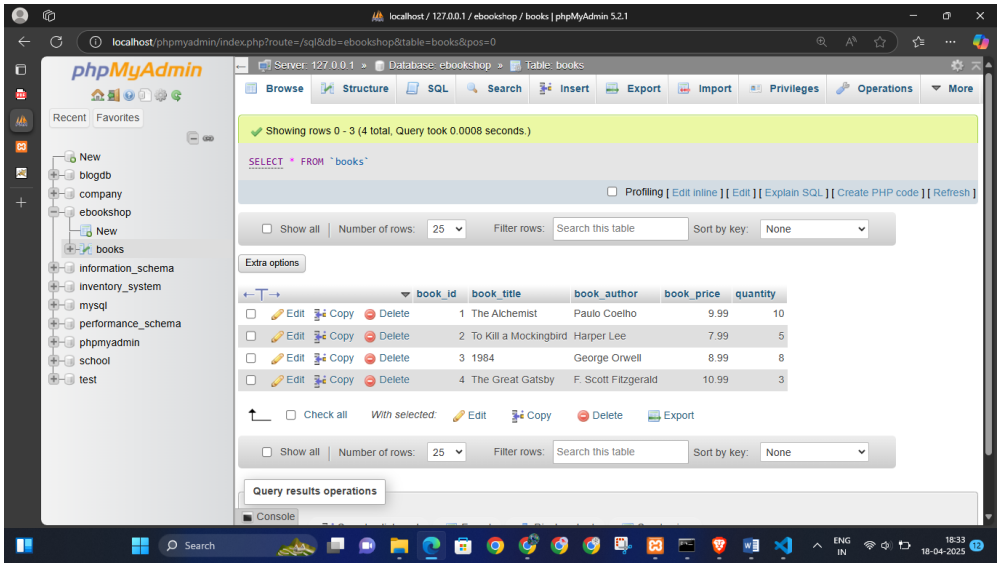
Web Technology Practical 5

Title: Implement the sample program demonstrating the use of Servlet. e.g., Create a database table ebookshop (book_id, book_title, book_author, book_price, quantity) using database like Oracle/MySQL etc. and display (use SQL select query) the table content using servlet

Ebookshop.sql

```
CREATE DATABASE IF NOT EXISTS ebookshop;

USE ebookshop;

CREATE TABLE IF NOT EXISTS books (
  book_id INT AUTO_INCREMENT PRIMARY KEY,
  book_title VARCHAR(255) NOT NULL,
  book_author VARCHAR(255) NOT NULL,
  book_price DECIMAL(10,2) NOT NULL,
  quantity INT NOT NULL
);

INSERT INTO books (book_title, book_author, book_price, quantity) VALUES
('The Alchemist', 'Paulo Coelho', 9.99, 10),
('To Kill a Mockingbird', 'Harper Lee', 7.99, 5),
('1984', 'George Orwell', 8.99, 8),
('The Great Gatsby', 'F. Scott Fitzgerald', 10.99, 3);
```

The database is as follows:



The code is as follows:

EbookShopServlet.java

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

public class EbookshopServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>Ebookshop</title></head><body>");
        out.println("<h1>Book List</h1>");
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/ebookshop", "root", "");
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM books");
            out.println("<table border='1'>");
            out.println("<tr><th>ID</th><th>Title</th><th>Author</th><th>Price</th><th>Quantity</th></tr>")
;
            while (rs.next()) {
                out.println("<tr><td>" + rs.getInt("book_id") + "</td>");
                out.println("<td>" + rs.getString("book_title") + "</td>");
                out.println("<td>" + rs.getString("book_author") + "</td>");
                out.println("<td>" + rs.getDouble("book_price") + "</td>");
                out.println("<td>" + rs.getInt("quantity") + "</td></tr>");
            }
            out.println("</table>");
            rs.close();
            stmt.close();
```

```
            conn.close();
        } catch (Exception e) {
            out.println("<p>Error: " + e.getMessage() + "</p>");
        }
        out.println("</body></html>");
    }
}
```

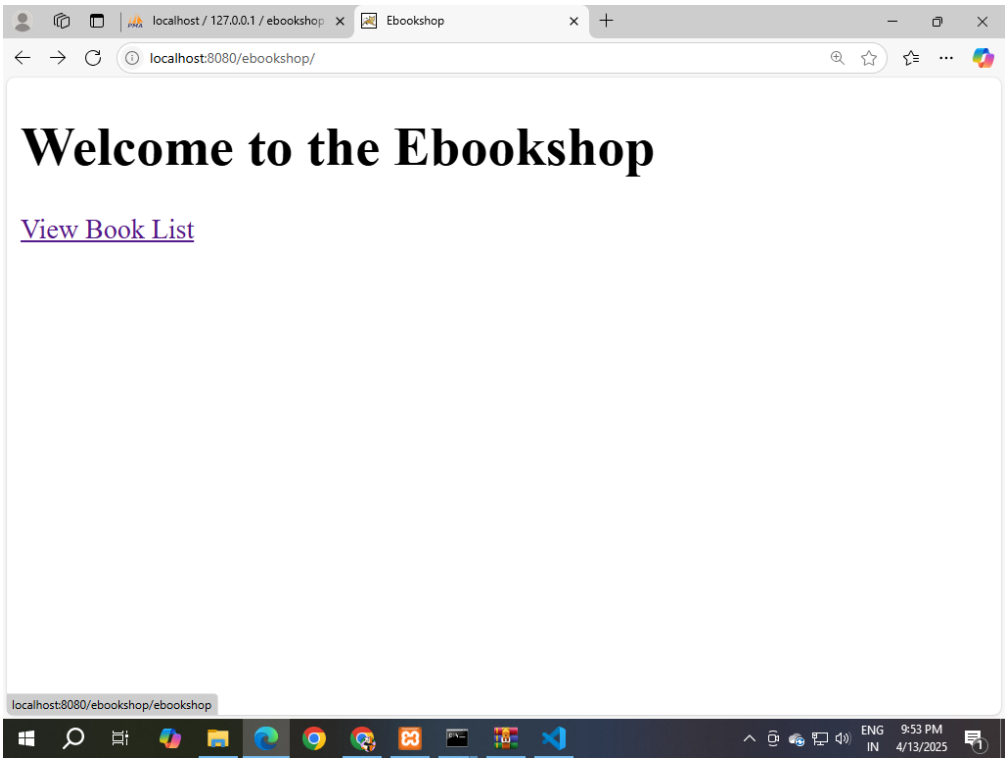
Index.html

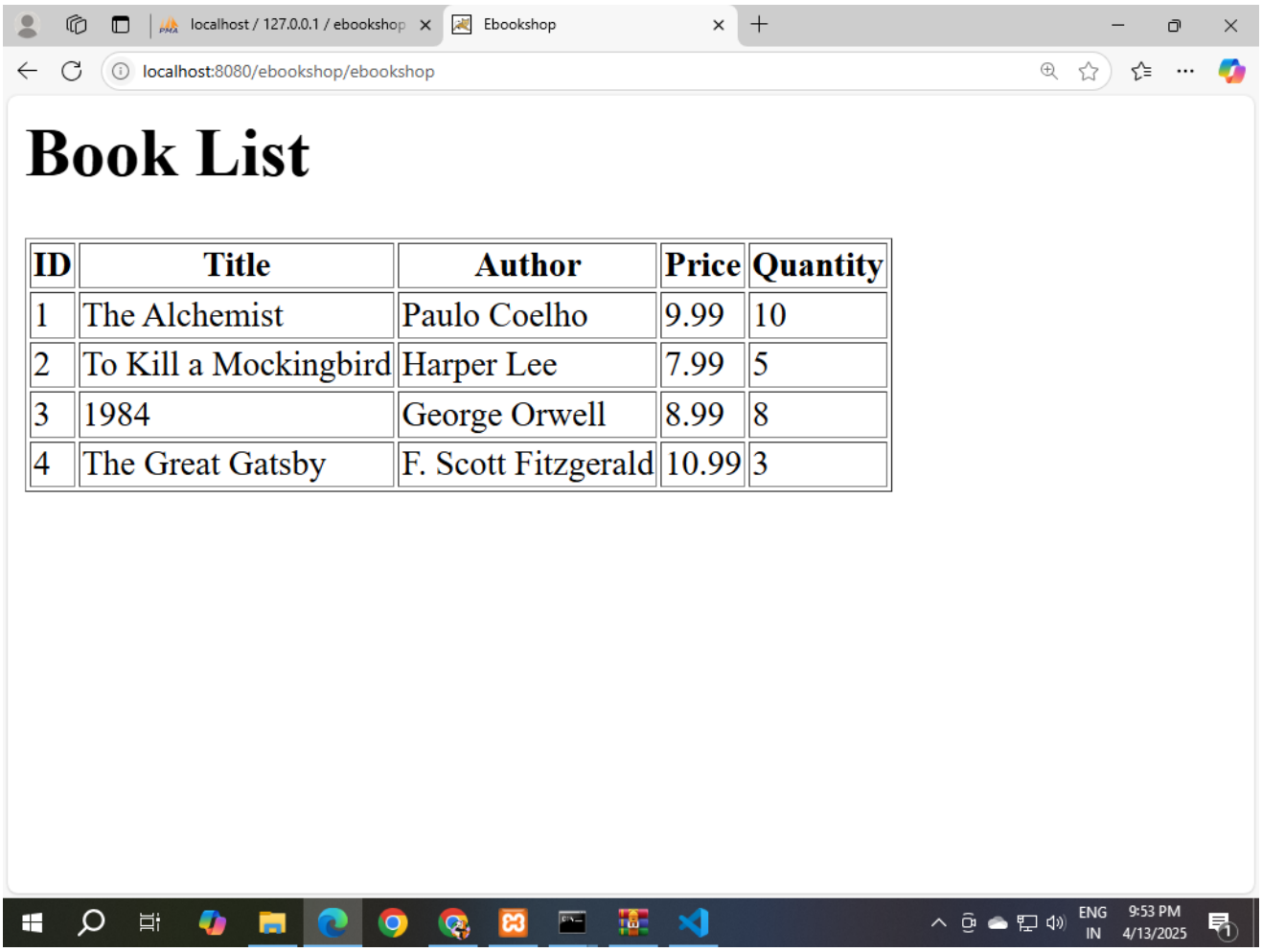
```
<!DOCTYPE html>
<html>
<head>
    <title>Ebookshop</title>
</head>
<body>
    <h1>Welcome to the Ebookshop</h1>
    <p><a href="/ebookshop">View Book List</a></p>
</body>
</html>
```

Web.xml (used for mapping the servlet class)

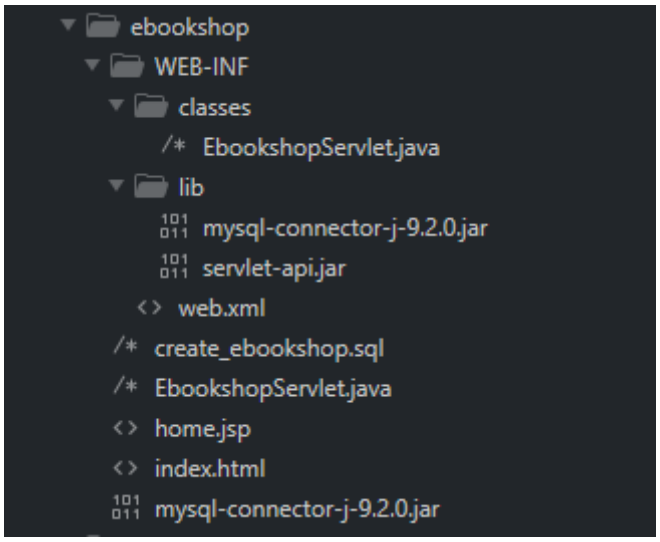
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         version="3.1">
    <servlet>
        <servlet-name>EbookshopServlet</servlet-name>
        <servlet-class>EbookshopServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>EbookshopServlet</servlet-name>
        <url-pattern>/ebookshop</url-pattern>
    </servlet-mapping>
</web-app>
```

The output is as follows:

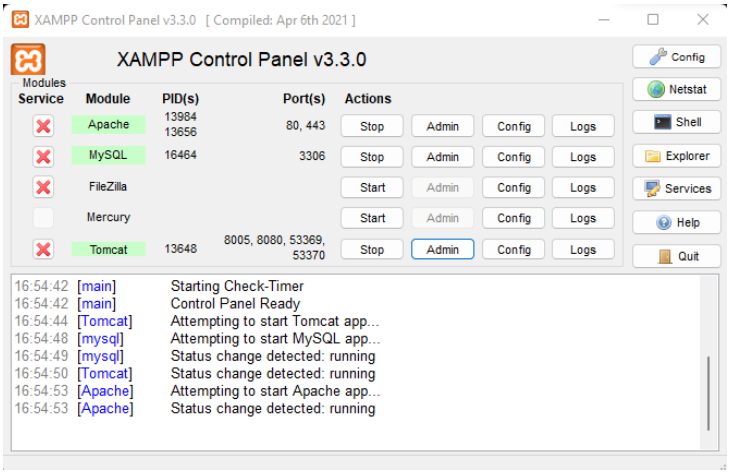


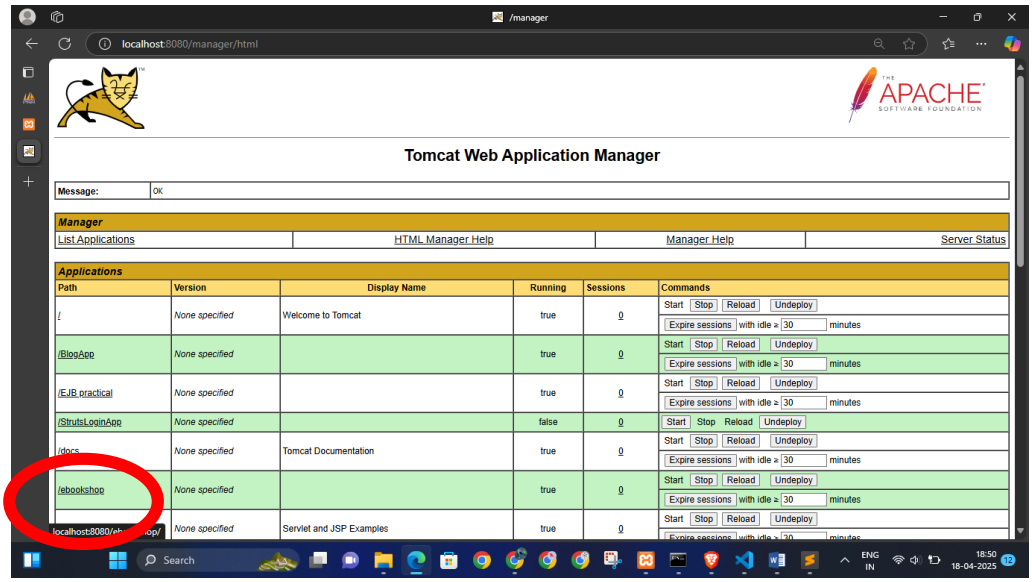


The directory structure is as follows:



The servlet is run using the XAMPP server TOMCAT server available inside it:





Conclusion: Thus in this way I have successfully created a servlet and retrieved information through it

Experiment - 06.

- **Aim:** Implement the program demonstrating the use of JSP.
  eg: Create a database table students-info (stud-id, stud-name, class, division, city) using database like oracle / mysql, etc. and display (use SQL select query) the table content using JSP.

- **Theory:** Introduction to JSP.

1) • JavaServer Pages (JSP) is a server-side technology that creates dynamic web applications.
• It allows developers to embed java code directly into HTML or XML pages and it makes web development more efficient.

• JSP is an advanced version of servelets. It provides enhanced capabilities for building scalable and platform independent web pages.

2) * How is JSP more advantageous than Servelets?

• JSP simplifies web development by combining the strengths of java with the flexibility of HTML.

* Some of the advantages of JSP over servelets are listed below -

For Educational Use Only

Page No. : 30.
Date. : / /

1. JSP code is easier to manage than servelets as it separates UI and business logic.
2. JSP minimizes the amount of code required for web applications.
3. Easily generates content dynamically in response to user interactions.
4. It provides access to the complete range of Java APIs for robust application development.
5. JSP is suitable for applications with growing user bases.

3] Key features of JSP -

- Platform Independence - Write once, run anywhere.
- It simplifies database interactions for dynamic content.
- It contains predefined objects like request, response, session and application reduce development time.
- It has built-in mechanisms for exception and error management.
- It supports custom tags and tag libraries.

4] JSP Architecture -

JSP follows a three layer architecture :

For Educational Use Only

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

1. Client layer – The browser sends a request to the server.

2. Web server layer – The server processes the request using a JSP engine.

3. Database / Backend layer – Interacts with the database and returns the response to the client.

5] Steps to create a JSP application –

1. Take any HTML file you have previously created.
2. Change the file extension from .html to .jsp.
3. Load the new .jsp file in browser.

When we load a JSP file for the first time.

- The JSP is converted into a java file.
- The Java file is compiled into a servelet.
- The compiled servelet is loaded and executed.

6] Adding dynamic content with JSP –

Example –

```
<html>
    <body>
        Hello! The time is now <%= new
        java.util.Date() %>
    </body>
</html>
```

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- Explanation -
- The <%= %> tags enclose a java expression.
- The new java.util.Date() expression retrieves the current date and time.
- When the JSP page is loaded in the browser, the java expression is evaluated at runtime, and the output is embedded into the HTML.
- Each time you reload the page, it displays the current time, demonstrating how JSP dynamically generates HTML content based on JAVA logic.

7] JSP Elements -

(a). Expression - This tag is used to output any data on the generated page. These data are automatically converted to a string and printed on output stream.

Syntax - <%= "Anything" %>

(b). Scriplets -
This allows inserting any amount of valid java code. These codes are placed in the _jspService() method by the JSP engine.

Syntax → <% // Java codes %>
Eg → <%
String name = "Hello";
out.println ("Variable " + name);
%>

Variables available to the JSP Scriptlets are -

- Request
- Response
- Session
- Out

(c) Directives - A jsp directive starts with <%@. In it we can import packages, define error-handling pages or configure session information for JSP page.

Syntax → <%@ directive attribute = "value" %>

Types of directives -

1. pages = It defines page settings.
2. include = It includes other files.
3. taglib = It declares a custom tag library.

(d). Declarations : This is used for defining functions and variables to be used in the JSP.
Variables and functions defined in the declaratives are class level and can be used anywhere on the JSP page.

Eg -
```
<%@ page import = "java.util.*" %>
<html>
    <body>
        <%!  Date theDate = new Date();
             Date getDate (){
                 System.out.println ("getDate()");
                 return theDate;
             }
        %>
```

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

```
Hello! The time is now <%= getDate() %>
</body>
</html>
```

8] Running a simple JSP page -

Steps to run JSP -

1. Save the JSP file using the .jsp extension (eg. hello.jsp
2. Start the server.
3. Place your application inside appropriate folder (eg. webapps for tomcat)
4. Open the browser and enter the JSP page URL.

http://localhost:port/Yourapplication/jspfile.

The JSP is compiled and executed.

9] Why use JSP?

- Embed java logic directly to HTML.
- To create dynamic pages that respond to user actions.
- TO customize content for each user or session.

* **Conclusion:** Thus we successfully used database with java server pages or JSP.

* * * * * * *

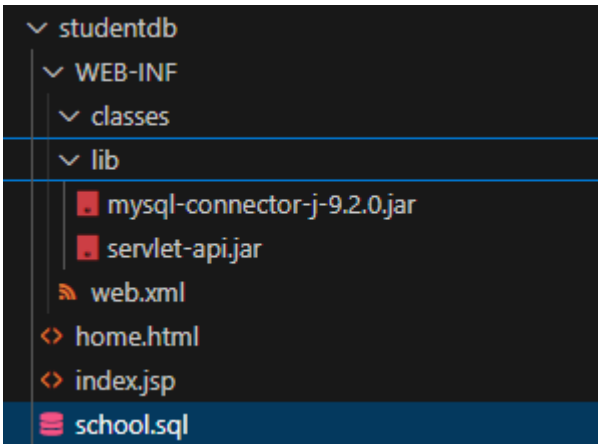
Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Practical 6

Title: Implement the program demonstrating the use of JSP. e.g., Create database table students_info (stud_id, stud_name, class, division, city) using database like Oracle/MySQL etc. and display (use SQL select query) the table content using JSP

The folder structure is as follows:



Let us create a database for the JSP retrieval:

School.sql

```
Create database if not exists school;
CREATE TABLE `students_info` (
  `stud_id` int(11) NOT NULL,
  `stud_name` varchar(100) DEFAULT NULL,
  `class` varchar(20) DEFAULT NULL,
  `division` varchar(5) DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL
);
INSERT INTO `students_info` (`stud_id`, `stud_name`, `class`, `division`, `city`) VALUES
(1, 'Piyusha', '10', 'A', 'New York'),
(2, 'Chinmay', '9', 'B', 'Los Angeles'),
(3, 'Rajendra', '10', 'A', 'Chicago'),
(4, 'Diana Prince', '8', 'C', 'Houston'),
(5, 'Ethan Hunt', '11', 'A', 'Phoenix'),
(6, 'Fiona Glenanne', '10', 'B', 'Philadelphia'),
```

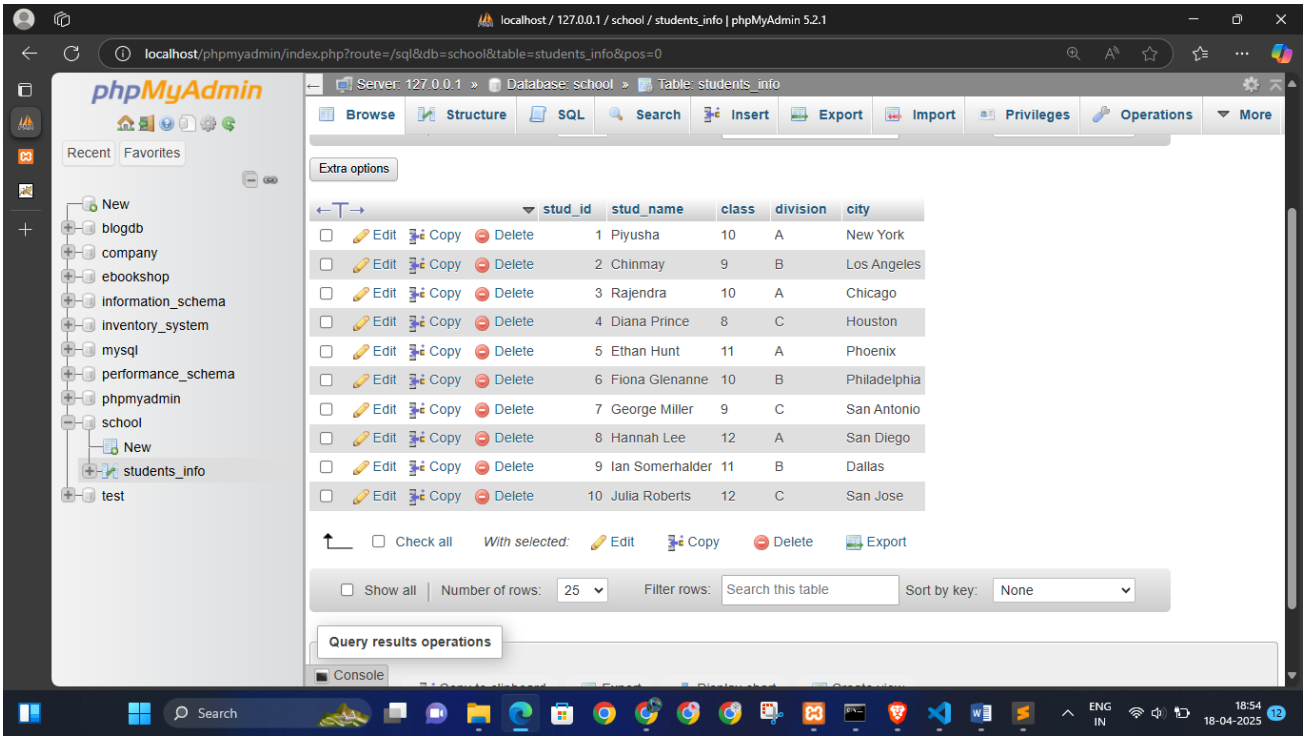
(7, 'George Miller', '9', 'C', 'San Antonio'),

(8, 'Hannah Lee', '12', 'A', 'San Diego'),

(9, 'Ian Somerhalder', '11', 'B', 'Dallas'),

(10, 'Julia Roberts', '12', 'C', 'San Jose');

The database created will look as follows:



The code is as follows:

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">
    <!-- Welcome file list to define the landing page -->
    <welcome-file-list>
        <welcome-file>home.html</welcome-file> <!-- This is the first page that loads -->
    </welcome-file-list>
    <!-- Optional: If you want to add specific URL pattern mappings for your JSP -->
    <servlet>
        <servlet-name>indexServlet</servlet-name>
        <jsp-file>/index.jsp</jsp-file>
    </servlet>
```

```
    <servlet-mapping>
        <servlet-name>indexServlet</servlet-name>
        <url-pattern>/index.jsp</url-pattern>
    </servlet-mapping>
</web-app>
```

Home.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Information Portal</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background: #f4f4f9;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            height: 100vh;
            margin: 0;
        }
        h1 {
            color: #333;
        }
        a.button {
            display: inline-block;
            padding: 12px 25px;
            font-size: 16px;
            color: white;
            background-color: #007BFF;
            text-decoration: none;
            border-radius: 6px;
```

```
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
            transition: background-color 0.3s ease;
        }
        a.button:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <h1>Welcome to the Piyusha's Student Info Portal</h1>
    <a class="button" href="index.jsp">View Student Records</a>
</body>
</html>
```

Index.jsp (actual jsp code that retrieves information form database)

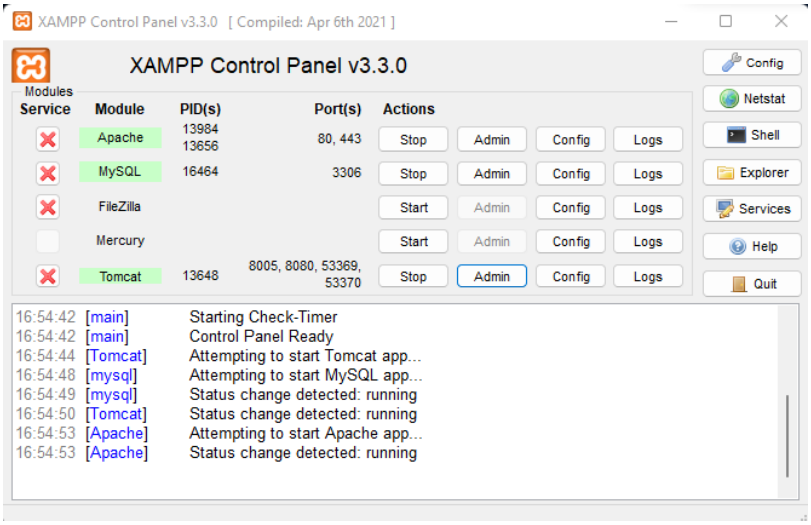
```
<%@ page import="java.sql.*" %>
<html>
<head>
    <title>Student Info</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 30px;
            background-color: #f4f4f9;
        }
        h1 {
            color: #333;
            text-align: center;
        }
        table {
            width: 80%;
            margin: auto;
            border-collapse: collapse;
```

```
            box-shadow: 0 2px 8px rgba(0,0,0,0.1);
            background-color: white;
        }
        th, td {
            padding: 12px 15px;
            text-align: center;
            border: 1px solid #ccc;
        }
        th {
            background-color: #007BFF;
            color: white;
        }
        tr:nth-child(even) {
            background-color: #f9f9f9;
        }
        tr:hover {
            background-color: #e9f1ff;
        }
        .error {
            color: red;
            text-align: center;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <h1>Student Information</h1>
    <table>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Class</th>
```

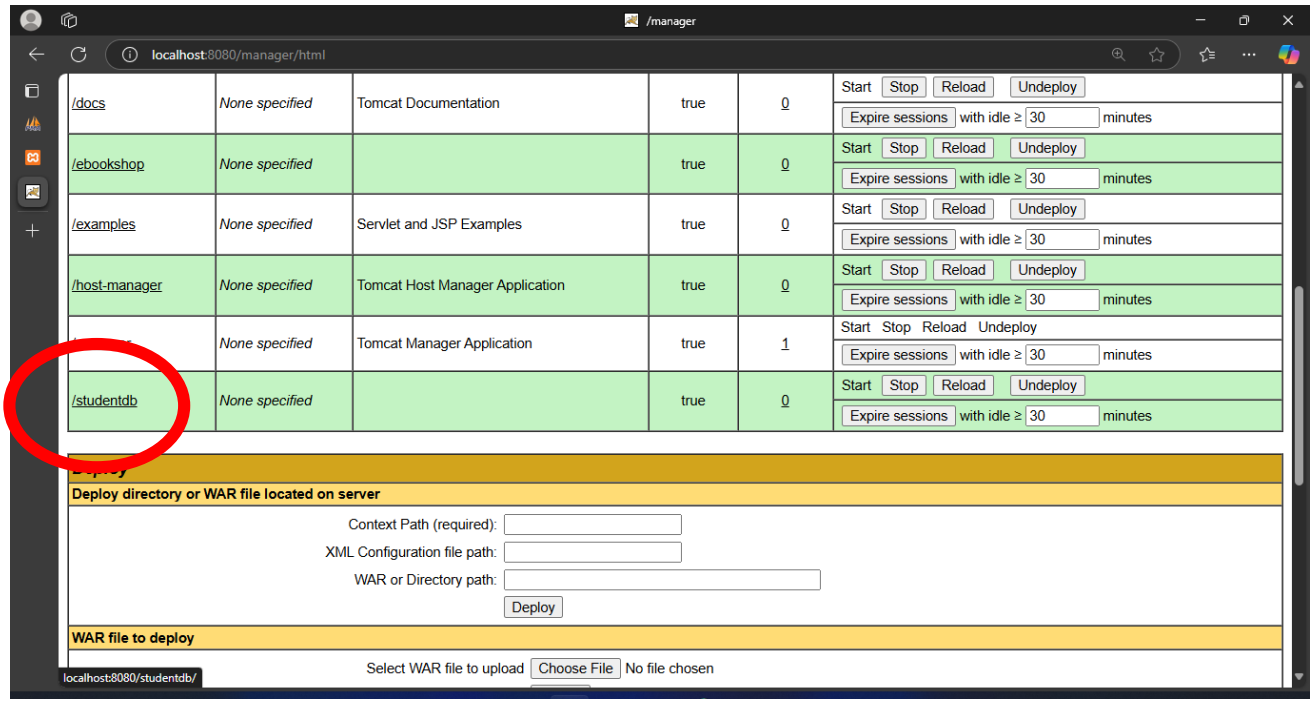
```
            <th>Division</th>
            <th>City</th>
        </tr>
        <%
            try {
                Class.forName("com.mysql.cj.jdbc.Driver");
                Connection con = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/school?useSSL=false&serverTimezone=UTC", "root", "");
                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM students_info");

                while(rs.next()) {
        %>
                    <tr>
                        <td><%= rs.getInt("stud_id") %></td>
                        <td><%= rs.getString("stud_name") %></td>
                        <td><%= rs.getString("class") %></td>
                        <td><%= rs.getString("division") %></td>
                        <td><%= rs.getString("city") %></td>
                    </tr>
        <%
                }
                con.close();
            } catch(Exception e) {
        %>
                <p class="error">Error: <%= e.getMessage() %></p>
        <%
            }
        %>
    </table>
</body>
</html>
```

The output is as follows:

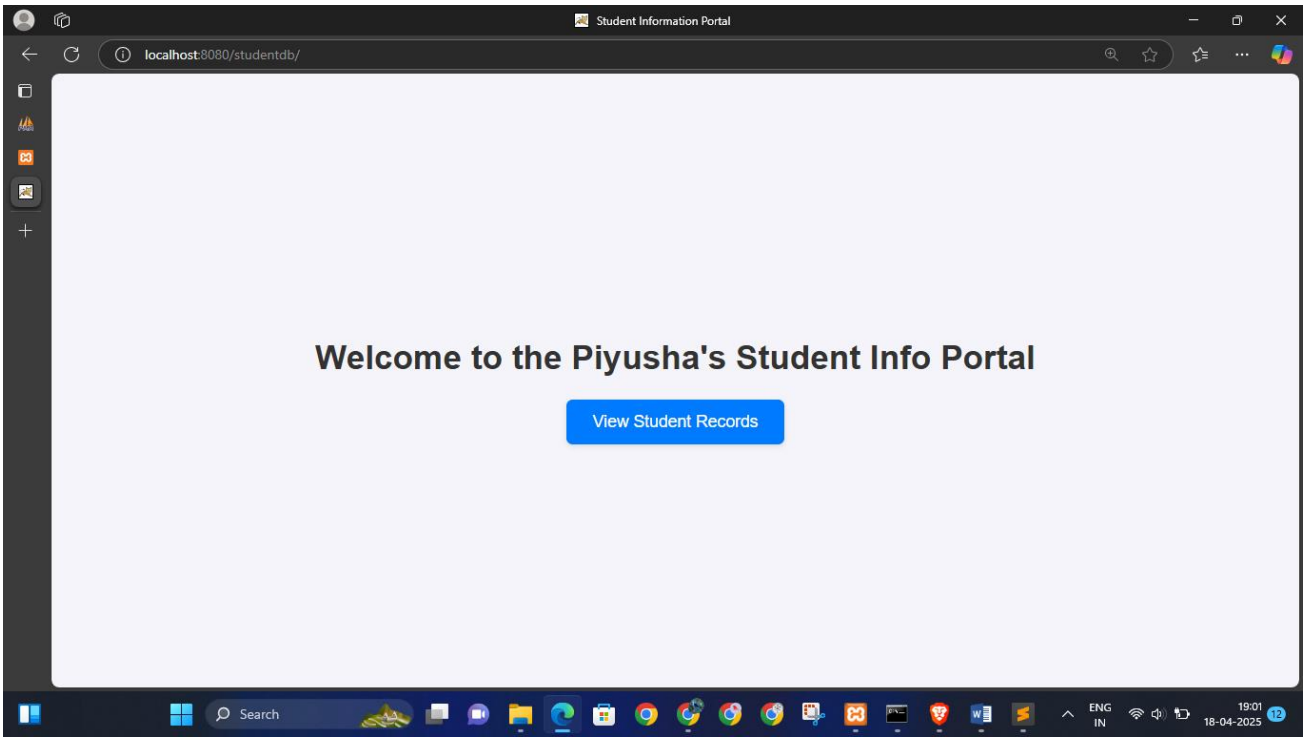
Start the tomcat server



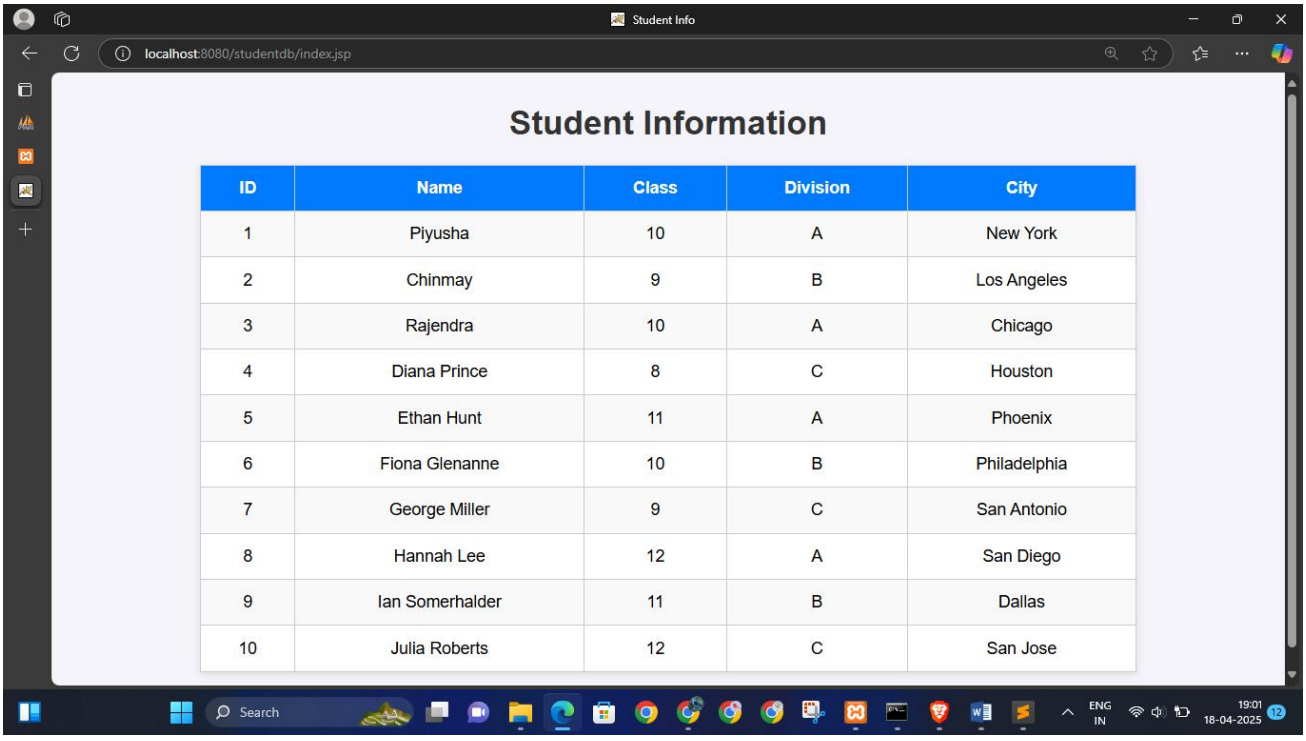
Then go to the tomcat manager admin interface:



The output will look like as follows:



The table is as follows:



Conclusion: Thus I have successfully created JSP pages and retrieved database information using mysql connector and JSP interface

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

Experiment - 07.

- **Aim:** Build a dynamic web application using PHP and mySQL.
  a. Create database tables in MySQL and create connection with PHP.
  b. Create the add, update, delete and retrieve functions in the PHP web app interacting with MySQL database.

- **Theory =>**

1) Introduction to PHP -

- PHP (Hypertext preprocessor) is a server side scripting language used for creating dynamic web applications
- It is widely used for database driven web apps due to its seamless integration with MySQL, an open source relational database management system.

- A PHP my-SQL web application enables users to add, update, delete and retriere data from a MySQL database dynamically.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

2) PHP and mySQL web application architecture.

A typical php web app follows a three-tier architecture.

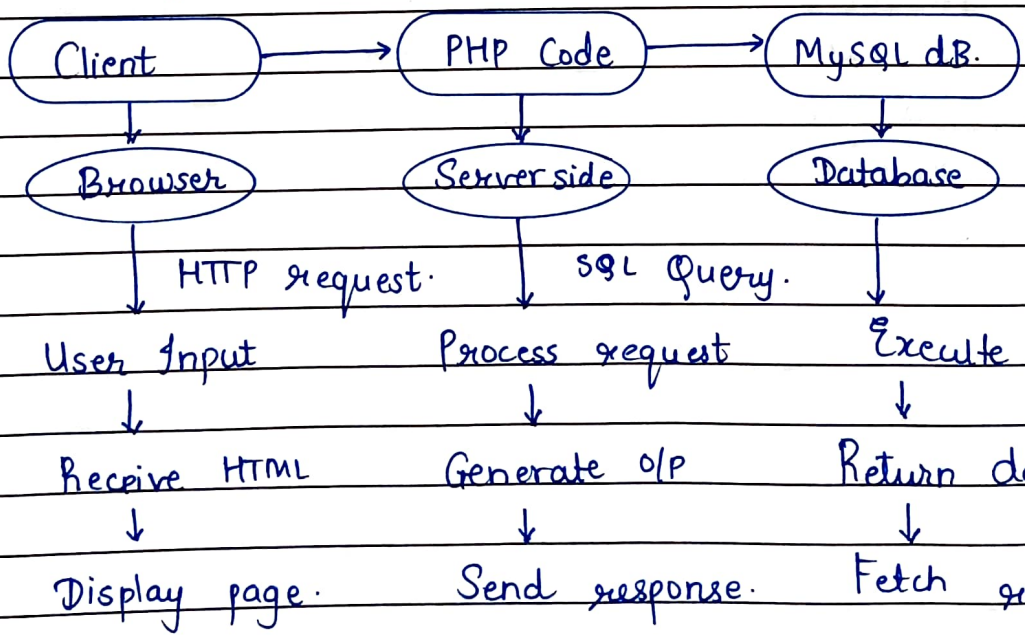
Components of PHP-MySQL web application -

1. Client side (Front end) -
   - Users interact with the application via a web browser.
   - HTML, CSS, JS are used for UI

2. Server side (PHP Backend) -
   - PHP processes requests, executes business logic and interacts with mysql.

3. Database layer (MySQL) - Stores and retrieves application data.

Block diagram -



A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

32

3] Steps to Build a dynamic PHP MySQL Web application

STEP 1: Set up MySQL Database and Tables.

1. Install mySQL database server.
   - Use XAMPP, WAMP or LAMP
   - Start MySQL service.
2. Create MySQL database.
3. Create required tables., insert sample data.

STEP 2: Estabilish connection between PHP and MySQL.

1. Enable MySQL Extension in PHP.
   - Ensure it is enabled in php.ini
2. Write PHP Script to connect to database.
   - Use mysqli_connect() or new PDO() to establish connection.
   - Store database credentials (hostname, username, password, dbname).
3. Handle Connection Errors.
   - Use die() or try-catch to manage error.

STEP 3: Implement CRUD operations (Create, read, update, delete).

3.1 Create: (Add data to database).

- Users enter details in HTML form.
- PHP processes form data and inserts using INSERT INTO Query.
- Validate user input to prevent SQL injection.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

3.2 Read (Retrieve data from database and display).

- PHP executes an SQL SELECT query to fetch.
- Retrieved data is displayed using HTML table.
- Use mysqli_fetch_assoc() or PDO::fetch() to extract data.

3.3 Update (Modify existing records)

- Use UPDATE query to modify existing records.

3.4 Delete (Remove records from database).

- Use the DELETE FROM query to remove the record.

STEP 4 : Handling forms and User input in PHP.

- To collect user input php uses the $_POST and $_GET superglobal variables.

1. HTML form -
   - Use <form method="POST"> for data submit.
   - <input type=" "> for user inputs.

2. PHP forms processing. -
   - Check if form is submitted using isset ($_POST['submit'])
   - Validate inputs (eg. empty(), filter_var()).

STEP 5: Displaying data in HTML table.

1. Use a while loop for data fetching.
   - Fetch rows using mysqli_fetch_assoc().
   - Display details inside a <table>

2. Enable sorting and filtering.
   - Use SQL order by for sorting.
   - Use search filters for refining results dynamically

STEP 6: Deploy PHP application.

- Run PHP scripts via localhost using XAMPP / WAMP.
- Check database connectivity.
- Access your website through server, localhost.

You can also implement user authentication and sessions -

* **Implement login and session management -**

1. Create user Table in MySQL.
   - Store username, password, email.

2. Implement login system.
   - Use session_start() to maintain login state.
   - Validate user credentials using select query.

3. Secure password storage.
   - Use password_hash() and password_verify() for secure authentication.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

* Security Best Practices –

1. Preventing SQL Injection.
   - Always use prepared statements instead of raw SQL queries.
   - Escape inputs using html specialchar().

2. Avoiding Cross site Scripting (XSS).
   - Sanitize user input before displaying.

3. Restricting unauthorized access.
   - Implement session management and role based authentication.
   - Restrict access to sensitive files using .htaccess.

* **Conclusion:** Thus we successfully implemented CRUD operations using PHP and mySQL.

* * * * * * *

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Practical 7

Title: Build a dynamic web application using PHP and MySQL.

a. Create database tables in MySQL and create connection with PHP.

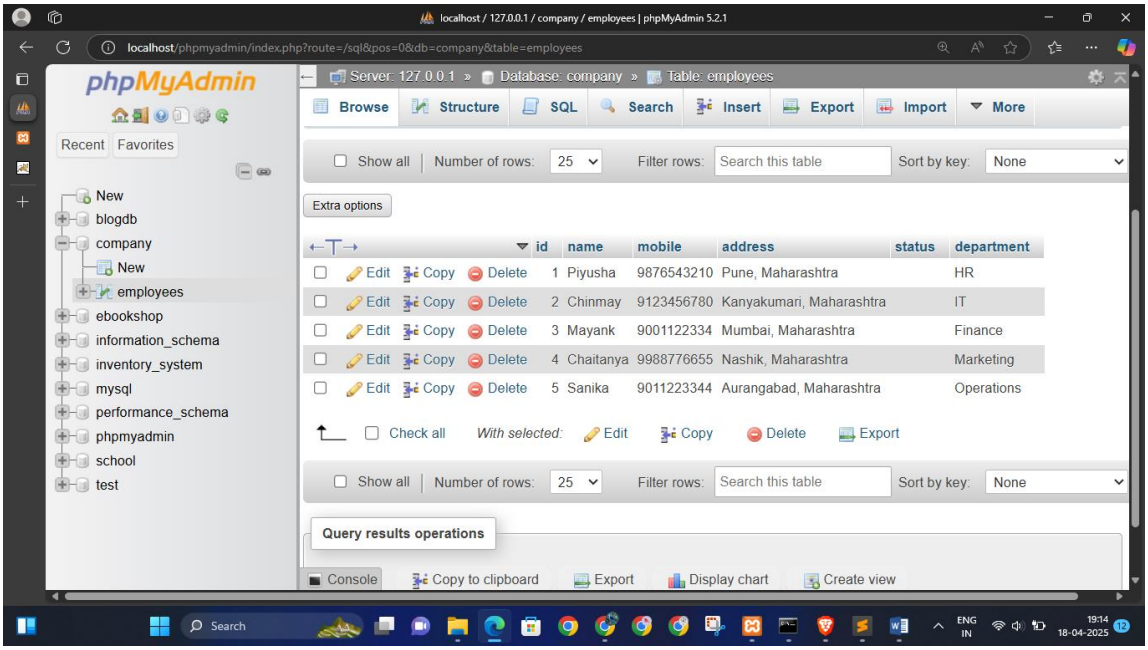
b. Create the add, update, delete and retrieve functions in the PHP web app interacting with MySQL database

Let us create a database first:

Employee_info.sql

```
CREATE DATABASE IF NOT EXISTS company;
USE company;
CREATE TABLE IF NOT EXISTS employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    mobile VARCHAR(15),
    address TEXT,
    status ENUM('Working', 'On Leave') DEFAULT 'Working',
    department VARCHAR(100)
);
```

The database structure will look like as follows:



The HTML, CSS and PHP code is combined in one file:

Employee_system.php

```
<?php
// Database connection
$conn = new mysqli("localhost", "root", "", "company");
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Insert or Update
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $name = $_POST['name'];
    $mobile = $_POST['mobile'];
    $address = $_POST['address'];
    $status = $_POST['status'];
    $department = $_POST['department'];

    if (isset($_POST['update_id']) && !empty($_POST['update_id'])) {
        $id = intval($_POST['update_id']);
        $sql = "UPDATE employees SET name='$name', mobile='$mobile', address='$address', status='$status',
            department='$department' WHERE id=$id";
    } else {

        $sql = "INSERT INTO employees (name, mobile, address, status, department)
                VALUES ('$name', '$mobile', '$address', '$status', '$department')";
    }
    $conn->query($sql);
    header("Location: employee_system.php");
    exit();
}

// Delete
if (isset($_GET['delete'])) {
```

```
    $id = $_GET['delete'];
    $conn->query("DELETE FROM employees WHERE id=$id");
    header("Location: employee_system.php");
    exit();
}

// Fetch for update
$updateData = null;
if (isset($_GET['edit'])) {
    $id = $_GET['edit'];
    $result = $conn->query("SELECT * FROM employees WHERE id=$id");
    $updateData = $result->fetch_assoc();
}
?>

<!DOCTYPE html>
<html>
<head>
    <title>Prestine Purchases - Employee Data System</title>
    <style>
        body {

            background-color: #f4f6f9;
            margin: 0;
            padding: 0;
        }
        header {
            background-color:rgb(236, 175, 221);
            color: Black;
            padding: 20px;
            text-align: center;
            font-size: 28px;
```

```
    font-weight: bold;
}
.container {
    padding: 20px;
    max-width: 1000px;
    margin: auto;
}
form {
    background-color: white;
    padding: 20px;
    border-radius: 12px;
    box-shadow: 0 0 10px #ccc;
    margin-bottom: 30px;
}
input, select, textarea {
    width: 100%;
    padding: 8px;
    margin: 8px 0 20px 0;
    border: 1px solid #ccc;
    border-radius: 6px;
}
button {
    padding: 10px 20px;
    background-color:rgb(128, 0, 85);
    color: white;
    border: none;
    border-radius: 6px;
    cursor: pointer;
}
button:hover {
    background-color:rgb(172, 133, 187);
}
```

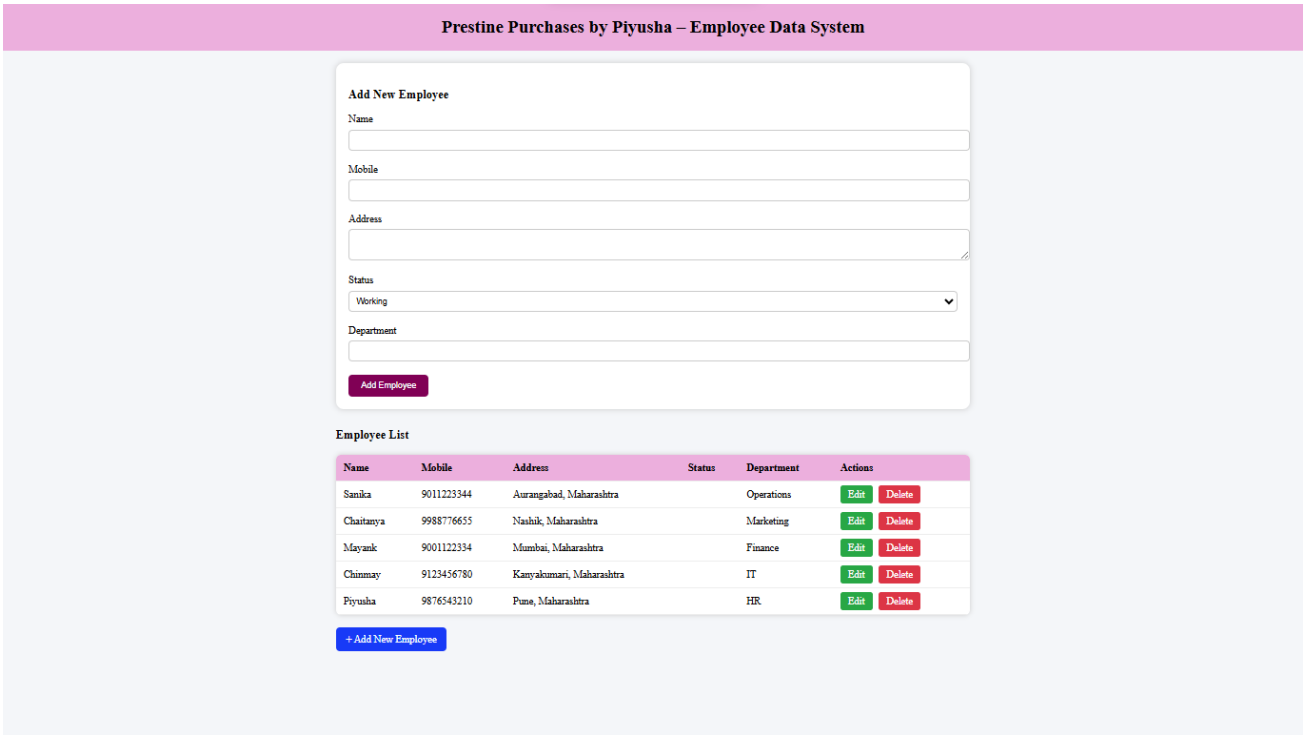
```
table {
    width: 100%;
    border-collapse: collapse;
    background: white;
    border-radius: 8px;
    overflow: hidden;
    box-shadow: 0 0 8px #ccc;
}
th, td {
    padding: 12px;
    border-bottom: 1px solid #eee;
    text-align: left;
}
th {
    background-color:rgb(236, 175, 221);
    color: black;
}
.action-buttons a {
    text-decoration: none;
    color: white;
    padding: 6px 12px;
    border-radius: 4px;
    margin-right: 6px;
}
.edit-btn {
    background-color: #28a745;
}
.delete-btn {
    background-color: #dc3545;
}
.add-btn {
    display: inline-block;
```

```
            margin-top: 20px;
            text-decoration: none;
            background-color:rgb(24, 58, 247);
            color: white;
            padding: 10px 16px;
            border-radius: 6px;
        }
    </style>
</head>
<body>
<header>Prestine Purchases by Piyusha – Employee Data System</header>
<div class="container">
    <form method="POST" action="employee_system.php">
        <h3><?= $updateData ? 'Update Employee' : 'Add New Employee' ?></h3>
        <input type="hidden" name="update_id" value="<?= $updateData['id'] ?? '' ?>">
        <label>Name</label>
        <input type="text" name="name" required value="<?= $updateData['name'] ?? '' ?>">

        <label>Mobile</label>
        <input type="text" name="mobile" required value="<?= $updateData['mobile'] ?? '' ?>">

        <label>Address</label>
        <textarea name="address" required><?= $updateData['address'] ?? '' ?></textarea>

        <label>Status</label>
        <select name="status">
            <option value="Working" <?= (isset($updateData['status']) && $updateData['status'] == 'Working') ?
             'selected' : '' ?>>Working</option>
            <option value="On Leave" <?= (isset($updateData['status']) && $updateData['status'] == 'On Leave')
             ? 'selected' : '' ?>>On Leave</option>
        </select>

        <label>Department</label>
```

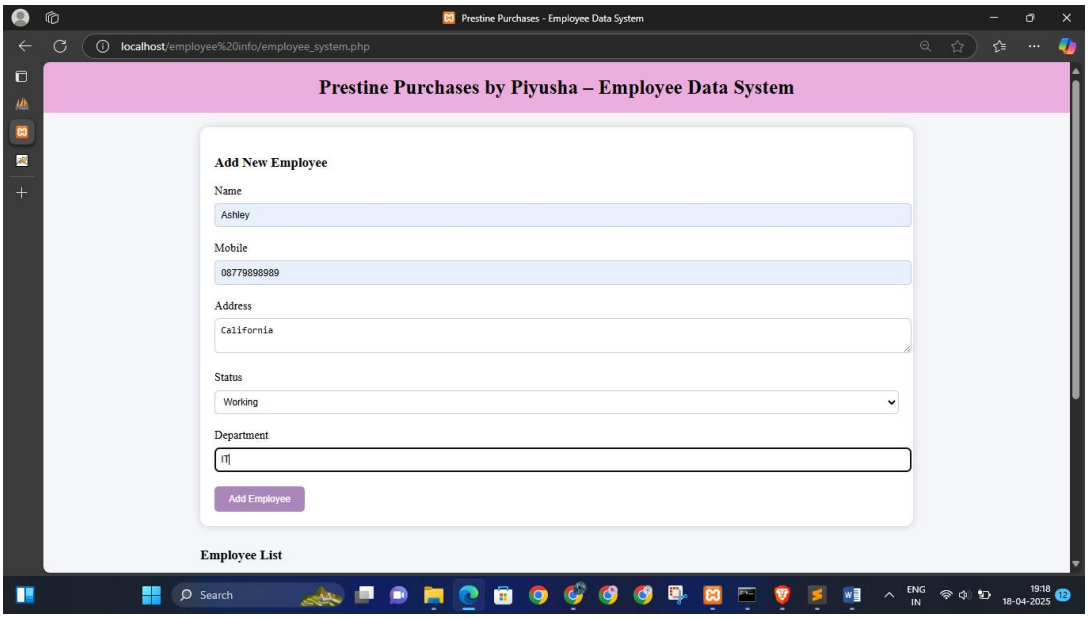
```
    <input type="text" name="department" required value="<?= $updateData['department'] ?? " ?>">

    <button type="submit"><?= $updateData ? 'Update Employee' : 'Add Employee' ?></button>
</form>

<h3>Employee List</h3>
<table>
    <tr>
        <th>Name</th>
        <th>Mobile</th>
        <th>Address</th>
        <th>Status</th>
        <th>Department</th>
        <th>Actions</th>
    </tr>
    <?php
    $result = $conn->query("SELECT * FROM employees ORDER BY id DESC");
    while ($row = $result->fetch_assoc()):
    ?>
    <tr>
        <td><?= htmlspecialchars($row['name']) ?></td>
        <td><?= htmlspecialchars($row['mobile']) ?></td>
        <td><?= htmlspecialchars($row['address']) ?></td>
        <td><?= htmlspecialchars($row['status']) ?></td>
        <td><?= htmlspecialchars($row['department']) ?></td>
        <td class="action-buttons">
            <a href="?edit=<?= $row['id'] ?>" class="edit-btn">Edit</a>
            <a href="?delete=<?= $row['id'] ?>" class="delete-btn" onclick="return confirm('Delete this
        employee?')">Delete</a>
        </td>
    </tr>
    <?php endwhile; ?>
</table>
```

```
    <a href="employee_system.php" class="add-btn">+ Add New Employee</a>
</div>
</body>
</html>
```

The output is as follows:



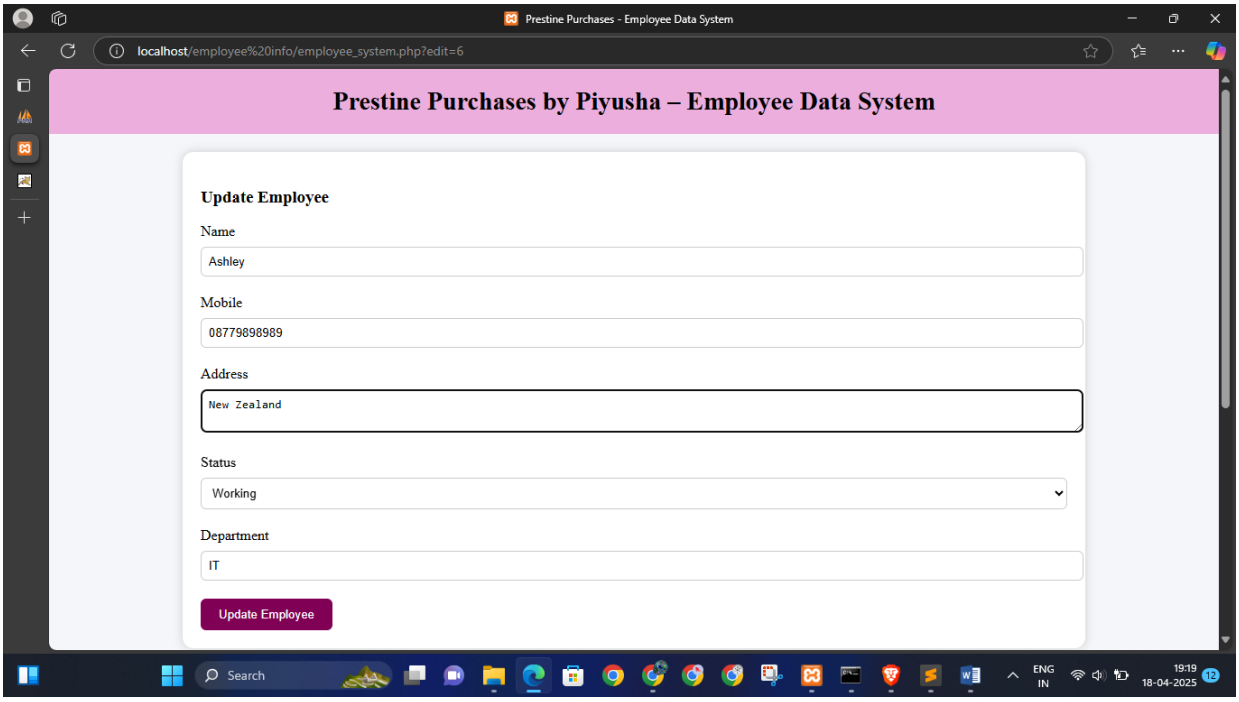
Inserting new employee:



Employee List

Name	Mobile	Address	Status	Department	Actions
Ashley	08779898989	California	Working	IT	Edit Delete
Sanika	9011223344	Aurangabad, Maharashtra		Operations	Edit Delete
Chaitanya	9988776655	Nashik, Maharashtra		Marketing	Edit Delete
Mayank	9001122334	Mumbai, Maharashtra		Finance	Edit Delete
Chinmay	9123456780	Kanyakumari, Maharashtra		IT	Edit Delete
Piyusha	9876543210	Pune, Maharashtra		HR	Edit Delete

Updating:



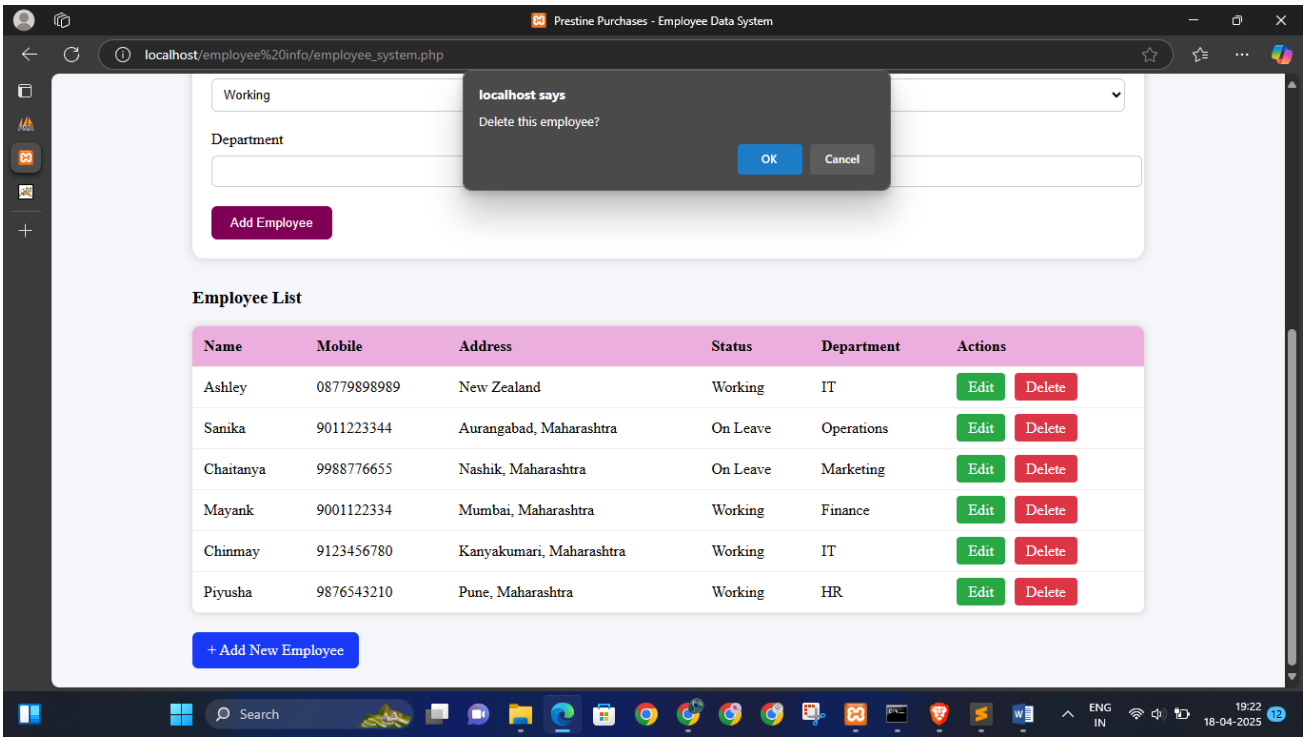
Information is updated:

Employee List

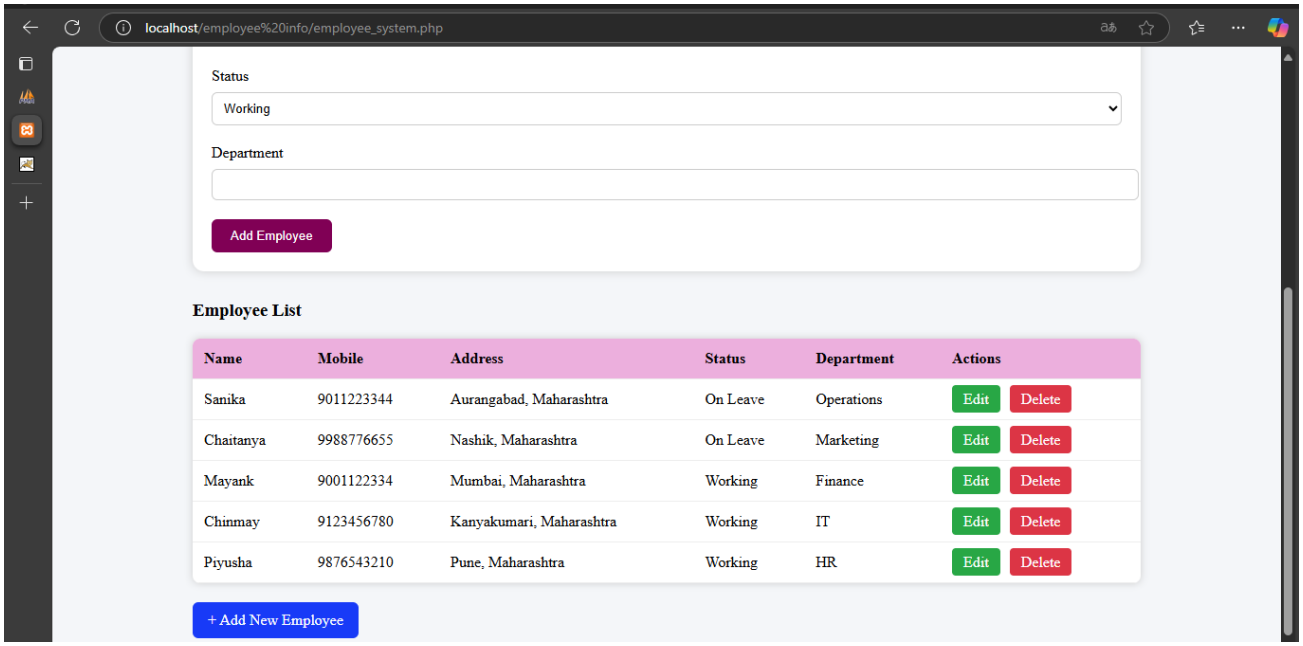
Name	Mobile	Address	Status	Department	Actions
Ashley	08779898989	New Zealand	Working	IT	Edit Delete
Sanika	9011223344	Aurangabad, Maharashtra	On Leave	Operations	Edit Delete
Chaitanya	9988776655	Nashik, Maharashtra	On Leave	Marketing	Edit Delete
Mayank	9001122334	Mumbai, Maharashtra	Working	Finance	Edit Delete
Chinmay	9123456780	Kanyakumari, Maharashtra	Working	IT	Edit Delete
Piyusha	9876543210	Pune, Maharashtra	Working	HR	Edit Delete

+ Add New Employee

Deletion:



The entry is deleted:



Conclusion: Thus I have successfully completed and created an application using PHP which adds, updates, deletes the information through a PHP interface using mysql database.

Experiment - 08.

- **Aim:** Design a login page with entries for name, email mobile number, and login button. Use struts and perform following validations-
  a. Validation for correct names.
  b. Validation for mobile numbers.
  c. Validation for email id.
  d. Validation if not entered any value.
  e. Redisplay for wrongly entered values with message
  f. Congratulations and welcome page upon successful entries.

- **Theory:** 1] STRUTS.

- Apache STRUTS is a java based open source framework used for building web applications.
- It follows the MVC (Model - view - controller) architecture, helping developers separate concerns in web applications making them easier to manage and scale.

STRUTS - provides -

✓ A structured way to handle user input.
✓ Automatic form validation using built in rules.
✓ Easy database interaction with javabeans and JDBC.
✓ Navigation control via XML configuration.

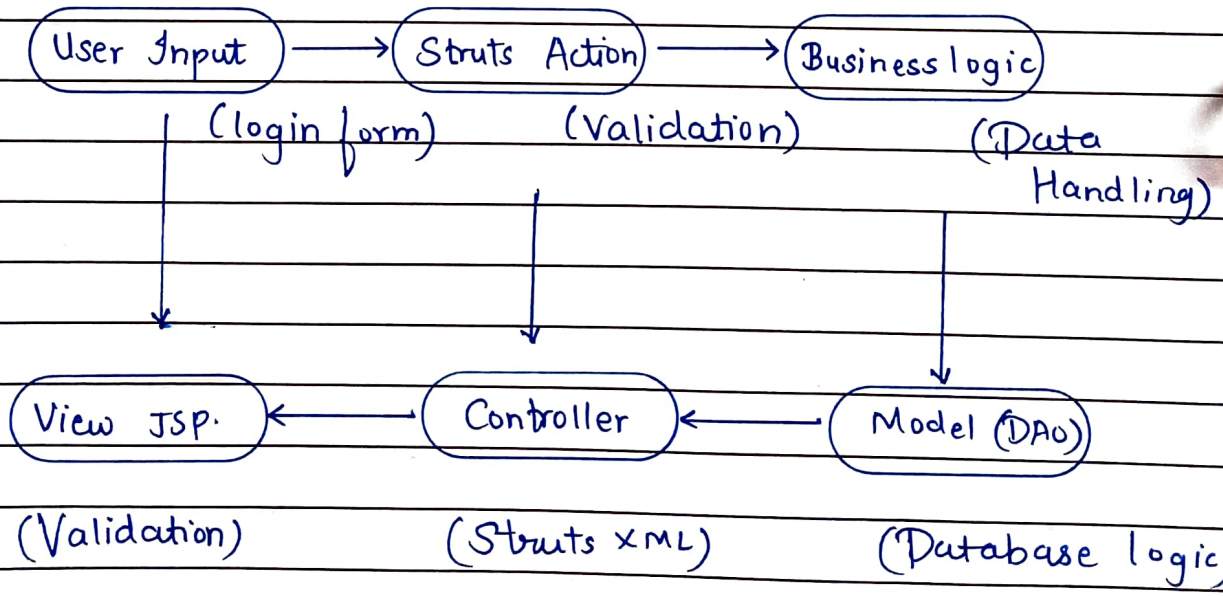
A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

2] For this login system we need:

1. JSP View - Displays the login form and error messages.
2. Action class (Controller): Processes user input, validates data, and directs navigation.
3. struts.xml - Defines form actions, page navigation, validation settings.
4. ActionForm (Model)- Holds user input (name, mobile, email) and provides getter, setter methods.

3] Struts MVC architecture for login page -

Flow diagram -



* Components -

1. Model - Defines business logic (Validation, database interaction)
2. View - JSP pages to display login form and messages.
3. Controller - Struts framework manages request processing via Action classes.

4] Steps to design a login page using struts.

STEP 1 : Set up struts framework.

1. Install Apache struts 2 in your project.
2. Configure web.xml for struts support.
3. Define struts.xml for routing actions.

STEP 2 : Create JSP form for login form.

- HTML form with feilds for Name, Mobile number, Email ID and login button.
- Action: Form submits data to struts action class for validation.
- JSP includes placeholders for displaying error messages dynamically.

STEP 3 : Define struts action class for form processing.

- Retrieves user input using getter, setter methods.
- Implements validation logic.
- Redirects the user to an appropriate page based on validation results.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

5] Implementing validations -

(1.) Validation for correct names.
- Should contain only alphabets and spaces.
- Minimum 2 characters required.

regex → ^[A-Za-z]{2,}$.

(2). Validation for mobile numbers -
- Should be exactly 10 digits.
- Must start with a digit from 6-9

Regex → ^[6-9]\d{9}$.

(3). Validation for email id -
- Should have @ symbol and a valid domain.
- Must start with alphabet or number.

Regex -

^[A-Za-z0-9.-%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$.

(4). Validation for empty fields. -
- Check if any input field is empty.
- Display custom error messages when a field is missing.

* **Conclusion:** Thus we implemented validation in the login form using struts.

* * * * * * *

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Practical 8

Title: Design a login page with entries for name, mobile number email id and login button. Use struts and perform following validations

a. Validation for correct names

b. Validation for mobile numbers

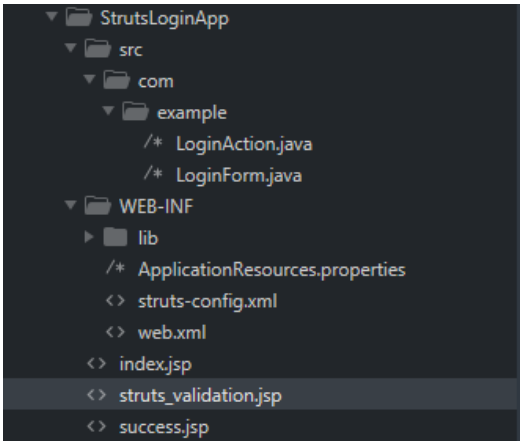
c. Validation for email id

d. Validation if no entered any value

e. Re-display for wrongly entered values with message

f. Congratulations and welcome page upon successful entries

The folder structure is as follows:



The code for each file is as follows:

LoginAction.java

```
package com.example;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.*;

public class LoginAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,

                        HttpServletRequest request, HttpServletResponse response) {


        LoginForm loginForm = (LoginForm) form;

        request.setAttribute("name", loginForm.getName());
```

```
        request.setAttribute("email", loginForm.getEmail());
        request.setAttribute("mobile", loginForm.getMobile());
        return mapping.findForward("success");
    }
}
```

LoginForm.java

```
package com.example;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMapping;
import javax.servlet.http.HttpServletRequest;
public class LoginForm extends ActionForm {
    private String name;
    private String mobile;
    private String email;
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getMobile() { return mobile; }
    public void setMobile(String mobile) { this.mobile = mobile; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    @Override
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
        ActionErrors errors = new ActionErrors();
        if (name == null || name.trim().isEmpty()) {
            errors.add("name", new ActionMessage("error.name.required"));
        } else if (!name.matches("[A-Za-z ]+")) {
            errors.add("name", new ActionMessage("error.name.invalid"));
        }
        if (mobile == null || mobile.trim().isEmpty()) {
            errors.add("mobile", new ActionMessage("error.mobile.required"));
        } else if (!mobile.matches("\d{10}")) {
            errors.add("mobile", new ActionMessage("error.mobile.invalid"));
```

```
        }
        if (email == null || email.trim().isEmpty()) {
            errors.add("email", new ActionMessage("error.email.required"));
        } else if (!email.matches("^\\S+@\\S+\.\S+$")) {
            errors.add("email", new ActionMessage("error.email.invalid"));
        }
        return errors;
    }
}
```

ApplicationResources.properties

```
error.name.required = Name is required.
error.name.invalid = Name must contain only letters and spaces.
error.mobile.required = Mobile number is required.
error.mobile.invalid = Mobile number must be 10 digits.
error.email.required = Email is required.
error.email.invalid = Invalid email format.
```

Struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="loginForm" type="com.example.LoginForm" />
  </form-beans>
  <action-mappings>
    <action path="/login" type="com.example.LoginAction" name="loginForm" scope="request"
          validate="true" input="/index.jsp">
      <forward name="success" path="/success.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

Web.xml

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

```
 <display-name>StrutsLoginApp</display-name>
 <servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
   <param-name>config</param-name>
   <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
 <taglib>
 <taglib-uri>http://struts.apache.org/tags-html</taglib-uri>
 <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
</web-app>
```

Success.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head><title>Success</title></head>
<body>
<h2>Welcome!</h2>
<p>Congratulations <b><%= request.getAttribute("name") %></b></p>
<p>Email: <%= request.getAttribute("email") %></p>
<p>Mobile: <%= request.getAttribute("mobile") %></p>
</body>
</html>
```

Struts_validation.jsp

```
<%@ page import="java.util.regex.*" %>
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    String name = request.getParameter("name");
    String mobile = request.getParameter("mobile");
    String email = request.getParameter("email");

    boolean isSubmitted = request.getParameter("submit") != null;
    boolean isValid = true;
    String nameError = "", mobileError = "", emailError = "";

    if (isSubmitted) {
        if (name == null || name.trim().equals("")) {
            nameError = "Name is required.";
            isValid = false;
        } else if (!name.matches("^[A-Za-z\\s]+$")) {
            nameError = "Name must contain only letters.";
            isValid = false;
        }

        if (mobile == null || mobile.trim().equals("")) {
            mobileError = "Mobile number is required.";
            isValid = false;
        } else if (!mobile.matches("^[0-9]{10}$")) {
            mobileError = "Mobile number must be 10 digits.";
            isValid = false;
        }

        if (email == null || email.trim().equals("")) {
            emailError = "Email is required.";
            isValid = false;
        } else {
            Pattern emailPattern = Pattern.compile("^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$");
```

```
            Matcher matcher = emailPattern.matcher(email);
            if (!matcher.matches()) {
                emailError = "Invalid email format.";
                isValid = false;
            }
        }
    }
%>
<html>
<head>
    <title>Login Page with JSP Validation</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background: #e8f0fe;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        .form-container {
            background: white;
            padding: 25px;
            border-radius: 12px;
            box-shadow: 0 0 15px rgba(0,0,0,0.1);
            width: 350px;
        }
        h2 {
            text-align: center;
            margin-bottom: 20px;
            color: #3366cc;
        }
```

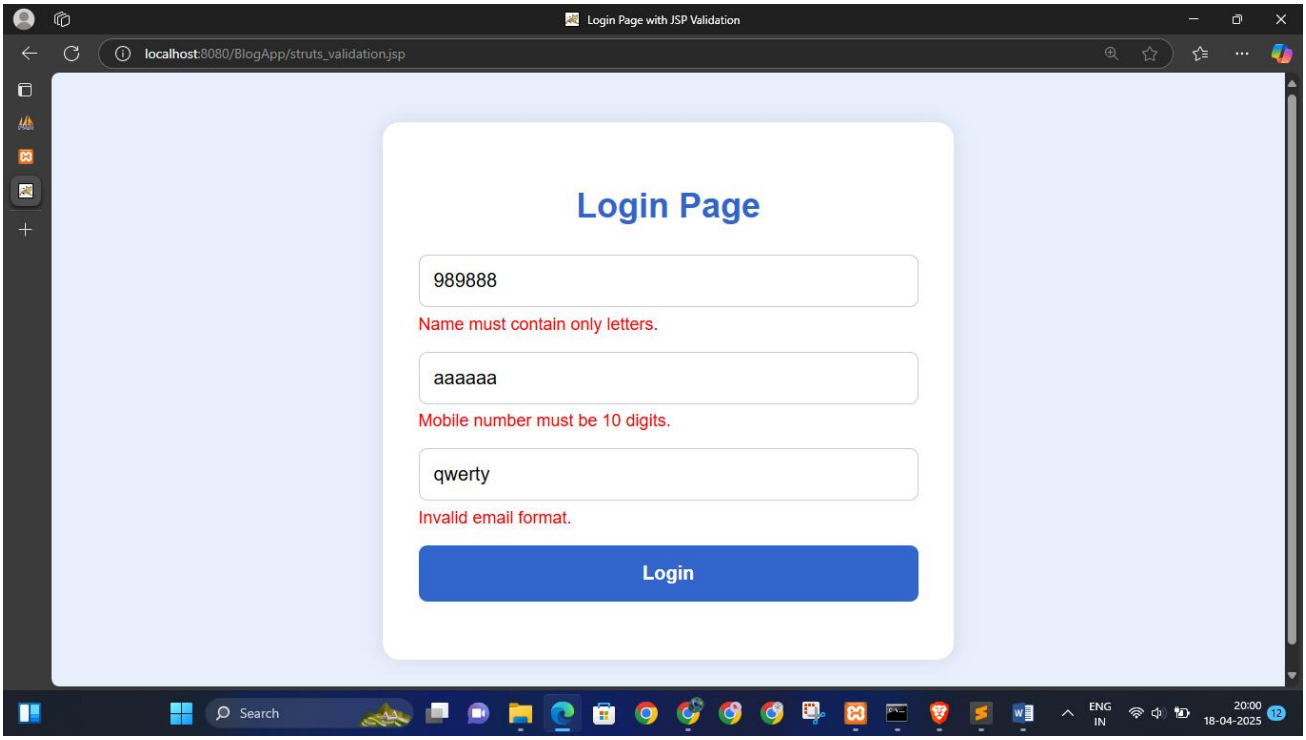
```
    input[type="text"] {
        width: 100%;
        padding: 10px;
        margin-bottom: 5px;
        border-radius: 6px;
        border: 1px solid #ccc;
    }
    .error {
        color: red;
        font-size: 12px;
        margin-bottom: 12px;
    }
    input[type="submit"] {
        width: 100%;
        padding: 12px;
        background: #3366cc;
        border: none;
        border-radius: 6px;
        color: white;
        font-weight: bold;
        cursor: pointer;
    }
    .welcome {
        background: #d4edda;
        color: #155724;
        padding: 15px;
        border-radius: 8px;
        text-align: center;
        font-size: 18px;
        font-weight: bold;
    }
</style>
```

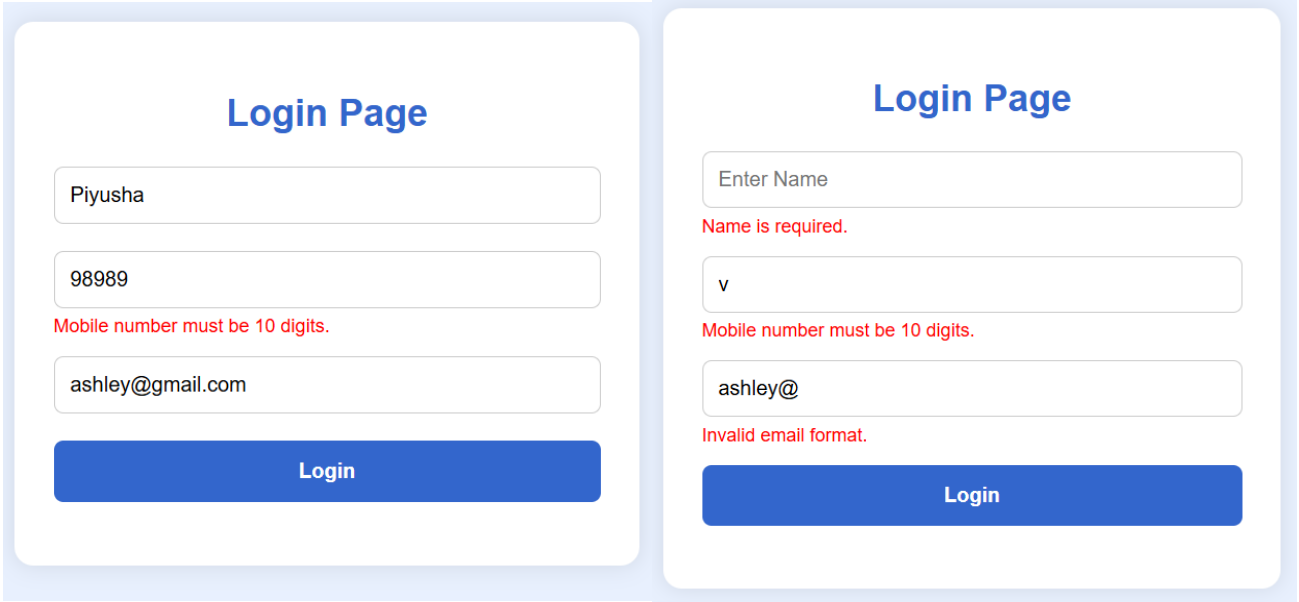
```
</head>
<body>
<div class="form-container">
    <%
        if (isSubmitted && isValid) {
    %>
        <div class="welcome">
            Congratulations <%= name %>!<br>Welcome to the system.
        </div>
    <%
        } else {
    %>
    <form method="post">
        <h2>Login Page</h2>


        <input type="text" name="name" placeholder="Enter Name" value="<%= name != null ? name : ""
            %>">
        <div class="error"><%= nameError %></div>


        <input type="text" name="mobile" placeholder="Enter Mobile Number" value="<%= mobile != null ?
            mobile : "" %>">
        <div class="error"><%= mobileError %></div>


        <input type="text" name="email" placeholder="Enter Email ID" value="<%= email != null ? email : ""
            %>">
        <div class="error"><%= emailError %></div>


        <input type="submit" name="submit" value="Login">
    </form>
    <%
        }
    %>
</div>
```

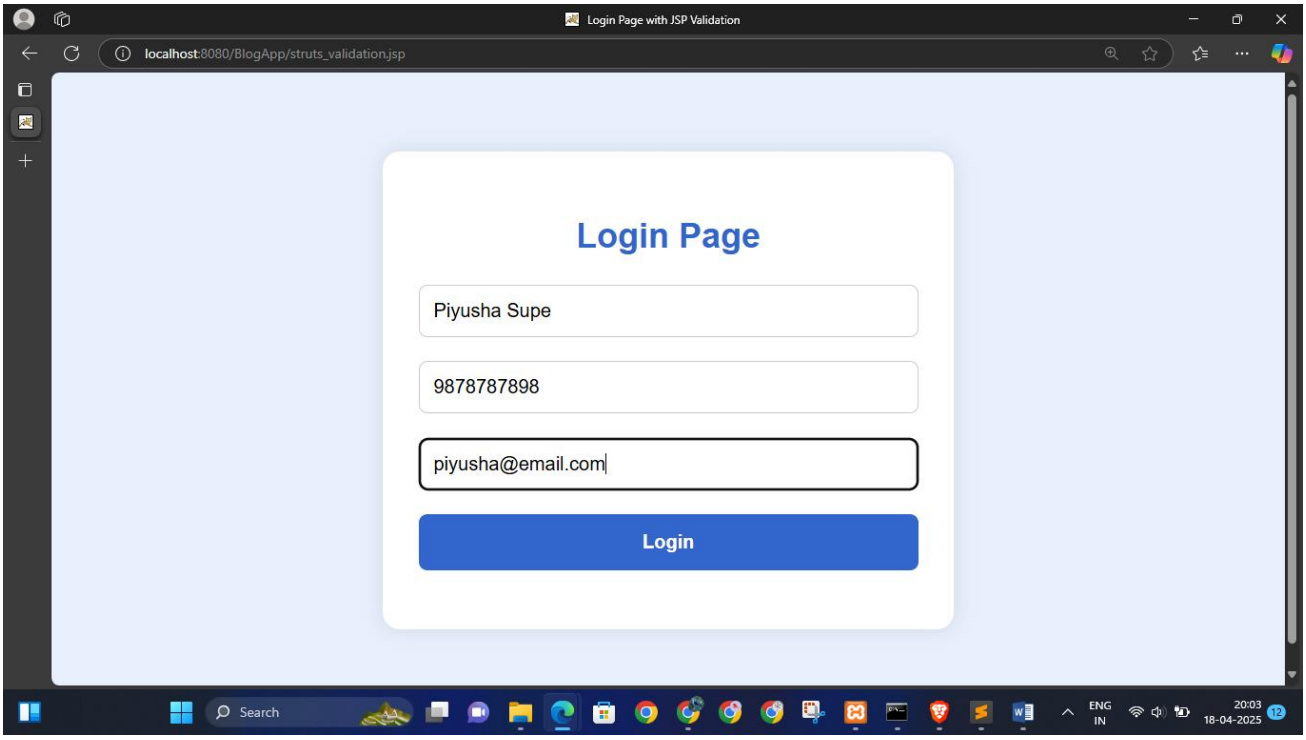
</body>

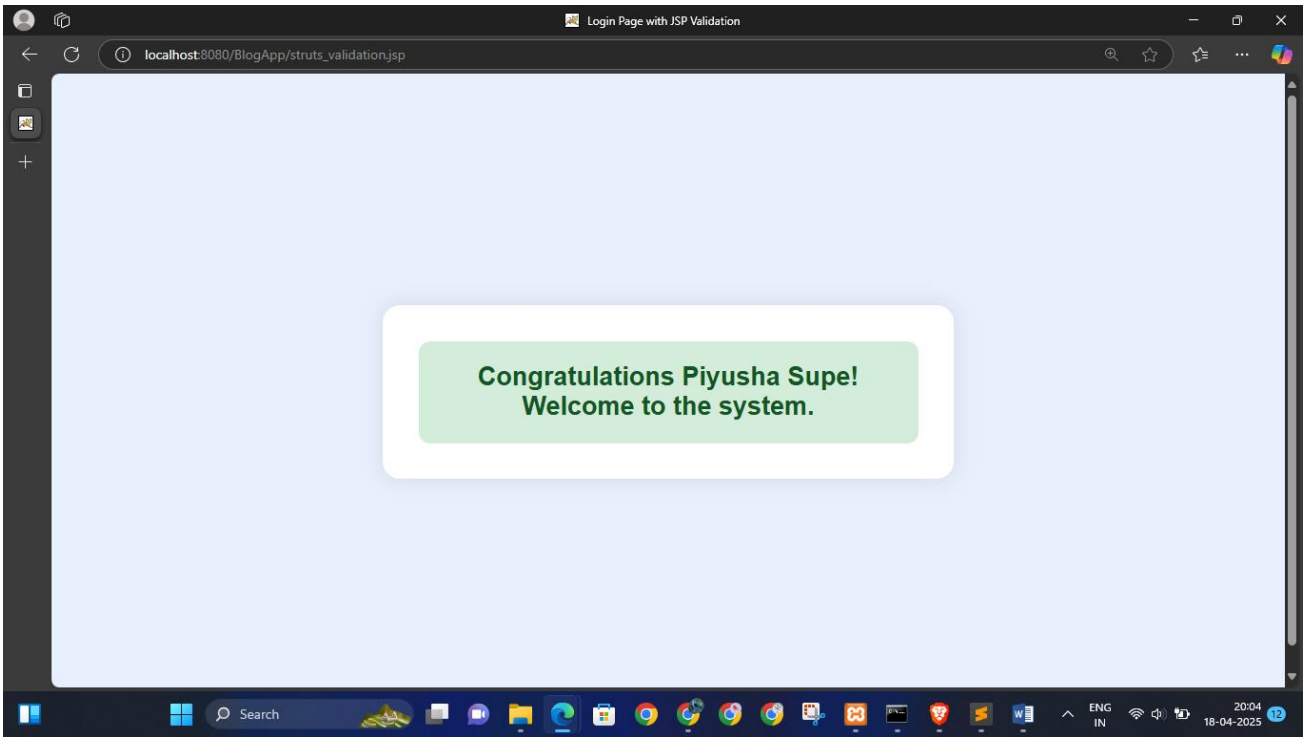
</html>

The output is as follows:









Conclusion: Thus in this way we have successfully configured the struts validation using the Apache struts framework

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

Experiment - 09.

- **Aim:** Design an application using angular JS.
  eg. Design registration (first name, last name, username, password) and login page using Angular JS.

- **Theory:**

(1). Angular JS:

- Angular JS is a javascript framework developed and maintained by google. It is used for building dynamic web applications, particularly single page applications

- Angular JS extends HTML with additional attributes and provides a structure for organizing and managing client side web applications.

(2). Advantages:

1. MVC support - It is based on model view controller architecture which provides separation of model, view and controller components.

2. Comprehensive:- Angular JS is a comprehensive solution for rapid frontend development.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

3. Unit Testing - The angular JS code is unit Testable.

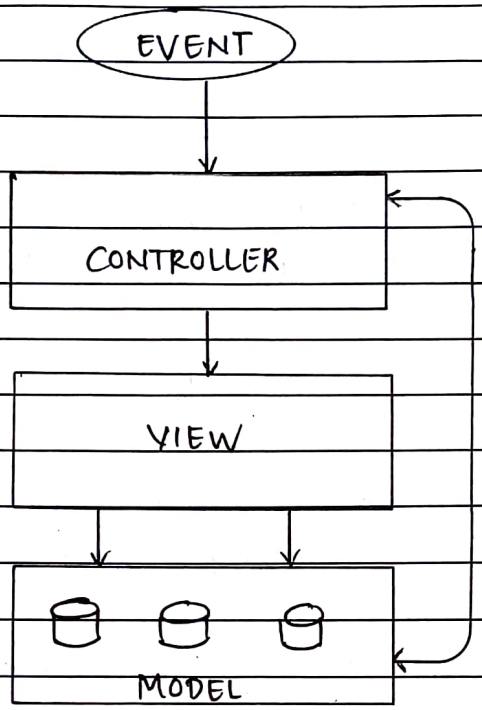
4. Reusable code - Angular JS supports for using reusable components.

5. less code - It supports less code and more functionality.

(3). Features of Angular JS -

1. Directives - designed to give HTML new functionality.
2. Filters - This feature allows to select subset from array of items.
3. Routing - This feature allows to switch between multiple views.
4. Services - There is support for many built in services for angular JS.
5. Data Binding - It allows the automatic synchronisation between model and view components.
6. Scope - There are some objects from model that can be correlated to controller and view.

(4). MVC Model -



1. Model - It is the lowest level of the pattern responsible for maintaining data.

2. View - It is responsible for displaying all or a portion of the data to the user.

3. Controller - It is a software code that controls the interactions between the model and view.

(5). An angular JS application consists of following three important parts -

1. ng-app - This directive defines and links an angularJS application to HTML.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

2. ng-model - This directive binds the values of Angular JS application data to HTML input controls.

3. ng-bind - This directive binds the Angular JS application data to HTML tags.

(6). Creating angular JS application -

Step 1 - Load framework - Being a pure Javascript framework, it can be added using <script> tag

```
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
```

Step 2: Define angular JS application using ng-app directive.

```
<div ng-app = " ">
...
</div>
```

Step 3: Define a model name using ng-model directive.

```
<p> Enter your name: <input type = "text"
ng-model = "name" ></p>
```

Step 4: Bind the value of above model defined using ng-bind directive.

```
<p> Hello <span ng-bind = "name" ></span>
!</p>
```

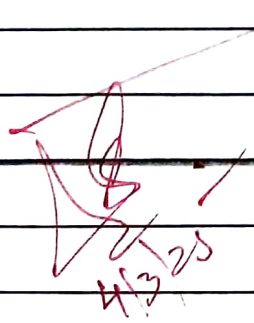
A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- How angular JS Integrates with HTML -

1. The ng-app directive indicates the start of AngularJS application.
2. The ng-model directive creates a model variable named name, which can be used with the HTML page and within the div having ng-app directive.
3. The ng-bind then uses the name model to be displayed in the HTML tag whenever user enters input in text box.
4. Closing tag indicates the end of AngularJS application.

- ng-init - This directive is used for initializing an angularJS application data. Used to assign values to variables.
- ng-repeat - This repeats the html elements for each item in a collection.

* **Conclusion:** Hence we successfully created an angular JS application using the different directives.



* * * * * * * *

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Practical 9

Title: Design an application using Angular JS. e.g., Design registration (first name, last name, username, password) and login page using Angular JS.

The file login.html is as follows:

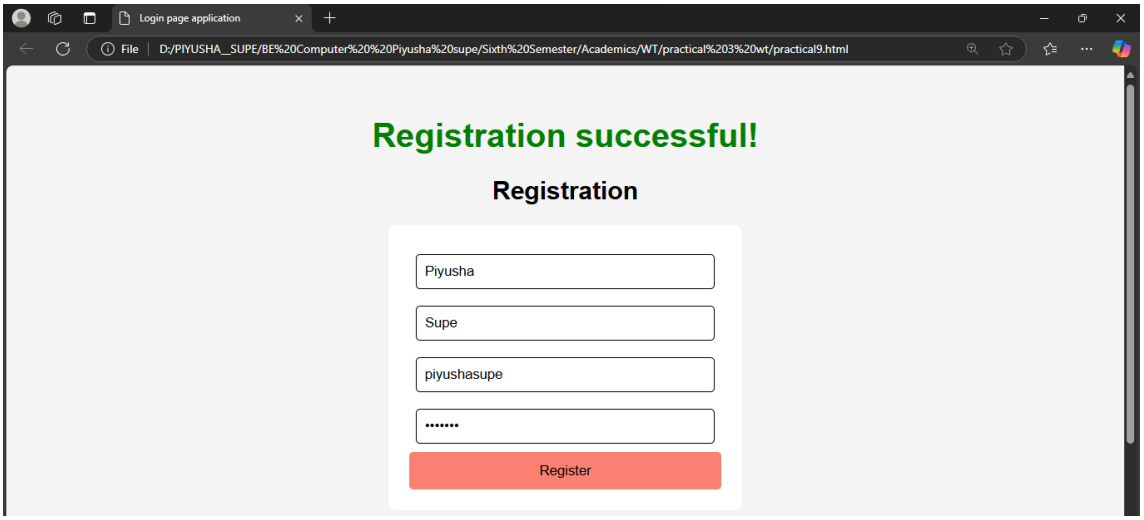
```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <title>Login page application</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
    <style type="text/css">
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    text-align: center;
    padding: 20px;
}
form {
    background: white;
    padding: 20px;
    width: 300px;
    margin: 10px auto;
    border-radius: 8px;
}
input {
    width: 90%;
    padding: 8px;
    margin: 8px 0;
    border: 1px solid black;
```

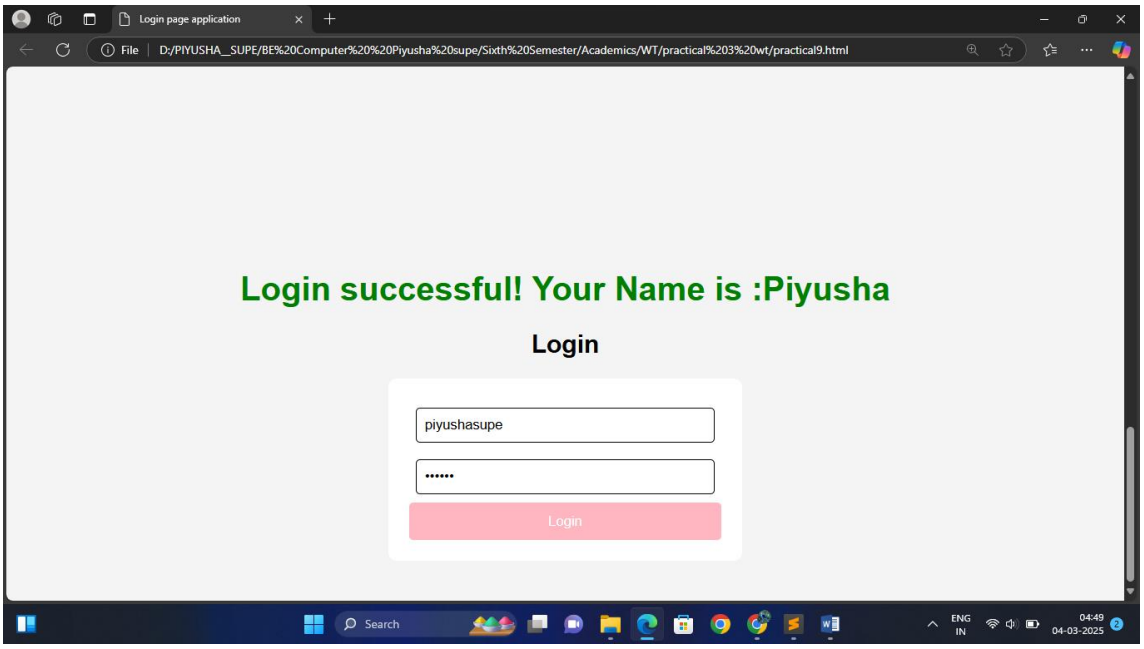
```
    border-radius: 4px;
}
button {
    width: 100%;color:black;
    padding: 10px;
    background: lightpink;
    color: white;
    border: none;
    border-radius: 4px;
}
button:hover {
    background: salmon;
    color: black;
}
p {
    font-weight: bold;
    color: #333;
    margin-top: 10px;
}
    </style>
</head>
<body ng-controller="MainController">
    <center>
        <h1 style="color:green;">{{ message }}</h1>
    <h2>Registration</h2>
    <form ng-submit="register()">
        <input type="text" ng-model="user.firstName" placeholder="First Name" required><br>
        <input type="text" ng-model="user.lastName" placeholder="Last Name" required><br>
        <input type="text" ng-model="user.username" placeholder="Username" required><br>
        <input type="password" ng-model="user.password" placeholder="Password" required><br>
        <button type="submit">Register</button>
    </form>
```

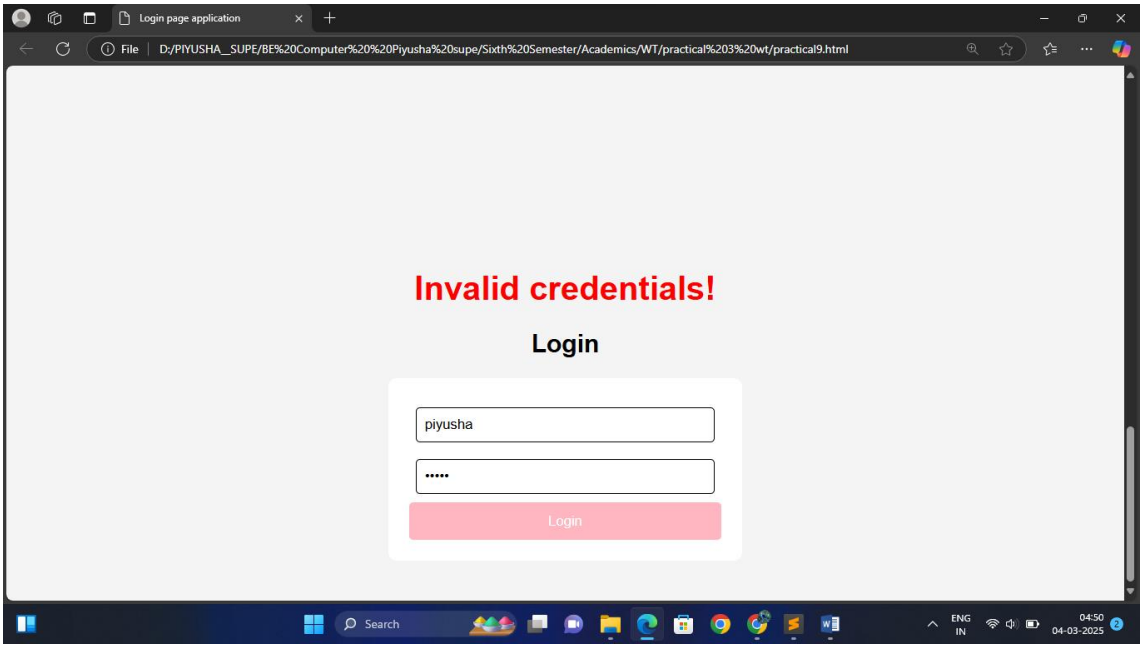
```
    <h2>Login</h2>
    <form ng-submit="login()">
        <input type="text" ng-model="loginData.username" placeholder="Username" required><br>
        <input type="password" ng-model="loginData.password" placeholder="Password" required><br>
        <button type="submit">Login</button>
    </form>
    <script>
        var app = angular.module('myApp', []);
        app.controller('MainController', function ($scope) {
            var registeredUser = {};

            $scope.register = function () {
                registeredUser = angular.copy($scope.user);
                $scope.message = "Registration successful!";
            };
            $scope.login = function () {
                if ($scope.loginData.username === registeredUser.username &&
                    $scope.loginData.password === registeredUser.password) {
                    $scope.message = "Login successful! Your Name is :" + registeredUser.firstName;
                } else {
                    $scope.message = "Invalid credentials!";
                }
            };
        });
    </script>
</center>
</body>
</html>
```

OUTPUT:

Registration


Login page application
File
D:/PIYUSHA_SUPE/BE%20Computer%20%20Piyusha%20supe/Sixth%20Semester/Academics/WT/practical%203%20wt/practical9.html
Registration successful!
Registration
Piyusha
Supe
piyushasupe
Register


Login page application
File
D:/PIYUSHA_SUPE/BE%20Computer%20%20Piyusha%20supe/Sixth%20Semester/Academics/WT/practical%203%20wt/practical9.html
Login successful! Your Name is :Piyusha
Login
piyushasupe
Login
Search
ENG
IN
04:49
04-03-2025


Login page application
File
D:/PIYUSHA_SUPE/BE%20Computer%20%20Piyusha%20supe/Sixth%20Semester/Academics/WT/practical%203%20wt/practical9.html
Invalid credentials!
Login
piyusha
Login
Search
ENG
IN
04:50
04-03-2025

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

Experiment - 10.

- **Aim:** Design and implement a business interface with necessary business logic for any web application using EJB.

  eg: Design and implement the web application logic for deposit and withdrawl amount transaction using EJB.

- **Theory:**

I] Introduction to Enterprise JavaBeans (EJB).

- Enterprise JavaBeans (EJB) is a serverside component based architecture for building scalable, secure, and transactional enterprise applications.

- EJB is particularly useful for applications that require business logic execution, distributed computing and database interactions.

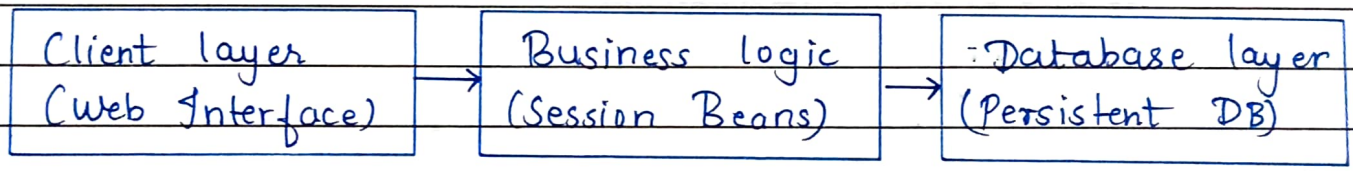
II] Why use EJB ?

For a transaction system like deposit and withdrawl, we need :-

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- Security - Transactions must be secured against unauthorized access.
- Concurrency management - Multiple users may deposit / withdraw simultaneously.
- Data Persistence: Transaction details must be stored in a data base.
- Transaction Management - To ensure consistency, transactions must follow ACID properties.

III] EJP Architecture / flow -



- Components we will use -

1. Client (JSP / Servelets / Web UI) - Allows users to input deposit / withdrawl details.

2. Session Beans (EJB business logic layer) - Implements deposit and withdrawl logic.

3. Entity Beans - (Database Handling) - Manages persistent storage for transactions.

4. JNDI (Java naming and Directory Interface) - Allows clients to locate and access EJB components.

5. Transaction Manager - Ensures deposit / withdraw transactions are automic.

* Steps to design and Implement EJB - Based application -

STEP 1: Setting up the EJB Environment.

- Install Java EE and EJB container (Wildfly, Glassfish, JBoss) etc.
- Configure database (MySQL, Oracle, etc) for storing transactions records.
- Create a JDBC connection - between EJB and database.

STEP 2: Designing the Bissiness Interface for transactions.

The interface will declare methods such as -

```
public interface BankingService {
    void deposit (double amount, int accountID);
    void withdraw (double amount, int accountID);
    double getBalance (int accountID);
}
```

STEP 03: Implementing the Business logic using Session beans.

These are two types of session beans -

1. Stateless beans (Best for transaction processing)
2. Stateful beans (Useful for maintaining user session

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

48.

Business logic for -

(1.) Deposit logic -

- Check if the account exists.
- Add the deposited amount to existing balance.
- Commit the transaction to update database.

(2). Withdrawl logic -

- Verify if account exists.
- Check if balance is sufficient
- Deduct withdrawl amount.
- Commit the transaction

STEP 4: Database connectivity using entity beans.

```
@ Entity
public class Account {
        @Id
        private int accountID;
        private string accountHolderName;
        private double balance;
        // getters and setters.
}
```

STEP 5: Managing Transactions using EJB.

EJB provides automatic transaction Management using annotations.

@Transaction Attribute.

```
public void deposit (double amount, int accountID)
{
    Account acc = entityManager.find (Account.class,
                  accountID);
    acc.setBalance (acc.getBalance () + amount);
    entityManager.merge (acc);
}
```

If transaction fails system automatically rolls back to prevent data corruption.

STEP 6: Exposing EJB Service to clients

EJB functions are called from a web interface (JSP/ Servelets).

1. User enters account ID and amount in a JSP form.
2. The request is sent to a Servelet.
3. Servelet invokes EJB methods deposit() or withdraw().
4. EJB updates the database.
5. Response is sent back to user.

Conclusion: Thus we successfully implemented business logic using EJB.

* * * * * * *

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology Practical 10

Title: Design and implement a business interface with necessary business logic for any web application using EJB. e.g., Design and implement the web application logic for deposit and withdraw amount transactions using EJB

The code is as follows:

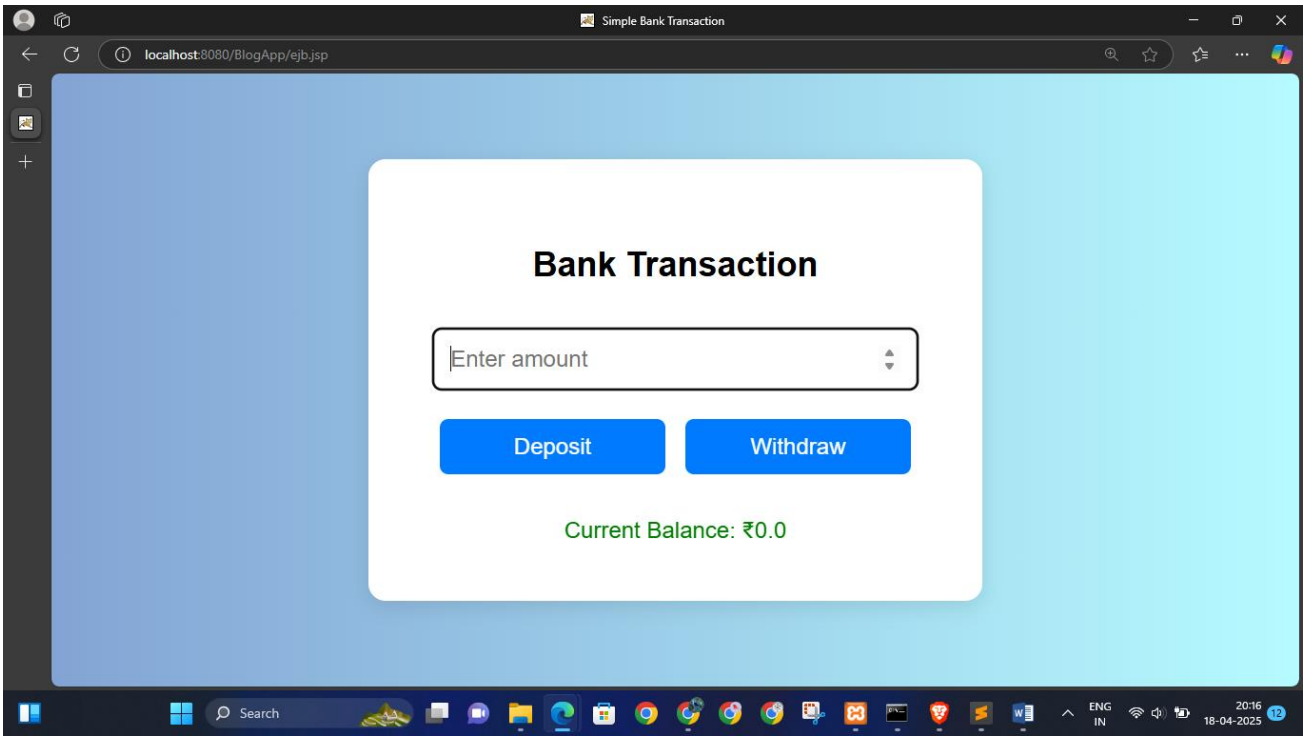
```
<%@ page import="javax.naming.*, com.ejb.TransactionBeanRemote" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Bank Transaction</title>
    <style>
        body {
            background: linear-gradient(to right, #83a4d4, #b6fbff);
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        .container {
            background: #fff;
            padding: 40px;
            border-radius: 10px;
            box-shadow: 0 5px 15px rgba(0,0,0,0.2);
            width: 350px;
            text-align: center;
        }
        h2 {
            margin-bottom: 20px;
            color: #333;
```

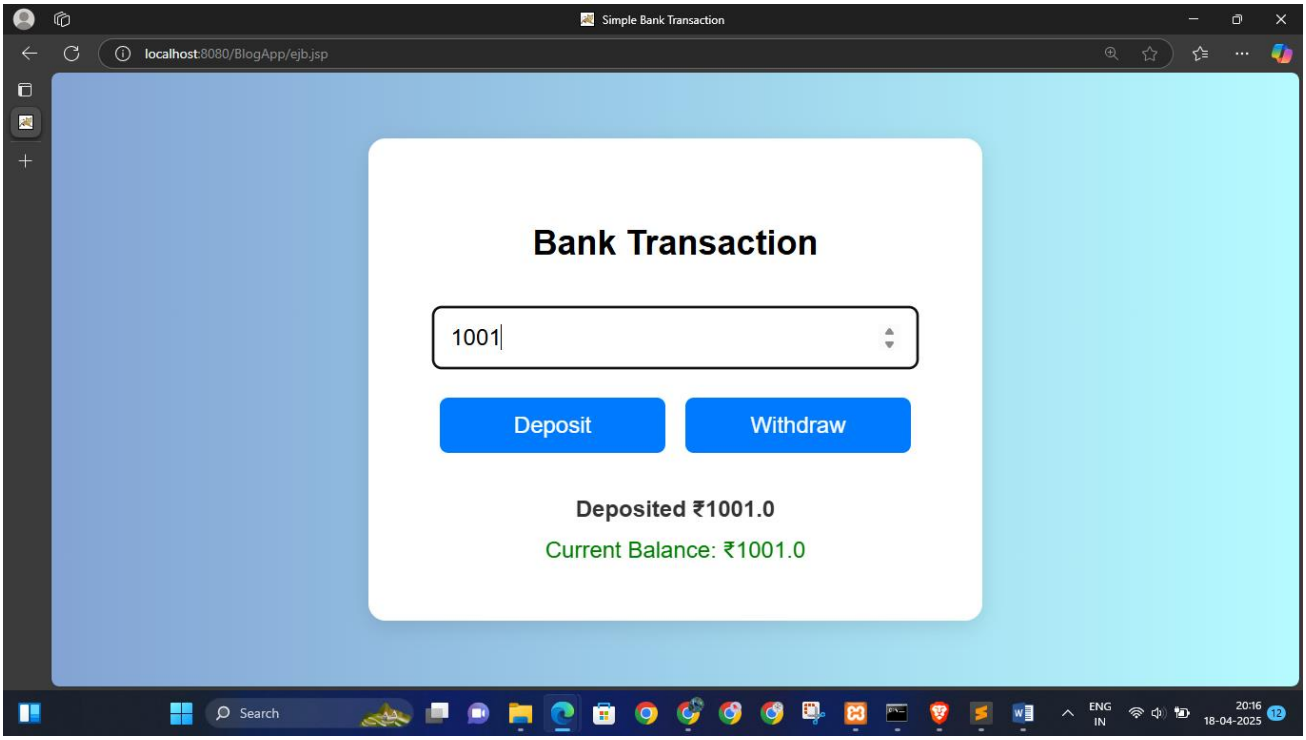
```
        }
        input {
            width: 90%;
            padding: 12px;
            margin: 10px 0;
            font-size: 16px;
            border: 1px solid #ccc;
            border-radius: 6px;
        }
        button {
            width: 45%;
            padding: 10px;
            margin: 10px 5px;
            font-size: 16px;
            color: white;
            background: #007bff;
            border: none;
            border-radius: 6px;
            cursor: pointer;
        }
        button:hover {
            background: #0056b3;
        }
        .result {
            margin-top: 20px;
            font-weight: bold;
            color: #222;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Bank Transaction</h2>
        <form method="post">
```

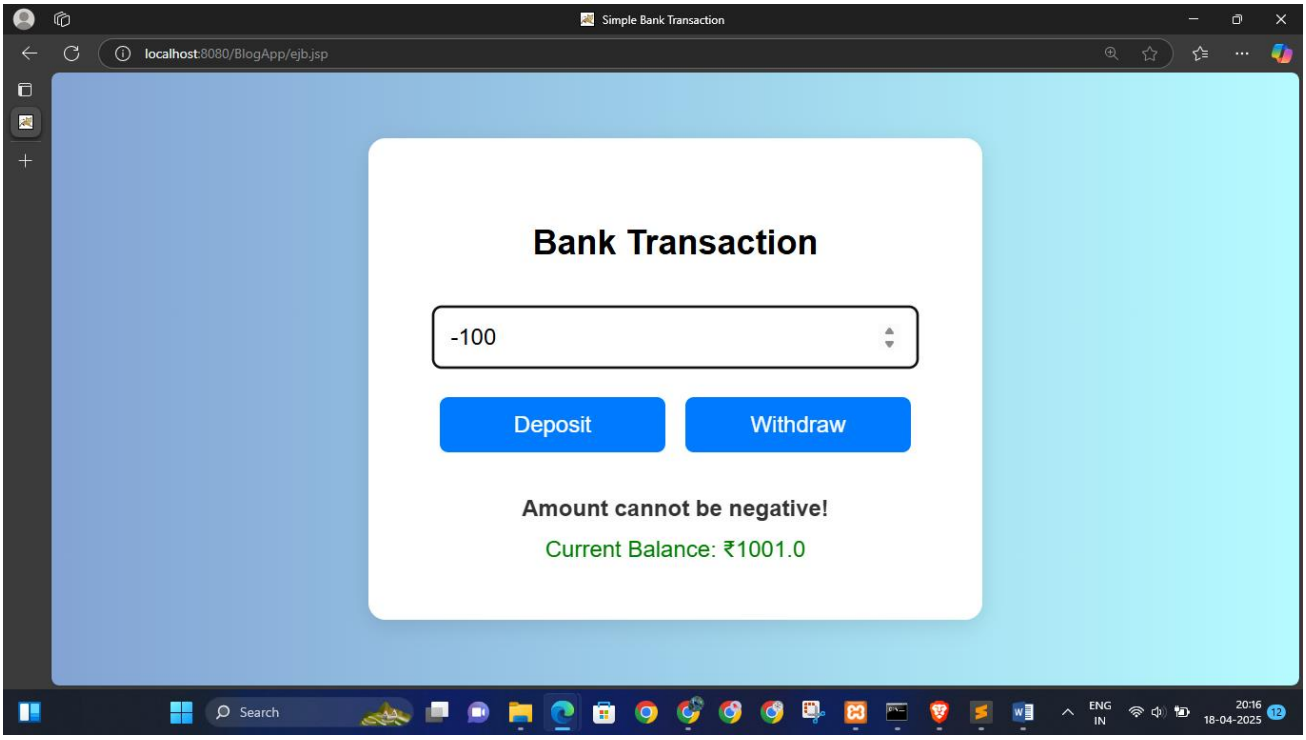
```
        <input type="text" name="account" placeholder="Account Number" required><br>
        <input type="number" name="amount" placeholder="Amount" required><br>
        <button name="action" value="deposit">Deposit</button>
        <button name="action" value="withdraw">Withdraw</button>
    </form>
    <div class="result">
        <%
            String result = "";
            try {
                String account = request.getParameter("account");
                String amountStr = request.getParameter("amount");
                String action = request.getParameter("action");
                if (account != null && amountStr != null && action != null) {
                    double amount = Double.parseDouble(amountStr);
                    InitialContext ctx = new InitialContext();
                    TransactionBeanRemote bean = (TransactionBeanRemote)
ctx.lookup("java:global/BankApp/TransactionBean!com.ejb.TransactionBeanRemote");
                    if ("deposit".equals(action)) {
                        double newBalance = bean.deposit(account, amount);
                        result = "Deposit successful. New balance: ₹" + newBalance;
                    } else if ("withdraw".equals(action)) {
                        double newBalance = bean.withdraw(account, amount);
                        result = "Withdraw successful. New balance: ₹" + newBalance;
                    }
                }
            } catch (Exception e) {
                result = "Error: " + e.getMessage();
            }
            out.println(result);
// Initialize account balance in session if not already set
    Double balance = (Double) session.getAttribute("balance");
    if (balance == null) {
        balance = 0.0;
        session.setAttribute("balance", balance);
```

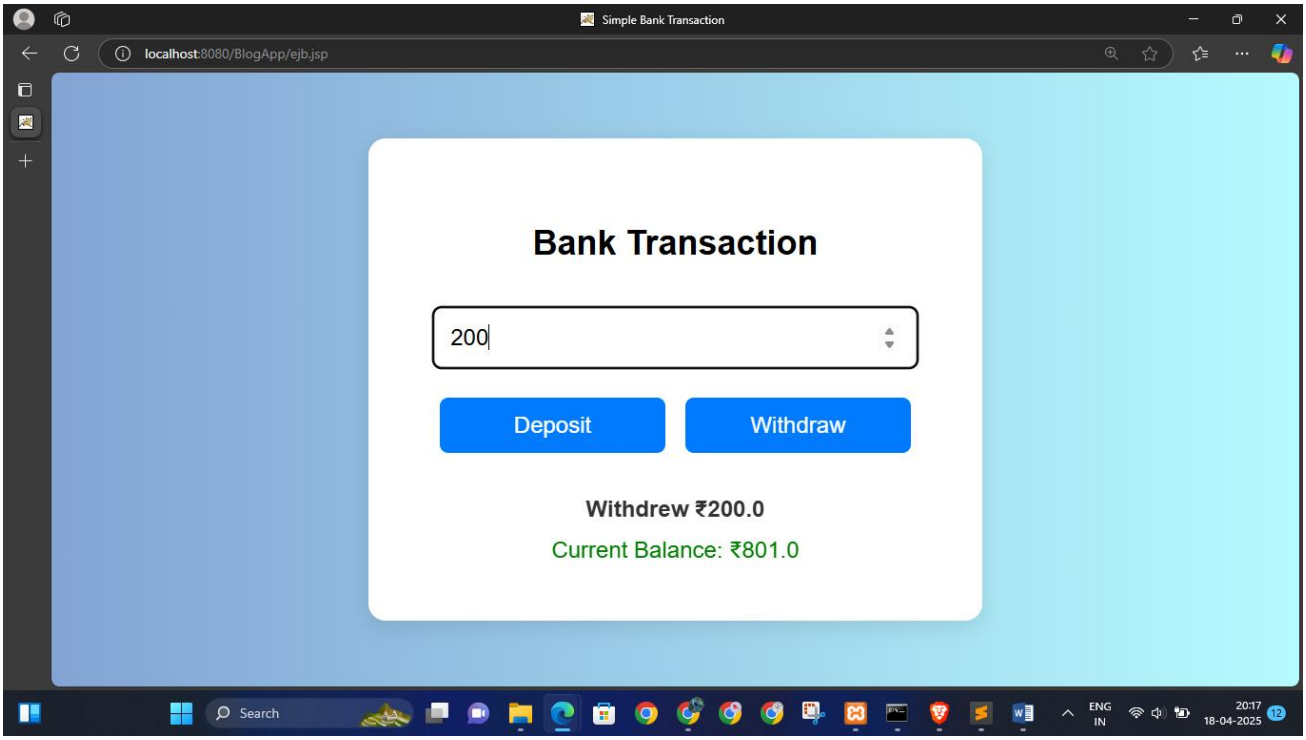
```
    }/*
    String message = "";
    String action = request.getParameter("action");
    String amountStr = request.getParameter("amount");
    if (action != null && amountStr != null) {
        try {
            double amount = Double.parseDouble(amountStr);
            if (amount < 0) {
                message = "Amount cannot be negative!";
            } else {
                if ("deposit".equals(action)) {
                    balance += amount;
                    message = "Deposited ₹" + amount;
                } else if ("withdraw".equals(action)) {
                    if (amount > balance) {
                        message = "Insufficient funds!";
                    } else {
                        balance -= amount;
                        message = "Withdrew ₹" + amount;
                    }
                }
                session.setAttribute("balance", balance);
            }
        } catch (NumberFormatException e) {
            message = "Invalid amount!";
        }
    }*/
            %>
        </div>
    </div>
</body>
</html>
```

The output is as follows:









Conclusion: Thus we successfully implemented business logic using java bean for bank transaction simple logic and a web interface

AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय

Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

Department of Computer Engineering

"WT Mini-project Report"

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

In

COMPUTER ENGINEERING

Submitted By

Name of the Student: Piyusha Rajendra Supe

Roll No: 23CO315

Under the Guidance of

Dr. S. F. Sayyad

ALL INDIA SHRI SHIVAJI MEMORIAL SOCIETY'S COLLEGE OF ENGINEERING PUNE-411001

Academic Year: 2024-25(Term-II)

Savitribai Phule Pune University

AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय

Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

Department of Computer Engineering

CERTIFICATE

This is to certify that **Piyusha Rajendra Supe** from **Third Year Computer Engineering** has successfully completed her work titled **"Web Technology Mini-project**" at AISSMS College of Engineering, Pune in the partial fulfillment of the Bachelor's Degree in Engineering.

Dr. S. F. Sayyad	**Dr. S. V. Athawale**	**Dr. D. S. Bormane**
(Faculty Guide)	(Head of Department)	(Principal)
Computer Engineering	Computer Engineering	AISSMSCOE, Pune

ACKNOWLEDGEMENT

It is with immense gratitude and deep respect that I take this opportunity to acknowledge the invaluable support and guidance I have received throughout the course of my Web Technology project. This endeavour has been a truly enriching experience—both intellectually and personally—and has significantly enhanced my understanding of web development tools, technologies, and real-world implementation strategies. First and foremost, I would like to express my heartfelt thanks to **Dr. S.F. Sayyad** for her expert guidance, unwavering support, and insightful feedback. Her encouragement and mentorship were instrumental in shaping the direction, depth, and quality of this project. I extend my sincere gratitude to the Head of the Department, whose leadership and commitment to academic excellence have fostered an environment of continuous learning and innovation. I am truly thankful for the resources, support, and opportunities provided under their stewardship. A special note of appreciation goes to the staff, whose timely assistance and cooperation ensured a smooth and efficient workflow during the development phase. Most importantly, I would like to express my deepest gratitude to my parents, whose unconditional love, patience, and constant encouragement have been my greatest source of strength. Their belief in my potential and their continuous emotional and moral support played a vital role in helping me stay focused and motivated throughout this journey. This project has not only strengthened my technical skills but has also taught me the value of perseverance, and continuous learning. I remain sincerely thankful to everyone who contributed to the successful completion of this project.

Academic Year: 2024-2025

Piyusha Rajendra Supe (23CO315)

TABLE OF CONTENTS

ABSTRACT

This Web Technology project presents the design and implementation of a dynamic and interactive web-based application utilizing a combination of client-side and server-side technologies. The objective of this project is to gain hands-on experience in building a complete web solution by integrating frontend and backend technologies effectively. The technologies used include **HTML**, **CSS**, and **JavaScript** for the frontend development, while **PHP**, **MySQL**, **Java**, and **JSP (JavaServer Pages)** are employed in the backend to manage logic, server communication, and data operations.

The frontend of the application is designed using HTML and CSS to structure and style the content, ensuring that the interface is user-friendly and responsive. JavaScript is used to enhance interactivity and provide a seamless user experience by enabling client-side validation and dynamic behaviour without the need to reload the page. The frontend serves as the primary interface through which users interact with the system, and careful attention is given to layout, accessibility, and responsive design.

On the server side, **PHP** is used for handling requests and processing data submitted from the client side. It acts as the bridge between the frontend and the database, managing essential functionalities such as form submission, data validation, and retrieval. **MySQL** is used as the relational database management system for storing and managing structured data securely and efficiently. Data such as user information, form inputs, or system configurations are stored in tables, with structured queries used for retrieval and manipulation.

To further enhance the scope of the project, **Java** and **JSP (JavaServer Pages)** are incorporated into the backend to demonstrate the integration of Java-based server-side technologies with the overall web application. JSP enables the creation of dynamic web content, allowing server-side Java code to be embedded within HTML pages. This hybrid approach facilitates real-time data rendering from the server and supports robust backend functionality using Java classes and servlet logic. The use of Java and JSP ensures scalability and platform independence, making the project extensible for future enhancements.

Security measures are considered throughout the development process, with input validation implemented at both client and server levels to minimize vulnerabilities such as SQL injection and unauthorized access. The system architecture follows a modular and layered approach, separating concerns between presentation, logic, and data layers. This structure not only improves maintainability but also aligns with good software engineering practices.

In conclusion, this project demonstrates a comprehensive understanding of web development using a diverse yet coherent set of technologies. It has provided an opportunity to build a fully functional web application by integrating HTML, CSS, JavaScript, PHP, MySQL, Java, and JSP effectively. The project has strengthened practical skills in both frontend and backend development, fostering a deeper appreciation for how modern web applications are designed, implemented, and maintained. It also opens avenues for future work involving full-stack development, advanced security, and cloud-based deployment strategies.

INTRODUCTION

Web development has become a cornerstone of modern software engineering, enabling interactive, user-centric, and platform-independent solutions accessible from anywhere in the world. As organizations increasingly rely on digital interfaces for communication, data processing, and service delivery, the demand for robust and dynamic web applications continues to grow. This project focuses on the development of a full-fledged web-based system using a well-defined combination of frontend and backend technologies, offering both practical knowledge and technical proficiency in web application architecture.

The core objective of this project is to design and implement a web application that showcases the seamless integration of both the client-side and server-side components. It enables the user to interact with a structured interface and perform meaningful operations such as data entry, retrieval, display, and validation, all while ensuring smooth communication with the backend systems. The technologies used in this project are industry-standard, providing a strong foundation for real-world development environments.

Technologies Used

Frontend:

- **HTML (Hypertext Mark-up Language)**: Defines the basic structure and content of web pages.
- **CSS (Cascading Style Sheets)**: Styles the application with layouts, colours, fonts, and ensures responsiveness across devices.
- **JavaScript**: Adds interactivity and dynamic behaviour, such as client-side validation, content updates without page reloads, and DOM manipulation.

Backend:

- **PHP (Hypertext Pre-processor)**: Handles server-side scripting, form processing, session management, and basic logic handling.
- **MySQL**: Acts as the database system to store, retrieve, and manage structured data efficiently using SQL queries.
- **Java**: Powers backend components, enabling scalable logic execution and integration with web services.
- **JSP (JavaServer Pages)**: Used for dynamically generating HTML content on the server side and handling server-side rendering using embedded Java code.

PROBLEM STATEMENT

Title: Inventory Management System

Description:

In any organization, effectively managing inventory and delegating tasks to responsible personnel are essential operations. Manual methods often lead to miscommunication, redundancy, inventory misplacement, or task delays. These inefficiencies can impact productivity, accuracy, and transparency. There is a pressing need for a digital solution that automates user role management, inventory tracking, and task assignments — ensuring that every user, whether admin, employee, or worker, can perform their role seamlessly in a secure and structured environment.

The goal of this project is to design and implement a web-based **Inventory Management System** that enables different types of users (admin, employee, worker) to interact with the system according to their roles. It supports functions such as user registration, role-based login, inventory addition and monitoring, task assignments, and completion tracking. The system is built using **HTML, CSS, JavaScript** for the frontend and **PHP, MySQL, Java, JSP** for the backend, ensuring both interactivity and robustness.

To address these issues, there is a critical need for a **centralized, digital Inventory Management System** that integrates all user roles into a unified platform. Such a system should allow:

- **Admins** to approve users and oversee the entire system.
- **Employees** to manage inventory and assign operational tasks.
- **Workers** to receive tasks and report completion.

By providing role-based access, secure authentication, and structured data handling, this system ensures smooth collaboration and real-time visibility into the organization's operations. Built using **HTML, CSS, JavaScript** for the frontend and **PHP, MySQL, Java, JSP** on the backend, the solution supports a modern, efficient, and scalable workflow that can be adapted to small or medium-sized enterprises.

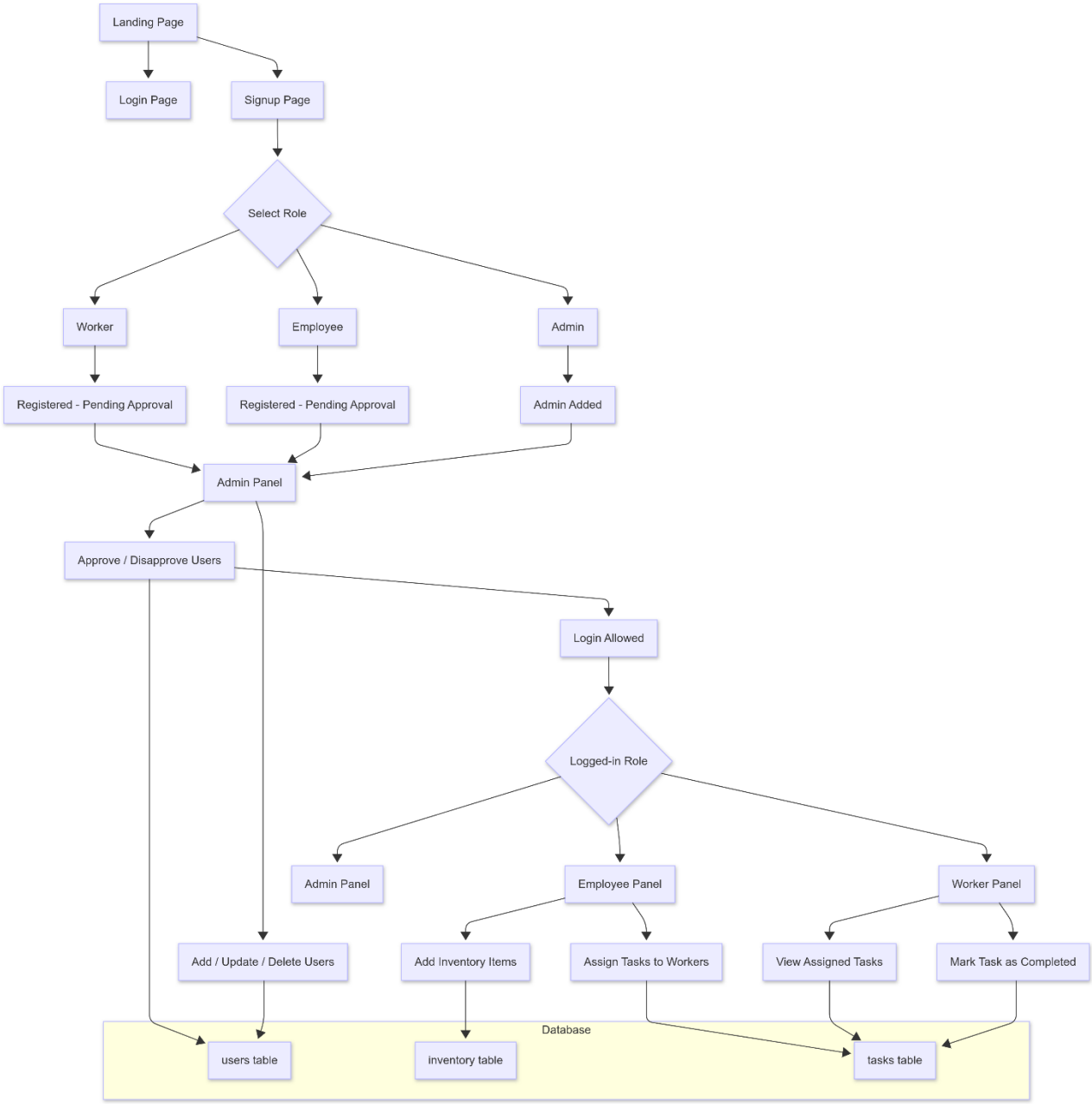
OVERVIEW

Flow of the Project

The project follows a logical and modular flow from landing to role-based operations. Below is a simplified summary of the system's workflow:

- **Landing Page**: Acts as the entry point of the application, offering navigation to login and signup pages.
- **User Registration**:
  - New users access the **Signup Page**.
  - During signup, users choose their role: **Admin**, **Employee**, or **Worker**.
  - Workers and employees are **registered with pending approval**.
  - Admin accounts are added directly or via admin backend.
- **Admin Dashboard**:
  - Admins can **approve or disapprove** pending users.
  - They can **add, update, or delete users** from the system.
  - Once approved, users can log in and access their respective panels.
- **Role-Based Panels**:
  - **Admin Panel**: Manages user permissions and oversees system activity.
  - **Employee Panel**:
    - Can **add inventory items** into the system.
    - Can **assign tasks** to workers.
  - **Worker Panel**:
    - Can **view assigned tasks**.
    - Can **mark tasks as completed**, updating the system in real time.
- **Database Design**:
  - users table: Stores all registered users and their roles/status.
  - inventory table: Stores inventory items and their details.
  - tasks table: Tracks task details, assignments, and completion statuses.

This structured flow ensures that every function is processed through proper authentication and data validation, ensuring integrity and role-based access control throughout the system.



Learning Outcomes

Through this project, the following practical and theoretical skills were developed:

- Designing structured and responsive web interfaces.
- Writing and maintaining clean client-side and server-side code.
- Implementing CRUD (Create, Read, Update, Delete) operations through PHP and JSP.
- Developing backend logic using Java and connecting it with frontend components.
- Managing database schemas, relationships, and queries using MySQL.
- Understanding the complete lifecycle of a web application—from design and development to deployment.

SYSTEM REQUIREMENTS

System Requirements for Inventory Management System Project

Below are the **hardware and software requirements** for developing and running the Inventory Management System effectively. This includes all necessary tools for frontend, backend, database, and server-side functionalities.

1. Hardware Requirements

- **Processor**: Intel Core i3 or higher
- **RAM**: Minimum 4 GB (8 GB recommended)
- **Hard Disk**: Minimum 500 GB (SSD preferred for faster processing)
- **Display**: Minimum resolution 1366x768
- **Network**: Internet connection for live testing and deployment

2. Software Requirements

Frontend Technologies

- **HTML5**: Mark-up language for webpage structure.
- **CSS3**: Styling the user interface.
- **JavaScript**: Interactivity and client-side validation.

Backend Technologies

- **PHP (≥ 7.x)**: Used for backend logic, user sessions, inventory and task management.
- **Java (≥ 8)**: Used along with JSP for the blog section of the website.
- **JSP (JavaServer Pages)**: For dynamic blog content display.
- **Apache Tomcat (≥ 9.x)**: Required to deploy and run the JSP and Java components.
- **Apache HTTP Server (≥ 2.4.x)**: To host and serve PHP applications.

Database

- **MySQL (≥ 5.7 or 8.x)**: Used to store and manage user data, inventory, and task records.
- **phpMyAdmin**: GUI tool for managing MySQL database conveniently.

3. Development & Deployment Tools

- **XAMPP** or **WAMP**: Bundled software package including Apache, MySQL, and PHP for local development.
- **NetBeans IDE (≥ 12.x)**: Preferred for Java and JSP development.
- **VS Code / Sublime Text**: For editing frontend and PHP files.
- **Browser (Chrome, Firefox)**: For testing and accessing the web application.

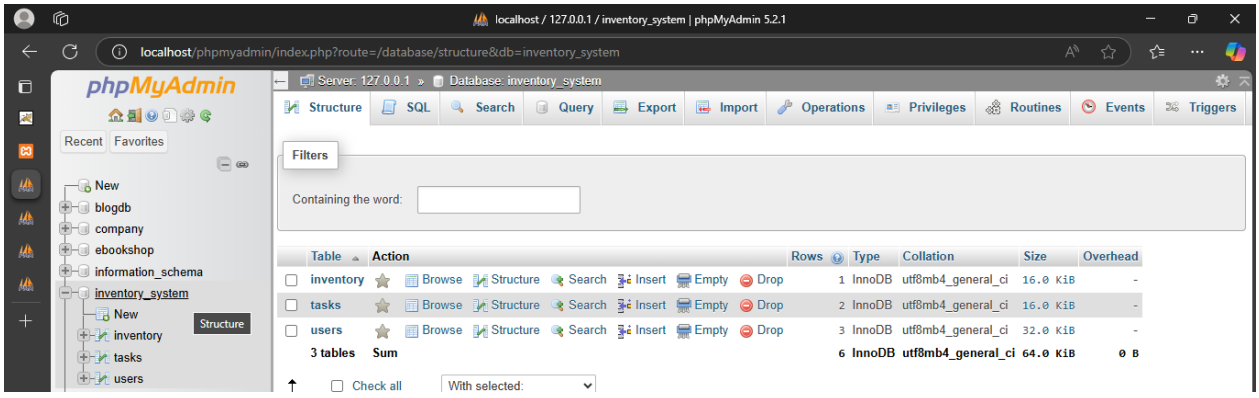
IMPLEMENTATION

Technology Distribution for the Inventory Management System

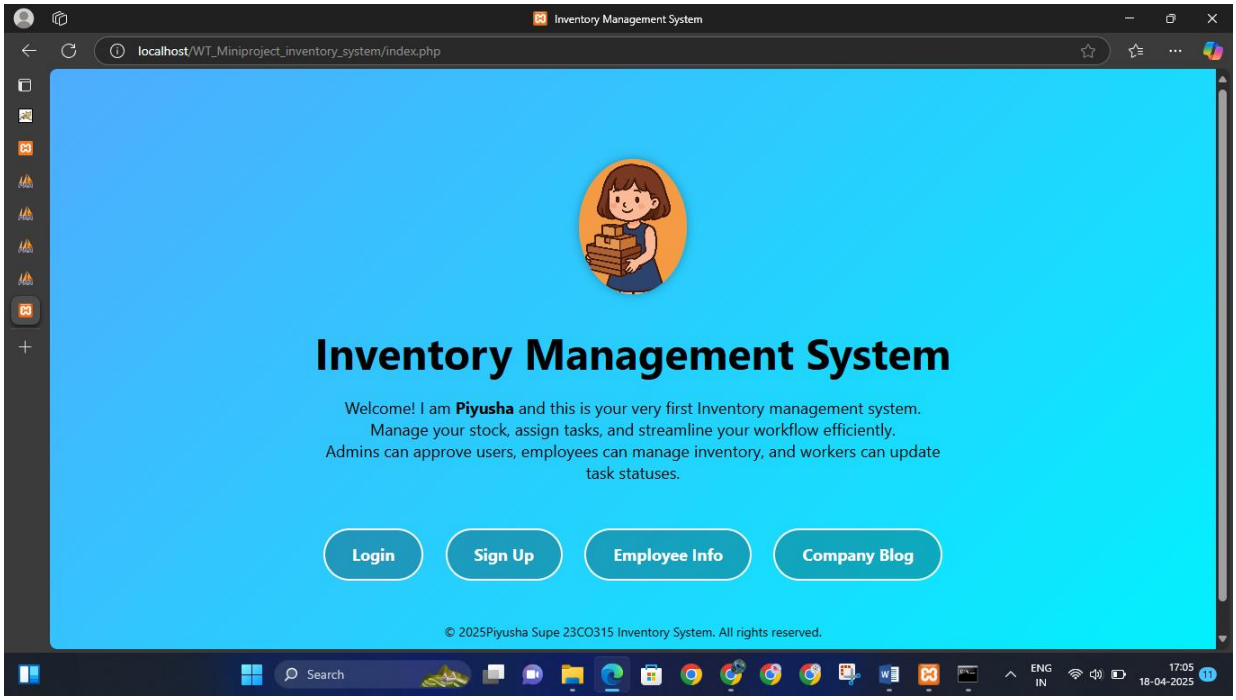
- **Frontend Technologies** (used throughout the entire application interface):
  - **HTML**: Provides the structural layout for all web pages.
  - **CSS**: Handles styling, responsiveness, and visual aesthetics.
  - **JavaScript**: Adds interactivity, form validation, and dynamic content behaviour.
- **Backend Technologies**:
  - **PHP**:
    - Core backend scripting language used for handling user authentication, form processing, inventory operations, and task management.
    - Interacts with the database to perform CRUD operations.
  - **MySQL**:
    - Relational database used to store and manage data related to users, inventory, and tasks.
    - Integrated with PHP for backend functionalities.
- **Blog Section (Company Blog Page Only)**:
  - **JSP (JavaServer Pages)**: Used for rendering the blog content dynamically.
  - **Java**: Implements server-side logic and interacts with JSP to manage blog data.
  - **MySQL**: Stores blog posts (title, content, date) and integrates with Java/JSP to retrieve and display blog content.

The implementation of the project is as follows:

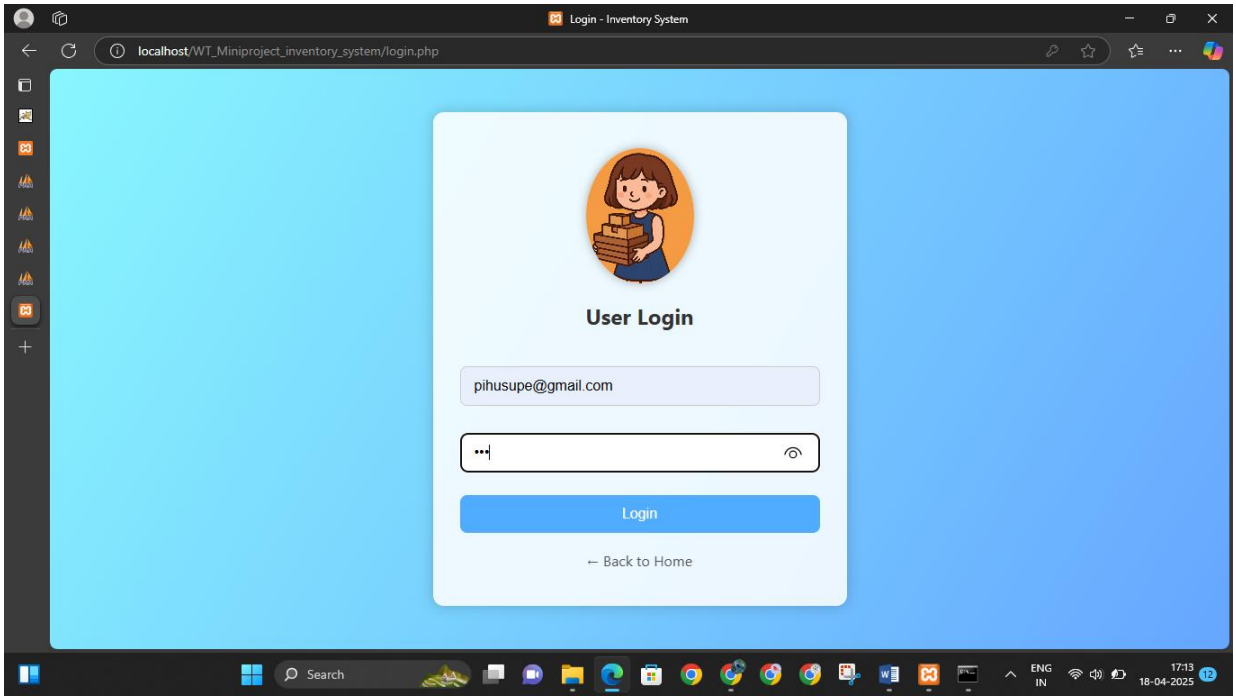
The database structure is



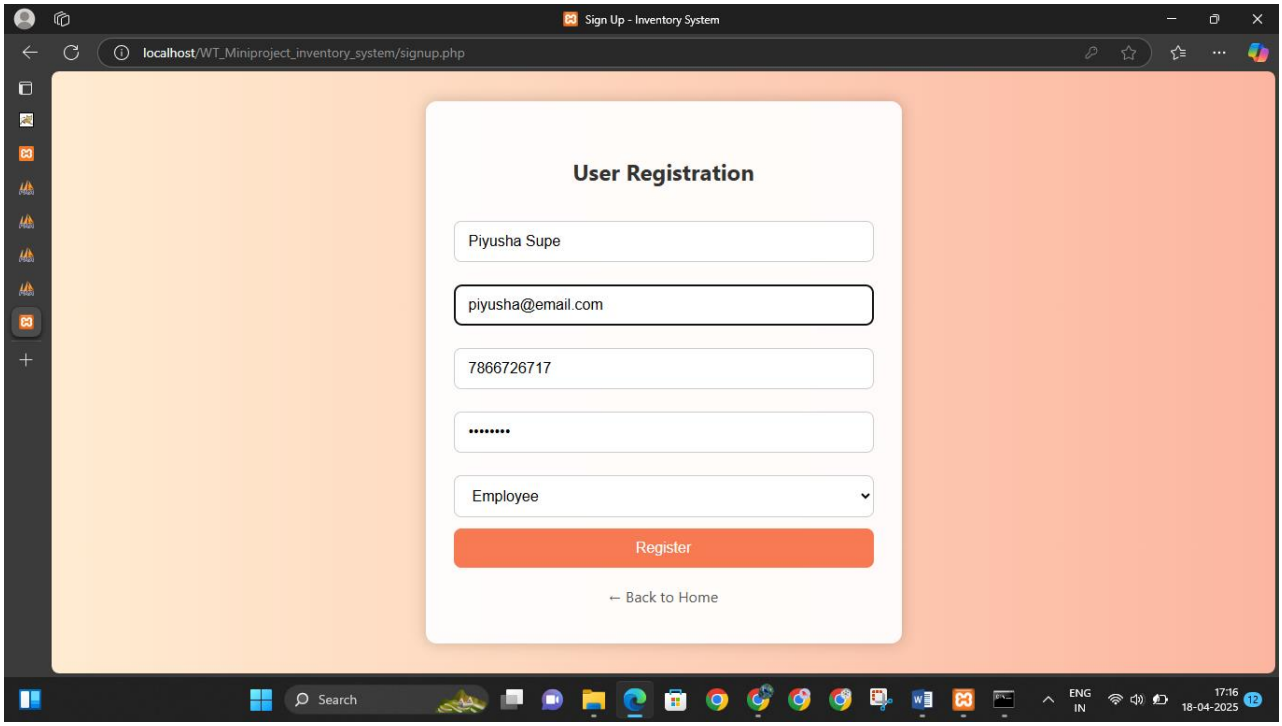
Landing page

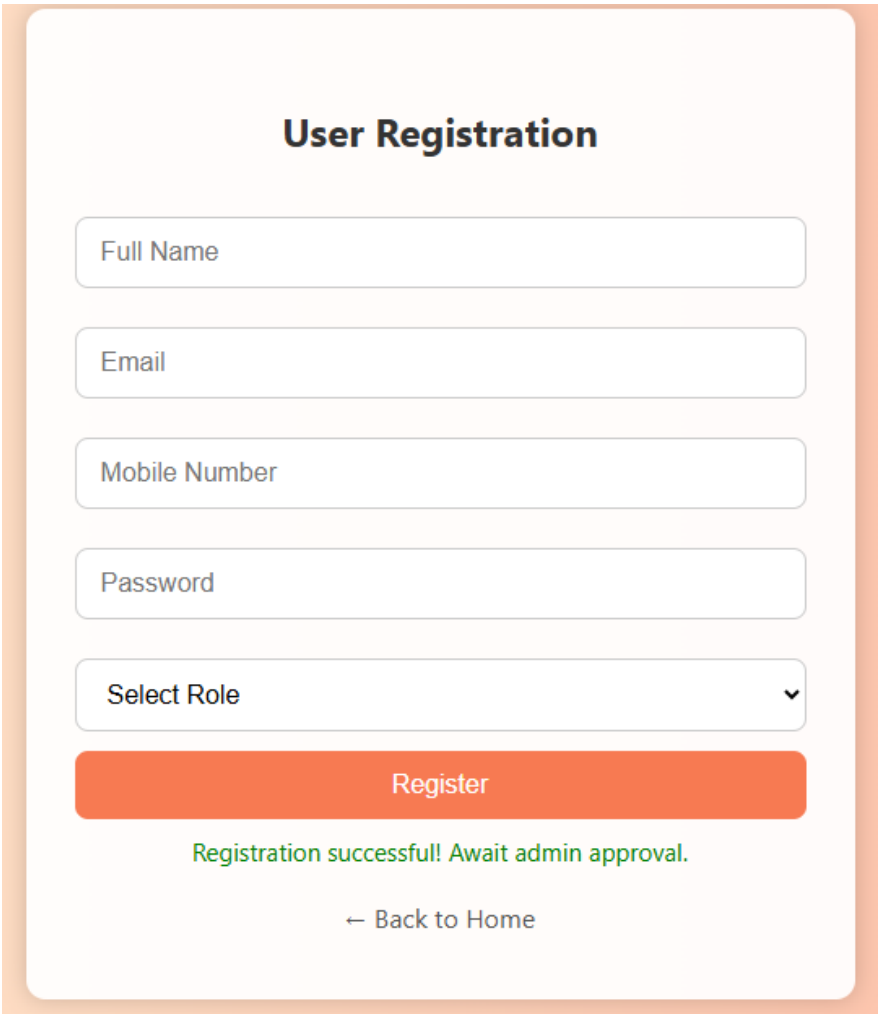


Login Page

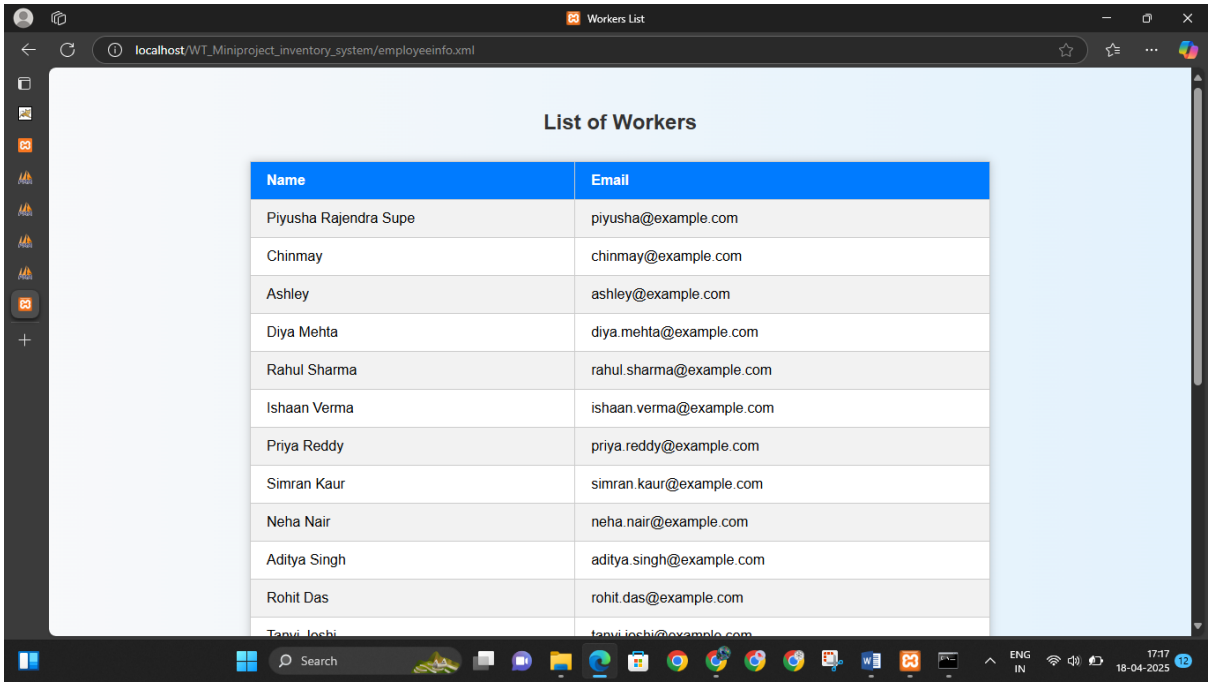


Sign up

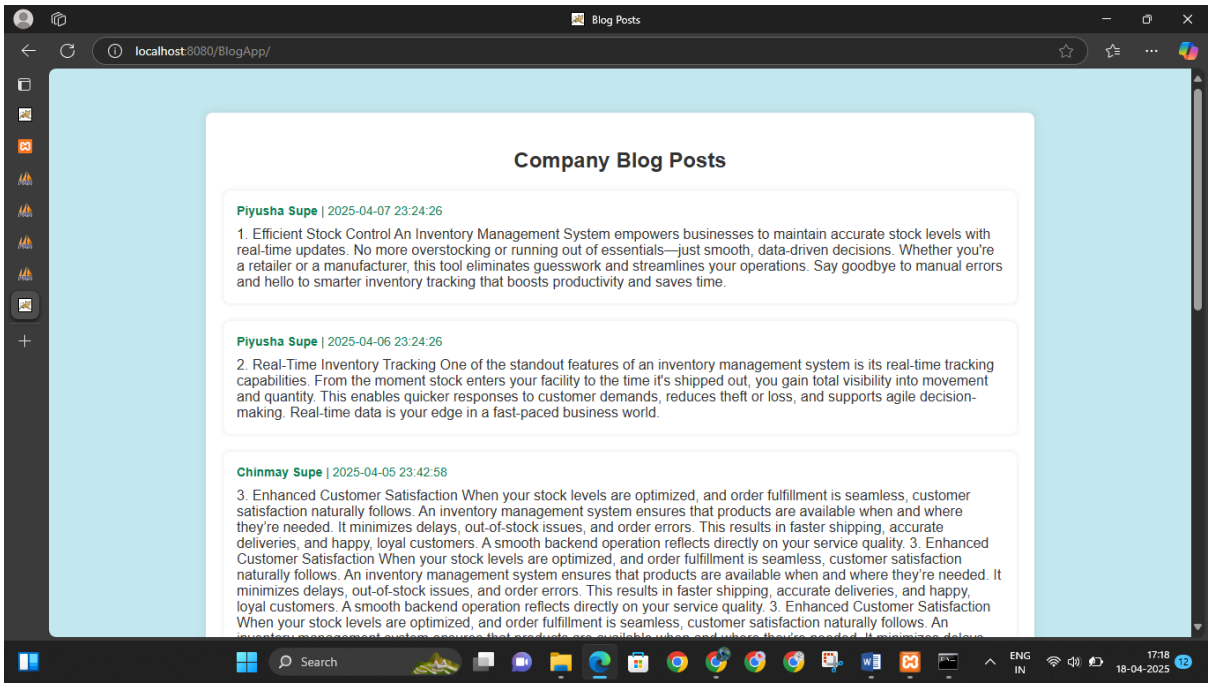

Sign Up - Inventory System
localhost/WT_Miniproject_inventory_system/signup.php
User Registration
Piyusha Supe
piyusha@email.com
7866726717
Employee
Register
← Back to Home


User Registration
Full Name
Email
Mobile Number
Password
Select Role
Register
Registration successful! Await admin approval.
← Back to Home

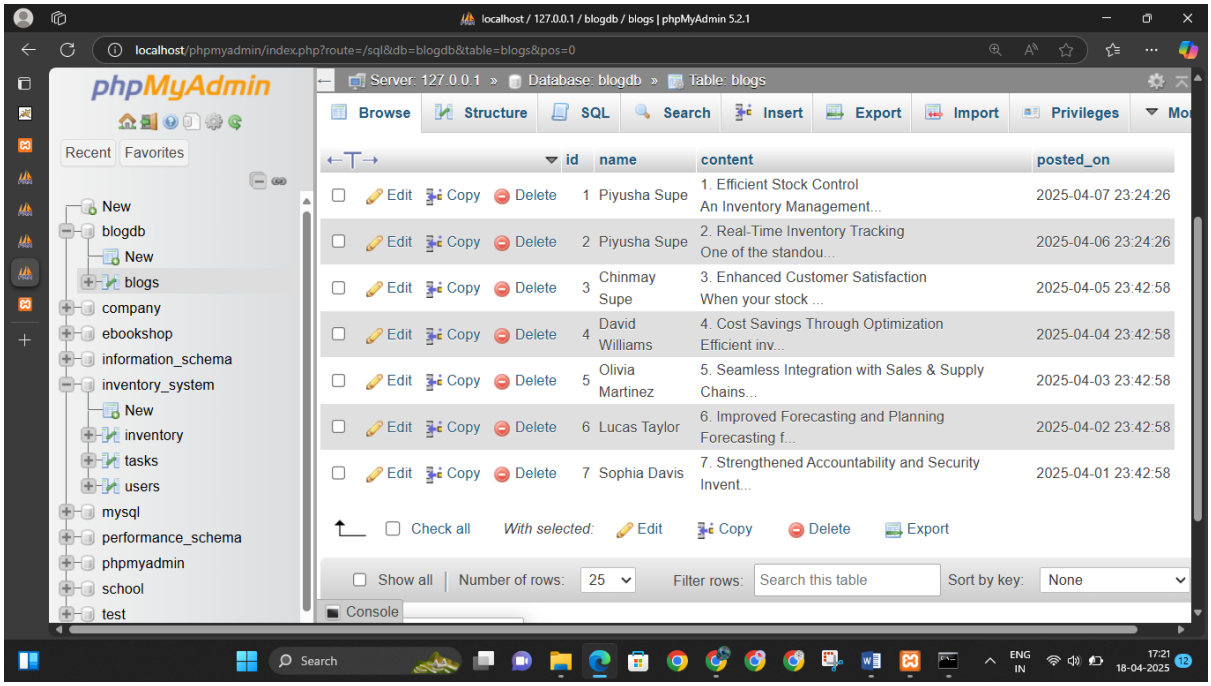
Use of XML for employee information representation and styled through XSL



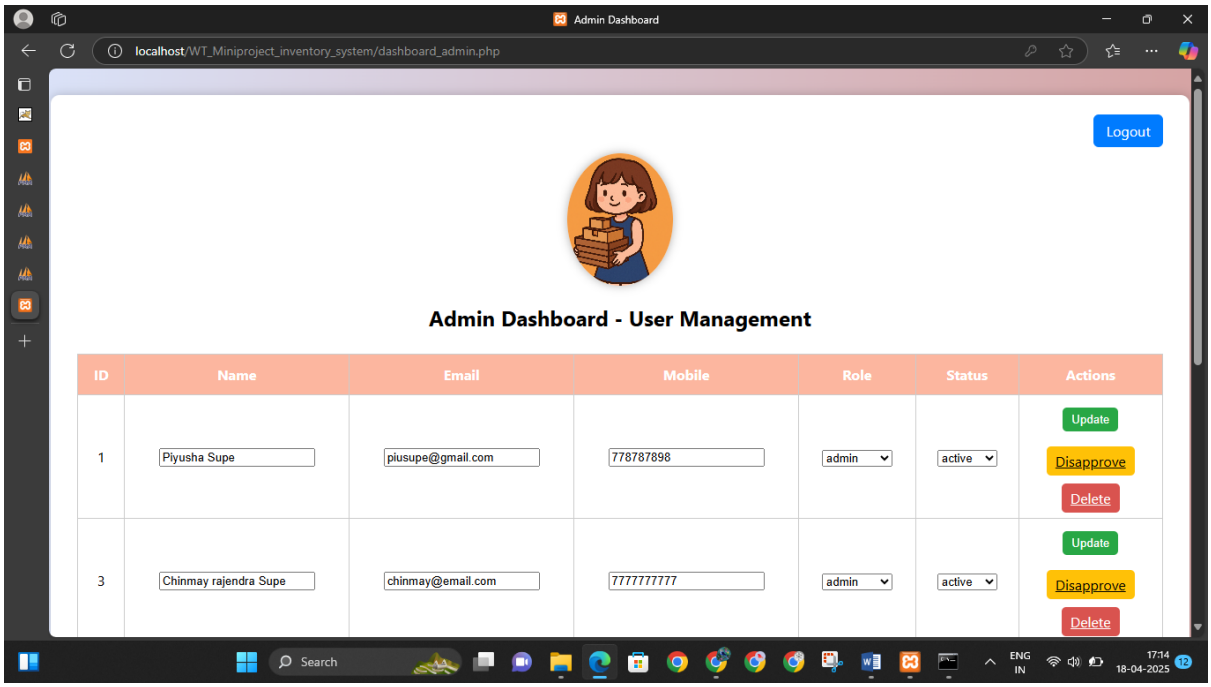
JSP for blog posts of company

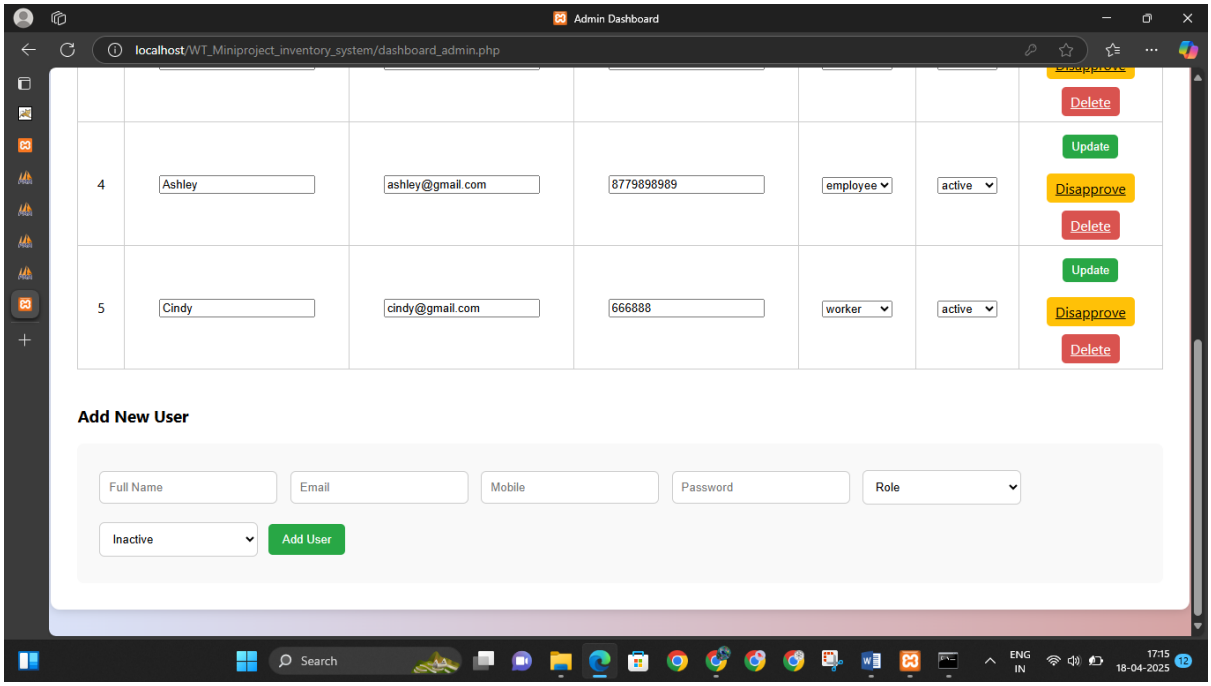


Database of Blogs (used by the JSP logic)

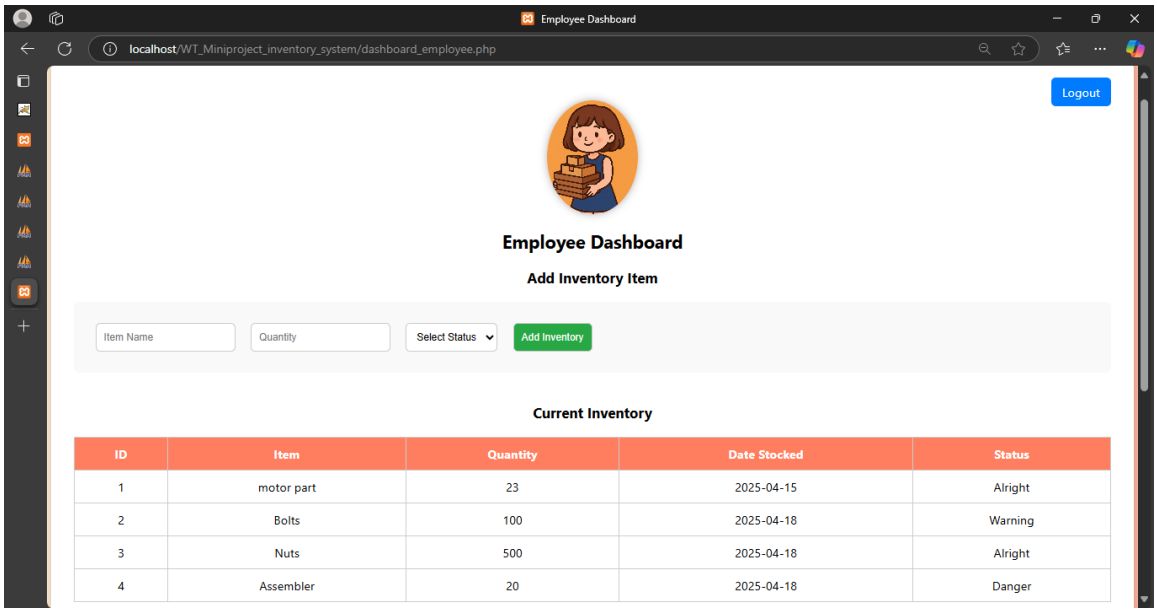


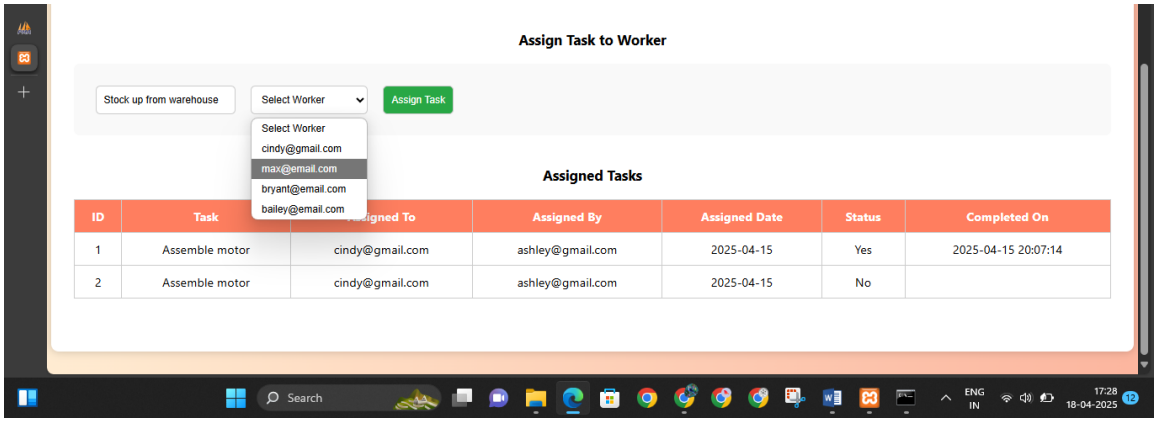
Admin dashboard





Employee dashboard

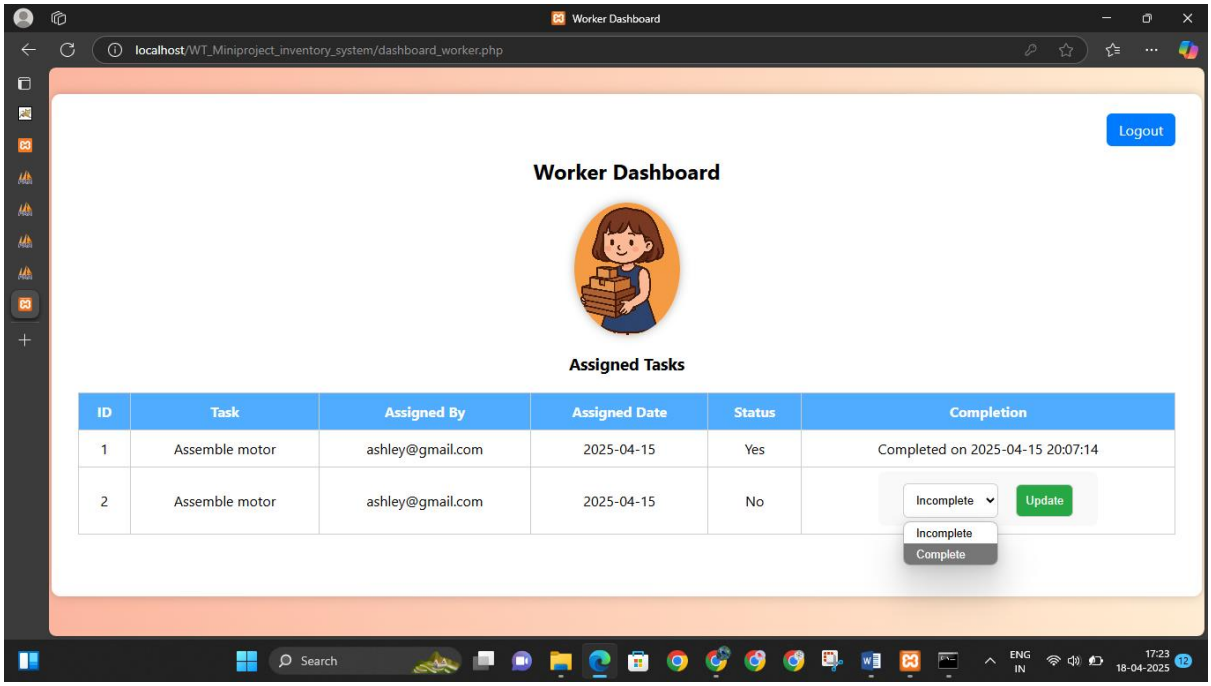


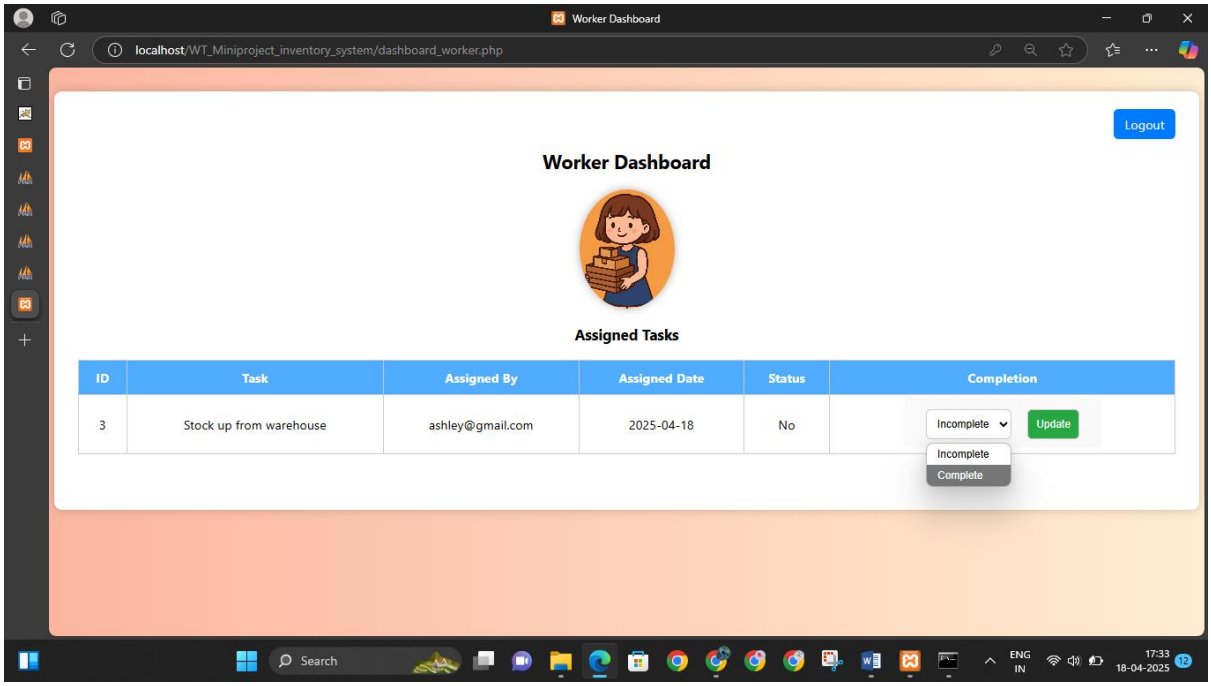


Assigned Tasks

ID	Task	Assigned To	Assigned By	Assigned Date	Status	Completed On
1	Assemble motor	cindy@gmail.com	ashley@gmail.com	2025-04-15	Yes	2025-04-15 20:07:14
2	Assemble motor	cindy@gmail.com	ashley@gmail.com	2025-04-15	No	
3	Stock up from warehouse	max@email.com	ashley@gmail.com	2025-04-18	No	

Worker dashboard





FUNCTIONALITY AND ADVANTAGE

Functionality of the Inventory Management System

The **Inventory Management System** offers a structured platform to manage users, inventory, and tasks with the following core features:

- **User Registration & Role Management**: Users register with a role (Admin, Employee, Worker) and access the system accordingly.
- **Admin Panel**: Admins manage user roles, approve registrations, and have full access to inventory and task data.
- **Employee Panel**: Employees can add and manage inventory, and assign tasks to workers.
- **Worker Panel**: Workers can view their assigned tasks, mark them as completed, and access inventory details.
- **Database Management**: MySQL database stores user, inventory, and task data, ensuring integrity and security.

Advantages of the Inventory Management System

- **Efficiency & Productivity**: Automates tasks like inventory tracking and task assignments, reducing manual effort and time.
- **Real-Time Monitoring**: Enables real-time updates for inventory levels and task status, allowing quicker decisions and actions.
- **Role-Based Access**: Ensures that each user has access only to the features relevant to their role, improving security and streamlining operations.
- **Centralized Data Management**: Consolidates all data into one system, simplifying management and reporting.
- **Scalability**: The system can be expanded to accommodate more users, inventory items, and functionalities as the business grows.
- **Enhanced Security**: Secure login, session management, and backend validation protect sensitive data from unauthorized access.
- **Cost & Time Efficiency**: Reduces errors and manual tracking, saving time and resources for better operational focus.
- **User-Friendly Interface**: A responsive frontend built with HTML, CSS, and JavaScript ensures a smooth user experience

CONCLUSION

The **Inventory Management System** developed for this project provides a comprehensive solution to the challenges businesses face in managing inventory, user roles, and task assignments. By integrating a web-based platform with role-based access, it ensures that each user—whether an admin, employee, or worker—has the appropriate tools and permissions to manage their responsibilities effectively.

With the use of **HTML, CSS, JavaScript**, the frontend of the system is designed to be user-friendly, responsive, and interactive. This allows for a seamless user experience, enabling quick navigation and real-time updates. On the backend, **PHP** and **MySQL** work together to securely handle user registrations, inventory management, and task assignments, ensuring data integrity and secure transactions.

The system's **role-based access control** helps maintain security, ensuring that users only access the features relevant to their roles. Admins have complete control, while employees can manage inventory and assign tasks, and workers can focus on completing tasks efficiently. This structure not only enhances security but also streamlines operations, reducing manual errors and improving overall workflow.

Additionally, the use of **JSP, Java**, and **MySQL** for the blog page section adds a dynamic feature, allowing the company to share important updates, news, or announcements with ease.

Overall, this **Inventory Management System** streamlines inventory tracking, task management, and user role handling, making it a powerful tool for organizations aiming to improve operational efficiency, reduce costs, and enhance real-time decision-making. The system's scalability ensures it can grow with the business, making it a long-term, adaptable solution.

- **The entire project demonstration is uploaded on the following YouTube link recorded by me:** https://youtu.be/YpT4w1UhkuA
- **The source code is available in my personal Git-hub repository:** https://github.com/PiyushaSupe/web_technology_miniproject/

REFERENCES

- https://www.geeksforgeeks.org/introduction-to-jsp/
- https://www.tutorialspoint.com/jsp/index.htm
- https://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html
- https://www.apachefriends.org/download.html
- https://www.php.net/
- https://www.php.net/manual/en/language.basic-syntax.php
- https://www.php.net/manual/en/refs.database.php
- https://www.php.net/manual/en/set.mysqlinfo.php
- https://www.php.net/manual/en/debugger.php
- https://www.microfocus.com/documentation/enterprise-developer/30pu12/ED-Eclipse/GUID-D075B3ED-BC49-4E73-A4BD-E3CBC0C1B696.html
- https://www.cs.virginia.edu/~up3f/cs4640/supplement/jsp-deployment.html

Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

Web Technology – Case Study

Title: Transform the blogging application from a loose collection of various resources (servlets, HTML documents, etc.) to an integrated web application that follows the MVC paradigm

1. Introduction

In the evolving world of web development, the need for scalable, maintainable, and well-structured applications is more pressing than ever. Traditional web applications often begin as fragmented systems—a mixture of HTML pages, embedded Java code in JSPs, and independent servlets. While this structure may suffice for smaller applications, it becomes increasingly difficult to manage as the application grows.

This case study presents a systematic transformation of such a loosely coupled blogging application into a robust, modular, and professionally structured web system. The architectural redesign adopts the **Model-View-Controller (MVC)** paradigm—an industry-standard design pattern that cleanly separates the data, logic, and presentation layers of a web application.

2. Background

The initial version of the blogging system consisted of:

- JSP pages embedded with JDBC code
- Direct SQL calls in the presentation layer
- HTML layout and Java logic mixed together
- No clear separation of responsibility

As the application was extended with new features like database updates, content filtering, and styling enhancements, the code became increasingly complex and hard to debug or maintain. This served as a strong motivation to refactor the application using MVC architecture, making it scalable, testable, and future-proof.

3. Objective of the Project

The main objective was to **refactor and modernize** the existing blogging application to:

- Follow the **MVC (Model-View-Controller)** architectural design pattern
- Promote **separation of concerns** between business logic, data access, and user interface
- Make the application **more maintainable, reusable, and extendable**
- Enhance **code readability** and **modularity**
- Create a strong foundation for future enhancements

4. Understanding MVC Architecture

The **MVC** pattern divides an application into three interconnected components:

Model:

Responsible for managing the data, logic, and rules of the application. In this project, the model includes:

- A **Blog.java** class representing blog entries
- A **Blog.java** class handling database operations (e.g., fetching blog posts)

View:

Handles the visual representation of the data. This includes the HTML and CSS used to present blog content to users. In this project, it's implemented in:

- **blog_view.jsp**, which dynamically displays the blog data passed from the controller

Controller:

Acts as an intermediary between Model and View. It processes incoming HTTP requests, calls the appropriate model methods, and forwards the results to the view. Here, it's implemented as:

- **BlogController.java** (a Servlet)

This architectural separation simplifies both development and maintenance, allowing individual parts of the application to evolve independently.

5. Implementation Overview

Model Layer

- **Blog.java**: A simple JavaBean class representing a blog post with attributes: name, content, and posted date.
- **Blog.java**: Manages database connection, SQL execution, and returns a list of Blog objects fetched from the database.

View Layer

- **blog_view.jsp**: A clean, structured JSP page displaying blog entries with dynamic content rendering. It uses HTML/CSS for styling and avoids embedding any SQL or JDBC logic.

Controller Layer

- **BlogController.java**: A servlet that handles HTTP GET requests, interacts with Blog to retrieve data, and forwards the blog list to the view using RequestDispatcher.

6. Benefits of the MVC-Based Transformation

Feature	Before (Loose Structure)	After (MVC Architecture)
Code Organization	Mixed Java, SQL, and HTML in JSP	Clear division between data, logic, view
Maintainability	Hard to trace or fix bugs	Easier to isolate and resolve issues
Reusability	Difficult to reuse code blocks	Modular components reused across the app
Extensibility	Adding new features is risky	New features added without disrupting others
Scalability	Poorly scalable structure	Scalable design for growing requirements

7. Technologies Used

- **Java EE (Servlets, JSP)**
- **JDBC** for backend database communication
- **MySQL** for persistent data storage
- **Apache Tomcat** for application deployment
- **HTML/CSS** for frontend styling and layout

8. Outcome

The refactored application is now a well-structured, professional-grade blogging platform that:

- Fetches and displays blog entries from a MySQL database
- Follows a true **MVC pattern** with clean component responsibilities
- Is easier to maintain, extend, and debug
- Serves as a strong example for learners and developers adopting best practices in web technology

9. Future Scope

This MVC foundation can easily support the addition of new features such as:

- User authentication and comment systems
- Blog categories and search functionality
- RESTful APIs for mobile integration
- Admin panel for post management
- AJAX for dynamic user experiences

10. Conclusion

This case study successfully demonstrates how a fragmented web application can be transformed into an organized, modular, and maintainable system using the **MVC architecture**. By adhering to the MVC design pattern, the blogging platform has not only improved in terms of performance and structure but also laid a strong groundwork for future development.

This transformation reflects the core principles of modern web technology—**modularity, clarity, scalability, and maintainability**—and fulfils the objective of evolving from a loose collection of servlets and JSPs to a unified, integrated web application.

To convert your existing JSP-based blog viewer into an **integrated MVC (Model-View-Controller)** web application, we need to **modularize the code into three clear layers**:

MVC Structure

1. Model (Blog.java & Blog.java)

Encapsulates database access and data objects.

Blog.java (Model class for a blog post):

```
public class Blog {
    private String name;
    private String content;
    private String postedOn;
    public Blog(String name, String content, String postedOn) {
        this.name = name;
        this.content = content;
        this.postedOn = postedOn;
    }

    public String getName() { return name; }
    public String getContent() { return content; }
    public String getPostedOn() { return postedOn; }
}
```

Blog.java (Handles DB operations):

```
import java.sql.*;
import java.util.*;
public class Blog {
    private String jdbcURL = "jdbc:mysql://localhost:3306/blogdb";
    private String jdbcUser = "root";
    private String jdbcPass = "";
    public List<Blog> getAllBlogs() {
        List<Blog> blogs = new ArrayList<>();
```

```
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(jdbcURL, jdbcUser, jdbcPass);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM blogs ORDER BY posted_on
DESC");
            while (rs.next()) {
                blogs.add(new Blog(
                    rs.getString("name"),
                    rs.getString("content"),
                    rs.getString("posted_on")
                ));
            }
            rs.close(); stmt.close(); conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return blogs;
    }
}
```

2. Controller (BlogController.java)

Handles incoming requests and passes data to the view.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.List;
public class BlogController extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        Blog blog = new Blog();
        List<Blog> blogList = blog.getAllBlogs();
        request.setAttribute("blogs", blogList);
```

```
        RequestDispatcher dispatcher = request.getRequestDispatcher("blog_view.jsp");
        dispatcher.forward(request, response);
    }
}
```

3. View (blog_view.jsp)

Only responsible for presenting data. No Java DB logic.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.*, model.Blog" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Blog Posts</title>
    <link rel="stylesheet" href="css/style.css" />
</head>
<body>
    <div class="container">
        <h2>Blog Posts</h2>
        <%
            List<Blog> blogs = (List<Blog>) request.getAttribute("blogs");
            for (Blog blog : blogs) {
        %>
            <div class="blog-entry">
                <div class="blog-meta"><strong><%= blog.getName() %></strong> | <%= blog.getPostedOn() %></div>
                <div class="blog-content"><%= blog.getContent() %></div>
            </div>
        <%
            }
        %>
    </div>
</body>
```

```
</html>
```

web.xml Configuration (Servlet Mapping)

```
<web-app>
    <servlet>
        <servlet-name>BlogController</servlet-name>
        <servlet-class>controller.BlogController</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>BlogController</servlet-name>
        <url-pattern>/blogs</url-pattern>
    </servlet-mapping>
</web-app>
```

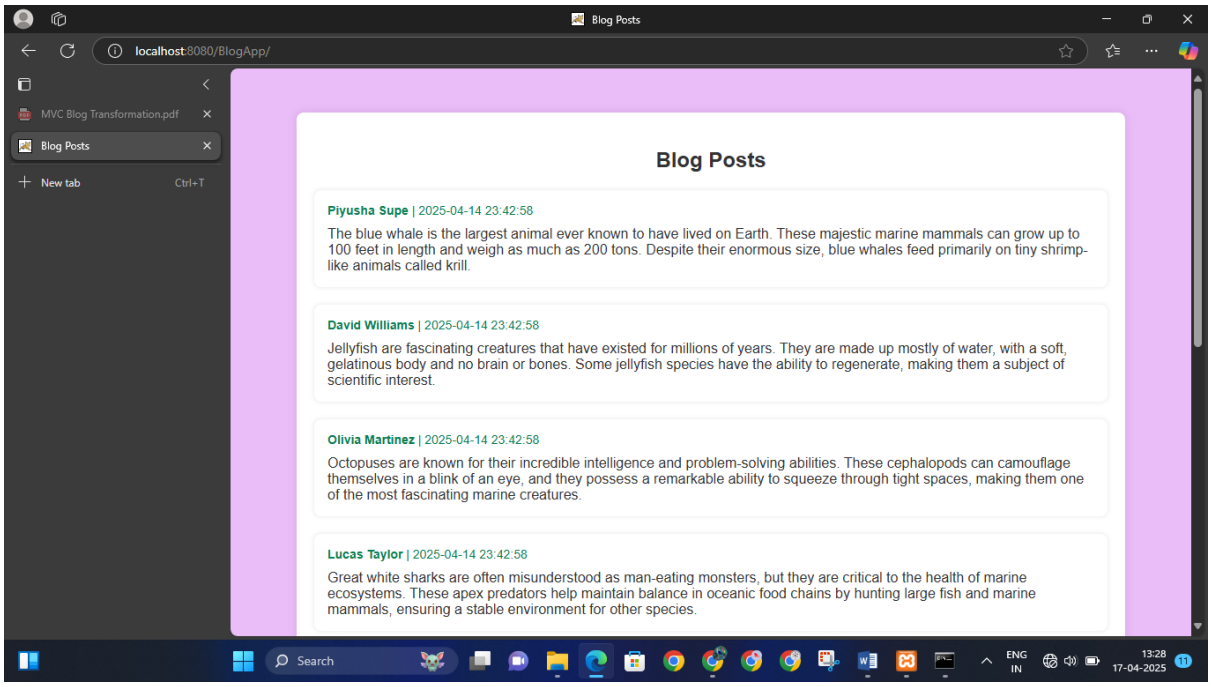
Style.css

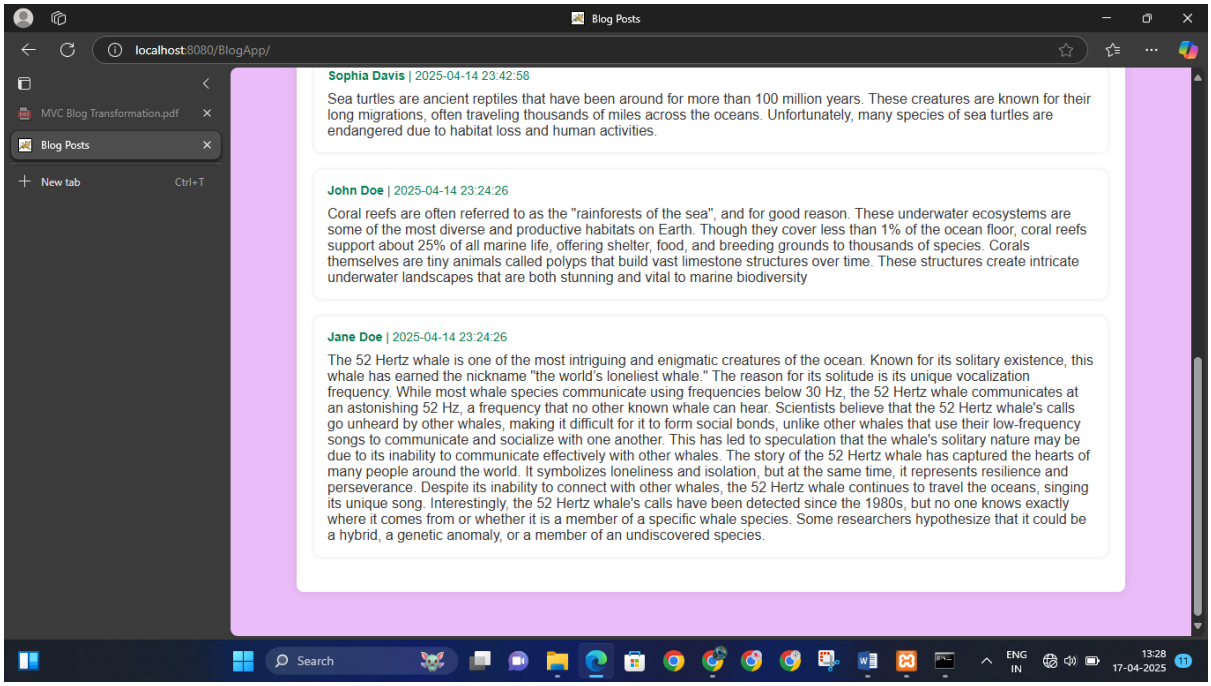
```
body {
    font-family: Arial, sans-serif;
    background-color: #eabcf8;
    margin: 0;
    padding: 0;
}

.container {
    max-width: 900px;
    margin: 50px auto;
    background-color: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h2 {
    text-align: center;
    color: #333;
```

```
}
.blog-entry {
    margin-bottom: 20px;
    padding: 15px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
.blog-meta {
    font-size: 14px;
    color: #0b7e52;
    margin-bottom: 10px;
}
.blog-content {
    font-size: 16px;
    color: #333;
}
```

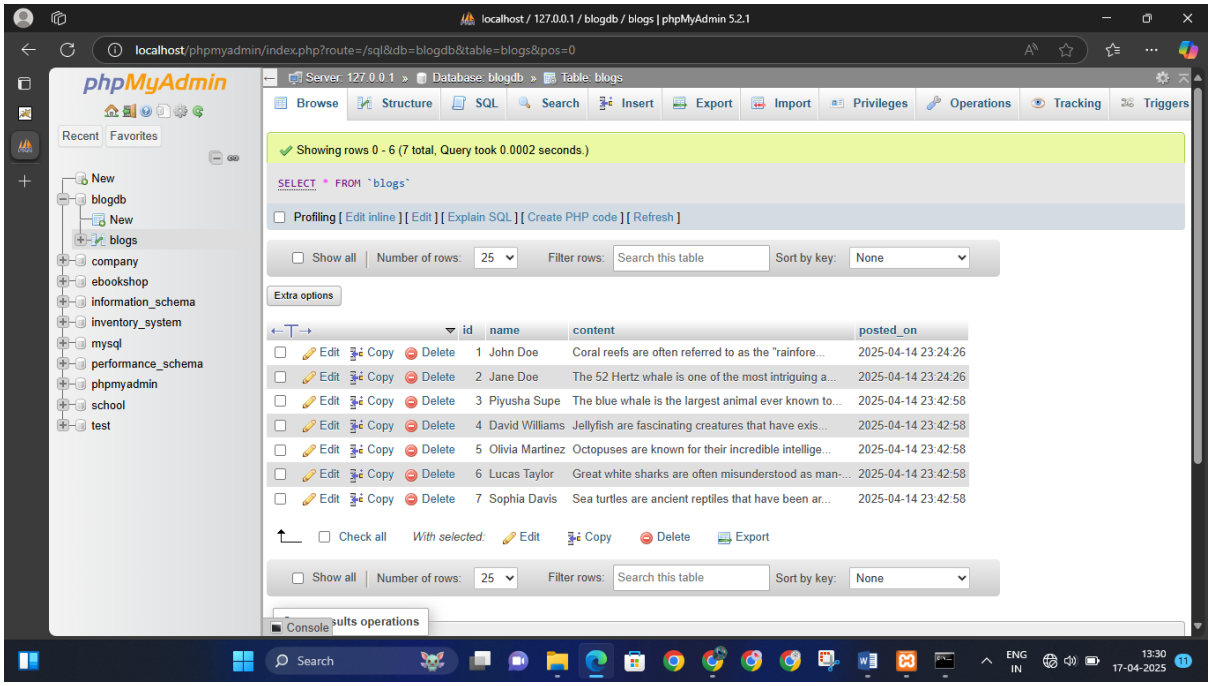
Access the App: URL:http://localhost:8080/BlogApp/

Output





Database Structure – database table



A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

T1902104336.

Name - Piyusha Rajendra Supe.
Roll No - 2320315. -(TE-B)

Assignment - 01.

* Web technology theory assignment:
Sr. 49. - Q. 46 to 50.

(46). Explain how session and cookies are used for session management in PHP.

→ • In php sessions and cookies are two essential mechanisms used for session management, which is the process of maintaining user specific data across multiple page requests.

1) How cookies in php work -

• A cookie is a small peice of data stored on the client side (browser) and sent to the server with every request.

• Used to store small amounts of data like user preferences, last visited time, etc.
• persistent across browser sessions (can be set to expire after a certain time).

• Set a cookie in php -

```
setcookie ("username", "JohnDoe", time() + (86400 * 7)
    , "7"); // Expire in 7 days.
```

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- Accessing a cookie -

```
echo $_cookie ["username"];
```

- Limitations of cookies -
  - → User can disable cookies in the browser.
  - → Not secure (stored on client side).
  - → Limited storage (usually upto 4kb).

2) Sessions in PHP -

- A session stores user information on the server side for the duration of user's interaction with the web application.
- When a session is started PHP generates a unique session ID.
- This session id is stored as a cookie on client side. (default - PHPSESSID).
- All session data is stored on the server and linked to session ID.

- How to start a session -

```
session_start();
```

- Store data in a session -

```
$_SESSION ["username"] = "John Doe";
```

- Access session data -

```
echo $_SESSION ["username"];
```

- Destroying a session -
  session_destroy().

(47) How does ASP.NET works? Also enlist the features of ASP.NET?

→ • ASP.NET is a web application framework developed by Microsoft that enables developers to build dynamic websites, web applications and web services. It runs on top of the .NET framework (or .NET Core / .NET 5+ for modern versions).

- Working of ASP.NET. -

1. Client request - user sends request via browser.
2. The request hits IIS (Internet Information Service) and IIS identifies the file type (.aspx, .cshtml). and forwards it to ASP.NET engine.
3. HTTP pipeline processes the request.
4. ASP.NET creates a page object or controller in MVC. based on request
5. Execution of code. - Server side logic executes (accessing database, processing user input).
6. It sends final HTML output back to client (browser) as a response. Browser renders it to HTML user.

4.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

- Features of ASP.NET -

- Compiled code - ASP.NET uses compiled code (C#, VB.NET), which runs faster than interpreted scripts.
- State management - Provides tools like session, view_state, cookies, for maintaining user data.
- Rich server controls - Built in server side controls simplify form and UI creation.
- Component based - pages can be divided into components.
- Built in security - Supports authentication, authorization, roles, and secure data handling.
- Event driven MVC forms.
- Asynchronous programming - methods for scalability and programming.
- SEO friendly URLs.
- Cross platform - runs on windows, linux, MacOS.
- Tool support - Integration with IDEs.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

(48). Write advantages of JSP over servelets, also explain life cycle of JSP.

→ • Advantages of JSP over servelet are –

1) JSP allows embedding Java code directly into HTML using tags like <% %>. Easier for UI development.

2) Promotes separation of presentation (HTML) and logic (Java).

3) Easier to update and maintain because its more readable and HTML like.

4) Provides implicit objects like request, response, session, etc.

5) JSPs are automatically compiled into Servlets by container (eg tomcat).

6) Designers can work on JSPs without needing deep java knowledge.

• LIFE CYCLE OF JSP –

*• Jsp life cycle is managed by web container (like apache tomcat) and similar to servelet life cycle.

* Phases are –

1. Translation phase –
   • The JSP file is translated into a servelet by the container.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

2. Compilation phase - A generated servelet is compiled into a class file.

3. Instantiation phase - An instance is compiled JSP is created.

4. Initialization phase - Called once when JSP is loaded. used to initialize resources like (eg. DB Connection). jsp init().

5. Request processing phase (-jspService()) -
   - Called each time the JSP is requested.
   - The actual logic of handling request and generating response happens here.

6. Destruction phase - jspdestroy() -
   - Called when JSP is taken out of service.
   - used to release resources.

(43). Write a code to create a PHP script to create a new database with 4 fields of your choice. and perform these operations. i) Insert ii) Delete iii) Update. iv) Display.

→ Program is as follows -

```
<? php
    $conn = new mysqli ("localhost", "root", " ", "simple-db");
```

```
$conn → query ("Create database if not exists simple-db");
$conn → query ("Create table if not exists students
        (id INT auto-increment primary-key,
        name varchar(50), email varchar(50),
        course varchar(50), age INT)");

$conn → query("INSERT into students values ('John',
        'john@email.com', 'PHP', 20)");
$conn → query("UPDATE students SET course = 'WebDev'
        where name = 'John'");
$conn → query ("DELETE from students where name =
        'Mark'");
$result = $conn → query ("SELECT * FROM Students");
echo "<table>";
while ($row = $result → fetch-assoc()) {
    echo "<tr><td>{$row['id']}</td><td>
        {$row['name']}</td><td>{$row['email']}
        </td><td>{$row['course']}</td><td>
        {$row['age']}</td></tr>";

echo "</table>";
$conn → close();
?>
```

Hence, this program covered all operations of the given statement.

→

8. Piyusha Supe
23C0315.

A.I.S.S.M.S
COLLEGE OF ENGINEERING
NO - 1, Kennedy Road, Near R.T.O Pune - 411001

(50). Explain life cycle of ASP.NET -

→ The phases are as follows -

1. Page request - Occurs when a user requests an .aspx page. It checks whether the page needs to be compiled or a cached version can be served.

2. Start - ASP.NET sets up basic page properties like -
• request • response • Is post back • UICULTURE.

3. Initialization - All controls on page are initialized. Each control gets a unique id. Settings like themes or master pages are applied.
Event - page_init().

4. Load - page_load() - Page is being loaded due to postback, control properties are restored. You can access form values and other data.

5. Post-back event handling - Result of post back, it calls event handlers like button click.

6. Rendering - It converts page controls into HTML and sent to browser.

7. Unload - Final clean up tasks, all objects are unloaded, after response completion.

Hence, this is how life cycle of ASP.NET is carried out.