**UNIVERSITY SCHOOL OF**

**INFORMATION**

**COMMUNICATION AND**

**TECHNOLOGY**



**WEB DEVELOPMENT - II LAB**

**SUBJECT CODE: ITE-465P**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**SUBMITTED TO:                                    SUBMITTED BY:**

Prof. Sanjay Kumar Malik Pranav Jain

00516403221

B.Tech (CSE-7th Semester)

2021-25

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **NAME OF PRACTICAL** | **DATE** | **SIGN** |
| 1 | Using various HTML tags for creating different web pages. |  |  |
| 2 | Using various Form tags for interactivity, authentication, date validation, etc. |  |  |
| 3 | Using Semantic HTML tags /tags associated with interactivity. |  |  |
| 4 | Using DHTML tags in concern to client server application. |  |  |
| 5 | Using JavaScript/CSS for dynamic web pages. |  |  |
| 6 | Utilize HTML/CSS and JavaScript frameworks (ReactJS, NextJS) to construct dynamic user interfaces. |  |  |
| 7 | Create various databases using SQL/MongoDB/or other to show interactivity . |  |  |
| 8 | Perform CRUD operations using React JS as frontend technology and Node JS as backend technology showing database interactivity using any database. |  |  |
| 9 | Develop robust back-end systems using Node.js. |  |  |
| 10 | Showing database interactivity using  PHP/Python/or any current technology used in web industry. |  |  |
| 11 | Any current web industry relevant example of database usage and interactivity using any suitable backend technology. |  |  |

**PRACTICAL 1**

**Aim:** Using various HTML tags for creating different web pages.

**CODE:**

1. **Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Home - My Website</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Welcome to My Website</h1>

<nav>

<ul>

<li><a href="index.html">Home</a></li>

<li><a href="about.html">About</a></li>

<li><a href="contact.html">Contact</a></li>

</ul>

</nav>

</header>

<main>

<section>

<h2>Introduction</h2>

<p>This website is a showcase of HTML and CSS skills, featuring various types of content and layout options.</p>

</section>

<section>

<h2>Featured Section</h2>

<p>Explore the other pages to learn more about me, view the gallery, or get in touch through the contact page.</p>

</section>

</main>

<footer>

<p>&copy; 2024 My Website</p>

</footer>

</body>

</html>

## About.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>About - My Website</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>About Me</h1>

<nav>

<ul>

<li><a href="index.html">Home</a></li>

<li><a href="about.html">About</a></li>

<li><a href="contact.html">Contact</a></li>

</ul>

</nav>

</header>

<main>

<section>

<h2>Who I Am</h2>

<p>This section provides a brief background about me, my interests, and my professional experience.</p>

</section>

<section>

<h3>Hobbies</h3>

<ul>

<li>Photography</li>

<li>Coding</li>

<li>Traveling</li>

</ul>

</section>

</main>

<footer>

<p>&copy; 2024 My Website</p>

</footer>

</body>

</html>

## Contact.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Contact - My Website</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Contact Me</h1>

<nav>

<ul>

<li><a href="index.html">Home</a></li>

<li><a href="about.html">About</a></li>

<li><a href="contact.html">Contact</a></li>

</ul>

</nav>

</header>

<main>

<section>

<h2>Get in Touch</h2>

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

<div>

<label for="name">Name:</label>

<input type="text" id="name" name="name" required>

</div><br>

<div>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

</div><br>

<div>

<label for="message">Message:</label>

<textarea id="message" name="message" required></textarea>

</div><br>

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

</form>

</section>

</main>

<footer>

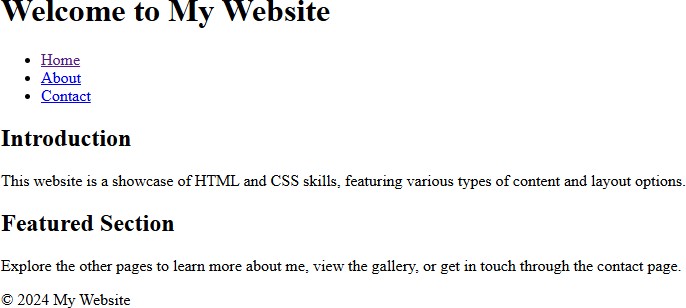
<p>&copy; 2024 My Website</p>

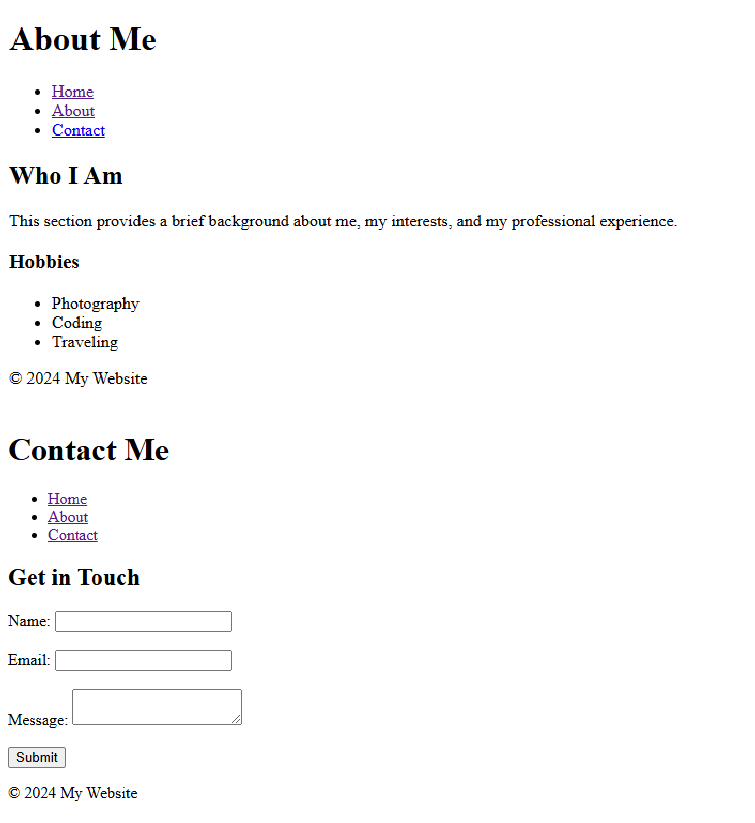
</footer>

</body>

</html>

**OUTPUT:**





# PRACTICAL 2

**Aim:** Using various Form tags for interactivity, authentication, date validation, etc.

**CODE:**

1. **Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Interactive Form Example</title>

<link rel="stylesheet" href="style.css">

<script src="script.js" defer></script>

</head>

<body>

<h1>Interactive Form with Validation</h1>

<!-- Form with various input fields -->

<form id="registrationForm" onsubmit="return validateForm()">

<!-- Name Input -->

<label for="name">Full Name:</label>

<input type="text" id="name" name="name" required>

<!-- Email Input -->

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<!-- Password Input -->

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<!-- Date of Birth Input -->

<label for="dob">Date of Birth:</label>

<input type="date" id="dob" name="dob" required>

<!-- Gender Radio Buttons -->

<label id="gender" for="gender">Gender:</label>

<div class="gender-options">

<label><input type="radio" name="gender" value="Male" required> Male</label>

<label><input type="radio" name="gender" value="Female"> Female</label>

<label><input type="radio" name="gender" value="Other"> Other</label>

</div>

<!-- Country Selection -->

<label for="country">Country:</label>

<select id="country" name="country" required>

<option value="">Select your country</option>

<option value="USA">USA</option>

<option value="Canada">Canada</option>

<option value="UK">UK</option>

<option value="Australia">Australia</option>

</select>

<!-- Terms and Conditions Checkbox -->

<label id="checkbox">

<input type="checkbox" id="terms" name="terms" required> I agree to the terms and conditions

</label>

<!-- Submit Button -->

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

</form>

<p id="message"></p>

</body>

</html>

## Style.css

body {

font-family: Arial, sans-serif; display: flex;

flex-direction: column; align-items: center;

}

h1 {

color: #333;

}

form { width: 30%;

margin: 20px; padding: 30px; border: 1px solid #ccc; border-radius: 8px;

}

label {

display: block; margin-top: 20px; color: #555;

}

input, select, button { width: 95%; padding: 8px; margin-top: 5px; font-size: 16px;

}

.gender-options label { display: inline-block; width: 25%;

}

.gender-options { display: flex;

align-items: center;

}

.gender-options input{ width: 15%;

}

#checkbox input{ width: 5%;

}

button {

background-color: #4CAF50; color: white;

border: none; cursor: pointer; margin-top: 15px;

}

button:hover {

background-color: #45a049;

}

#message { color: red;

font-weight: bold;

}

## Script.js

function validateForm() {

const name = document.getElementById("name").value; const email = document.getElementById("email").value;

const password = document.getElementById("password").value; const dob = document.getElementById("dob").value;

const terms = document.getElementById("terms").checked; const message = document.getElementById("message");

// Clear previous message message.textContent = "";

// Password length validation if (password.length < 6) {

message.textContent = "Password must be at least 6 characters long.";

return false;

}

// Date of birth validation (age 18+) const dobDate = new Date(dob); const today = new Date();

const age = today.getFullYear() - dobDate.getFullYear();

const monthDifference = today.getMonth() - dobDate.getMonth();

if (monthDifference < 0 || (monthDifference === 0 && today.getDate() < dobDate.getDate())) { age--;

}

if (age < 18) {

message.textContent = "You must be at least 18 years old to register."; return false;

}

// Terms and conditions validation if (!terms) {

message.textContent = "You must agree to the terms and conditions."; return false;

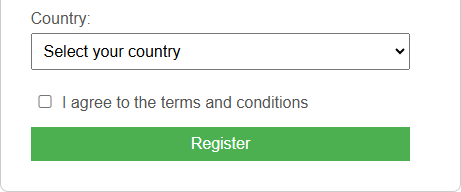
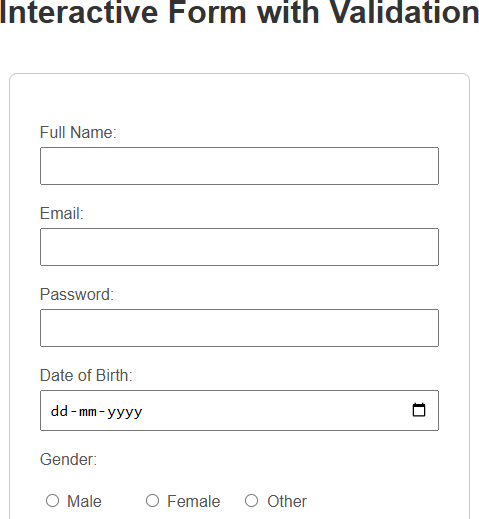
}

// Display success message if all validations pass message.textContent = "Registration successful!"; message.style.color = "green";

return false; // Prevent form submission for demonstration

}

**OUTPUT:**



# PRACTICAL 3

**Aim:** Using Semantic HTML tags associated with interactivity.

**CODE:**

1. **Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Semantic HTML Webpage</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<!-- Header -->

<header>

<h1>My Semantic HTML Webpage</h1>

<p>Welcome to a page structured with semantic HTML elements for clarity and accessibility.</p>

</header>

<!-- Navigation -->

<nav>

<ul>

<li><a href="#home">Home</a></li>

<li><a href="#about">About</a></li>

<li><a href="#articles">Articles</a></li>

<li><a href="#contact">Contact</a></li>

</ul>

</nav>

<!-- Main Content -->

<main>

<!-- About Section -->

<section id="about">

<h2>About Us</h2>

<p>This section provides information about the purpose and mission of our website.</p>

</section>

<!-- Articles Section -->

<section id="articles">

<h2>Featured Articles</h2>

<!-- Article 1 -->

<article>

<h3>Understanding Semantic HTML</h3>

<p>Semantic HTML improves accessibility and SEO by clearly defining sections of a webpage.</p>

</article>

<!-- Article 2 -->

<article>

<h3>Why Semantic Tags Matter</h3>

<p>Using semantic tags helps screen readers and search engines interpret your content more effectively.</p>

</article>

</section>

<!-- Sidebar -->

<aside>

<h3>Related Links</h3>

<ul>

<li><a href="https://developer.mozilla.org/en-US/docs/Web/HTML">HTML Documentation</a></li>

<li><a href="https://[www.w3schools.com/html/](http://www.w3schools.com/html/)">HTML Tutorials</a></li>

</ul>

</aside>

</main>

<!-- Footer -->

<footer>

<p>&copy; 2024 My Semantic HTML Webpage. All rights reserved.</p>

<p>Contact us at: <a href="<mailto:info@example.com>">[info@example.com](mailto:info@example.com)</a></p>

</footer>

</body>

</html>

## Style.css

body {

font-family: Arial, sans-serif; margin: 0;

padding: 0;

}

header {

background-color: #4CAF50; color: white;

padding: 20px; text-align: center;

}

nav ul {

list-style-type: none; background-color: #333; padding: 20px;

text-align: center; margin: 0px;

}

nav ul li { display: inline;

margin: 0 45px;

}

nav a {

color: white;

text-decoration: none;

}

main {

display: flex; padding: 20px;

}

section{ padding: 10px;

}

#about{

width: 25vw;

}

#articles{ width: 48vw;

}

aside {

flex: 1;

background-color: #f4f4f4; padding: 20px;

margin-left: 20px;

}

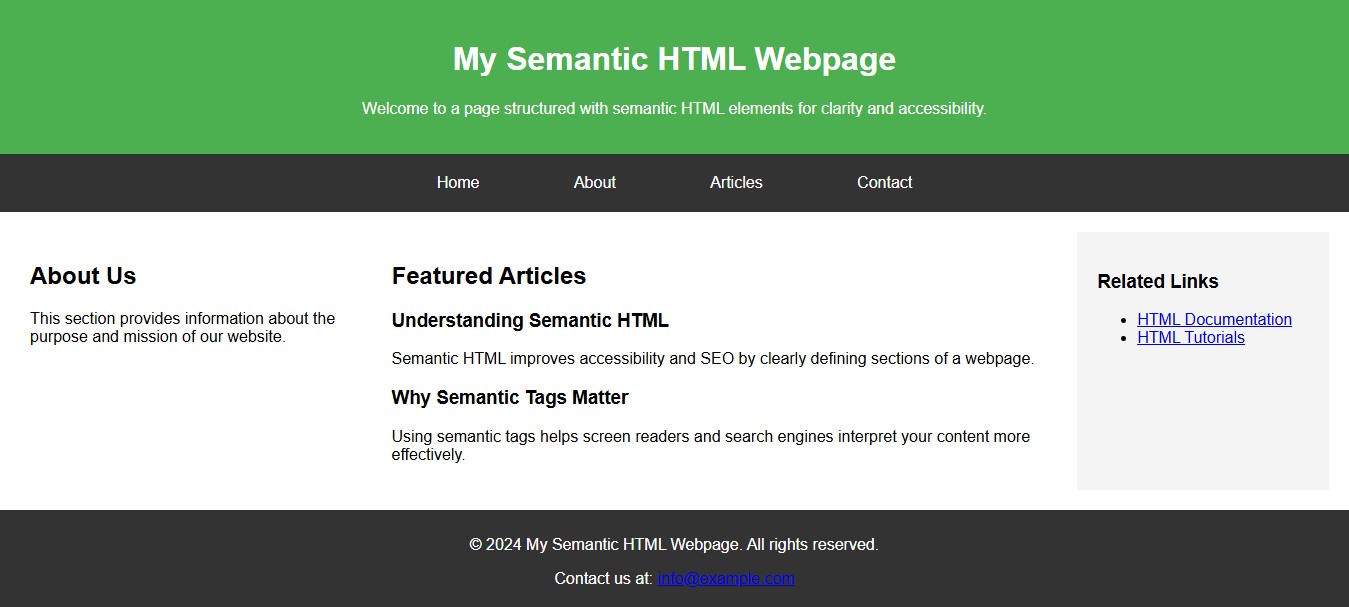
footer {

background-color: #333; color: white;

text-align: center; padding: 10px;

}

**OUTPUT:**



# PRACTICAL 4

**Aim:** Using DHTML tags in concern to client server application.

**CODE:**

1. **Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>DHTML Example Page</title>

<link rel="stylesheet" href="style.css">

<script src="script.js" defer></script>

</head>

<body>

<!-- Header Section -->

<header>

<h1 id="dynamicText">Welcome to My DHTML Page!</h1>

<button onclick="changeTextColor()">Click to Change Color</button>

</header>

<!-- Main Content Section -->

<main>

<section>

<h2>About DHTML</h2>

<p>DHTML is a combination of HTML, CSS, and JavaScript that allows web pages to be more interactive and dynamic.</p>

<button onclick="toggleContent()">Show More</button>

<div id="extraContent" style="display: none;">

<p>This additional content is revealed using JavaScript. DHTML allows for changes to HTML content, styles, and elements based on user interactions without reloading the page.</p>

</div>

</section>

<section>

<h2>Interactive Elements</h2>

<p>Move your mouse over the box to see it change color!</p>

<div id="colorBox" onmouseover="changeBoxColor()" onmouseout="resetBoxColor()"> Hover over me!

</div>

</section>

</main>

<!-- Footer Section -->

<footer>

<p>&copy; 2024 DHTML Example Page</p>

</footer>

</body>

</html>

## Style.css

body {

font-family: Arial, sans-serif; margin: 0;

padding: 0; display: flex;

flex-direction: column; align-items: center; text-align: center;

}

header {

background-color:rgb(252, 252, 137); color: darkgreen;

padding: 20px; width: 100%;

}

button {

margin-top: 10px; padding: 10px 20px; cursor: pointer;

}

main {

display: flex;

flex-direction: row;

}

section{

padding: 35px 35px; max-width: 55%;

}

#colorBox { width: 150px; height: 150px;

background-color: #f4f4f4; margin: 20px auto; border: 2px solid #333; display: flex;

align-items: center; justify-content: center; color: #333;

font-weight: bold;

}

footer {

background-color: #333; color: white;

padding: 10px; width: 100%;

}

## Script.js

// Change the color of the text in the header on button click function changeTextColor() {

const headerText = document.getElementById("dynamicText");

headerText.style.color = headerText.style.color === "red" ? "blue" : (headerText.style.color === "blue" ? "darkgreen" : "red");

}

// Toggle visibility of extra content in the "About DHTML" section function toggleContent() {

const extraContent = document.getElementById("extraContent"); if (extraContent.style.display === "none") {

extraContent.style.display = "block";

} else {

extraContent.style.display = "none";

}

}

// Change box color on mouseover function changeBoxColor() {

document.getElementById("colorBox").style.backgroundColor = "crimson";

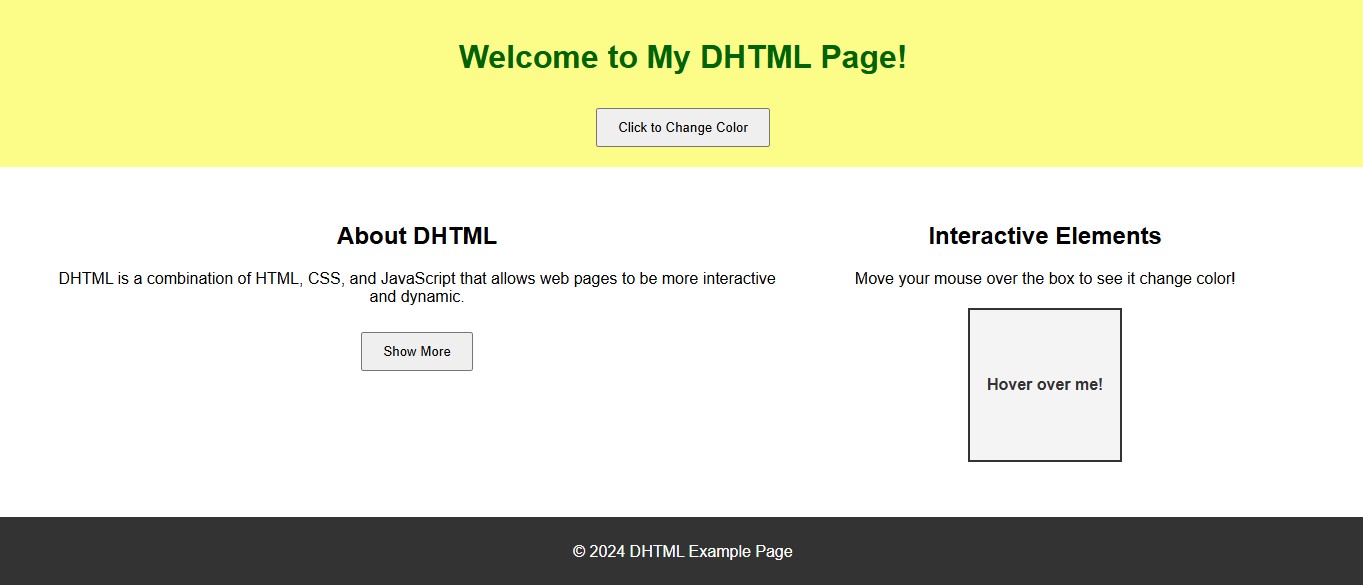
}

// Reset box color on mouseout function resetBoxColor() {

document.getElementById("colorBox").style.backgroundColor = "#f4f4f4";

}

**OUTPUT:**



# PRACTICAL 5

**Aim:** Using JavaScript for dynamic web pages.

**CODE:**

1. **Index.html**

<!DOCTYPE html>

<html>

<head>

<title>Todo List</title>

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

</head>

<body>

<div class="todo-container">

<h1>Todo List</h1>

<input type="text" id="taskInput" placeholder="Enter new task">

<button id="addTaskButton">Add Task</button>

<ul id="taskList"></ul>

</div>

<script src="script.js"></script>

</body>

</html>

## Style.css

.todo-container { width: 400px; margin: 0 auto; padding: 20px;

border: 1px solid #ccc; border-radius: 5px; background-color: #f5f5f5;

}

#taskInput { width: 70%; padding: 5px;

}

#addTaskButton { padding: 6px 10px;

}

ul {

list-style-type: none; padding: 0;

}

li {

display: flex;

justify-content: space-between; padding: 5px;

border-bottom: 1px solid #ccc;

}

button {

margin-left: 10px;

}

## Script.js

const taskInput = document.getElementById("taskInput");

const addTaskButton = document.getElementById("addTaskButton"); const taskList = document.getElementById("taskList");

let tasks = JSON.parse(localStorage.getItem("tasks")) || []; displayTasks();

addTaskButton.addEventListener("click", addTask); function addTask() {

const taskText = taskInput.value.trim(); if (taskText) {

tasks.push(taskText); saveTasks(); displayTasks(); taskInput.value = "";

}

else{

alert("Please enter a task.");

}

}

function displayTasks() { taskList.innerHTML = ""; tasks.forEach((task, index) => {

const li = document.createElement("li"); li.innerHTML = `

<span>${task}</span>

<button onclick="editTask(${index})">Edit</button>

<button onclick="deleteTask(${index})">Delete</button>

});

}

`; taskList.appendChild(li);

function editTask(index) {

const newTaskText = prompt("Edit task:", tasks[index]); if (newTaskText !== null) {

tasks[index] = newTaskText.trim();

saveTasks(); displayTasks();

}

}

function deleteTask(index) {

if(confirm("Do you want to delete this task?")){ tasks.splice(index, 1);

saveTasks(); displayTasks();

}

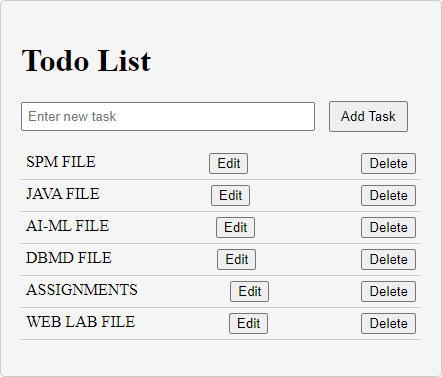
}

function saveTasks() {

localStorage.setItem("tasks", JSON.stringify(tasks));

}

**OUTPUT:**



# PRACTICAL 6

**Aim:** Utilize HTML/CSS and JavaScript frameworks (ReactJS, NextJS) to construct dynamic user interfaces.

**CODE:**

1. **App.js**

import React from 'react'; import './App.css';

import Calculator from './Calculator';

function App() { return (

<div className="App">

<h1><center>Simple Calculator</center></h1>

<Calculator />

</div>

);

}

export default App;

## Calculator.js

import React, { useState } from 'react';

const Calculator = () => {

const [input, setInput] = useState(''); const [result, setResult] = useState('');

const handleButtonClick = (value) => {

// Prevent multiple decimal points in the same number

if (value === '.' && input.split(/[\+\-\\*\/]/).pop().includes('.')) { return;

}

setInput((prevInput) => prevInput + value);

};

const calculateResult = () => { try {

// Handle percentage calculation

let processedInput = input.replace(/(\d+)%/g, (match, number) => number / 100); setResult(eval(processedInput));

} catch (error) { setResult('Error');

}

};

const clearInput = () => {

setInput('');

setResult('');

};

const deleteLastCharacter = () => { setInput((prevInput) => prevInput.slice(0, -1));

};

return (

<div className="calculator">

<div className="display">

<div className="input">{input}</div>

<div className="result">{result}</div>

</div>

<div className="buttons">

<button onClick={() => handleButtonClick('1')}>1</button>

<button onClick={() => handleButtonClick('2')}>2</button>

<button onClick={() => handleButtonClick('3')}>3</button>

<button onClick={() => handleButtonClick('+')}>+</button>

<button onClick={() => handleButtonClick('4')}>4</button>

<button onClick={() => handleButtonClick('5')}>5</button>

<button onClick={() => handleButtonClick('6')}>6</button>

<button onClick={() => handleButtonClick('-')}>-</button>

<button onClick={() => handleButtonClick('7')}>7</button>

<button onClick={() => handleButtonClick('8')}>8</button>

<button onClick={() => handleButtonClick('9')}>9</button>

<button onClick={() => handleButtonClick('\*')}>\*</button>

<button onClick={() => handleButtonClick('00')}>00</button>

<button onClick={() => handleButtonClick('0')}>0</button>

<button onClick={() => handleButtonClick('.')}>.</button>

<button onClick={() => handleButtonClick('/')}>/</button>

<button onClick={deleteLastCharacter}>DEL</button>

<button onClick={clearInput}>C</button>

<button onClick={calculateResult}>=</button>

<button onClick={() => handleButtonClick('%')}>%</button>

</div>

</div>

);

};

export default Calculator;

## App.css

.calculator { width: 320px; margin: 0 auto; padding: 20px;

border: 1px solid #ccc; border-radius: 10px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

}

.display {

border: 1px solid #ddd; padding: 10px; background-color: #f9f9f9; border-radius: 5px;

text-align: right;

}

.input {

font-size: 24px;

}

.result {

font-size: 32px; color: #333;

}

.buttons { display: grid;

grid-template-columns: repeat(4, 1fr); gap: 10px;

margin-top: 20px;

}

button { padding: 20px; font-size: 18px; border: none;

border-radius: 5px; background-color: #007BFF; color: white;

cursor: pointer;

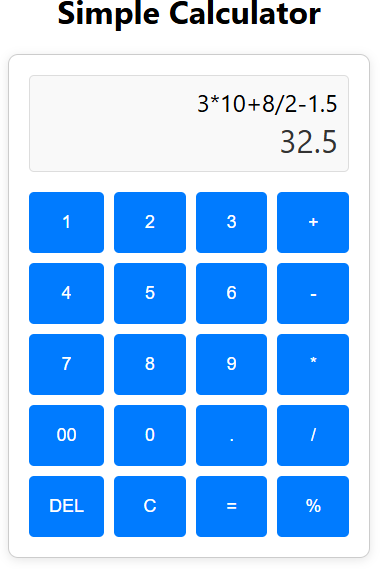
}

button:hover {

background-color: #0056b3;

}

**OUTPUT:**



# PRACTICAL 7

**Aim:** Create various databases and show interactivity

**Code:**

CREATE DATABASE db1;

CREATE TABLE db1.users (

id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(100),

email VARCHAR(100)

);

**UserServlet.java**

package com.example;

import jakarta.servlet.ServletException;   
import jakarta.servlet.annotation.WebServlet;   
import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;   
import jakarta.servlet.http.HttpServletResponse;  
 import java.io.IOException;

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

import java.sql.PreparedStatement;  
 import java.sql.ResultSet;

@WebServlet("/users")

public class UserServlet extends HttpServlet {

private static final long serialVersionUID = 1L; private static final String URL =

"jdbc:mysql://localhost:3306/db1";

private static final String USER = "root"; private static final String PASSWORD = "root";

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

{

Connection connection = null; ResultSet resultSet = null;  
 try {

Class.forName("com.mysql.cj.jdbc.Driver");

connection = DriverManager.getConnection(URL, USER,

PASSWORD);

String sql = "SELECT \* FROM users";   
PreparedStatement preparedStatement =

connection.prepareStatement(sql);

resultSet = preparedStatement.executeQuery();  
 request.setAttribute("userList", resultSet);

request.getRequestDispatcher("users.jsp").forward(request, response);

} catch (Exception e) { e.printStackTrace();

} finally {

try {

if (resultSet != null) resultSet.close(); if (connection != null) connection.close();

} catch (Exception e) { e.printStackTrace();

}

}

}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

String name = request.getParameter("name"); String email = request.getParameter("email");

String action = request.getParameter("action"); Connection connection = null;

PreparedStatement preparedStatement = null; try {

Class.forName("com.mysql.cj.jdbc.Driver");

connection = DriverManager.getConnection(URL, USER,

PASSWORD);

if ("add".equals(action)) {

String sql = "INSERT INTO users (name, email)

VALUES (?, ?)";

preparedStatement =

connection.prepareStatement(sql);

preparedStatement.setString(1, name); preparedStatement.setString(2, email); preparedStatement.executeUpdate();

} else if ("update".equals(action)) {

String userId = request.getParameter("userId"); String sql = "UPDATE users SET name = ?, email = ?

WHERE id = ?";

preparedStatement =

connection.prepareStatement(sql);

preparedStatement.setString(1, name); preparedStatement.setString(2, email); preparedStatement.setInt(3,

Integer.parseInt(userId));

preparedStatement.executeUpdate();

}

response.sendRedirect("users");

} catch (Exception e) { e.printStackTrace();

} finally {

try {

if (preparedStatement != null) preparedStatement.close();

if (connection != null) connection.close();

} catch (Exception e) { e.printStackTrace();

}

}

}

}

**users.jsp**

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

<%@ page import="java.sql.ResultSet" %>

<%@ page import="java.sql.SQLException" %>

<%@ page import="java.sql.Connection,java.sql.DriverManager, java.sql.PreparedStatement" %>

<!DOCTYPE html>

<html>

<head>

<title>User List</title>

</head>

<body>

<h1>User List</h1>

<%

ResultSet userList = (ResultSet)

request.getAttribute("userList"); if (userList != null) {

out.println("<table border='1'>");

out.println("<tr><th>ID</th><th>Name</th><th>Email</th><th>Actions

</th></tr>");

while (userList.next()) {

int userId = userList.getInt("id");

String userName = userList.getString("name"); String userEmail = userList.getString("email");

out.println("<tr>");

out.println("<td>" + userId + "</td>"); out.println("<td>" + userName + "</td>"); out.println("<td>" + userEmail + "</td>"); out.println("<td>");

out.println("<a href='?action=edit&userId=" + userId + "'>Edit</a>");

out.println(" <a href='users?action=delete&userId=" + userId + "' onclick='return confirm(\"Are you sure you want to delete this user?\");'>Delete</a>");

out.println("</td>"); out.println("</tr>");

}

out.println("</table>");

} else {

out.println("<p>No users found.</p>");

}

%>

<h2>Add User</h2>

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

<input type="hidden" name="action" value="add"> Name: <input type="text" name="name" required><br>

Email: <input type="email" name="email" required><br>

<input type="submit" value="Add User">

</form>

<%

// Check for edit action

String action = request.getParameter("action"); if ("edit".equals(action)) {

int userId = Integer.parseInt(request.getParameter("userId"));

Connection connection = null; PreparedStatement preparedStatement = null; ResultSet rs = null;

try {

// Fetch user details for editing Class.forName("com.mysql.cj.jdbc.Driver"); connection =

DriverManager.getConnection("jdbc:mysql://localhost:3306/db1", "root", "root");

String sql = "SELECT \* FROM users WHERE id = ?";

preparedStatement = connection.prepareStatement(sql);

preparedStatement.setInt(1, userId); rs = preparedStatement.executeQuery();

%>

value="update">

if (rs.next()) {

String userName = rs.getString("name"); String userEmail = rs.getString("email");

<h2>Edit User</h2>

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

<input type="hidden" name="action"

<input type="hidden" name="userId"

value="<%= userId %>">

Name: <input type="text" name="name"

value="<%= userName %>" required><br>

Email: <input type="email" name="email" value="<%= userEmail %>" required><br>

<input type="submit" value="Update User">

</form>

<%

}

} catch (Exception e) {

e.printStackTrace(); // For debugging purposes

} finally {

try {

if (rs != null) rs.close();

if (preparedStatement != null) preparedStatement.close();

if (connection != null) connection.close();

} catch (Exception e) {

e.printStackTrace(); // For debugging purposes

}

}

} else if ("delete".equals(action)) {

// Handle delete action

String userId = request.getParameter("userId"); Connection connection = null;

PreparedStatement preparedStatement = null;

try {

// Establish database connection Class.forName("com.mysql.cj.jdbc.Driver"); connection =

DriverManager.getConnection("jdbc:mysql://localhost:3306/db1", "root", "root");

String sql = "DELETE FROM users WHERE id = ?"; preparedStatement =

connection.prepareStatement(sql);

preparedStatement.setInt(1, Integer.parseInt(userId));

preparedStatement.executeUpdate();

// Redirect back to the users page after deletion response.sendRedirect("users");

} catch (Exception e) {

e.printStackTrace(); // For debugging purposes

} finally {

try {

if (preparedStatement != null) preparedStatement.close();

if (connection != null) connection.close();

} catch (Exception e) {

e.printStackTrace(); // For debugging purposes

}

}

}

%>

</body>

</html>

**OUTPUT:**

# 

# 

# PRACTICAL 8

**Aim:** Perform CRUD operations using React JS as frontend technology and Node JS as backend technology showing database interactivity using any database.

**CODE:**

**FRONTEND**

## EmployeeForm.js

import React from "react";

import { useForm } from "react-hook-form"; import { useNavigate } from "react-router-dom";

const EmployeeForm = () => {

const { register, handleSubmit } = useForm(); const navigate = useNavigate();

const createEmployee = async (data) => { const savedUserResponse = await fetch(

`${process.env.REACT\_APP\_BASE\_URL}/createUser`,

{

method: "POST", headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({ ...data }),

}

);

console.log("FORM RESPONSE. ", savedUserResponse);

navigate("/")

};

return (

<div>

<form onSubmit={handleSubmit(createEmployee)} className="mt-8">

<div className="space-y-5">

<div>

<label htmlFor="name"

className="text-base font-medium text-gray-900 dark:text-gray-200"

>

{" "}

Employee Name{" "}

</label>

<div className="mt-2.5">

<input

className="flex h-10 w-full rounded-md border border-gray-300 bg-transparent py-2 px-3 text-sm placeholder:text-gray-400 focus:outline-none focus:ring-1 focus:ring-gray-400 focus:ring-

offset-1 disabled:cursor-not-allowed disabled:opacity-50 dark:border-gray-700 dark:text-gray-50 dark:focus:ring-gray-400 dark:focus:ring-offset-gray-900"

type="text"

placeholder="Enter You Full Name"

{...register("name")}

></input>

</div>

</div>

<div>

<label htmlFor="email"

className="text-base font-medium text-gray-900 dark:text-gray-200"

>

{" "}

Employee Email Id{" "}

</label>

<div className="mt-2.5">

<input

className="flex h-10 w-full rounded-md border border-gray-300 bg-transparent py-2 px-3 text-sm placeholder:text-gray-400 focus:outline-none focus:ring-1 focus:ring-gray-400 focus:ring- offset-1 disabled:cursor-not-allowed disabled:opacity-50 dark:border-gray-700 dark:text-gray-50 dark:focus:ring-gray-400 dark:focus:ring-offset-gray-900"

type="email" placeholder="Enter Your Email"

{...register("email")}

></input>

</div>

</div>

<div>

<label htmlFor="title"

className="text-base font-medium text-gray-900 dark:text-gray-200"

>

{" "}

Employee Title{" "}

</label>

<div className="mt-2.5">

<input

className="flex h-10 w-full rounded-md border border-gray-300 bg-transparent py-2 px-3 text-sm placeholder:text-gray-400 focus:outline-none focus:ring-1 focus:ring-gray-400 focus:ring- offset-1 disabled:cursor-not-allowed disabled:opacity-50 dark:border-gray-700 dark:text-gray-50 dark:focus:ring-gray-400 dark:focus:ring-offset-gray-900"

type="text"

placeholder="Enter Your Employee Title"

{...register("title")}

></input>

</div>

</div>

<div>

<label htmlFor="department"

className="text-base font-medium text-gray-900 dark:text-gray-200"

>

{" "}

Employee Department{" "}

</label>

<div className="mt-2.5">

<input

className="flex h-10 w-full rounded-md border border-gray-300 bg-transparent py-2 px-3 text-sm placeholder:text-gray-400 focus:outline-none focus:ring-1 focus:ring-gray-400 focus:ring- offset-1 disabled:cursor-not-allowed disabled:opacity-50 dark:border-gray-700 dark:text-gray-50 dark:focus:ring-gray-400 dark:focus:ring-offset-gray-900"

type="text"

placeholder="Enter Your Employee Department"

{...register("department")}

></input>

</div>

</div>

<div>

<label htmlFor="role"

className="text-base font-medium text-gray-900 dark:text-gray-200"

>

{" "}

Employee Role{" "}

</label>

<div className="mt-2.5">

<input

className="flex h-10 w-full rounded-md border border-gray-300 bg-transparent py-2 px-3 text-sm placeholder:text-gray-400 focus:outline-none focus:ring-1 focus:ring-gray-400 focus:ring- offset-1 disabled:cursor-not-allowed disabled:opacity-50 dark:border-gray-700 dark:text-gray-50 dark:focus:ring-gray-400 dark:focus:ring-offset-gray-900"

type="text"

placeholder="Enter Your Employee Role"

{...register("role")}

></input>

</div>

</div>

<div>

<button type="submit"

className="inline-flex w-full items-center justify-center rounded-md bg-indigo-600 px-3.5 py-2.5 text-base font-semibold leading-7 text-white hover:bg-indigo-500"

>

Create Employeee

<svg xmlns="<http://www.w3.org/2000/svg>"

fill="none" viewBox="0 0 24 24" strokeWidth={1.5} stroke="currentColor"

className="ml-2 h-4 w-4"

>

<path strokeLinecap="round" strokeLinejoin="round"

d="M4.5 12h15m0 0l-6.75-6.75M19.5 12l-6.75 6.75"

/>

</svg>

</button>

</div>

</div>

</form>

</div>

);

};

export default EmployeeForm;

## HomePage.js

import React, { useEffect, useState } from "react"; import { Link } from "react-router-dom";

const HomePage = () => {

const [empData, setEmpData] = useState();

const getAllData = async () => { try {

const getPeople = await fetch(

`${process.env.REACT\_APP\_BASE\_URL}/getallUsers`,

{

method: "GET", headers: {

"Content-Type": "application/json",

},

}

);

const res = await getPeople.json(); setEmpData(res);

} catch (error) { console.log(error);

}

};

useEffect(() => { getAllData();

},[]);

console.log(empData);

// console.log(empData); return (

<>

<section className="container px-4 mx-auto py-4">

<div className="flex items-center justify-between">

<div>

<h2 className="text-lg font-medium text-gray-800 dark:text-white"> Employees

</h2>

<p className="mt-1 text-sm text-gray-500 dark:text-gray-300"> This is a list of all employees. You can add new employees, edit or delete existing ones.

</p>

</div>

<Link to={"/addemployee"}>

<div>

<button className="rounded-md bg-indigo-600 px-3.5 py-1.5 text-sm font-semibold leading-7 text-white hover:bg-indigo-500 ">

Add Employee

</button>

</div>

</Link>

</div>

<div className="flex flex-col mt-6">

<div className="-mx-4 -my-2 overflow-x-auto sm:-mx-6 lg:-mx-8">

<div className="inline-block min-w-full py-2 align-middle md:px-6 lg:px-8">

<div className="overflow-hidden border border-gray-200 dark:border-gray-700 md:rounded-lg">

<table className="min-w-full divide-y divide-gray-200 dark:divide-gray-700">

<thead className="bg-gray-50 dark:bg-gray-800">

<tr>

<th

scope="col"

className="py-3.5 px-4 text-sm font-normal text-left rtl:text-right text-gray-500 dark:text-gray-400"

>

<span>Employee</span>

</th>

<th

scope="col"

className="px-12 py-3.5 text-sm font-normal text-left rtl:text-right text-gray-500 dark:text-gray-400"

>

Title

</th>

<th

scope="col"

className="px-4 py-3.5 text-sm font-normal text-left rtl:text-right text-gray-500 dark:text-gray-400"

>

Role

</th>

</tr>

</thead>

900">

300">

<tbody className="bg-white divide-y divide-gray-200 dark:divide-gray-700 dark:bg-gray-

{empData?.data.map((person) => (

<tr key={person.name}>

<td className="py-4 px-4 whitespace-nowrap">

<div className="flex items-center">

<div className="flex-shrink-0 h-10 w-10">

<img

className="h-10 w-10 rounded-full object-cover" src={person.image}

alt=""

/>

</div>

<div className="ml-4">

<div className="text-sm font-medium text-gray-900 dark:text-white">

{person.name}

</div>

<div className="text-sm text-gray-500 dark:text-gray-300">

{person.email}

</div>

</div>

</div>

</td>

<td className="px-12 py-4 whitespace-nowrap">

<div className="text-sm text-gray-900 dark:text-white">

{person.title}

</div>

<div className="text-sm text-gray-500 dark:text-gray-300">

{person.department}

</div>

</td>

<td className="px-4 py-4 whitespace-nowrap text-sm text-gray-500 dark:text-gray-

{person.role}

</td>

</tr>

))}

</tbody>

</table>

</div>

</div>

</div>

</div>

</section>

</>

);

};

export default HomePage;

## App.js

import { Route, Routes } from "react-router-dom"; import HomePage from "./pages/HomePage";

import CreateEmployeePage from "./pages/CreateEmployeePage";

function App() { return (

<div>

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/addemployee" element={<CreateEmployeePage />} />

</Routes>

</div>

);

}

export default App;

## BACKEND

1. **Database.js**

const mongoose = require("mongoose"); require("dotenv");

const dbConnect = () => { mongoose

.connect(process.env.DATABASE\_URL, { useNewUrlParser: true, useUnifiedTopology: true,

})

.then(() => console.log("DB CONNECTION SUCCESS"))

.catch((err) => {

console.log(`DB CONNECTION ISSUES`); console.error(err.message); process.exit(1);

});

};

module.exports = dbConnect;

## Controller (CreateUser.js)

const User = require("../models/User");

exports.createUser = async (req, res) => { try {

console.log("req body", req.body);

const { name, email, title, department, role } = req.body; if (!name || !email || !title || !role || !department) { console.log("not all fields...");

return res.status(400).json({ status: 400,

message: "Please fill all fields",

});

}

const user = await User.create({ name,

email, title,

department, role,

image: `https://api.dicebear.com/5.x/initials/svg?seed=${name}`,

});

return res.status(200).json({ status: 201,

message: "User created successfully", data: user,

});

} catch (error) { console.log("error", error); return res.status(500).json({ status: 500,

message: error.message,

});

}

};

## Controller (getUser.js)

const User = require("../models/User"); exports.getUser = async (req, res) => {

try {

const userData = await User.find({}); res.json({ success: true, data: userData });

} catch (error) {

res.status(500).json({ success: false, error: error });

}

};

## Model (User.js)

const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({ name: {

type: String, required: true,

},

email: {

type: String, required: true, unique: true,

},

title: {

type: String, required: true,

},

department: { type: String, required: true, maxLength: 20,

},

role: {

type: String, required: true,

},

image: {

type: String, required: true,

},

});

module.exports = mongoose.model("User", userSchema);

## Routes (User.js)

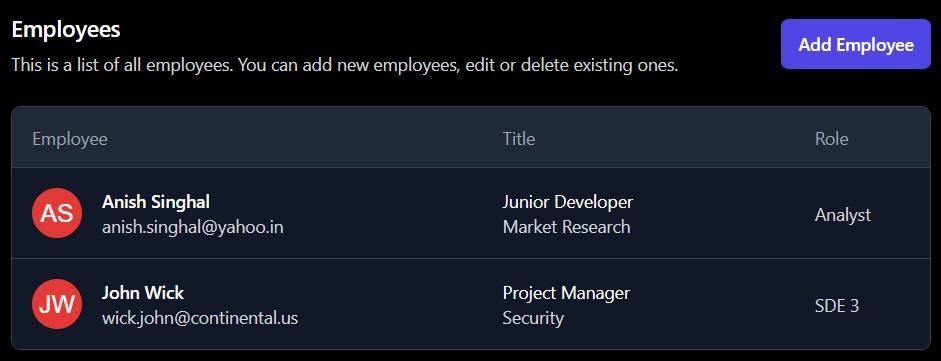
const express = require("express"); const router = express.Router();

const { createUser } = require("../controller/createUser"); const { getUser } = require("../controller/getUsers"); router.post("/createUser", createUser); router.get("/getallUsers", getUser);

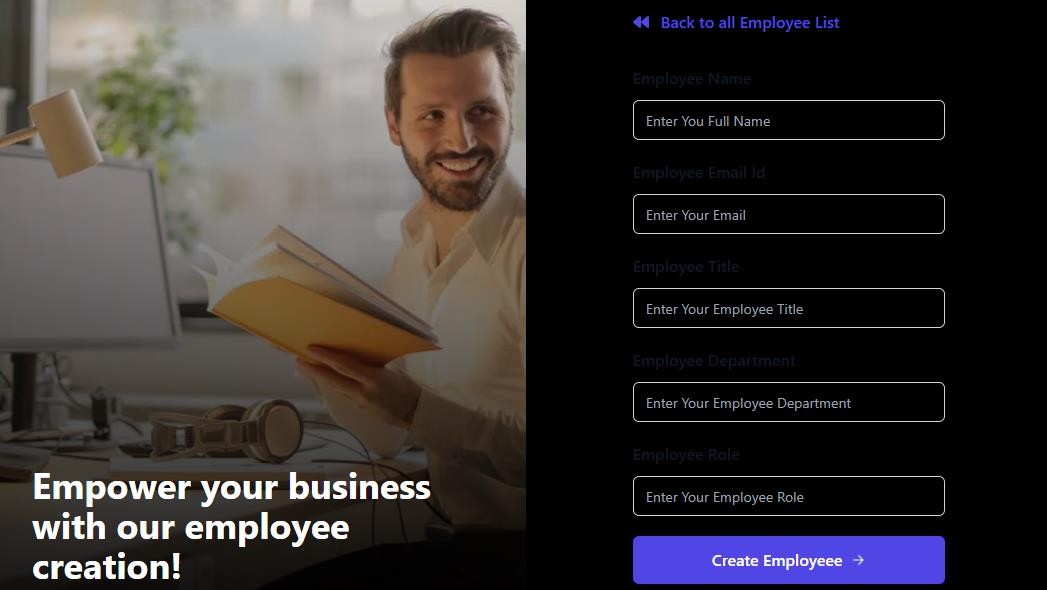
module.exports = router;

**OUTPUT:**

**Home Page**



**Employee Form Page**



# PRACTICAL 9

**Aim:** Develop robust back-end systems using Node.js.

**CODE:**

1. **Index.ejs**

<!DOCTYPE html>

<html lang="en">

<head>

<style>

body {

font-family: cursive; display: flex;

align-items: center; flex-direction: column; margin: 0;

padding: 20px; background-color: #f9f9f9;

}

h1 {

font-size: 2rem; color: #333;

margin-bottom: 20px;

}

form {

display: flex;

align-items: center; gap: 10px;

margin-bottom: 20px;

}

label {

font-weight: bold;

}

input[type="text"] { padding: 8px; width: 300px;

border: 1px solid #ddd; border-radius: 4px;

}

button {

padding: 8px 16px; background-color: #4CAF50; color: #fff;

border: none; border-radius: 4px; cursor: pointer;

}

button:hover {

background-color: #45a049;

}

table {

width: 100%;

max-width: 600px; border-collapse: collapse; margin-top: 20px; background-color: #fff; border: 1px solid #ddd; border-radius: 8px; overflow: hidden;

box-shadow: 0 4px 8px rgba(0,0,0,0.1);

}

th, td {

padding: 12px 15px; text-align: left;

border-bottom: 1px solid #ddd;

}

th {

background-color: #4CAF50; color: white;

font-weight: bold;

}

tr:hover {

background-color: #f1f1f1;

}

td:nth-child(4) { text-align: center;

}

</style>

<meta charset="UTF-8">

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

<title>Home Page</title>

</head>

<body>

<h1>URL Shortener</h1>

<% if (locals.id) { %>

<p>URL Generated: http://localhost:8001/url/<%= id %></p>

<% } %>

<div>

<form method="POST" action="/url">

<label>Enter Your Original URL</label>

<input

type="text" name="url"

placeholder="https://example.com"

/>

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

</form>

</div>

<div>

<% if(locals.urls){ %>

<table>

<thead>

<tr>

<th>S.No</th>

<th>ShortID</th>

<th>Redirect</th>

<th>Clicks</th>

</tr>

</thead>

<tbody>

<% urls.forEach( (url,index) => { %>

<tr>

<td><%= index + 1 %></td>

<td><%= url.shortId %></td>

<td><%= url.redirectURL %></td>

<td><%= url.visitHistory.length %></td>

</tr>

<% } ) %>

</tbody>

</table>

<% } %>

</div>

</body>

</html>

## database model (model > url.js )

const mongoose = require('mongoose');

const urlSchema = new mongoose.Schema({ shortId:{

type:String, require:true, unique:true,

},

redirectURL: { type: String, require: true,

},

visitHistory: [{

timestamp: {type:Number}

}],

},{timestamps:true}

);

## Controller (controller > url.js )

const URL = mongoose.model('url' , urlSchema); module.exports = URL;

const shortid = require("shortid");

const URL = require('../models/url.js');

async function handleGenerateNewShortURL(req, res){ const body = req.body;

if(!body.url) return res.status(400).json({error: 'url is required'}) const shortID = shortid();

await URL.create({ shortId: shortID, redirectURL: body.url, visitHistory: [],

});

return res.render('home' , { id: shortID,

});

}

async function handleGetAnalytics(req , res){ const shortId = req.params.shortId;

const result = await URL.findOne({shortId});

return res.json({totalClicks: result.visitHistory.length , analytics: result.visitHistory})

}

module.exports = { handleGenerateNewShortURL, handleGetAnalytics,

}

## Router (routes > staticRouter.js )

const express = require("express"); const router = express.Router(); const URL = require("../models/url")

router.get('/', async(req , res) => {

const allurls = await URL.find({}); return res.render('home' , {

urls: allurls

})

})

module.exports = router;

## Router (routes > url.js )

const express = require("express"); const router = express.Router();

const { handleGenerateNewShortURL , handleGetAnalytics } = require('../controllers/url')

router.post('/' , handleGenerateNewShortURL); router.get('/analytics/:shortId' , handleGetAnalytics); module.exports = router;

## Index.js

const express = require('express') const app = express();

const path = require('path') const PORT = 8001;

const urlRoute = require('./routes/url');

const {connectToMongoDB} = require("./connect"); const URL = require("./models/url");

app.set("view engine" , "ejs"); app.set('views' , path.resolve("./views"));

const staticRoute = require("./routes/staticRouter");

app.use(express.json()); app.use(express.urlencoded({extended:false}));

connectToMongoDB('mongodb://localhost:27017/short-url')

.then(()=>console.log('Mongo DB connected'));

app.use("/url" , urlRoute); app.get('/url/:shortID' , async (req ,res)=>{

const shortId = req.params.shortID;

const entry = await URL.findOneAndUpdate(

{

shortId

},

{

$push: { visitHistory: {

timestamp: Date.now(),

},

},

}

);

return res.redirect(entry.redirectURL);

});

app.use('/' , staticRoute);

app.listen(PORT , () => console.log(`Server Starrted at PORT at ${PORT}`))

## Connect.js

const mongoose = require('mongoose'); mongoose.set("strictQuery" , true); async function connectToMongoDB(url){

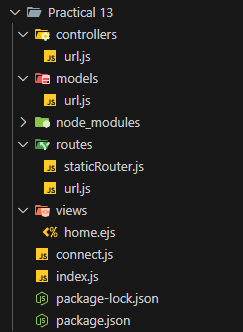
return mongoose.connect(url);

}

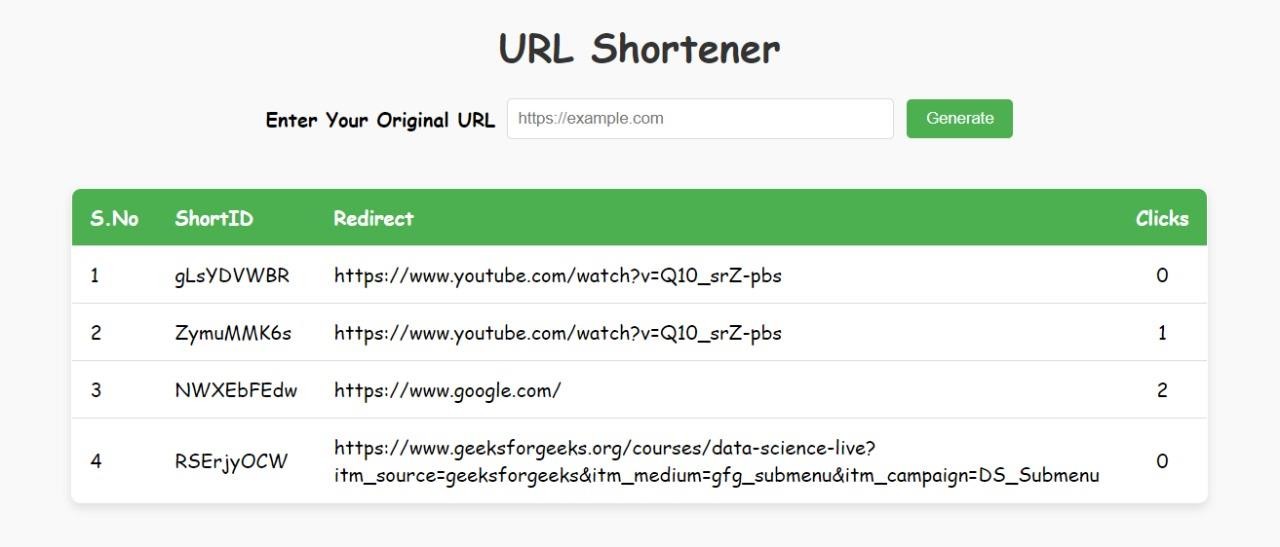
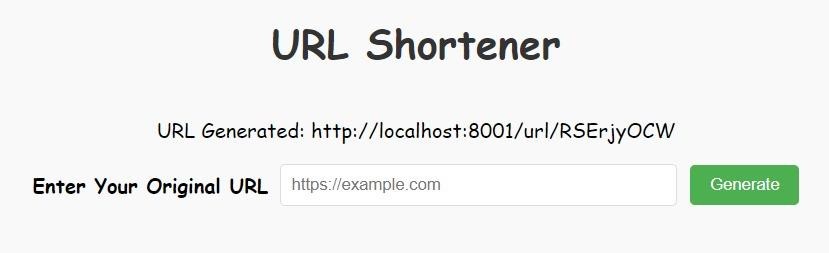
module.exports = { connectToMongoDB

}

**File Structure:**



**OUTPUT:**



# PRACTICAL 10

**Aim:** Showing database interactivity using PHP/Python/or any current technology used in web industry.

**CODE:**

1. **Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Weather App</title>

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

</head>

<body>

<div class="container">

<h1>Weather App</h1>

<input type="text" id="cityInput" placeholder="Enter city name">

<button onclick="getWeather()">Get Weather</button>

<div id="weatherResult"></div>

</div>

<script src="../static/script.js"></script>

</body>

</html>

## Style.css

body {

font-family: Arial, sans-serif; display: flex;

justify-content: center; align-items: center; height: 100vh;

margin: 0;

background-color: #f4f4f9;

}

.container {

text-align: center; width: 300px;

}

h1 {

color: #333;

}

#cityInput { width: 80%;

padding: 10px; margin-top: 10px; margin-bottom: 10px;

}

button {

padding: 10px 15px; background-color: #007bff; color: white;

border: none; cursor: pointer; border-radius: 5px;

}

button:hover {

background-color: #0056b3;

}

#weatherResult { margin-top: 20px; font-size: 1.2em;

}

## Script.js

function getWeather() {

const city = document.getElementById("cityInput").value; fetch("/get\_weather", {

method: "POST", headers: {

"Content-Type": "application/json"

},

body: JSON.stringify({ city: city })

})

.then(response => response.json())

.then(data => {

if (data.error) { document.getElementById("weatherResult").innerText = data.error;

} else {

document.getElementById("weatherResult").innerHTML = `

<p><strong>City:</strong> ${data.city}</p>

<p><strong>Temperature:</strong> ${data.temperature}°C</p>

<p><strong>Feels Like:</strong> ${data.feels\_like}°C</p>

<p><strong>Weather:</strong> ${data.weather}</p>

<p><strong>Wind Speed:</strong> ${data.wind\_speed} m/s</p>

`;

}

})

.catch(error => { console.error("Error:", error);

document.getElementById("weatherResult").innerText = "An error occurred. Please try again.";

});

}

## app.py (For Backend)

from flask import Flask, render\_template, request, jsonify import requests

app = Flask( name )

API\_KEY = "e4c2903df887d05c7ce3dc39fb60a143"

BASE\_URL = ["http:](http://api.openweathermap.org/data/2.5/weather)/[/api.openweathermap.org/data/2.5/weather](http://api.openweathermap.org/data/2.5/weather)"

@app.route('/') def index():

return render\_template("index.html")

@app.route('/get\_weather', methods=['POST']) def get\_weather():

data = request.get\_json() city = data.get("city")

if not city:

return jsonify({"error": "City name is required"}), 400

try:

params = {'q': city, 'appid': API\_KEY, 'units': 'metric'} response = requests.get(BASE\_URL, params=params) weather\_data = response.json()

if weather\_data['cod'] == 200:

return jsonify({ "city": city,

"temperature": weather\_data['main']['temp'], "feels\_like": weather\_data['main']['feels\_like'], "weather": weather\_data['weather'][0]['description'], "wind\_speed": weather\_data['wind']['speed']

})

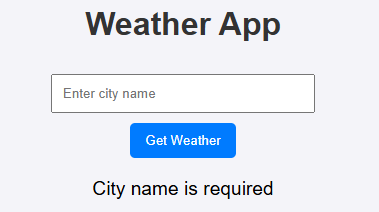
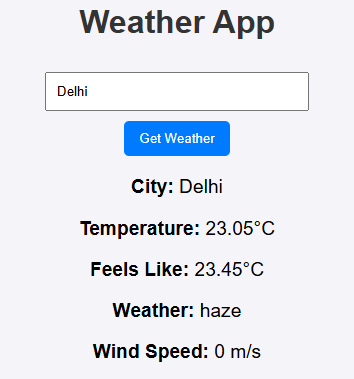
else:

return jsonify({"error": "City not found"}), 404 except Exception as e:

return jsonify({"error": str(e)}), 500

if name == ' main ': app.run(debug=True)

**OUTPUT:**



# PRACTICAL 11

**Aim:** Any current web industry relevant example of database usage and interactivity using any suitable backend technology.

**THEORY:**

Backend Technology:

* Framework: Node.js with Express.js
* Database: MySQL or PostgreSQL

Database Structure:

An e-commerce website would have a complex database structure, but here's a simplified version:

* Products:
  + product\_id (Primary Key)
  + name
  + description
  + price
  + category\_id (Foreign Key)
  + stock\_quantity
  + image\_url
* Categories:
  + category\_id (Primary Key)
  + name
* Users:
  + user\_id (Primary Key)
  + name
  + email
  + password\_hash
  + address
* Orders:
  + order\_id (Primary Key)
  + user\_id (Foreign Key)
  + order\_date
  + total\_amount
* Order\_Items:
  + order\_item\_id (Primary Key)
  + order\_id (Foreign Key)
  + product\_id (Foreign Key)
  + quantity
  + price

Interactivity and Database Usage:

1. Product Search:
   * User enters a keyword in the search bar.
   * Frontend sends the keyword to the backend.
   * Backend queries the Products table using SQL to find matching products.
   * Results are returned to the frontend and displayed to the user.
2. Adding Products to Cart:
   * User clicks the "Add to Cart" button.
   * Frontend sends the product ID and quantity to the backend.
   * Backend checks the Products table for stock availability.
   * If available, a new entry is added to the user's shopping cart (which could be a session-based cart or a database-backed cart).
3. Checkout Process:
   * User proceeds to checkout.
   * Frontend sends user information and cart items to the backend.
   * Backend validates user information and calculates the total order amount.
   * A new Order is created in the Orders table.
   * Corresponding Order\_Items are created for each product in the cart.
   * The Products table is updated to reflect the reduced stock quantity.
4. User Reviews:
   * User submits a review for a product.
   * Frontend sends the review text, rating, and product ID to the backend.
   * A new review is added to the database, potentially linked to the Products table.
5. Personalized Recommendations:
   * Backend analyzes user purchase history and browsing behavior.
   * Machine learning algorithms can be used to recommend products based on user preferences.
   * Recommended products are fetched from the database and displayed on the frontend.