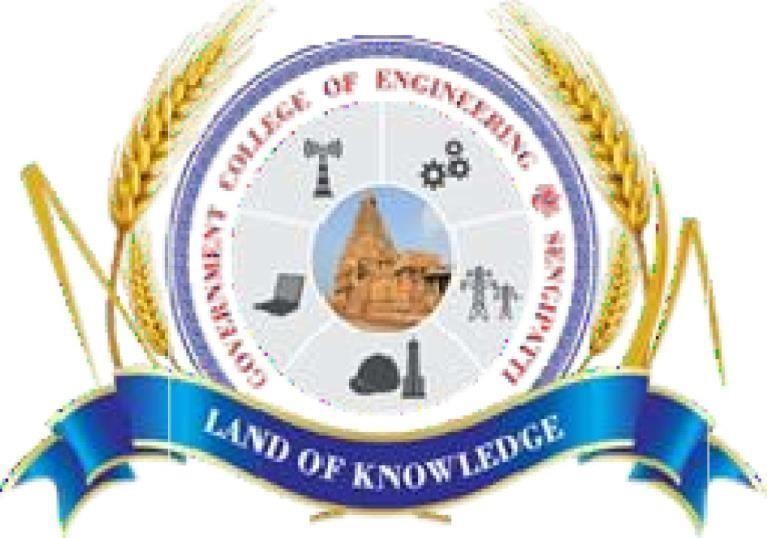
**GOVERNMENT COLLEGE OF ENGINEERING**

**(Affiliated to Anna University, Chennai) THANJAVUR - 613402**



**DEPARTMENT OF**

**ELECTRONICS & COMMUNICATION ENGINEERING**

# NM1050 – SaaS LABORATORY

**RECORD NOTE BOOK**

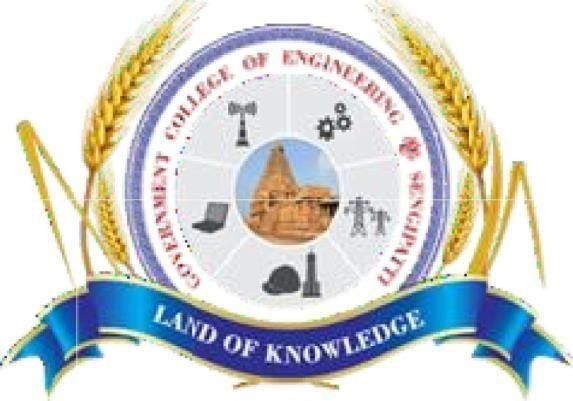
# NAME :

**REG NO :**

# YEAR :

**GOVERNMENT COLLEGE OF ENGINEERING**

# (Affiliated to Anna University, Chennai) THANJAVUR - 613402



**BONAFIDE CERTIFICATE**

NAME :

YEAR : SEMESTER :

REGISTER No.:

Certified that this is the Bonafide Record of work done by the above

student in the…………………………………………………….Laboratory

during the year …………………………..

**STAFF IN CHARGE HEAD OF THE**

**DEPARTMENT**

Submitted for University Practical Examination held on

## **INTERNAL EXAMINER EXTERNAL EXAMINER**

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp. No** | **Date** | **Name of the Experiment** | **Page No** | **Signature** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**INTRODUCTION TO HTML AND BASIC TAGS**

DATE:

EX.NO:01

**AIM:**

To understand the structure of an HTML document and explore the use of basic tags to create a simple webpage.

**PROGRAM**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Experiment 1: Introduction to HTML</title>

</head>

<body>

**<!-- Heading -->**

<h1>Welcome to HTML Basics</h1>

**<!-- Paragraph -->**

<p>HTML is the foundation of web development. It defines the structure and layout of a webpage.</p>

**<!-- Subheading -->**

<h2>Features of HTML</h2>

<p>Some of the key features of HTML include:</p>

**<!-- Unordered List -->**

<ul>

<li>Easy to learn and use.</li>

<li>Platform independent and supported by all browsers.</li>

<li>Supports multimedia like images, videos, and audio.</li>

</ul>

**<!-- Subheading -->**

<h2>Basic HTML Example</h2>

**<!-- Ordered List -->**

<p>Steps to create a simple webpage:</p>

<ol>

<li>Define the structure using HTML tags.</li>

<li>Style it with CSS (optional).</li>

<li>Add interactivity with JavaScript (optional).</li>

</ol>

**<!-- Hyperlink -->**

<p>Learn more about HTML at

<a href="https://www.w3schools.com/html/" target="\_blank">W3Schools</a>.

</p>

**<!-- Image -->**

<h2>HTML Logo</h2>

<img src="https://www.w3.org/html/logo/downloads/HTML5\_Logo\_512.png" alt="HTML Logo" width="200">

**<!-- Footer -->**

<footer>

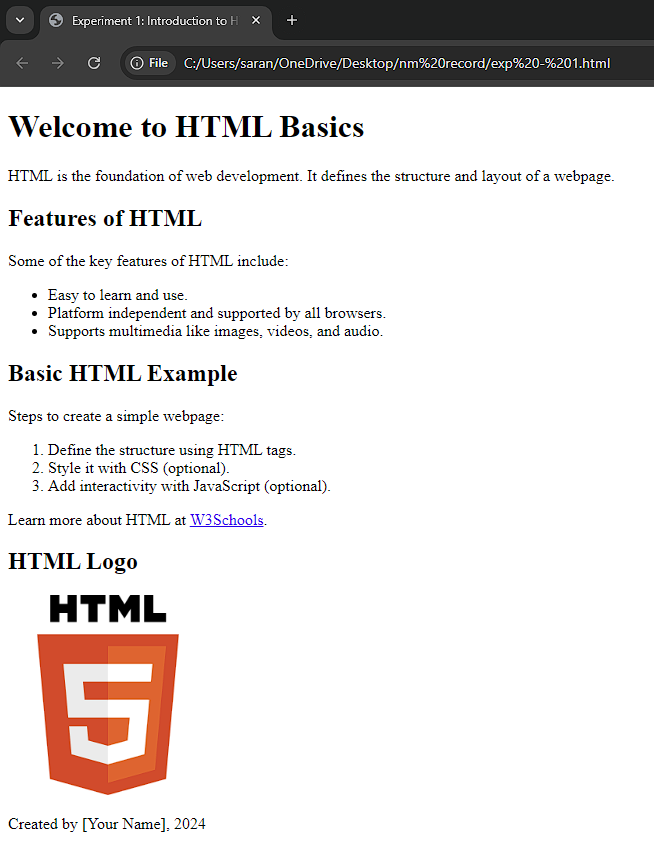
<p>Created by [Your Name], 2024</p>

</footer>

</body>

</html>

**OUTPUT:**



**RESULT:**

The experiment was successfully conducted. A basic HTML document was created and rendered in a web browser, showcasing the use of headings, paragraphs, and lists.

**STYLING WEB PAGES WITH CSS**

EX.NO:02

DATE:

**AIM:**

To understand and apply CSS properties to style a web page, enhancing its visual appeal and user experience.

**PROGRAM:**

**HTML File:** index.html

    <!DOCTYPE html>

    <html lang="en">

    <head>

        <meta charset="UTF-8">

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

        <title>Styling with CSS</title>

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

    </head>

    <body>

        <header>

            <h1>Welcome to My Styled Web Page</h1>

        </header>

        <nav>

            <ul>

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

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

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

            </ul>

        </nav>

        <main>

            <section>

                <h2>About This Page</h2>

                <p>This page demonstrates basic CSS styling techniques for web pages.</p>

            </section>

            <section>

                <h2>Features</h2>

                <ul>

                    <li>Responsive Design</li>

                    <li>Custom Fonts</li>

                    <li>Hover Effects</li>

                </ul>

            </section>

        </main>

        <footer>

            <p>&copy; 2024 My Styled Page</p>

        </footer>

    </body>

    </html>

**CSS File: styles.css**

**/\* General Styling \*/**

    body {

        font-family: Arial, sans-serif;

        margin: 0;

        padding: 0;

        line-height: 1.6;

        background-color: #f4f4f4;

        color: #333;

    }

**/\* Header Styling \*/**

    header {

        background-color: #4CAF50;

        color: white;

        padding: 10px 0;

        text-align: center;

    }

**/\* Navigation Menu \*/**

    nav ul {

        list-style: none;

        padding: 0;

        display: flex;

        justify-content: center;

        background-color: #333;

    }

    nav ul li {

        margin: 0 10px;

    }

    nav ul li a {

        text-decoration: none;

        color: white;

        padding: 10px;

        display: inline-block;

    }

    nav ul li a:hover {

        background-color: #4CAF50;

        border-radius: 5px;

    }

**/\* Main Section \*/**

    main {

        padding: 20px;

    }

    h2 {

        color: #4CAF50;

    }

**/\* Footer Styling \*/**

    footer {

        background-color: #333;

        color: white;

        text-align: center;

        padding: 10px 0;

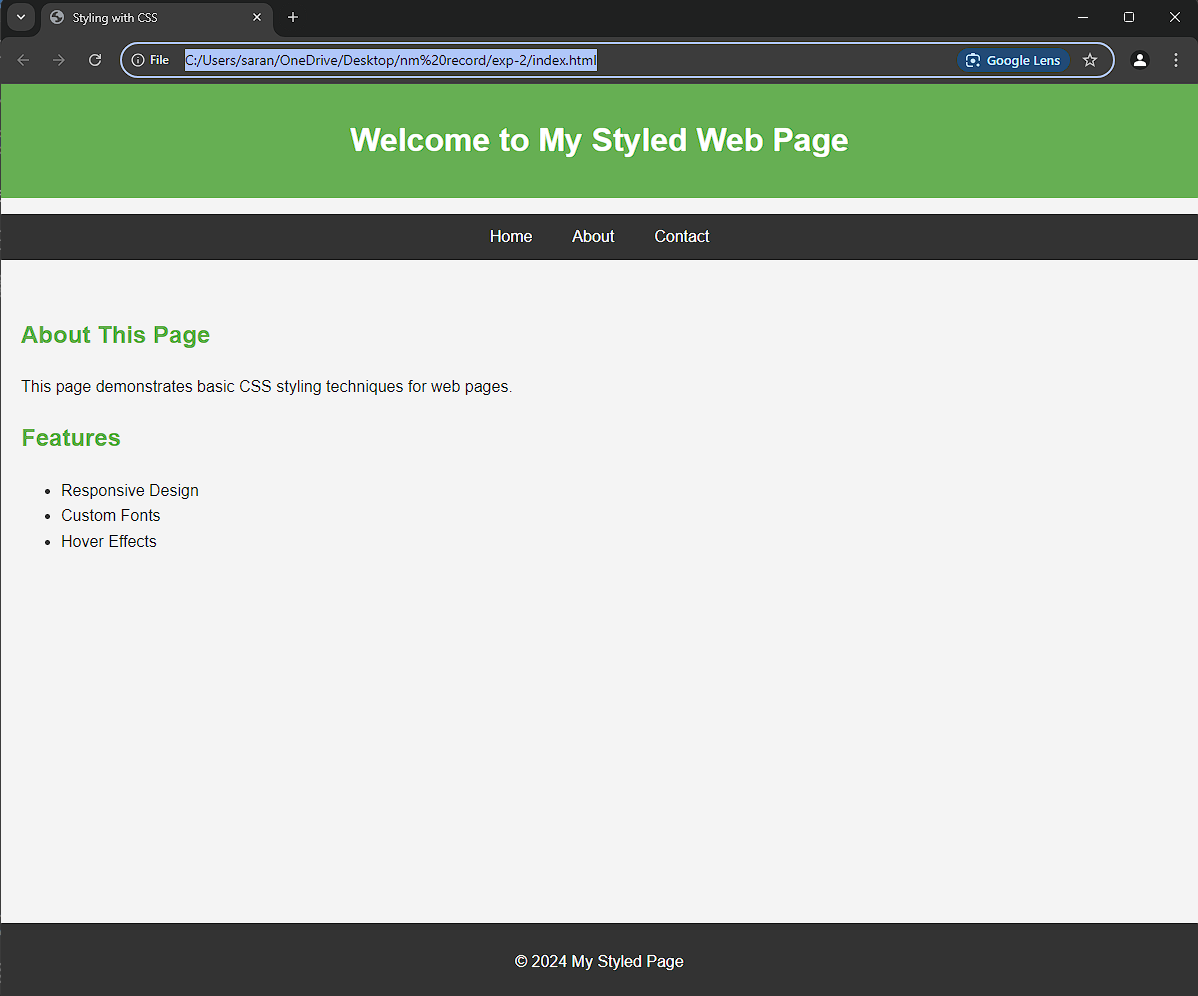
        position: fixed;

        bottom: 0;

        width: 100%;

    }

**OUTPUT:**



**RESULT:**

The experiment successfully demonstrated the use CSS to style an HTML web page, enhancing its visual appearance and structure.

**RESPONSIVE WEB DESIGN WITH FLEXBOX**

EX.NO:03

DATE:

**AIM:**

To implement responsive web layouts using CSS Flexbox, ensuring elements adapt seamlessly to different screen sizes and resolutions.

**PROGRAM:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Responsive Web Design with Flexbox</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: Arial, sans-serif;

line-height: 1.5;

}

header {

background-color: #333;

color: #fff;

padding: 10px;

text-align: center;

}

.container {

display: flex;

flex-wrap: wrap;

gap: 10px;

padding: 10px;

}

.box {

flex: 1 1 calc(33.333% - 20px); **/\* 3 items per row with gaps \*/**

background-color: #f4f4f4;

padding: 20px;

border: 1px solid #ccc;

text-align: center;

}

**/\* Responsive design for smaller devices \*/**

@media (max-width: 768px) {

.box {

flex: 1 1 calc(50% - 20px); /\* 2 items per row \*/

}

}

@media (max-width: 480px) {

.box {

flex: 1 1 100%; /\* 1 item per row \*/

}

}

</style>

</head>

<body>

<header>

<h1>Responsive Web Design with Flexbox</h1>

</header>

<div class="container">

<div class="box">Box 1</div>

<div class="box">Box 2</div>

<div class="box">Box 3</div>

<div class="box">Box 4</div>

<div class="box">Box 5</div>

<div class="box">Box 6</div>

</div>

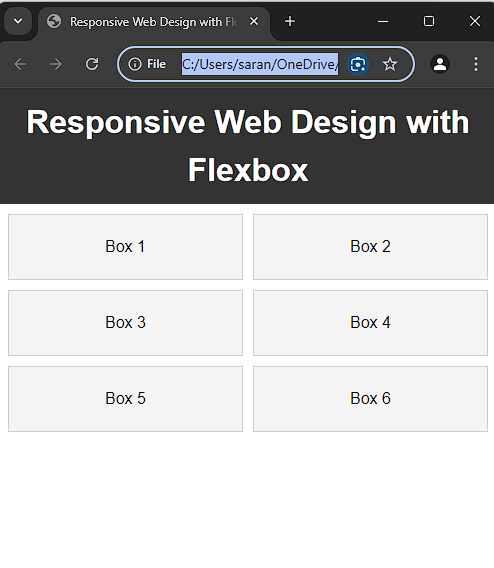
</body>

</html>

**OUTPUT:**

****

**In large Devices**

****

**In smaller Devices**

**RESULT:**

The responsive web layout was successfully implemented using CSS Flexbox. The design adapts smoothly to various screen sizes, displaying three, two on desktops, tablets, and mobiles, respectively.

**BUILDING INTERACTIVE FORMS IN HTML**

EX.NO:04

DATE:

**AIM:**

To design and implement interactive HTML forms with validation mechanisms to ensure data accuracy and user experience

**PROGRAM:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Interactive Form with Validation</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

padding: 0;

background-color: #f4f4f9;

}

form {

background-color: #fff;

padding: 20px;

border-radius: 5px;

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

width: 300px;

margin: 0 auto;

}

input[type="text"], input[type="email"], input[type="password"], input[type="submit"] {

width: 100%;

padding: 10px;

margin: 10px 0;

border-radius: 4px;

border: 1px solid #ccc;

}

input[type="submit"] {

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #45a049;

}

.error {

color: red;

font-size: 12px;

}

</style>

</head>

<body>

<h2>Registration Form</h2>

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

<label for="username">Username:</label>

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

<span id="usernameError" class="error"></span>

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

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

<span id="emailError" class="error"></span>

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

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

<span id="passwordError" class="error"></span>

<input type="submit" value="Submit">

</form>

<script>

function validateForm() {

let username = document.getElementById("username").value;

let email = document.getElementById("email").value;

let password = document.getElementById("password").value;

let usernameError = document.getElementById("usernameError");

let emailError = document.getElementById("emailError");

let passwordError = document.getElementById("passwordError");

let valid = true;

**// Clear previous error messages**

usernameError.innerHTML = "";

emailError.innerHTML = "";

passwordError.innerHTML = "";

**// Validate username**

if (username.length < 5) {

usernameError.innerHTML = "Username must be at least 5 characters long.";

valid = false;

}

**// Validate email format**

let emailPattern = /^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;

if (!emailPattern.test(email)) {

emailError.innerHTML = "Please enter a valid email address.";

valid = false;

}

**// Validate password strength**

if (password.length < 8) {

passwordError.innerHTML = "Password must be at least 8 characters long.";

valid = false;

}

return valid;

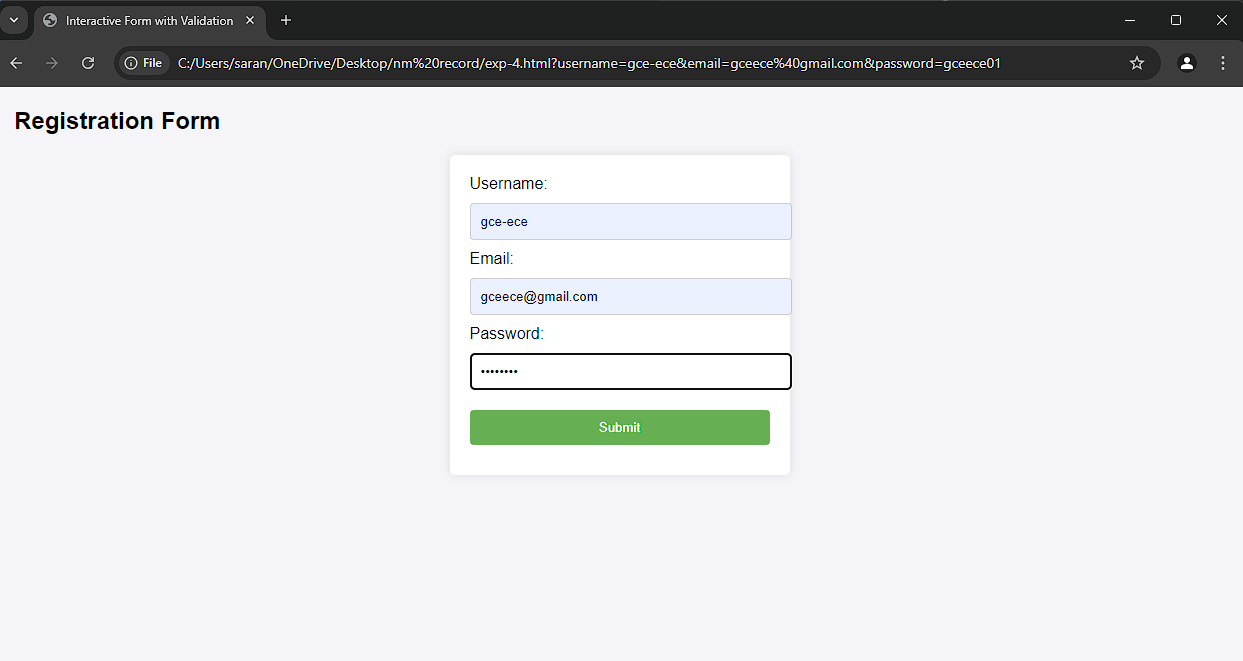
}

</script>

</body>

</html>

**OUTPUT:**



**RESULT:**

This experiment demonstrates the creation of interactive HTML forms with basic validation and has been implemented successfully.

**INTRODUCTION TO REST API AND FETCHING DATA**

EX.NO:05

DATE:

**AIM:**

To understand the concept of REST APIs and simulate REST API requests using JavaScript to fetch data from a given API.

**PROGRAM:**

**HTML File:** index.html

<!DOCTYPE html>

    <html lang="en">

    <head>

        <meta charset="UTF-8">

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

        <title>Fetch Data from REST API</title>

        <style>

            body {

                font-family: Arial, sans-serif;

                background-color: #f4f4f9;

                margin: 0;

                padding: 0;

                display: flex;

                flex-direction: column;

                align-items: center;

            }

            h1 {

                color: #333;

                margin-top: 20px;

            }

            .post {

                background-color: #fff;

                border: 1px solid #ddd;

                border-radius: 8px;

                padding: 15px;

                margin: 10px 0;

                box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

                width: 80%;

                max-width: 600px;

            }

            .post h3 {

                font-size: 18px;

                color: #2c3e50;

            }

            .post p {

                font-size: 14px;

                color: #7f8c8d;

            }

            .error {

                color: #e74c3c;

                font-size: 16px;

            }

            #output {

                margin-top: 30px;

                width: 100%;

                display: flex;

                flex-direction: column;

                align-items: center;

            }

        </style>

    </head>

    <body>

        <h1>Data from REST API</h1>

        <div id="output">

            <!-- Data fetched from the API will be displayed here -->

        </div>

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

    </body>

    </html>

**JavaScript File:** script.js

**// Define the API URL**

const apiURL = 'https://jsonplaceholder.typicode.com/posts';

**// Function to fetch data from the API**

async function fetchData() {

    try {

        const response = await fetch(apiURL);

        // Check if the response is OK (status code 200-299)

        if (!response.ok) {

            throw new Error('Failed to fetch data. Please try again later.');

        }

        const data = await response.json();

**// Call function to display the data on the webpage**

        displayData(data);

    } catch (error) {

        // Display error message to the user if the fetch operation fails

        displayError(error.message);

    }

}

**// Function to display data in the HTML**

function displayData(data) {

    const outputDiv = document.getElementById('output');

    outputDiv.innerHTML = ''; // Clear any previous data

**// Loop through the data array and create HTML elements for each post**

    data.forEach(item => {

        const postDiv = document.createElement('div');

        postDiv.classList.add('post');

        postDiv.innerHTML = `

            <h3>${item.title}</h3>

            <p>${item.body}</p>

            <hr>

        `;

        outputDiv.appendChild(postDiv);

    });

}

**// Function to display error messages**

function displayError(message) {

    const outputDiv = document.getElementById('output');

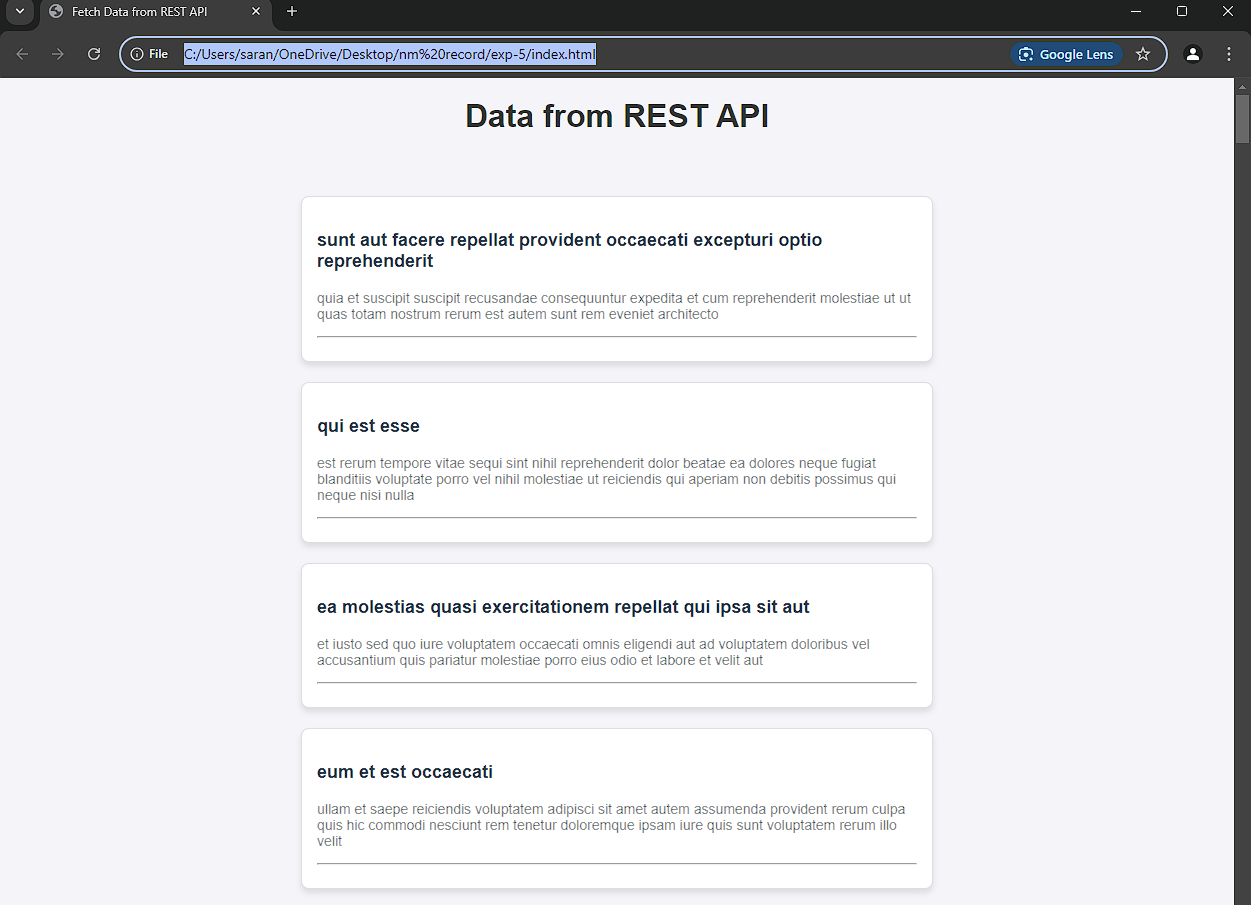
    outputDiv.innerHTML = `<p class="error">${message}</p>`;

}

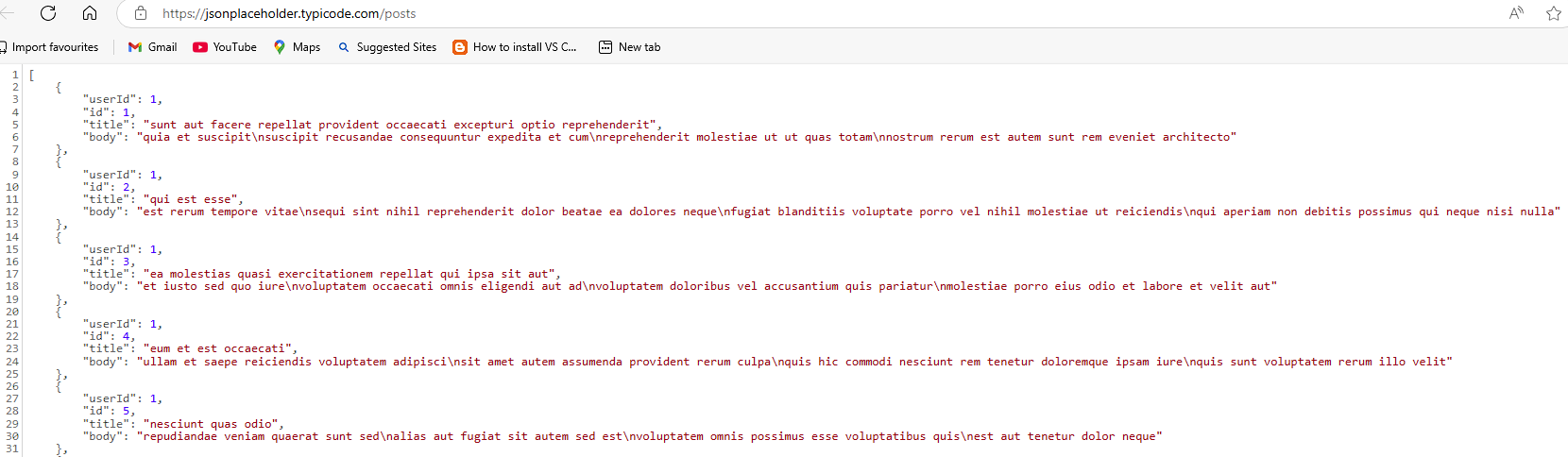
**// Call the fetchData function when the page loads**

window.onload = fetchData;

**OUTPUT:**



**Data displayed from API**

****

**REST API Data**

**RESULT:**

Thus the program will simulate fetching data from a REST API using JavaScript. The fetched data is then dynamically displayed on the webpage. This experiment demonstrates how to interact with REST APIs, handle the responses, and display the data on the web page.

**CRUD OPERATIONS IN MYSQL**

EX.NO:06

DATE:

**AIM:**

To perform the basic CRUD (Create, Read, Update, and Delete) operations in MySQL to manage data in a database.

**PROGRAM:**

**-- Step 1: Create a Table**

CREATE TABLE Employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

age INT NOT NULL,

position VARCHAR(50)

);

**-- Step 2: Insert Data**

INSERT INTO Employees (name, age, position)

VALUES

('John Doe', 30, 'Software Engineer'),

('Jane Smith', 25, 'Data Analyst'),

('Alice Johnson', 28, 'Product Manager'),

('Bob Brown', 35, 'DevOps Engineer');

**-- Step 3: Read Data (Retrieve all rows)**

SELECT \* FROM Employees;

**-- Step 4: Update Data**

-- Example: Update the position of 'John Doe'

UPDATE Employees

SET position = 'Senior Software Engineer', age = 31

WHERE name = 'John Doe';

**-- Step 5: Delete Data**

-- Example: Delete the record of 'Bob Brown'

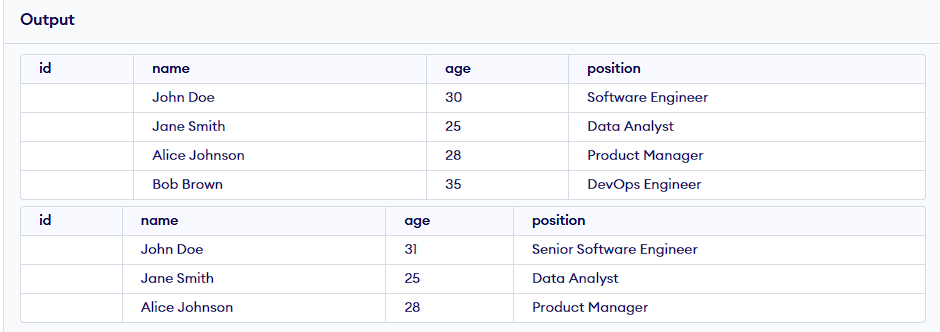
DELETE FROM Employees

WHERE name = 'Bob Brown';

**-- Step 6: Read Data Again**

SELECT \* FROM Employees;

**OUTPUT:**

****

**RESULT:**

Thus, the program has been executed and output is verified successfully using MATLAB

R-2021a software.

**DEPLOYING A WEB APPLICATION WITH AWS EC2**

DATE:

EX.NO:07

**AIM:**

To deploy a simple web application on AWS EC2, leveraging its virtual server capabilities to host dynamic and static content, and demonstrate the process of cloud-based application deployment.

**PROGRAM:**

**Step 1: Launch an EC2 Instance**

1. Open the **AWS Management Console**.
2. Go to **EC2 Dashboard** → **Launch Instance**.
3. Configure the instance:
   * AMI: **Amazon Linux 2 AMI**.
   * Instance Type: **t2.micro**.
   * Security Group: Add a rule to allow HTTP (Port 80) access from anywhere.
   * Create or use an existing key pair to access the instance.

**Step 2: Connect to the EC2 Instance**

Use the following command to SSH into the instance:

ssh -i /path/to/your-key.pem ec2-user@<public-ip>

**Step 3: Install and Configure the Web Server**

1. Update packages:

sudo yum update -y

1. Install Apache (HTTPD) and PHP:

sudo yum install -y httpd php

1. Start the Apache web server:

sudo systemctl start httpd

sudo systemctl enable httpd

**Step 4: Deploy a Simple Application**

Create a basic PHP application:

echo "<?php echo 'Hello, AWS EC2!'; ?>" | sudo tee /var/www/html/index.php

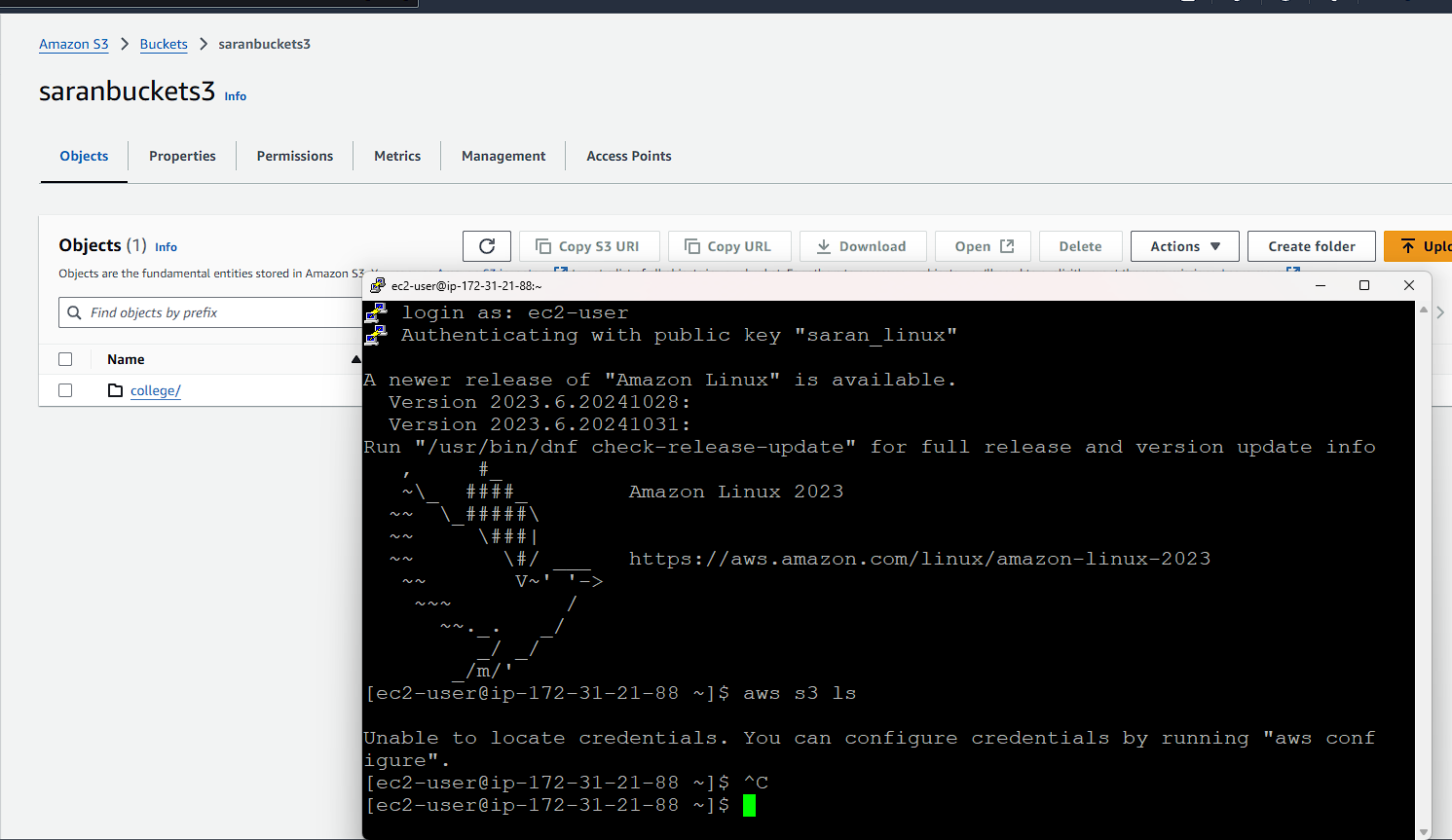
**Step 5: Access the Application**

1. Open a browser and visit:

http://<public-ip>

Replace <public-ip> with the public IP address of the EC2 instance.

**OUTPUT:**

****

**RESULT:**

Thus, the experiment successfully deployed a web application on AWS EC2, making it accessible via the instance's public IP Address.

**RESPONSIVE WEB DESIGN USING BOOTSTRAP**

EX.NO:08

DATE:

**AIM:**

To design and implement a responsive multi-column layout using Bootstrap framework, demonstrating adaptability across various screen sizes.

**PROGRAM:**

    <!DOCTYPE html>

    <html lang="en">

    <head>

        <meta charset="UTF-8">

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

        <title>Responsive Web Design</title>

**<!-- Bootstrap CSS -->**

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

        <style>

            body {

                font-family: 'Arial', sans-serif;

                background-color: #f8f9fa;

            }

            .container {

                margin-top: 50px;

            }

            .custom-column {

                border-radius: 10px;

                box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

                transition: transform 0.3s, box-shadow 0.3s;

            }

            .custom-column:hover {

                transform: scale(1.05);

                box-shadow: 0 8px 12px rgba(0, 0, 0, 0.2);

            }

            h1 {

                font-size: 2.5rem;

                margin-bottom: 30px;

                color: #343a40;

            }

            h3 {

                font-size: 1.5rem;

            }

            p {

                font-size: 1rem;

                line-height: 1.6;

            }

        </style>

</head>

    <body>

        <div class="container text-center">

            <h1>Responsive Web Design with Bootstrap</h1>

            <div class="row g-4">

**<!-- Column 1 -->**

                <div class="col-lg-4 col-md-6 col-sm-12">

                    <div class="custom-column bg-primary text-white p-4">

                        <h3>Column 1</h3>

                        <p>This column demonstrates a responsive layout. It will resize and reposition based on the screen size.</p>

                    </div>

                </div>

**<!-- Column 2 -->**

                <div class="col-lg-4 col-md-6 col-sm-12">

                    <div class="custom-column bg-secondary text-white p-4">

                        <h3>Column 2</h3>

                        <p>This column highlights the adaptability of Bootstrap's grid system for various devices.</p>

                    </div>

                </div>

**<!-- Column 3 -->**

                <div class="col-lg-4 col-md-12 col-sm-12">

                    <div class="custom-column bg-success text-white p-4">

                        <h3>Column 3</h3>

                        <p>This column is stacked on smaller screens, ensuring content is readable and user-friendly.</p>

                    </div>

                </div>

            </div>

        </div>

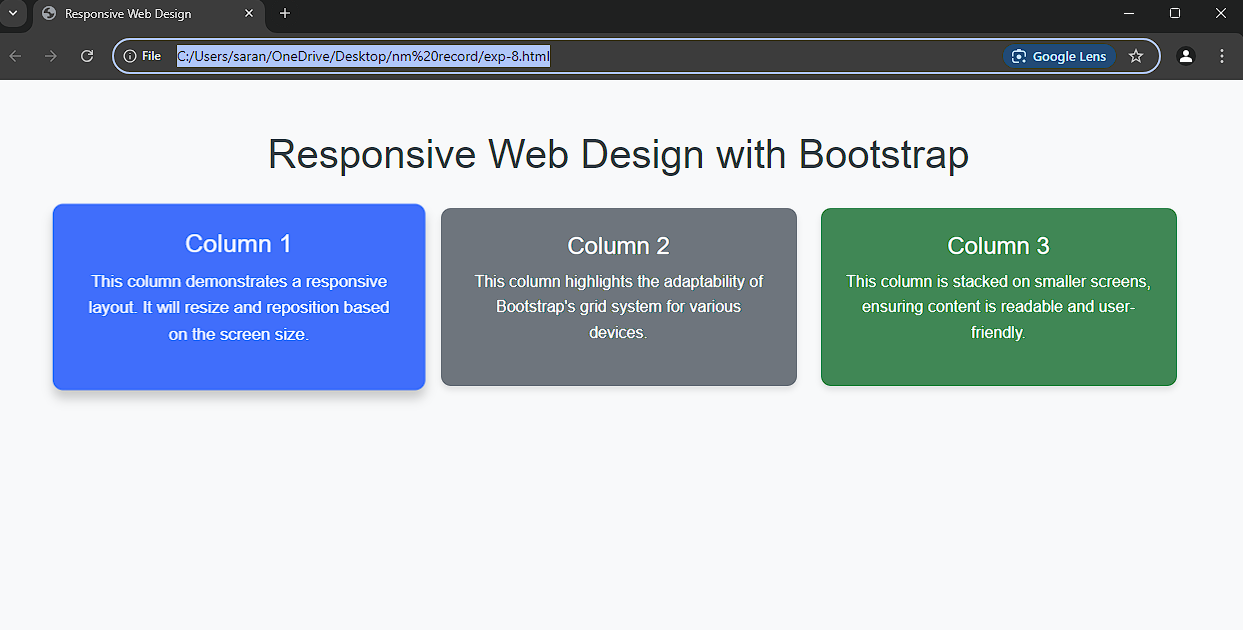
**<!-- Bootstrap JS -->**

        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

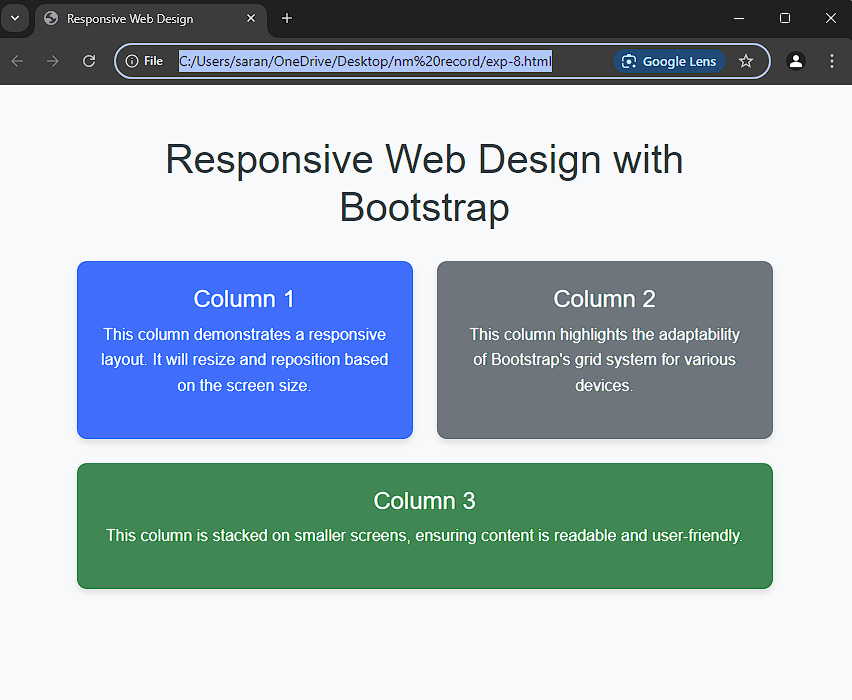
    </body>

    </html>

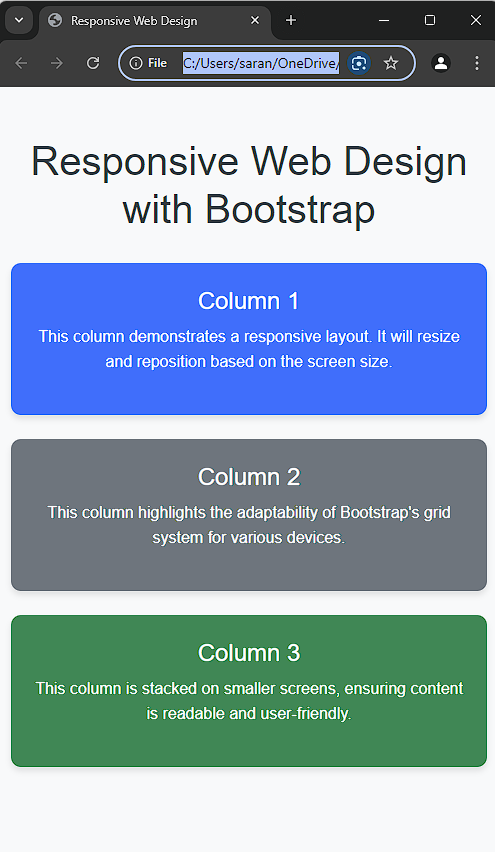
**OUTPUT:**



**In larger Screen**

****

**In medium Screen**



**In smaller Screen**

**RESULT:**

Thus, the experiment demonstrates the use of Bootstrap classes to design a responsive multi-column layout. The webpage adjusts its layout dynamically based on the screen size, ensuring a seamless user experience across devices.

**IMPLEMENTING USER AUTHENTICATION SYSTEM**

EX.NO:09

DATE:

**AIM:**

To design and implement a basic user authentication system using JavaScript and backend logic, ensuring secure login functionality.

**PROGRAM:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Enhanced User Authentication</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f4f4f9;

            text-align: center;

            padding: 50px;

        }

        form {

            background: #fff;

            padding: 20px;

            border-radius: 8px;

            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

            display: inline-block;

        }

        input {

            padding: 10px;

            margin: 10px;

            width: 90%;

            border: 1px solid #ccc;

            border-radius: 5px;

        }

        button {

            padding: 10px 15px;

            border: none;

            background: #4CAF50;

            color: white;

            border-radius: 5px;

            cursor: pointer;

        }

        .error {

            color: red;

        }

        .success {

            color: green;

        }

    </style>

</head>

<body>

    <h1>Enhanced User Authentication</h1>

**<!-- Login Form -->**

    <div id="loginForm">

        <h2>Login</h2>

        <input type="text" id="loginUsername" placeholder="Username" required />

        <input type="password" id="loginPassword" placeholder="Password" required />

        <button type="button" onclick="authenticateUser()">Login</button>

        <p id="loginMessage"></p>

    </div>

**<!-- Registration Form -->**

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

        <h2>Register</h2>

        <input type="text" id="regUsername" placeholder="Username" required />

        <input type="password" id="regPassword" placeholder="Password" required />

        <input type="password" id="confirmPassword" placeholder="Confirm Password" required />

        <button type="button" onclick="registerUser()">Register</button>

        <p id="regMessage"></p>

    </div>

    <p id="toggleLink" onclick="toggleForms()">Don't have an account? Register here</p>

    <script>

**// Toggle between login and registration forms**

        function toggleForms() {

            const loginForm = document.getElementById('loginForm');

            const regForm = document.getElementById('registrationForm');

            loginForm.style.display = loginForm.style.display === 'none' ? 'block' : 'none';

            regForm.style.display = regForm.style.display === 'none' ? 'block' : 'none';

        }

**// Register user (stores data in localStorage)**

        function registerUser() {

            const username = document.getElementById('regUsername').value;

            const password = document.getElementById('regPassword').value;

            const confirmPassword = document.getElementById('confirmPassword').value;

            if (password !== confirmPassword) {

                document.getElementById('regMessage').textContent = "Passwords do not match!";

                document.getElementById('regMessage').classList.add('error');

                return;

            }

**// Check if the user already exists**

            const storedUsers = JSON.parse(localStorage.getItem('users')) || [];

            if (storedUsers.find(user => user.username === username)) {

                document.getElementById('regMessage').textContent = "Username already exists!";

                document.getElementById('regMessage').classList.add('error');

                return;

            }

**// Save the new user (hashed password is for illustration; use basic encryption)**

            storedUsers.push({ username, password });

            localStorage.setItem('users', JSON.stringify(storedUsers));

            document.getElementById('regMessage').textContent = "Registration successful! You can now login.";

            document.getElementById('regMessage').classList.add('success');

            toggleForms(); // Switch to login form

        }

        // Authenticate user (compares with localStorage data)

        function authenticateUser() {

            const username = document.getElementById('loginUsername').value;

            const password = document.getElementById('loginPassword').value;

            const storedUsers = JSON.parse(localStorage.getItem('users')) || [];

**// Find user and check password**

            const user = storedUsers.find(user => user.username === username);

            if (user && user.password === password) {

                document.getElementById('loginMessage').textContent = "Login successful!";

                document.getElementById('loginMessage').classList.add('success');

            } else {

                document.getElementById('loginMessage').textContent = "Invalid username or password!";

                document.getElementById('loginMessage').classList.add('error');

            }

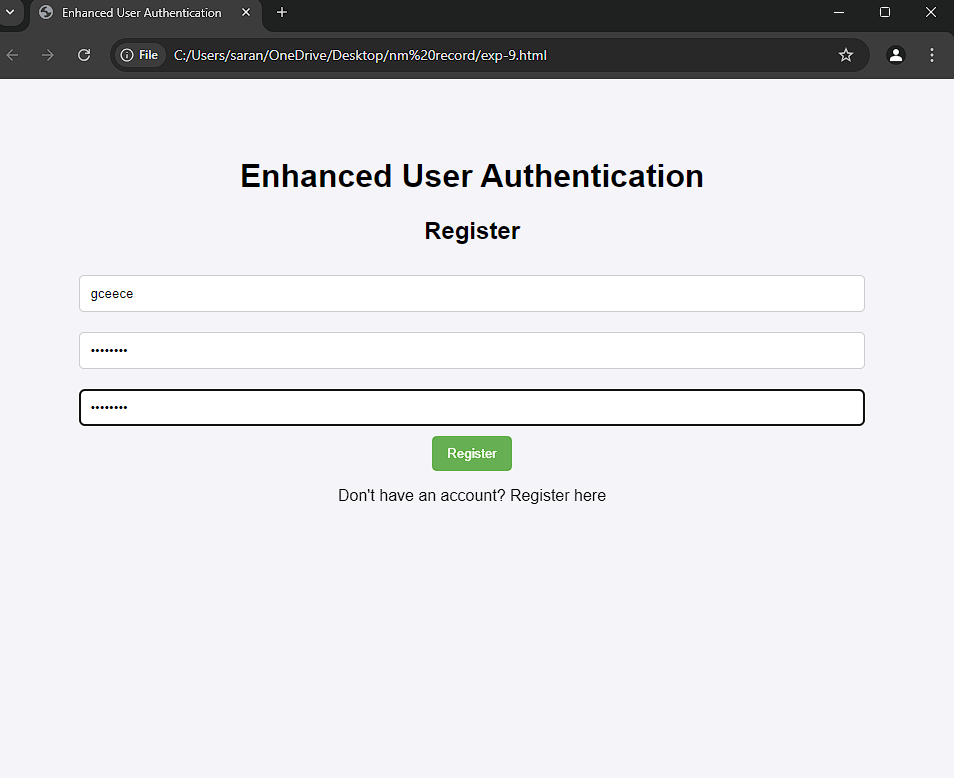
        }

    </script>

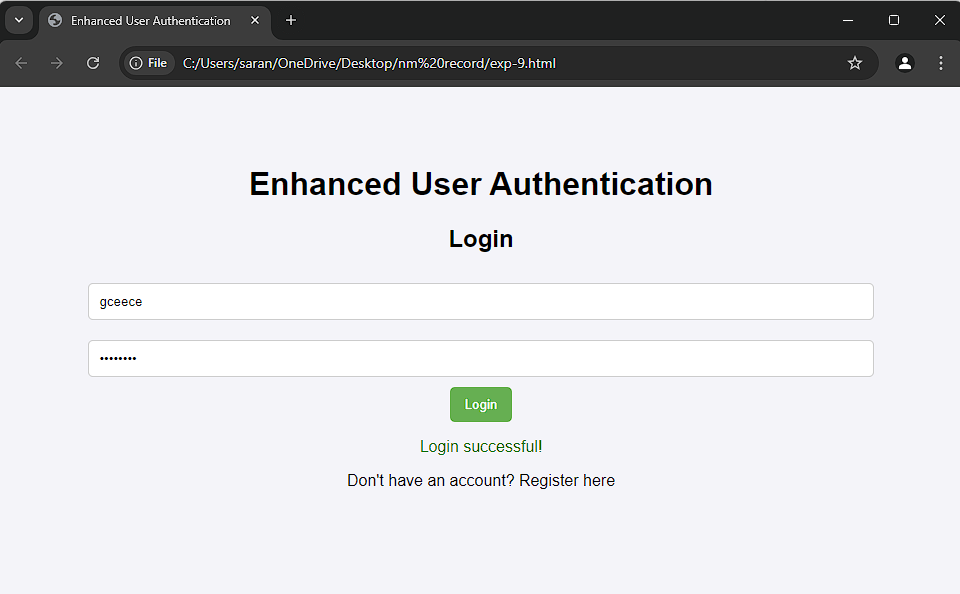
</body>

</html>

**OUTPUT:**

****

**Register Page**

**Login Page**

**RESULT:**

Thus, the program has been executed and output is verified successfullyby check the register and login user authentication System using HTML and JS.

**USING DOCKER FOR CONTAINERIZATION OF APPLICATIONS**

EX.NO:10

DATE:

**AIM:**

To understand and implement containerization using Docker by creating a Python web application, building a Docker image, and running it in a container.

**PROGRAM:**

**Python:** app.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

    return 'Hello, Docker World!'

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Dockerfile**

**# Use an official Python runtime as the base image**

FROM python:3.8-slim

**# Set the working directory**

WORKDIR /app

**# Copy the current directory contents into the container at /app**

COPY . /app

**# Install any needed packages specified in requirements.txt**

RUN pip install --no-cache-dir -r requirements.txt

**# Make port 5000 available to the world outside this container**

EXPOSE 5000

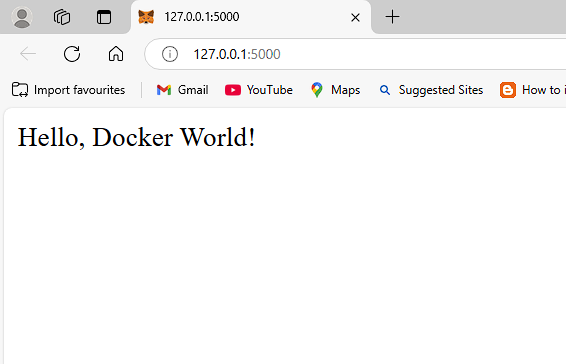
**# Define environment variable**

ENV FLASK\_APP=app.py

**# Run the application when the container launches**

CMD ["flask", "run", "--host", "0.0.0.0"]

**OUTPUT:**

****

**RESULT:**

Thus, the Flask application is successfully running inside a Docker container, accessible via http://localhost:5000 displaying "Hello, Docker World!".

**BUILDING THE BACKEND FOR A REAL-TIME CHAT APPLICATION USING NODE.JS**

EX.NO:11

DATE:

**AIM:**

The aim of this experiment is to develop the backend for a real-time chat application using Node.js and Socket.IO for real-time messaging functionality.

**PROGRAM:**

**HTML File:** index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Real-time Chat Application</title>

    <style>

        body {

            font-family: Arial, sans-serif;

        }

        #chat-box {

            width: 300px;

            height: 300px;

            border: 1px solid #ccc;

            padding: 10px;

            overflow-y: scroll;

            margin-bottom: 10px;

        }

        #chat-input {

            width: 240px;

        }

        #send-button {

            padding: 5px 10px;

            cursor: pointer;

        }

        .user-list {

            margin-bottom: 10px;

        }

        .user-list span {

            display: block;

        }

    </style>

</head>

<body>

    <h1>Real-time Chat Application</h1>

    <div class="user-list">

        <h3>Users:</h3>

        <ul id="users"></ul>

    </div>

    <div id="chat-box"></div>

    <input type="text" id="chat-input" placeholder="Type a message..." />

    <button id="send-button">Send</button>

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

    <script>

        const socket = io();

        // Update the list of users when someone connects/disconnects

        socket.on('users', (users) => {

            const userList = document.getElementById('users');

            userList.innerHTML = ''; // Clear the current list

            users.forEach((user, index) => {

                const userElement = document.createElement('span');

                userElement.textContent = `User ${index + 1}`;

                userList.appendChild(userElement);

            });

        });

        // Listen for incoming messages and display them in the chat box

        socket.on('chat message', (msg) => {

            const chatBox = document.getElementById('chat-box');

            const messageDiv = document.createElement('div');

            messageDiv.textContent = msg;

            chatBox.appendChild(messageDiv);

            chatBox.scrollTop = chatBox.scrollHeight; // Auto-scroll to the latest message

        });

        // Send a message when the send button is clicked

        document.getElementById('send-button').addEventListener('click', () => {

            const message = document.getElementById('chat-input').value;

            if (message.trim()) {

                socket.emit('chat message', message);

                document.getElementById('chat-input').value = ''; // Clear the input field

            }

        });

        // Allow pressing "Enter" to send the message

        document.getElementById('chat-input').addEventListener('keypress', (e) => {

            if (e.key === 'Enter') {

                document.getElementById('send-button').click();

            }

        });

    </script>

</body>

</html>

**JavaScript Fie:** server.js

// server.js

const express = require('express');

const http = require('http');

const socketIo = require('socket.io');

// Initialize the app and HTTP server

const app = express();

const server = http.createServer(app);

const io = socketIo(server);

// Serve the static HTML file for the client-side

app.use(express.static('public'));

// Array to keep track of connected users

let users = [];

// Listen for connections from clients

io.on('connection', (socket) => {

    console.log('A user connected');

    // Add the user to the users array

    users.push(socket.id);

    // Emit the current list of users to all clients

    io.emit('users', users);

    // Listen for chat messages from clients

    socket.on('chat message', (msg) => {

        // Broadcast the message to all other users

        io.emit('chat message', msg);

    });

    // Handle user disconnect

    socket.on('disconnect', () => {

        console.log('A user disconnected');

        // Remove the user from the users array

        users = users.filter(user => user !== socket.id);

        io.emit('users', users);

    });

});

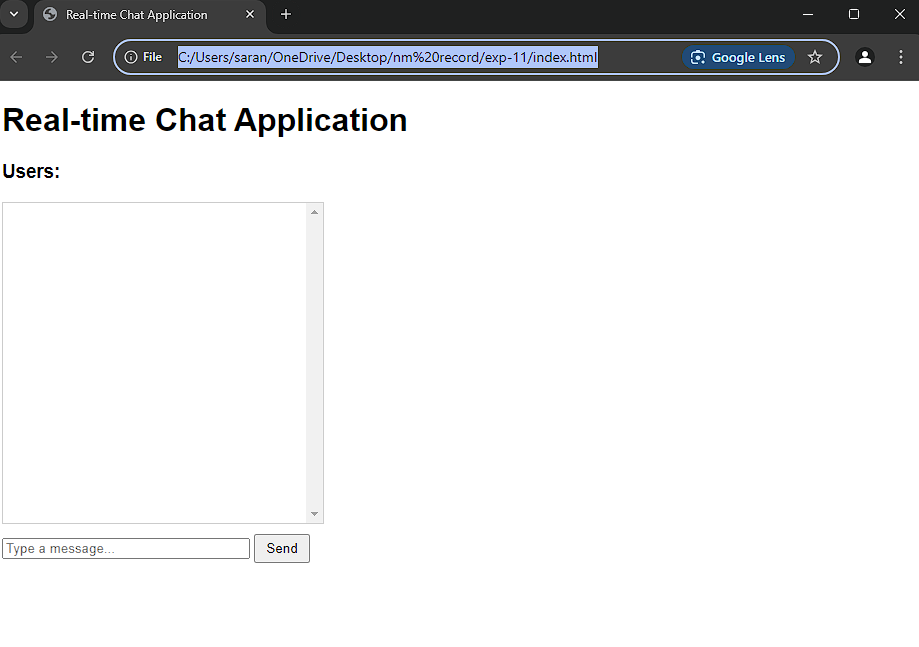
// Start the server on port 3000

server.listen(3000, () => {

    console.log('Server is running on http://localhost:3000');

});

**OUTPUT:**

****

**RESULT:**

Thus, the chat application has been successfully verified and is able to send and receive messages simultaneously in real-time