**Exercise 9**

**9. Dynamic website using Node JS and Express**

**Aim:**

To learn to implement a dynamic website using NodeJS and Express

**Algorithm:**

1. Start
2. Import express with require keyword
3. Import ejs with require keyword
4. Create an app by calling the express()function provided by the express framework.
5. Tell the express server to use EJS using app.use() and set ejs as view engine using app.set()
6. With app.get() we are configuring our first route in our website with req and res and render a file using res.render().
7. Repeat step 4 to create multiple routes of a website
8. Stop

**Code:**

index.js

const express = require('express')

const app = express()

const port = 3000

app.set('view engine', 'ejs')

app.use(express.urlencoded({ extended: true }))

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

let items = [

{ id: 1, name: 'Item 1', price: 10 ,desc:'Item 1 has a price of 10 and is of poor quality'},

{ id: 2, name: 'Item 2', price: 20 ,desc:'Item 2 has a price of 20 and is of medium quality'},

{ id: 3, name: 'Item 3', price: 30 ,desc:'Item 3 has a price of 30 and is of high quality'},

]

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

res.render('home', { items })

})

app.get('/items/:id', (req, res) => {

const id = parseInt(req.params.id)

const item = items.find(item => item.id === id)

if (item) {

res.render('item', { item })

} else {

res.status(404).send('Item not found')

}

})

app.post('/items', (req, res) => {

const { name, price ,desc} = req.body

const id = items.length + 1

const item = { id, name, price: parseInt(price) ,desc}

items.push(item)

res.redirect('/')

})

app.listen(port, () => {

console.log(`Example app listening at http://localhost:${port}`)

})

Home.ejs

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Home</title>

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

</head>

<body>

<h1>Items</h1>

<ul>

<% items.forEach(item => { %>

<li><a href="/items/<%= item.id %>"><%= item.name %> - <%= item.price %></a></li>

<% }) %>

</ul>

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

<div>

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

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

</div>

<div>

<label for="price">Price:</label>

<input type="number" id="price" name="price">

</div>

<div>

<label for="desc">Description:</label>

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

</div>

<button type="submit">Add item</button>

</form>

</body>

</html>

<style>

body {

font-family: Arial, sans-serif;

font-size: 16px;

line-height: 1.5;

margin: 20px;

padding: 0;

}

ul {

list-style: none;

margin: 0;

padding: 0;

}

li {

margin-bottom: 10px;

}

form {

margin-top: 20px;

}

label {

display: inline-block;

margin-bottom: 5px;

}

input[type="text"],

input[type="number"] {

border: 1px solid #ccc;

border-radius: 3px;

padding: 5px;

margin-bottom: 10px;

}

button[type="submit"] {

background-color: #007bff;

color: #fff;

border: none;

border-radius: 3px;

padding: 10px 15px;

cursor: pointer;

}

button[type="submit"]:hover {

background-color: #0069d9;

}

</style>

item.ejs

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title><%= item.name %></title>

</head>

<body>

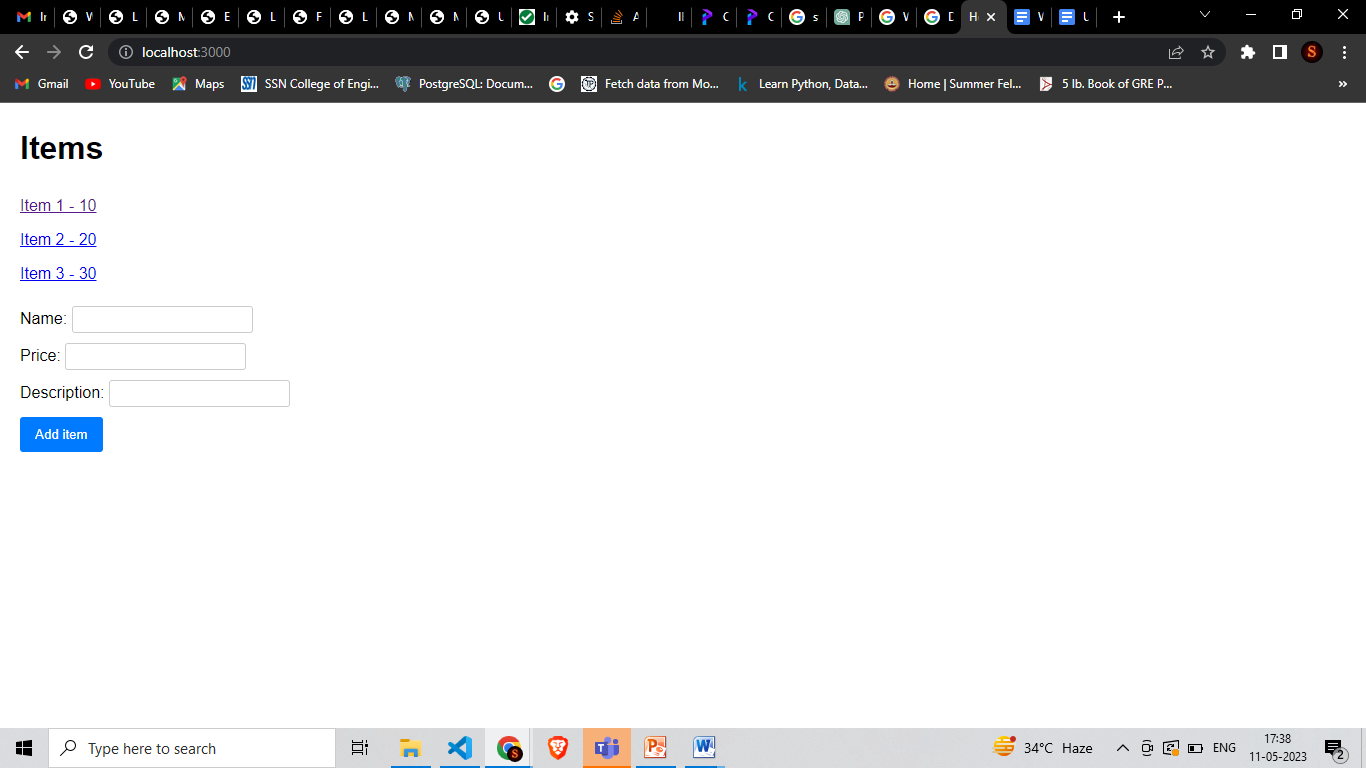
<h1><%= item.name %> - <%= item.price %></h1>

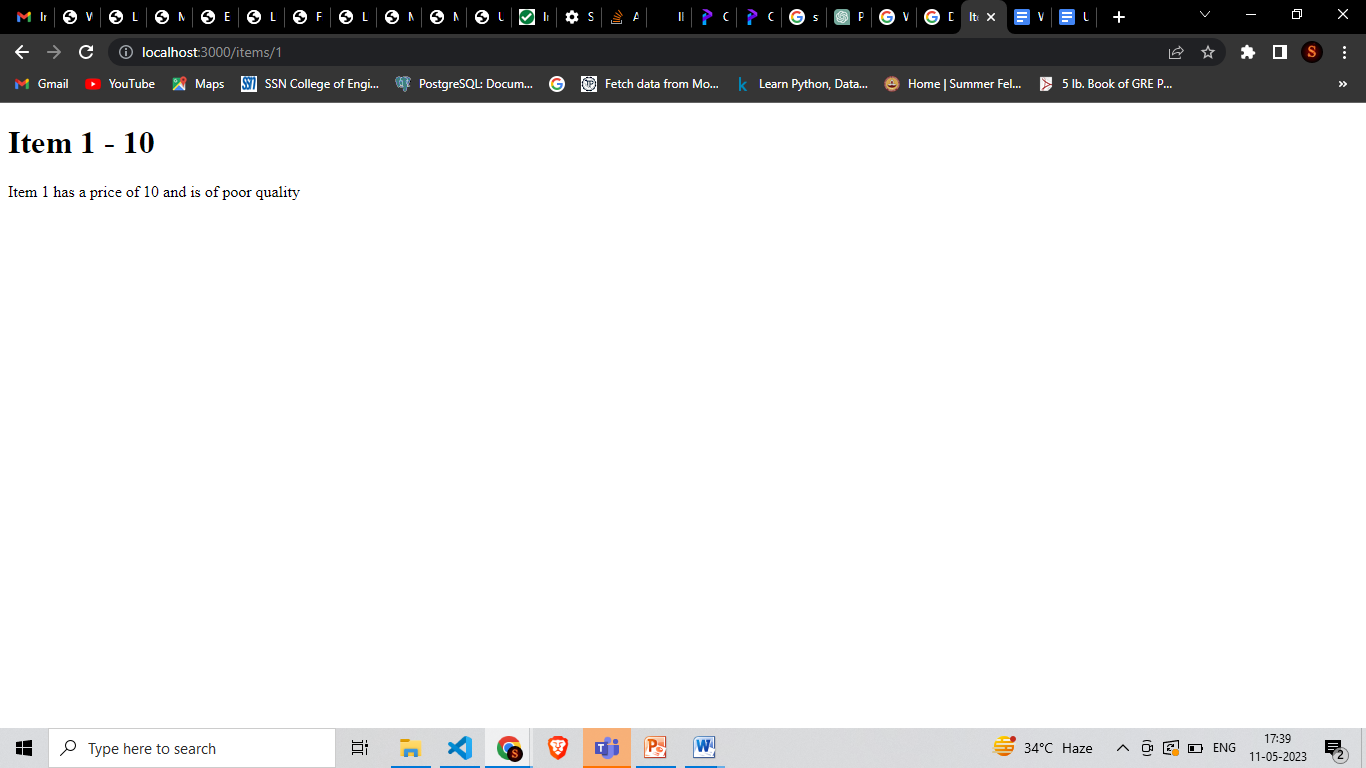
<p><%= item.desc %></p>

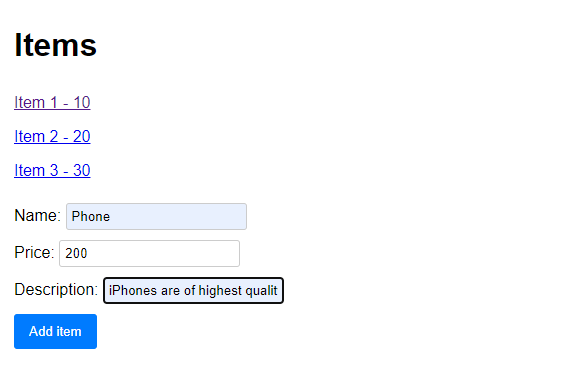
</body>

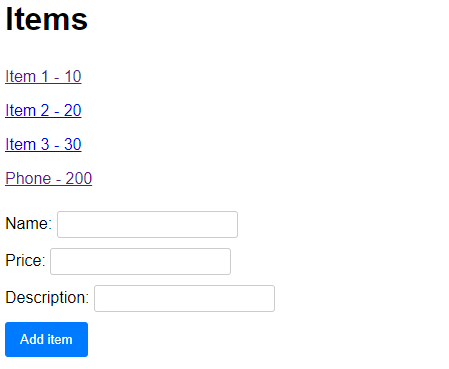
</html>

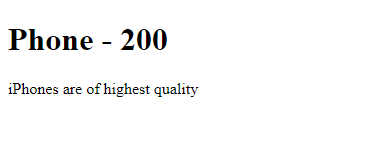
**Output:**

****

****

****

****

****

**Result:**

**Exercise 10**

**10. Simple web application using MEAN Stack**

**Aim:**

To learn to implement a simple web application using MEAN Stack

**Algorithm:**

1. Create a homepage called index.html
2. Create another page for contact.html which contains the form Contact Us.
3. Use angular to perform routing from home page to contact page.
4. Use express and node to connect to mongodb
5. From the form, get form details using express.
6. Store the form details in mongodb

**Code:**

index.html

<!DOCTYPE html>

<html lang="en">

<head>

<base href="/">

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

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

<title>Sporty!</title>

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

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

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

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>

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

</head>

<body ng-app="myApp">

<nav id="header">

<a href="#" ng-click="count=1"><div id="logo"><img src="images/logo.png"></div></a>

<!-- <h3><a href="#">SPORTY</a></h3> -->

<ul>

<li><a href="#!contact" ng-click="count=0">Contact Us</a></li>

<li><a href="#!sell" ng-click="count=0">Sell equipment</a></li>

<li><a href="#!search" ng-click="count=0">Buy equipment</a></li>

</ul>

</nav>

<div ng-view="!count"></div>

<div ng-show="count">

<div class="slideshow-container">

<!-- Full-width images with number and caption text -->

<div class="mySlides fade">

<img src="images/bg2.jpg" style="width:100%">

<div class="text">Buy or sell any equipment</div>

</div>

<div class="mySlides fade">

<video width="100%" controls autoplay muted>

<source src="images/video.mp4" type="video/mp4">

Your browser does not support the video element.

</video>

</div>

<div class="mySlides fade">

<img src="images/bg3.jpg" style="width:100%">

<div class="text">Verified products</div>

</div>

<!-- Next and previous buttons -->

<a class="prev" onclick="plusSlides(-1)">&#10094;</a>

<a class="next" onclick="plusSlides(1)">&#10095;</a>

</div>

<br>

<!-- The dots/circles -->

<div style="text-align:center">

<span class="dot" onclick="currentSlide(1)"></span>

<span class="dot" onclick="currentSlide(2)"></span>

<span class="dot" onclick="currentSlide(3)"></span>

</div>

<!-- <div id="logo"><img src="bg1.jfif"></div> -->

<div class="grid-container">

<div class="card">

<img src="images/indoor.jpg" alt="Card Image">

<h2>Indoor sports</h2>

<p>Chess, Table Tennis, Billards equipment are more..</p>

<a href="#" class="button">Read More</a>

</div>

<div class="card">

<img src="images/outdoor.jpg" alt="Card Image" height="200">

<h2>Outdoor sports</h2>

<p>Cricket, Football, Tennis, Volleyball and more..</p>

<a href="#" class="button">Read More</a>

</div>

<div class="card">

<img src="images/gym.jpg" alt="Card Image">

<h2>Fitness Equipment</h2>

<p>Gym equipment for dumbell, treadmill and others..</p>

<a href="#" class="button">Read More</a>

</div>

</div>

<div id="review">

Listen to a satisfied customer

<audio controls>

<source src="images/audio.mp3" type="audio/mpeg">

Your browser does not support the audio element.

</audio>

</div>

</div>

</body>

</html>

<script>

let slideIndex = 1;

carousel();

function carousel() {

var i;

var x = document.getElementsByClassName("mySlides");

for (i = 0; i < x.length; i++) {

x[i].style.display = "none";

}

slideIndex++;

if (slideIndex > x.length) {slideIndex = 1}

x[slideIndex-1].style.display = "block";

setTimeout(carousel, 10000); // Change image every 2 seconds

}

showSlides(slideIndex);

// Next/previous controls

function plusSlides(n) {

showSlides(slideIndex += n);

}

// Thumbnail image controls

function currentSlide(n) {

showSlides(slideIndex = n);

}

function showSlides(n) {

let i;

let slides = document.getElementsByClassName("mySlides");

let dots = document.getElementsByClassName("dot");

if (n > slides.length) {slideIndex = 1}

if (n < 1) {slideIndex = slides.length}

for (i = 0; i < slides.length; i++) {

slides[i].style.display = "none";

}

for (i = 0; i < dots.length; i++) {

dots[i].className = dots[i].className.replace(" active", "");

}

slides[slideIndex-1].style.display = "block";

dots[slideIndex-1].className += " active";

}

</script>

Contact.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

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

<title>Sporty!</title>

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

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

<!-- <script src="contactus.js"></script> -->

</head>

<body>

<form id="contactForm" action="/success" method="post">

<h2>Contact Us</h2>

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

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

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

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

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

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

<input type="submit" value="Submit" class="button" style="font-size: 15px;" onclick="validate()">

<!-- <button type="submit" class="button" style="font-size: 15px;">Submit</button> -->

</form>

</body>

<style>

form {

background-color: #f2f2f2;

border-radius: 5px;

padding: 20px;

max-width: 500px;

margin: auto;

font-family: Arial, sans-serif;

font-size: 16px;

line-height: 1.5;

}

label {

display: block;

margin-bottom: 10px;

font-weight: bold;

}

input[type="text"],

input[type="email"],

textarea {

width: 100%;

padding: 10px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

margin-bottom: 20px;

font-size: 16px;

}

</style>

Routing.js

const app = angular.module("myApp", ["ngRoute"]);

app.config(function ($routeProvider) {

$routeProvider

.when("/search", {

templateUrl:"search.html",

controller:"appcontroller"

})

.when("/contact", {

templateUrl: "contact.html"

})

});

App.js

// Load express module

const express = require('express');

const ejs = require('ejs');

const bodyParser = require('body-parser');

const nodemailer = require('nodemailer');

// Initialize app

const app = express();

// app.set('view engine', 'ejs');

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({

extended: true

}));

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

const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1:27017/testdb');

const db = mongoose.connection;

const Contact = require('./contact\_schema');

console.log(Contact)

// Check for DB connection

db.once('open', function(){

console.log("Connected to MongoDB successfully!");

});

db.on('error', function(err){

console.log(err);

});

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

res.sendFile(\_\_dirname + '/index.html');

});

app.post('/success', (req, res) => {

const data = new Contact({

name: req.body.name,

email: req.body.email,

message: req.body.message

});

console.log(data);

data.save()

.then(() => {

console.log('Message saved successfully!');

})

.catch(err => {

console.log(err);

});

res.send('Message saved successfully!<br>Message:<br>Name:', data.name,'<br>Email:',data.email,'<br>Message:',data.message);

});

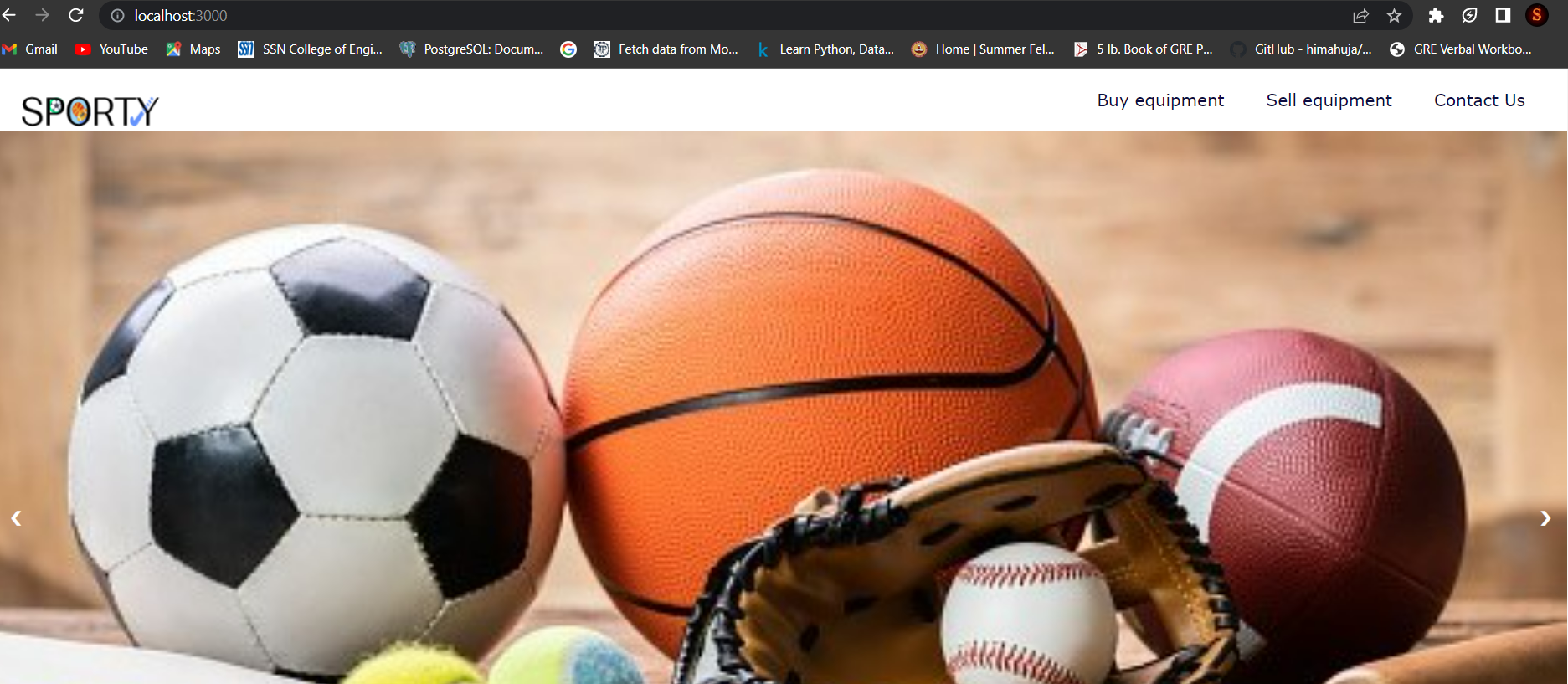
// Start server with port 3000

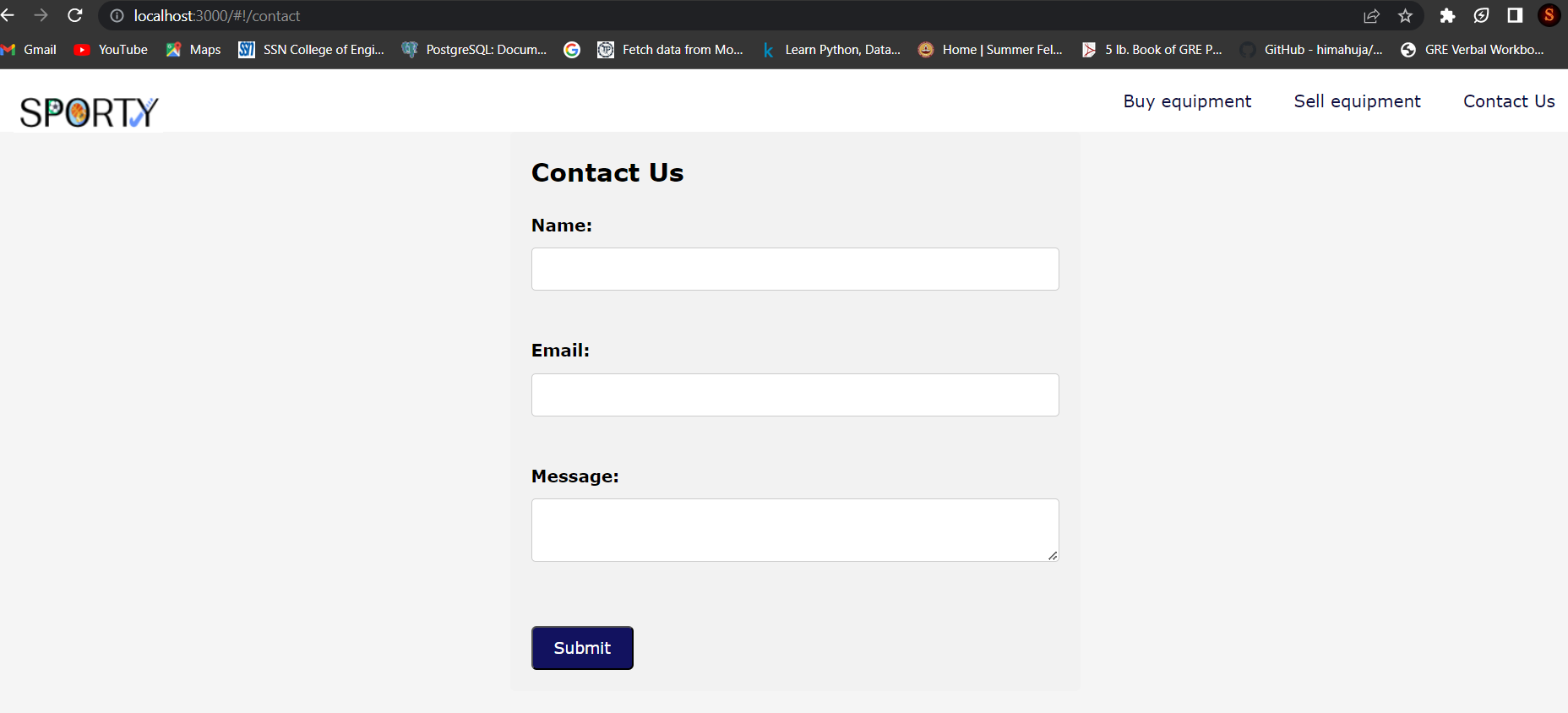
app.listen(3000, function(){

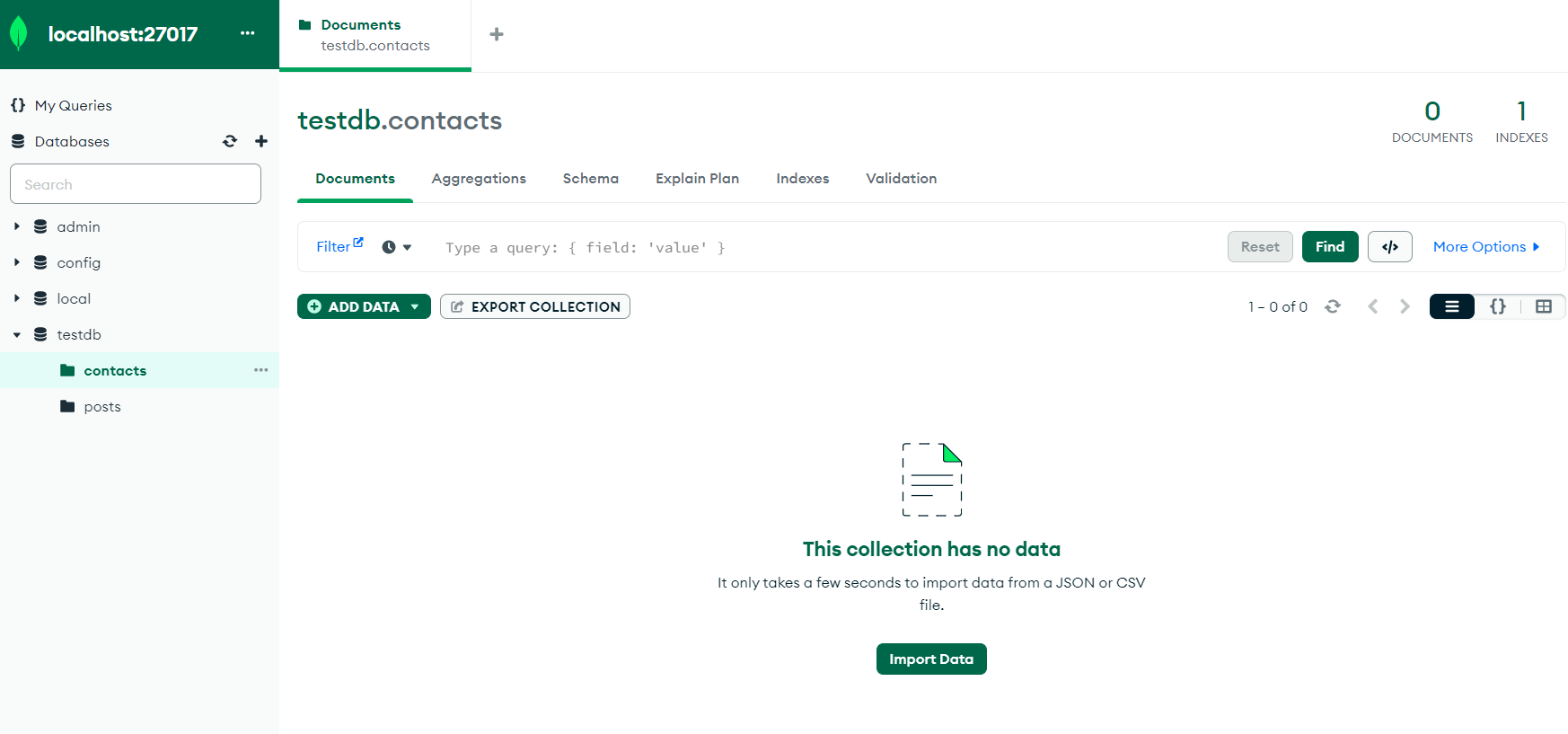
console.log("Server started on localhost:3000");

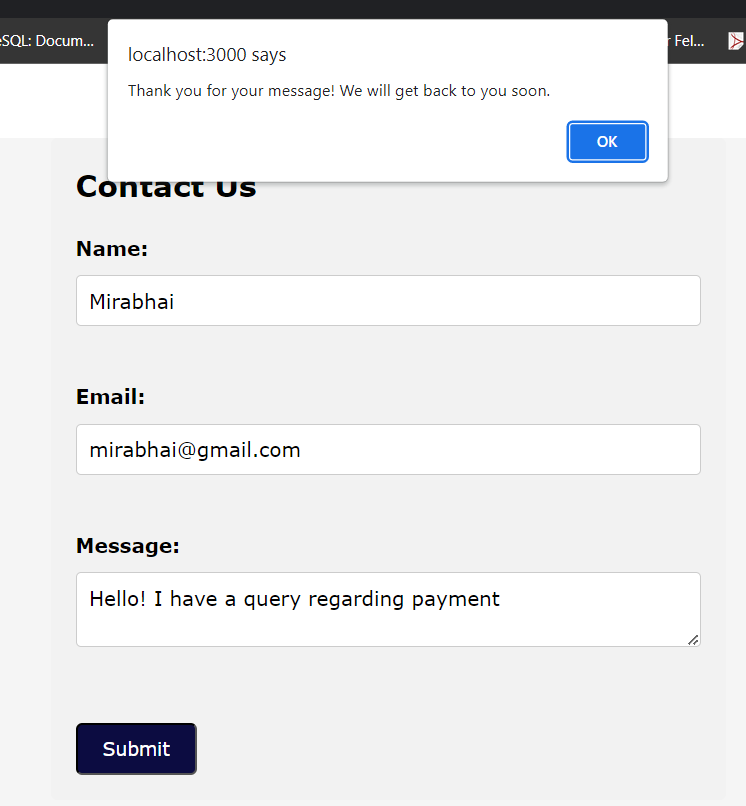
});

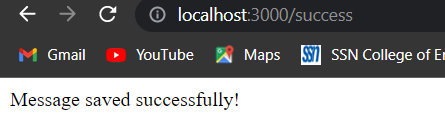
**Output:**

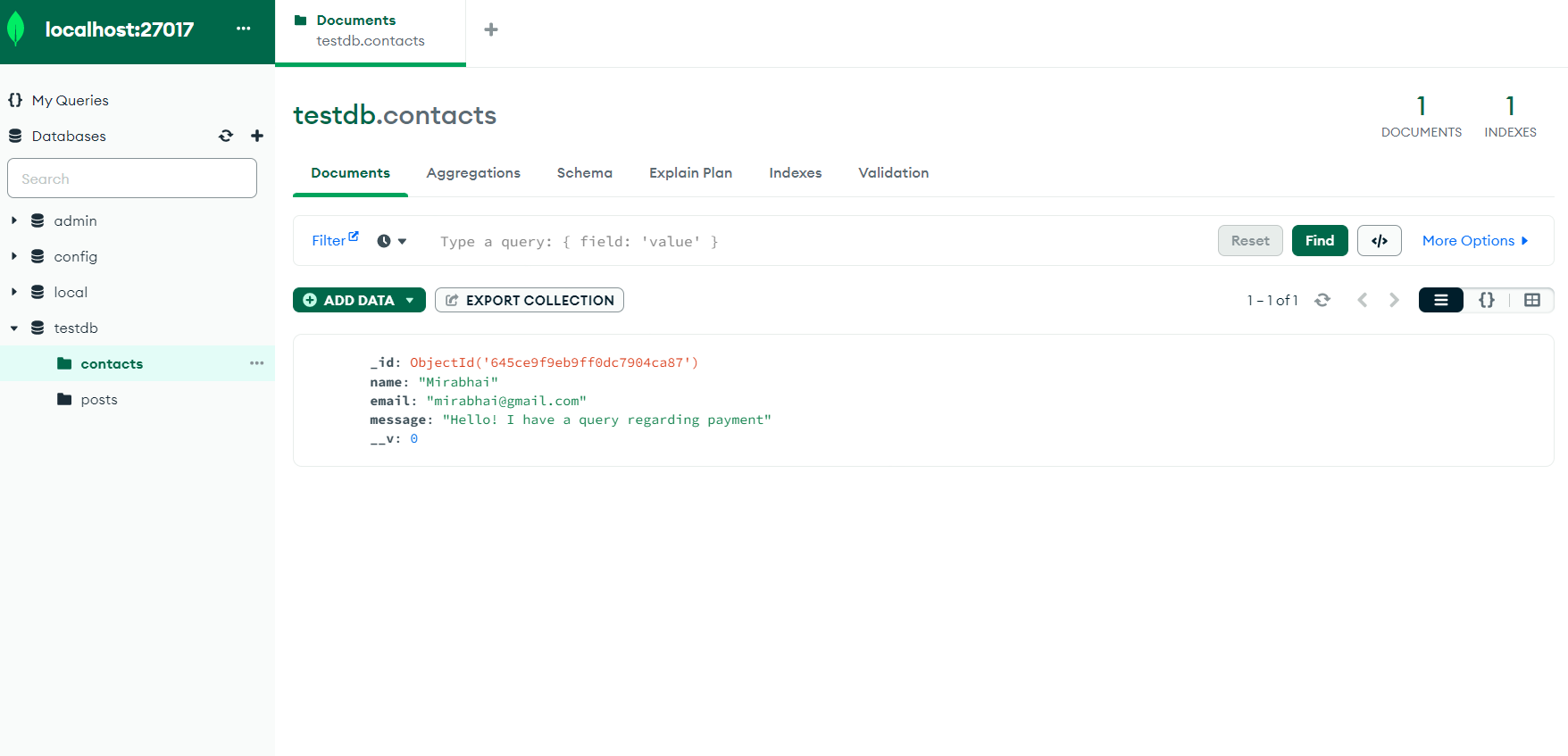
****

****

****

****

****

****

**Result:**

**Exercise 11**

**11. Chat application with multiple rooms using Node and Express**

**Aim:**

To implement a multi-room chat application using Node and Express frameworks

**Algorithm:**

1. We will be using Node.js for the back end, which will be responsible for creating rooms and managing users in those rooms.
2. Additionally, we will be using Socket.io to enable real-time, bidirectional communication between the web server and the client (browser).
3. Create app.js file and include the above packages to our code. Also, set other options like body-parser, view engine, public directory, and port address.
4. Create a route to render index.ejs and bind the app to listen to the specified port.
5. Type node app.js in the terminal to make sure it is working properly. After you see the message “Server is running on port 3000,” open the browser and go to http://localhost:3000 to check if the app is running
6. In index.ejs, the user is presented with a form where they can enter their name and a room name. The user name and room name submitted will be passed to the route room and will render room.ejs.
7. The room route will pass it to URL params by parsing URL with query string middleware. We will be fetching the user name and room name from the client side to enable communication.
8. For handling Socket on the client side, create a file chat.js(inside the public folder). Include scripts in room.ejs to import the Socket library and chat.js.
9. In chat.js: when the user writes a message and clicks send, the room name, message, and the name of the user are sent to the server. It is also receiving messages from the server about other user's messages and current online users, and displaying them to the HTML page.
10. For handling Socket on the server side, create a file socket.js(inside the utils folder). This will be responsible for receiving user messages, room names, and user names. Then the messages will then be sent to the respective rooms, depending on the room name.
11. Create a file called getUsers.js(inside the utils folder). users is an object which stores the users currently online in memory. getUsers is a function for storing users with their room names in users object whenever a new user connects.

CODE:

app.js:

const express = require('express');

const bodyParser = require('body-parser');

const socket = require('socket.io')

const app = express();

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

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

app.set('view engine', 'ejs');

var port = process.env.PORT || 3000;

//Render Index page

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

res.render('index')

})

//Start Server

const server = app.listen(port, () => {

console.log(`Server Running on port ${port}`)

})

//Get username and roomname from form and pass it to room

app.post('/room', (req, res) => {

roomname = req.body.roomname;

username = req.body.username;

res.redirect(`/room?username=${username}&roomname=${roomname}`)

})

//Rooms

app.get('/room', (req, res)=>{

res.render('room')

})

const io = socket(server);

require('./utils/socket')(io);

chat.js:

//To get HTML elements

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

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

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

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

const roomMessage = document.querySelector('.room-message');

const users = document.querySelector('.users');

//Socket server URL

const socket = io.connect('http://localhost:3000');

//Fetch URL Params from URL

const queryString = window.location.search;

const urlParams = new URLSearchParams(queryString);

const username = urlParams.get('username');

const roomname = urlParams.get('roomname');

console.log(username, roomname);

//Display the roomname the user is connected to

roomMessage.innerHTML = `Connected in room ${roomname}`

//Emitting username and roomname of newly joined user to server

socket.emit('joined-user', {

username: username,

roomname: roomname

})

//Sending data when user clicks send

send.addEventListener('click', () =>{

socket.emit('chat', {

username: username,

message: message.value,

roomname: roomname

})

message.value = '';

})

//Sending username if the user is typing

message.addEventListener('keypress', () => {

socket.emit('typing', {username: username, roomname: roomname})

})

//Displaying if new user has joined the room

socket.on('joined-user', (data)=>{

output.innerHTML += '<p>--> <strong><em>' + data.username + ' </strong>has Joined the Room</em></p>';

})

//Displaying the message sent from user

socket.on('chat', (data) => {

output.innerHTML += '<p><strong>' + data.username + '</strong>: ' + data.message + '</p>';

feedback.innerHTML = '';

document.querySelector('.chat-message').scrollTop = document.querySelector('.chat-message').scrollHeight

})

//Displaying if a user is typing

socket.on('typing', (user) => {

feedback.innerHTML = '<p><em>' + user + ' is typing...</em></p>';

})

//Displaying online users

socket.on('online-users', (data) =>{

users.innerHTML = ''

data.forEach(user => {

users.innerHTML += `<p>${user}</p>`

});

})

getUsers.js:

//Store connected Users

var users = {}

//Funtion to get users online in a room

function getUsers(arr){

onlineUsers = []

arr.forEach((onlineUser) => {

onlineUsers.push(Object.values(onlineUser)[0])

})

return onlineUsers

}

module.exports = {getUsers, users};

socket.js:

const {getUsers, users} = require('./getUsers');

//Socket connection

function socket(io) {

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

socket.on('joined-user', (data) =>{

//Storing users connected in a room in memory

var user = {};

user[socket.id] = data.username;

if(users[data.roomname]){

users[data.roomname].push(user);

}

else{

users[data.roomname] = [user];

}

//Joining the Socket Room

socket.join(data.roomname);

//Emitting New Username to Clients

io.to(data.roomname).emit('joined-user', {username: data.username});

//Send online users array

io.to(data.roomname).emit('online-users', getUsers(users[data.roomname]))

})

//Emitting messages to Clients

socket.on('chat', (data) =>{

io.to(data.roomname).emit('chat', {username: data.username, message: data.message});

})

//Broadcasting the user who is typing

socket.on('typing', (data) => {

socket.broadcast.to(data.roomname).emit('typing', data.username)

})

//Remove user from memory when they disconnect

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

var rooms = Object.keys(socket.rooms);

var socketId = rooms[0];

var roomname = rooms[1];

users[roomname].forEach((user, index) => {

if(user[socketId]){

users[roomname].splice(index, 1)

}

});

//Send online users array

io.to(roomname).emit('online-users', getUsers(users[roomname]))

})

})

}

module.exports = socket;

index.ejs:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Chat App</title>

</head>

<body>

<div class="content">

<div>

<p>Welcome to Chat App</p>

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

<input type="text" placeholder="Enter your Name" class="username" name="username" required>

<input type="text" placeholder="Enter Room Name" class="roomname" name="roomname" required>

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

</form>

</div>

</div>

</body>

</html>

room.ejs:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Chat APP</title>

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

<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.3.1/socket.io.js"></script>

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

</head>

<body>

<h1 class="room-message"></h1>

<div class="window">

<div class="chat-message">

<div id="output"></div>

<div id="feedback"></div>

</div>

<div class='fields'>

<input type="text" id="message" placeholder="Enter message">

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

</div>

</div>

<div class="online">

<p class="users-online">Users Online</p>

<div class="users">

</div>

</div>

</body>

</html>

style.css:

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: rgb(231, 216, 228);

}

/\* -------------- room.ejs -------------- \*/

h1 {

text-align: center;

margin-top: 15px;

}

input {

width: 100%;

height: 40px;

text-indent: 10px;

}

button {

width: 100%;

height: 50px;

background-color: #1a2eda;

color: white;

font-size: larger;

font-weight: bold;

outline: none;

cursor: pointer;

border: none;

}

.fields {

width: 100%;

}

.window {

margin: auto;

margin-top: 20px;

width: 700px;

height: 500px;

border: 2px solid #c2c6b6;

display: flex;

flex-direction: column;

align-content: space-between;

}

.chat-message {

display: block;

width: 100%;

height: 100%;

overflow: auto;

}

#output, #feedback {

margin-left: 10px;

}

.online {

height: 100vh;

width: 300px;

position: absolute;

left: 0px;

top: 0px;

background-color: rgb(110, 103, 103);

}

.online .users-online {

background-color: rgb(110, 103, 103);

color: white;

text-align: center;

font-size: 40px;

font-weight: bolder;

}

.users {

margin-left: 40px;

margin-top: 50px;

}

.users p {

background-color: rgb(110, 103, 103);

color: white;

}

/\* -------------- index.ejs -------------- \*/

.content {

height: 100vh;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

}

.content p {

font-size: 40px;

font-weight: 700;

margin-bottom: 40px;

}

.content input {

margin-bottom: 20px;

height: 50px;

border: 1px solid black;

border-radius: 7px;

}

.content input[type="submit"] {

cursor: pointer;

}

::placeholder {

font-weight: 700;

}

.content div {

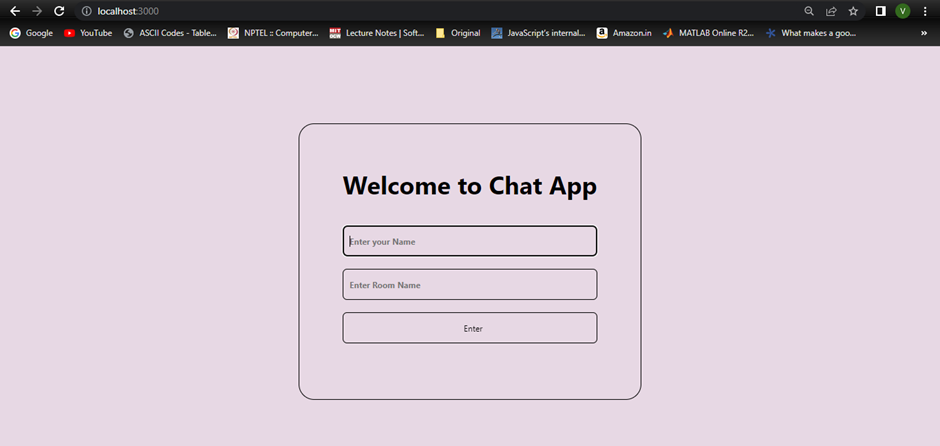
border: 2px solid black;

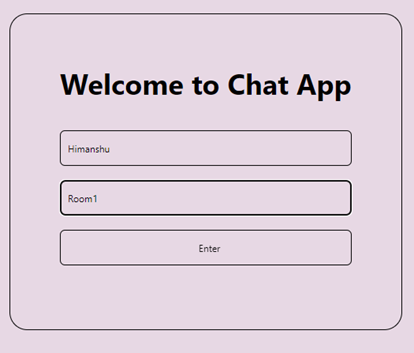
padding: 70px;

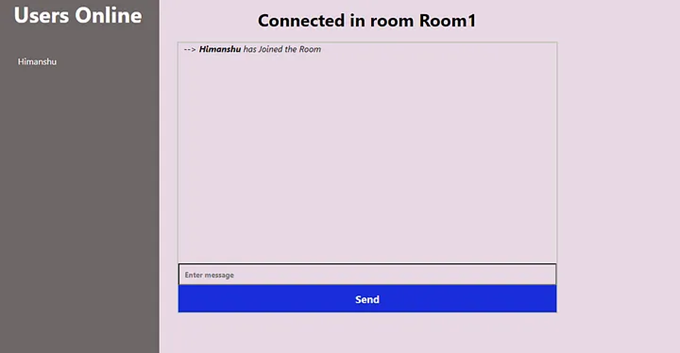
border-radius: 25px;

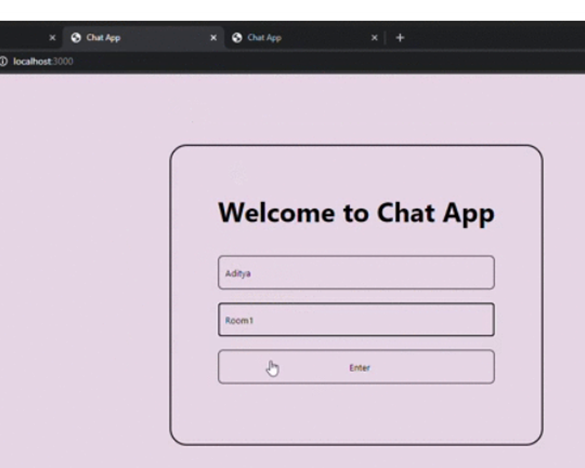
}

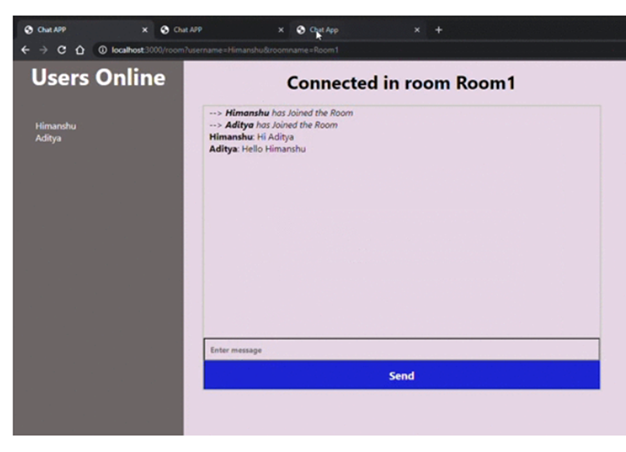
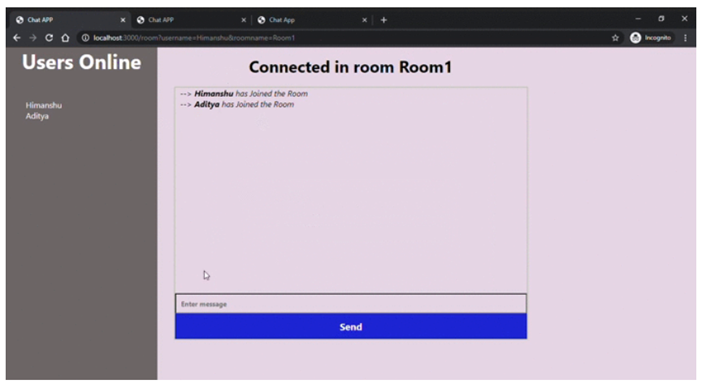
OUTPUT:











**Result:**