**FULL STACK DEVELOPMENT**

**LAB MANUAL**

**B.Tech VI Semester**

**(R22 REGULATION)**

**COMPUTER SCIENCE AND ENGINEERING**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**VAAGDEVI COLLEGE OF ENGINEERING**

**BOLLIKUNTA, WARANGAL URBAN, TELANGANA-506005**

**List of Experiments**

**1. Create an application to setup node JS environment and display “Hello World”.**

**2. Create a Node JS application for user login system.**

**3. Write a Node JS program to perform read, write and other operations on a file.**

**4. Write a Node JS program to read form data from query string and generate response using NodeJS**

**5. Create a food delivery website where users can order food from a particular restaurant listed inthe website for handling http requests and responses using NodeJS.**

**6. Implement a program with basic commands on databases and collections using MongoDB.**

**7. Implement CRUD operations on the given dataset using MongoDB.**

**8. Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB.**

**9. Develop an angular JS form to apply CSS and Events.**

**10. Develop a Job Registration form and validate it using angular JS.**

**11. Write an angular JS application to access JSON file data of an employee from a server using $http service.**

**12. Develop a web application to manage student information using Express and Angular JS.**

**13. Write a program to create a simple calculator Application using React JS.**

**14. Write a program to create a voting application using React JS**

**15. Develop a leave management system for an organization where users can apply different types of leaves such as casual leave and medical leave. They also can view the available number of days using react application.**

**16. Build a music store application using react components and provide routing among the web pages.**

**17. Create a react application for an online store which consist of registration, login, product information pages and implements routing to navigate through these pages**

**WEEK-1: Create an application to setup node JS environment and display “Hello World”.**

Step1: Open Visual Studio Code

Step 2: Create a folder called **Hello-World** and create a java script file (hello.js) in your folder

**Filename**: hello.js

**// Code**

console.log (“Hello World”)

**Output:**

**Hello World**

**WEEK 2: Create a Node JS application for user login system.**

We'll create a simple user login system where a user can submit a username and password, and the system will authenticate them based on predefined credentials.

**Step 1: Create your project Folder:**

Create a folder where you’ll put your Node.js application. Create a folder called **Login-System**

**Step 2: Install necessary dependencies:**

Initialize a Node.js Project: Inside your project folder, initialize a new Node.js project by running:

PS E:\Login-System>**npm init –y**

-You'll need `express` for handling HTTP requests and `body-parser `for parsing request data.

PS E:\Login-System> npm install express body-parser

**Step 3: Create the login system:**

Create a java script file in your folder

File name: **server.js**

**//Code**

const express = require('express');

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

const app = express();

const port = 3005;

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

// Predefined users (In real-world applications, this would come from a database)

const users = {

username: 'admin',

password: 'password123'

};

// Serve the login form

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

res.send(`

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

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

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

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

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

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

</form>

`);

});

// Handle login POST request

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

const { username, password } = req.body;

if (username === users.username && password === users.password) {

res.send('Login successful!');

} else {

res.send('Invalid credentials');

}

});

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

**OUTPUT:**

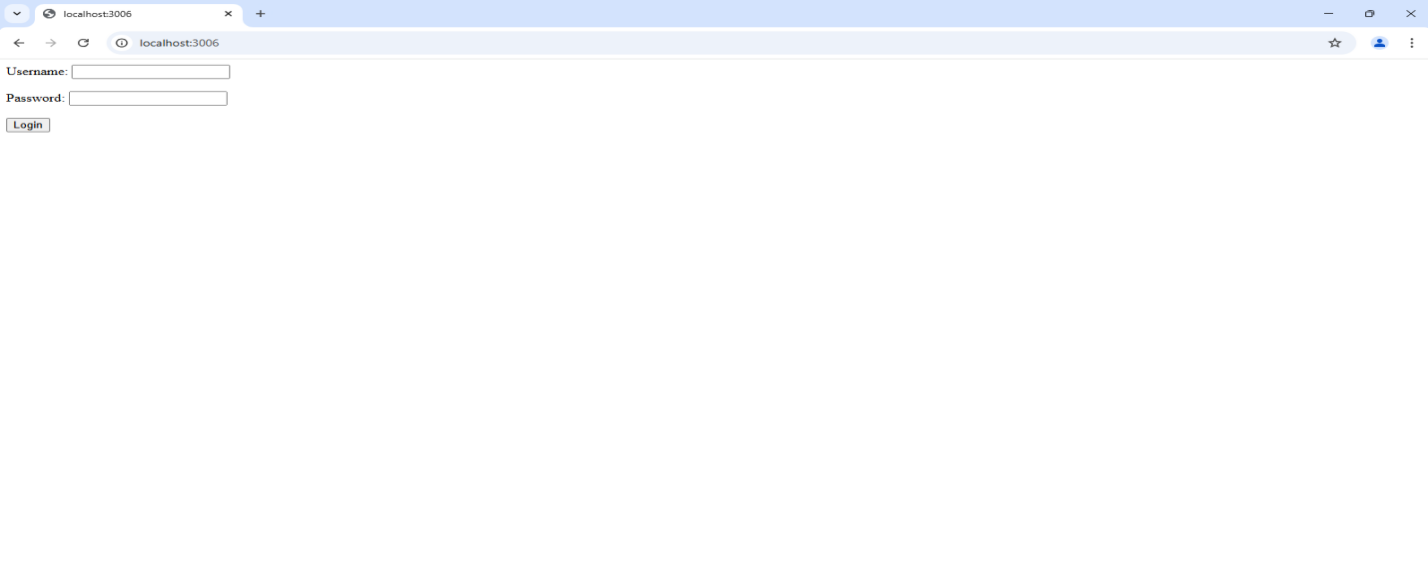
PS E:\FSD\Login-System>**node server.js**

Server is running at <http://localhost:3005>

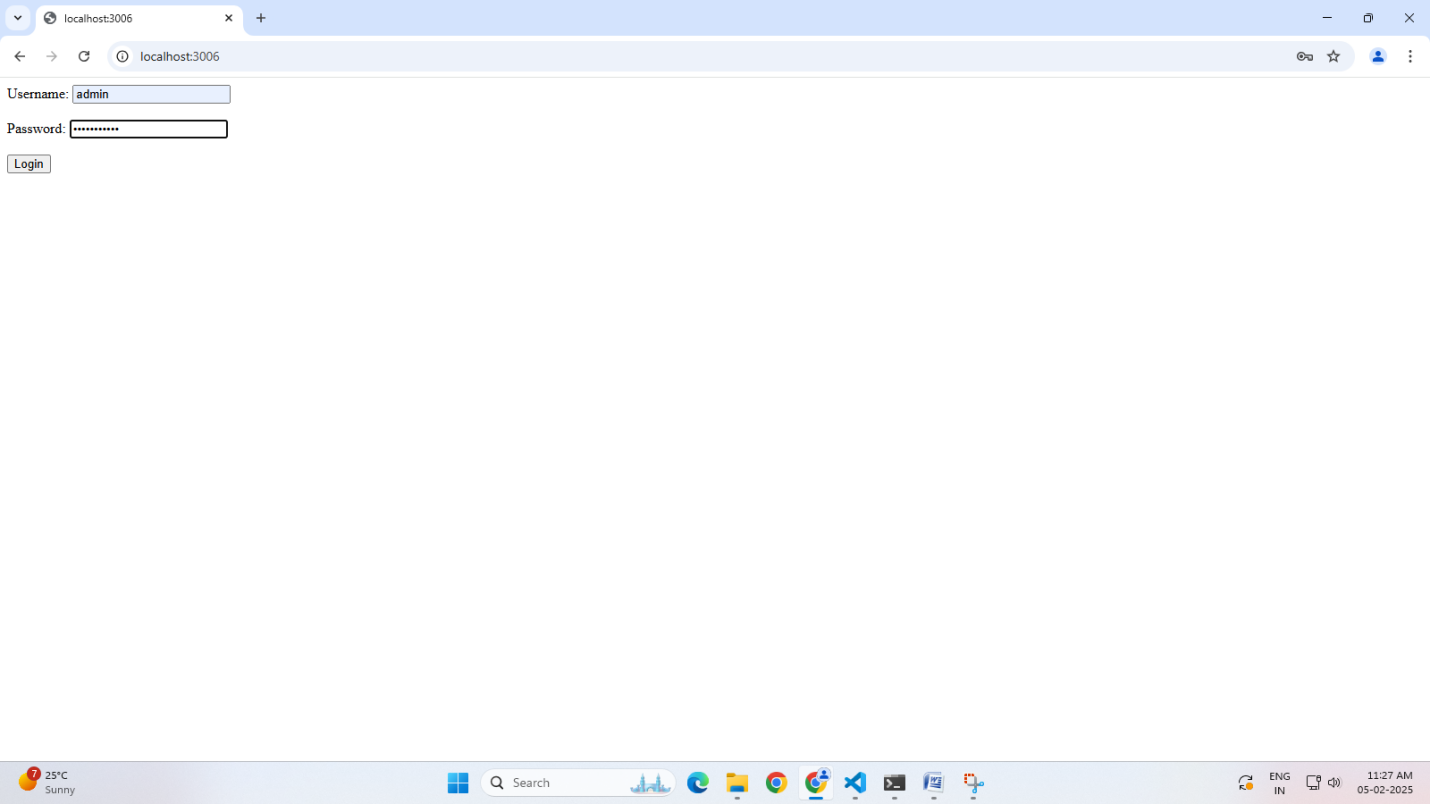
Open Your Browser: Open any browser (Google Chrome,Firefox,etc)

In the address bar, type <http://localhost:3005> and press Enter

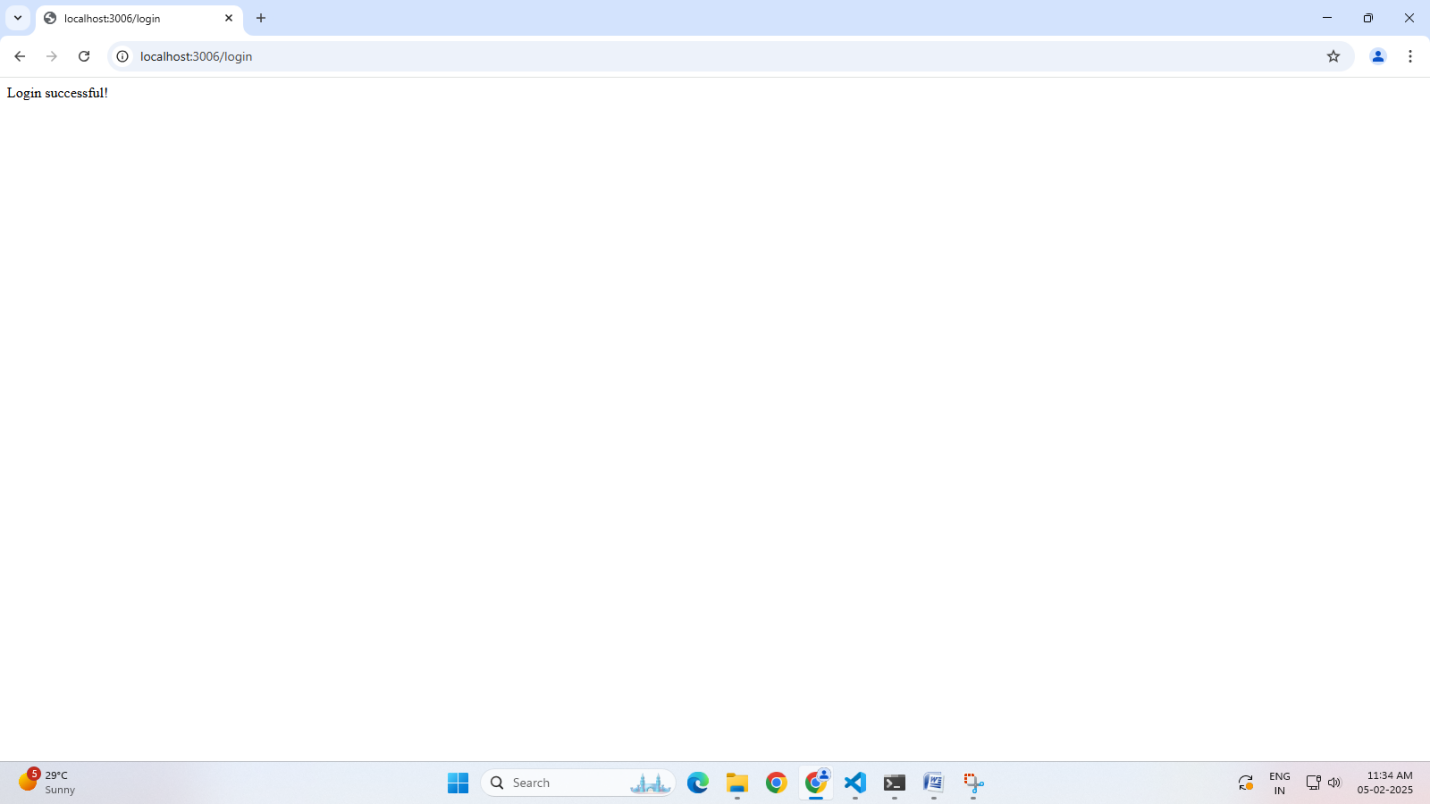
You see the login form displayed in your browser. It will ask for a username and password.



Enter the username **admin** and the password **password123** in the form. Click the login Button



If the credentials match, you will see the message **Login successful!**



If the credentials do not match, you will see the message **Invalid credentials.**

**WEEK-3 Write a Node JS program to perform read, write and other operations on a file.**

**Filename: File-Operations.js**

const fs = require('fs');

// File path

const filePath = 'C:/Users/VAAGDEVI/Desktop/sample.txt';

// 1. Write to a file

console.log('Starting file write...');

fs.writeFile(filePath, 'Hello, Node.js file operations!', (err) => {

if (err) {

console.log('Error writing to file:', err);

return;

}

console.log('File written successfully.');

// 2. Read from the file

console.log('Reading file...');

fs.readFile(filePath, 'utf8', (err, data) => {

if (err) {

console.log('Error reading from file:', err);

return;

}

console.log('File content:', data);

//3. Append to the file

console.log('Appending to file...');

fs.appendFile(filePath, '\nAppended text.', (err) => {

if (err) {

console.log('Error appending to file:', err);

return;

}

console.log('Text appended successfully.');

//4. Read the file again to see the changes

console.log('Reading updated file...');

fs.readFile(filePath, 'utf8', (err, data) => {

if (err) {

console.log('Error reading from file:', err);

return;

}

console.log('Updated file content:', data);

//5. Delete the file

console.log('Deleting file...');

fs.unlink(filePath, (err) => {

if (err) {

console.log('Error deleting the file:', err);

return;

}

console.log('File deleted successfully.');

});

});

});

});

});

**Output:**

PS C:\Users\VAAGDEVI\Desktop\FSD> **node File-Operations.js**

Starting file write...

File written successfully.

Reading file...

File content: Hello, Node.js file operations!

Appending to file...

Text appended successfully.

Reading updated file...

Updated file content: Hello, Node.js file operations!

Appended text.

Deleting file...

File deleted successfully.

**WEEK-4:** **Read Form Data from Query String and Generate Response in Node.js**

PS C:\Users\VAAGDEVI\Desktop\FSD> **npm init -y**

Wrote to C:\Users\VAAGDEVI\Desktop\FSD\package.json:

{

"name": "fsd",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\"&& exit 1"

},

"keywords": [],

"author": "",

"license": "ISC",

"type": "commonjs",

"description": ""

}

PS C:\Users\VAAGDEVI\Desktop\FSD> **npm install express body-parser**

added 69 packages, and audited 70 packages in 2s

14 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

File Name: **week-4.js**

constexpress=require('express');

constapp=express();

constport=3008;

// Route to handle form data sent through query string

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

  res.send(`

    <form method="GET" action="/greet">

      <label for="name">Name:</label>

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

      <button type="submit">Greet Me</button>

    </form>

  `);

});

// Route to process query string and generate response

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

  constname=req.query.name;

  if (name) {

    res.send(`Hello, ${name}! Welcome to the Node.js application.`);

  } else {

    res.send('Please provide a name.');

  }

});

app.listen(port, () => {

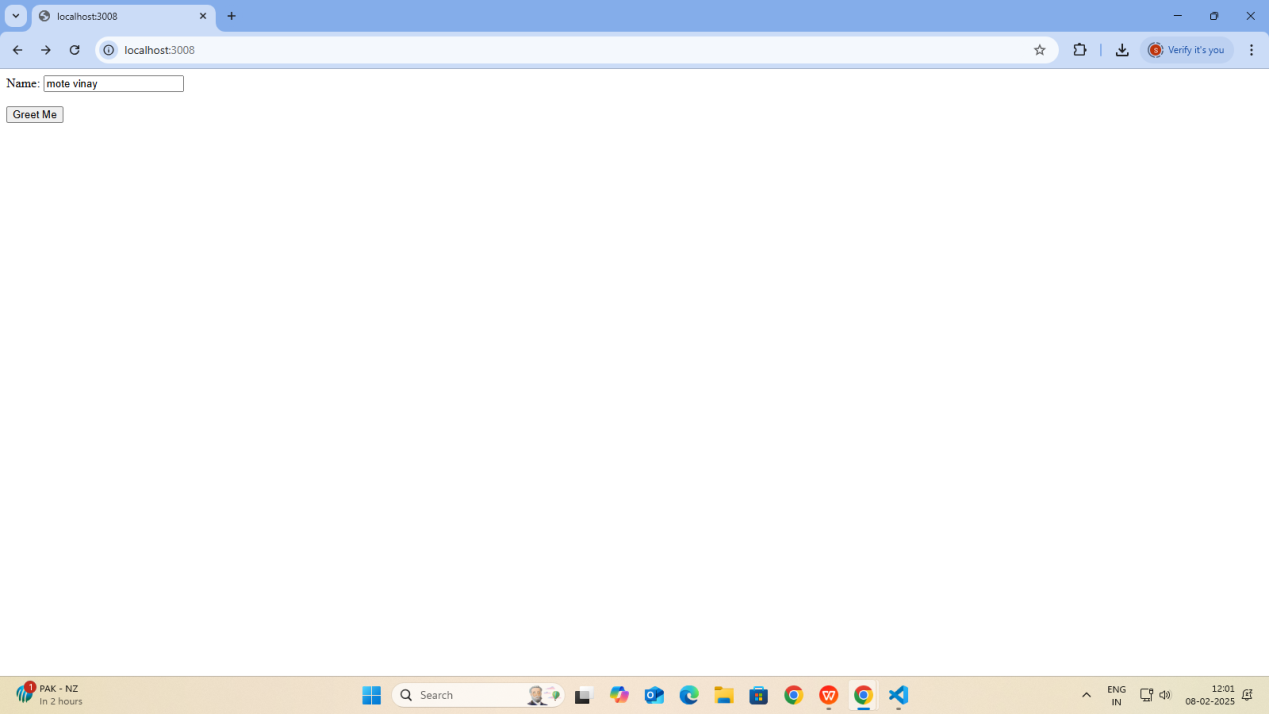
  console.log(`Server is running at http://localhost:${port}`);

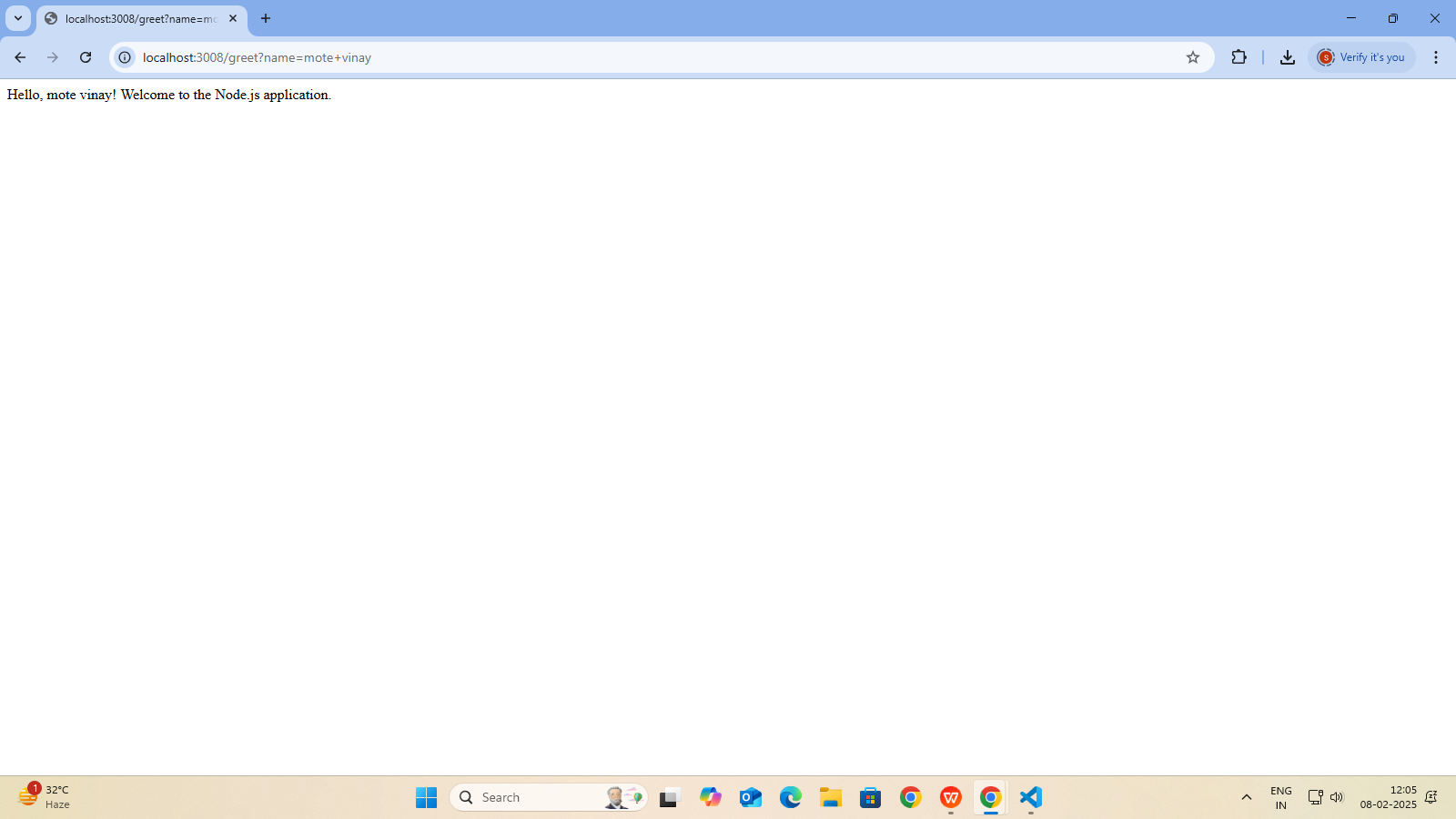
});

**OUTPUT:**

PS C:\Users\VAAGDEVI\Desktop\FSD>**node week-4.js**

Server is running at http://localhost:3008





**WEEK-5. Create a food delivery website where users can order food from a particular restaurant listed in the website for handling http requests and responses using NodeJS.**

To create a simple food delivery website using Node.js where users can order food from a particular restaurant, you will need to follow these steps:

**Step 1: Set up Node.js Project**

Create a new project directory:

mkdir **food-delivery-app**

cd **food-delivery-app**

**Step 2:** Initialize a new Node.js project:

**npm init -y**

**Step 3:** Install required dependencies: You'll need the Express framework to handle HTTP requests and responses.

**npm install express body-parser**

**Step 4:** Create the main server file: Create a file called **server.js** in the project directory

const express = require('express');

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

const cors = require('cors');

const path = require('path');

const app = express();

const PORT = process.env.PORT || 3000;

// Middleware

app.use(cors());

app.use(bodyParser.json());

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

app.use(express.json());

// Sample restaurant data

const restaurants = [

    {

        id: 1,

        name: "Tasty Bites",

        cuisine: "Italian",

        rating: 4.5,

        deliveryTime: "30-40 min",

        image: "/images/restaurant1.jpg",

        menu: [

            {

                id: 1,

                name: "Margherita Pizza",

                price: 399,

                description: "Fresh tomatoes, mozzarella, basil, and olive oil",

                image: "/images/pizza.jpg"

            },

            {

                id: 2,

                name: "Pasta Carbonara",

                price: 349,

                description: "Creamy sauce with pancetta and parmesan",

                image: "/images/pasta.jpg"

            },

            {

                id: 3,

                name: "Tiramisu",

                price: 249,

                description: "Classic Italian dessert with coffee and mascarpone",

                image: "/images/tiramisu.jpg"

            }

        ]

    },

    {

        id: 2,

        name: "Spice Garden",

        cuisine: "Indian",

        rating: 4.3,

        deliveryTime: "35-45 min",

        image: "/images/restaurant2.jpg",

        menu: [

            {

                id: 4,

                name: "Butter Chicken",

                price: 449,

                description: "Creamy tomato curry with tender chicken",

                image: "/images/butter-chicken.jpg"

            },

            {

                id: 5,

                name: "Vegetable Biryani",

                price: 299,

                description: "Aromatic rice with mixed vegetables and spices",

                image: "/images/biryani.jpg"

            },

            {

                id: 6,

                name: "Garlic Naan",

                price: 69,

                description: "Freshly baked bread with garlic and butter",

                image: "/images/naan.jpg"

            }

        ]

    },

    {

        id: 3,

        name: "Sushi Master",

        cuisine: "Japanese",

        rating: 4.7,

        deliveryTime: "25-35 min",

        image: "/images/restaurant3.jpg",

        menu: [

            {

                id: 7,

                name: "California Roll",

                price: 449,

                description: "Crab, avocado, and cucumber roll",

                image: "/images/california-roll.jpg"

            },

            {

                id: 8,

                name: "Salmon Nigiri",

                price: 499,

                description: "Fresh salmon over pressed sushi rice",

                image: "/images/salmon-nigiri.jpg"

            },

            {

                id: 9,

                name: "Tempura Udon",

                price: 399,

                description: "Thick noodles in hot broth with tempura",

                image: "/images/tempura-udon.jpg"

            }

        ]

    }

];

// Routes

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

    res.json(restaurants);

});

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

    const restaurant = restaurants.find(r => r.id === parseInt(req.params.id));

    if (!restaurant) return res.status(404).json({ message: 'Restaurant not found' });

    res.json(restaurant);

});

// Order endpoint (mocked, no DB)

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

    const { items, totalAmount, customerDetails } = req.body;

    if (!items || items.length === 0) {

        return res.status(400).json({ error: 'No items in order' });

    }

    // Mock order confirmation

    const order = {

        id: Date.now(), // Mock ID

        items,

        totalAmount,

        customerDetails,

        status: 'pending'

    };

    console.log('Received order:', order);

    res.status(201).json({ message: 'Order placed successfully!', order });

});

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

    res.sendFile(path.join(\_\_dirname, 'public', 'index.html'));

});

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`);

});

**Step 5:** Create the main orders file: Create a file called **orders.js** in the project directory

// Mock order storage (in-memory array)

const orders = [];

function createOrder(data) {

  const { restaurantId, items, totalAmount, customerDetails } = data;

  const newOrder = {

    id: Date.now(), // mock unique ID

    restaurantId,

    items: items.map(item => ({

      foodItemId: item.id,

      quantity: item.quantity || 1,

      price: item.price

    })),

    totalAmount,

    customerDetails,

    status: 'pending',

    orderDate: new Date()

  };

  orders.push(newOrder);

  return newOrder;

}

function getAllOrders() {

  return orders;

}

module.exports = {

  createOrder,

  getAllOrders

};

**Step 6:** Create a folder name **public** in thatcreate file name **app.js** and create folder name **images** in **public**

let cart = [];

let restaurants = [];

function formatPrice(price) {

    console.log('Formatting price:', price);

    const formatted = `₹${price.toFixed(0)}`;

    console.log('Formatted price:', formatted);

    return formatted;

}

// Fetch restaurants when the page loads

window.addEventListener('DOMContentLoaded', async () => {

    try {

        const response = await fetch('/api/restaurants');

        restaurants = await response.json();

        console.log('Fetched restaurants:', restaurants);

        displayRestaurants();

    } catch (error) {

        console.error('Error fetching restaurants:', error);

    }

});

function displayRestaurants() {

    const restaurantsList = document.getElementById('restaurants-list');

    restaurantsList.innerHTML = restaurants.map(restaurant => `

        <div class="restaurant-card">

            <div class="restaurant-image">

                <img src="${restaurant.image}" alt="${restaurant.name}">

            </div>

            <div class="restaurant-info">

                <h2>${restaurant.name}</h2>

                <p class="cuisine">${restaurant.cuisine}</p>

                <div class="restaurant-meta">

                    <span class="rating">⭐ ${restaurant.rating}</span>

                    <span class="delivery-time">🕒 ${restaurant.deliveryTime}</span>

                </div>

            </div>

            <div class="menu">

                <h3>Menu</h3>

                ${restaurant.menu.map(item => `

                    <div class="menu-item">

                        <div class="menu-item-image">

                            <img src="${item.image}" alt="${item.name}">

                        </div>

                        <div class="menu-item-info">

                            <h4>${item.name}</h4>

                            <p class="description">${item.description}</p>

                            <div class="price-action">

                                <span class="price">${formatPrice(item.price)}</span>

                                <button onclick="addToCart(${restaurant.id}, ${item.id})">Add to Cart</button>

                            </div>

                        </div>

                    </div>

                `).join('')}

            </div>

        </div>

    `).join('');

}

function addToCart(restaurantId, itemId) {

    const restaurant = restaurants.find(r => r.id === restaurantId);

    const item = restaurant.menu.find(i => i.id === itemId);

    cart.push({

        ...item,

        restaurantId

    });

    updateCart();

}

function updateCart() {

    const cartItems = document.getElementById('cart-items');

    const cartCount = document.getElementById('cart-count');

    const cartTotal = document.getElementById('cart-total');

    cartCount.textContent = cart.length;

    cartItems.innerHTML = cart.map((item, index) => `

        <div class="cart-item">

            <div class="cart-item-image">

                <img src="${item.image}" alt="${item.name}">

            </div>

            <div class="cart-item-info">

                <h4>${item.name}</h4>

                <p class="description">${item.description}</p>

                <div class="price-action">

                    <span class="price">${formatPrice(item.price)}</span>

                    <button onclick="removeFromCart(${index})">Remove</button>

                </div>

            </div>

        </div>

    `).join('');

    const total = cart.reduce((sum, item) => sum + item.price, 0);

    cartTotal.textContent = formatPrice(total);

}

function removeFromCart(index) {

    cart.splice(index, 1);

    updateCart();

}

function toggleCart() {

    const cartSidebar = document.getElementById('cart-sidebar');

    cartSidebar.classList.toggle('active');

}

function checkout() {

    if (cart.length === 0) {

        alert('Your cart is empty!');

        return;

    }

    const modal = document.getElementById('checkout-modal');

    modal.style.display = 'block';

}

document.getElementById('checkout-form').addEventListener('submit', async (e) => {

    e.preventDefault();

    const formData = new FormData(e.target);

    const customerDetails = {

        name: formData.get('name'),

        email: formData.get('email'),

        address: formData.get('address'),

        phone: formData.get('phone')

    };

    console.log('Customer Details:', customerDetails);

    try {

        const response = await fetch('/api/orders', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({

                items: cart,

                totalAmount: cart.reduce((sum, item) => sum + item.price, 0),

                customerDetails

            })

        });

        if (!response.ok) {

            const errorText = await response.text(); // Grab the error text (might be HTML)

            throw new Error(`Server error: ${errorText}`);

        }

        const result = await response.json();

        if (response.ok) {

            alert('Order placed successfully!');

            cart = [];

            updateCart();

            document.getElementById('checkout-modal').style.display = 'none';

            document.getElementById('checkout-form').reset();

        } else {

            throw new Error(result.message);

        }

    } catch (error) {

        alert('Error placing order: ' + error.message);

    }

});

**Step 7**: create a file names index.html and styles.css in the public folder

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Food Delivery App</title>

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

    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600&display=swap" rel="stylesheet">

</head>

<body>

    <header>

        <nav>

            <div class="logo">FoodExpress</div>

            <div class="cart-icon" onclick="toggleCart()">

                🛒 <span id="cart-count">0</span>

            </div>

        </nav>

    </header>

    <main>

        <section class="restaurants-container">

            <h1>Our Restaurants</h1>

            <div id="restaurants-list"></div>

        </section>

        <div id="cart-sidebar" class="cart-sidebar">

            <h2>Your Cart</h2>

            <div id="cart-items"></div>

            <div class="cart-total">

                <p>Total: ₹<span id="cart-total">0</span></p>

                <button onclick="checkout()" class="checkout-btn">Checkout</button>

            </div>

        </div>

    </main>

    <div id="checkout-modal" class="modal">

        <div class="modal-content">

            <h2>Checkout</h2>

            <form id="checkout-form">

                <input type="text" name="name" placeholder="Name" required>

                <input type="email" name="email" placeholder="Email" required>

                <input type="text" name="address" placeholder="Address" required>

                <input type="tel" name="phone" placeholder="Phone" required>

                <button type="submit" class="checkout-btn">Place Order</button>

            </form>

        </div>

    </div>

    <script src="app.js?v=2"></script>

</body>

</html>

**styles.css**

\* {

    margin: 0;

    padding: 0;

    box-sizing: border-box;

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

}

body {

    background-color: #f5f5f5;

}

header {

    background-color: #ffffff;

    box-shadow: 0 2px 5px rgba(0,0,0,0.1);

    padding: 1rem 2rem;

}

nav {

    display: flex;

    justify-content: space-between;

    align-items: center;

    max-width: 1200px;

    margin: 0 auto;

}

.logo {

    font-size: 1.5rem;

    font-weight: 600;

    color: #ff4757;

}

.cart-icon {

    cursor: pointer;

    font-size: 1.2rem;

}

main {

    max-width: 1200px;

    margin: 2rem auto;

    padding: 0 1rem;

}

.restaurants-container h1 {

    margin-bottom: 2rem;

    color: #2d3436;

}

#restaurants-list {

    display: grid;

    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

    gap: 2rem;

}

.restaurant-card {

    background: white;

    border-radius: 10px;

    padding: 1rem;

    box-shadow: 0 2px 5px rgba(0,0,0,0.1);

    overflow: hidden;

}

.restaurant-image {

    margin: -1rem -1rem 1rem -1rem;

    height: 200px;

    overflow: hidden;

}

.restaurant-image img {

    width: 100%;

    height: 100%;

    object-fit: cover;

}

.restaurant-info {

    margin-bottom: 1.5rem;

}

.restaurant-meta {

    display: flex;

    gap: 1rem;

    margin-top: 0.5rem;

    color: #666;

}

.cuisine {

    color: #666;

    font-style: italic;

}

.rating {

    color: #ffa41c;

}

.menu {

    margin-top: 1.5rem;

}

.menu h3 {

    margin-bottom: 1rem;

    color: #2d3436;

}

.menu-item {

    display: grid;

    grid-template-columns: 100px 1fr;

    gap: 1rem;

    padding: 1rem 0;

    border-bottom: 1px solid #eee;

}

.menu-item-image {

    width: 100px;

    height: 100px;

    border-radius: 8px;

    overflow: hidden;

}

.menu-item-image img {

    width: 100%;

    height: 100%;

    object-fit: cover;

}

.menu-item-info h4 {

    margin-bottom: 0.5rem;

    color: #2d3436;

}

.description {

    color: #666;

    font-size: 0.9rem;

    margin-bottom: 0.5rem;

}

.price-action {

    display: flex;

    justify-content: space-between;

    align-items: center;

    margin-top: 0.5rem;

}

.price {

    font-weight: 600;

    color: #2d3436;

}

button {

    background: #ff4757;

    color: white;

    border: none;

    padding: 0.5rem 1rem;

    border-radius: 5px;

    cursor: pointer;

    transition: background 0.3s ease;

}

button:hover {

    background: #ff6b81;

}

.cart-sidebar {

    position: fixed;

    right: -400px;

    top: 0;

    width: 400px;

    height: 100vh;

    background: white;

    box-shadow: -2px 0 5px rgba(0,0,0,0.1);

    padding: 2rem;

    transition: right 0.3s ease;

}

.cart-sidebar.active {

    right: 0;

}

.cart-total {

    position: absolute;

    bottom: 2rem;

    left: 2rem;

    right: 2rem;

}

.cart-item {

    display: grid;

    grid-template-columns: 80px 1fr;

    gap: 1rem;

    padding: 1rem 0;

    border-bottom: 1px solid #eee;

}

.cart-item-image {

    width: 80px;

    height: 80px;

    border-radius: 8px;

    overflow: hidden;

}

.cart-item-image img {

    width: 100%;

    height: 100%;

    object-fit: cover;

}

.cart-item-info h4 {

    margin-bottom: 0.25rem;

    color: #2d3436;

}

.checkout-btn {

    width: 100%;

    padding: 1rem;

    background: #ff4757;

    color: white;

    border: none;

    border-radius: 5px;

    cursor: pointer;

    margin-top: 1rem;

}

.modal {

    display: none;

    position: fixed;

    top: 0;

    left: 0;

    width: 100%;

    height: 100%;

    background: rgba(0,0,0,0.5);

}

.modal-content {

    background: white;

    padding: 2rem;

    border-radius: 10px;

    width: 90%;

    max-width: 500px;

    margin: 2rem auto;

}

#checkout-form {

    display: flex;

    flex-direction: column;

    gap: 1rem;

}

#checkout-form input {

    padding: 0.5rem;

    border: 1px solid #ddd;

    border-radius: 5px;

}

@media (max-width: 768px) {

    .cart-sidebar {

        width: 100%;

        right: -100%;

    }

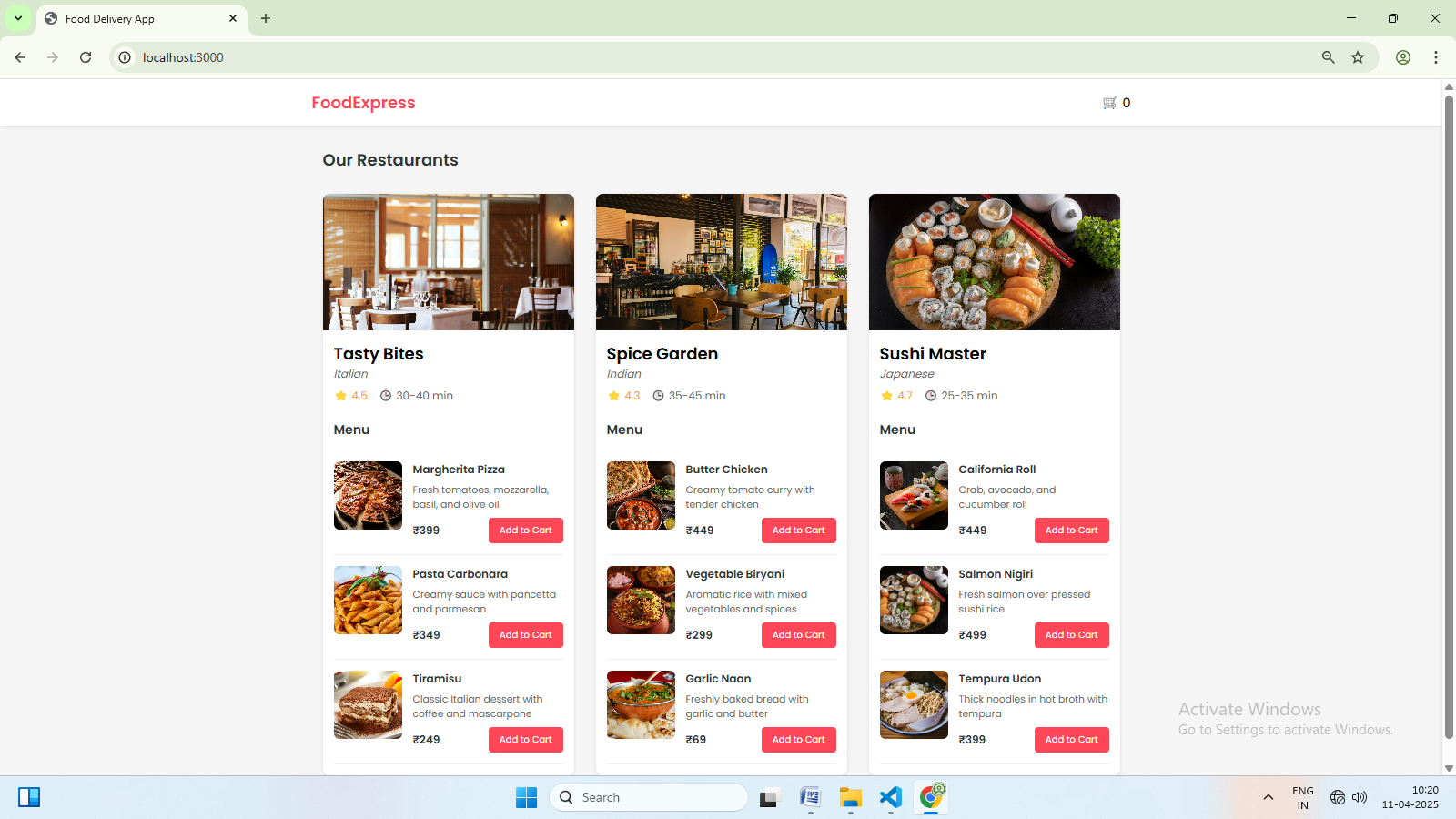
}

**Open Terminal**

C:\Users\VAAGDEVI\Downloads\food-delivery-app> **node server.js**

**Server is running on port 3000**

Open Browser and Type[**http://localhost:3000/**](http://localhost:3000/)

****

**WEEK-6: Implement a program with basic commands on databases and collections using MongoDB.**

To implement a program with basic commands on databases and collections using MongoDB, you will need to install MongoDB, set up a MongoDB instance, and use the MongoDB Node.js driver to interact with the database.

Step 1: Install MongoDB and the MongoDB Node.js Driver

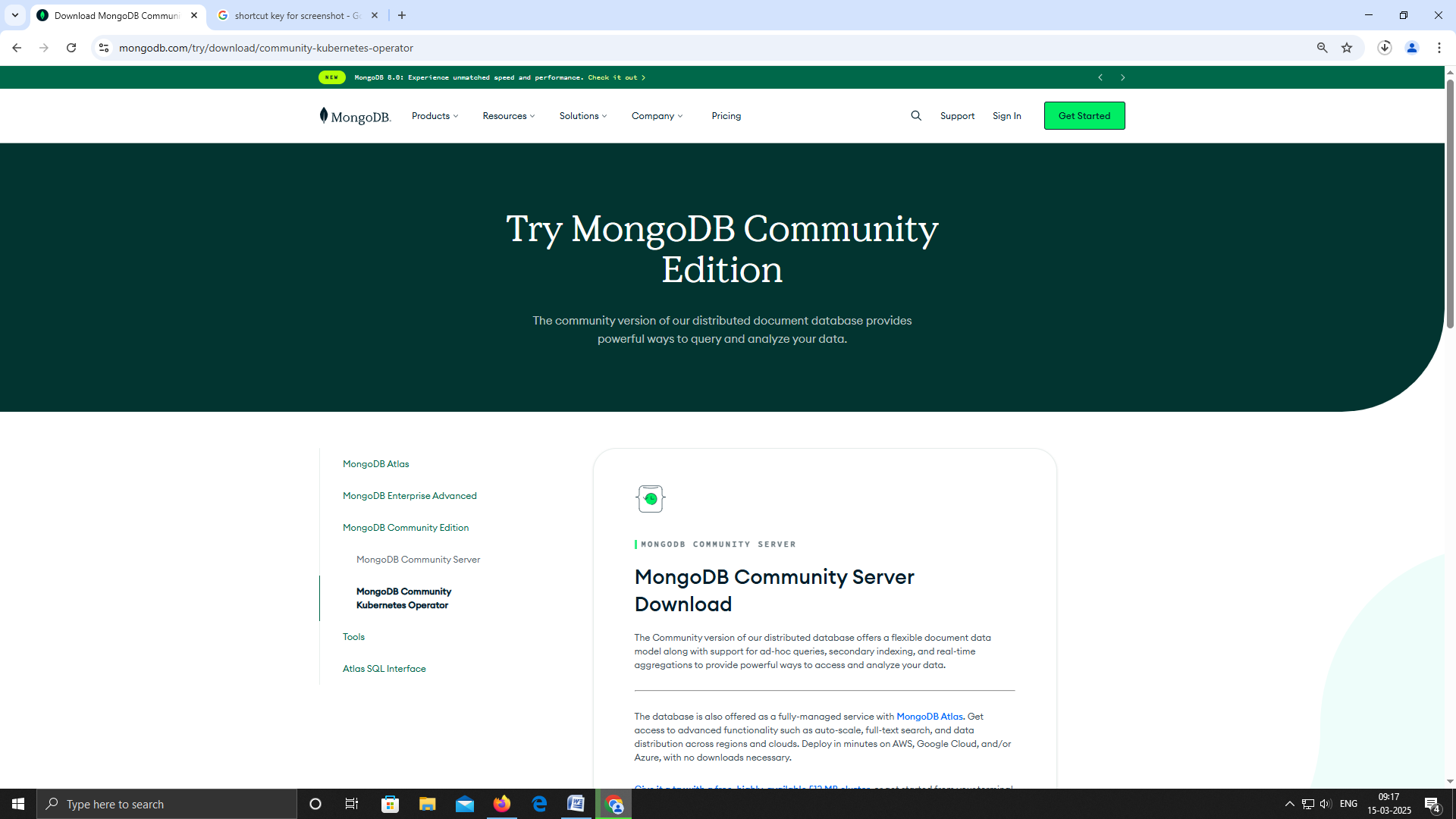
**Mongodb installation**

**1. Download the MongoDB Installer:**

* Go to the MongoDB Download Center.
* You can also install directly by following this link

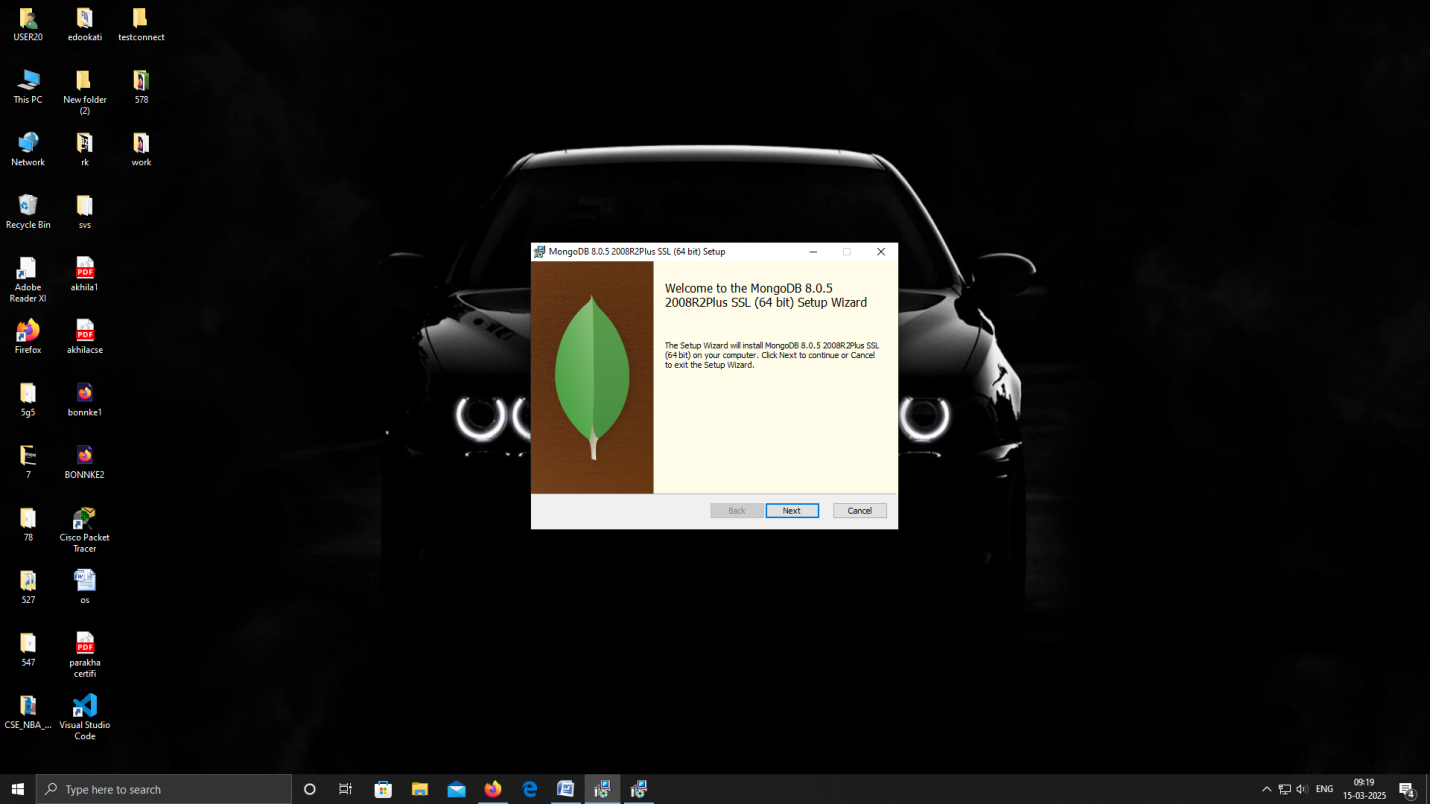
https://www.mongodb.com/try/download/community

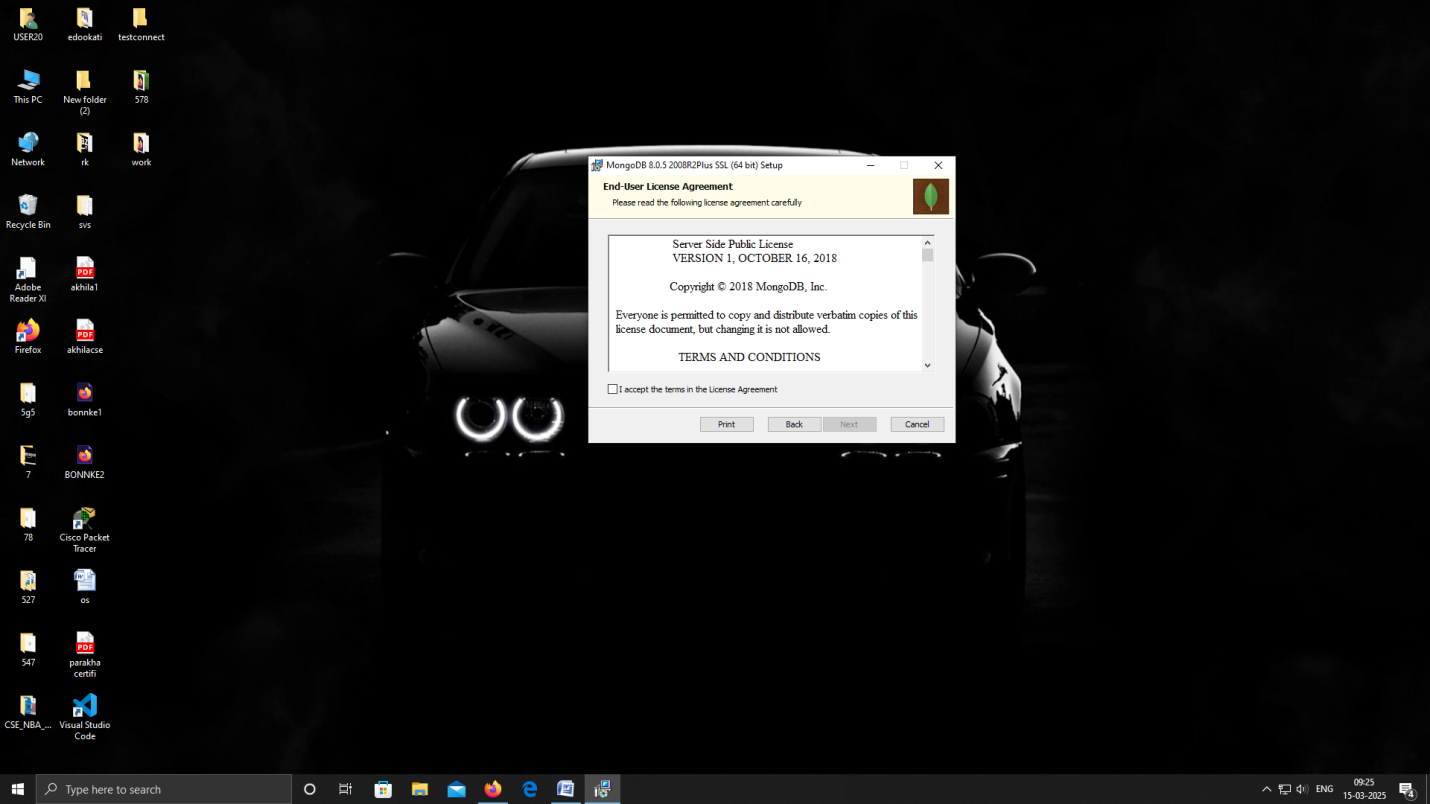
* Select the "Windows" platform and choose the "msi" package.
* Download the latest version of MongoDB Community Edition.

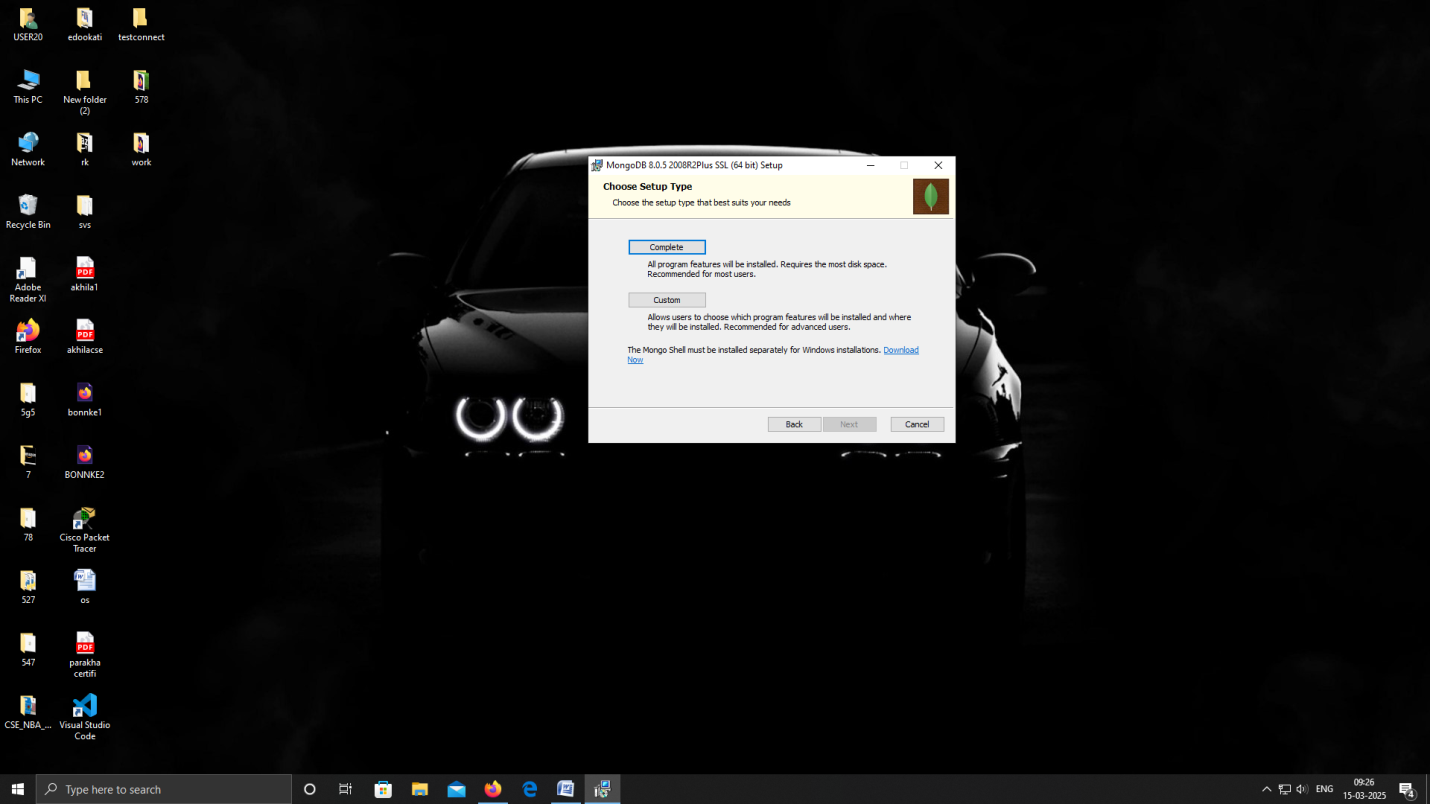
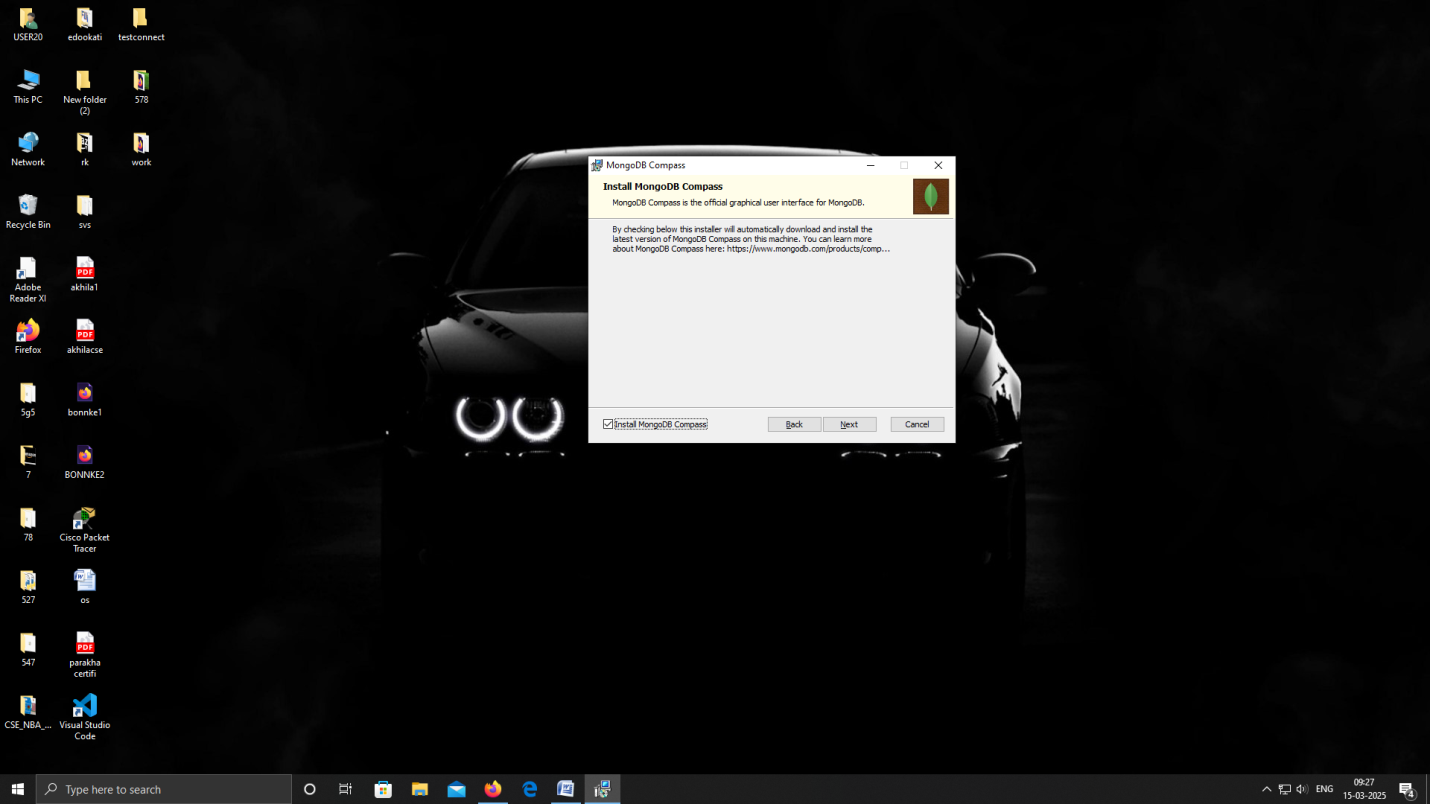


**2. Run the Installer**:

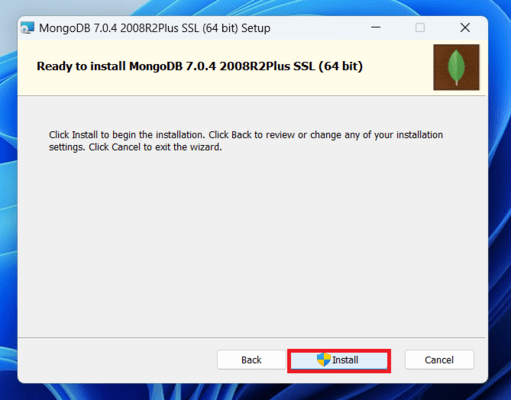
* Locate the downloaded .msi file.
* Double-click the file to start the installation wizard.
* Follow the on-screen instructions, including choosing a setup type (e.g., "Complete").





********

* Click theInstall button to start the MongoDB installation process:



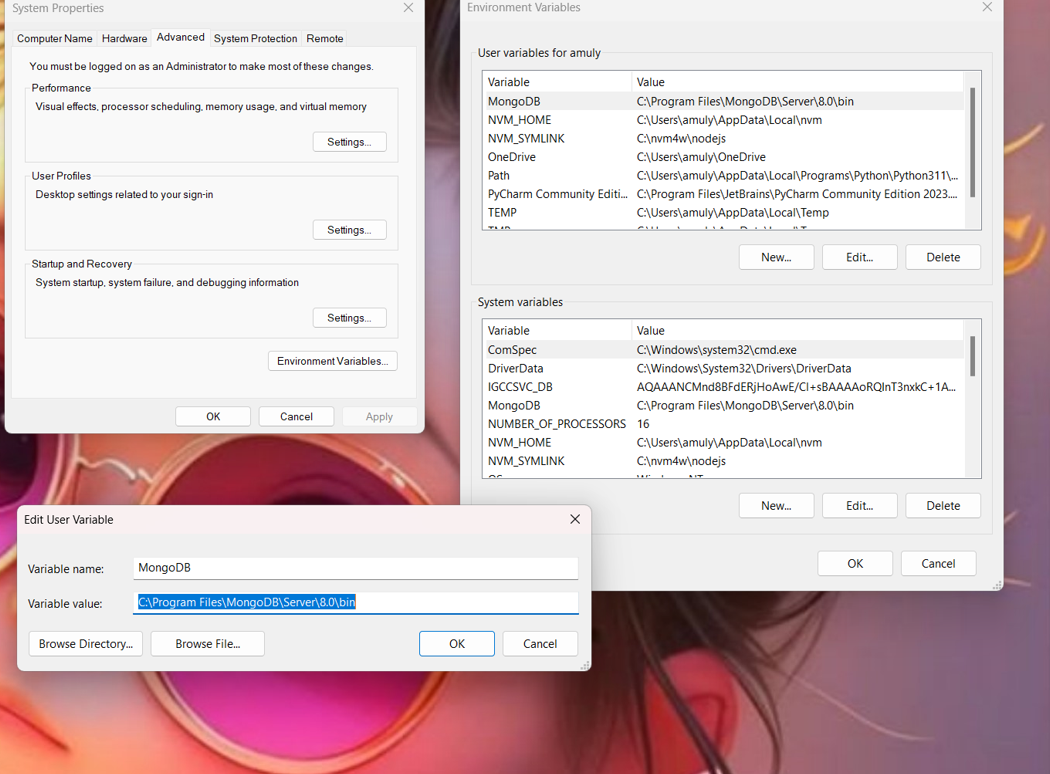
* After clicking on the install button installation of MongoDb begins:
* 

Step 4: Complete Installation

* Now clickthe Finish buttonto complete the MongoDB installation process:

Step 5: Set Environment Variables

* Now we go to the location where MongoDB installed  and copy the path
* Now, to create an environment variable open system **properties >> Environment Variable >> System variable >> path >> Edit Environment variable**
* paste the copied link to your environment system and **click Ok**:



**Run MongoDB Server (mongod)**

**Step 1. Start MongoDB Service**

* After setting the environment variable, we will run the MongoDB server, i.e. **mongod**.
* So, open the **command prompt** and run the following command:

**mongod**

When you run this command you will get an error i.e. **C:/data/db/ not found**.

**Step 2. Create Required Folders**

* Now, Open **C**drive and create a folder named “**data**”
* Inside the **data**folder create another folder named “**db**“.

**Step 3. Restart MongoDB**

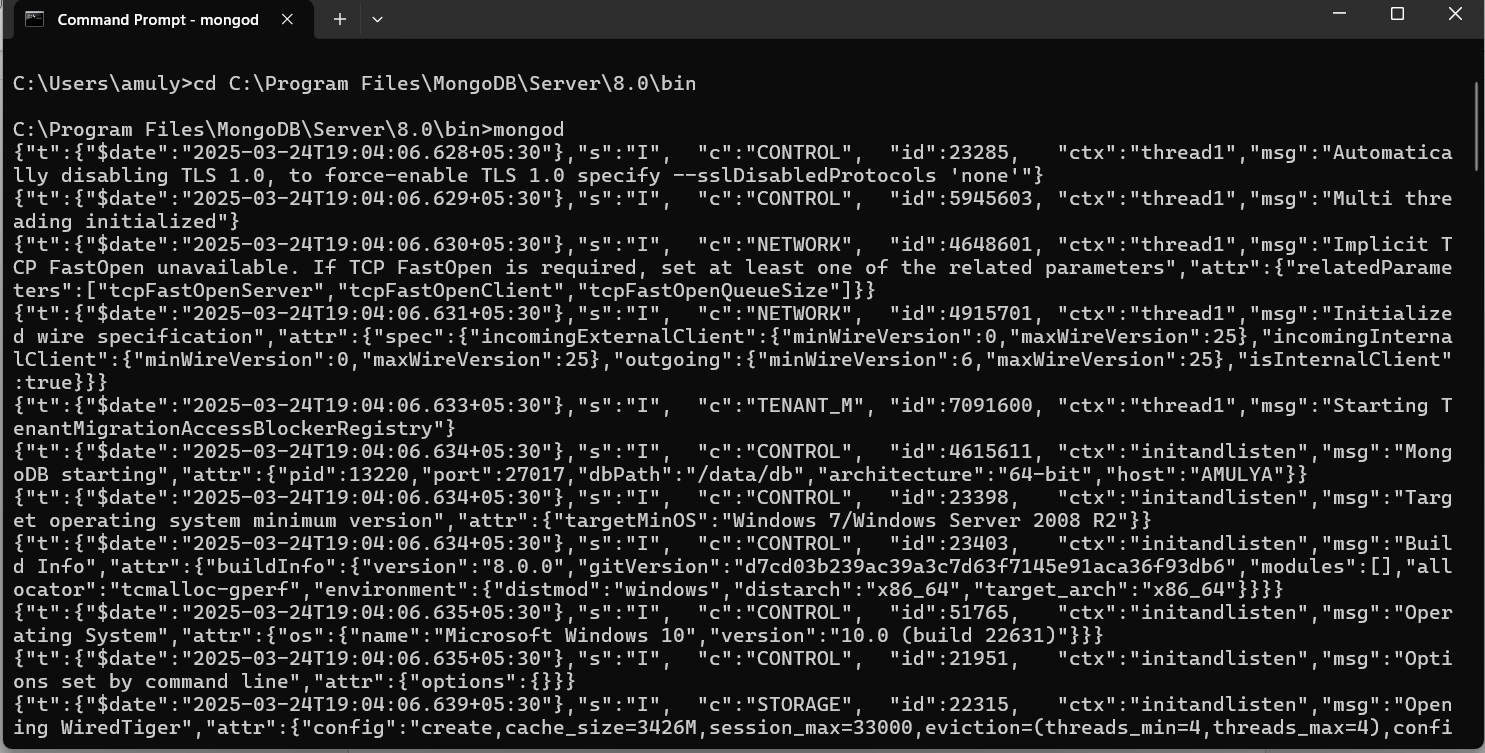
* After creating these folders. Again open the command prompt and run the following command:

**mongod**

* Now, this time the MongoDB server (i.e., mongod) will run successfully.
* If you’re getting any error enter to the location where the mongodb is located i.e

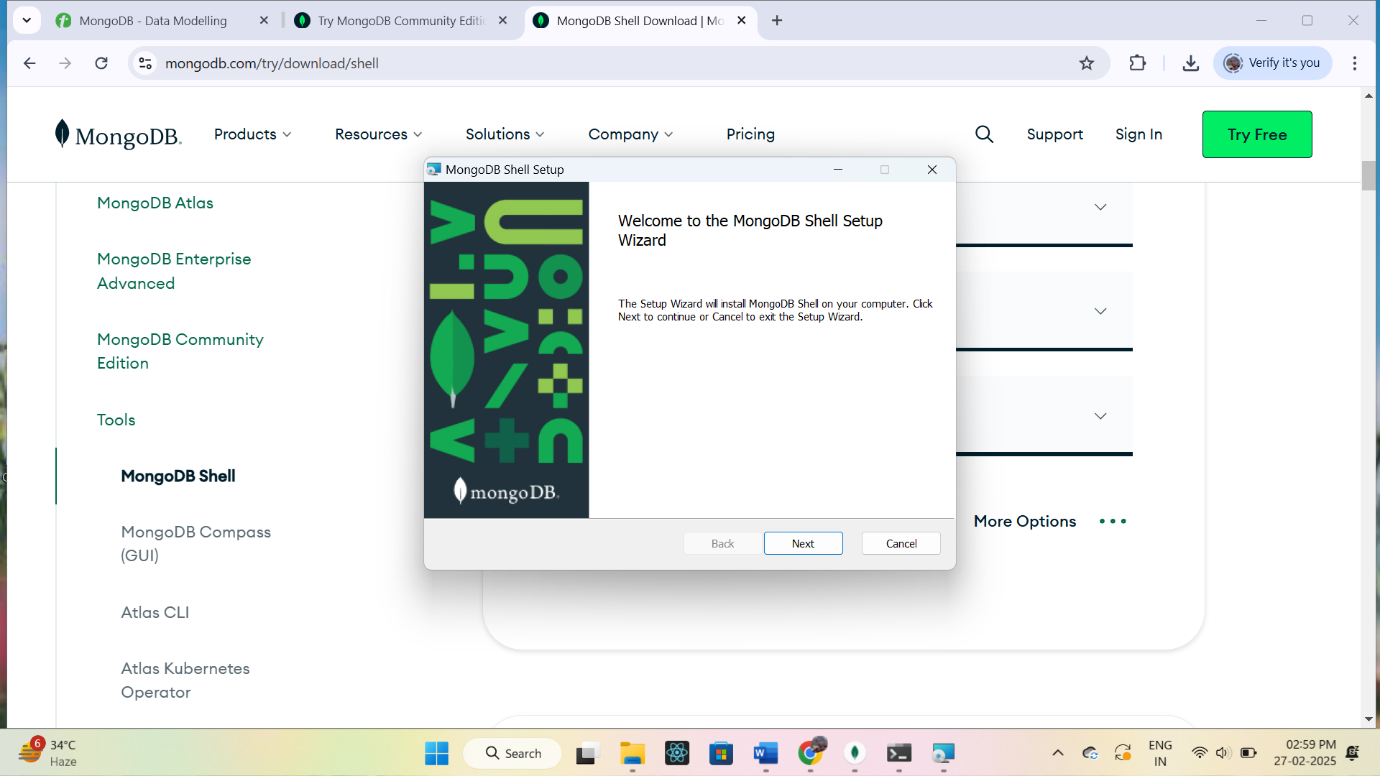
The path that you have given in Environment variables

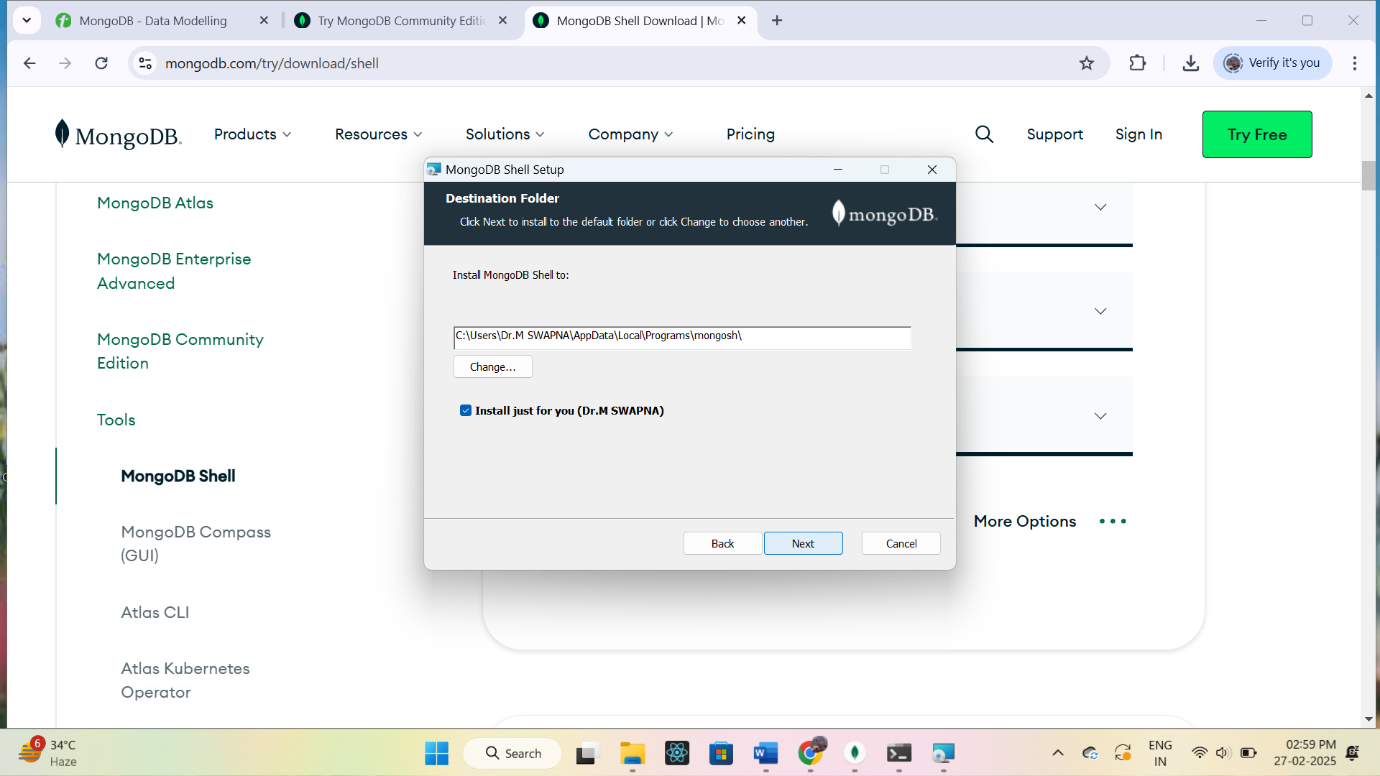
* Run the above command again **i.e mongod**



**Installing mongo shell:-**

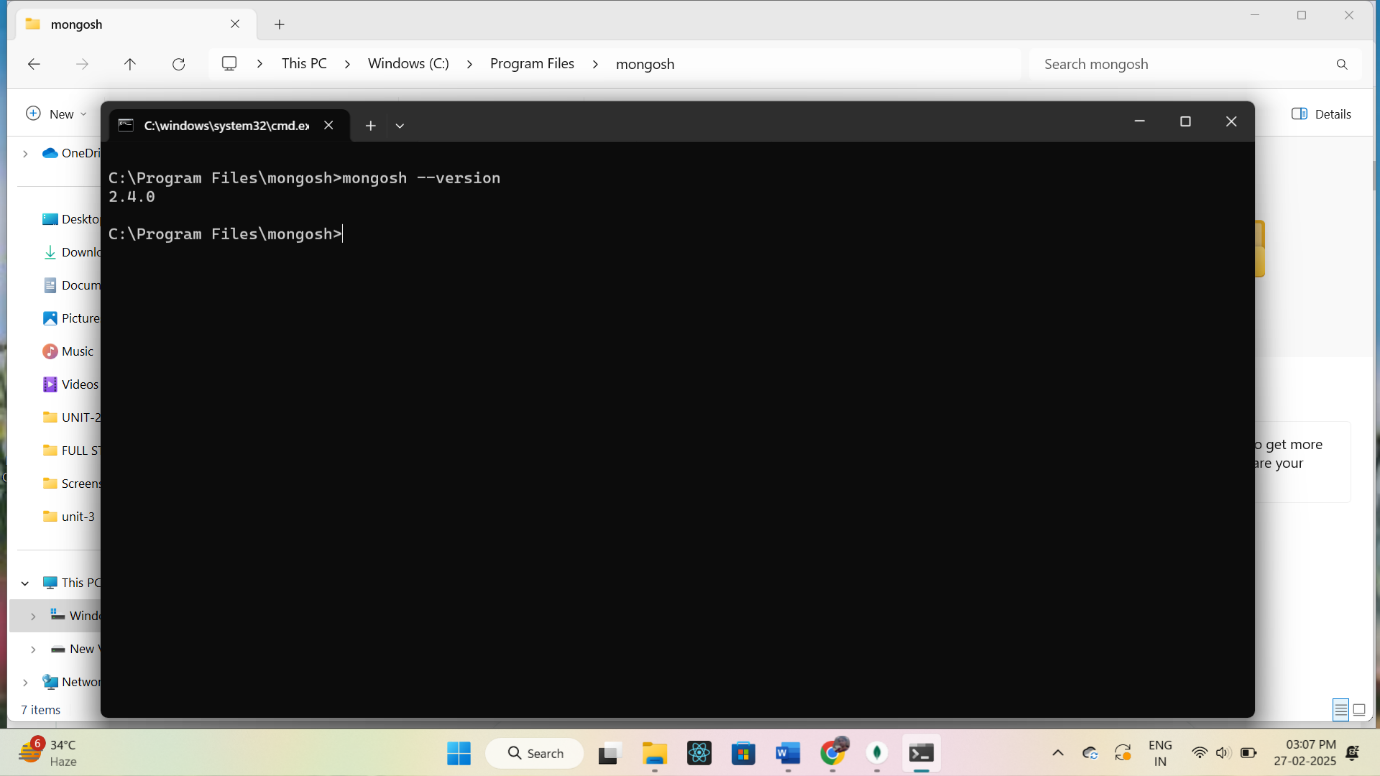
<https://www.mogodb.com/try/download/shell>





To verify the installation open the command prompt and issue the command mongosh

The test>prompt will appear



**Run the MongoDB Shell (mongosh)**

**Step 1. Connect to MongoDB Server with mongosh**

* Now we are going to connect our server (mongod) with the mongo shell. So, keep that mongod window
* open a new command prompt window and type:

mongosh

* You are now connected to the MongoDB shell.

Please do not close the mongod window if you close this window your server will stop working and it will not able to connect with the mongo shell.

**Step 2. Create a Database**

Now we can make a new database, collections, and documents in our shell. Use the following command within the mongosh shell to create a new database:

**use database\_name**

The use Database\_name command makes a new [database](https://www.geeksforgeeks.org/what-is-database/) in the system if it does not exist, if the database exists it uses that database:

**use fsd**

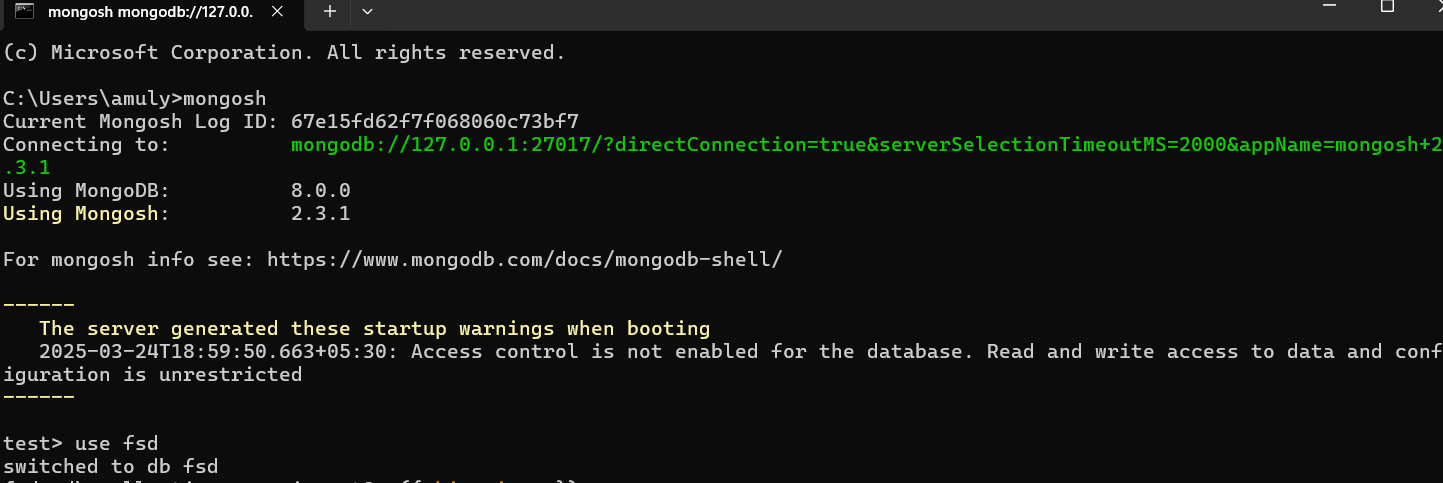
**Step 3: Add Data to a Collection**

Insert a document into a collection using:

**db.collection\_name.insertOne({field: value})**

The**db.Collection\_name** command makes a new collection in the fsd database and the [insertOne()](https://www.geeksforgeeks.org/mongodb-insertone-method-db-collection-insertone/) method inserts the document in the **student** collection:

**db.student.insertOne({shivani:100})**



Create and handle database in MongoDB using various commands.

**1) Displaying all the databases present in the system**

testDB>show dbs

**2) Drop Database**

use testDB

testDB>db.dropDatabase()

Collection:-

We can perform various operations such as insert,delete and update on the collection

1) Create collection:-

We can create collection explicitly using **create collection** command

test>use mystudent

mystudent>db.createCollection("student\_details")

2) Display collection:-

mystudent> show collections

show student\_details

mystudent>

3) Drop collection:-

mystudent>db.student\_details.drop()

mystudent>

4) insert documents with in the collection:-

> db.createCollection("empDetails")

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

5) delete documents:-

db.allstudents.remove({First\_Name: "Fathima"})

6) update Documents:-

student\_details=[{"name":"xyz","age":35},

{"name":"abc","age":60}]

db.student\_details.updateOne( { name: "xyz" }, { $set: { age:0 } } )

**Week-7 Implement CRUD operations on the given dataset using MongoDB.**

**mongodb-crud-project/**

**├── data/**

**│ └── dataset.json**

**├── src/**

**│ └── db.js**

**│ └── crudOperations.js**

**└── package.json**

* **dataset.json - Sample dataset.**
* **db.js - Handles MongoDB connection.**
* **crudOperations.js - Implements CRUD operations.**

**Step 1: Create the Project Folder**

mkdir mongodb-crud-project

cd mongodb-crud-project

**Step 2: Initialize Node.js Project**

npm init -y

**Step 3: Install MongoDB Driver**

npm install mongodb

**Step 4: Setting Up MongoDB Connection (db.js)**

const { MongoClient } = require('mongodb');

//**imports the MongoClient class from the mongodb package.**

// Replace with your MongoDB URI

const uri = 'mongodb://127.0.0.1:27017';

//**The uri specifies the MongoDB connection string.**

const client = new MongoClient(uri);

//new MongoClient(uri) creates a new instance of the MongoClient class.

//The client object will be used to connect to and interact with the MongoDB server.

async function connectDB() {

try {

await client.connect();

//This tells the client to connect to the MongoDB server.Since this is an asynchronous operation, await ensures it completes before moving to the next step.

console.log('Connected to MongoDB');

const db = client.db('crudDB');

//Selects (or creates if it doesn't exist) a database named **"crudDB"**.

return db;

} catch (error) {

console.error('Connection error:', error);

}

}

module.exports = connectDB;

**Step 5: Implementing CRUD Operations (crudOperations.js)**

const connectDB = require('./db');

// Create

async function createDocument(collectionName, document) {

const db = await connectDB();

const collection = db.collection(collectionName);

const result = await collection.insertOne(document);

console.log('Document inserted:', result.insertedId);

}

// Read

async function readDocuments(collectionName, query = {}) {

const db = await connectDB();

const collection = db.collection(collectionName);

const documents = await collection.find(query).toArray();

console.log('Documents found:', documents);

return documents;

}

// Update

async function updateDocument(collectionName, filter, updateDoc) {

const db = await connectDB();

const collection = db.collection(collectionName);

const result = await collection.updateOne(filter, { $set: updateDoc });

console.log('Document updated:', result.modifiedCount);

}

// Delete

async function deleteDocument(collectionName, filter) {

const db = await connectDB();

const collection = db.collection(collectionName);

const result = await collection.deleteOne(filter);

console.log('Document deleted:', result.deletedCount);

}

// Export functions

module.exports = {

createDocument,

readDocuments,

updateDocument,

deleteDocument

};

**Step 6: Running the Operations**

**(mongodb-crud-project/ index.js)**

const crud = require('./src/crudOperations');

async function runCRUD() {

await crud.createDocument('users', { name: 'John Doe', age: 30 });

await crud.readDocuments('users');

await crud.updateDocument('users', { name: 'John Doe' }, { age: 31 });

await crud.deleteDocument('users', { name: 'John Doe' });

}

runCRUD();

**Step 7: Run the Project**

node index.js

**Output:-**

**Connected to MongoDB**

**Document inserted: new ObjectId('67eaf37531fe49d7cdb9b78f')**

**Connected to MongoDB**

**Documents found: [**

**{**

**\_id: new ObjectId('67eaf37531fe49d7cdb9b78f'),**

**name: 'John Doe',**

**age: 30**

**}**

**]**

**Connected to MongoDB**

**Document updated: 1**

**Connected to MongoDB**

**Document deleted: 1**

**Week-8 Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB.**

**Project Folder Structure**

**mongo\_project/**

**├── app.js**

**├── package.json**

**└── data/**

**└── sampleCollection.json**

**Step 1: Setup Your Project**

**1) Initialize Node.js Project:**

mkdir mongo\_project

cd mongo\_project

npm init -y

**2) Install MongoDB Driver:**

npm install mongodb

**Step 2: Sample Data**

**Create a sampleCollection.json file inside the data folder:**

[

{ "name": "Alice", "age": 25, "city": "New York" },

{ "name": "Bob", "age": 30, "city": "Los Angeles" },

{ "name": "Charlie", "age": 22, "city": "Chicago" },

{ "name": "David", "age": 35, "city": "Houston" },

{ "name": "Eva", "age": 28, "city": "Phoenix" }

]

**Step 3: Node.js Script (app.js)**

const { MongoClient } = require('mongodb');

// Connection URL

const uri = 'mongodb://127.0.0.1:27017';

const dbName = 'testdb';

const client = new MongoClient(url);

async function main() {

try {

await client.connect();

console.log("Connected to MongoDB");

const db = client.db(dbName);

const collection = db.collection('users');

// Insert sample data

const sampleData = require('./data/sampleCollection.json');

await collection.insertMany(sampleData);

// 1️) Count Documents

const count = await collection.countDocuments();

console.log(`Total Documents: ${count}`);

// 2️) Limit Documents

const limitedDocs = await collection.find().limit(3).toArray();

console.log('Limited Documents:', limitedDocs);

// 3️) Sort Documents

const sortedDocs = await collection.find().sort({ age: -1 }).toArray();

console.log('Sorted by Age (Descending):', sortedDocs);

// 4️) Skip Documents

const skippedDocs = await collection.find().skip(2).limit(2).toArray();

console.log('Skipped First 2, Limited 2:', skippedDocs);

} catch (err) {

console.error(err);

} finally {

await client.close();

}

}

main();

**Step 4: Run the Script**

node app.js

**output:-**

Connected to MongoDB

Total Documents: 15

Limited Documents: [

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cac'),

name: 'Alice',

age: 25,

city: 'New York'

},

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cad'),

name: 'Bob',

age: 30,

city: 'Los Angeles'

},

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cae'),

name: 'Charlie',

age: 22,

city: 'Chicago'

}

]

Sorted by Age (Descending): [

{

\_id: new ObjectId('67eaec3ffae7261fc43f0caf'),

name: 'David',

age: 35,

city: 'Houston'

},

{

\_id: new ObjectId('67eaecf862d7a2342a70a3b2'),

name: 'David',

age: 35,

city: 'Houston'

},

{

\_id: new ObjectId('67eaed1ab29ad7a79df43c42'),

name: 'David',

age: 35,

city: 'Houston'

},

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cad'),

name: 'Bob',

age: 30,

city: 'Los Angeles'

},

{

\_id: new ObjectId('67eaecf862d7a2342a70a3b0'),

name: 'Bob',

age: 30,

city: 'Los Angeles'

},

{

\_id: new ObjectId('67eaed1ab29ad7a79df43c40'),

name: 'Bob',

age: 30,

city: 'Los Angeles'

},

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cb0'),

name: 'Eva',

age: 28,

city: 'Phoenix'

},

{

\_id: new ObjectId('67eaecf862d7a2342a70a3b3'),

name: 'Eva',

age: 28,

city: 'Phoenix'

},

{

\_id: new ObjectId('67eaed1ab29ad7a79df43c43'),

name: 'Eva',

age: 28,

city: 'Phoenix'

},

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cac'),

name: 'Alice',

age: 25,

city: 'New York'

},

{

\_id: new ObjectId('67eaecf862d7a2342a70a3af'),

name: 'Alice',

age: 25,

city: 'New York'

},

{

\_id: new ObjectId('67eaed1ab29ad7a79df43c3f'),

name: 'Alice',

age: 25,

city: 'New York'

},

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cae'),

name: 'Charlie',

age: 22,

city: 'Chicago'

},

{

\_id: new ObjectId('67eaecf862d7a2342a70a3b1'),

name: 'Charlie',

age: 22,

city: 'Chicago'

},

{

\_id: new ObjectId('67eaed1ab29ad7a79df43c41'),

name: 'Charlie',

age: 22,

city: 'Chicago'

}

]

Skipped First 2, Limited 2: [

{

\_id: new ObjectId('67eaec3ffae7261fc43f0cae'),

name: 'Charlie',

age: 22,

city: 'Chicago'

},

{

\_id: new ObjectId('67eaec3ffae7261fc43f0caf'),

name: 'David',

age: 35,

city: 'Houston'

}

]

**Week-9 Develop an angular JS form to apply CSS and Events**

**Project Structure**

angularjs-form-project/

├── index.html

├── app.js

├── style.css

└── README.md

**Step 1: Set Up the Project**

1. Create a folder named angularjs-form-project.
2. Inside this folder, create the following files:
   * index.html
   * app.js
   * style.css

**Step 2: Add AngularJS to Your Project**

**In index.html:**

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<meta charset="UTF-8">

<title>AngularJS Form Example</title>

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

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

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

</head>

<body ng-controller="FormController">

<h1>AngularJS Form Example</h1>

<form name="myForm" ng-submit="submitForm()" novalidate>

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

<input type="text" id="name" ng-model="formData.name" required>

<br><br>

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

<input type="email" id="email" ng-model="formData.email" required>

<br><br>

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

</form>

<p class="message" ng-if="message">{{ message }}</p>

</body>

</html>

**Step 3: Write AngularJS Logic**

**In app.js:**

var app = angular.module('myApp', []);

app.controller('FormController', function($scope) {

$scope.formData = {};

$scope.message = '';

$scope.submitForm = function() {

if ($scope.myForm.$valid) {

$scope.message = 'Form Submitted Successfully!';

console.log('Form Data:', $scope.formData);

} else {

$scope.message = 'Please fill out all fields correctly.';

}

};

});

**Step 4: Add CSS for Styling**

**In style.css:**

body {

font-family: Arial, sans-serif;

background-color: #f4f4f9;

padding: 20px;

text-align: center;

}

form {

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

display: inline-block;

text-align: left;

}

input {

width: 100%;

padding: 10px;

margin: 5px 0 15px 0;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

background-color: #28a745;

color: white;

padding: 10px 20px;

border: none;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

.message {

color: green;

font-weight: bold;

}

**Step 5: Run the Project**

**start index.html**



**Week-10**

**Develop a Job Registration form and validate it using angular JS.**

Project Folder Structure

job-registration/

├── index.html

├── app.js

* **index.html**: The main HTML file for the form.
* **app.js**: AngularJS script file to control form behavior.

**2️) Setting Up the Project**

<!DOCTYPE html>

<html lang="en" ng-app="jobApp">

<head>

<meta charset="UTF-8">

<title>Job Registration Form</title>

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

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

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

</head>

<body ng-controller="JobController">

<h2>Job Registration Form</h2>

<form name="jobForm" ng-submit="submitForm()" novalidate>

<!-- Name Field -->

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

<input type="text" id="name" name="name" ng-model="job.name" required>

<span style="color:red" ng-show="jobForm.name.$touched && jobForm.name.$invalid">Name is required.</span>

<br><br>

<!-- Email Field -->

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

<input type="email" id="email" name="email" ng-model="job.email" required>

<span style="color:red" ng-show="jobForm.email.$touched && jobForm.email.$invalid">Invalid email.</span>

<br><br>

<!-- Job Title -->

<label for="jobTitle">Job Title:</label>

<input type="text" id="jobTitle" name="jobTitle" ng-model="job.jobTitle" required>

<span style="color:red" ng-show="jobForm.jobTitle.$touched && jobForm.jobTitle.$invalid">Job Title is required.</span>

<br><br>

<!-- Experience Field -->

<label for="experience">Experience (in years):</label>

<input type="number" id="experience" name="experience" ng-model="job.experience" min="0" required>

<span style="color:red" ng-show="jobForm.experience.$touched && jobForm.experience.$invalid">Enter a valid number.</span>

<br><br>

<button type="submit" ng-disabled="jobForm.$invalid">Submit</button>

</form>

<div ng-if="submitted">

<h3>Registration Successful!</h3>

<p><strong>Name:</strong> {{ job.name }}</p>

<p><strong>Email:</strong> {{ job.email }}</p>

<p><strong>Job Title:</strong> {{ job.jobTitle }}</p>

<p><strong>Experience:</strong> {{ job.experience }} years</p>

</div>

</body>

</html>

**3) AngularJS Script (app.js)**

**// app.js**

var app = angular.module('jobApp', []);

app.controller('JobController', ['$scope', function($scope) {

$scope.job = {};

$scope.submitted = false;

$scope.submitForm = function() {

if ($scope.jobForm.$valid) {

$scope.submitted = true;

console.log('Form Data:', $scope.job);

} else {

$scope.submitted = false;

alert('Please fill out the form correctly.');

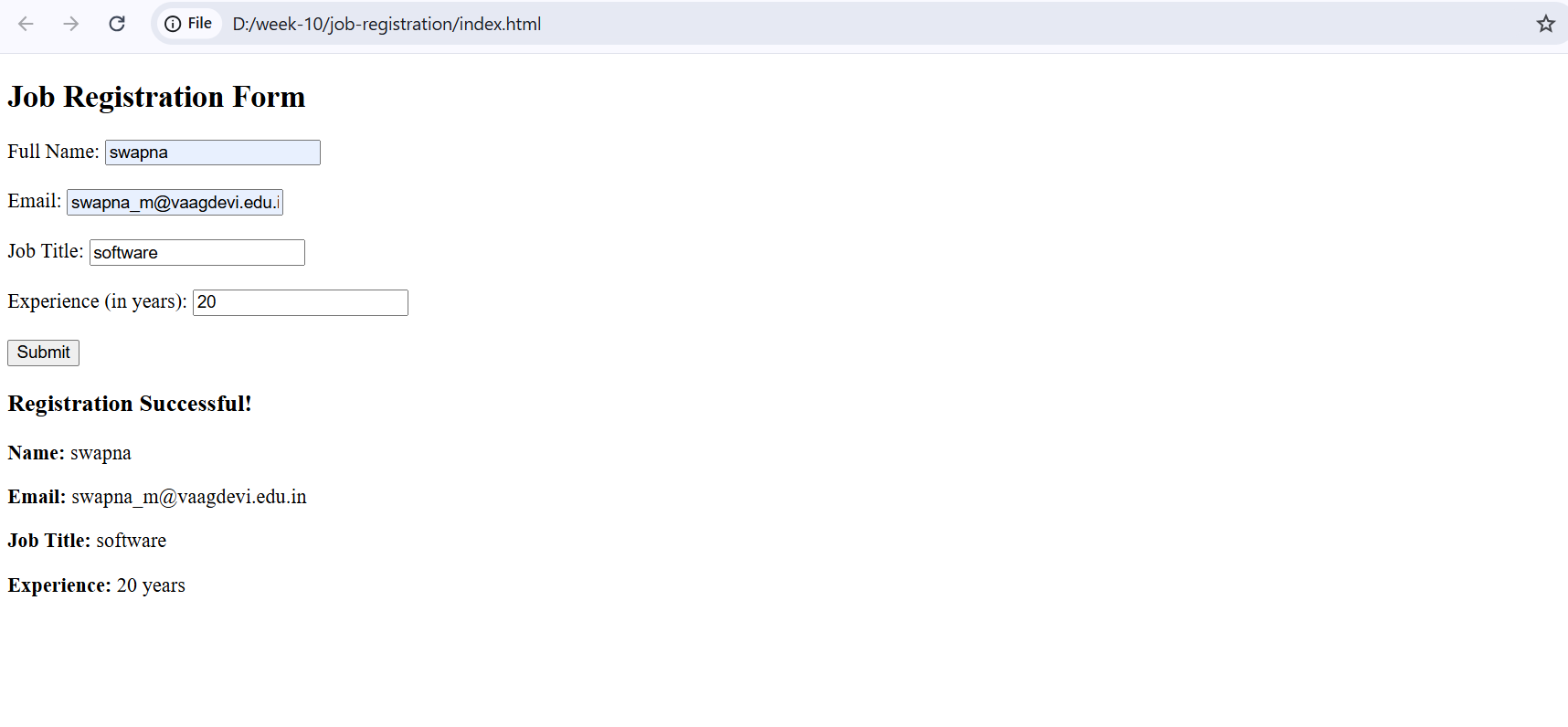
}

};

}]);

**Running the Project**

**start index.html**



**Week-11 Write an angular JS application to access JSON file data of an employee from a server using $http service.**

**Step 1: Set Up Your Project Folder**

Create a folder for your project, for example:

employee-app/

├── index.html

└── employees.json

**Step 2: Create employees.json**

Create a file named employees.json inside the folder. Add the following sample JSON content:

**PS C:\Users\VAAGDEVI\Downloads\employee-app> code employees.json**

[

{

"id": 1,

"name": "Alice Johnson",

"position": "Software Engineer",

"department": "Development"

},

{

"id": 2,

"name": "Bob Smith",

"position": "Project Manager",

"department": "Management"

}

]

**Step 3: Create index.html**

Create a file named index.html and paste the following AngularJS app and UI:

PS C:\Users\VAAGDEVI\Downloads\employee-app> code index.html

<!DOCTYPE html>

<html ng-app="employeeApp">

<head>

<meta charset="UTF-8">

<title>Employee List</title>

<!-- Load AngularJS from CDN -->

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<script>

// Define the AngularJS application

var app = angular.module('employeeApp', []);

// Create a controller

app.controller('EmployeeController', function($scope, $http) {

$scope.employees = [];

// Use $http to fetch JSON data

$http.get('employees.json')

.then(function(response) {

$scope.employees = response.data;

}, function(error) {

console.error('Error fetching employee data:', error);

});

});

</script>

</head>

<body ng-controller="EmployeeController">

<h1>Employee List</h1>

<table border="1" cellpadding="10">

<tr>

<th>ID</th>

<th>Name</th>

<th>Position</th>

<th>Department</th>

</tr>

<tr ng-repeat="employee in employees">

<td>{{ employee.id }}</td>

<td>{{ employee.name }}</td>

<td>{{ employee.position }}</td>

<td>{{ employee.department }}</td>

</tr>

</table>

</body>

</html>

}]);

**Step 8: Running the Application**

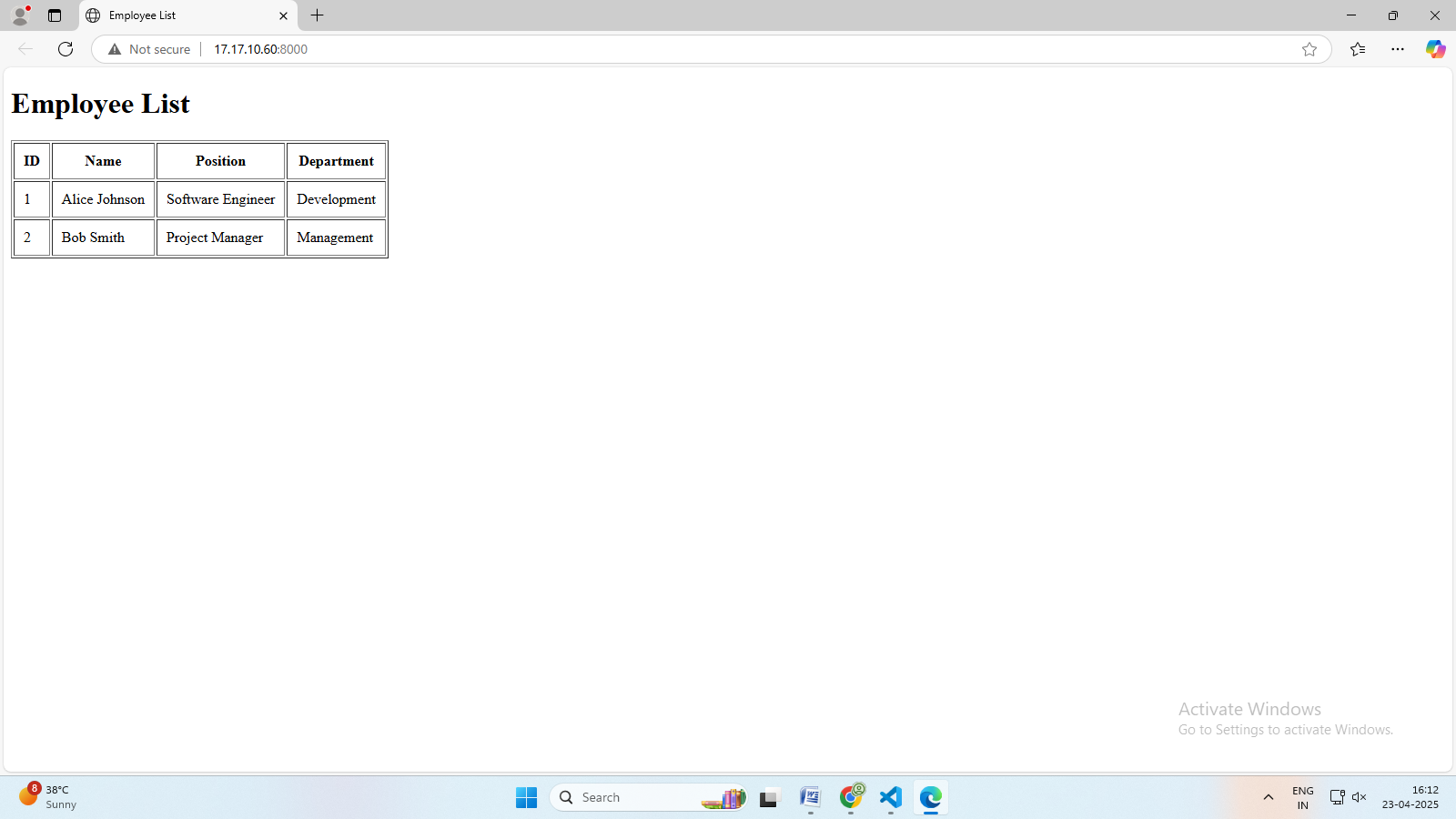
PS C:\Users\VAAGDEVI\Downloads\employee-app> npm install -g http-server

**added 48 packages in 2s**

**15 packages are looking for funding**

**run `npm fund` for details**

**PS C:\Users\VAAGDEVI\Downloads\employee-app> http-server -p 8000**

****

**WEEK 12: Develop a web application to manage student information using Express and Angular JS.**

A web application to manage student information using Express.js for the backend and AngularJS for the frontend.

student-management /

│

├── backend/

│ ├── package.json # Node.js dependencies and scripts

│ └── server.js # Express server with student API

│

└── frontend/

├── index.html # Main AngularJS HTML file

├── app.js # AngularJS application logic

└── style.css # Styles for the frontend

Explanation:

backend/: Contains all server-side code and configuration.

package.json: Lists dependencies (express, body-parser, cors).

server.js: Implements the REST API for student management.

frontend/: Contains all client-side code.

index.html: The main HTML file for your AngularJS app.

app.js: Contains AngularJS code for interacting with the backend API.

style.css: Stylesheet for the frontend UI.

First, let's create the backend using Express.js:

1. Create a directory for your project and navigate into it:

mkdir student-management

cd student-manager

2. Create a folder inside the student-manger

mkdir backend

PS C:\Users\VAAGDEVI\student-management> mkdir backend

Directory: C:\Users\VAAGDEVI\student-management

Mode LastWriteTime Length Name

---- ------------- ------ ----

d----- 23-04-2025 14:12 backend

3. Initialize a Node.js project:

PS C:\Users\VAAGDEVI\student-management> cd backend

PS C:\Users\VAAGDEVI\student-management\backend> npm init -y

Wrote to C:\Users\VAAGDEVI\student-management\backend\package.json:

{

"name": "backend",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC",

"description": ""

}

PS C:\Users\VAAGDEVI\student-management\backend> code server.js

const express = require('express');

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

const cors = require('cors');

const app = express();

const PORT = 3010;

app.use(cors());

app.use(bodyParser.json());

// Predefined student data

let students = [

{ rollNumber: "101", name: "Aarav", department: "CSE" },

{ rollNumber: "102", name: "Diya", department: "ECE" },

{ rollNumber: "103", name: "Rohan", department: "EEE" },

{ rollNumber: "104", name: "Sneha", department: "IT" },

{ rollNumber: "105", name: "Manoj", department: "MECH" }

];

// Optional: Root route

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

res.send('Student Management Backend is running');

});

// Get all students

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

res.json(students);

});

// Add new student

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

const student = req.body;

students.push(student);

res.status(201).json(student);

});

// Update student

app.put('/api/students/:rollNumber', (req, res) => {

const { rollNumber } = req.params;

const updatedStudent = req.body;

let found = false;

students = students.map(s => {

if (s.rollNumber === rollNumber) {

found = true;

return updatedStudent;

}

return s;

});

if (found) {

res.json(updatedStudent);

} else {

res.status(404).json({ message: 'Student not found' });

}

});

// Delete student

app.delete('/api/students/:rollNumber', (req, res) => {

const { rollNumber } = req.params;

const initialLength = students.length;

students = students.filter(s => s.rollNumber !== rollNumber);

if (students.length < initialLength) {

res.json({ message: 'Student deleted' });

} else {

res.status(404).json({ message: 'Student not found' });

}

});

app.listen(PORT, () => {

console.log(`Server running on http://localhost:${PORT}`);

});

Ctrl+S Save the File

PS C:\Users\VAAGDEVI\student-management\backend> npm install express

added 66 packages, and audited 67 packages in 2s

14 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

PS C:\Users\VAAGDEVI\student-management\backend>cd..

PS C:\Users\VAAGDEVI\student-management> mkdir frontend

Directory: C:\Users\VAAGDEVI\student-management

Mode LastWriteTime Length Name

---- ------------- ------ ----

d----- 23-04-2025 14:16 frontend

PS C:\Users\VAAGDEVI\student-management> cd frontend

PS C:\Users\VAAGDEVI\student-management\frontend> code app.js

angular.module('studentApp', [])

.controller('StudentController', function($http) {

const vm = this;

vm.students = [];

vm.student = {};

vm.editMode = false;

vm.getStudents = function() {

$http.get('http://localhost:3010/api/students').then(resp => {

vm.students = resp.data;

});

};

vm.addOrUpdateStudent = function() {

if (vm.editMode) {

$http.put('http://localhost:3010/api/students/' + vm.student.rollNumber, vm.student).then(() => {

vm.getStudents();

vm.resetForm();

});

} else {

$http.post('http://localhost:3010/api/students', vm.student).then(() => {

vm.getStudents();

vm.resetForm();

});

}

};

vm.editStudent = function(student) {

vm.student = angular.copy(student);

vm.editMode = true;

};

vm.deleteStudent = function(rollNumber) {

$http.delete('http://localhost:3010/api/students/' + rollNumber).then(() => {

vm.getStudents();

});

};

vm.resetForm = function() {

vm.student = {};

vm.editMode = false;

};

vm.getStudents();

});

Ctrl+S Save the File

PS C:\Users\VAAGDEVI\student-management\frontend> code index.html

<!DOCTYPE html>

<html ng-app="studentApp">

<head>

<meta charset="UTF-8">

<title>Student Manager</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

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

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

</head>

<body ng-controller="StudentController as ctrl">

<div class="container">

<h1>Student Manager</h1>

<form ng-submit="ctrl.addOrUpdateStudent()">

<input type="text" ng-model="ctrl.student.name" placeholder="Name" required />

<input type="text" ng-model="ctrl.student.rollNumber" placeholder="Roll Number" required ng-readonly="ctrl.editMode" />

<input type="text" ng-model="ctrl.student.class" placeholder="Class" required />

<input type="text" ng-model="ctrl.student.section" placeholder="Section" required />

<input type="email" ng-model="ctrl.student.email" placeholder="Email" required />

<button type="submit">{{ctrl.editMode ? 'Update' : 'Add'}} Student</button>

<button type="button" ng-click="ctrl.resetForm()" ng-if="ctrl.editMode">Cancel</button>

</form>

<table>

<thead>

<tr>

<th>Name</th>

<th>Roll Number</th>

<th>Class</th>

<th>Section</th>

<th>Email</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<tr ng-repeat="student in ctrl.students">

<td>{{student.name}}</td>

<td>{{student.rollNumber}}</td>

<td>{{student.class}}</td>

<td>{{student.section}}</td>

<td>{{student.email}}</td>

<td>

<button ng-click="ctrl.editStudent(student)">Edit</button>

<button ng-click="ctrl.deleteStudent(student.rollNumber)">Delete</button>

</td>

</tr>

</tbody>

</table>

</div>

</body>

</html>

Ctrl+S Save the File

PS C:\Users\VAAGDEVI\student-management\frontend> code style.css

body { font-family: Arial, sans-serif; background: #f8f9fa; }

.container { max-width: 800px; margin: 40px auto; background: #fff; padding: 24px; border-radius: 8px; box-shadow: 0 2px 8px rgba(0,0,0,0.1); }

h1 { text-align: center; }

form { margin-bottom: 24px; display: flex; flex-wrap: wrap; gap: 8px; }

form input { flex: 1 1 150px; padding: 8px; border: 1px solid #ccc; border-radius: 4px; }

form button { padding: 8px 16px; border: none; border-radius: 4px; background: #007bff; color: #fff; cursor: pointer; }

form button[ng-if] { background: #6c757d; }

table { width: 100%; border-collapse: collapse; }

th, td { padding: 8px; border: 1px solid #ddd; text-align: left; }

th { background: #f1f1f1; }

td button { margin-right: 8px; padding: 4px 8px; border: none; border-radius: 4px; background: #28a745; color: #fff; cursor: pointer; }

td button:last-child { background: #dc3545; }

Ctrl+S Save the File

PS C:\Users\VAAGDEVI\student-management\frontend> cd..

PS C:\Users\VAAGDEVI\student-management> cd backend

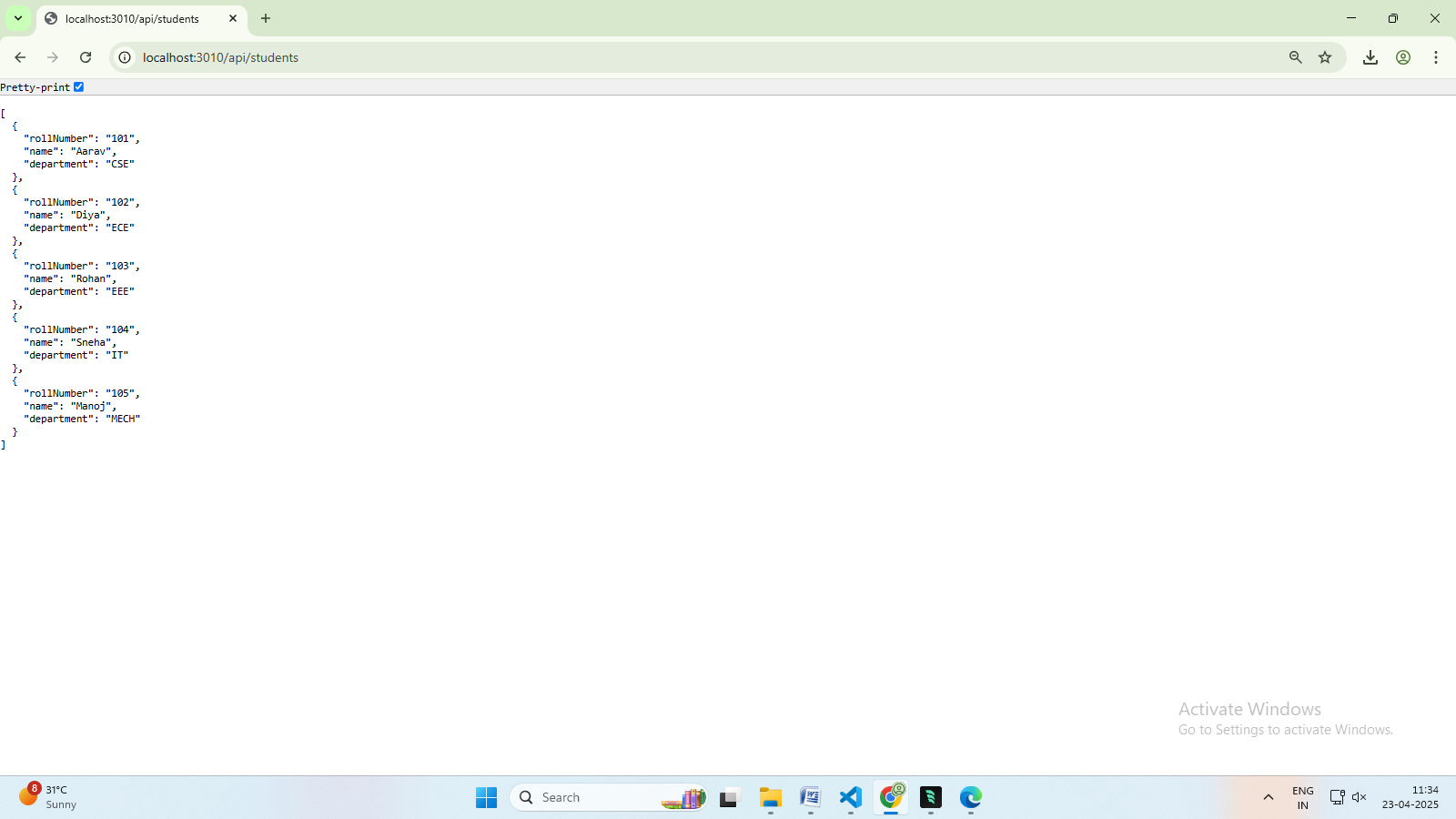
PS C:\Users\VAAGDEVI\student-management\backend> npm install cors

PS C:\Users\VAAGDEVI\student-management\backend> node server.js

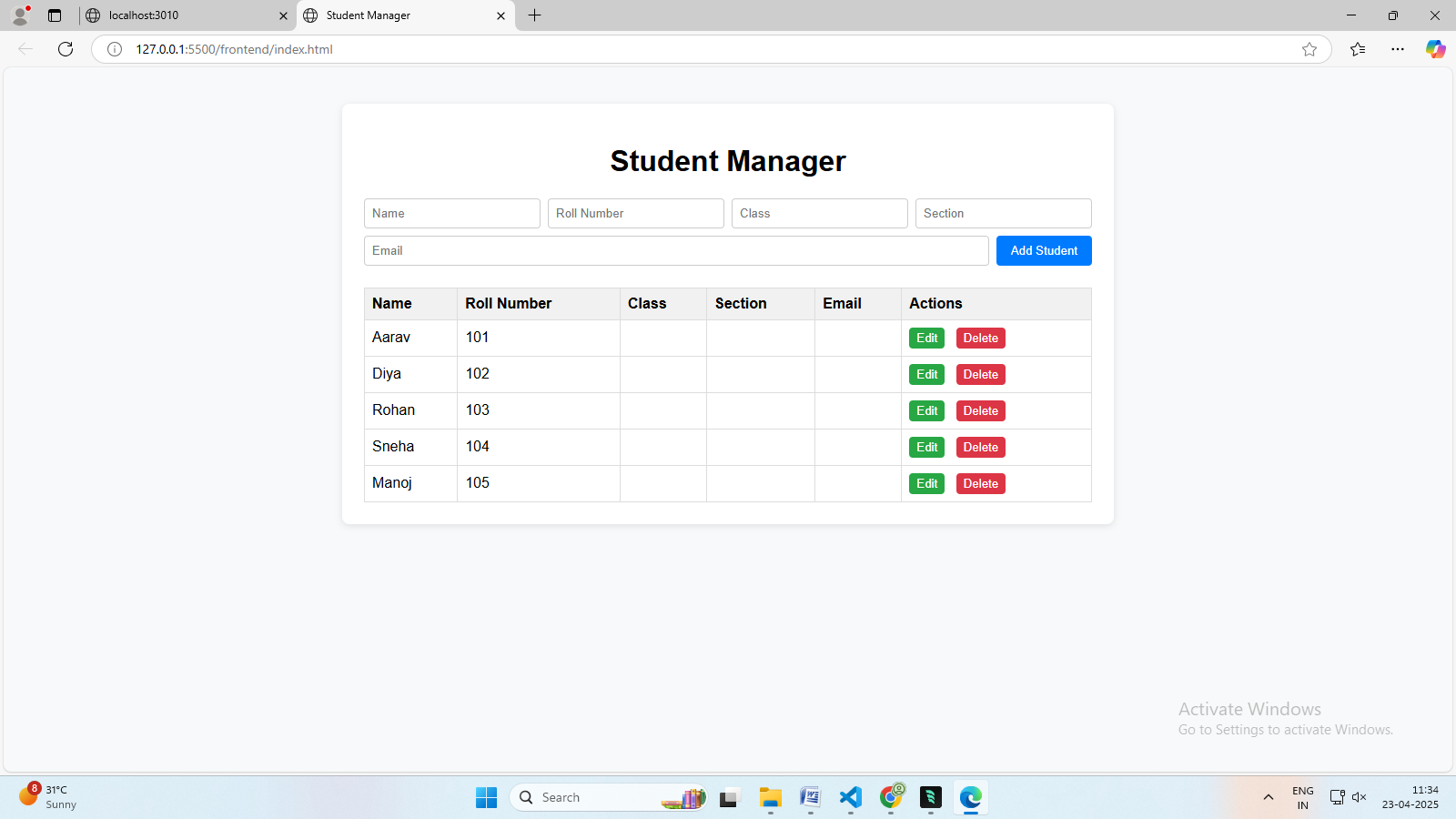
Server running on <http://localhost:3010>

Open Browser and enter the Url:

<http://localhost:3010/api/students>



Open index.html with Live Server:



**Week-13:** **Write a program to create a simple calculator Application using React JS.**

1. Project Structure

calculator-app/

├── public/

│ └── index.html

├── src/

│ ├── components/

│ │ └── Calculator.js

│ ├── App.js

│ ├── index.js

│ └── styles.css

├── package.json

└── README.md

**Step 1:** Set Up React Project

npx create-react-app calculator-app

cd calculator-app

**Step 2: Project Setup**

1. **Go to the src folder and create a folder named components.**
2. Inside the components folder, create a file named Calculator.js.
3. Add a CSS file styles.css for basic styling.

**Step 3: Create the Calculator Component**

// src/components/Calculator.js

import React, { useState } from 'react';

import '../styles.css';

const Calculator = () => {

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

const handleClick = (value) => {

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

};

const handleClear = () => {

setInput('');

};

const handleEvaluate = () => {

try {

setInput(eval(input).toString());

} catch (error) {

setInput('Error');

}

};

return (

<div className="calculator">

<div className="display">{input || '0'}</div>

<div className="buttons">

{['7', '8', '9', '/'].map((btn) => (

<button key={btn} onClick={() => handleClick(btn)}>{btn}</button>

))}

{['4', '5', '6', '\*'].map((btn) => (

<button key={btn} onClick={() => handleClick(btn)}>{btn}</button>

))}

{['1', '2', '3', '-'].map((btn) => (

<button key={btn} onClick={() => handleClick(btn)}>{btn}</button>

))}

{['0', '.', '=', '+'].map((btn) => (

<button

key={btn}

onClick={() => btn === '=' ? handleEvaluate() : handleClick(btn)}

>

{btn}

</button>

))}

<button className="clear" onClick={handleClear}>C</button>

</div>

</div>

);

};

export default Calculator;

Step 4: Add Styles

src/styles.css

.calculator {

width: 300px;

margin: 50px auto;

padding: 20px;

background: #f3f3f3;

border-radius: 10px;

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

}

.display {

background: #222;

color: #fff;

font-size: 2em;

text-align: right;

padding: 10px;

border-radius: 5px;

margin-bottom: 10px;

min-height: 50px;

}

.buttons button {

width: 22%;

padding: 15px;

margin: 1%;

font-size: 1.5em;

cursor: pointer;

border: none;

border-radius: 5px;

background: #ddd;

transition: background 0.3s;

}

.buttons button:hover {

background: #ccc;

}

.clear {

background: red;

color: white;

}

.clear:hover {

background: darkred;

}

Step 5: Update App.js

// src/App.js

import React from 'react';

import Calculator from './components/Calculator';

function App() {

return (

<div className="App">

<Calculator />

</div>

);

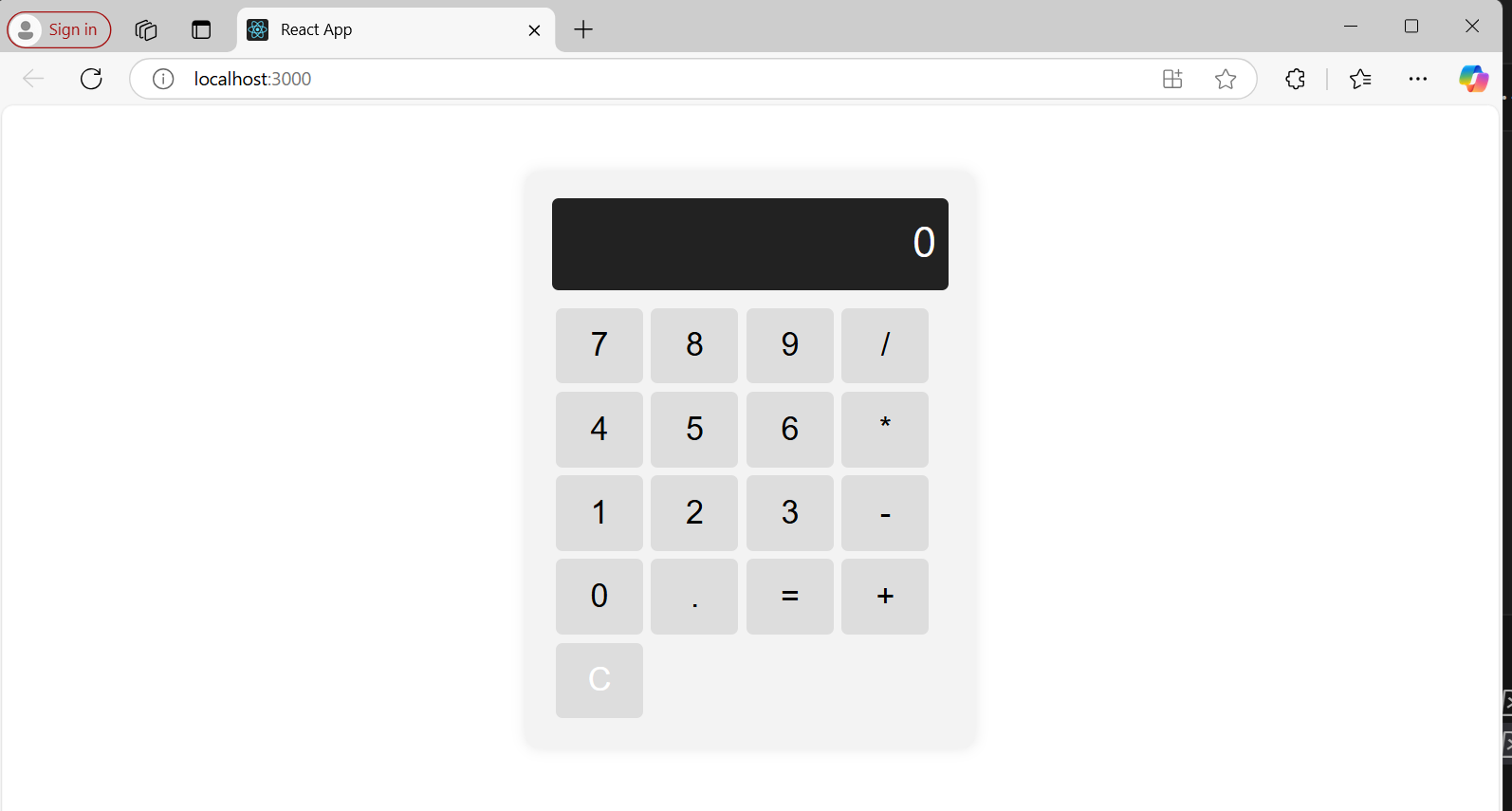
}

export default App;

**Step 6: Run the App**

npm start

This will start your React app at http://localhost:3000.



**Week-14: Write a program to create a voting application using React JS**

voting-app/

├── public/

│ └── index.html

├── src/

│ ├── components/

│ │ ├── CandidateList.js

│ │ ├── Candidate.js

│ │ └── VoteButton.js

│ ├── App.js

│ ├── index.js

│ └── App.css

├── package.json

└── README.md

**Step 1: Setting Up the Project**

npx create-react-app voting-app

cd voting-app

npm start

**Step 2: Creating Components**

**src/components/Candidate.js**

import React from 'react';

const Candidate = ({ candidate, onVote }) => {

return (

<div className="candidate">

<h3>{candidate.name}</h3>

<p>Votes: {candidate.votes}</p>

<button onClick={() => onVote(candidate.id)}>Vote</button>

</div>

);

};

export default Candidate;

**Step:-2** src/components/VoteButton.js

import React from 'react';

const VoteButton = ({ onVote }) => {

return (

<button onClick={onVote} className="vote-button">

Vote

</button>

);

};

export default VoteButton;

**Step:-3** src/components/CandidateList.js

import React from 'react';

import Candidate from './Candidate';

const CandidateList = ({ candidates, onVote }) => {

return (

<div>

{candidates.map((candidate) => (

<Candidate key={candidate.id} candidate={candidate} onVote={onVote} />

))}

</div>

);

};

export default CandidateList;

**Step 4: The Main App Component**

src/App.js

import React, { useState } from 'react';

import CandidateList from './components/CandidateList';

import './App.css';

function App() {

const [candidates, setCandidates] = useState([

{ id: 1, name: 'Alice', votes: 0 },

{ id: 2, name: 'Bob', votes: 0 },

{ id: 3, name: 'Charlie', votes: 0 },

]);

const handleVote = (id) => {

const updatedCandidates = candidates.map((candidate) =>

candidate.id === id

? { ...candidate, votes: candidate.votes + 1 }

: candidate

);

setCandidates(updatedCandidates);

};

return (

<div className="App">

<h1>Voting Application</h1>

<CandidateList candidates={candidates} onVote={handleVote} />

</div>

);

}

export default App;

**Step 5: Basic Styling**

src/App.css

.App {

text-align: center;

font-family: Arial, sans-serif;

margin: 20px;

}

.candidate {

border: 1px solid #ccc;

padding: 15px;

margin: 10px auto;

width: 300px;

background-color: #f9f9f9;

border-radius: 8px;

}

button {

padding: 10px 20px; background-color: #007bff;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

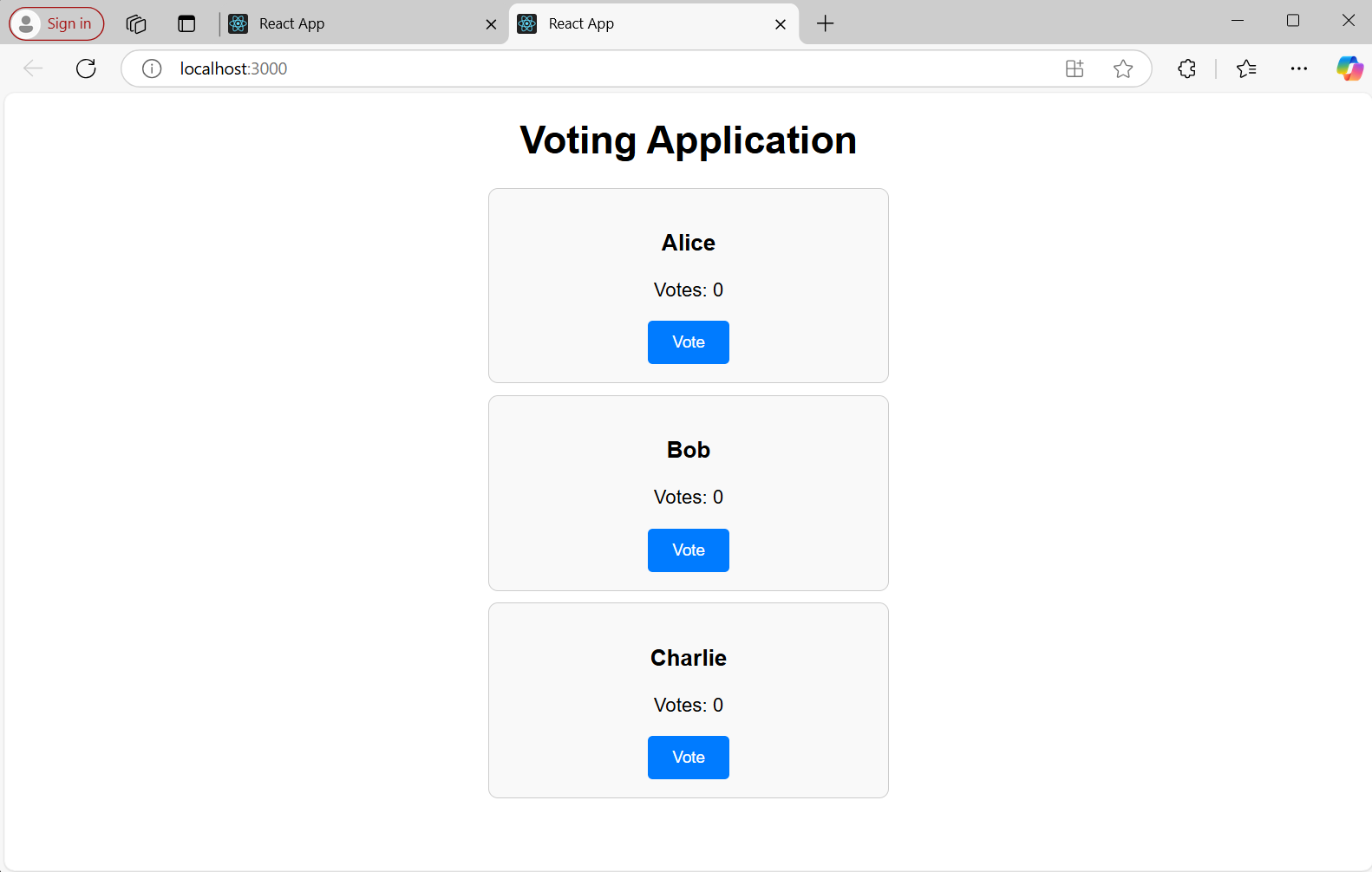
button:hover {

background-color: #0056b3;

}

**Step 6: Run the Application**

npm start



**Week-15: Develop a leave management system for an organization where users can apply different types of leaves such as casual leave and medical leave. They also can view the available number of days using react application.**

1. Project Structure

leave-management-system/

├── public/

│ └── index.html

├── src/

│ ├── components/

│ │ ├── LeaveForm.js

│ │ ├── LeaveList.js

│ │ └── Dashboard.js

│ ├── App.js

│ ├── index.js

│ ├── services/

│ │ └── leaveService.js

│ └── styles/

│ └── App.css

├── .gitignore

├── package.json

└── README.md

**2. Setting Up the React Project**

1. **Create the React App:**

npx create-react-app leave-management-system

cd leave-management-system

1. Install Axios (for API requests):

npm install axios

1. Backend (Simulated with JSON Server for Simplicity)

npm install -g json-server

**Create a db.json file:**

{

"leaves": [

{ "id": 1, "type": "Casual Leave", "days": 5, "status": "Approved" },

{ "id": 2, "type": "Medical Leave", "days": 3, "status": "Pending" }

]

}

Run the JSON Server:

json-server --watch db.json --port 5000

**4. Frontend Implementation**

**App.js**

**import React from 'react';**

**import Dashboard from './components/Dashboard';**

**import './styles/App.css';**

**function App() {**

**return (**

**<div className="App">**

**<h1>Leave Management System</h1>**

**<Dashboard />**

**</div>**

**);**

**}**

**export default App;**

**Dashboard.js**

import React, { useState, useEffect } from 'react';

import LeaveForm from './LeaveForm';

import LeaveList from './LeaveList';

import leaveService from '../services/leaveService';

const Dashboard = () => {

const [leaves, setLeaves] = useState([]);

useEffect(() => {

fetchLeaves();

}, []);

const fetchLeaves = async () => {

const data = await leaveService.getLeaves();

setLeaves(data);

};

return (

<div>

<LeaveForm onLeaveAdded={fetchLeaves} />

<LeaveList leaves={leaves} />

</div>

);

};

export default Dashboard;

**LeaveForm.js**

import React, { useState } from 'react';

import leaveService from '../services/leaveService';

const LeaveForm = ({ onLeaveAdded }) => {

const [type, setType] = useState('');

const [days, setDays] = useState('');

const handleSubmit = async (e) => {

e.preventDefault();

await leaveService.addLeave({ type, days: parseInt(days), status: 'Pending' });

setType('');

setDays('');

onLeaveAdded();

};

return (

<form onSubmit={handleSubmit}>

<h2>Apply for Leave</h2>

<label>Type of Leave:</label>

<input type="text" value={type} onChange={(e) => setType(e.target.value)} required />

<label>Number of Days:</label>

<input type="number" value={days} onChange={(e) => setDays(e.target.value)} required />

<button type="submit">Apply Leave</button>

</form>

);

};

export default LeaveForm;

**LeaveList.js**

import React from 'react';

const LeaveList = ({ leaves }) => {

return (

<div>

<h2>Leave Records</h2>

<ul>

{leaves.map((leave) => (

<li key={leave.id}>

{leave.type} - {leave.days} days - {leave.status}

</li>

))}

</ul>

</div>

);

};

export default LeaveList;

**leaveService.js**

import axios from 'axios';

const API\_URL = 'http://localhost:5000/leaves';

const getLeaves = () => axios.get(API\_URL).then((res) => res.data);

const addLeave = (leave) => axios.post(API\_URL, leave);

export default { getLeaves, addLeave };

1. **Styling (App.css)**

body {

font-family: Arial, sans-serif;

background-color: #f9f9f9;

padding: 20px;

text-align: center;

}

form {

background-color: #fff;

padding: 20px;

border-radius: 5px;

margin: 20px auto;

width: 300px;

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

}

input, button {

width: 100%;

padding: 10px;

margin: 10px 0;

border: 1px solid #ddd;

border-radius: 4px;

}

button {

background-color: #007bff;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

**6. Run the Application**

**1. Start the React App:**

npm start

**2. Start the JSON Server (if using):**

json-server --watch db.json --port 5000

**Week-16: Build a music store application using react components and provide routing among the web pages.**

music-store/

├── public/

│ └── index.html

├── src/

│ ├── components/

│ │ ├── Header.js

│ │ ├── Footer.js

│ │ ├── Home.js

│ │ ├── Products.js

│ │ ├── ProductDetail.js

│ │ └── Cart.js

│ ├── App.js

│ ├── index.js

│ └── data.js

├── package.json

└── README.md

**Step 1: Setting Up the React Project**

1. **Create the React App:**

npx create-react-app music-store

cd music-store

**2.Install React Router:**

npm install react-router-dom

**Step 2: Setting Up React Router**

**Edit src/App.js to define the routes:**

import React from "react";

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Header from "./components/Header";

import Footer from "./components/Footer";

import Home from "./components/Home";

import Products from "./components/Products";

import ProductDetail from "./components/ProductDetail";

import Cart from "./components/Cart";

function App() {

return (

<Router>

<Header />

<Routes>

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

<Route path="/products" element={<Products />} />

<Route path="/products/:id" element={<ProductDetail />} />

<Route path="/cart" element={<Cart />} />

</Routes>

<Footer />

</Router>

);

}

export default App;

**Step 3: Creating Components**

**1.Header (src/components/Header.js):**

import React from "react";

import { Link } from "react-router-dom";

const Header = () => (

<header>

<h1>Music Store</h1>

<nav>

<Link to="/">Home</Link> |

<Link to="/products">Products</Link> |

<Link to="/cart">Cart</Link>

</nav>

</header>

);

export default Header;

**2. Footer (src/components/Footer.js):**

import React from "react";

const Footer = () => (

<footer>

<p>&copy; 2025 Music Store. All rights reserved.</p>

</footer>

);

export default Footer;

**3. Home (src/components/Home.js):**

import React from "react";

const Home = () => (

<div>

<h2>Welcome to Music Store!</h2>

<p>Find the best music instruments and albums here.</p>

</div>

);

export default Home;

**4.Products (src/components/Products.js):**

import React from "react";

import { Link } from "react-router-dom";

import data from "../data";

const Products = () => (

<div>

<h2>Products</h2>

<ul>

{data.products.map((product) => (

<li key={product.id}>

<Link to={`/products/${product.id}`}>{product.name}</Link>

</li>

))}

</ul>

</div>

);

export default Products;

**5.ProductDetail (src/components/ProductDetail.js):**

import React from "react";

import { useParams } from "react-router-dom";

import data from "../data";

const ProductDetail = () => {

const { id } = useParams();

const product = data.products.find((p) => p.id === parseInt(id));

if (!product) return <p>Product not found!</p>;

return (

<div>

<h2>{product.name}</h2>

<p>{product.description}</p>

<p>Price: ${product.price}</p>

</div>

);

};

export default ProductDetail;

**6.Cart (src/components/Cart.js):**

import React from "react";

const Cart = () => (

<div>

<h2>Your Cart</h2>

<p>Your cart is empty.</p>

</div>

);

export default Cart;

**Step 4: Sample Data (src/data.js)**

const data = {

products: [

{ id: 1, name: "Guitar", description: "Acoustic Guitar", price: 299 },

{ id: 2, name: "Piano", description: "Digital Piano", price: 499 },

{ id: 3, name: "Drum Set", description: "5-Piece Drum Set", price: 699 },

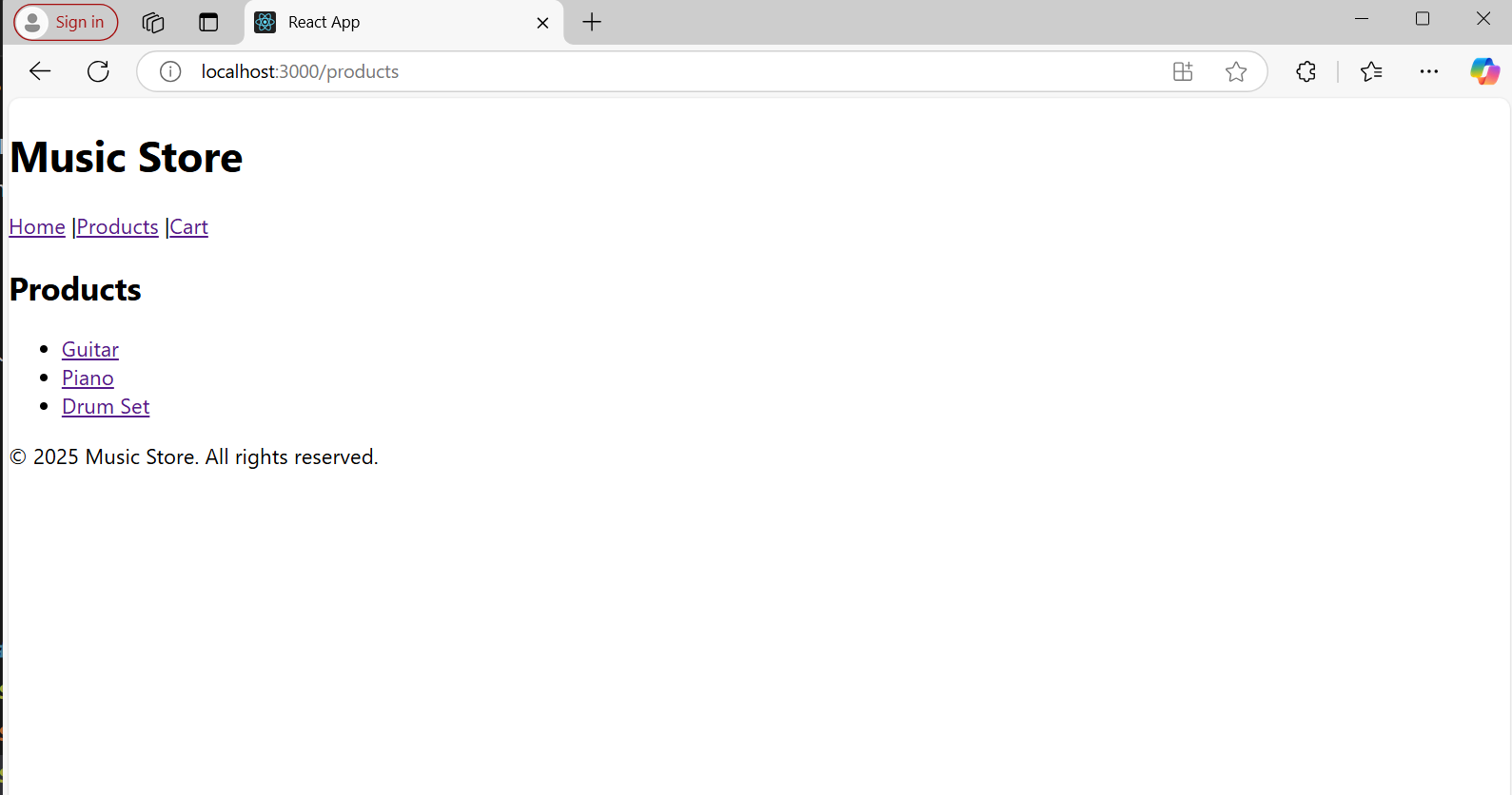
],

};

export default data;

**Step 5: Run the App**

npm start



**WEEK-17: Create a react application for an online store which consist of registration, login, product information pages and implements routing to navigate through these pages**

Project Structure

online-store/

├── public/

│ └── index.html

├── src/

│ ├── components/

│ │ ├── Register.js

│ │ ├── Login.js

│ │ ├── ProductInfo.js

│ ├── App.js

│ ├── index.js

│ └── App.css

├── package.json

└── README.md

Step 1: Create the React App

PS C:\Users\VAAGDEVI\Desktop> npx create-react-app online-store

Need to install the following packages:

create-react-app@5.1.0

Ok to proceed? (y) y

Creating a new React app in C:\Users\VAAGDEVI\Desktop\online-store.

Installing packages. This might take a couple of minutes.

Installing react, react-dom, and react-scripts with cra-template...

added 1324 packages in 42s

PS C:\Users\VAAGDEVI\Desktop> cd online-store

Step 2: Install React Router

**What is react-router-dom?**

It is a **package** that allows your React app to handle **multiple pages (routes)** without reloading the whole page — just like a single-page app should behave.

PS C:\Users\VAAGDEVI\Desktop\online-store> npm install react-router-dom

added 5 packages, and audited 1347 packages in 3s

268 packages are looking for funding

run `npm fund` for details

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:

npm audit fix --force

Run `npm audit` for details.npm install react-router-dom

**Step 3: Create Component Files**

**PS C:\Users\VAAGDEVI\Desktop\online-store> mkdir src/components**

**Directory: C:\Users\VAAGDEVI\Desktop\online-store\src**

**Mode LastWriteTime Length Name**

**---- ------------- ------ ----**

**d----- 24-04-2025 13:18 components**

**PS C:\Users\VAAGDEVI\Desktop\online-store> cd src**

**PS C:\Users\VAAGDEVI\Desktop\online-store\src> cd components**

Inside src/components/, create:

* Register.js
* Login.js
* ProductInfo.js

### Step 4: Add Component Code

#### PS C:\Users\VAAGDEVI\Desktop\online-store\src\components> code Register.js

Register.js

import React from 'react';

const Register = () => {

  return (

    <div>

      <h2>Register</h2>

      <form>

        <input type="text" placeholder="Username" required /><br /><br />

        <input type="email" placeholder="Email" required /><br /><br />

        <input type="password" placeholder="Password" required /><br /><br />

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

      </form>

    </div>

  );

};

export default Register;

Ctrl+S Save the Code

Login.js

PS C:\Users\VAAGDEVI\Desktop\online-store\src\components> code Login.js

import React from 'react';

const Login = () => {

  return (

    <div>

      <h2>Login</h2>

      <form>

        <input type="email" placeholder="Email" required /><br /><br />

        <input type="password" placeholder="Password" required /><br /><br />

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

      </form>

    </div>

  );

};

export default Login;

ProductInfo.js

import React from 'react';

const products = [

  {

    name: 'Laptop',

    description: 'A high performance laptop.',

    price: '$999'

  },

  {

    name: 'Phone',

    description: 'A smartphone with a great camera.',

    price: '$599'

  },

  {

    name: 'Headphones',

    description: 'Noise cancelling headphones.',

    price: '$199'

  }

];

const ProductInfo = () => {

  return (

    <div style={{ display: 'flex', flexDirection: 'column', alignItems: 'center', padding: '2rem' }}>

      <h2>Product Information</h2>

      <div style={{ display: 'flex', gap: '1rem', flexWrap: 'wrap', justifyContent: 'center' }}>

        {products.map((product, index) => (

          <div key={index} style={{

            border: '1px solid #ccc',

            borderRadius: '8px',

            padding: '1rem',

            width: '200px',

            textAlign: 'center'

          }}>

            <h3>{product.name}</h3>

            <p>{product.description}</p>

            <strong>{product.price}</strong>

          </div>

        ))}

      </div>

    </div>

  );

};

export default ProductInfo;

Step 5: Set Up Routing in App.js

In src go to App.js and update the code

**PS C:\Users\VAAGDEVI\Desktop\online-store\src\components> cd..**

**PS C:\Users\VAAGDEVI\Desktop\online-store\src> code App.js**

App.js

import React from 'react';

import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

import Register from './components/Register';

import Login from './components/Login';

import ProductInfo from './components/ProductInfo';

import './App.css';

function App() {

  return (

    <Router>

      <div className="App">

        <nav style={{ marginBottom: '20px' }}>

          <Link to="/register">Register</Link> |{' '}

          <Link to="/login">Login</Link> |{' '}

          <Link to="/product">Product Info</Link>

        </nav>

        <Routes>

          <Route path="/register" element={<Register />} />

          <Route path="/login" element={<Login />} />

          <Route path="/product" element={<ProductInfo />} />

        </Routes>

      </div>

    </Router>

  );

}

export default App;

**Step 6: Run the App**

**PS C:\Users\VAAGDEVI\Desktop\online-store\src> npm start**

