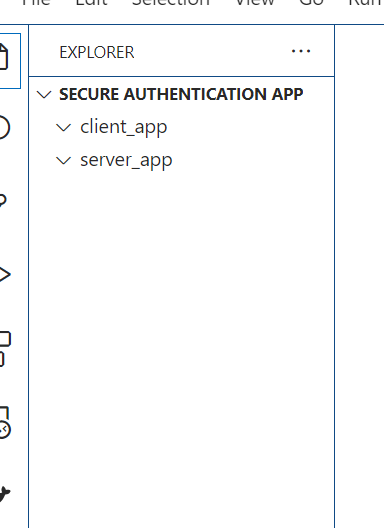
**Answer Key**

**Step 1: Create the project structure.**

* 1. Create the Folder as Secure\_Authentication\_App which contains two sub folder as client\_app and server\_app
  2. Then open the folder in VS Code editor

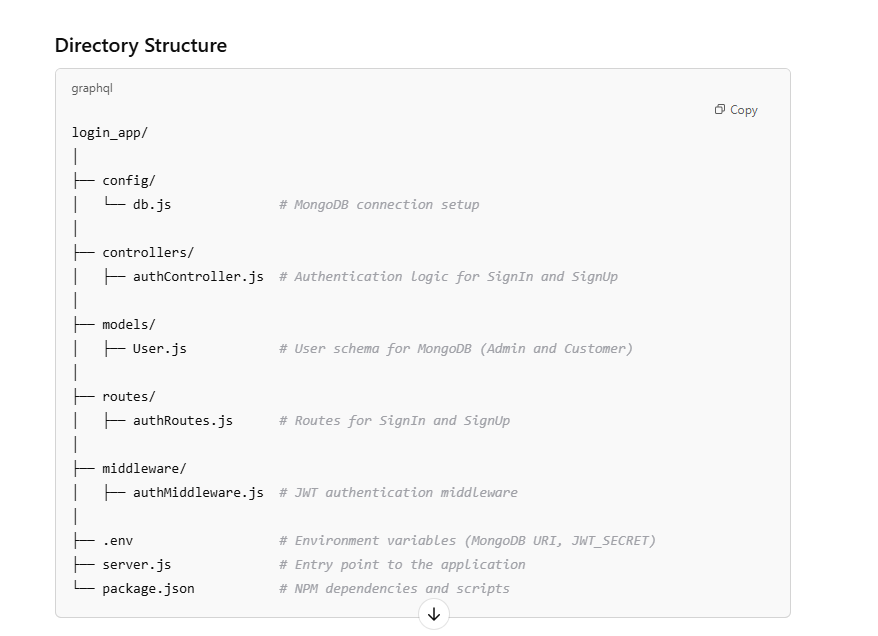


**Step 2: Create the end point using backend technology with help of chat gpt.**

* 1. Open the chat gpt and ask the prompt message

“Provide the directory structure as well as code for login\_app using Express.js with MongoDB database for admin as well as customer for the SignIn and SignUp endpoints”

* 1. The directory structure



* 1. follow all these steps to create the project.
  2. Install Dependencies inside a server\_app folder

**npm init -y**

**npm install express mongoose bcryptjs jsonwebtoken dotenv body-parser**

* 1. .env file

PORT=5000

MONGO\_URI=mongodb://localhost:27017/login\_app

JWT\_SECRET=your\_secret\_key

* 1. config/db.js - MongoDB Connection

const mongoose = require('mongoose');

require('dotenv').config();

const connectDB = async () => {

    try {

        await mongoose.connect(process.env.MONGO\_URI);

        console.log('MongoDB connected...');

    } catch (error) {

        console.error(error);

        process.exit(1);

    }

};

module.exports = connectDB;

* 1. models/User.js - User Model (Admin & Customer)

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({

    username: {

        type: String,

        required: true,

        unique: true

    },

    password: {

        type: String,

        required: true

    },

    role: {

        type: String,

        enum: ['admin', 'customer'],

        default: 'customer'

    }

});

// Hash password before saving

userSchema.pre('save', async function(next) {

    if (!this.isModified('password')) {

        return next();

    }

    this.password = await bcrypt.hash(this.password, 10);

});

// Method to check password

userSchema.methods.matchPassword = async function(enteredPassword) {

    return await bcrypt.compare(enteredPassword, this.password);

};

const User = mongoose.model('User', userSchema);

module.exports = User;

* 1. controllers/authController.js - Authentication Logic

const jwt = require('jsonwebtoken');

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

// Sign Up

const signup = async (req, res) => {

    const { username, password, role } = req.body;

    try {

        const userExists = await User.findOne({ username });

        if (userExists) {

            return res.status(400).json({ message: 'User already exists' });

        }

        const user = new User({ username, password, role });

        await user.save();

        res.status(201).json({ message: 'User created successfully', user });

    } catch (error) {

        res.status(500).json({ message: 'Server error' });

    }

};

// Sign In

const signin = async (req, res) => {

    const { username, password } = req.body;

    try {

        const user = await User.findOne({ username });

        if (!user) {

            return res.status(401).json({ message: 'Invalid credentials' });

        }

        const isMatch = await user.matchPassword(password);

        if (!isMatch) {

            return res.status(401).json({ message: 'Invalid credentials' });

        }

        const token = jwt.sign(

            { id: user.\_id, username: user.username, role: user.role },

            process.env.JWT\_SECRET,

            { expiresIn: '1h' }

        );

        res.status(200).json({ message: 'Login successful', token });

    } catch (error) {

        res.status(500).json({ message: 'Server error' });

    }

};

module.exports = { signup, signin };

* 1. routes/authRoutes.js - Routes for SignUp and SignIn

const express = require('express');

const { signup, signin } = require('../controllers/authController');

const router = express.Router();

// Sign Up Route

router.post('/signup', signup);

// Sign In Route

router.post('/signin', signin);

module.exports = router;

* 1. middleware/authMiddleware.js - JWT Authentication Middleware

const jwt = require('jsonwebtoken');

const protect = (req, res, next) => {

    const token = req.header('Authorization')?.split(' ')[1];

    if (!token) {

        return res.status(401).json({ message: 'No token, authorization denied' });

    }

    try {

        const decoded = jwt.verify(token, process.env.JWT\_SECRET);

        req.user = decoded;

        next();

    } catch (error) {

        res.status(401).json({ message: 'Token is not valid' });

    }

};

module.exports = { protect };

* 1. server.js - Entry Point to the Application

const express = require('express');

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

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

const authRoutes = require('./routes/authRoutes');

require('dotenv').config();

const app = express();

// Middleware

app.use(bodyParser.json());

// Connect to database

connectDB();

// Routes

app.use('/api/auth', authRoutes);

// Start the server

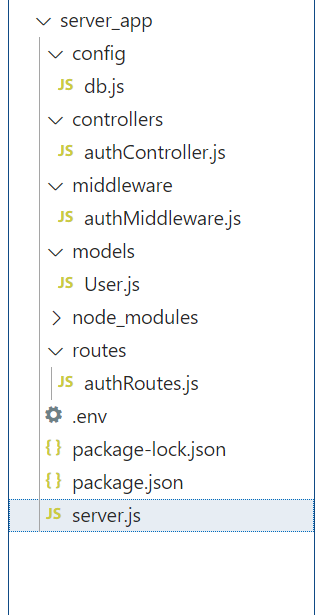
const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

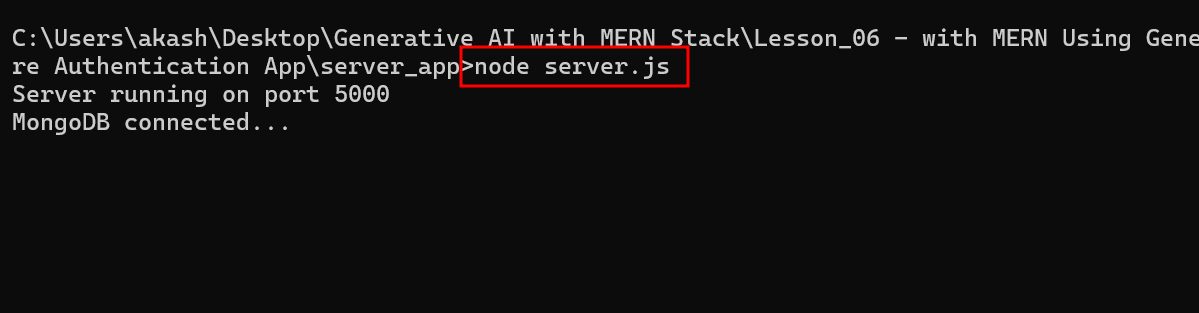
    console.log(`Server running on port ${PORT}`);

});

* 1. after created folder and file in VS code. The folder structure as



2.13 Run the application using node server.js file



**Step 3: Testing backend technology using post man client.**

3.1 Open the post man client

3.2 Admin account creation

Method : post

URL : <http://localhost:5000/api/auth/signup>

Header property : content-type – application/json

Data in body part as : for admin user

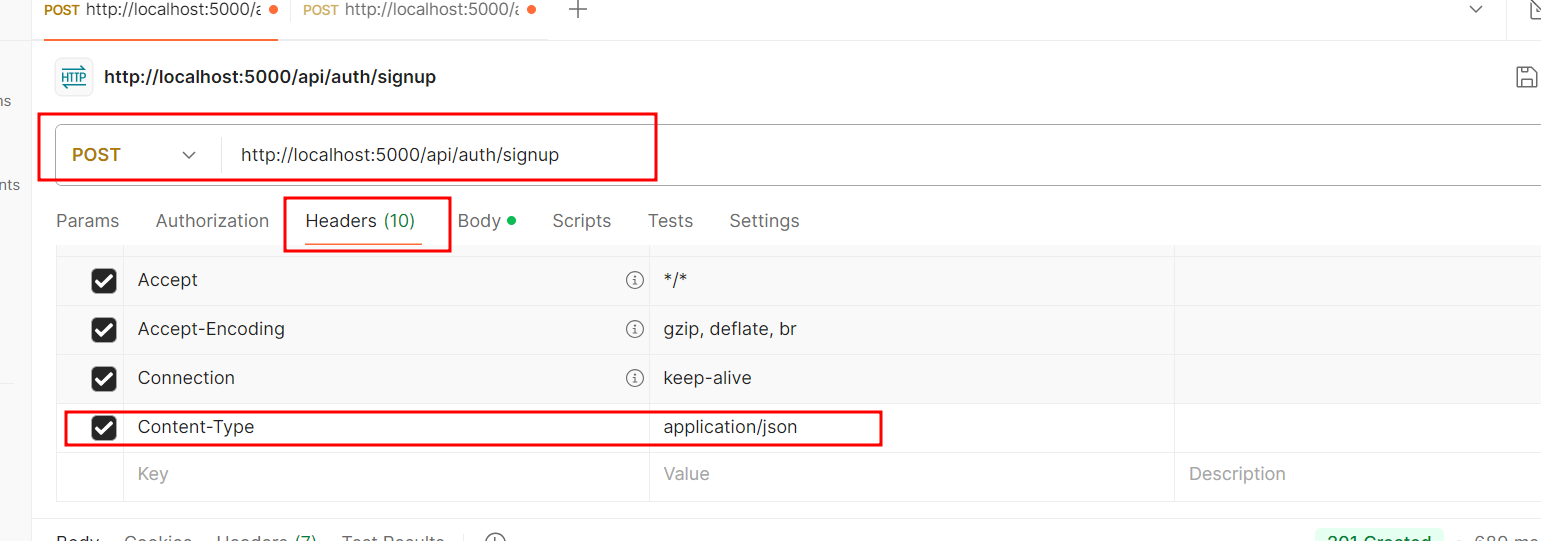
{

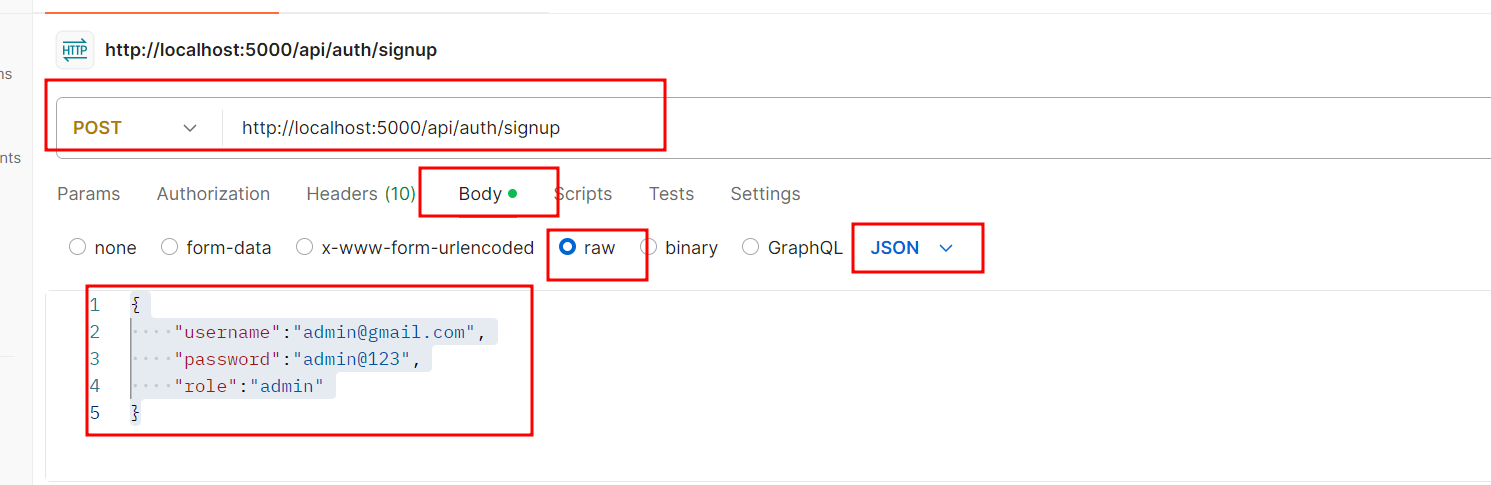
    "username":"admin@gmail.com",

    "password":"admin@123",

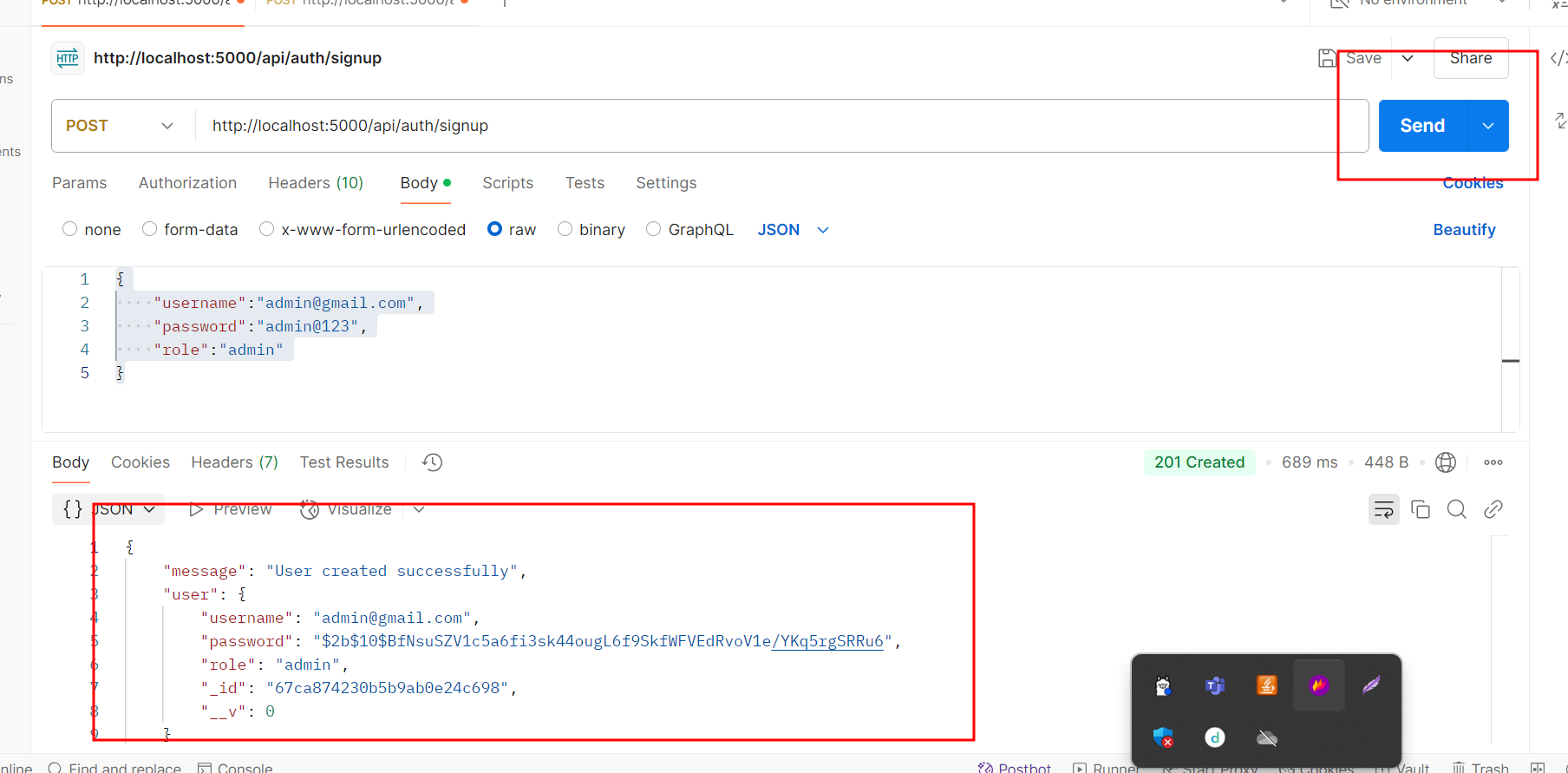
    "role":"admin"

}

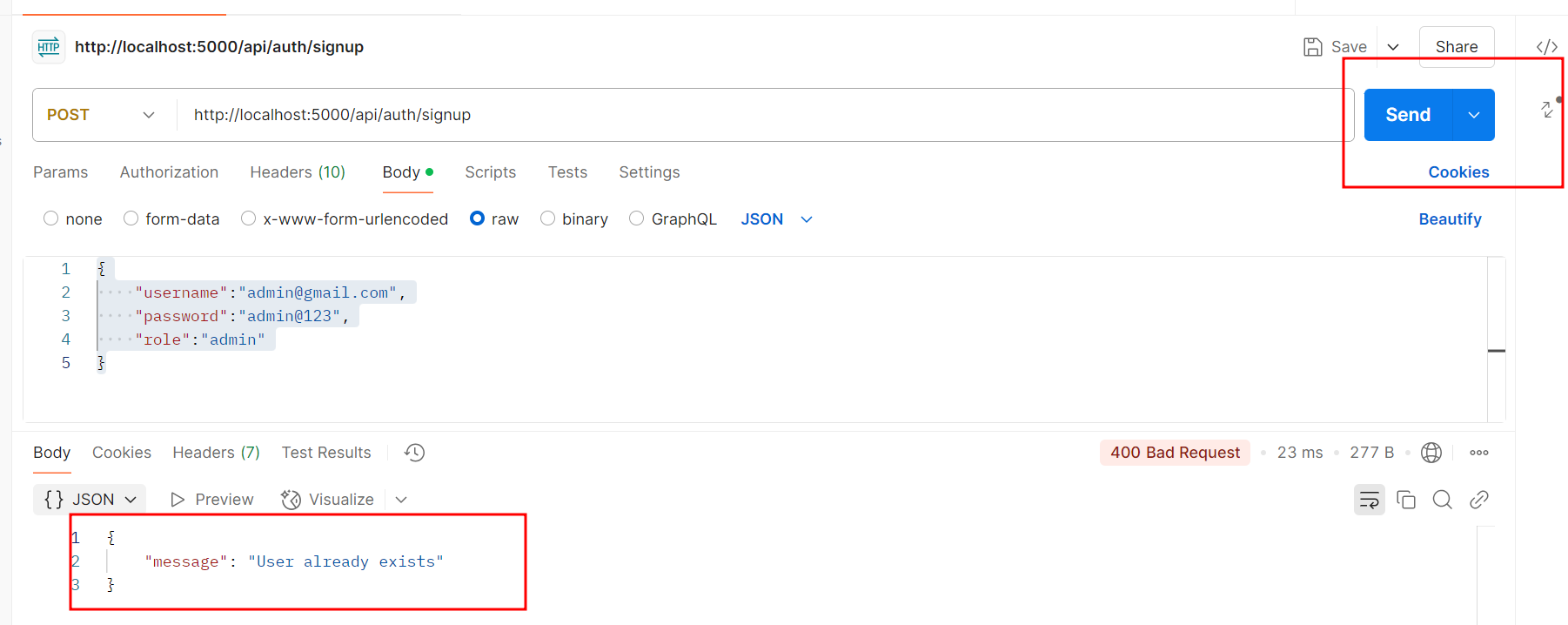




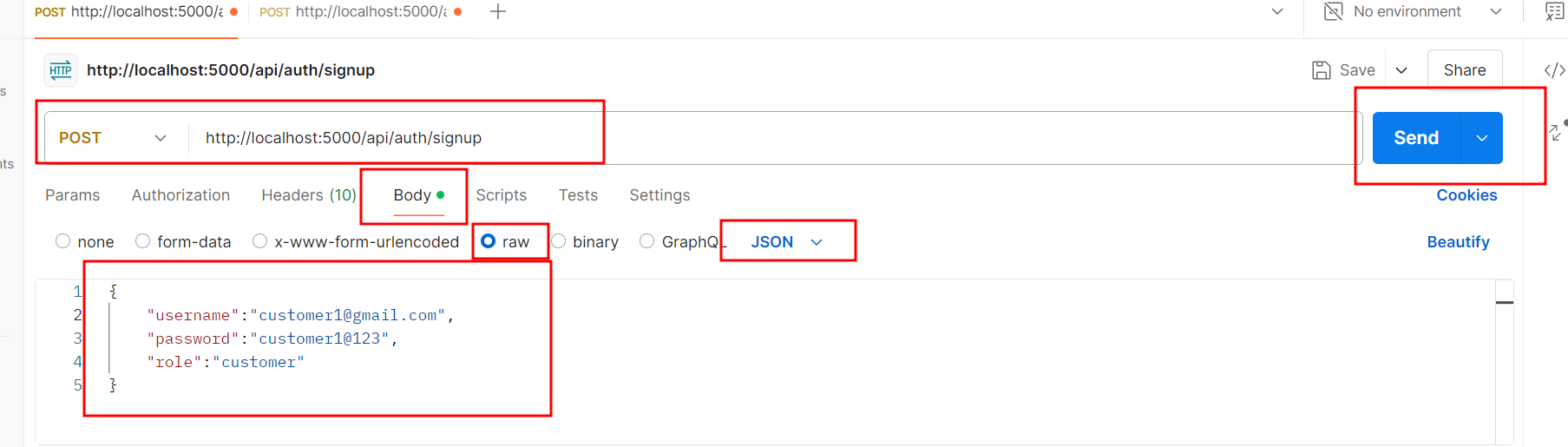
Now click on submit button



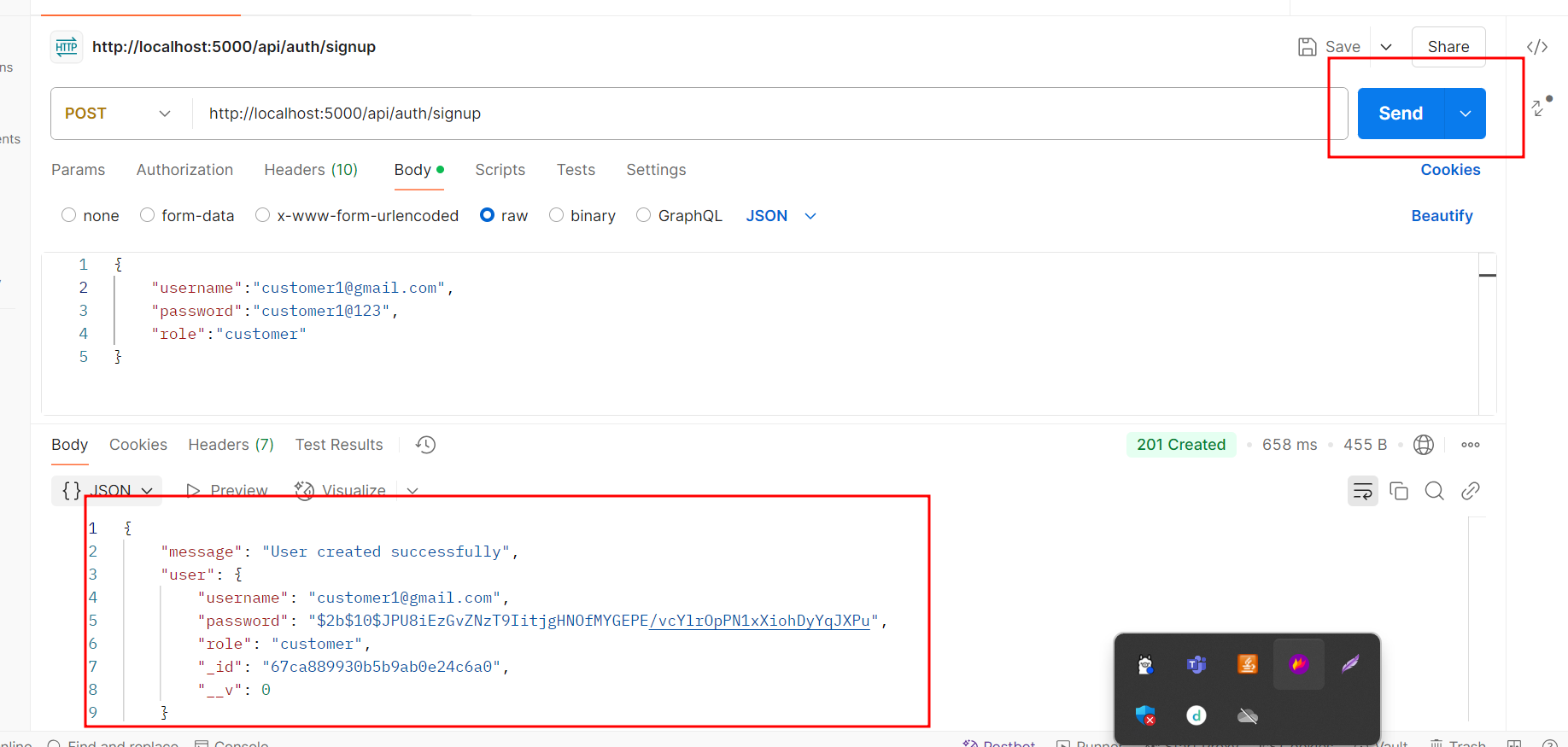
If you try to create once again same account



3.3 Customer account creation

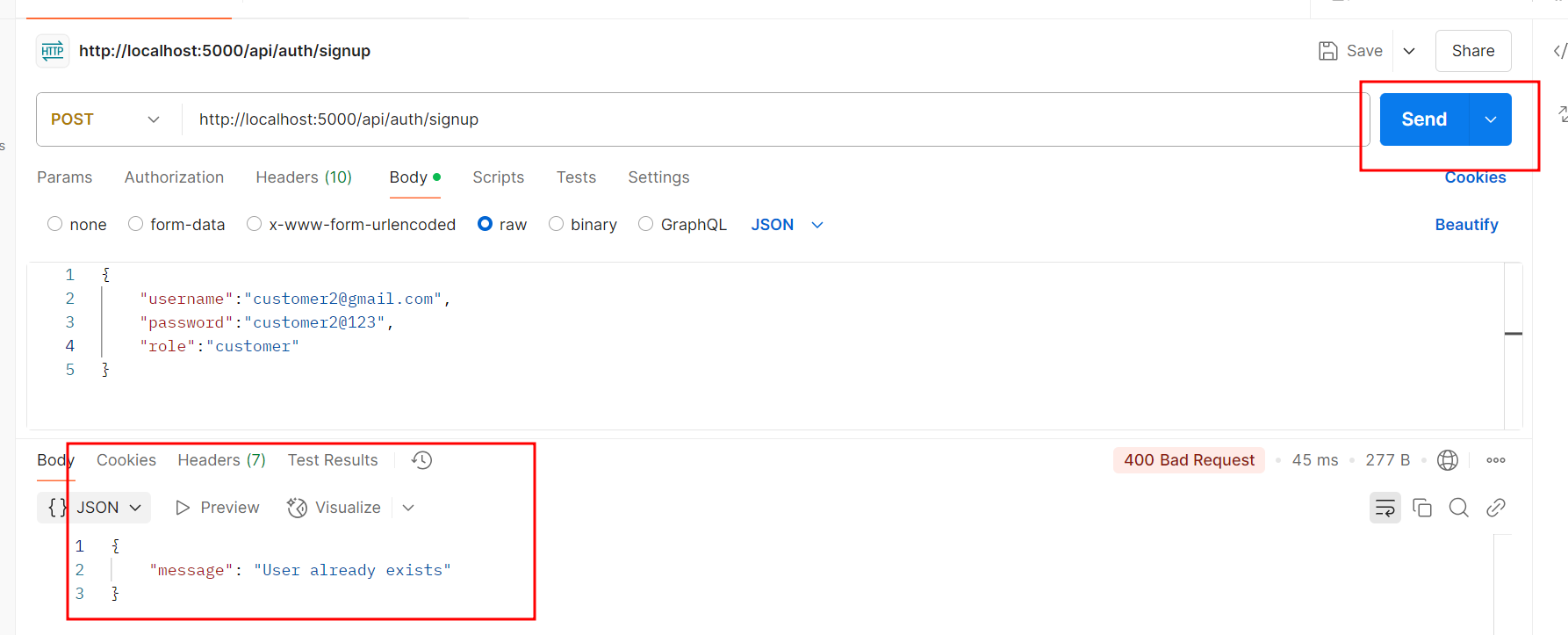


After click on send button



Create more customer account

If you create same customer account, we will get the error as

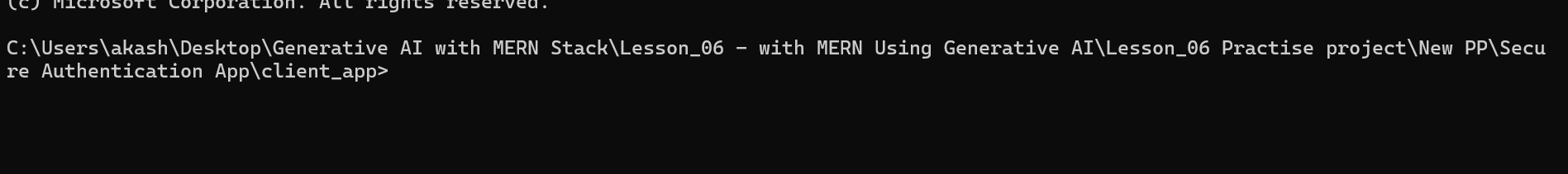


**Step 4: Create the frontend using react js technology with help of chat gpt.**

**“**react js frontend application for admin and customer signing and signup with jwt token with vite framework**”**

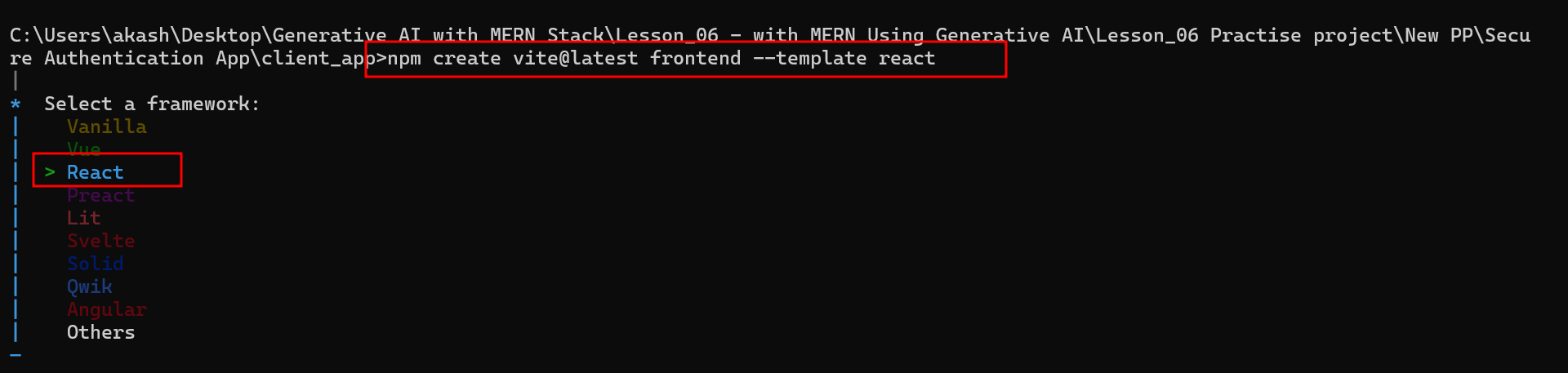
4.1 create the react js project using vite framework

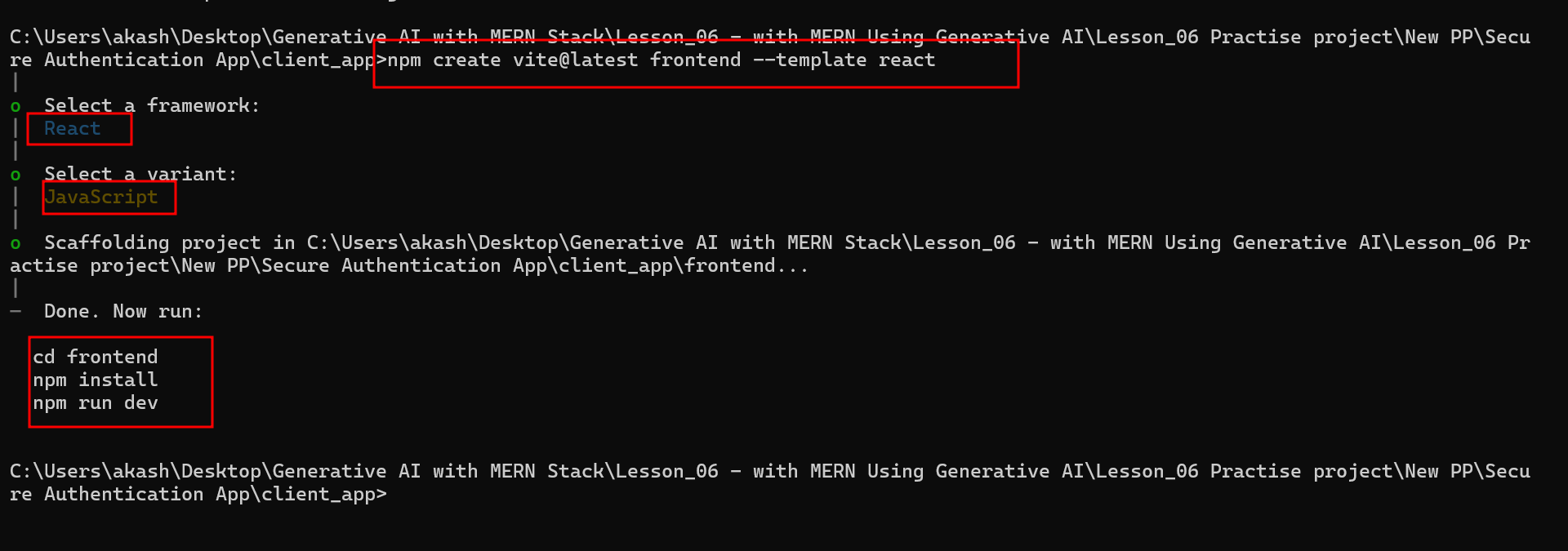
Open the command prompt or terminal inside client\_app folder



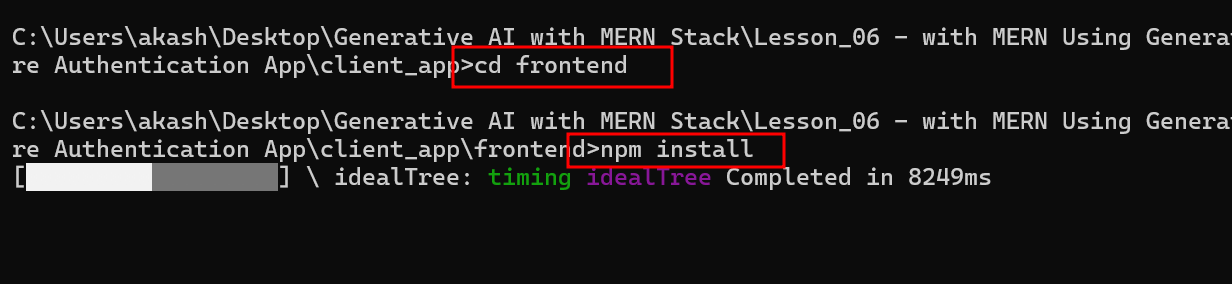
4.2 create the project

**npm create vite@latest frontend --template react**

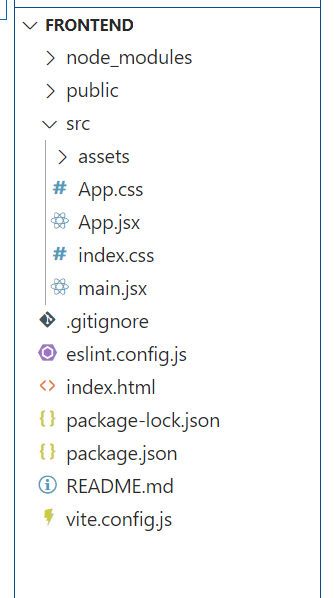




4.3 move inside a project and run the command as npm install to installed required dependencies to run the react js project



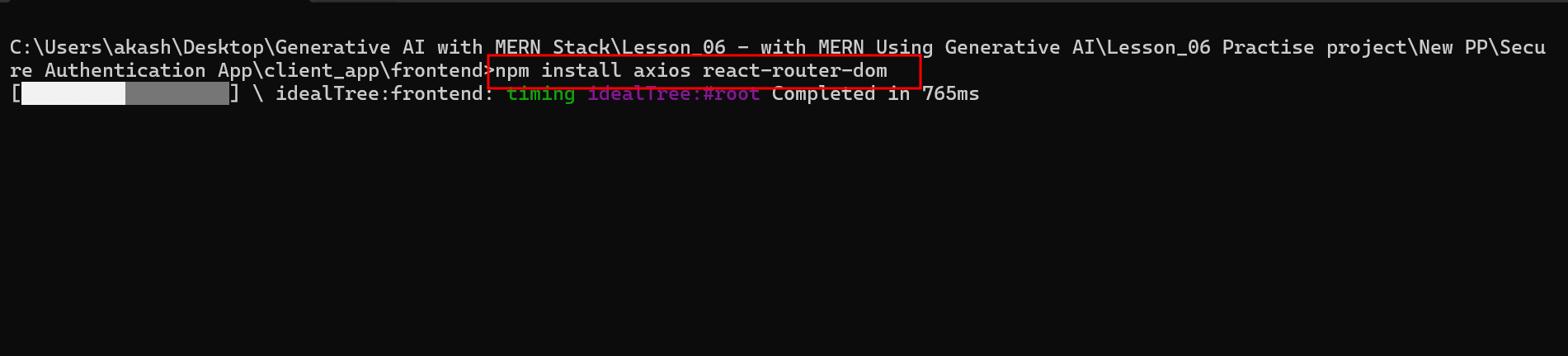
4.4 open the project in VS code



4.5 **Install Dependencies**

You will need axios for API requests and react-router-dom for routing.

**npm install axios react-router-dom**



4.6 **Set up the Directory Structure**

The directory structure for this frontend app will be:



4.7 **Set up .env for Environment Variables**

In the root of your frontend directory, create a .env file for storing the API base URL.

**.env** file

VITE\_API\_URL=http://localhost:5000/api/auth

4.8 Create the Axios Configuration (src/api.js)

Set up Axios to handle API requests to the backend.

// src/api.js

// src/api.js

import axios from "axios";

// Create an Axios instance for API calls

const api = axios.create({

  baseURL: import.meta.env.VITE\_API\_URL,  // Get the base URL from the .env file

  headers: {

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

  },

});

export default api;

4.9 **Create the SignIn Component (src/components/SignIn.jsx)**

This component will allow users to log in by providing their username and password.

// src/components/SignIn.jsx

import React, { useState } from "react";

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

import api from "../api";

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

const SignIn = () => {

  const [username, setUsername] = useState("");

  const [password, setPassword] = useState("");

  const [error, setError] = useState("");

  const navigate = useNavigate();

  const handleSignIn = async (e) => {

    e.preventDefault();

    try {

      const response = await api.post("/signin", { username, password });

      localStorage.setItem("token", response.data.token); // Save the JWT token

      localStorage.setItem("username", username); // Save the username

      navigate("/dashboard");

    } catch (err) {

      setError("Invalid username or password");

    }

  };

  return (

    <div className="auth-container">

      <h2>Sign In</h2>

      {error && <p className="error-message">{error}</p>}

      <form onSubmit={handleSignIn} className="auth-form">

        <div>

          <label>Username:</label>

          <input

            type="text"

            value={username}

            onChange={(e) => setUsername(e.target.value)}

            required

          />

        </div>

        <div>

          <label>Password:</label>

          <input

            type="password"

            value={password}

            onChange={(e) => setPassword(e.target.value)}

            required

          />

        </div>

        <button type="submit">Sign In</button>

      </form>

      <p>

        Don't have an account? <Link to="/signup">Sign Up</Link>

      </p>

    </div>

  );

};

export default SignIn;

4.10 **Create the SignUp Component (src/components/SignUp.jsx)**

This component will allow new users to create an account.

// src/components/SignUp.jsx

import React, { useState } from "react";

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

import api from "../api";

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

const SignUp = () => {

  const [username, setUsername] = useState("");

  const [password, setPassword] = useState("");

  const [role, setRole] = useState("customer");

  const [error, setError] = useState("");

  const navigate = useNavigate();

  const handleSignUp = async (e) => {

    e.preventDefault();

    try {

      await api.post("/signup", { username, password, role });

      navigate("/signin"); // Redirect to SignIn after successful signup

    } catch (err) {

      setError("Error creating user, please try again.");

    }

  };

  return (

    <div className="auth-container">

      <h2>Sign Up</h2>

      {error && <p className="error-message">{error}</p>}

      <form onSubmit={handleSignUp} className="auth-form">

        <div>

          <label>Username:</label>

          <input

            type="text"

            value={username}

            onChange={(e) => setUsername(e.target.value)}

            required

          />

        </div>

        <div>

          <label>Password:</label>

          <input

            type="password"

            value={password}

            onChange={(e) => setPassword(e.target.value)}

            required

          />

        </div>

        <div>

          <label>Role:</label>

          <select value={role} onChange={(e) => setRole(e.target.value)}>

            <option value="customer">Customer</option>

            <option value="admin">Admin</option>

          </select>

        </div>

        <button type="submit">Sign Up</button>

      </form>

      <p>

        Already have an account? <Link to="/signin">Sign In</Link>

      </p>

    </div>

  );

};

export default SignUp;

4.11 **Create the Dashboard Component (src/components/Dashboard.jsx)**

This component will be the user dashboard and can display data based on the user role.

// src/components/Dashboard.js

import React from "react";

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

const Dashboard = () => {

  const navigate = useNavigate();

  // Get the username and token from localStorage

  const username = localStorage.getItem("username");

  const handleLogout = () => {

    // Clear JWT token and username from localStorage

    localStorage.removeItem("token");

    localStorage.removeItem("username");

    // Redirect to SignIn page

    navigate("/signin");

  };

  return (

    <div className="dashboard-container">

      <h2>Welcome to the Dashboard</h2>

      <p>Welcome, {username}!</p>

      <button onClick={handleLogout} className="logout-button">

        Logout

      </button>

    </div>

  );

};

export default Dashboard;

4.12 **Set up Routing in App.jsx**

// src/App.jsx

import React from "react";

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

import SignIn from "./components/SignIn";

import SignUp from "./components/SignUp";

import Dashboard from "./components/Dashboard";

import "./App.css";

function App() {

  return (

    <Router>

      <div className="App">

        <Routes>

          <Route path="/signin" element={<SignIn />} />

          <Route path="/signup" element={<SignUp />} />

          <Route path="/dashboard" element={<Dashboard />} />

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

        </Routes>

      </div>

    </Router>

  );

}

export default App;

4.13 **Styling (Optional)**

You can add some basic styling for the forms and other elements in App.css.

/\* src/App.css \*/

/\* src/App.css \*/

\* {

  box-sizing: border-box;

  margin: 0;

  padding: 0;

}

body {

  font-family: Arial, sans-serif;

  background-color: #e9ecef;

  padding: 20px;

  display: flex;

  justify-content: center;

  align-items: center;

  height: 100vh;

}

.App {

  display: flex;

  justify-content: center;

  align-items: center;

  height: 100%;

  width: 100%;

}

.auth-container {

  display: flex;

  flex-direction: column;

  justify-content: center;

  align-items: center;

  max-width: 400px;

  width: 100%;

  padding: 30px;

  background-color: #f4f4f9;

  border-radius: 8px;

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

}

.auth-form {

  width: 100%;

  display: flex;

  flex-direction: column;

}

.auth-form input,

.auth-form select {

  padding: 12px;

  margin: 10px 0;

  border-radius: 4px;

  border: 1px solid #ccc;

  font-size: 16px;

}

.auth-form button {

  padding: 12px;

  background-color: #007bff;

  color: white;

  border: none;

  border-radius: 4px;

  cursor: pointer;

  font-size: 16px;

}

.auth-form button:hover {

  background-color: #0056b3;

}

.error-message {

  color: red;

  margin-bottom: 10px;

}

p {

  margin-top: 10px;

}

a {

  color: #007bff;

  text-decoration: none;

}

a:hover {

  text-decoration: underline;

}

h2 {

  margin-bottom: 20px;

  font-size: 24px;

  text-align: center;

  color: #333;

}

p {

  text-align: center;

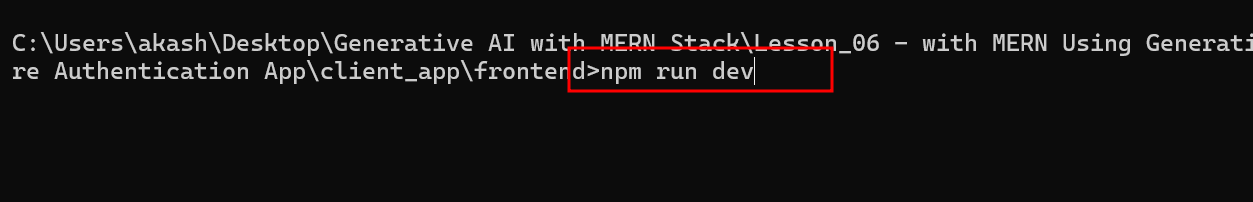
  font-size: 16px;

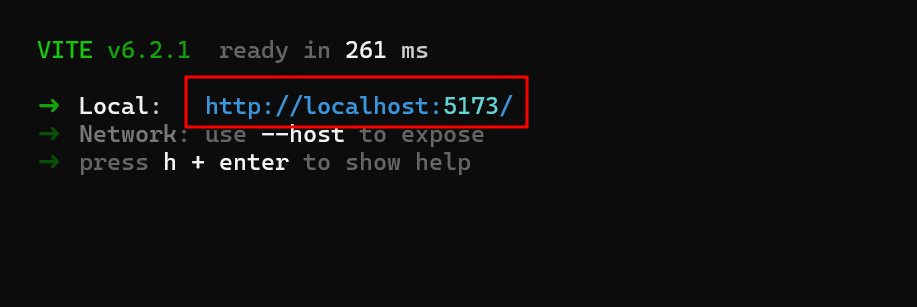
}

4.15 **Run the Application**

Now, start the Vite development server:

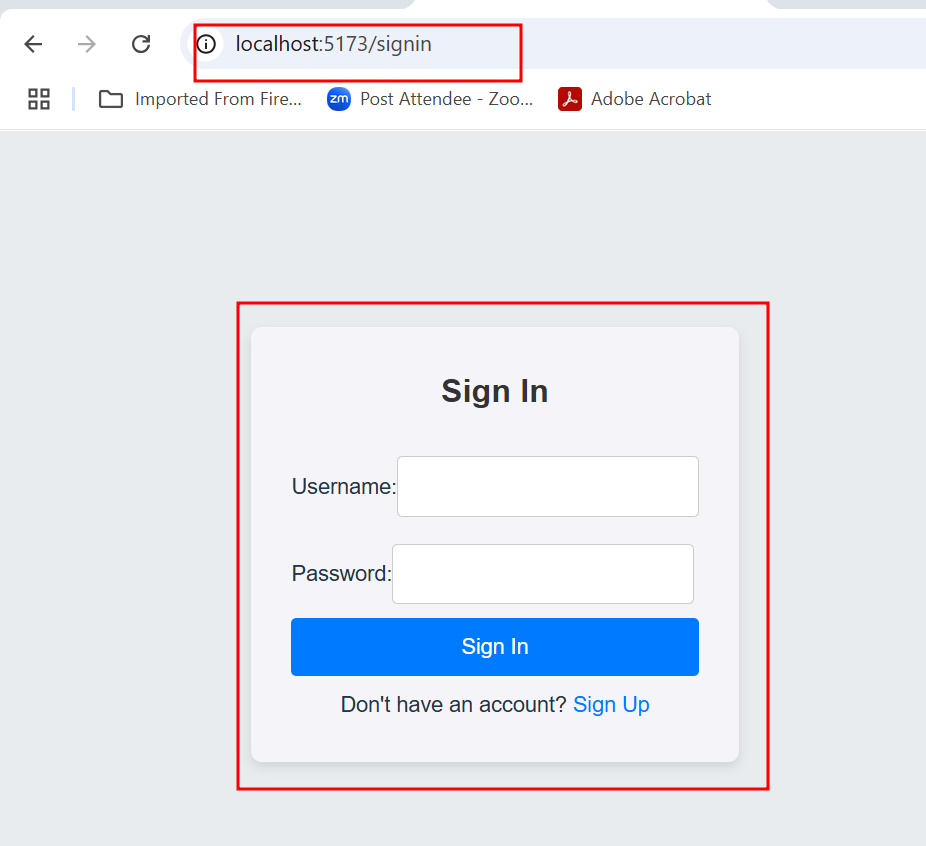
**npm run dev**



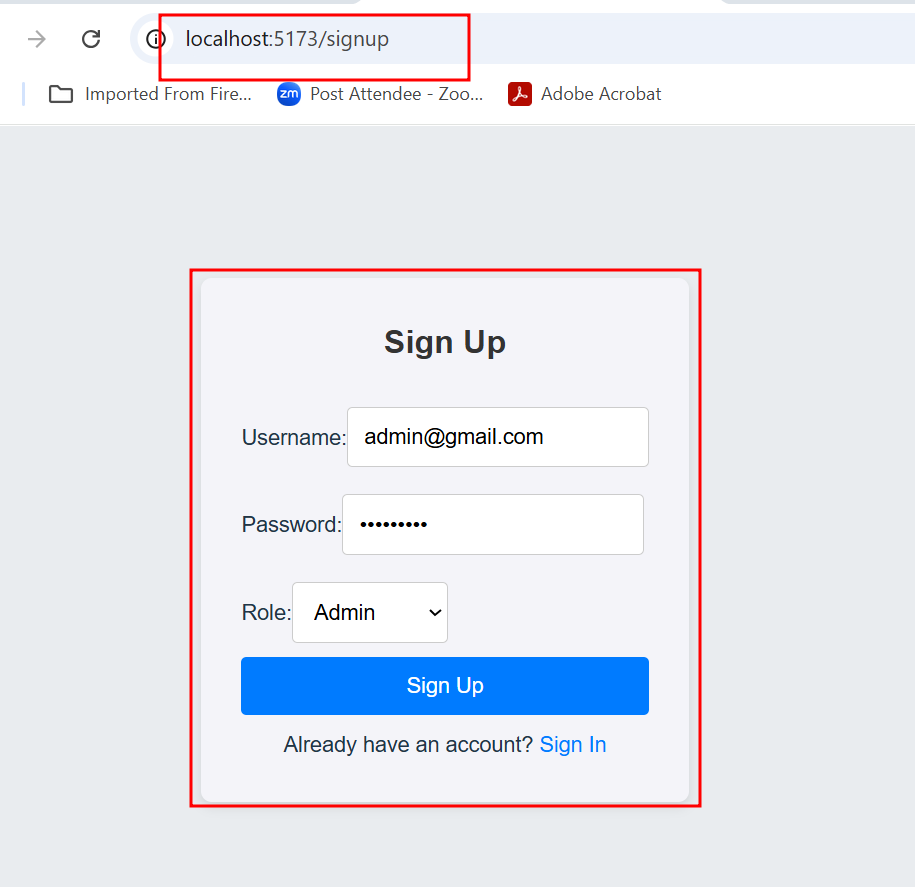


4.16 open this url on browser

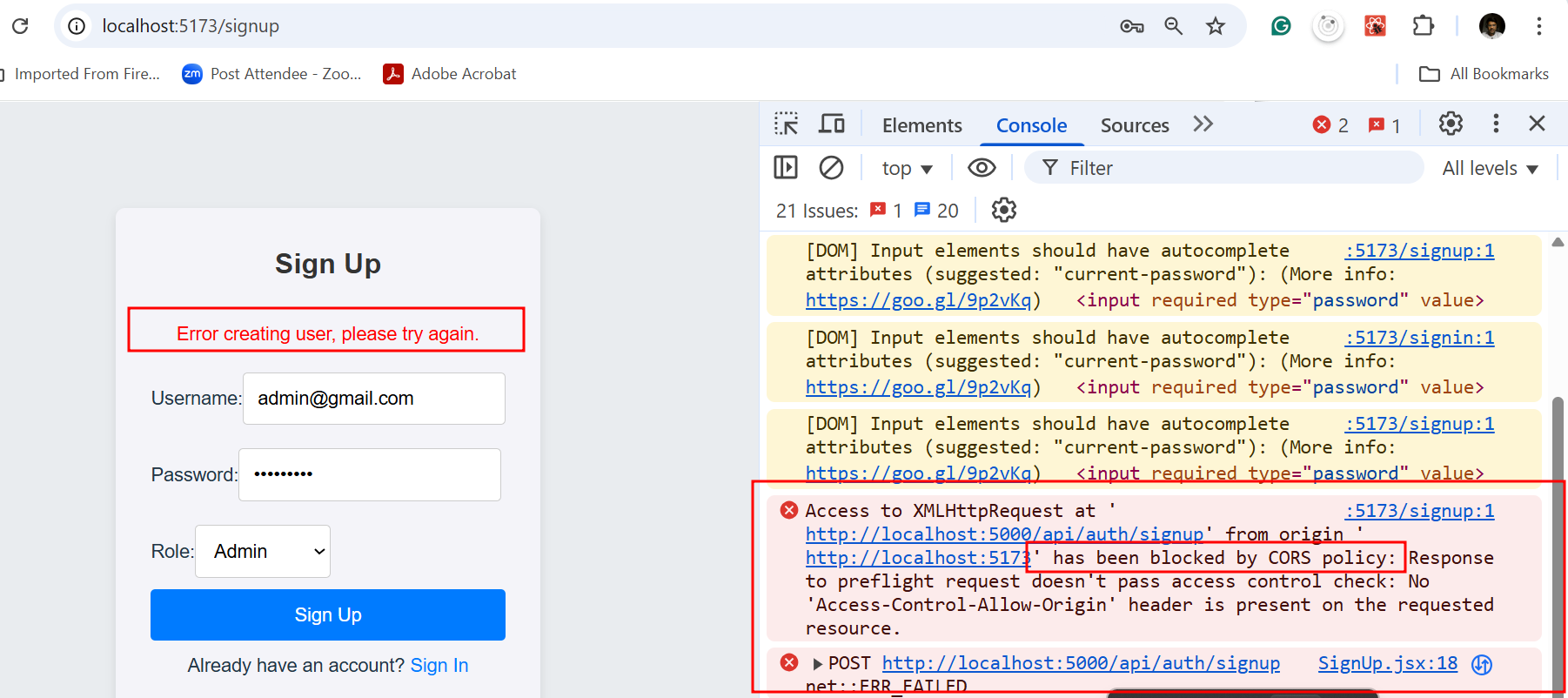
<http://localhost:5173>



4.17 try to create the admin account.

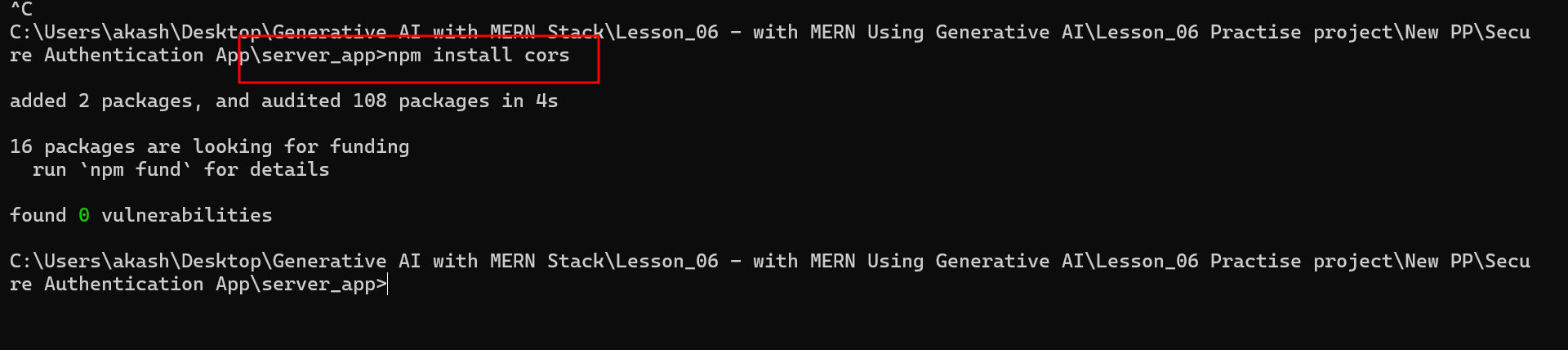


4.18 try to create the admin account

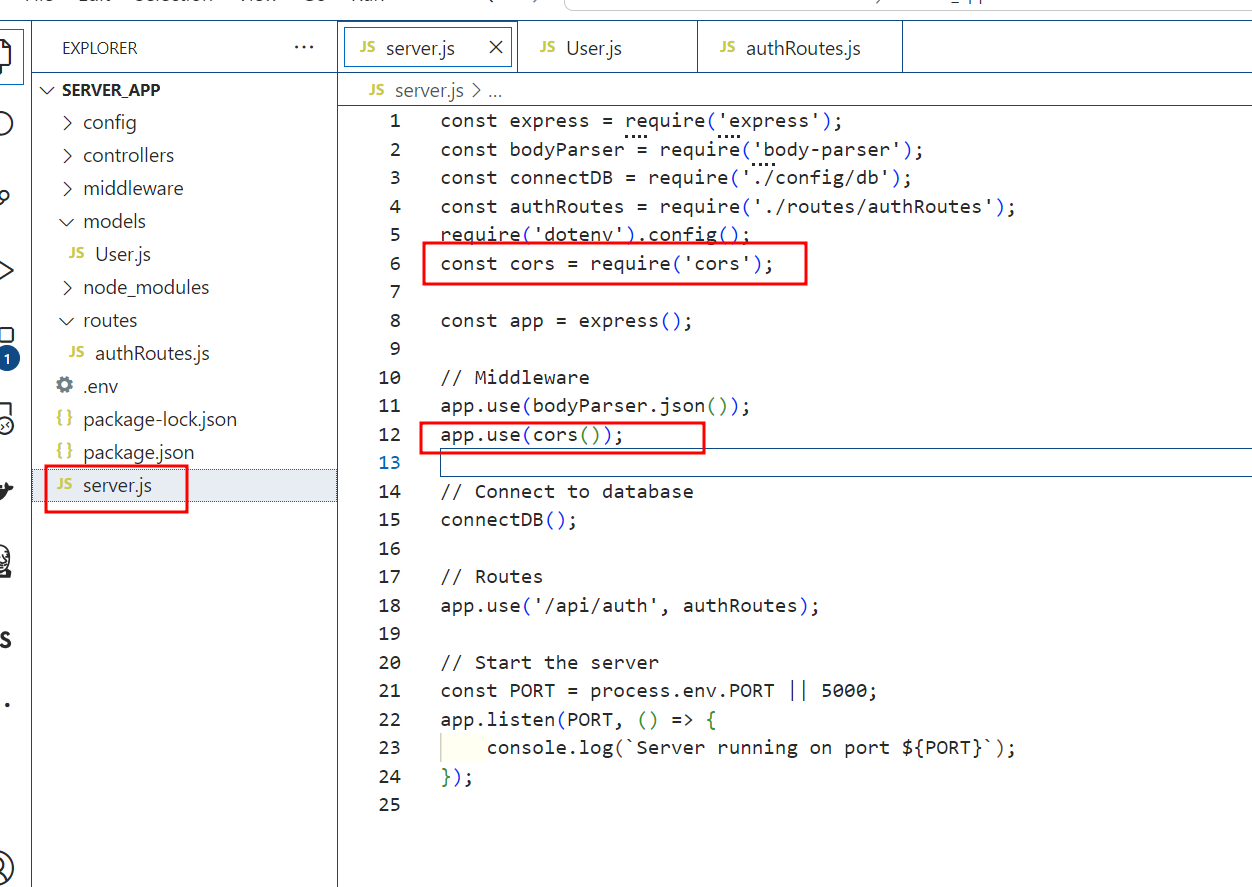


If you get error as cors (cross origin resource sharing) issue. Then in backend technology you need to install cors node js module and enable middleware

4.19 In backend technology terminology



4.20 open the server.js file and add cors middle module



Updated server.js file

const express = require('express');

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

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

const authRoutes = require('./routes/authRoutes');

require('dotenv').config();

const cors = require('cors');

const app = express();

// Middleware

app.use(bodyParser.json());

app.use(cors());

// Connect to database

connectDB();

// Routes

app.use('/api/auth', authRoutes);

// Start the server

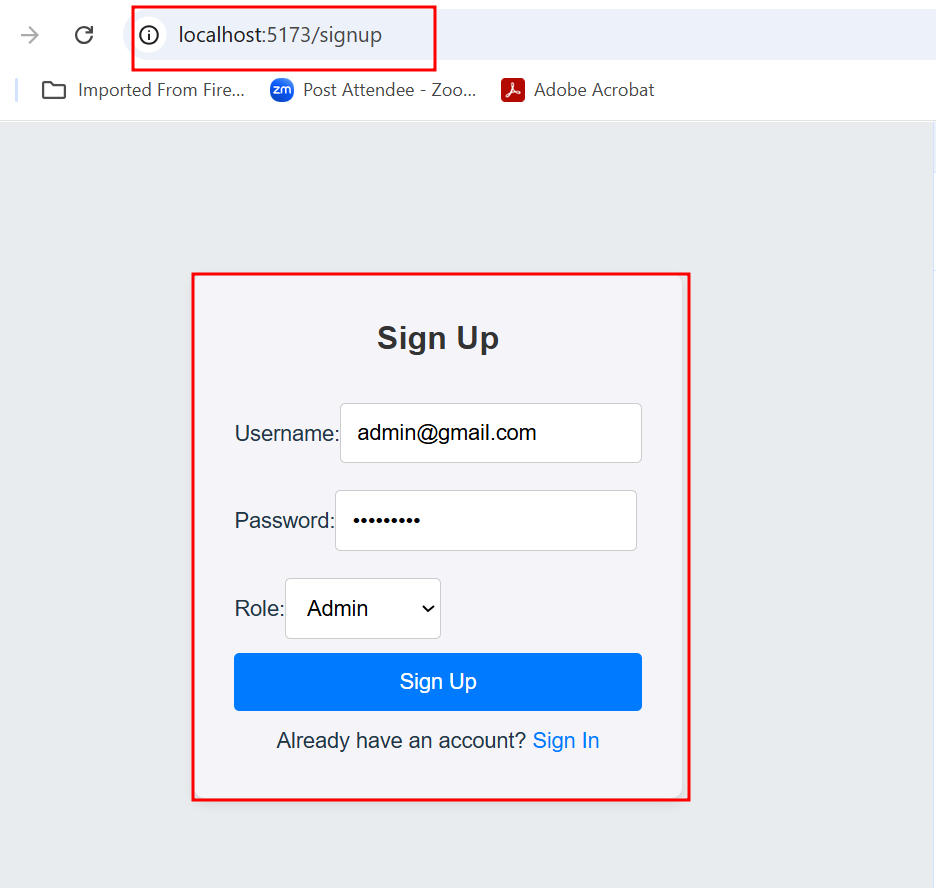
const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

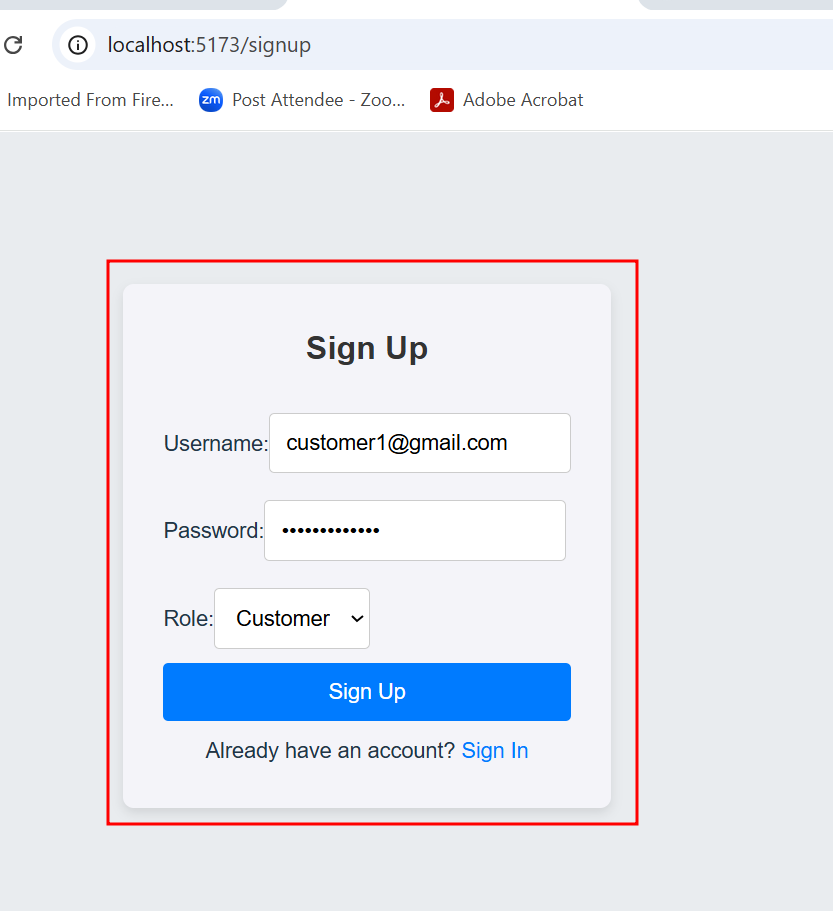
    console.log(`Server running on port ${PORT}`);

});

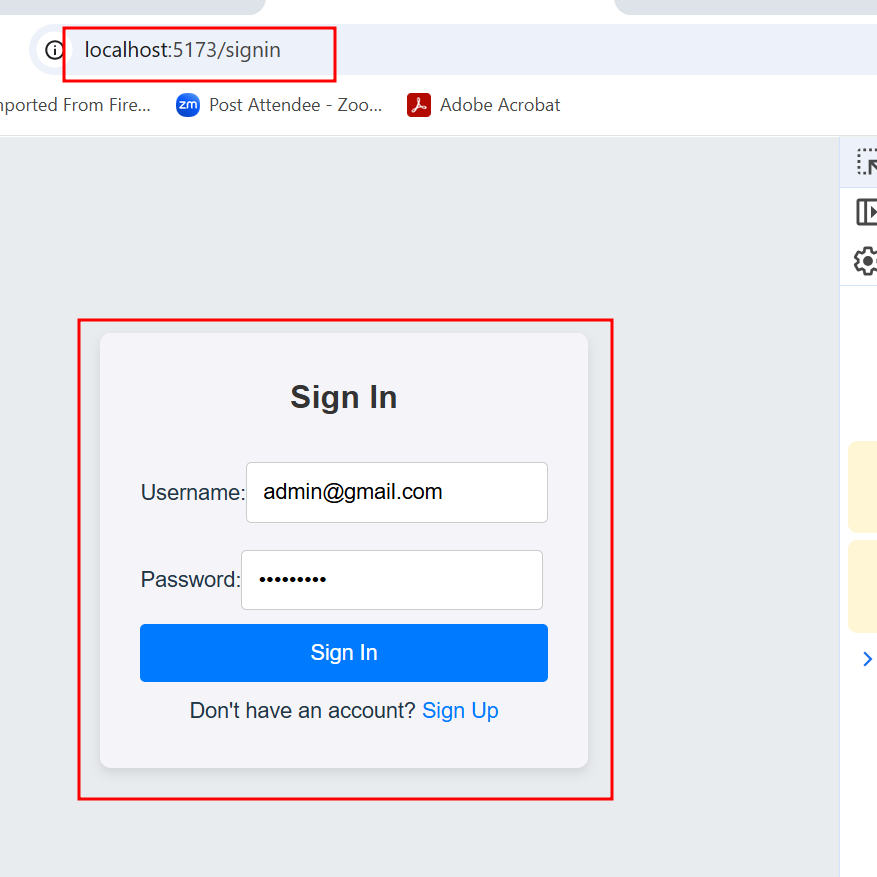
4.21 re run the application



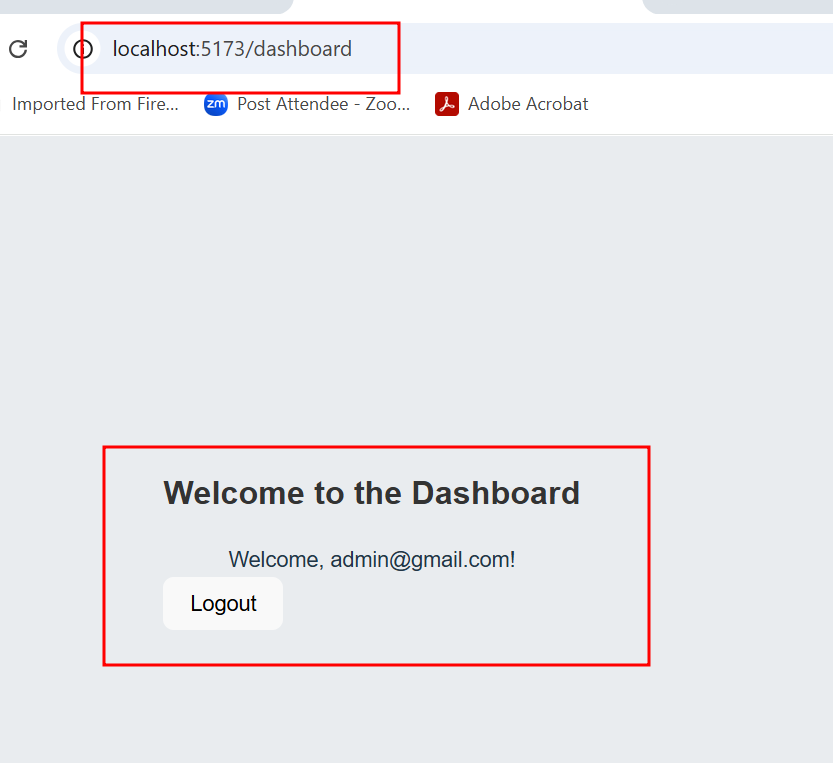
4.22 Now you can create more than one account for user as well as admin

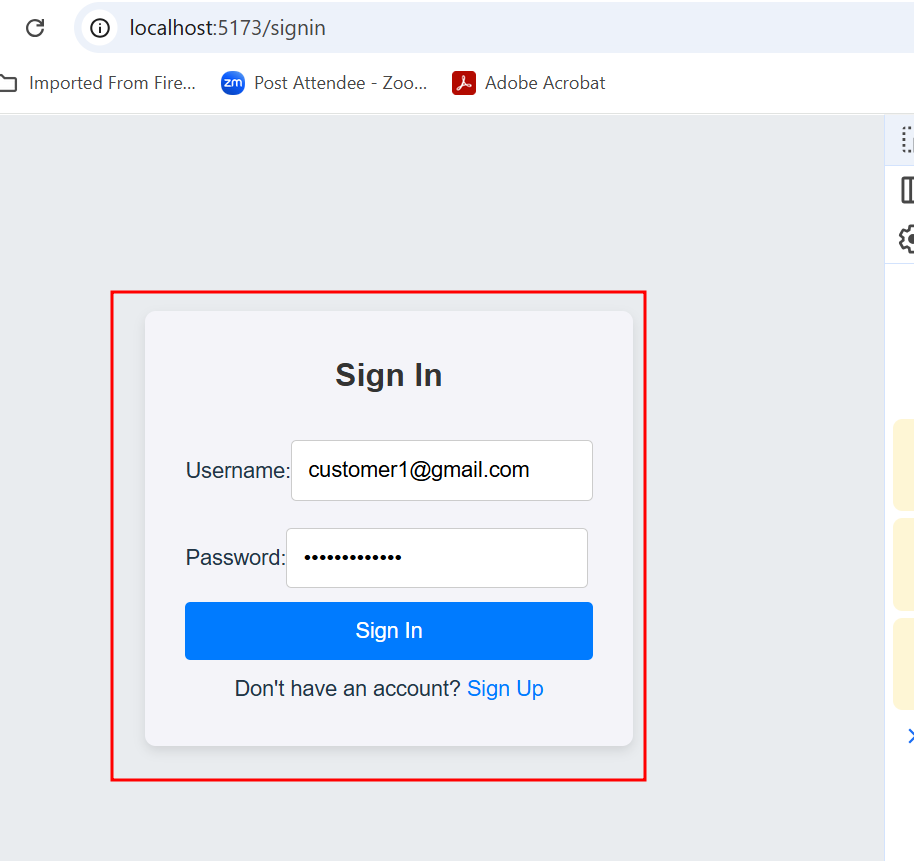


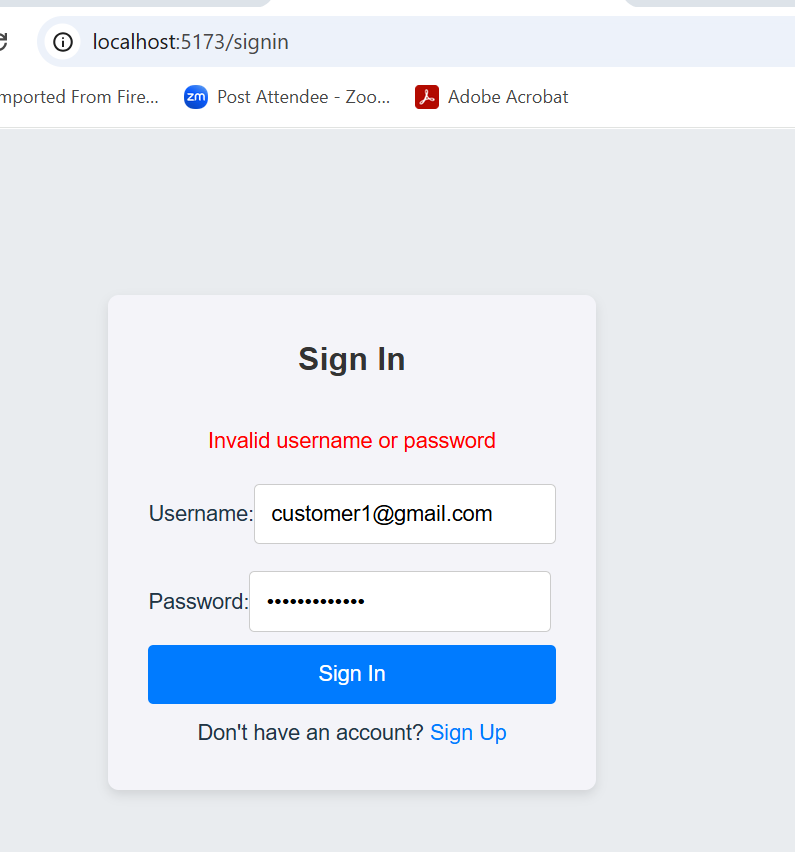
4.23 Now you can do the SignIn with admin as well as customer account



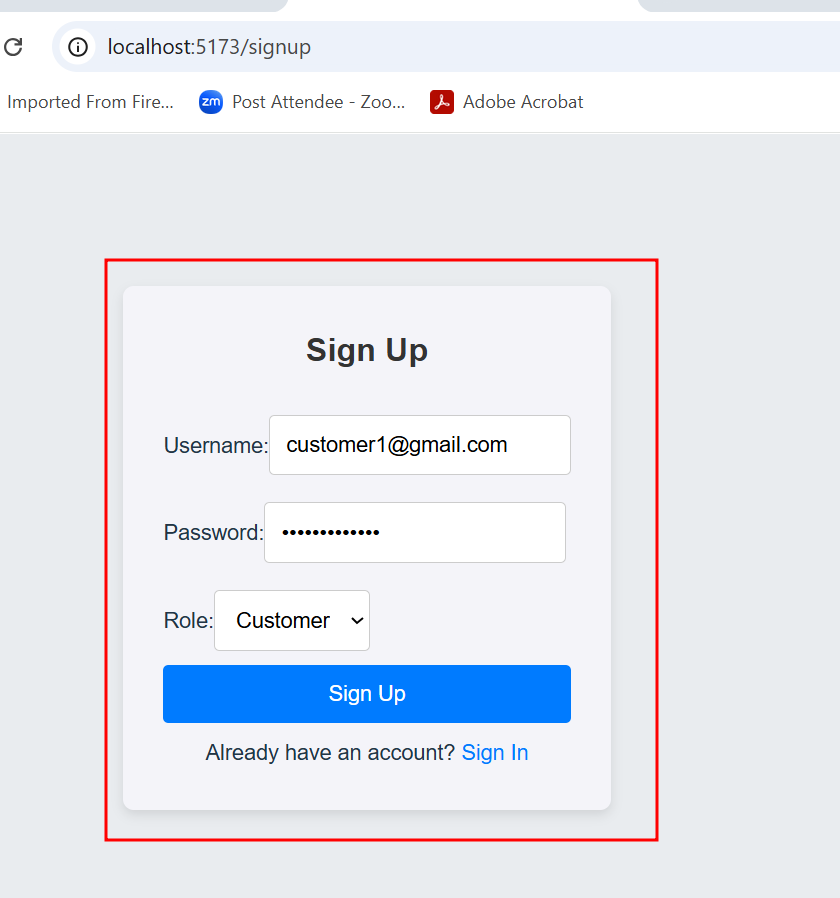
4.24 after account created successfully



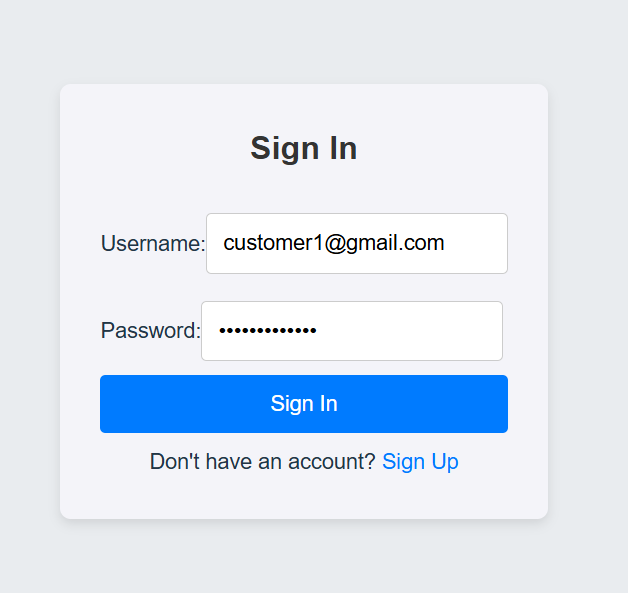


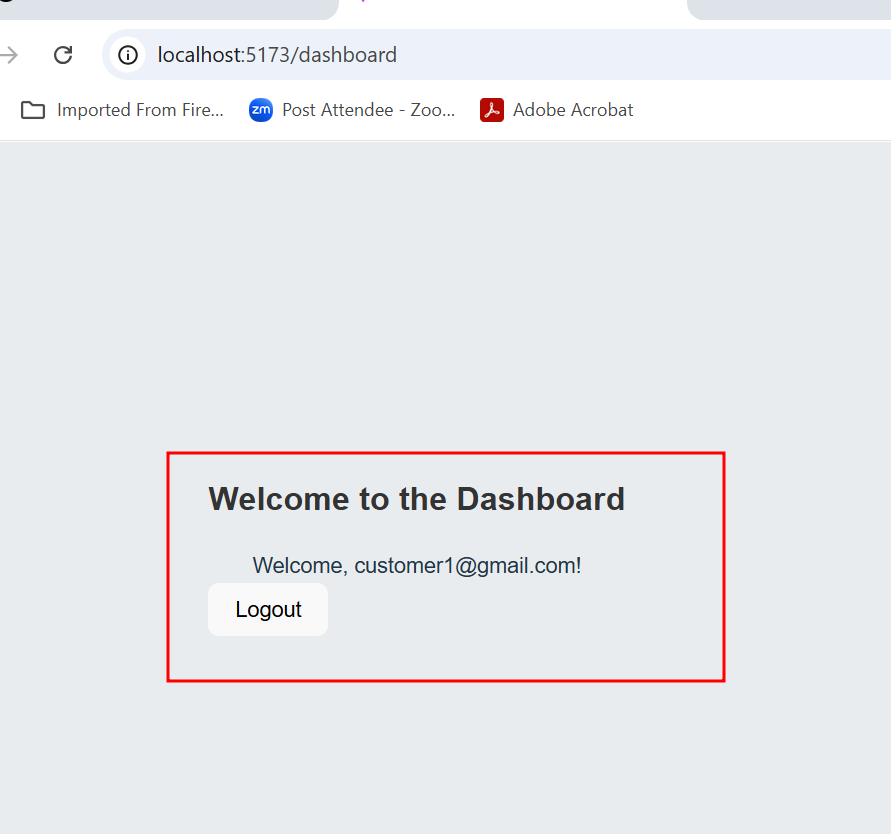


4.25 create the customer account



4.26 Do SignIn for customer login





4.27 verify tables created in database

