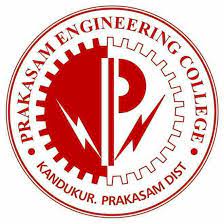
****

**PRAKASAM engineering college**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

KAKINADA – 533 003, Andhra Pradesh, India

B.Tech CSE (R23-COURSE STRUCTURE & SYLLABUS)

III Year I Semester

 FULL STACK DEVELOPMENT - 2

L T P C

0 1 2 2

**Course Objectives:**

The main objectives of the course are to

• Make use of router, template engine and authentication using sessions to develop

application in Express JS.

• Build a single page application using RESTful APIs in Express JS

• Apply router and hooks in designing React JS application

• Make use of MongoDB queries to perform CRUD operations on document database

**Experiments covering the Topics:**

• Express JS – Routing, HTTP Methods, Middleware, Templating, Form Data

• Express JS – Cookies, Sessions, Authentication, Database, RESTful APIs

• React JS – Render HTML, JSX, Components – function & Class, Props and States, Styles, Respond to Events

• React JS – Conditional Rendering, Rendering Lists, React Forms, React Router, Updating the Screen

• React JS – Hooks, Sharing data between Components, Applications – To-do list and Quiz

• MongoDB – Installation, Configuration, CRUD operations, Databases, Collections and Records

**Sample Experiment**s:

1. Express JS – Routing, HTTP Methods, Middleware.

a. Write a program to define a route, Handling Routes, Route Parameters, Query

Parameters and URL building.

b. Write a program to accept data, retrieve data and delete a specified resource using http

methods.

c. Write a program to show the working of middleware.

2. Express JS – Templating, Form Data

a. Write a program using a templating engine.

b. Write a program to work with form data.

3. Express JS – Cookies, Sessions, Authentication

a. Write a program for session management using cookies and sessions.

b. Write a program for user authentication.

4. Express JS – Database, RESTful APIs

a. Write a program to connect MongoDB database using Mangoose and perform CRUD

operations.

b. Write a program to develop a single page application using RESTful APIs.

5. ReactJS – Render HTML, JSX, Components – function & Class

a. Write a program to render HTML to a web page.

b. Write a program for writing markup with JSX.

c. Write a program for creating and nesting components (function and class).

6. ReactJS – Props and States, Styles, Respond to Events

a. Write a program to work with props and states.

b. Write a program to add styles (CSS & Sass Styling) and display data.

c. Write a program for responding to events.

7. ReactJS – Conditional Rendering, Rendering Lists, React Forms

a. Write a program for conditional rendering.

b. Write a program for rendering lists.

c. Write a program for working with different form fields using react forms.

8. ReactJS – React Router, Updating the Screen

a. Write a program for routing to different pages using react router.

b. Write a program for updating the screen.

9. ReactJS – Hooks, Sharing data between Components

a.Write a program to understand the importance of using hooks.

b. Write a program for sharing data between components.

10. MongoDB – Installation, Configuration, CRUD operations

a. Install MongoDB and configure ATLAS

b. Write MongoDB queries to perform CRUD operations on document using insert(),

find(), update(), remove()

11. MongoDB – Databases, Collections and Records

a. Write MongoDB queries to Create and drop databases and collections.

b. Write MongoDB queries to work with records using find(), limit(), sort(),

createIndex(), aggregate().

12. Augmented Programs: (Any 2 must be completed)

a. Design a to-do list application using NodeJS and ExpressJS.

b. Design a Quiz app using ReactJS.

c. Complete the MongoDB certification from MongoDB University website.

Text Books:

1. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and

Node, Vasan Subramanian, 2nd edition, APress, O’Reilly.

2. Node.Js in Action, Mike Cantelon, Mark Harter, T.J. Holowaychuk, Nathan

Rajlich,Manning Publications. (Chapters 1-11)

3. React Quickly, AzatMardan,Manning Publications(Chapters 1-8,12-14)

Web Links:

1. ExpressJS - https://www.tutorialspoint.com/expressjs

2. ReactJS - https://www.w3schools.com/REACT (and) https://react.dev/learn#

3. MongoDB - https://learn.mongodb.com/learning-paths/introduction-to-mongodb

1. Express JS – Routing, HTTP Methods, Middleware.

a. Write a program to define a route, Handling Routes, Route Parameters, Query Parameters and URL building.

b. Write a program to accept data, retrieve data and delete a specified resource using http methods.

c. Write a program to show the working of middleware.

a. Write a program to define a route, Handling Routes, Route Parameters, Query Parameters and URL building.

const express = require('express');

const app = express();

const port = 3000;

// 1. Basic route

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

res.send('Welcome to the Express.js routing example!');

});

// 2. Route with route parameters

// Example URL: /user/123/profile

app.get('/user/:userId/profile', (req, res) => {

const userId = req.params.userId;

res.send(`User Profile page for user with ID: ${userId}`);

});

// 3. Route with query parameters

// Example URL: /search?term=nodejs&sort=asc

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

const term = req.query.term || 'none';

const sort = req.query.sort || 'default';

res.send(`Search results for term: ${term}, sorted by: ${sort}`);

});

// 4. URL building example (sending a link back to user)

// This route will dynamically create a URL based on parameters and send it back

app.get('/build-url/:category/:id', (req, res) => {

const category = req.params.category;

const id = req.params.id;

// Build URL with query params

const url = `/products/${category}/${id}?ref=homepage&utm=summer\_sale`;

res.send(`Built URL: <a href="${url}">${url}</a>`);

});

// Example route to test built URL

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

const { category, id } = req.params;

const { ref, utm } = req.query;

res.send(`Product Page - Category: ${category}, ID: ${id}, Referrer: ${ref}, Campaign: ${utm}`);

});

app.listen(port, () => {

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

});

* Defining routes
* Handling route parameters
* Handling query parameters
* Building URLs dynamically

### **How to run this code:**

1. Save it in a file, e.g., app.js

Make sure you have Express installed:  
  
  
  
npm install express

Run the server:  
  
  
node app.js

1. Open your browser and test:  
   * http://localhost:3000/
   * http://localhost:3000/user/123/profile
   * http://localhost:3000/search?term=nodejs&sort=asc
   * http://localhost:3000/build-url/electronics/456
   * Click the link generated by /build-url to see the product page with query parameters

**Output:**

Server is Successfully Running and App is listening on port 3000

b. Write a program to accept data, retrieve data and delete a specified resource using http methods.

const express = require('express');

const app = express();

const port = 3000;

// Middleware to parse JSON bodies

app.use(express.json());

let items = [];

let currentId = 1;

// POST /items - Create new item

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

const { name, description } = req.body;

if (!name) {

return res.status(400).json({ error: 'Name is required' });

}

const newItem = {

id: currentId++,

name,

description: description || '',

};

items.push(newItem);

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

});

// GET /items - Get all items

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

res.json(items);

});

// GET /items/:id - Get item by ID

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

const id = parseInt(req.params.id);

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

if (!item) {

return res.status(404).json({ error: 'Item not found' });

}

res.json(item);

});

// DELETE /items/:id - Delete item by ID

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

const id = parseInt(req.params.id);

const index = items.findIndex((i) => i.id === id);

if (index === -1) {

return res.status(404).json({ error: 'Item not found' });

}

const deleted = items.splice(index, 1);

res.json({ message: 'Item deleted successfully', item: deleted[0] });

});

app.listen(port, () => {

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

});

**How to run**

Save the above as app.js

**Run:**

npm install express

**node app.js**

Use these exact Windows CMD curl commands to test:

Test commands for Windows CMD

Create an item (POST):

curl -X POST http://localhost:3000/items -H "Content-Type: application/json" -d "{\"name\":\"Item 1\", \"description\":\"Test item\"}"

Get all items (GET):

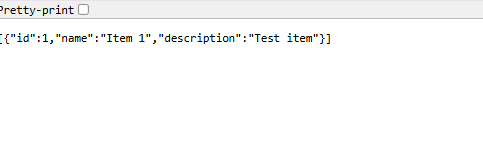
curl http://localhost:3000/items

Get an item by ID (GET):

curl http://localhost:3000/items/1

Delete an item by ID (DELETE):

curl -X DELETE <http://localhost:3000/items/1>



c. Write a program to show the working of middleware.

const express = require('express');

const app = express();

const port = 3000;

// Middleware 1: Logger - logs method and URL for every request

app.use((req, res, next) => {

console.log(`[Logger] ${req.method} ${req.url}`);

next(); // Pass control to the next middleware

});

// Middleware 2: Add a timestamp to the request object

app.use((req, res, next) => {

req.requestTime = new Date().toISOString();

next();

});

// Route handler - sends response including the timestamp added by middleware

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

res.send(`Hello! Request received at: ${req.requestTime}`);

});

// Route with middleware specific to this route only

function checkAuth(req, res, next) {

const authorized = req.query.auth === 'secret';

if (authorized) {

console.log('User authorized');

next();

} else {

res.status(401).send('Unauthorized: Missing or wrong auth query parameter');

}

}

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

res.send('Welcome to the protected route!');

});

app.listen(port, () => {

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

});

How this works:

Logger middleware runs for every request and logs method + URL.

Timestamp middleware adds a requestTime property to the req object.

The root route / returns the time the request was received.

The /protected route uses a route-specific middleware (checkAuth) that checks for an auth=secret query parameter, and only allows access if authorized.

Test it out:

Open in browser or curl:

http://localhost:3000/

See logged method and URL in the console and the timestamp in the response.

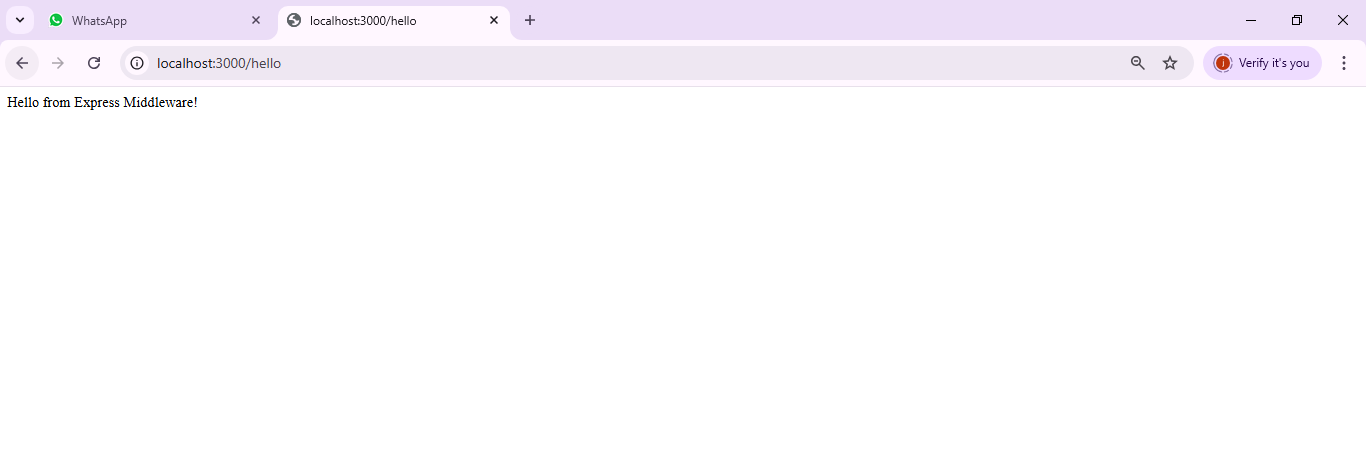
Try accessing protected route without auth:

http://localhost:3000/protected

— you’ll get Unauthorized.

Now access protected route with correct auth:

http://localhost:3000/protected?auth=secret



2A program using a template engine in Express.js can be created by following these steps, using EJS as an example template engine:

Step 1. Project Setup:

Create a new directory for the project and navigate into it.

mkdir express-template-app

cd express-template-app

npm init -y

Initialize npm:

Code

npm init -y

Install Express and EJS.

Code

npm install express ejs

Step 2. Express Application Setup (app.js):

Create a file named app.js.

Require Express and create an instance of the application.

Configure Express to use EJS as the view engine and specify the directory where template files will be stored.

code:

const express = require('express');

const app = express();

const path = require('path'); // Required for path.join

// Set EJS as the view engine

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

// Set the directory for views (templates)

app.set('views', path.join(\_\_dirname, 'views'));

// Define a route to render a template

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

// Data to pass to the template

const data = {

title: 'Express EJS Example',

message: 'Welcome to our template-powered application!'

};

// Render the 'index.ejs' template with the provided data

res.render('index', data);

});

// Start the server

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

app.listen(PORT, () => {

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

});

Step 3. Create the Template File (views/index.ejs):

* Create a directory named views in the project root.
* Inside views, create a file named index.ejs.
* Add HTML structure and EJS syntax to display dynamic content.

code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title><%= title %></title>

</head>

<body>

<h1><%= message %></h1>

<p>This content is rendered using EJS.</p>

</body>

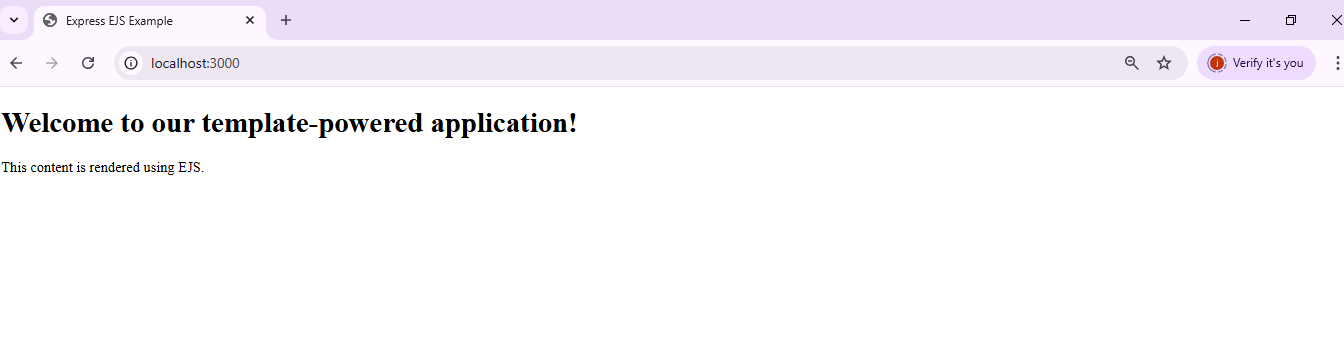
</html>

Step 4. Run the Application:

Execute the app.js file using Node.js:

node app.js

Open a web browser and navigate to http://localhost:3000. The rendered content from index.ejs will be displayed, with the dynamic title and message values injected from the Express route.



2 b.Working with form data

Working with form data in Express.js typically involves setting up an Express server to serve an HTML form and then processing the submitted data on the server-side.

### **How to run this code**

1. Save it as app.js.
2. Make sure you have Node.js installed.
3. Run npm init -y to create a package.json.
4. Run npm install express to install Express.
5. Run the app with node app.js.
6. Open http://localhost:3000 in your browser, fill the form, and submit.

1. Project Setup:

Create a new directory for the project and initialize a Node.js project:

Code

mkdir express-form-example

cd express-form-example

npm init -y

npm install express

2. HTML Form (public/index.html):

Create an index.html file inside a public directory to serve the form:

### **Project Structure**

express-form-example/

├── app.js

└── public/

└── index.html

### **1. app.js**

const express = require('express');

const app = express();

const port = 3000;

// Middleware to parse urlencoded form data

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

// Serve static files from the 'public' directory

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

// Handle form submission at /submit-form

app.post('/submit-form', (req, res) => {

const { name, email } = req.body;

res.send(`Received your submission! Name: ${name}, Email: ${email}`);

});

app.listen(port, () => {

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

});

### **2. public/index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Express Form Example</title>

</head>

<body>

<h1>Submit Your Information</h1>

<form action="/submit-form" method="POST">

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

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

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

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

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

</form>

</body>

</html>

### **How to run**

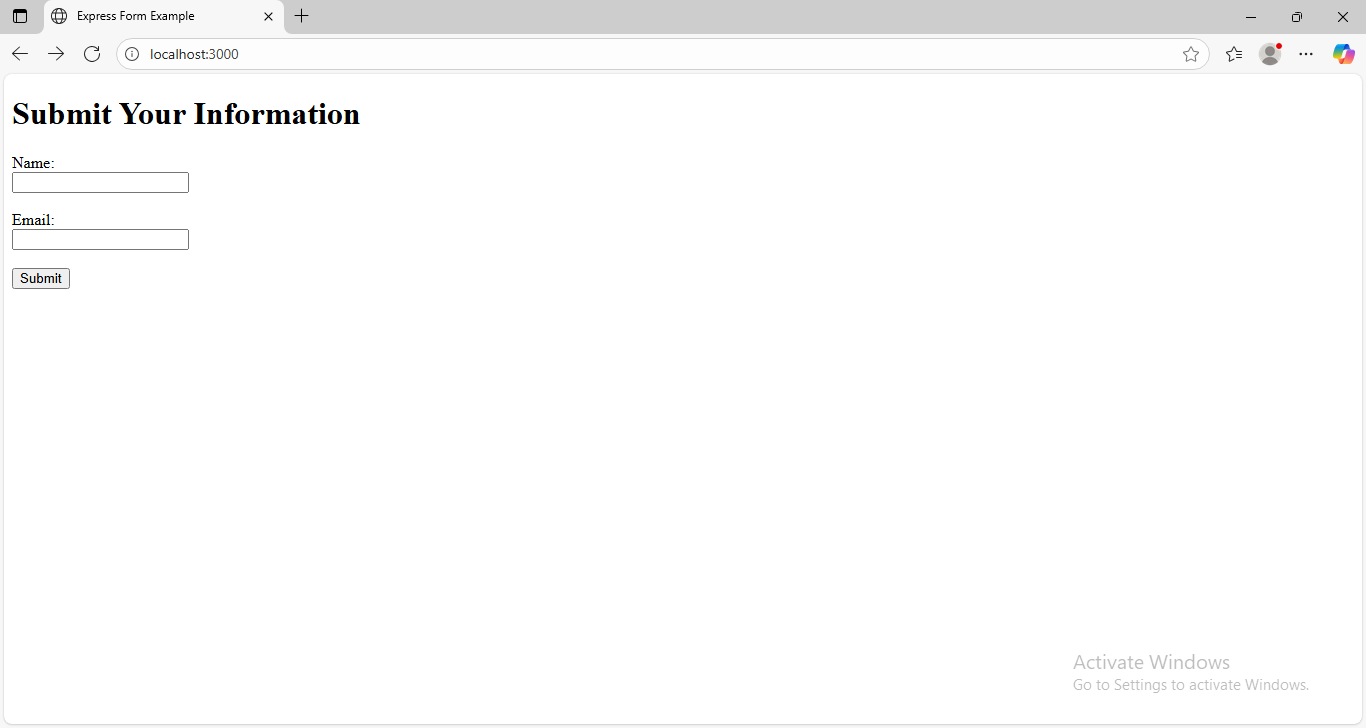
1. Create your project folder and files as above.
2. Run these commands in your project root:

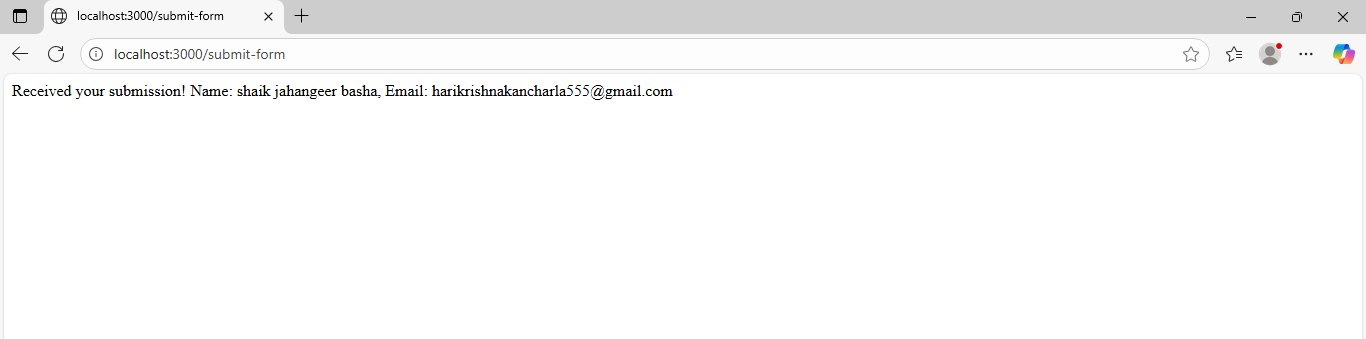
npm init -y

npm install express

node app.js

1. Open your browser to http://localhost:3000/
2. Fill out the form and submit — you’ll see a confirmation page with the submitted data.





3. Express JS – Cookies, Sessions, Authentication

a. Write a program for session management using cookies and sessions.

b. Write a program for user authentication.

a. Write a program for session management using cookies and sessions.

>npm install express express-session cookie-parser

>npm install express express-session

Code:

const express = require('express');

const session = require('express-session');

const app = express();

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

// Basic session setup

app.use(session({

secret: 'secret',

saveUninitialized: true,

resave: false

}));

// Home route

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

if (req.session.user) {

res.send(`Hello ${req.session.user} <a href="/logout">Logout</a>`);

} else {

res.send(`<form method="POST" action="/login">

<input name="user" placeholder="Username" required>

<button>Login</button>

</form>`);

}

});

// Login route

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

req.session.user = req.body.user;

res.redirect('/');

});

// Logout route

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

req.session.destroy(() => res.redirect('/'));

});

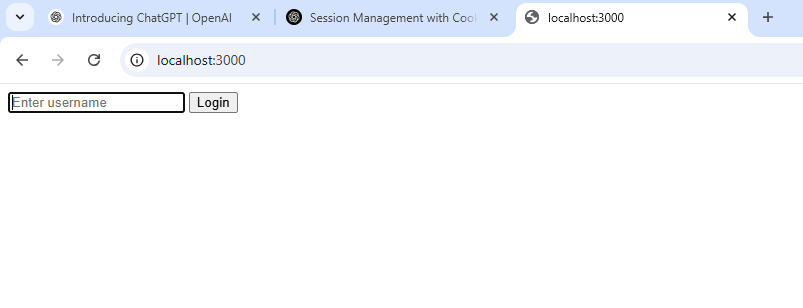
// Start server

app.listen(3000, () => console.log('Running on http://localhost:3000'));

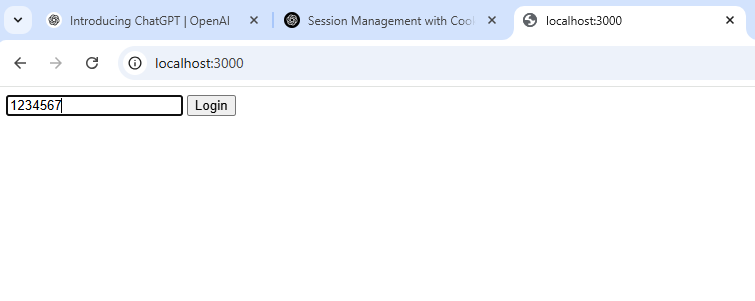
Output:

>node [app.js](http://app.js)

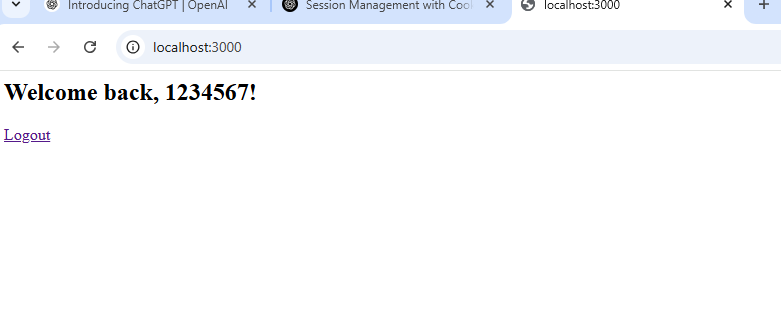
Then open:<http://localhost:3000>



Input:



Output



3 b. Write a program for user authentication.

code:

const express = require('express');

const session = require('express-session');

const app = express();

const PORT = 3000;

// Middleware

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

// Session setup

app.use(session({

secret: 'auth-secret',

resave: false,

saveUninitialized: true

}));

// Hardcoded users (for demo only)

const users = [

{ username: 'admin', password: 'admin123' },

{ username: 'user', password: 'user123' }

];

// Home/Login page

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

res.send(`

<h2>Login</h2>

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

<input name="username" placeholder="Username" required><br>

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

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

</form>

`);

});

// Login logic

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

const { username, password } = req.body;

const user = users.find(u => u.username === username && u.password === password);

if (user) {

req.session.user = user.username;

res.redirect('/dashboard');

} else {

res.send('Invalid credentials. <a href="/">Try again</a>');

}

});

// Protected route

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

if (req.session.user) {

res.send(`

<h2>Welcome, ${req.session.user}!</h2>

<a href="/logout">Logout</a>

`);

} else {

res.redirect('/');

}

});

// Logout route

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

req.session.destroy(() => {

res.redirect('/');

});

});

// Start server

app.listen(PORT, () => {

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

});

Output:

Another code:

const express = require('express');

const session = require('express-session');

const app = express();

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

// Setup session

app.use(session({

secret: 'secret',

resave: false,

saveUninitialized: true

}));

// Hardcoded user

const USER = { username: 'admin', password: '123' };

// Login page

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

res.send(`

<h2>Login</h2>

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

<input name="username" required placeholder="Username"><br>

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

<button>Login</button>

</form>

`);

});

// Login logic

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

const { username, password } = req.body;

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

req.session.user = username;

res.redirect('/dashboard');

} else {

res.send('Invalid login. <a href="/">Try again</a>');

}

});

// Protected route

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

if (req.session.user) {

res.send(`Welcome ${req.session.user}! <a href="/logout">Logout</a>`);

} else {

res.redirect('/');

}

});

// Logout

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

req.session.destroy(() => {

res.redirect('/');

});

});

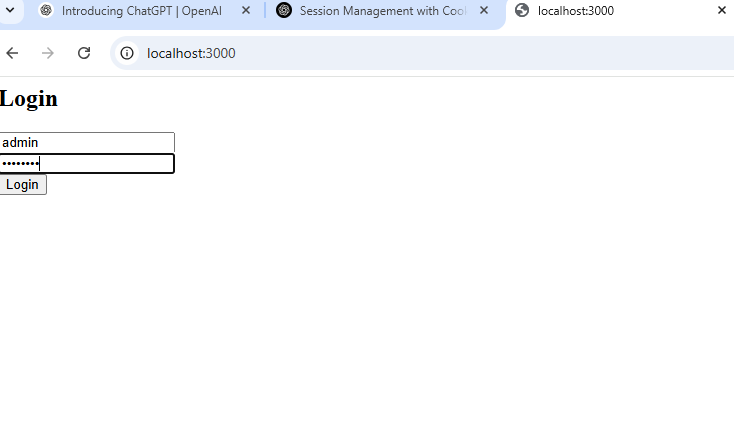
// Start server

app.listen(3000, () => {

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

});

Output:



4. Express JS – Database, RESTful APIs

a. Write a program to connect MongoDB database using Mangoose and perform CRUD

operations.

b. Write a program to develop a single page application using RESTful APIs.

a. Write a program to connect MongoDB database using Mangoose and perform CRUD

Operations.

Required to install

>npm init -y

>npm install express mongoose

>npm install express mongoose body-parser

Code: [app.js](http://app.js) or

Create [appformmoong.js](http://appformmoon.js) file

const express = require('express');

const mongoose = require('mongoose');

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

const app = express();

app.use(bodyParser.json());

// Replace with your MongoDB Atlas URI or local MongoDB URI

const mongoURI = 'mongodb://localhost:27017/dataex1';

mongoose.connect(mongoURI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'Connection error:'));

db.once('open', () => console.log('Connected to MongoDB'));

// Define a schema and model

const userSchema = new mongoose.Schema({

name: String,

age: Number,

email: String,

});

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

// CRUD API routes

// Create user

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

try {

const user = new User(req.body);

const saved = await user.save();

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

} catch (err) {

res.status(400).json({ error: err.message });

}

});

// Read all users

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

const users = await User.find();

res.json(users);

});

// Read single user by id

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

try {

const user = await User.findById(req.params.id);

if (!user) return res.status(404).json({ error: 'User not found' });

res.json(user);

} catch {

res.status(400).json({ error: 'Invalid ID' });

}

});

// Update user by id

app.put('/users/:id', async (req, res) => {

try {

const updated = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });

if (!updated) return res.status(404).json({ error: 'User not found' });

res.json(updated);

} catch {

res.status(400).json({ error: 'Invalid ID' });

}

});

// Delete user by id

app.delete('/users/:id', async (req, res) => {

try {

const deleted = await User.findByIdAndDelete(req.params.id);

if (!deleted) return res.status(404).json({ error: 'User not found' });

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

} catch {

res.status(400).json({ error: 'Invalid ID' });

}

});

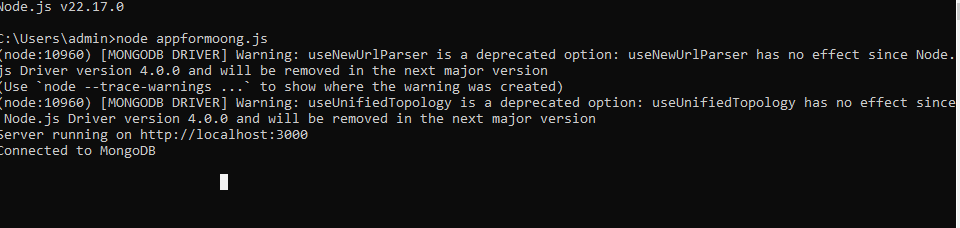
// Start server

const PORT = 3000;

app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));

Output:

>node appformoong.js



4 b. Write a program to develop a single page application using RESTful APIs.

Here’s a minimal example of a frontend using plain HTML + JavaScript that talks to the above API.

Create a file index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>User Manager SPA</title>

</head>

<body>

<h1>User Manager</h1>

<form id="userForm">

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

<input type="number" id="age" placeholder="Age" required />

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

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

</form>

<ul id="userList"></ul>

<script>

const API\_URL = 'http://localhost:3000/users';

// Load users from API and display

async function loadUsers() {

const res = await fetch(API\_URL);

const users = await res.json();

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

list.innerHTML = '';

users.forEach(user => {

const li = document.createElement('li');

li.textContent = `${user.name} (${user.age}) - ${user.email} `;

// Delete button

const delBtn = document.createElement('button');

delBtn.textContent = 'Delete';

delBtn.onclick = async () => {

await fetch(`${API\_URL}/${user.\_id}`, { method: 'DELETE' });

loadUsers();

};

li.appendChild(delBtn);

list.appendChild(li);

});

}

// Add user form submit handler

document.getElementById('userForm').onsubmit = async (e) => {

e.preventDefault();

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

const age = parseInt(document.getElementById('age').value);

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

await fetch(API\_URL, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ name, age, email }),

});

e.target.reset();

loadUsers();

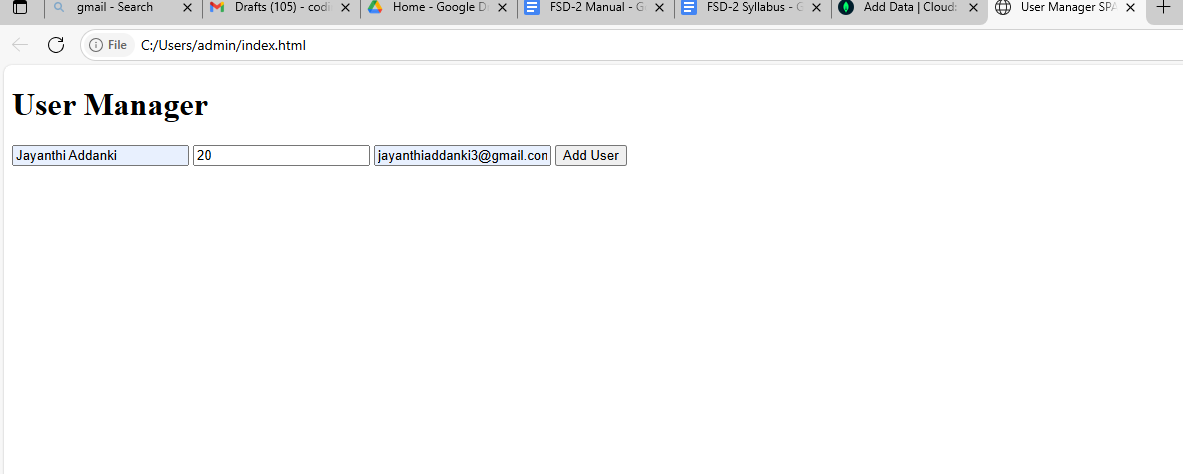
};

loadUsers();

</script>

</body>

</html>



5. ReactJS – Render HTML, JSX, Components – function & Class

a. Write a program to render HTML to a web page.

b. Write a program for writing markup with JSX.

c. Write a program for creating and nesting components (function and class).

a. Write a program to render HTML to a web page.

>npx create-react-app my-app

Need to install the following packages:

create-react-app@5.1.0

Ok to proceed? (y) y

npm warn deprecated fstream-ignore@1.0.5: This package is no longer supported.

npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.

npm warn deprecated uid-number@0.0.6: This package is no longer supported.

npm warn deprecated rimraf@2.7.1: Rimraf versions prior to v4 are no longer supported

npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

npm warn deprecated fstream@1.0.12: This package is no longer supported.

npm warn deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.

create-react-app is deprecated.

You can find a list of up-to-date React frameworks on react.dev

For more info see:https://react.dev/link/cra

This error message will only be shown once per install.

Creating a new React app in C:\Users\Admin\my-app.

Installing packages. This might take a couple of minutes.

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

added 1322 packages in 1m

269 packages are looking for funding

run `npm fund` for details

Git repo not initialized Error: Command failed: git --version

at genericNodeError (node:internal/errors:983:15)

at wrappedFn (node:internal/errors:537:14)

at checkExecSyncError (node:child\_process:883:11)

at execSync (node:child\_process:955:15)

at tryGitInit (C:\Users\Admin\my-app\node\_modules\react-scripts\scripts\init.js:46:5)

at module.exports (C:\Users\Admin\my-app\node\_modules\react-scripts\scripts\init.js:276:7)

at [eval]:3:14

at runScriptInThisContext (node:internal/vm:209:10)

at node:internal/process/execution:449:12

at [eval]-wrapper:6:24 {

status: 1,

signal: null,

output: [ null, null, null ],

pid: 7496,

stdout: null,

stderr: null

}

Installing template dependencies using npm...

added 18 packages, and changed 1 package in 7s

269 packages are looking for funding

run `npm fund` for details

Removing template package using npm...

removed 1 package, and audited 1340 packages in 5s

269 packages are looking for funding

run `npm fund` for details

9 vulnerabilities (3 moderate, 6 high)

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

npm audit fix --force

Run `npm audit` for details.

Success! Created my-app at C:\Users\Admin\my-app

Inside that directory, you can run several commands:

npm start

Starts the development server.

npm run build

Bundles the app into static files for production.

npm test

Starts the test runner.

npm run eject

Removes this tool and copies build dependencies, configuration files

and scripts into the app directory. If you do this, you can’t go back!

We suggest that you begin by typing:

cd my-app

npm start

Happy hacking!

>cd my-app

my-app>npm start

A complete log of this run can be found in: C:\Users\Admin\AppData\Local\npm-cache\\_logs\2025-07-09T11\_16\_10\_949Z-debug-0.log

(node:9372) [DEP\_WEBPACK\_DEV\_SERVER\_ON\_AFTER\_SETUP\_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.

(Use `node --trace-deprecation ...` to show where the warning was created)

(node:9372) [DEP\_WEBPACK\_DEV\_SERVER\_ON\_BEFORE\_SETUP\_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.app@0.1.0 start

> react-scripts start

Starting the development server...

Compiled successfully!

You can now view my-app in the browser.

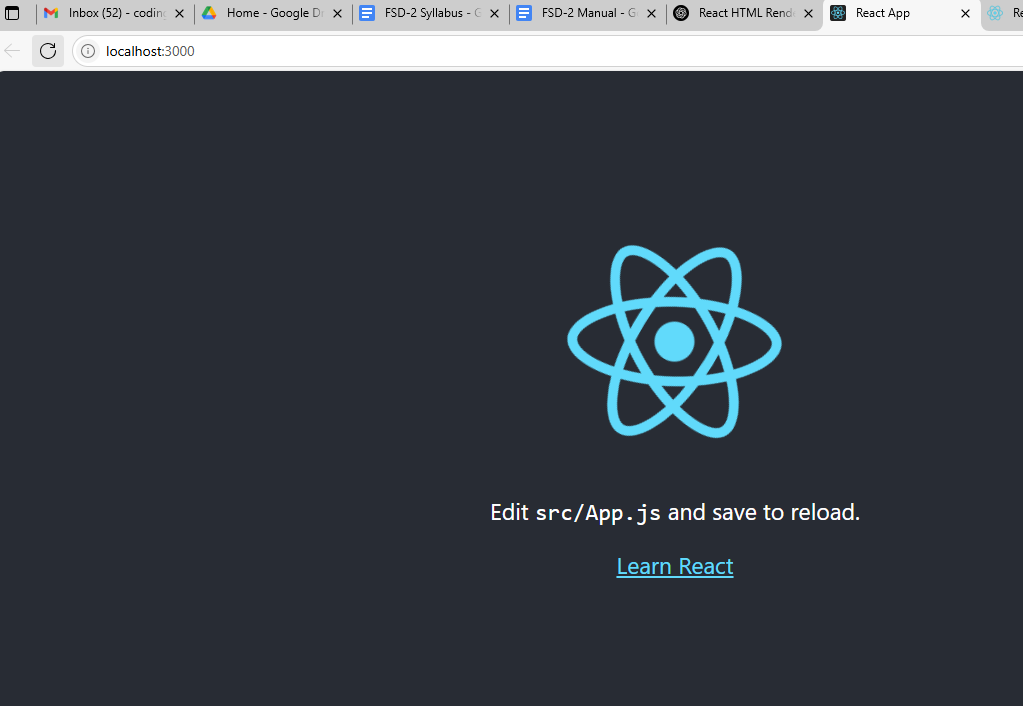
Local: http://localhost:3000

On Your Network: http://172.16.212.193:3000

Note that the development build is not optimized.

To create a production build, use npm run build.

webpack compiled successfully



Next goto folder

C drive

users

Admin

My-app

Src

[app.js](http://app.js)

Edit this [app.js](http://app.js) with the following code

import React from 'react';

function App() {

return (

<div>

<h1>Hello, React!</h1>

<p>This is a simple React app rendering HTML content.</p>

<ul>

<li>Item one</li>

<li>Item two</li>

<li>Item three</li>

</ul>

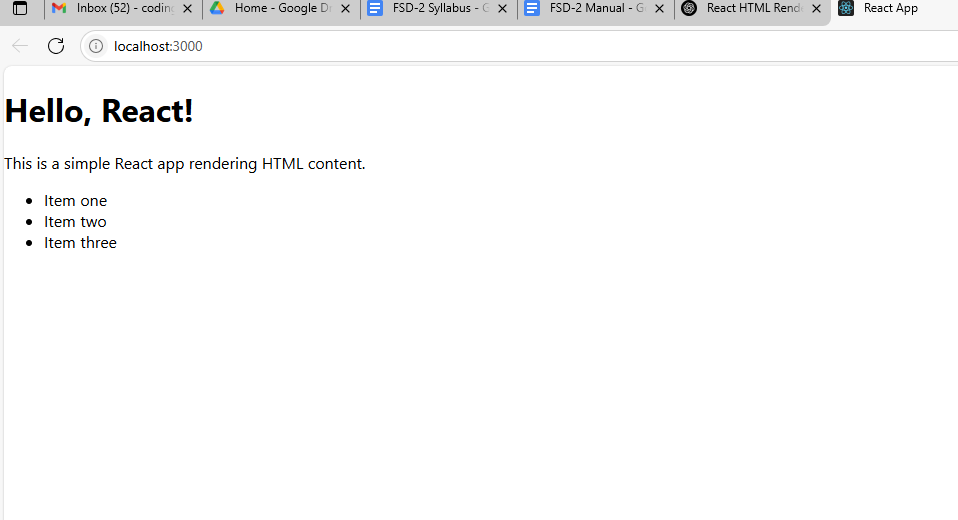
</div>

);

}

export default App;

Output:



b. Write a program for writing markup with JSX.

Next goto folder

C drive

users

Admin

My-app

Src

[index.js](http://index.js)

Write the following code in [index.js](http://index.js)

import React from 'react';

import ReactDOM from 'react-dom/client';

function MarkupExample() {

return (

<div>

<h2>JSX Markup Example</h2>

<p>This paragraph is written using JSX syntax.</p>

<section>

<h3>List of fruits:</h3>

<ul>

<li>Apple 🍎</li>

<li>Banana 🍌</li>

<li>Cherry 🍒</li>

</ul>

</section>

</div>

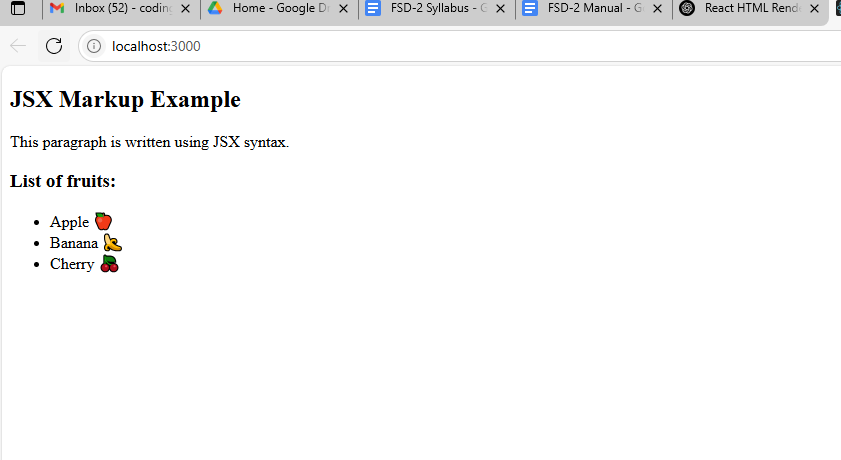
);

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<MarkupExample />);

Output:



c. Write a program for creating and nesting components (function and class).

Next goto folder

C drive

users

Admin

My-app

Src

[index.js](http://index.js)

Write the following code in [index.js](http://index.js)

import React, { Component } from 'react';

import ReactDOM from 'react-dom/client';

// Function Component (Child)

function Greeting(props) {

return <h2>Hello, {props.name}!</h2>;

}

// Class Component (Parent)

class App extends Component {

render() {

return (

<div>

<h1>Welcome to My React App</h1>

{/\* Nesting Function Component \*/}

<Greeting name="Alice" />

<Greeting name="Bob" />

<Greeting name="Charlie" />

</div>

);

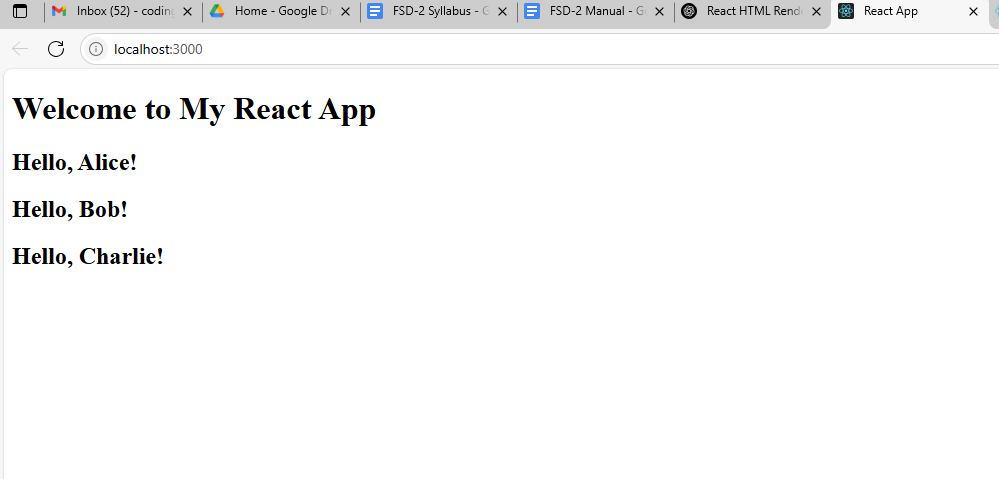
}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);

Output:



6. ReactJS – Props and States, Styles, Respond to Events

a. Write a program to work with props and states.

b. Write a program to add styles (CSS & Sass Styling) and display data.

c. Write a program for responding to events.

a. Write a program to work with props and states.

import React, { useState } from 'react';

function Counter({ initialCount }) {

// Initialize state with the value received from props

const [count, setCount] = useState(initialCount);

// Function to increment count

const increment = () => {

setCount(count + 1);

};

return (

<div style={{ textAlign: 'center', marginTop: '2rem' }}>

<h2>Counter Component</h2>

<p>Starting from prop: {initialCount}</p>

<p>Current count (state): {count}</p>

<button onClick={increment}>Increment</button>

</div>

);

}

// Usage example

export default function App() {

return <Counter initialCount={5} />;

}

### **Explanation:**

* initialCount is passed to Counter as a **prop**.
* The component uses useState to initialize its internal state (count) with the prop value.
* When you click the button, it updates the state (count) independently of the prop.
* This shows how props provide initial values or configuration, and state manages dynamic, interactive data.

Next goto folder

C drive

users

Admin

My-app

Src

Replace the content of [App.js](http://app.js) with the above code.

b. Write a program to add styles (CSS & Sass Styling) and display data.

Use **CSS styling** with an external .css file

Use **Sass styling** with an external .scss file

Display some data in a styled component

**Project Structure:**

/src

|-- App.js

|-- App.css (CSS file)

|-- DataList.scss (Sass file)

1. [App.js](http://app.js)

import React from 'react';

import './App.css'; // Import CSS file

import './DataList.scss'; // Import Sass file

const data = [

{ id: 1, name: 'Apple', color: 'Red', price: '$1' },

{ id: 2, name: 'Banana', color: 'Yellow', price: '$0.5' },

{ id: 3, name: 'Cherry', color: 'Red', price: '$2' },

];

export default function App() {

return (

<div className="app-container">

<h1 className="app-title">Fruit List</h1>

<ul className="data-list">

{data.map((item) => (

<li key={item.id} className="data-item">

<h3 className="fruit-name">{item.name}</h3>

<p>Color: <span className="fruit-color">{item.color}</span></p>

<p>Price: <span className="fruit-price">{item.price}</span></p>

</li>

))}

</ul>

</div>

);

}

2. App.css (CSS file for overall app styling)

.app-container {

max-width: 600px;

margin: 2rem auto;

padding: 1rem;

font-family: Arial, sans-serif;

background-color: #f9f9f9;

border-radius: 8px;

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

}

.app-title {

text-align: center;

color: #333;

}

3. DataList.scss (Sass file for the list styling)

$data-item-padding: 1rem;

$data-item-border: #ddd;

.data-list {

list-style: none;

padding: 0;

.data-item {

background-color: white;

margin-bottom: 1rem;

padding: $data-item-padding;

border: 1px solid $data-item-border;

border-radius: 5px;

transition: box-shadow 0.3s ease;

&:hover {

box-shadow: 0 0 8px rgba(0, 150, 136, 0.5);

}

.fruit-name {

margin: 0;

color: #009688;

}

.fruit-color {

font-weight: bold;

color: #e91e63;

}

.fruit-price {

font-weight: bold;

color: #3f51b5;

}

}

}

### 

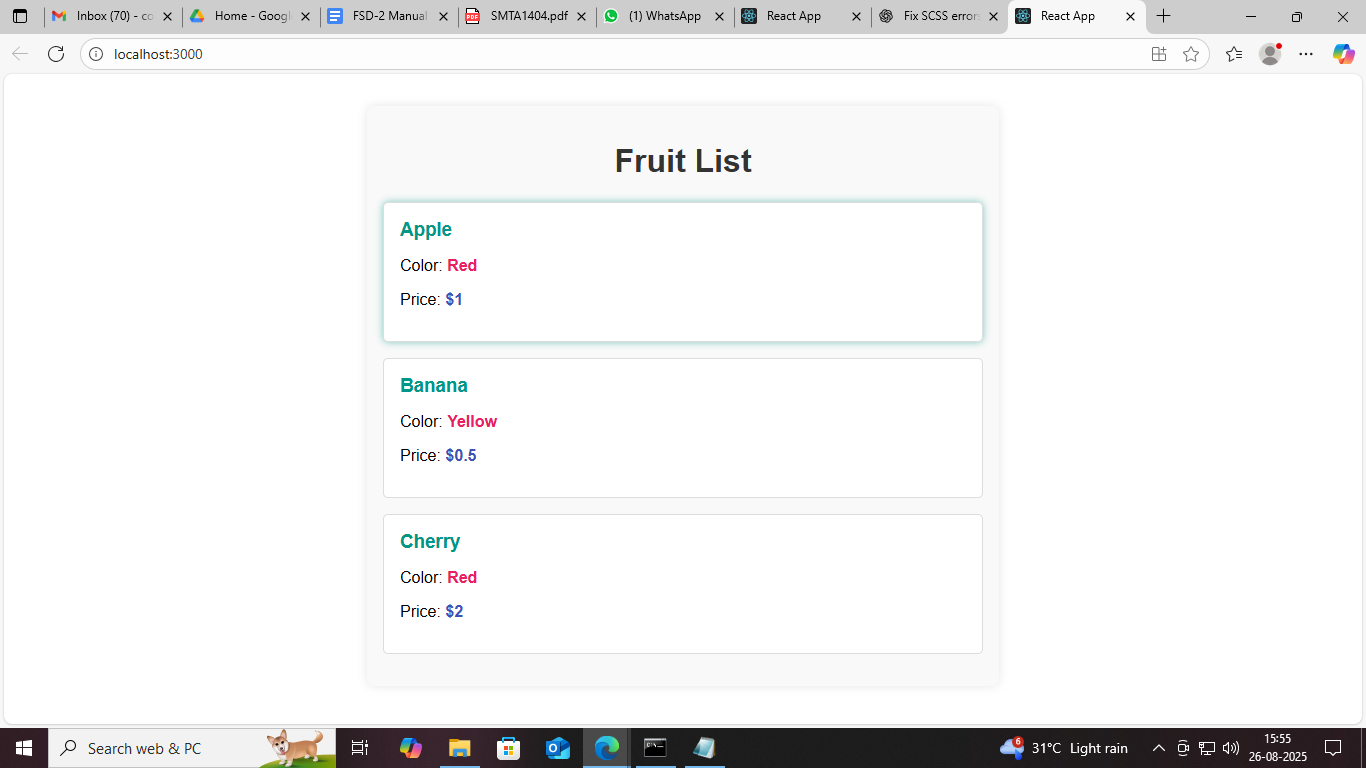
### 

### 

### **How to run this with Sass support?**

If you created your project with **Create React App**, Sass support comes built-in! Just make sure you:

1. Save the Sass file as .scss (like above)
2. Import the Sass file in your component like we did: import './DataList.scss';
3. Run your React app (npm start)



c. Write a program for responding to events.

### **What this does:**

* Tracks text input with onChange event.
* Updates the displayed message as you type.
* Tracks how many times the button has been clicked with onClick event.
* Updates and displays the click count dynamically.

Replace your [App.js](http://app.js) with the following code

import React, { useState } from 'react';

export default function EventDemo() {

const [message, setMessage] = useState('');

const [clickCount, setClickCount] = useState(0);

// Handle button click event

const handleButtonClick = () => {

setClickCount(clickCount + 1);

};

// Handle input change event

const handleInputChange = (event) => {

setMessage(event.target.value);

};

return (

<div style={{ padding: '2rem', fontFamily: 'Arial, sans-serif' }}>

<h2>React Event Handling Demo</h2>

{/\* Input field \*/}

<input

type="text"

placeholder="Type something here"

value={message}

onChange={handleInputChange}

style={{ padding: '0.5rem', fontSize: '1rem', width: '300px' }}

/>

{/\* Display the input message \*/}

<p>You typed: <strong>{message}</strong></p>

{/\* Button with click event \*/}

<button

onClick={handleButtonClick}

style={{

padding: '0.5rem 1rem',

fontSize: '1rem',

cursor: 'pointer',

backgroundColor: '#007bff',

color: 'white',

border: 'none',

borderRadius: '4px',

}}

>

Click me

</button>

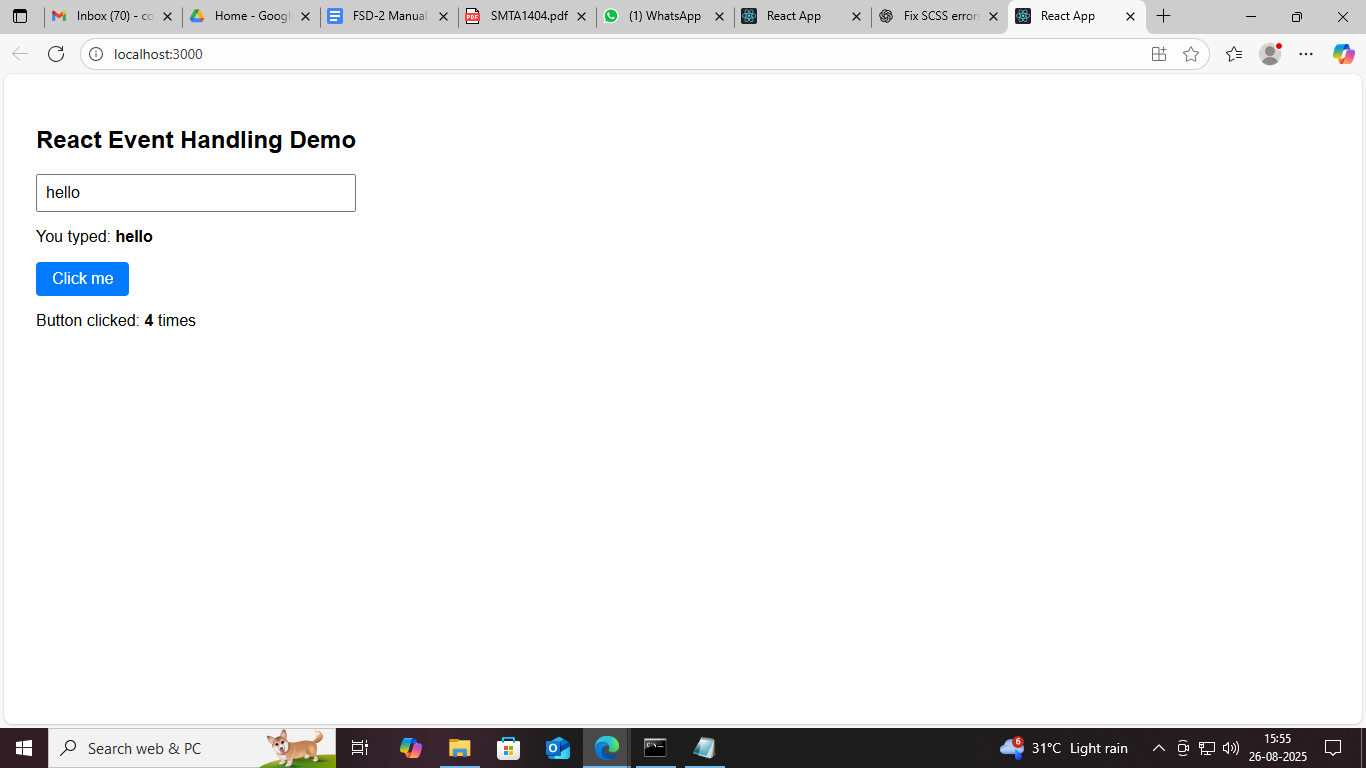
{/\* Display number of clicks \*/}

<p>Button clicked: <strong>{clickCount}</strong> times</p>

</div>

);

}



7. ReactJS – Conditional Rendering, Rendering Lists, React Forms

a. Write a program for conditional rendering.

b. Write a program for rendering lists.

c. Write a program for working with different form fields using react forms.

a. Write a program for conditional rendering.

rendering different UI based on a condition (state or props).

Replace your [App.js](http://app.js) with the following code

import React, { useState } from 'react';

export default function ConditionalRenderingDemo() {

const [isLoggedIn, setIsLoggedIn] = useState(false);

// Toggle login state

const toggleLogin = () => {

setIsLoggedIn(!isLoggedIn);

};

return (

<div style={{ fontFamily: 'Arial, sans-serif', padding: '2rem' }}>

<h2>Conditional Rendering Demo</h2>

{/\* Conditional rendering \*/}

{isLoggedIn ? (

<div>

<h3>Welcome back, User!</h3>

<p>You are logged in.</p>

</div>

) : (

<div>

<h3>Please log in</h3>

<p>You are currently not logged in.</p>

</div>

)}

<button

onClick={toggleLogin}

style={{

marginTop: '1rem',

padding: '0.5rem 1rem',

fontSize: '1rem',

cursor: 'pointer',

borderRadius: '4px',

border: 'none',

backgroundColor: isLoggedIn ? '#f44336' : '#4CAF50',

color: 'white',

}}

>

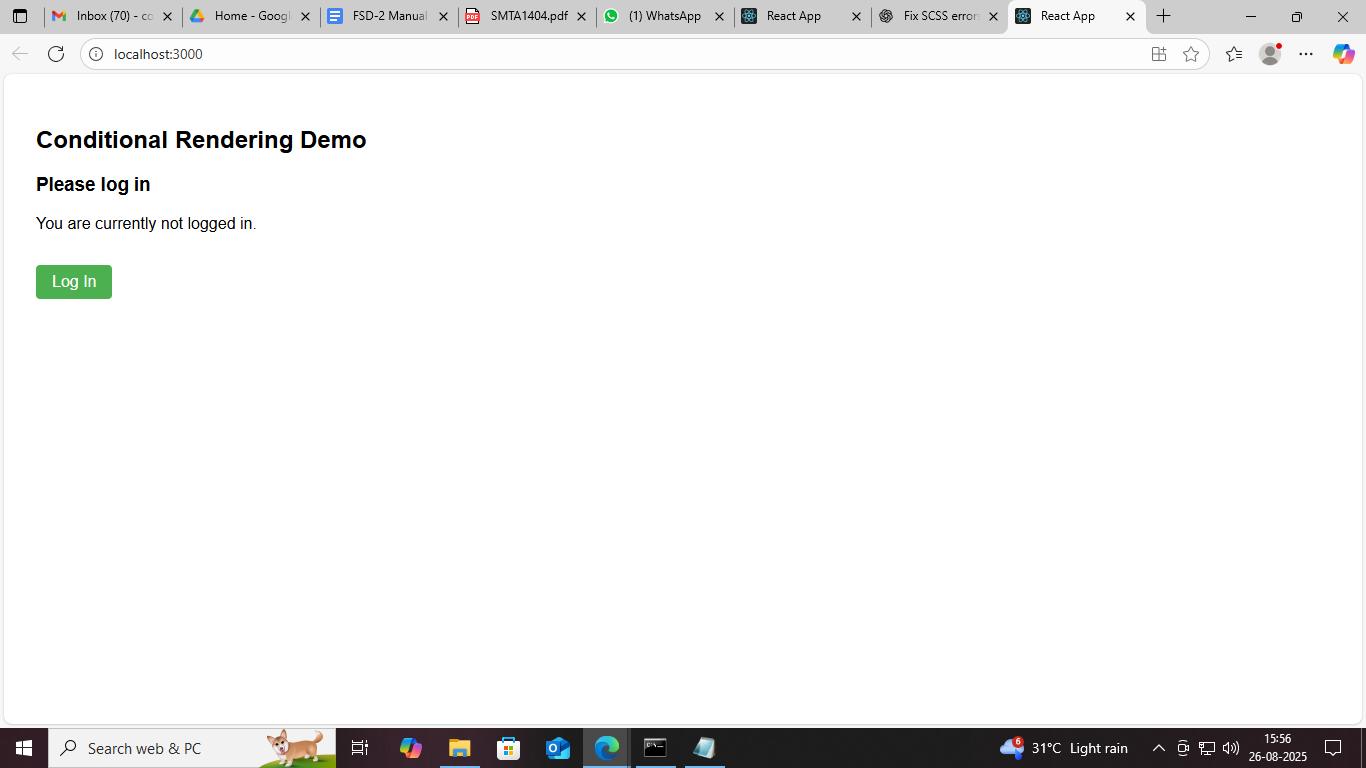
{isLoggedIn ? 'Log Out' : 'Log In'}

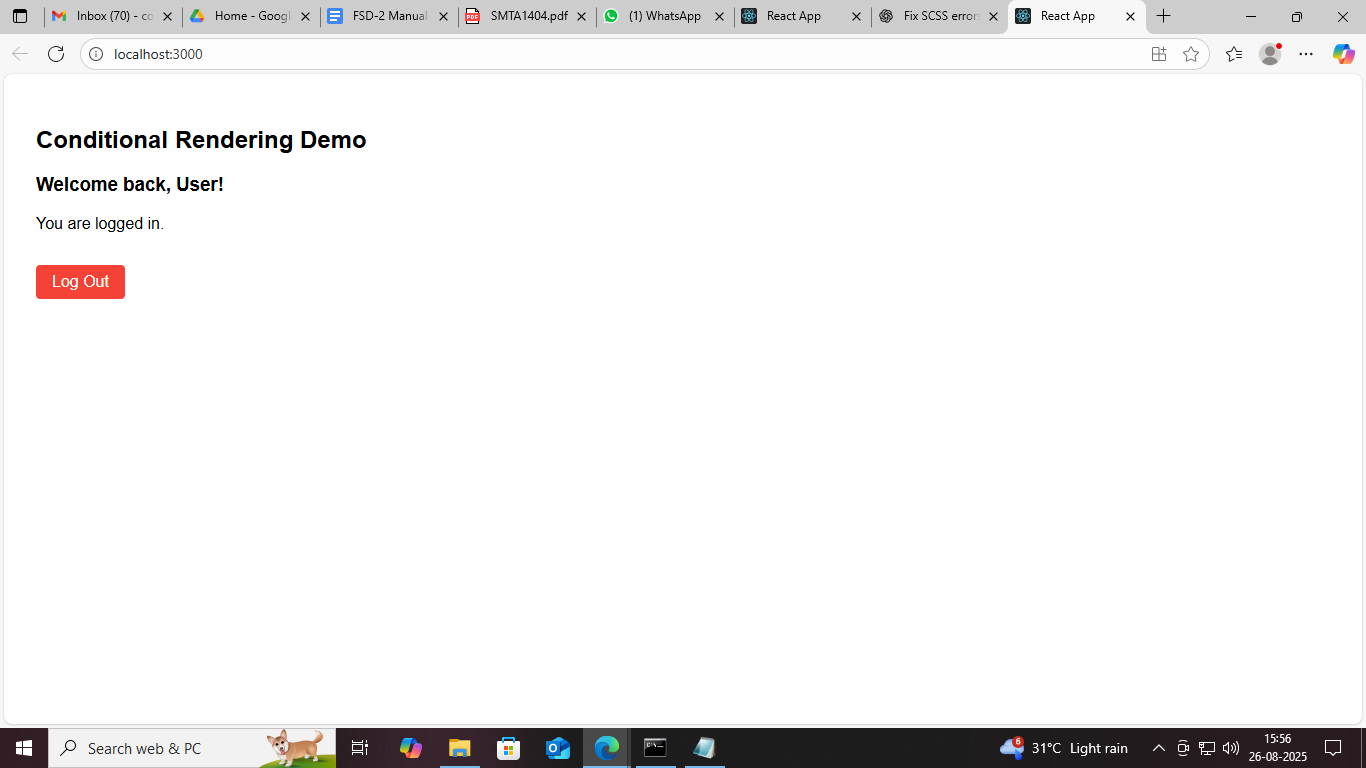
</button>

</div>

);

}





b. Write a program for rendering lists.

**rendering lists** using the .map() method.

replace your App.js with this code

import React from 'react';

export default function ListRenderingDemo() {

// Sample data array

const fruits = [

{ id: 1, name: 'Apple', color: 'Red' },

{ id: 2, name: 'Banana', color: 'Yellow' },

{ id: 3, name: 'Orange', color: 'Orange' },

{ id: 4, name: 'Grapes', color: 'Purple' },

];

return (

<div style={{ padding: '2rem', fontFamily: 'Arial, sans-serif' }}>

<h2>List Rendering Example</h2>

<ul>

{fruits.map((fruit) => (

<li key={fruit.id} style={{ marginBottom: '0.5rem' }}>

<strong>{fruit.name}</strong> — Color: {fruit.color}

</li>

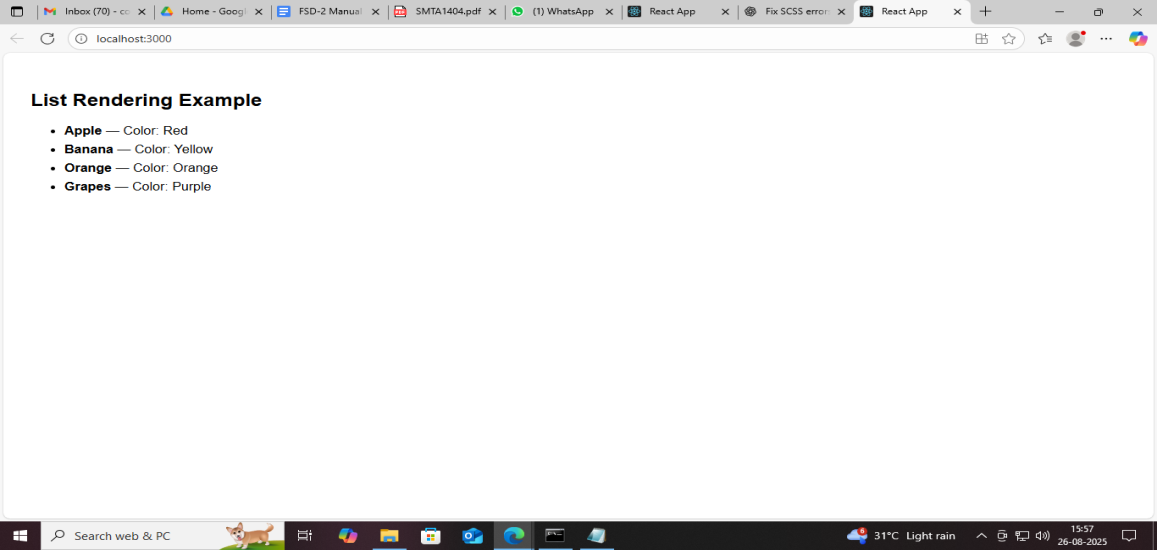
))}

</ul>

</div>

);

}



7 c. Write a program for working with different form fields using react forms.

how to work with **different form fields** (text input, checkbox, radio buttons, select dropdown) using controlled components in React forms:

Replace your App.js content with this component or import and render it.

import React, { useState } from 'react';

export default function MultiFieldForm() {

// Form state

const [formData, setFormData] = useState({

username: '',

agreeToTerms: false,

gender: '',

favoriteFruit: 'apple',

});

// Handle change for all form fields

const handleChange = (event) => {

const { name, value, type, checked } = event.target;

setFormData((prevData) => ({

...prevData,

[name]: type === 'checkbox' ? checked : value,

}));

};

// Handle form submission

const handleSubmit = (event) => {

event.preventDefault();

alert(`Form Submitted:\n${JSON.stringify(formData, null, 2)}`);

};

return (

<div style={{ maxWidth: '400px', margin: '2rem auto', fontFamily: 'Arial, sans-serif' }}>

<h2>React Form Example</h2>

<form onSubmit={handleSubmit}>

{/\* Text Input \*/}

<label>

Username:

<input

type="text"

name="username"

value={formData.username}

onChange={handleChange}

style={{ display: 'block', margin: '0.5rem 0 1rem' }}

required

/>

</label>

{/\* Checkbox \*/}

<label>

<input

type="checkbox"

name="agreeToTerms"

checked={formData.agreeToTerms}

onChange={handleChange}

/>

I agree to the terms and conditions

</label>

<br /><br />

{/\* Radio Buttons \*/}

<fieldset>

<legend>Gender:</legend>

<label>

<input

type="radio"

name="gender"

value="male"

checked={formData.gender === 'male'}

onChange={handleChange}

required

/>

Male

</label>

<br />

<label>

<input

type="radio"

name="gender"

value="female"

checked={formData.gender === 'female'}

onChange={handleChange}

/>

Female

</label>

<br />

<label>

<input

type="radio"

name="gender"

value="other"

checked={formData.gender === 'other'}

onChange={handleChange}

/>

Other

</label>

</fieldset>

<br />

{/\* Select Dropdown \*/}

<label>

Favorite Fruit:

<select

name="favoriteFruit"

value={formData.favoriteFruit}

onChange={handleChange}

style={{ display: 'block', margin: '0.5rem 0 1rem' }}

>

<option value="apple">Apple</option>

<option value="banana">Banana</option>

<option value="orange">Orange</option>

<option value="grape">Grape</option>

</select>

</label>

<button type="submit" style={{ padding: '0.5rem 1rem', cursor: 'pointer' }}>

Submit

</button>

</form>

{/\* Display Form Data \*/}

<div style={{ marginTop: '2rem' }}>

<h3>Current Form Data:</h3>

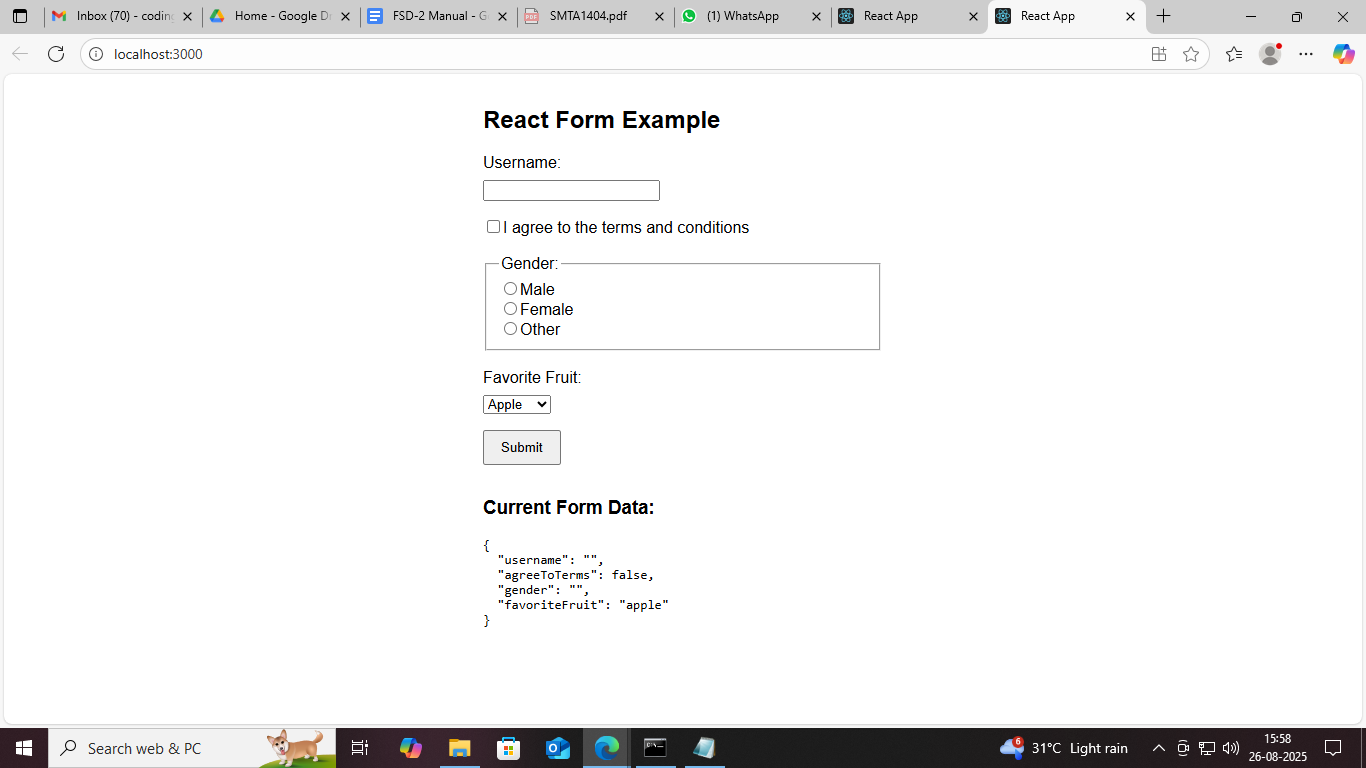
<pre>{JSON.stringify(formData, null, 2)}</pre>

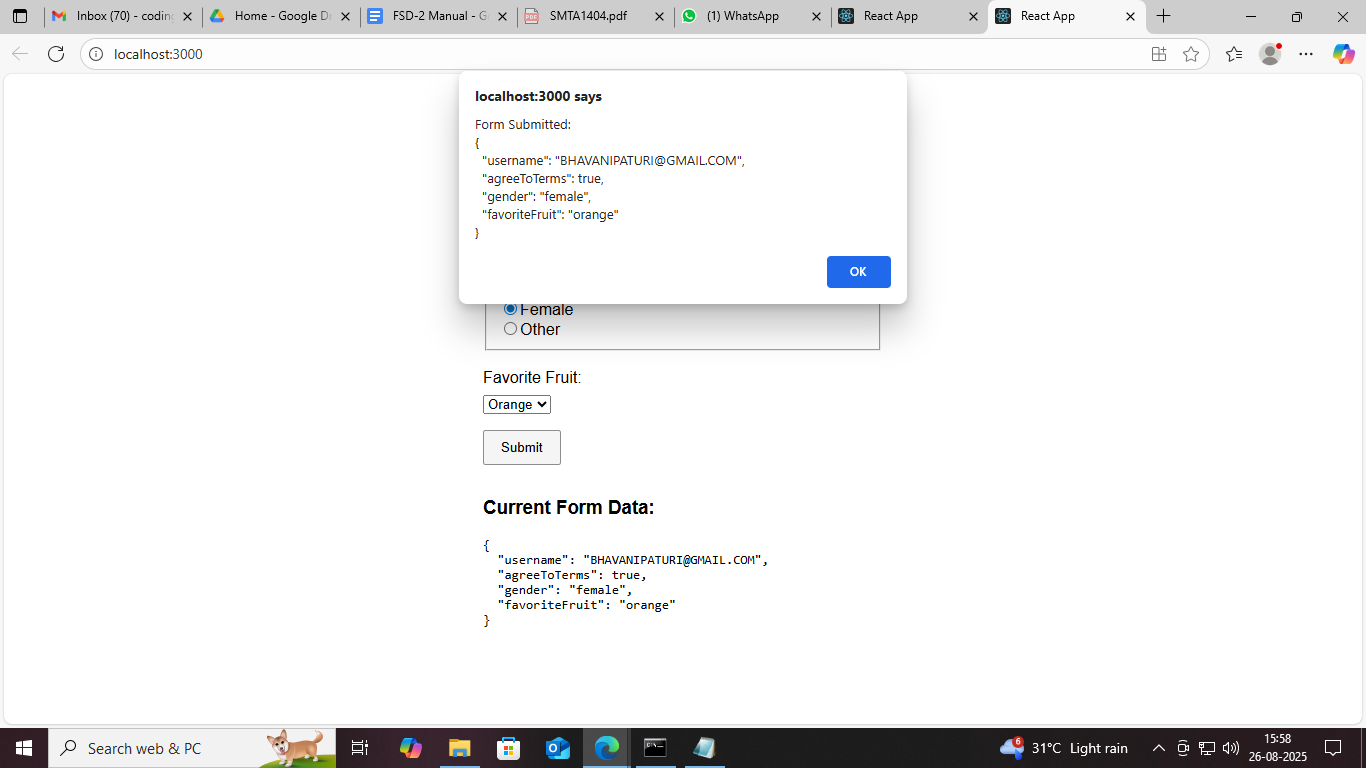
</div>

</div>

);

}





8. ReactJS – React Router, Updating the Screen

a. Write a program for routing to different pages using react router.

b. Write a program for updating the screen.

a. Write a program for routing to different pages using react router.

Make sure you have react-router-dom installed:

npm install react-router-dom

//React Router example with multiple pages:

// App.js

import React from 'react';

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

// Home page component

function Home() {

return <h2>Home Page</h2>;

}

// About page component

function About() {

return <h2>About Page</h2>;

}

// Contact page component

function Contact() {

return <h2>Contact Page</h2>;

}

export default function App() {

return (

<Router>

<div style={{ fontFamily: 'Arial, sans-serif' }}>

<nav style={{ padding: '1rem', backgroundColor: '#eee' }}>

<Link to="/" style={{ marginRight: '1rem' }}>Home</Link>

<Link to="/about" style={{ marginRight: '1rem' }}>About</Link>

<Link to="/contact">Contact</Link>

</nav>

<div style={{ padding: '1rem' }}>

<Routes>

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

<Route path="/about" element={<About />} />

<Route path="/contact" element={<Contact />} />

</Routes>

</div>

</div>

</Router>

);

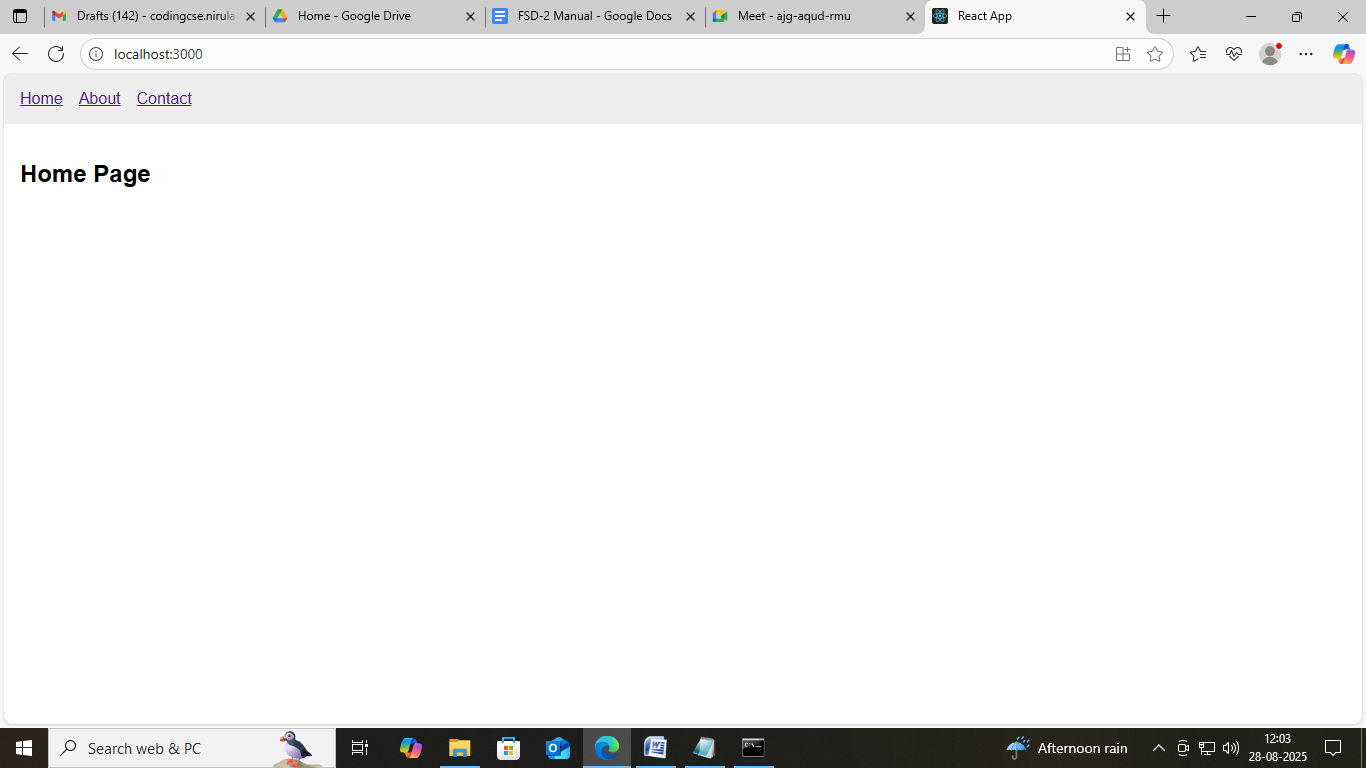
}

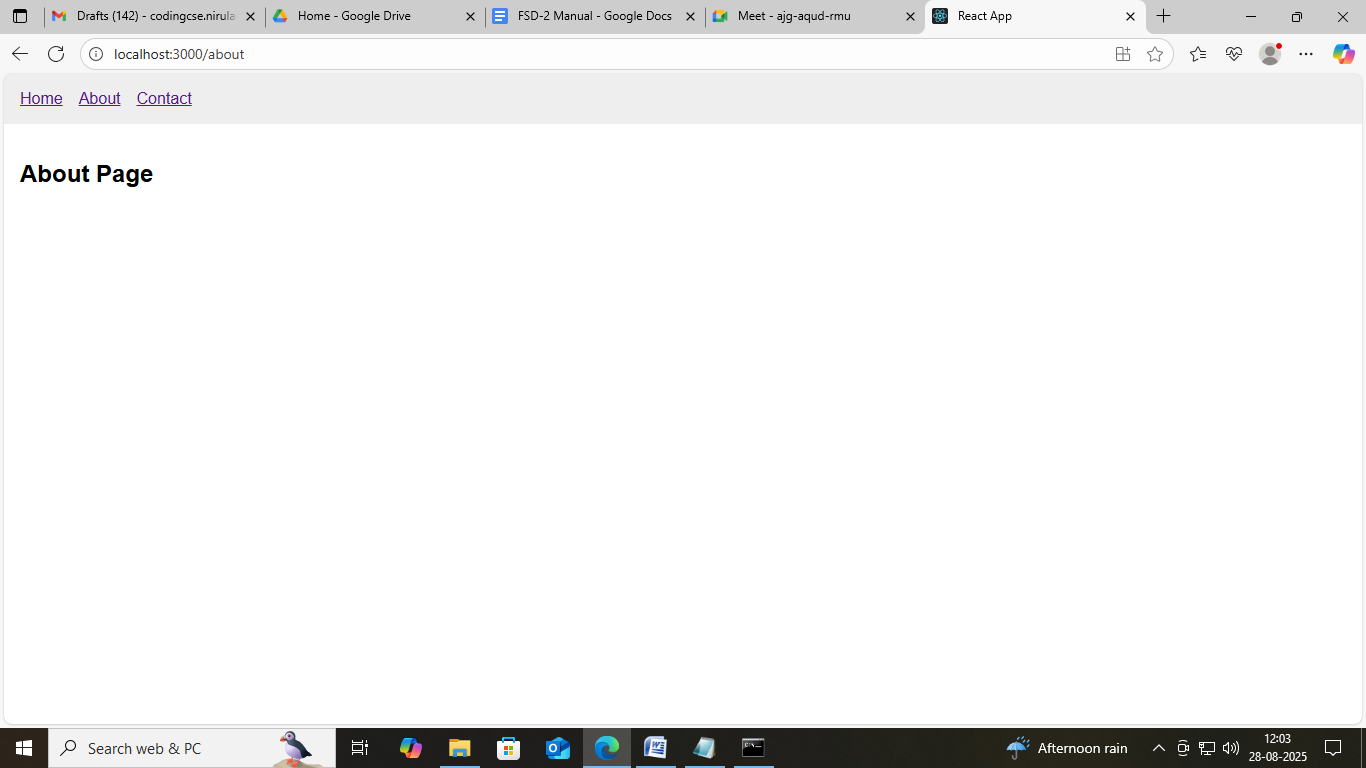
### **Explanation:**

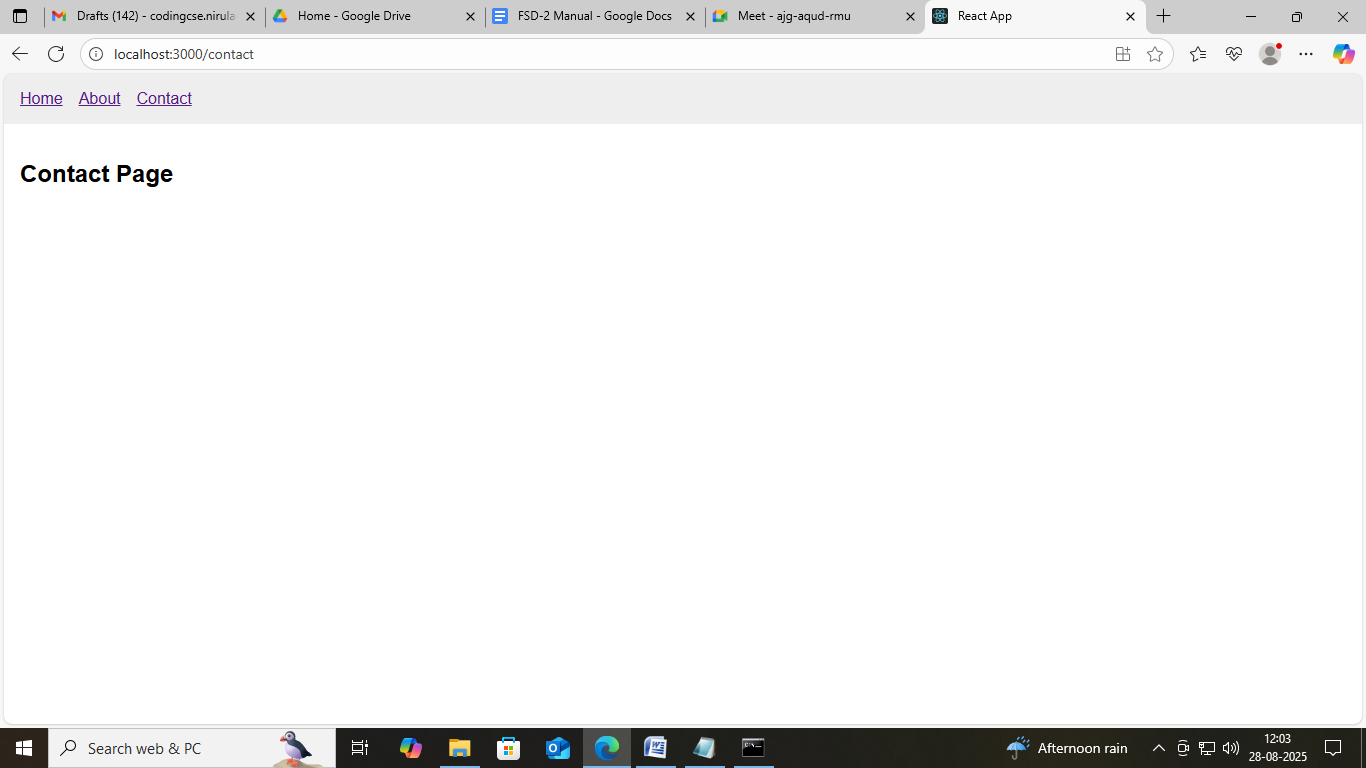
* BrowserRouter (aliased as Router) wraps the app and enables routing.
* Link components provide navigation without reloading the page.
* Routes contains multiple Route components defining paths and their respective components.
* When you visit /about, it shows the About page, etc.

### **How to run:**

1. Make sure react-router-dom is installed.
2. Replace your App.js with this code.
3. Run the app with npm start.
4. Use the navigation links to switch pages without page reload.







b. Write a program for updating the screen.

React program for updating the screen on user interaction:

import React, { useState } from 'react';

export default function ScreenUpdateDemo() {

const [count, setCount] = useState(0);

// Increment count state to update screen

const handleIncrement = () => {

setCount(count + 1);

};

// Decrement count state to update screen

const handleDecrement = () => {

setCount(count - 1);

};

return (

<div style={{ fontFamily: 'Arial, sans-serif', padding: '2rem', textAlign: 'center' }}>

<h2>Screen Update Demo</h2>

<p>Current Count: <strong>{count}</strong></p>

<button

onClick={handleIncrement}

style={{ padding: '0.5rem 1rem', marginRight: '1rem', cursor: 'pointer' }}

>

Increment

</button>

<button

onClick={handleDecrement}

style={{ padding: '0.5rem 1rem', cursor: 'pointer' }}

>

Decrement

</button>

</div>

);

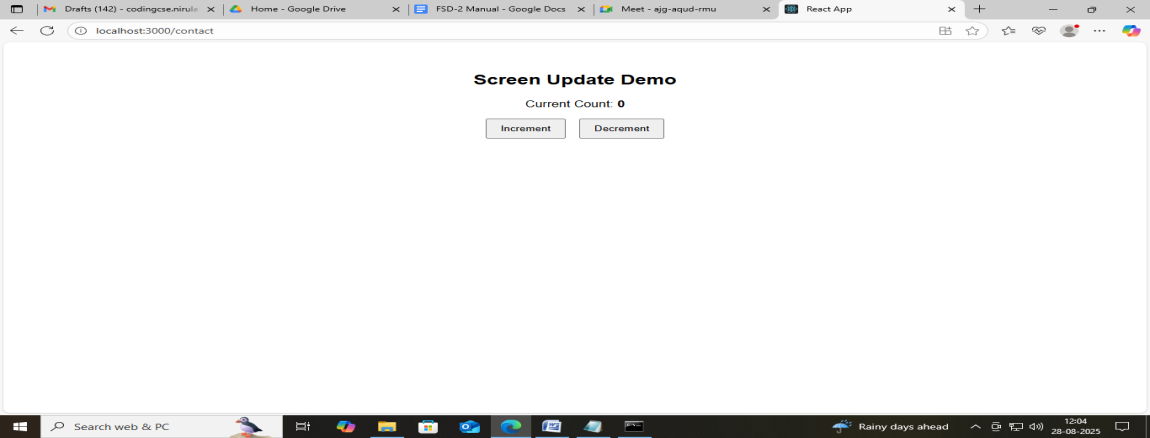
}

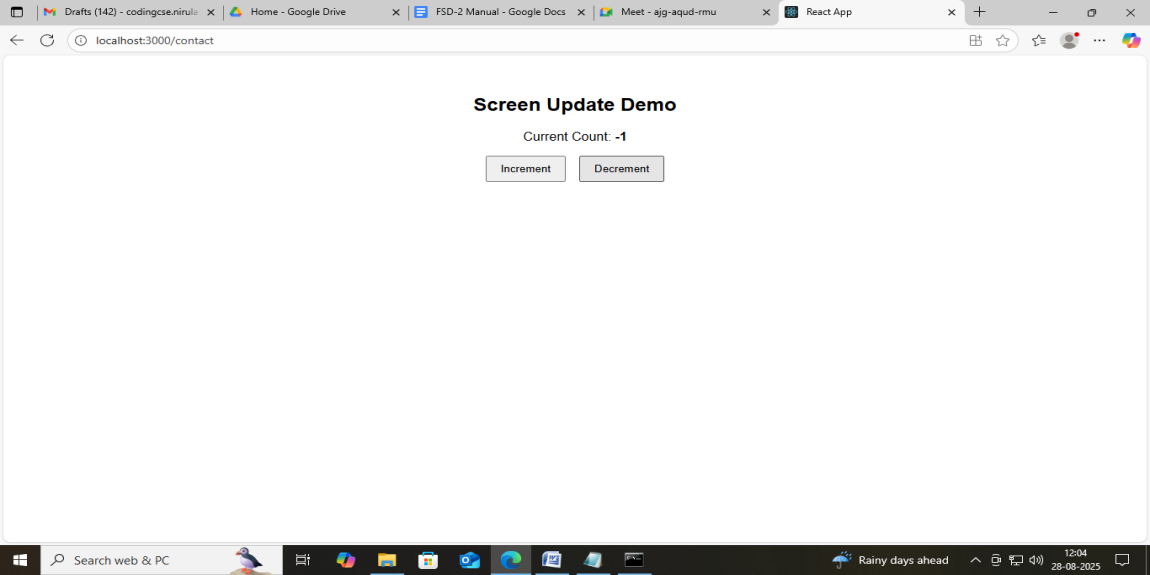
### **How it works:**

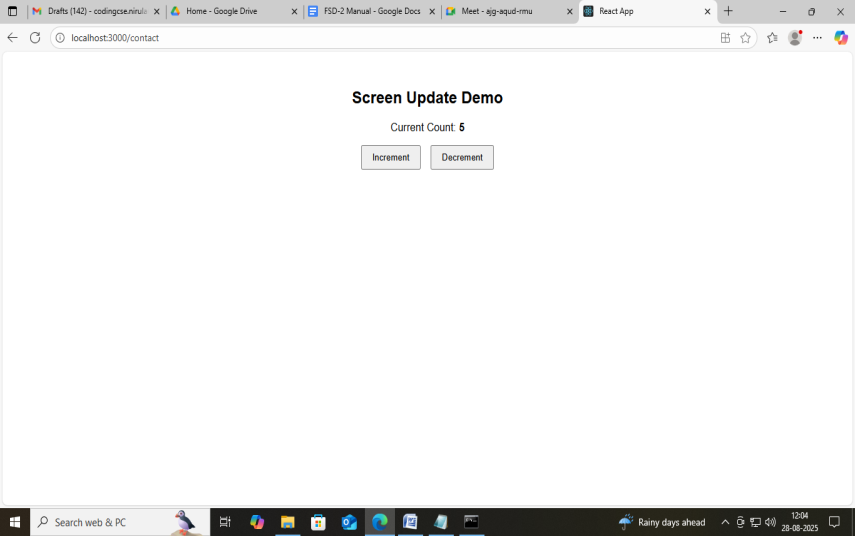
* The count is stored in React state (count).
* Clicking **Increment** or **Decrement** buttons updates the count state.
* Updating state causes React to re-render the component and update what’s displayed on the screen immediately.

### **To run:**

Replace your App.js content with this code or create a new component with it and render it.







9. ReactJS – Hooks, Sharing data between Components

a.Write a program to understand the importance of using hooks.

b. Write a program for sharing data between components.

a.Write a program to understand the importance of using hooks.

React program illustrating **why hooks are important** by comparing a class component (without hooks) and a function component (with hooks) — focusing on managing state.

import React, { useState } from 'react';

// Class component without hooks (old way)

class CounterClass extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

increment = () => {

this.setState({ count: this.state.count + 1 });

};

render() {

return (

<div style={{ border: '1px solid #ccc', padding: '1rem', marginBottom: '1rem' }}>

<h3>Class Component (No Hooks)</h3>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

// Function component with hooks (modern way)

function CounterHook() {

const [count, setCount] = useState(0);

const increment = () => setCount(count + 1);

return (

<div style={{ border: '1px solid #4caf50', padding: '1rem' }}>

<h3>Function Component (With Hooks)</h3>

<p>Count: {count}</p>

<button onClick={increment}>Increment</button>

</div>

);

}

export default function App() {

return (

<div style={{ fontFamily: 'Arial, sans-serif', padding: '2rem' }}>

<h2>Understanding the Importance of React Hooks</h2>

<CounterClass />

<CounterHook />

<p>

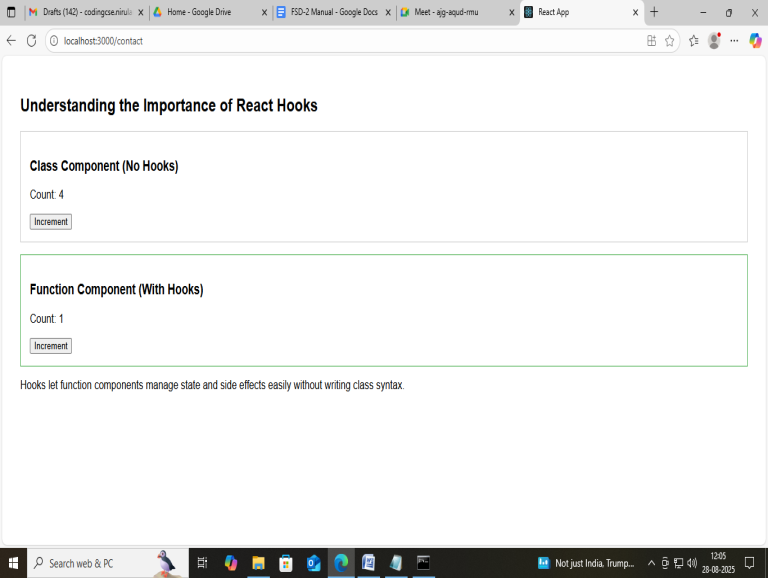
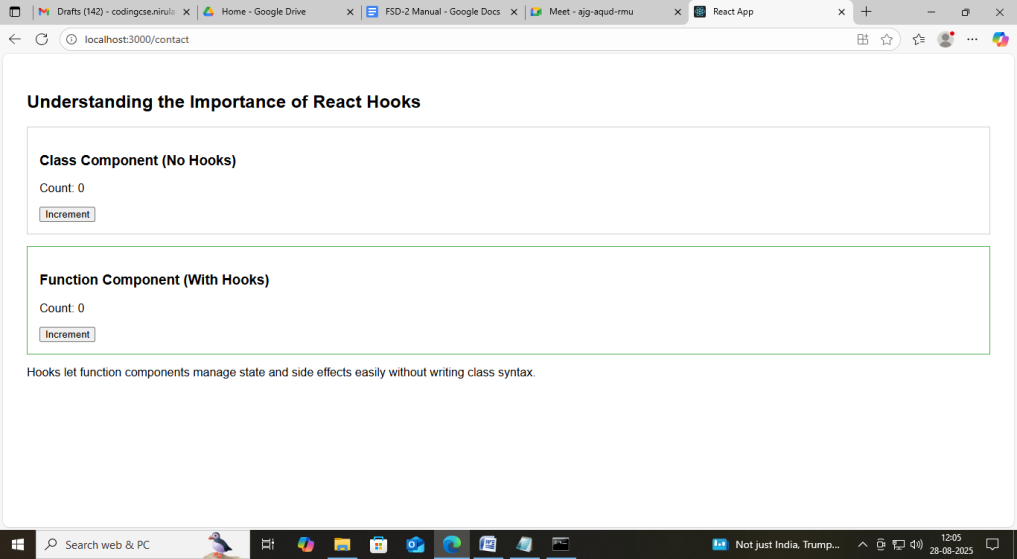
Hooks let function components manage state and side effects easily without writing class syntax.

</p>

</div>

);

}



### **How to run:**

Replace your App.js content with this code and run your React app.

b. Write a program for sharing data between components.

React program demonstrating **sharing data between components** using **props** and **lifting state up**.

import React, { useState } from 'react';

// Child component to display data passed from parent

function DisplayMessage({ message }) {

return (

<div style={{ marginTop: '1rem', padding: '1rem', backgroundColor: '#e0f7fa' }}>

<h3>Message from Parent:</h3>

<p>{message}</p>

</div>

);

}

// Child component with input to update parent's state

function MessageInput({ onMessageChange }) {

return (

<div>

<input

type="text"

placeholder="Type your message"

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

style={{ padding: '0.5rem', width: '300px' }}

/>

</div>

);

}

// Parent component holds the shared state

export default function ParentComponent() {

const [message, setMessage] = useState('');

return (

<div style={{ fontFamily: 'Arial, sans-serif', padding: '2rem' }}>

<h2>Sharing Data Between Components</h2>

{/\* Pass handler to child to update parent's state \*/}

<MessageInput onMessageChange={setMessage} />

{/\* Pass data from parent to child via props \*/}

<DisplayMessage message={message} />

</div>

);

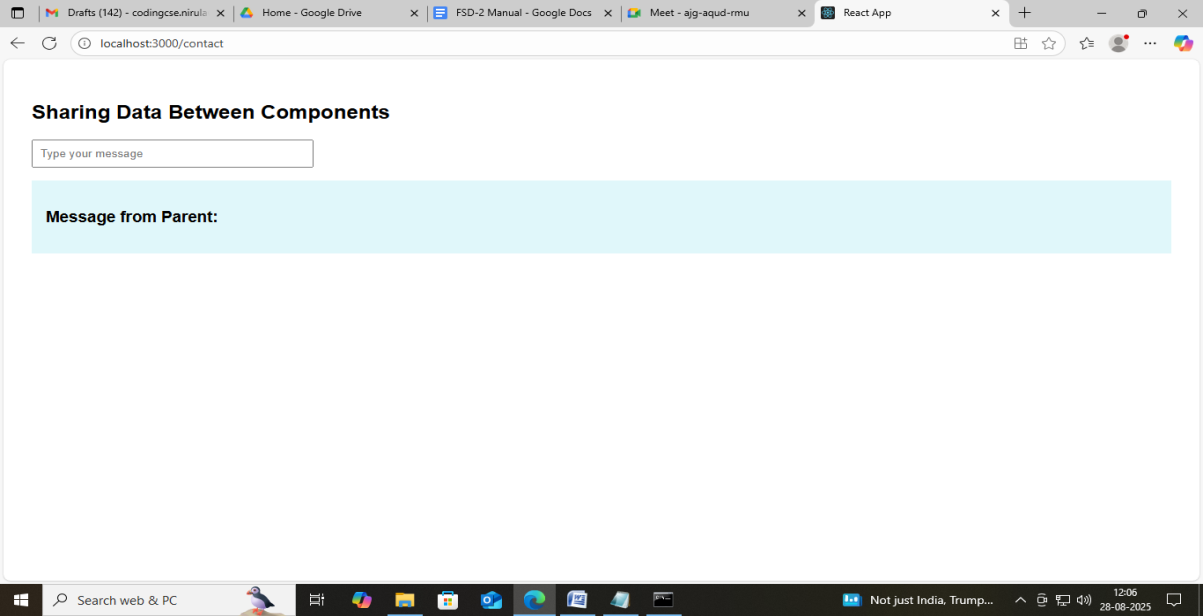
}

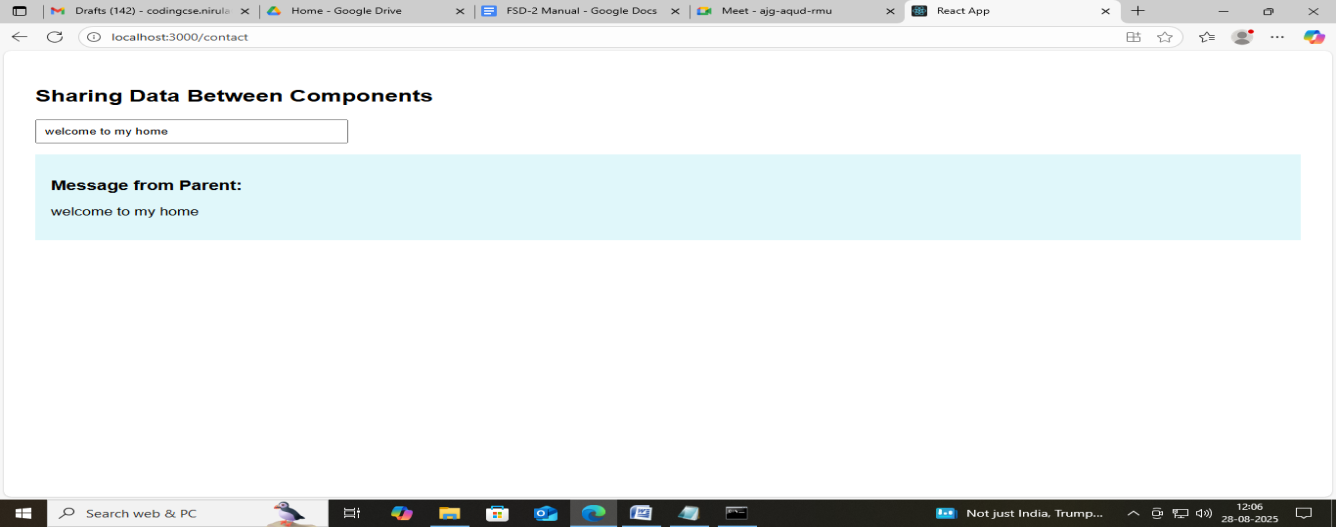
### **How it works:**

* **ParentComponent** holds the shared state message.
* **MessageInput** is a child that lets the user type a message and calls onMessageChange prop to update parent's state.
* **DisplayMessage** is another child that receives the current message via props and displays it.
* This way, **data flows down** via props, and **updates flow up** via callback functions.

### **How to run:**

Replace your App.js content with this code or create and render ParentComponent.





10. MongoDB – Installation, Configuration, CRUD operations

a. Install MongoDB and configure ATLAS

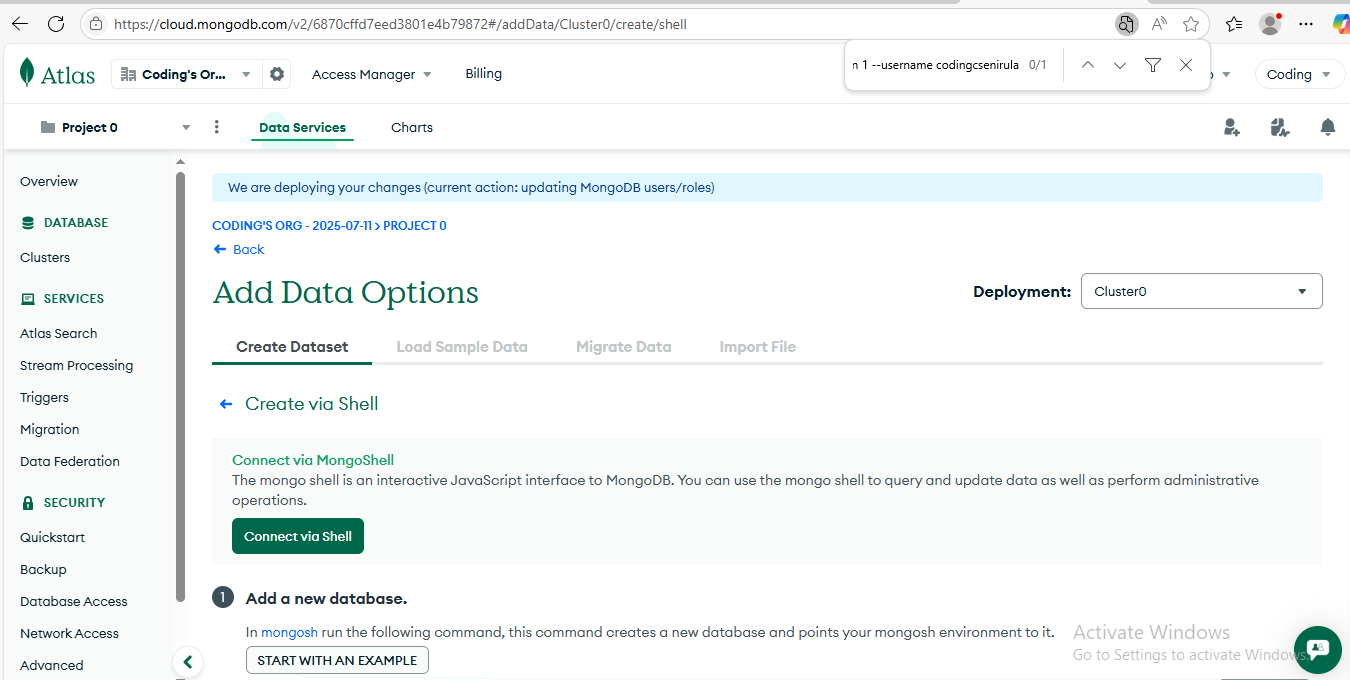
b. Write MongoDB queries to perform CRUD operations on document using insert(),

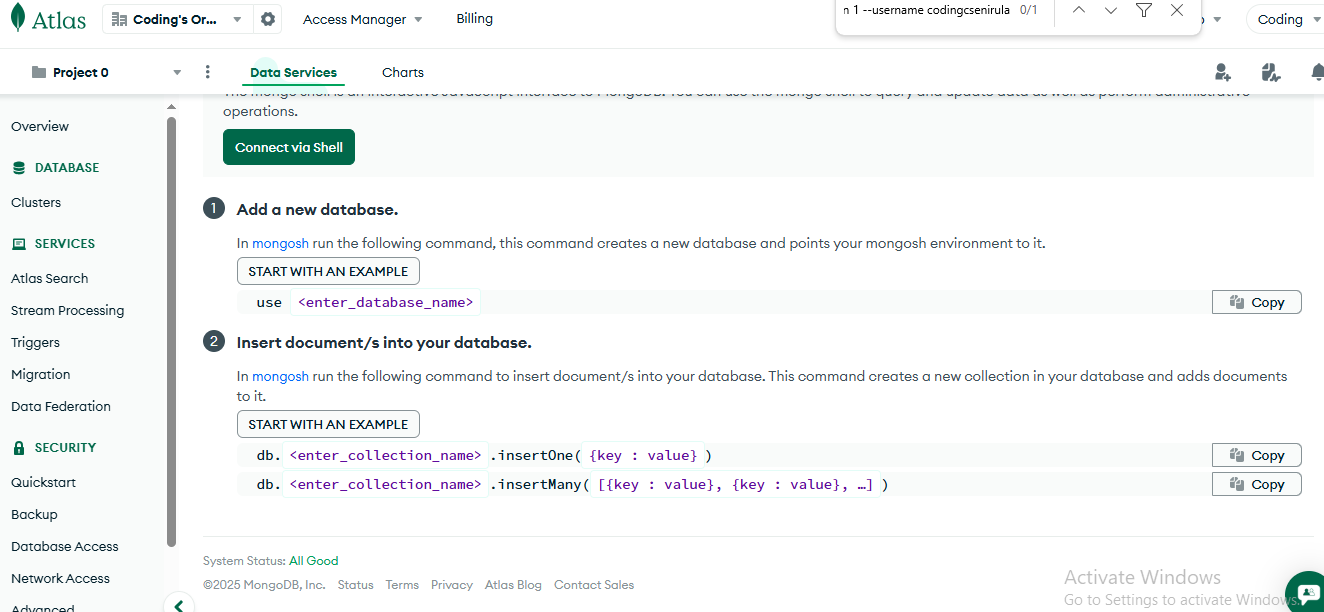
find(), update(), remove()

Install

Mongo DB Compass.

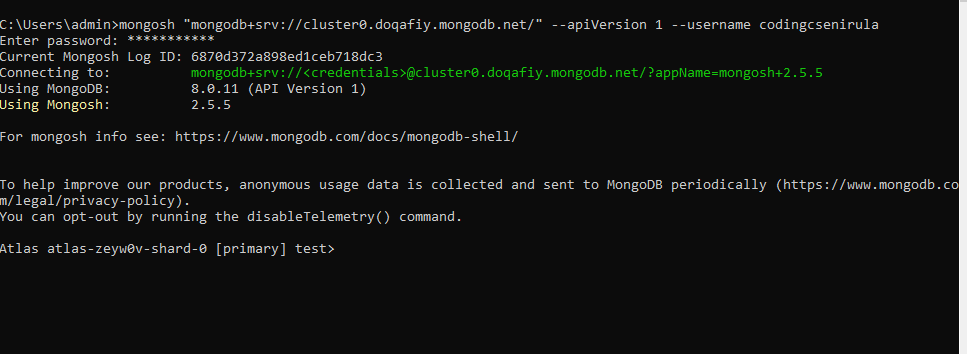
from compass connect with create new cluster of atlas. It will redirect you with google cloud account. Create a data base with help of cloud of cluster from mongo DB.





commandPrompt>

| mongosh "mongodb+srv://cluster0.doqafiy.mongodb.net/" --apiVersion 1 --username codingcsenirula |
| --- |



>use dataex1

Database created

### **1. Insert Document**

>db.users.insertOne({

name: "Alice",

age: 28,

email: "alice@example.com"

});

Database created

To check the inserted document we use find

### **Find Documents**

> db.users.find().pretty();



>db.users.find();

db.users.find({ age: { $gt: 25 } });



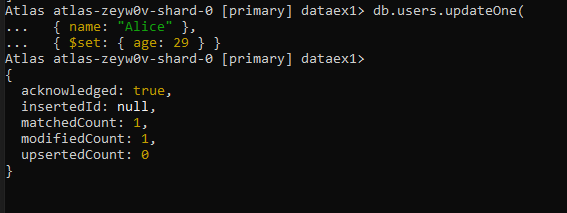
Update a document (e.g., change Alice’s age to 29):

>db.users.updateOne(

{ name: "Alice" },

{ $set: { age: 29 } }

)



Remove a document (e.g., remove Alice):

>db.users.deleteOne({ name: "Alice" })

### **1. Insert One Document at a Time (insertOne())**

db.users.insertOne({

name: "Bob",

age: 32,

email: "bob@example.com"

});

You can run this command multiple times with different data.

### **2. Insert Multiple Documents at Once (insertMany())**

db.users.insertMany([

{ name: "Charlie", age: 25, email: "charlie@example.com" },

{ name: "Diana", age: 30, email: "diana@example.com" },

{ name: "Eve", age: 27, email: "eve@example.com" }

]);

This inserts several documents in a single command, which is faster and cleaner.

### **Example in Mongo Shell or Atlas shell:**

use dataex1

db.users.insertMany([

{ name: "Charlie", age: 25, email: "charlie@example.com" },

{ name: "Diana", age: 30, email: "diana@example.com" },

{ name: "Eve", age: 27, email: "eve@example.com" }

]);

To exit the **MongoDB shell** (the mongo or mongosh prompt), just type:

exit

and press **Enter**.

Alternatively, you can press:

* **Ctrl + C** twice (in some terminals)
* Or **Ctrl + D**

11. MongoDB – Databases, Collections and Records

a. Write MongoDB queries to Create and drop databases and collections.

b. Write MongoDB queries to work with records using find(), limit(), sort(),

createIndex(), aggregate().

a. Write MongoDB queries to Create and drop databases and collections. b. Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

###### 

## **a. Create and Drop Databases and Collections**

### **Create Database**

In MongoDB, a database is created implicitly when you first store data in it.

// Switch to (or create) a database named 'mydb'

use mydb

### **Create Collection**

// Create a collection explicitly

db.createCollection("mycollection")

Note: Collections also get created implicitly when you insert documents into them.

### **Drop Collection**

db.mycollection.drop()

### **Drop Database**

db.dropDatabase()

b. Write MongoDB queries to work with records using find(), limit(), sort(),

createIndex(), aggregate().

## **b. Queries to Work with Records**

### **1. find() — Query documents**

// Find all documents

db.mycollection.find()

// Find documents matching condition

db.mycollection.find({ age: { $gt: 20 } })

### **2. limit() — Limit number of results**

db.mycollection.find().limit(5) // Get first 5 documents

### **3. sort() — Sort results**

// Sort by age ascending

db.mycollection.find().sort({ age: 1 })

// Sort by age descending

db.mycollection.find().sort({ age: -1 })

### **4. createIndex() — Create an index on a field**

db.mycollection.createIndex({ email: 1 }) // Ascending index on email

### **5. aggregate() — Perform aggregation operations**

Example: Group documents by status and count number of documents per status

db.mycollection.aggregate([

{ $group: { \_id: "$status", count: { $sum: 1 } } }

])

Example: Filter documents with age > 20, then sort by age descending, limit to 3

db.mycollection.aggregate([

{ $match: { age: { $gt: 20 } } },

{ $sort: { age: -1 } },

{ $limit: 3 }

])

12. Augmented Programs: (Any 2 must be completed)

a. Design a to-do list application using NodeJS and ExpressJS.

b. Design a Quiz app using ReactJS.

c. Complete the MongoDB certification from MongoDB University website.

## **a. To-Do List Application using NodeJS + ExpressJS**

### **Features**

* Add tasks
* View tasks
* Mark tasks as completed
* Delete tasks

### **Step 1: Setup**

npm init -y

npm install express body-parser cors

### **Step 2: Create server.js**

const express = require('express');

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

const cors = require('cors');

const app = express();

app.use(cors());

app.use(bodyParser.json());

let tasks = []; // In-memory task list (use DB in real apps)

let id = 1;

// Get all tasks

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

res.json(tasks);

});

// Add a new task

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

const task = { id: id++, title: req.body.title, completed: false };

tasks.push(task);

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

});

// Toggle task completed

app.put('/tasks/:id', (req, res) => {

const task = tasks.find(t => t.id == req.params.id);

if (task) {

task.completed = req.body.completed;

res.json(task);

} else {

res.status(404).json({ error: 'Task not found' });

}

});

// Delete task

app.delete('/tasks/:id', (req, res) => {

tasks = tasks.filter(t => t.id != req.params.id);

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

});

const PORT = 3000;

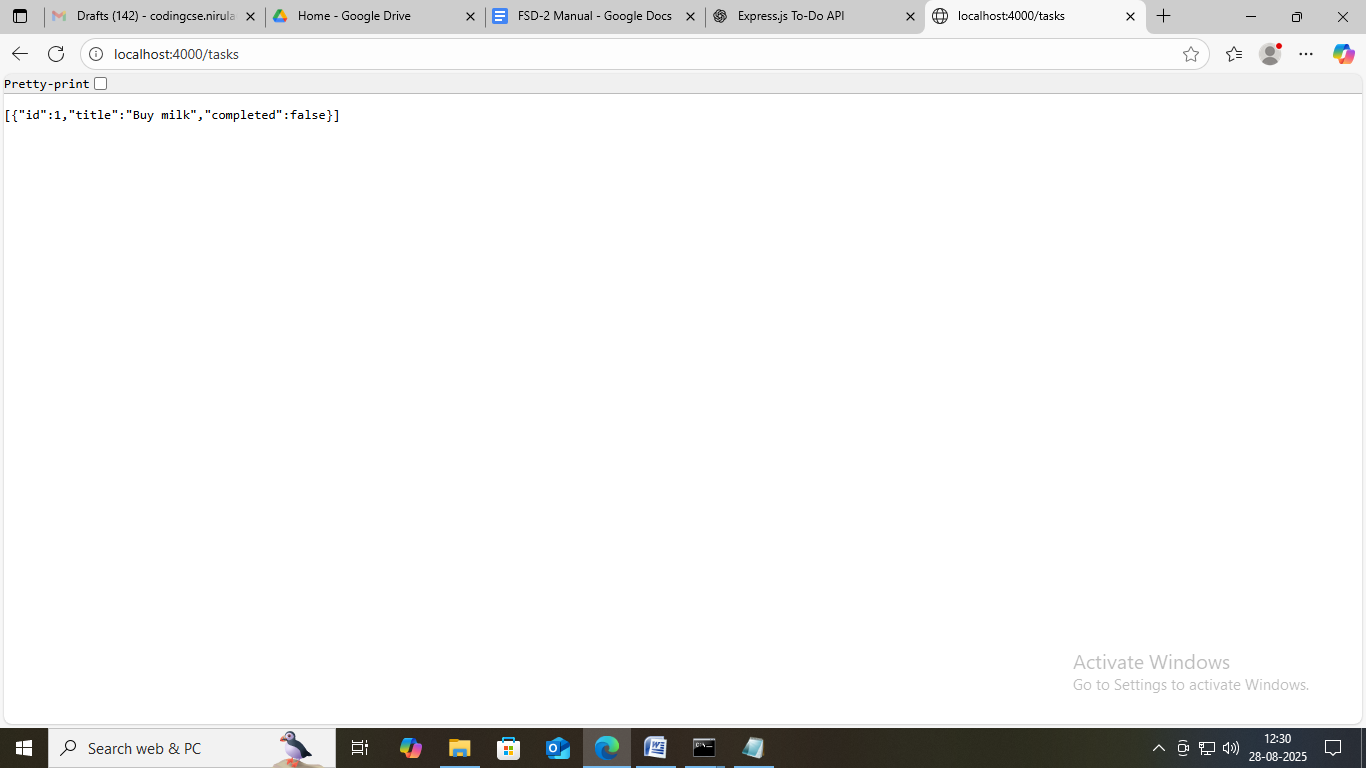
app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));

### **How to run**

node [server.js](http://server.js)

C:\Users\admin\56>curl -X POST http://localhost:4000/tasks -H "Content-Type: application/json" -d "{\"title\":\"Buy milk\"}"

{"id":1,"title":"Buy milk","completed":false}



## **b. Quiz App using ReactJS**

### **Features**

* Display quiz questions with multiple choices
* Select answer and submit
* Show score at the end

### **Step 1: Create React App**

npx create-react-app quiz-app

cd quiz-app

npm start

### **Step 2: Replace src/App.js with:**

import React, { useState } from 'react';

const questions = [

{

question: "What is 2 + 2?",

options: ["3", "4", "5", "6"],

answer: "4"

},

{

question: "What is the capital of France?",

options: ["Berlin", "London", "Paris", "Madrid"],

answer: "Paris"

},

{

question: "Which language runs in a browser?",

options: ["Java", "C", "Python", "JavaScript"],

answer: "JavaScript"

}

];

function App() {

const [current, setCurrent] = useState(0);

const [score, setScore] = useState(0);

const [showScore, setShowScore] = useState(false);

const [selected, setSelected] = useState(null);

const handleAnswer = () => {

if (selected === questions[current].answer) {

setScore(score + 1);

}

setSelected(null);

if (current + 1 < questions.length) {

setCurrent(current + 1);

} else {

setShowScore(true);

}

};

return (

<div style={{ maxWidth: 600, margin: 'auto', padding: 20 }}>

<h1>Quiz App</h1>

{showScore ? (

<div>

<h2>Your Score: {score} / {questions.length}</h2>

<button onClick={() => {

setCurrent(0);

setScore(0);

setShowScore(false);

}}>Restart Quiz</button>

</div>

) : (

<div>

<h2>{questions[current].question}</h2>

<ul style={{ listStyle: 'none', padding: 0 }}>

{questions[current].options.map(option => (

<li key={option} style={{ marginBottom: 10 }}>

<label>

<input

type="radio"

name="answer"

value={option}

checked={selected === option}

onChange={() => setSelected(option)}

/>

{' '}{option}

</label>

</li>

))}

</ul>

<button onClick={handleAnswer} disabled={selected === null}>

Submit Answer

</button>

</div>

)}

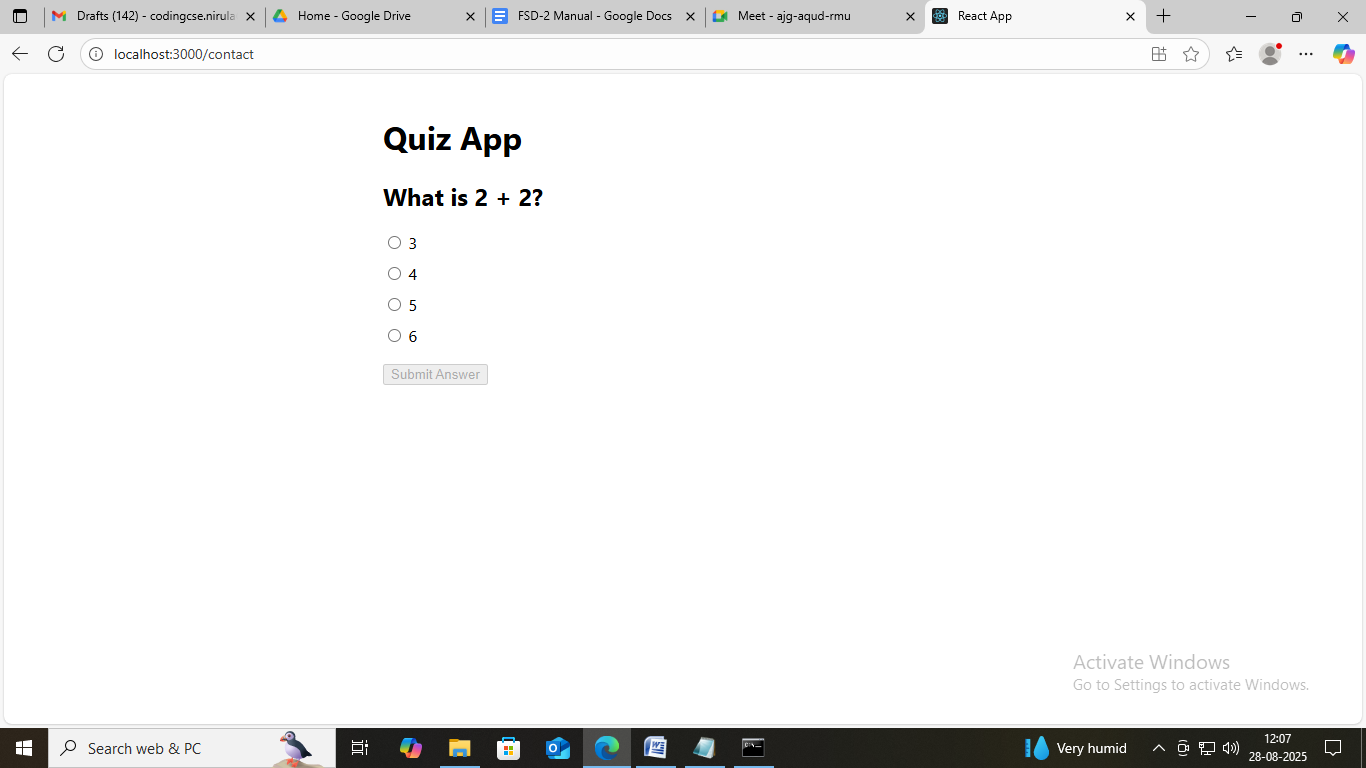
</div>

);

}

export default App;

### **Step 3: Open http://localhost:3000 in browser to try it out!**



Another example for Prakasam Engineering College:

import React, { useState } from 'react';

function App() {

const [studentName, setStudentName] = useState('');

const [regNo, setRegNo] = useState('');

const [answers, setAnswers] = useState({

q1: '',

q2: '',

q3: '',

});

const [submitted, setSubmitted] = useState(false);

const [score, setScore] = useState(0);

const correctAnswers = {

q1: 'a',

q2: 'b',

q3: 'c',

};

const handleAnswerChange = (e) => {

const { name, value } = e.target;

setAnswers((prev) => ({ ...prev, [name]: value }));

};

const handleSubmit = (e) => {

e.preventDefault();

let calculatedScore = 0;

Object.keys(correctAnswers).forEach((key) => {

if (answers[key] === correctAnswers[key]) {

calculatedScore += 1;

}

});

setScore(calculatedScore);

setSubmitted(true);

};

return (

<div>

<h1>VNITSW College Examination</h1>

{!submitted ? (

<form onSubmit={handleSubmit}>

<div>

<label>Student Name: </label>

<input

type="text"

value={studentName}

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

required

/>

</div>

<div>

<label>Registration Number: </label>

<input

type="text"

value={regNo}

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

required

/>

</div>

<div>

<h3>Section 1</h3>

<p>1. What is the capital of France?</p>

<label><input type="radio" name="q1" value="a" onChange={handleAnswerChange} /> Paris</label>

<label><input type="radio" name="q1" value="b" onChange={handleAnswerChange} /> London</label>

<label><input type="radio" name="q1" value="c" onChange={handleAnswerChange} /> Berlin</label>

</div>

<div>

<h3>Section 2</h3>

<p>2. What is 2 + 2?</p>

<label><input type="radio" name="q2" value="a" onChange={handleAnswerChange} /> 3</label>

<label><input type="radio" name="q2" value="b" onChange={handleAnswerChange} /> 4</label>

<label><input type="radio" name="q2" value="c" onChange={handleAnswerChange} /> 5</label>

</div>

<div>

<h3>Section 3</h3>

<p>3. What is H2O?</p>

<label><input type="radio" name="q3" value="a" onChange={handleAnswerChange} /> Oxygen</label>

<label><input type="radio" name="q3" value="b" onChange={handleAnswerChange} /> Hydrogen</label>

<label><input type="radio" name="q3" value="c" onChange={handleAnswerChange} /> Water</label>

</div>

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

</form>

) : (

<div>

<h2>Result</h2>

<p>Student Name: {studentName}</p>

<p>Registration No: {regNo}</p>

<p>Total Score: {score} / 3</p>

</div>

)}

</div>

);

}

export default App;