* What is Redux?

Redux is a popular open-source JavaScript library for managing application state. It's particularly useful for complex applications where keeping track of state changes can become cumbersome.

* What is Redux Thunk used for?

Redux Thunk is a middleware specifically designed for Redux. It tackles a key limitation of Redux: handling asynchronous operations like fetching data from an API.

Redux Thunk is a popular middleware for React Redux. You can create services that return a function hence the name Thunk. The creators of Thunk call this function ‘action creators’.

* What is Pure Component? When to use Pure Component over Component?

In React applications, components are responsible for rendering UI elements based on their props (data passed down from parent components) and state (internal data managed by the component). When a component's props or state change, React typically re-renders the component and its children to reflect the updates.

Pure Components are a special type of class-based component in React that optimizes performance by avoiding unnecessary re-renders. They achieve this by briefly comparing their props and state during the update phase.

**When to Use Pure Components:**

* **Components with Expensive Renders:** If a component has a complex render function or performs expensive (time-consuming) calculations, using a Pure Component can significantly improve performance by re-rendering only when necessary.
* **Components Receiving Frequently Updated Props:** If a component receives props from a parent component that updates frequently, even if the props themselves haven't changed, a Pure Component can prevent unnecessary re-renders.
* What is the second argument that can optionally be passed tosetState and what is its purpose?

The second argument that can optionally be passed to setState in React is a **callback function**. This callback function gets executed **after** the state update has been completed and the component has been re-rendered.

Here's a breakdown of its purpose:

**Ensuring Code Execution After State Update:**

* setState is asynchronous in React. This means the state update might not happen immediately after you call it.
* If you rely on the updated state value in your code, using the callback function guarantees that your code runs only after the state has been truly updated.

**Common Use Cases:**

* **Performing Side Effects:** If your state update involves side effects, like making an API call or manipulating the DOM directly, using the callback function ensures these actions happen after the state is truly reflected in the component.
* **Accessing Updated State Values:** If you need to access the **updated** state value within your code, using the callback function provides you with the most recent state after the update.

**Example:**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

handleClick = () => {

this.setState({ count: this.state.count + 1 }, () => {

console.log("State updated:", this.state.count);

});

}

render() {

return (

<div>

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

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

</div>

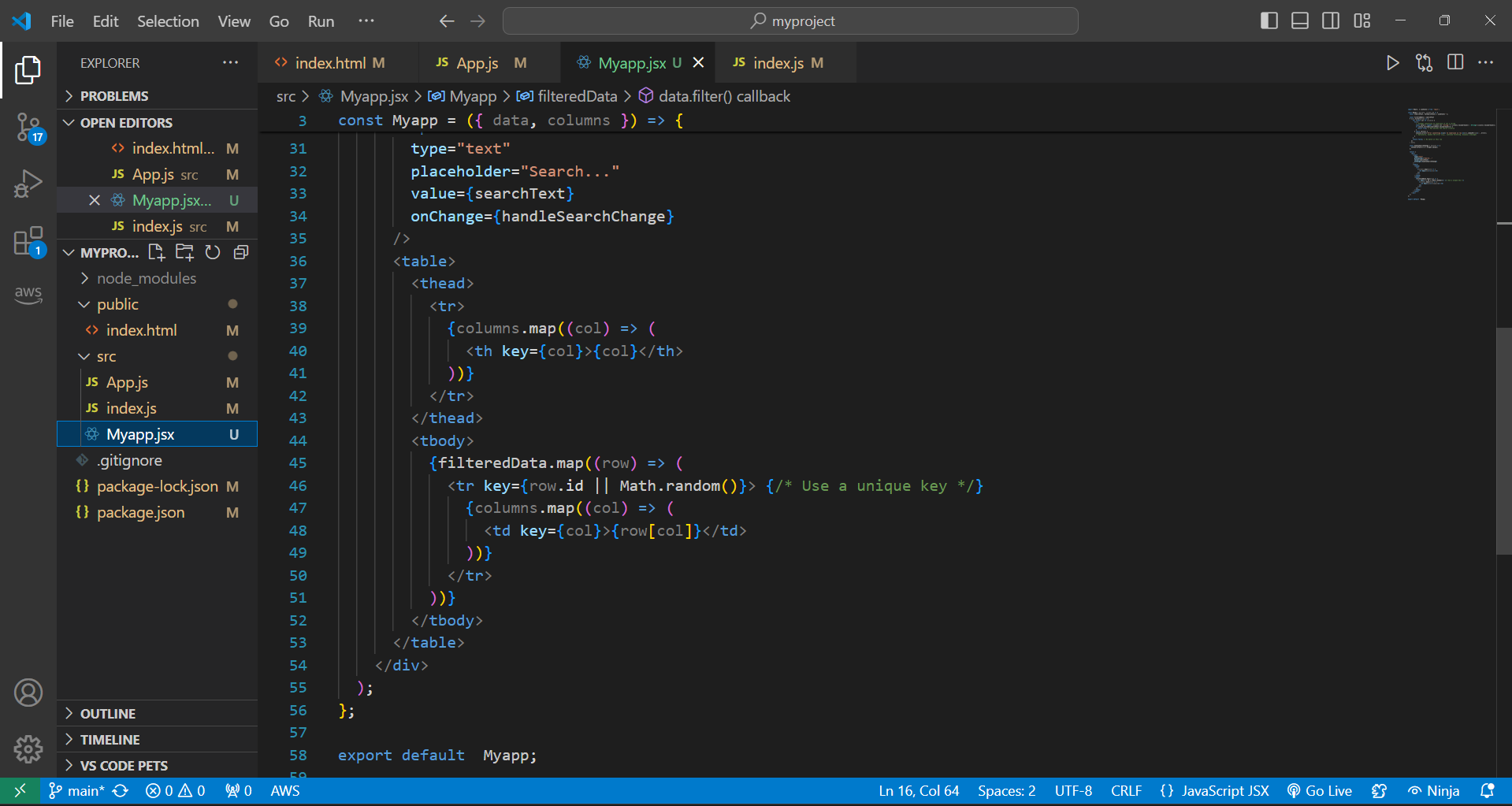
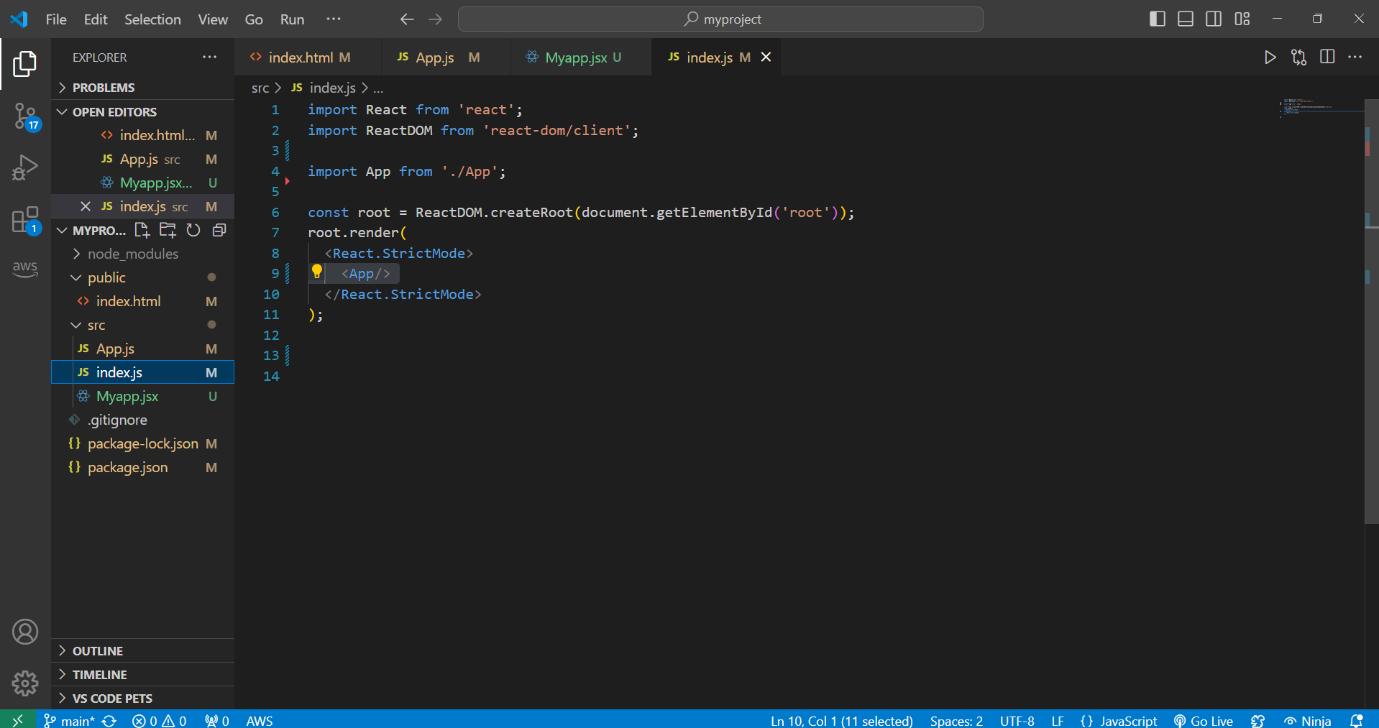
);

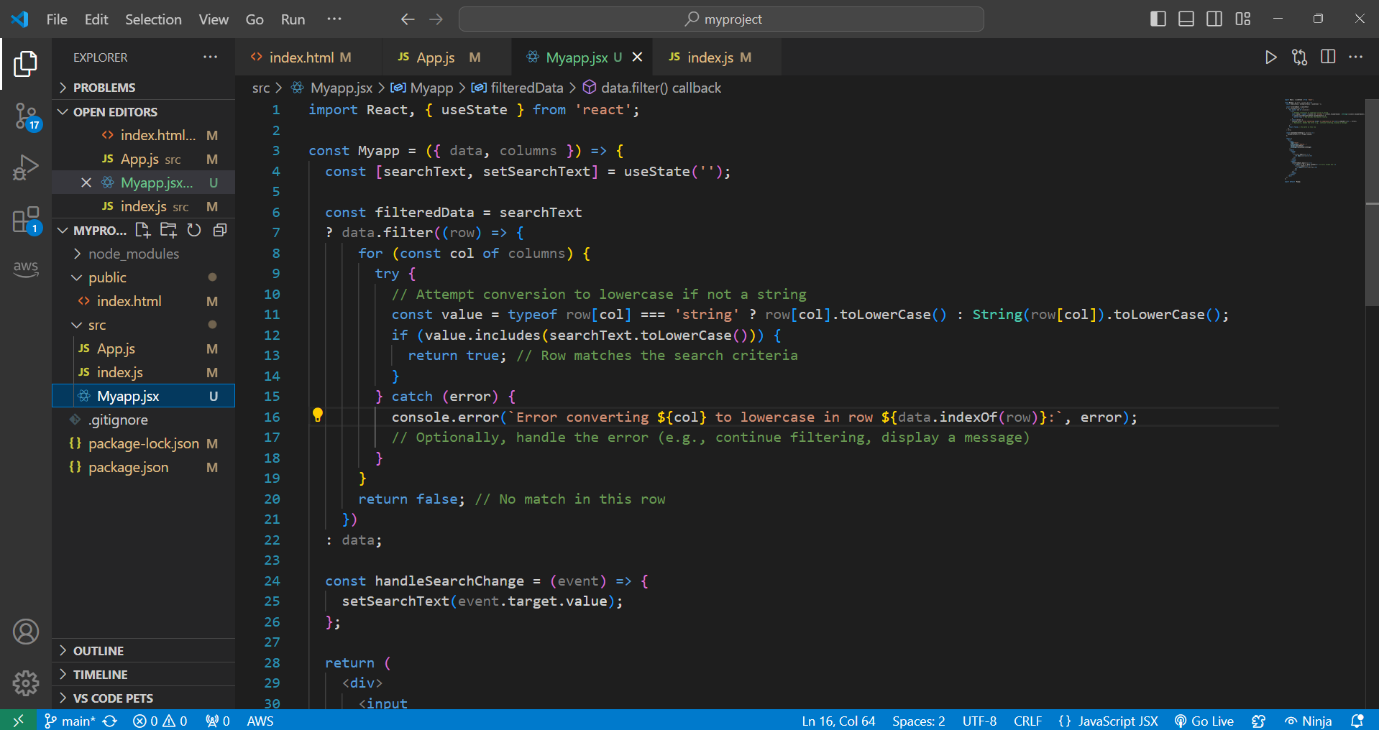
}

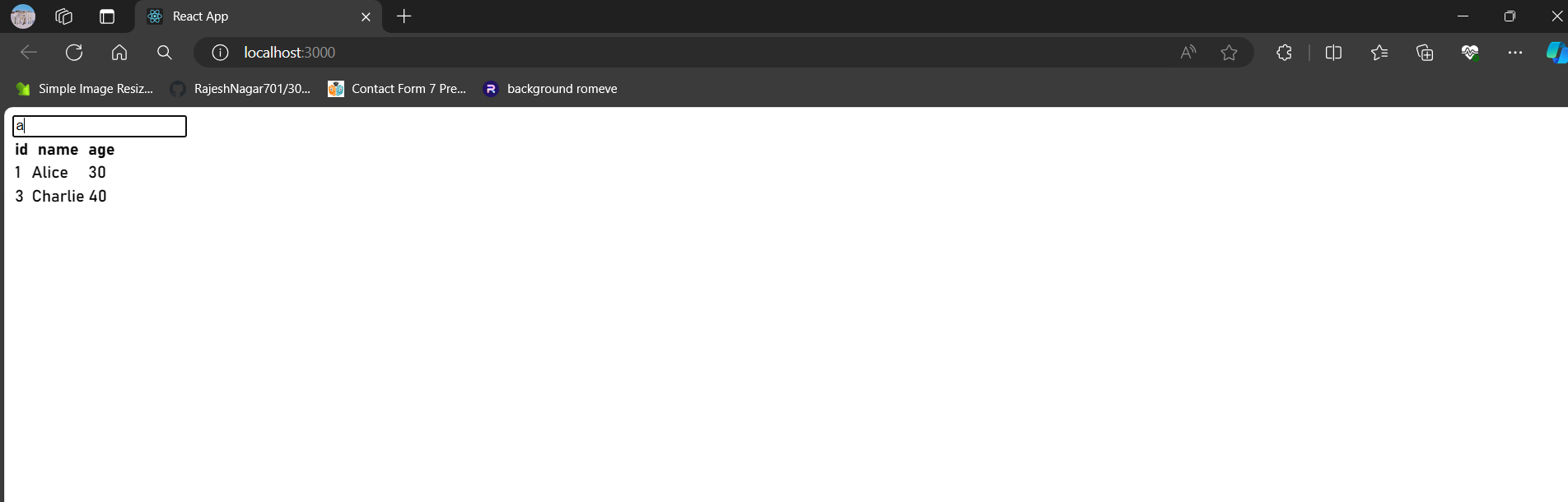
}

In this example, the callback function ensures that the console.log statement prints the **updated** value of the count after the state has been incremented.

* Create a Table and Search data from a table using React Js.







● Create Login registration with the CRUD Application using API (Redux)

Code login:

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

import Header2 from '../component/Header'

import Footer from '../component/Footer'

import { Link, useNavigate } from 'react-router-dom'

import { toast } from 'react-toastify'

import axios from 'axios'

import Header from '../component/Header'

function Logina() {

const redirect=useNavigate(); // redirect any routes

useEffect(()=>{

if(localStorage.getItem('idadmin'))

{

redirect('/dashboard');

}

},[]);

const [formvalue, setFormvalue] = useState({

email: "",

password: "",

});

const getform = (e) => {

setFormvalue({ ...formvalue, [e.target.name]: e.target.value });

console.log(formvalue);

}

const validation = () => {

var result = true;

if (formvalue.email == "") {

toast.error('Email Field is required');

result = false;

return false;

}

if (formvalue.password == "") {

toast.error('Password Field is required');

result = false;

return false;

}

return result;

}

const submithandel = async (e) => {

e.preventDefault(); // stop page reload

if (validation()) {

const res = await axios.get(`http://localhost:3000/admin?email=${formvalue.email}`);

console.log(res);

if (res.data.length > 0) {

if (res.data[0].password == formvalue.password) {

// create session

localStorage.setItem('idadmin',res.data[0].id);

localStorage.setItem('nameadmin',res.data[0].name);

toast.success('Login Success');

redirect('/dashboard');

}

else {

toast.error('Password incorrect');

return false;

}

}

else {

toast.error('Email does not Exist');

return false;

}

}

}

return (

<div className="container-xxl bg-white p-0">

<Header/>

{/\* Contact Start \*/}

<div className="container-xxl py-5">

<div className="container py-5 px-lg-5">

<div className="wow fadeInUp" data-wow-delay="0.1s">

<p className="section-title text-secondary justify-content-center"><span />Login Us<span /></p>

<h1 className="text-center mb-5">Login For Services</h1>

</div>

<div className="row justify-content-center">

<div className="col-lg-7">

<div className="wow fadeInUp" data-wow-delay="0.3s">

<form action="" method="post" onSubmit={submithandel}>

<div className="row g-3">

<div className="col-md-6">

<div className="form-floating">

<input type="email" className="form-control" name="email" value={formvalue.email} onChange={getform} id="email" placeholder="Your Email" />

<label htmlFor="email">Your Email</label>

</div>

</div>

<div className="col-md-6">

<div className="form-floating">

<input type="password" className="form-control" name="password" value={formvalue.password} onChange={getform} id="email" placeholder="Your Email" />

<label htmlFor="email">Your Password</label>

</div>

</div>

<div className="col-12">

<button className="btn btn-primary w-100 py-3" type="submit">Login</button>

<Link to="/signpage">If you not Registered then Register Here</Link>

</div>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

{/\* Contact End \*/}

<Footer />

</div>

)

}

export default Logina

**Login, Registration, and CRUD with Redux and API**

Here's a breakdown of creating a Login, Registration, and CRUD application using Redux and API:

**1. Project Setup:**

* Create a React project using create-react-app:

Bash

npx create-react-app my-crud-app

cd my-crud-app

Use code [with caution.](/faq#coding)

content\_copy

* Install necessary dependencies:

Bash

npm install axios redux react-redux redux-thunk

Use code [with caution.](/faq#coding)

content\_copy

**2. Redux Store Setup:**

* Create a store.js file to manage application state:

import { createStore, applyMiddleware } from 'redux'; import thunk from 'redux-thunk'; import rootReducer from './reducers'; // Import your combined reducers

const store = createStore(rootReducer, applyMiddleware(thunk));

export default store;

\*\*3. Reducers:\*\*

- Create separate reducers for user authentication (`auth`), data (`data`), and any other functionalities in your application. These reducers will handle state changes based on dispatched actions.

\*\*4. API Calls:\*\*

- Create a service file (`api.js`) to handle API calls using `axios`:

```javascript

import axios from 'axios';

const baseUrl = 'http://your-api-endpoint'; // Replace with your API URL

export const login = async (credentials) => {

const response = await axios.post(`${baseUrl}/login`, credentials);

return response.data;

};

export const register = async (userData) => {

const response = await axios.post(`${baseUrl}/register`, userData);

return response.data;

};

// Add functions for CRUD operations here (GET, POST, PUT, DELETE)

**5. Login and Registration Components:**

* Create components (Login.jsx and Register.jsx) for user login and registration forms.
* Dispatch actions to trigger API calls (login/register) using useDispatch hook and handle responses to update user state and authentication status.

**6. CRUD Components:**

* Create components for creating, reading, updating, and deleting data (AddData.jsx, DataList.jsx, etc.).
* Dispatch actions to fetch data from the API using useEffect hook and handle responses to update the data state.
* Use forms and dispatch actions to create, update, and delete data based on user interaction.

**7. Connecting Components to Redux:**

* Use connect from react-redux to connect components to the Redux store.
* Access state and dispatch actions using mapStateToProps and mapDispatchToProps.

**8. Authentication:**

* Store authentication information (e.g., token) in the Redux store or local storage.
* Include authorization headers in API calls when needed (e.g., Authorization: Bearer <token>).

Implement protected routes that require user authentication to access specific functionalities.

**Note:** This is a high-level overview. Implementing specific functionalities will involve writing more code for forms, data handling, UI elements, and error handling.