**Lesson 08 Demo 04**

**Creating a React Toolkit Slice**

**Objective:** To create a React toolkit slice application following Redux best practices using the Redux Toolkit, interacting with static JSON files for data handling, and addressing errors

**Tools required:** Node.js and React.js

**Prerequisites:** None

Steps to be followed:

1. Create and set up the React project
2. Create a features/posts, store, and components folder
3. Modify the App.js file
4. Modify the index.js file
5. Test the application

**Step 1: Create and set up the React project**

* 1. Open a terminal window and run the following command to create a React application:

**npx create-react-app react-redux-slice-best-practice**

**A screenshot of a computer

Description automatically generated**

* 1. Open the created React application folder (**react-redux-slice-best-practice**) in VS Code by clicking on **File** in the top left corner and selecting **Open Folder**

**A screenshot of a computer

Description automatically generated**

* 1. Click on the **Open** button

A screenshot of a computer

Description automatically generated

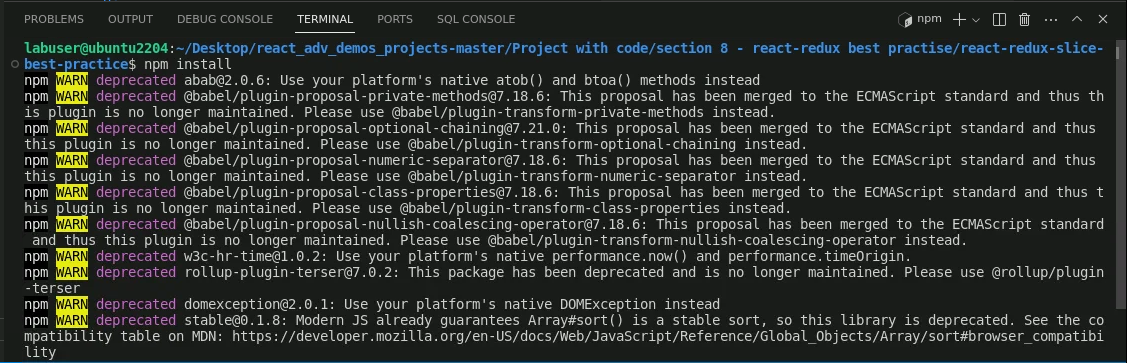
The folder structure appears as follows:

A screenshot of a computer program

Description automatically generated

* 1. Inside the project, open the **TERMINAL** and run the following command to install the required dependencies:

**npm install**



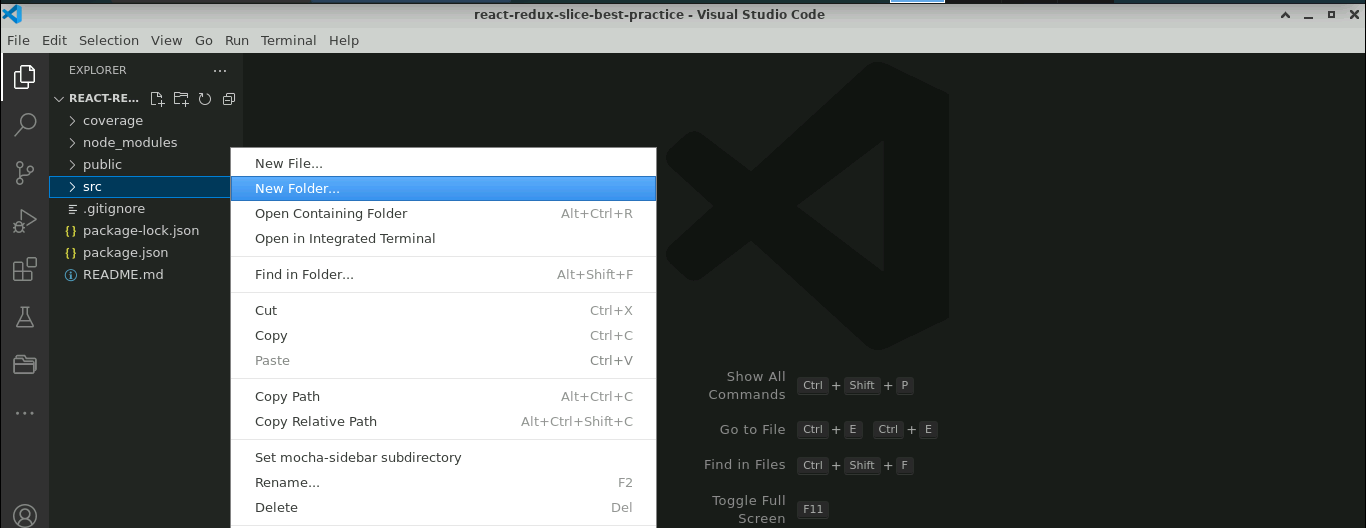
* 1. Open the **package.json** file and view the external dependencies

**A screen shot of a computer

Description automatically generated**

**Step 2: Create a features/posts, store, and components folder**

1. Right-click on the **src** folder and select **New Folder**

****

1. Create three folders named **features/posts**, **store**, and **Components**

**A screenshot of a computer

Description automatically generated**

1. Inside the **features/posts** folder, create a file named **postsSlice.js** and enter the following code:

**import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'**

**import axios from 'axios'**

**const url = 'https://jsonplaceholder.typicode.com/posts'**

**const initialState = {**

**postItems: [],**

**status: 'idle',**

**error: null,**

**}**

**// Get all the posts from the API**

**export const getPosts = createAsyncThunk('posts/getPosts', async (thunkAPI) => {**

**try {**

**const res = await axios.get(url)**

**return res.data**

**} catch (err) {**

**// console.error(err.message)**

**return thunkAPI.rejectWithValue({ error: err.message })**

**}**

**})**

**// Handle POST request to create a new post**

**export const addPost = createAsyncThunk(**

**// The name of the action**

**'posts/addPost',**

**// The payload creator**

**async (initialPost, thunkAPI) => {**

**try {**

**const res = await axios.post(url, initialPost)**

**return res.data**

**} catch (err) {**

**return thunkAPI.rejectWithValue({ error: err.message })**

**}**

**}**

**)**

**const postsSlice = createSlice({**

**/\* The name of the slice[this will also be used as the action type string**

**in combination with the extraReducer name i.e posts/getPosts or posts/addPost]**

**\*/**

**name: 'posts',**

**// initialState: initialState[ES6 destructuring syntax]**

**initialState,**

**// Add reducers for the synchronous actions on the UI**

**reducers: {},**

**// Add extra reducers for the asynchronous actions on the UI**

**extraReducers:(builder)=> {**

**builder.**

**addCase(getPosts.pending, (state, action) => {**

**// When data is being fetched**

**state.status = 'loading'**

**}).**

**addCase(getPosts.fulfilled, (state, action) => {**

**// When data is fetched successfully**

**state.status = 'successful'**

**state.postItems = state.postItems.concat(action.payload)**

**}).**

**addCase(getPosts.rejected, (state, action) => {**

**// When data is fetched unsuccessfully**

**state.status = 'failed'**

**state.error = action.error.message**

**console.error(state.error)**

**}).**

**addCase(addPost.fulfilled, (state, action) => {**

**// Add the new post created on the UI to the existing posts**

**state.postItems.push(action.payload)**

**})**

**}**

**})**

**// Export the reducer logic from the slice**

**export default postsSlice.reducer**

**A screenshot of a computer program

Description automatically generated**

1. Inside the **store** folder, create a file named **store.js** and enter the following code:

**// Pull in configureStore API**

**import { configureStore } from '@reduxjs/toolkit';**

**// Pull in the postsSlice reducer and rename it to postsReducer**

**import postsReducer from '../features/posts/postsSlice';**

**// Create the Redux store and pass in the postsReducer as the initial data**

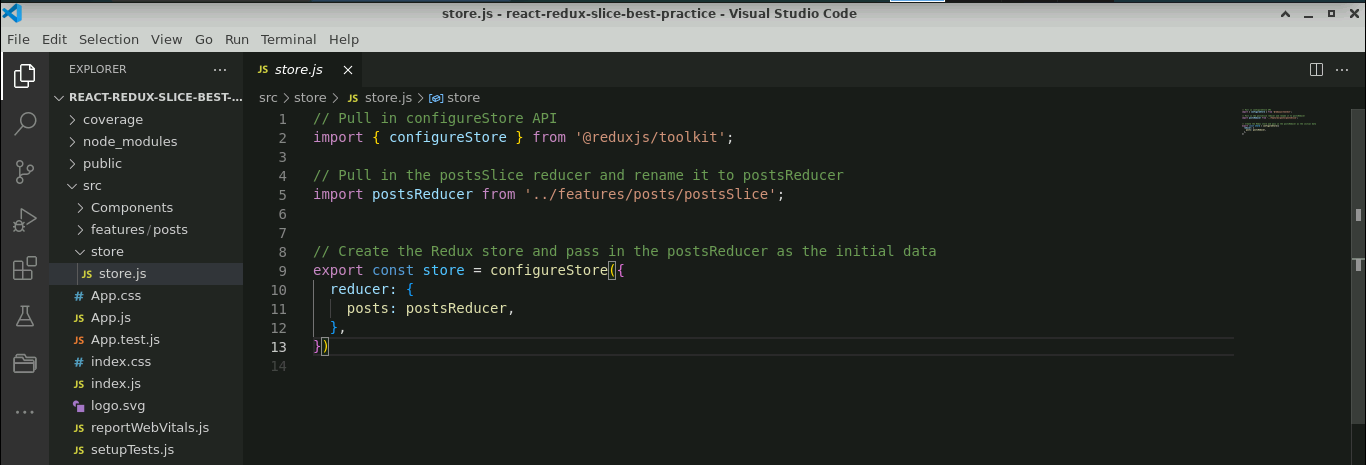
**export const store = configureStore({**

**reducer: {**

**posts: postsReducer,**

**},**

**})**

****

* 1. Inside the **Components** folder, create a file named **CreatePost.js** and enter the following code:

**import { useState } from 'react'**

**import { useDispatch } from 'react-redux'**

**import { useNavigate } from 'react-router-dom'**

**import { addPost } from '../features/posts/postsSlice'**

**const CreatePost = () => {**

**// Set the initial state for the form**

**const [title, setTitle] = useState('')**

**const [body, setBody] = useState('')**

**const [addPostRequestStatus, setAddPostRequestStatus] = useState('idle')**

**// Get the dispatch function**

**const dispatch = useDispatch()**

**// Get the navigate function [replace the history.push() method]**

**const navigate = useNavigate()**

**// Handle form field value changes**

**const onTitleChange = (e) => setTitle(e.target.value)**

**const onBodyChange = (e) => setBody(e.target.value)**

**/\***

**Get the Boolean value based on whether the form is empty or not && the post request status.**

**We use the Boolean value returned to toggle the disbale status submit button**

**\*/**

**const canSavePost =**

**[title, body].every(Boolean) && addPostRequestStatus === 'idle'**

**// Handle form submission**

**const handleAddPost = async (e) => {**

**e.preventDefault()**

**const post = { title, body }**

**if (canSavePost) {**

**try {**

**setAddPostRequestStatus('pending')**

**await dispatch(addPost(post)).unwrap()**

**setTitle('')**

**setBody('')**

**navigate('/')**

**} catch (err) {**

**console.error('Unable to create post:', err)**

**} finally {**

**setAddPostRequestStatus('idle')**

**}**

**}**

**}**

**return (**

**<div className="container">**

**<div className="row">**

**<div className='col-4'>**

**<h3>Create Post</h3>**

**</div>**

**</div>**

**<div className="row">**

**<h5>Add New Post</h5>**

**<form onSubmit={handleAddPost}>**

**<div className="form-group">**

**<label htmlFor="title">Title</label>**

**<input**

**type="text"**

**id="title"**

**name="title"**

**onChange={onTitleChange}**

**value={title}**

**className='form-control'**

**/>**

**<br/>**

**<label>Content</label>**

**<br/>**

**<textarea**

**id="bodyContent"**

**name="bodyContent"**

**cols="28"**

**rows="5"**

**onChange={onBodyChange}**

**value={body}**

**className='form-control'**

**/>**

**<br/>**

**<button type="submit" className="btn btn-danger" disabled={!canSavePost}>**

**Post**

**</button>**

**</div>**

**</form>**

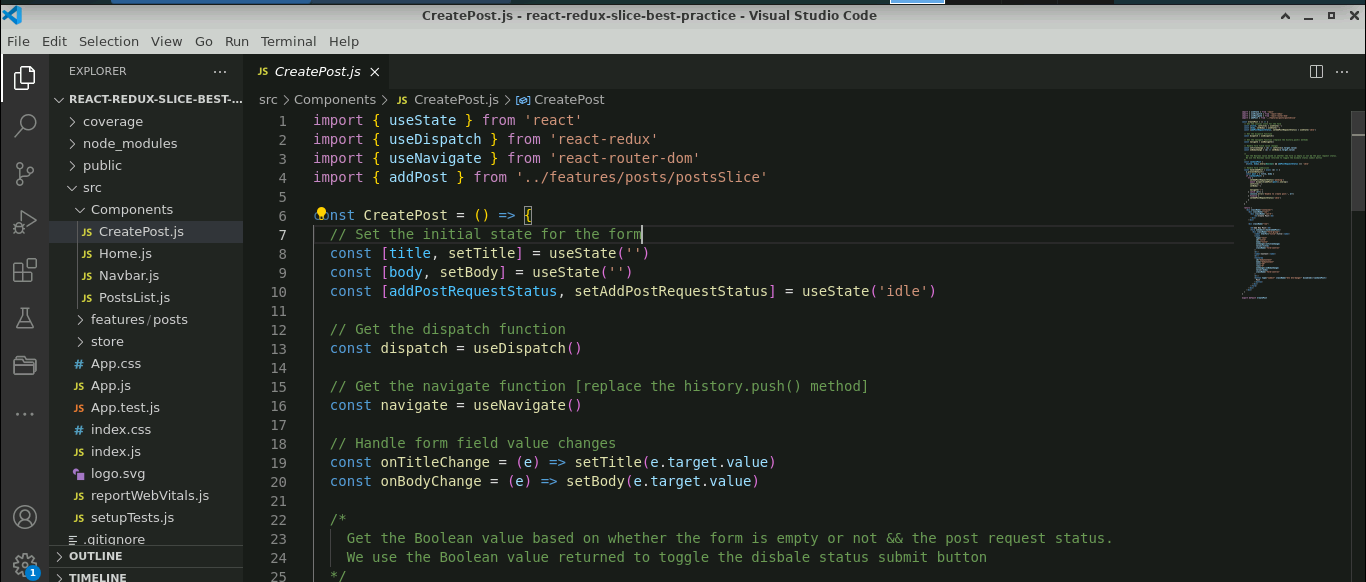
**</div>**

**</div>**

**)**

**}**

**export default CreatePost**

****

* 1. Inside the **Components** folder, create a file named **Home.js** and enter the following code:

**import Navbar from './Navbar'**

**import PostsList from './PostsList'**

**const Home = () => {**

**return (**

**<>**

**<Navbar />**

**<main>**

**<h1>Posts</h1>**

**<PostsList />**

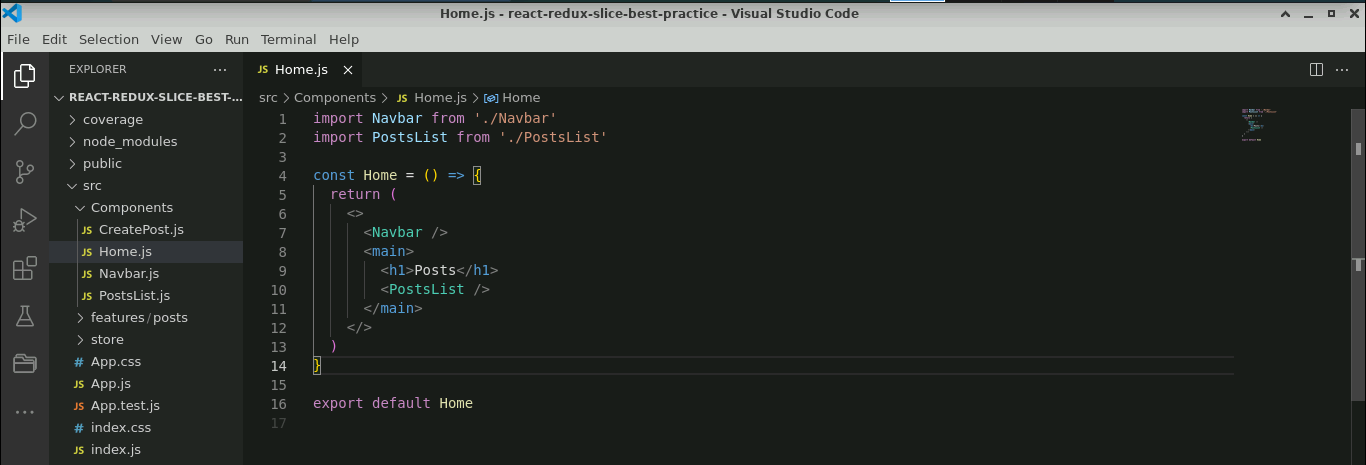
**</main>**

**</>**

**)**

**}**

**export default Home**

****

* 1. Inside the **Components** folder, create a file named **Navbar.js** and enter the following:

**import { Link } from 'react-router-dom'**

**const Navbar = () => {**

**return (**

**<nav>**

**<div>**

**<ul style={{"listStyle":"none"}}>**

**<li>**

**<Link**

**to="/"**

**style={{**

**textTransform: 'uppercase',**

**fontSize: '2rem',**

**fontWeight: 700,**

**}}**

**>**

**React with Redux toolkit with best practise**

**</Link>**

**</li>**

**<li>**

**<Link to="/create" className="create-btn">**

**Create Post**

**</Link>**

**</li>**

**</ul>**

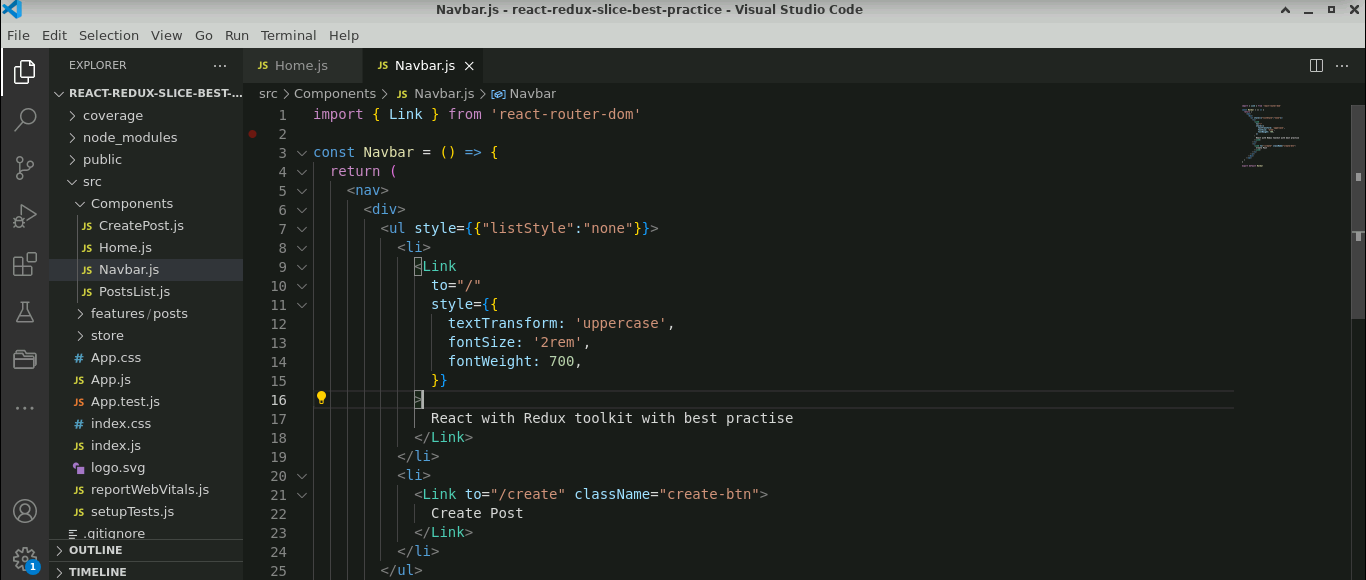
**</div>**

**</nav>**

**)**

**}**

**export default Navbar**

****

* 1. Inside the **Components** folder, create a file named **PostList.js** and enter the following code:

**import { useEffect } from 'react'**

**import { useDispatch, useSelector } from 'react-redux'**

**import { getPosts } from '../features/posts/postsSlice'**

**const PostsList = () => {**

**const dispatch = useDispatch()**

**// Get the posts from the store**

**const posts = useSelector((state) => state.posts)**

**// Pull the post properties**

**const { postItems, status, error } = posts**

**useEffect(() => {**

**// eslint-disable-next-line no-unused-vars**

**let isMounted = true**

**// If status is 'idle', then fetch the posts data from the API**

**if (status === 'idle') {**

**dispatch(getPosts())**

**}**

**// Cleanup function**

**return () => {**

**isMounted = false**

**}**

**// eslint-disable-next-line react-hooks/exhaustive-deps**

**}, [status, dispatch])**

**let bodyContent**

**if (status === 'loading') {**

**bodyContent = <div className="loader"></div>**

**} else if (status === 'successful') {**

**// Sort the posts by id in descending order**

**const sortedPosts = postItems.slice().sort((a, b) => b.id - a.id)**

**// Map through the sorted posts and display them**

**bodyContent = sortedPosts.map((post,index) => (**

**<div key={index}>**

**<h3>{post.title}</h3>**

**<p>{post.body}</p>**

**</div>**

**))**

**} else {**

**// Display the error message**

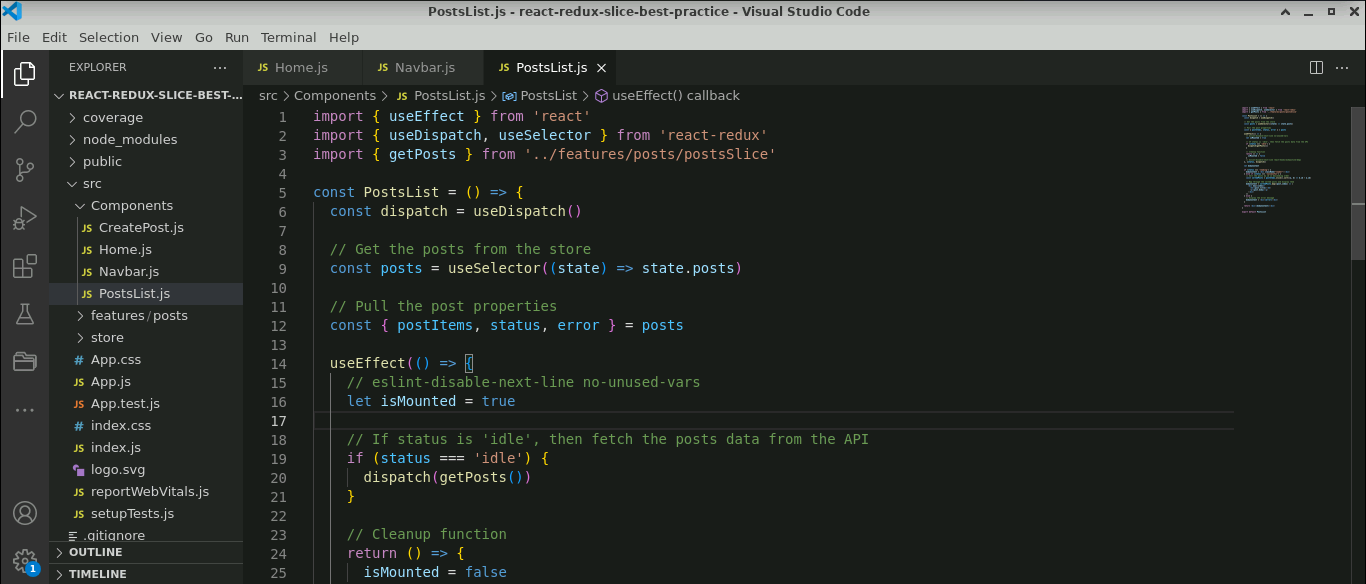
**bodyContent = <div>{error}</div>**

**}**

**return <div>{bodyContent}</div>**

**}**

**export default PostsList**

****

**Step 3: Modify the App.js file**

1. In the **App.js** file, add the routing details as shown below:

**import { Route, Routes } from 'react-router-dom'**

**import CreatePost from './Components/CreatePost'**

**import Home from './Components/Home'**

**function App() {**

**return (**

**<div className="container">**

**<Routes>**

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

**<Route exact path="/create" element={<CreatePost />} />**

**</Routes>**

**</div>**

**)**

**}**

**export default App**

**A screenshot of a computer

Description automatically generated**

**Step 4: Modify the index.js file**

1. Inside the **index.js** file, add the following configuration details of Redux:

**import React from 'react';**

**import ReactDOM from 'react-dom/client';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**import { BrowserRouter as Router } from 'react-router-dom'**

**import { store } from './store/store'**

**import { Provider } from 'react-redux'**

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

**root.render(**

**<React.StrictMode>**

**<Provider store={store}>**

**<Router>**

**<App />**

**</Router>**

**</Provider>**

**</React.StrictMode>**

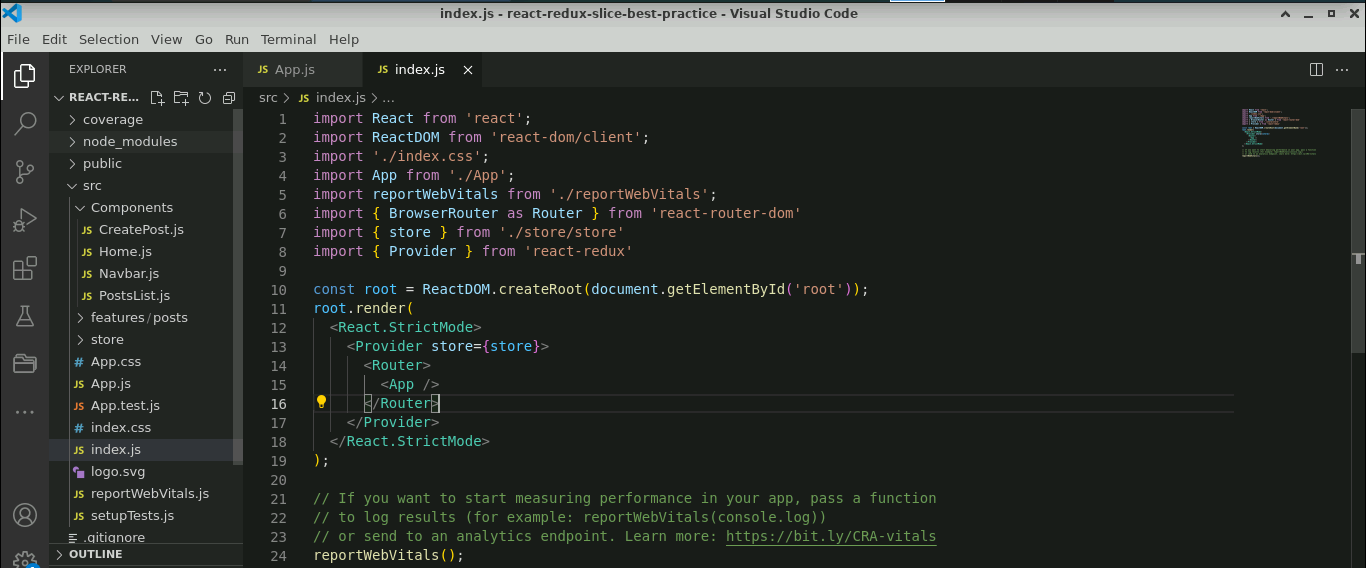
**);**

**// If you want to start measuring performance in your app, pass a function**

**// to log results (for example: reportWebVitals(console.log))**

**// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals**

**reportWebVitals();**

****

**Step 5: Test the application**

* 1. Open the terminal and run the command shown below to execute the application:

**npm start**

**A screen shot of a computer

Description automatically generated**

The output appears as shown below:

A screenshot of a computer

Description automatically generated

* 1. Click on **Create Post** (refer to the above screenshot)

A screenshot of a computer

Description automatically generated

* 1. Under the **Add New Post** page, enter the **Title** and **Content** and click on the **Post** as shown below:

A screenshot of a computer

Description automatically generated

The output appears as shown below:

A screenshot of a computer

Description automatically generated

You have successfully created a React toolkit slice application following Redux best practices using the Toolkit, where a slice is employed to manage posts, interact with static JSON files for data handling, and address errors.