**Lesson 08 Demo 02**

**react-toolkit-slice-best-practise**

**Objective:** To demonstrate the react with redux interact with static json file with proper way to handle data and handling error.

**Tools required:** Node JS and React JS

**Prerequisites:** HTML, CSS, JavaScript ES5/ES6, Basic React Concept

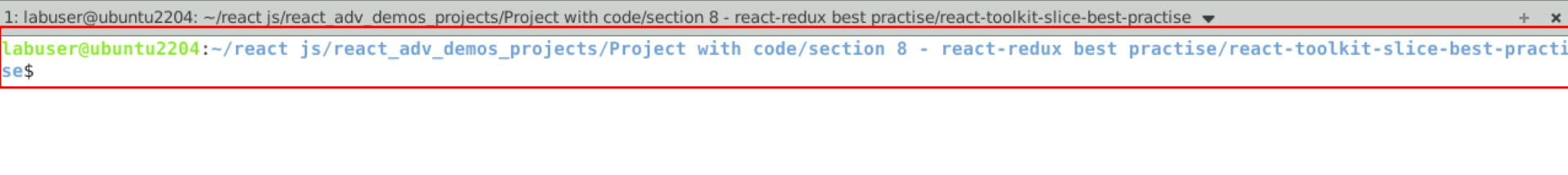
**Note** : All react js project already created with version 18.x with Sample App.js file

**Steps to be followed:**

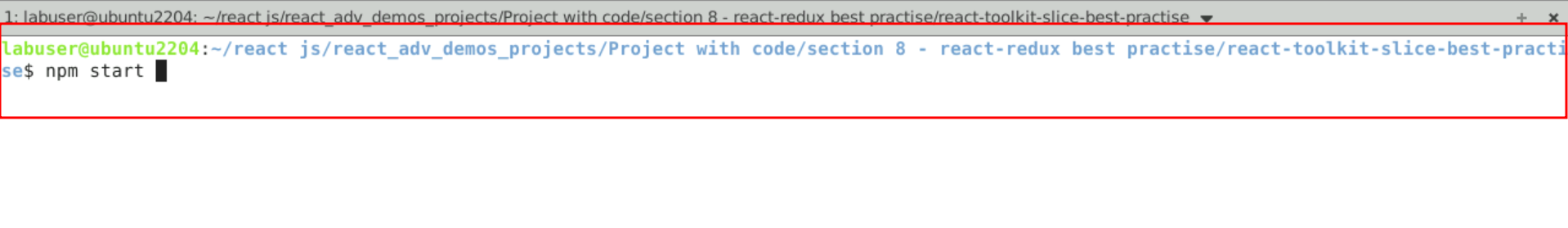
1. Set up for react js project
2. Create features folder which contains postSlice.js file.
3. Create store folder which contains store.js file
4. Now create Component folder and inside this folder create which contains CreatePost.js, Home.js, Navbar.js and PostList.js file
5. Create reducers folder which contains employeeReducer.js file
6. In index.js file provide the store configuration details.
7. Now we run the application using npm start

**Step 1: Set up for react js project**

1. Open a terminal window inside a React JS pre-created project ie **react-toolkit-slice-best-practise**

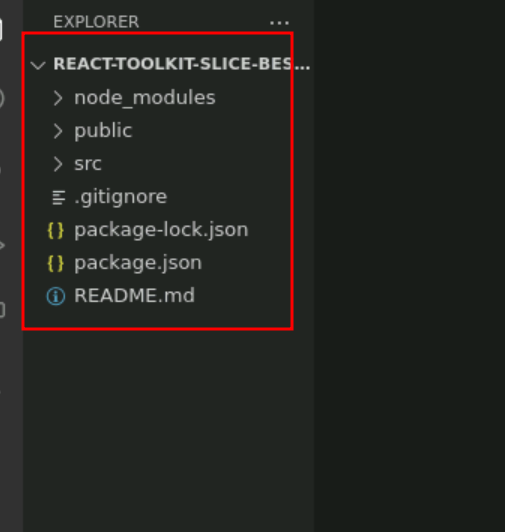


1. Now you need to run the command as **npm install.** This command helps us to installed all required dependencies mention in package.json file in local machine in the form of node\_module folder.

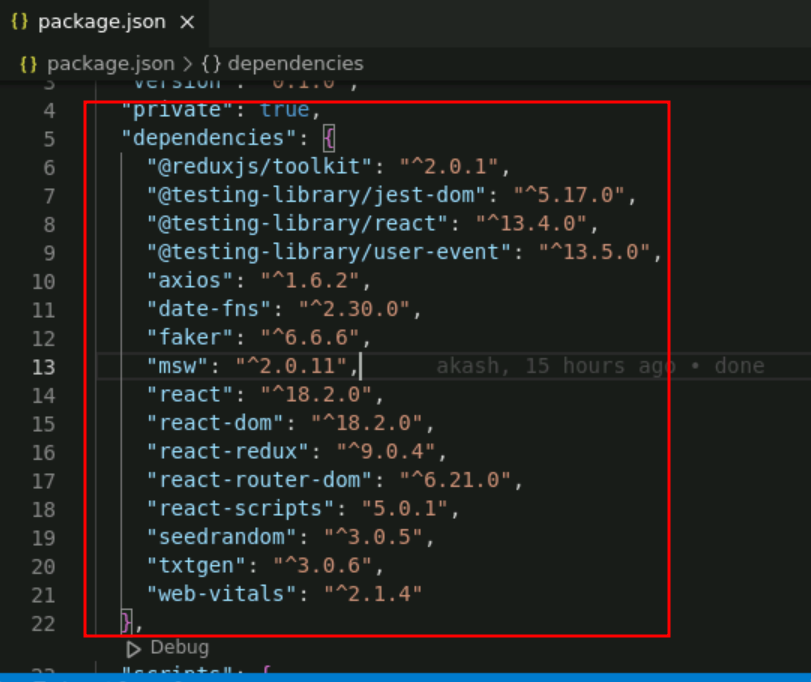


1. Now open **react-redux-slice-best-practise** folder in VS Code Editor

Note: short cut to open write **code .**



1.4 now open package.json file and view external dependencies.

****

**Step 2: Create features folder which contains postSlice.js file**

2.1 postSclie.js file which contains action, reducer, createAsyncThnk 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

**Step 3 : Create store folder which contains store.js file**

3.1 store.js. This file contains store details with the help of reducer.

**store.js**

// 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,

},

})

**Step 4:** Now create Component folder and inside this folder create which contains CreatePost.js, Home.js, Navbar.js and PostList.js file

4.1 CreateStore.js this file is responsible to store the post with title and description on redux store.

**CreateStore.js**

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

4.2 Now create PostList.js file which is responsible to display all post details which interact with fake rest api.

**PostList.js**

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

4.3 NavBar.js component is responsible to display CreatePost.js components.

**NavBar.js**

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

4.4 Home.js component is responsible to display Navbar.js and PostList.js component

**Home.js**

import Navbar from './Navbar'

import PostsList from './PostsList'

const Home = () => {

return (

<>

<Navbar />

<main>

<h1>Posts</h1>

<PostsList />

</main>

</>

)

}

export default Home

**4.5** In App.js file provide the routing details about the CreatePost.js and PostList.js file details

**App.js**

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

**Step 5: In index.js file provide the store configuration details.**

5.1 index.js contain all configuration details.

index.js

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

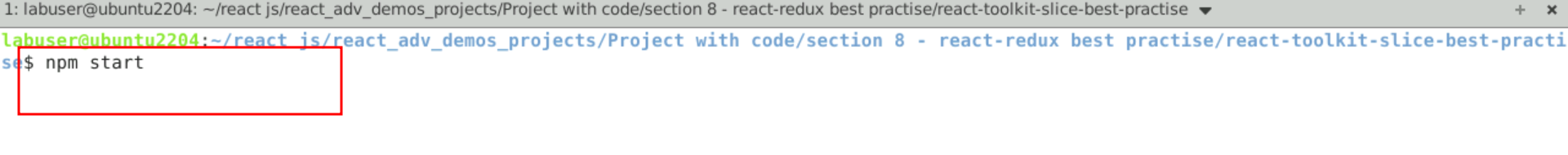
// to log results (for example: reportWebVitals(console.log))

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

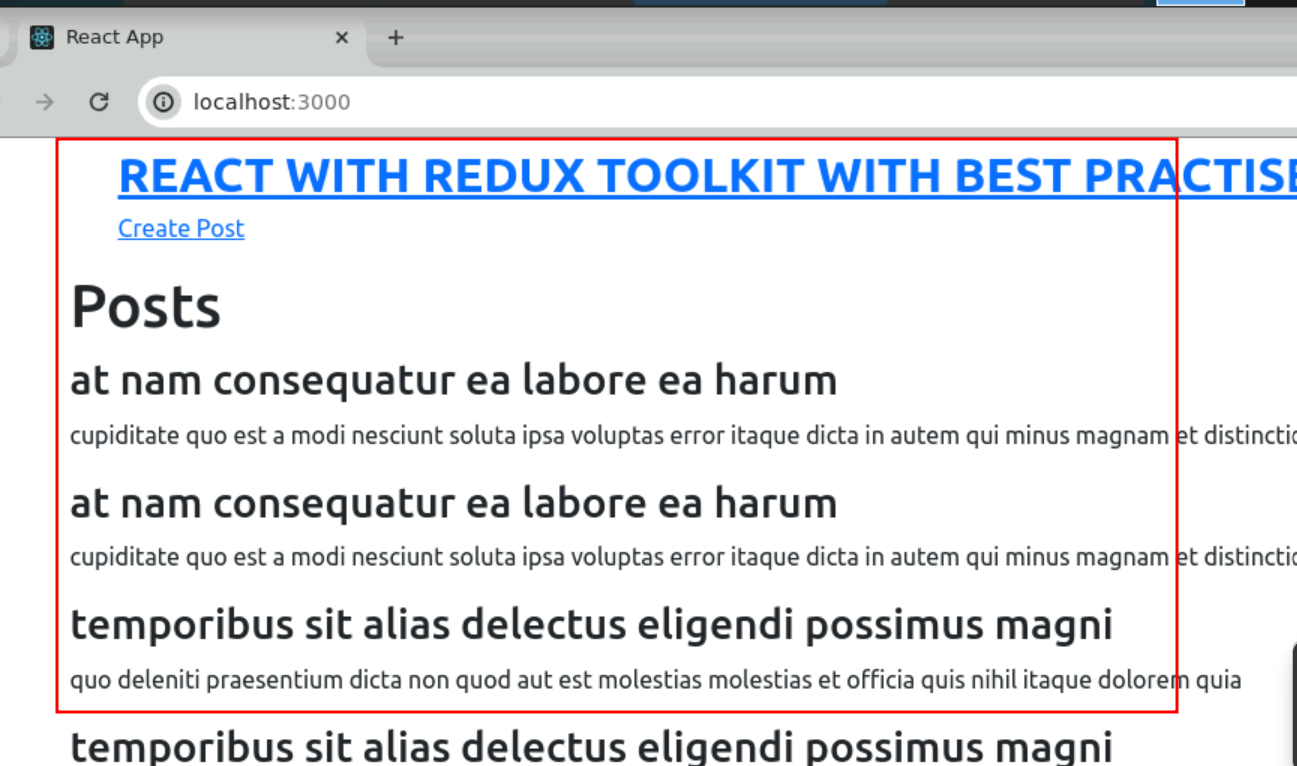
reportWebVitals();

**Step 6 : Now we run the application using npm start**

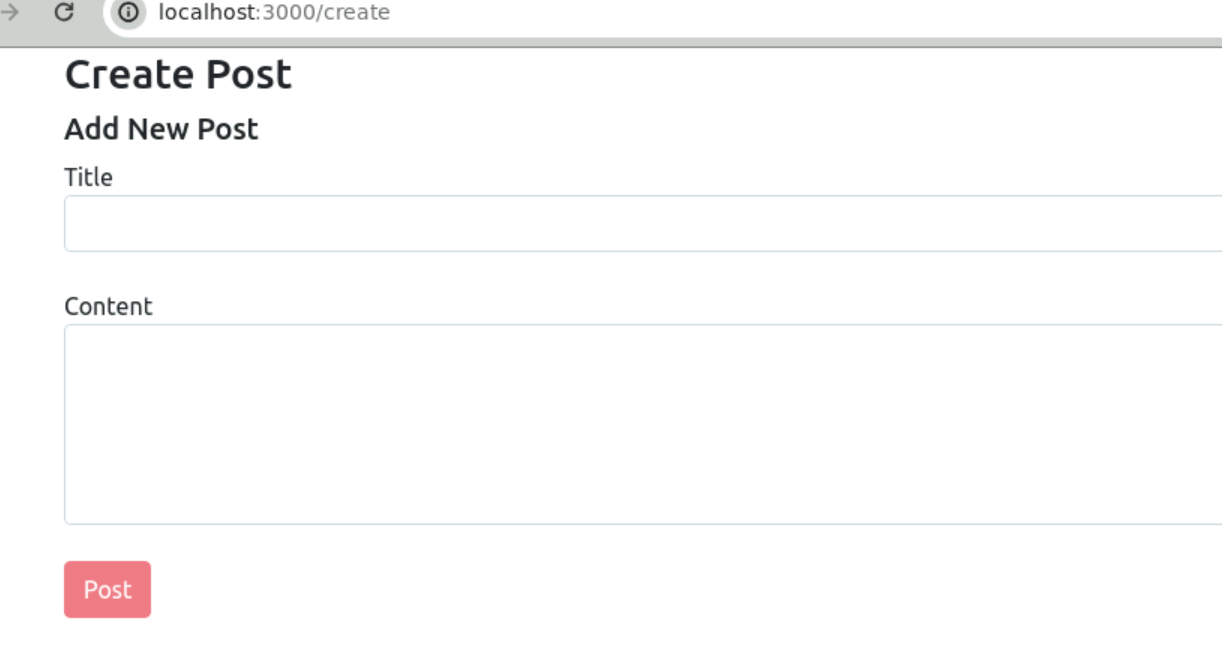
**6.1** Now run the application usiing command as **npm start**



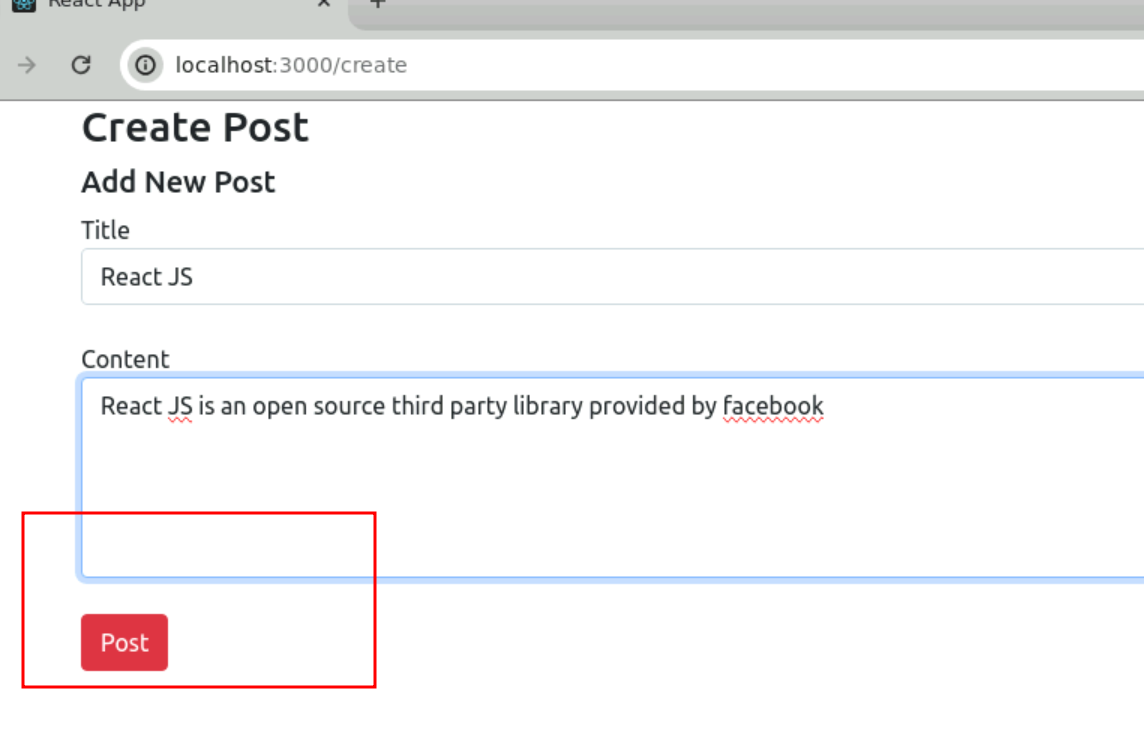
**6.2** If you see the output on browser



6.3 Now you can create new post. But those information only store temporary in redux store not in backend. Because we are using fake rest api to consume json data.



6.4 create new post



6.5 Now you can view added post.

