|  |
| --- |
| **React** |
|  |

|  |
| --- |
| **CRUD With React-Redux** |
| **Post.js** |
| import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';  // It is used to define our endpoints and allow to create the API slice  export const postApi = createApi({   // The unique key that defines where the Redux store will store our cache.   reducerPath: 'postApi',   // The base query to request data.   // RTK Query ships with fetchBaseQuery, which is a lightweight fetch wrapper that automatically handles request headers and response parsing in a manner similar to common libraries like axios.   baseQuery: fetchBaseQuery({    baseUrl: 'https://jsonplaceholder.typicode.com/',   }),   // The set of operations that we want to perform against the server.  **endpoints: (builder) => ({**  // Show all data  **getAllPost: builder.query({**  **query: () => ({**  **url: 'posts',**  **method: 'GET'**  **})**  **}),**  // Show data with specific ID  **getPostById: builder.query({**  **query: (id) => {**  **console.log("ID:", id)**  **return {**  **url: `posts/${id}`,**  **method: 'GET'**  **}**  **}**  **}),**  // Show all with LIMIT  **getPostByLimit: builder.query({**  **query: (num) => {**  **console.log("Limit Number:", num)**  **return {**  **url: `posts?\_limit=${num}`,**  **method: 'GET'**  **}**  **}**  **}),**  // Delete data  **deletePost: builder.mutation({**  **query: (id) => {**  **console.log("Delete ID:", id)**  **return {**  **url: `posts/${id}`,**  **method: 'DELETE'**  **}**  **}**  **}),**  // Insert New Data  **createPost: builder.mutation({**  **query: (newPost) => {**  **console.log("Create Post: ", newPost)**  **return {**  **url: `posts`,**  **method: 'POST',**  **body: newPost,**  **headers: {**  **'Content-type': 'application/json; charset=UTF-8',**  **}**  **}**  **}**  **}),**  // Update Data  **updatePost: builder.mutation({**  **query: (updatePostData) => {**  **console.log("Update Post: ", updatePostData)**  **const { id, ...data } = updatePostData**  **console.log("Actual Update Post: ", data)**  **return {**  **url: `posts/${id}`,**  **method: 'PUT',**  **body: data,**  **headers: {**  **'Content-type': 'application/json; charset=UTF-8',**  **}**  **}**  **}**  **}),**  **}),**  // Export hooks for usage in functional components, which are auto-generated based on the defined endpoints  export const { useGetAllPostQuery, useGetPostByIdQuery, useGetPostByLimitQuery, useDeletePostMutation, useCreatePostMutation, useUpdatePostMutation } = postApi |
| **Store.js** |
| import { configureStore } from '@reduxjs/toolkit'  import { setupListeners } from '@reduxjs/toolkit/query';  import { postApi } from '../services/post';  **export const store = configureStore({**    // reducerPath and reducer are created for us, which we can pass straight into the reducer parameter of configureStore.  **reducer: {**  **[postApi.reducerPath]: postApi.reducer**  **},**    // middleware is also created for us, which will allow us to take advantage of caching, invalidation, polling, and the other features of RTK Query.  **middleware: (getDefaultMiddleware) =>**  **getDefaultMiddleware().concat(postApi.middleware),**  **})**  // It will enable to refetch the data on certain events, such as refetchOnFocus and refetchOnReconnect.  **setupListeners(store.dispatch)** |

|  |
| --- |
| **Wrap <App /> with Provider (Index.js)** |
| import React from 'react';  import ReactDOM from 'react-dom';  import { BrowserRouter } from "react-router-dom";  import './index.css';  import App from './App';  import { Provider } from 'react-redux'  import {store} from './services/store'  ReactDOM.render(    <React.StrictMode>        <BrowserRouter>        <Provider store={store} >            <App />        </Provider>        </BrowserRouter>    </React.StrictMode>,    document.getElementById('root')  ); |
| **Now you can use API (Fetch All data of API)** |
| import React from "react";  import { Card, Button } from "react-bootstrap";  import { CartPlus } from 'react-bootstrap-icons';  import {useGetAllCategoriesQuery} from '../services/Post'  const Products = () => {  **const {data} = useGetAllCategoriesQuery()**  **console.log(data)**    return (      <>       {data===undefined?'no data found'        :data.map(prod=>{         return (**<p>{prod.name}**)  }      </>    );  };  export default Products; |

|  |
| --- |
| CRUD End-Points |
|  |
|  |