# React Context API with TanStack Query for API Operations

This document provides a structured approach to implement API operations using React Context API and TanStack Query (React Query) for a React application. The implementation includes 'GET', 'GET by ID', 'POST (create)', 'PUT (update)', and 'DELETE' operations. This setup allows for centralized API management using Context API, making it easy to access API operations throughout the application.

## 1. PostsContext.js

This file contains the implementation of the PostsContext using React Context API and TanStack Query. It provides functions for fetching all posts, fetching a post by ID, creating a new post, updating an existing post, and deleting a post.

import React, { createContext, useContext, useState } from "react";  
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";  
import axios from "axios";  
  
const PostsContext = createContext();  
const BASE\_URL = "http://localhost:4000/posts";  
  
// API functions  
const fetchPosts = async () => {  
 const response = await axios.get(BASE\_URL);  
 return response.data;  
};  
  
const fetchPostById = async (postId) => {  
 const response = await axios.get(\`\${BASE\_URL}/\${postId}\`);  
 return response.data;  
};  
  
const createPost = async (newPost) => {  
 const response = await axios.post(BASE\_URL, newPost);  
 return response.data;  
};  
  
const updatePost = async ({ postId, updatedData }) => {  
 const response = await axios.put(\`\${BASE\_URL}/\${postId}\`, updatedData);  
 return response.data;  
};  
  
const deletePost = async (postId) => {  
 const response = await axios.delete(\`\${BASE\_URL}/\${postId}\`);  
 return response.data;  
};  
  
export const PostsProvider = ({ children }) => {  
 const queryClient = useQueryClient();  
 const [display, setDisplay] = useState(false);  
//the mutate is a function,and any parameter given to mutate function will pass that paramerters to above async function

//even refetch is a function

// Fetch all posts  
 const { data: posts, isLoading, isError, error, refetch } = useQuery({  
 queryKey: ["posts"],  
 queryFn: fetchPosts,  
 });  
  
 // Fetch post by ID  
 const { data: postById, refetch: refetchPostById } = useQuery({  
 queryKey: ["postById"],  
 queryFn: () => fetchPostById(selectedPostId),  
 enabled: false,  
 });  
  
 // Create a new post  
 const { mutate: createNewPost } = useMutation({  
 mutationFn: createPost,  
 onSuccess: () => {  
 queryClient.invalidateQueries(["posts"]);  
 alert("Post created successfully");  
 },  
 });  
  
 // Update an existing post  
 const { mutate: updateExistingPost } = useMutation({  
 mutationFn: updatePost,  
 onSuccess: () => {  
 queryClient.invalidateQueries(["posts"]);  
 alert("Post updated successfully");  
 },  
 });  
  
 // Delete a post  
 const { mutate: deletePostById, isSuccess, isError: deleteError } = useMutation({  
 mutationFn: deletePost,  
 onSuccess: () => {  
 queryClient.invalidateQueries(["posts"]);  
 },  
 });  
  
 // Toggle display state  
 const handleToggle = () => setDisplay(!display);  
  
 const handleDelete = (id) => {  
 deletePostById(id); //mutate fuctions the alias  
 if (isSuccess) {  
 alert(\`Post with ID \${id} deleted successfully\`);  
 } else if (deleteError) {  
 alert("Error deleting post");  
 }  
 };  
  
 const handleCreate = (newPost) => {  
 createNewPost(newPost); //mutate fuctions the alias  
 };  
  
 const handleUpdate = (postId, updatedData) => {  
 updateExistingPost({ postId, updatedData });//mutate fuctions the alias  
 };  
  
 const handleFetchById = (postId) => {  
 refetchPostById(postId); //mutate fuctions the alias  
 };  
  
 return (  
 <PostsContext.Provider  
 value={{  
 posts,  
 isLoading,  
 isError,  
 error,  
 refetch,  
 postById,  
 handleToggle,  
 display,  
 handleDelete,  
 handleCreate,  
 handleUpdate,  
 handleFetchById,  
 }}  
 >  
 {children}  
 </PostsContext.Provider>  
 );  
};  
  
export const usePosts = () => {  
 return useContext(PostsContext);  
};

## 2. Wrapping Application with PostsProvider

In your main application file (e.g., main.jsx or App.jsx), wrap your app with the PostsProvider to make the context available throughout the component tree.

import React from "react";  
import ReactDOM from "react-dom/client";  
import App from "./App";  
import { PostsProvider } from "./PostsContext";  
  
ReactDOM.createRoot(document.getElementById("root")).render(  
 <React.StrictMode>  
 <PostsProvider>  
 <App />  
 </PostsProvider>  
 </React.StrictMode>  
);

## 3. Example Usage in a Component

The PostsManager component demonstrates how to use the context API functions such as fetching posts, creating a new post, updating a post, deleting a post, and fetching a post by ID.

import React, { useState } from "react";  
import { usePosts } from "./PostsContext";  
  
const PostsManager = () => {  
 const {  
 posts,  
 isLoading,  
 isError,  
 error,  
 handleDelete,  
 handleCreate,  
 handleUpdate,  
 handleFetchById,  
 postById,  
 display,  
 handleToggle,  
 } = usePosts();  
  
 const [newPost, setNewPost] = useState({ title: "", body: "" });  
 const [selectedPostId, setSelectedPostId] = useState("");  
 const [updatedData, setUpdatedData] = useState({ title: "", body: "" });  
  
 if (isLoading) return <p>Loading...</p>;  
 if (isError) return <p>Error: {error.message}</p>;  
  
 const handleCreatePost = () => {  
 handleCreate(newPost);  
 };  
  
 const handleUpdatePost = () => {  
 handleUpdate(selectedPostId, updatedData);  
 };  
  
 const handleFetchPostById = () => {  
 handleFetchById(selectedPostId);  
 };  
  
 return (  
 <div>  
 <button onClick={handleToggle}>  
 {display ? "Hide Posts" : "Show Posts"}  
 </button>  
  
 {display && (  
 <ul>  
 {posts.map((post) => (  
 <li key={post.id}>  
 {post.title}  
 <button onClick={() => handleDelete(post.id)}>Delete</button>  
 </li>  
 ))}  
 </ul>  
 )}  
  
 <h3>Create New Post</h3>  
 <input type="text" placeholder="Title" value={newPost.title} onChange={(e) => setNewPost({ ...newPost, title: e.target.value })} />  
 <input type="text" placeholder="Body" value={newPost.body} onChange={(e) => setNewPost({ ...newPost, body: e.target.value })} />  
 <button onClick={handleCreatePost}>Create Post</button>  
  
 <h3>Update Post</h3>  
 <input type="text" placeholder="Post ID" value={selectedPostId} onChange={(e) => setSelectedPostId(e.target.value)} />  
 <input type="text" placeholder="Updated Title" value={updatedData.title} onChange={(e) => setUpdatedData({ ...updatedData, title: e.target.value })} />  
 <input type="text" placeholder="Updated Body" value={updatedData.body} onChange={(e) => setUpdatedData({ ...updatedData, body: e.target.value })} />  
 <button onClick={handleUpdatePost}>Update Post</button>  
  
 <h3>Fetch Post by ID</h3>  
 <input type="text" placeholder="Post ID" value={selectedPostId} onChange={(e) => setSelectedPostId(e.target.value)} />  
 <button onClick={handleFetchPostById}>Fetch Post</button>  
  
 {postById && (  
 <div>  
 <h4>Post Details</h4>  
 <p>ID: {postById.id}</p>  
 <p>Title: {postById.title}</p>  
 <p>Body: {postById.body}</p>  
 </div>  
 )}  
 </div>  
 );  
};  
  
export default PostsManager;