**React Query**

package.json for **react-course-part2-starter-main** project

{

  "name": "react-app-starter",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "tsc && vite build",

    "preview": "vite preview"

  },

  "dependencies": {

    "axios": "^1.3.4",

    "bootstrap": "^5.2.3",

    "react": "^18.2.0",

    "react-dom": "^18.2.0"

  },

  "devDependencies": {

    "@types/react": "^18.0.28",

    "@types/react-dom": "^18.0.11",

    "@vitejs/plugin-react": "^3.1.0",

    "typescript": "^4.9.3",

    "vite": "^4.2.0"

  }

}

TodoList.tsx

import axios from 'axios';

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

interface Todo {

  id: number;

  title: string;

  userId: number;

  completed: boolean;

}

const TodoList = () => {

  const [todos, setTodos] = useState<Todo[]>([]);

  const [error, setError] = useState('');

  useEffect(() => {

    axios

      .get('https://jsonplaceholder.typicode.com/todos')

      .then((res) => setTodos(res.data))

      .catch((error) => setError(error));

  }, []); //Empty array, is dependencies, when to call useEffect() again

  if (error) return <p>{error}</p>;

  return (

    <ul className="list-group">

      {todos.map((todo) => (

        <li key={todo.id} className="list-group-item">

          {todo.title}

        </li>

      ))}

    </ul>

  );

};

export default TodoList;

Problems:

1. No request cancellation when component is unmounted.

We use useEffect() to execute code with side-effects. We should pass a clean-up function, for un-doing thing we did before. In case of http request, we should cancel them.

As in react 18, the strict mode is enabled, which causes each component to render twice.

1. No separation of concerns

Querying logic leaks into the component. If we require the same piece of data in another part of the application, there is no opportunity to reuse it.

1. No retries.

If an error happens, we show the user an error and move on.

1. No automatic refresh

If data changes while the user is on this page, they don’t see the changes unless they refresh the page.

1. No caching

Caching is the process of storing data in a place where it can be accessed more quickly and efficiently in the future.

This is where React Query comes into play.

React Query is a powerful library for managing data fetching and caching in React applications.

Redux is a popular state management library for JavaScript applications. It allows us to store the state or the data of an application in a single global store.

Redux is difficult to learn and has much boilerplate code.

React Query is a lot simpler and lightweight.

**Setting up React Query**

* npm install @tanstack/react-query@4.28

main.tsx

import { QueryClient, QueryClientProvider } from "@tanstack/react-query";

const queryClient = new QueryClient();

Wrap <App> with QueryClientProvider.

<QueryClientProvider client={queryClient}>

  <App />

</QueryClientProvider>

QueryClient is the core object used for managing and caching remote data in React query.

**Fetching Data**

useQuery({

    queryKey: ['todos']

})

Or

queryKey: ['todos', 'completed']

queryKey: ['todos', { completed: true }]

useQuery will return Todo[] in case of success or Error object in case of failure.

queryKey: Unique id for the query. Used internally for caching. Anytime we retrieve a piece of data from the backend, the data is stored in the cache and will be accessible via this key.

Type of data we want to store.

2nd value of array, values like completed for storing completed todos. Depends on how we want to configure our stack.

data: todos //alias for data.

const fetchTodos = () => {

return axios

        .get<Todo[]>("https://jsonplaceholder.typicode.com/todos")

        .then((res) => res.data);

};

const { data: todos, error } = useQuery<Todo[], Error>({

    queryKey: ["todos"],

    queryFn: fetchTodos,

  });

queryFn: Function for fetching data from the backend. This function should return a promise that resolves to data or throws an error.

React doesn’t care with which mechanism we fetch data. It’s only concerned with managing and caching data.

axios.get() returns a response object.

TodoList.tsx

import { useQuery } from "@tanstack/react-query";

import axios from "axios";

interface Todo {

  id: number;

  title: string;

  userId: number;

  completed: boolean;

}

const TodoList = () => {

  const fetchTodos = () => {

    return axios

      .get<Todo[]>("https://jsonplaceholder.typicode.com/todos")

      .then((res) => res.data);

  };

  const { data: todos, error } = useQuery<Todo[], Error>({

    queryKey: ["todos"],

    queryFn: fetchTodos,

  });

  if (error) return <p>{error.message}</p>;

  return (

    <ul className="list-group">

      {todos?.map((todo) => (

        <li key={todo.id} className="list-group-item">

          {todo.title}

        </li>

      ))}

    </ul>

  );

};

export default TodoList;

**Showing Loading indicator**

const { data: todos, error, isLoading } = useQuery<Todo[], Error>({

    queryKey: ["todos"],

    queryFn: fetchTodos,

});

if (isLoading) return <p>Loading...</p>;

Problem: No separation of concerns for the querying logic

TodoList.tsx

import useTodos from "./hooks/useTodo";

const TodoList = () => {

  const { data: todos, error, isLoading } = useTodos();

  if (isLoading) return <p>Loading...</p>;

  if (error) return <p>{error.message}</p>;

  return (

    <ul className="list-group">

      {todos?.map((todo) => (

        <li key={todo.id} className="list-group-item">

          {todo.title}

        </li>

      ))}

    </ul>

  );

};

export default TodoList;

hooks/useTodo.ts

import { useQuery } from "@tanstack/react-query";

import axios from "axios";

interface Todo {

    id: number;

    title: string;

    userId: number;

    completed: boolean;

}

const useTodos = () => {

    const fetchTodos = () => {

        return axios

          .get<Todo[]>("https://jsonplaceholder.typicode.com/todos")

          .then((res) => res.data);

    };

    return useQuery<Todo[], Error>({

        queryKey: ["todos"],

        queryFn: fetchTodos,

    });

}

export default useTodos;

**Using React Query DevTools**

* npm install @tanstack/react-query-devtools@4.28

main.tsx

import { ReactQueryDevtools } from "@tanstack/react-query-devtools";

ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(

  <React.StrictMode>

    <QueryClientProvider client={queryClient}>

      <App />

      <ReactQueryDevtools />

    </QueryClientProvider>

  </React.StrictMode>

);

Should only goes in development build.

Trigger loading: Triggers the loading scenario, displace what is seen in case of actual loading.

Trigger error: Trigger the error scenario and displace the error message configured.

**Customizing Query Settings**

main.tsx

const queryClient = new QueryClient({

  defaultOptions: {

    queries: {

      retry: 3,

      cacheTime: 300000, //5mins

      staleTime: 10 \* 1000, //10s

      refetchOnWindowFocus: false,

      refetchOnReconnect: false,

      refetchOnMount: false,

    },

  },

});

retry: Retry the api call. Default 3

cacheTime: If the query has no observer, i.e., no component is using the query, the query is considered inactive. Thus, remove from the cache after 5mins. Garbage Collection.

staleTime: How long data is considered as fresh. Default zero.

For Auto-refresh

refetchOnWindowFocus: When the window is refocused. Default true

refetchOnReconnect: When the network is reconnected. Default true

refetchOnMount: When a component is mounted. Default true

Generally, we don’t have to update the default query settings except for staleTime.

We can override all above configuration for individual queries.

useTodo.ts

const useTodos = () => {

    const fetchTodos = () => {

        return axios

          .get<Todo[]>("https://jsonplaceholder.typicode.com/todos")

          .then((res) => res.data);

    };

    return useQuery<Todo[], Error>({

        queryKey: ["todos"],

        queryFn: fetchTodos,

        staleTime: 10 \* 1000

    });

}

Posts exercise:

PostList.tsx

import usePosts from "./hooks/usePosts";

const PostList = () => {

  const { data: posts, error, isLoading } = usePosts();

  if (isLoading) return <p>Loading...</p>;

  if (error) return <p>{error.message}</p>;

  return (

    <ul className="list-group">

      {posts?.map((post) => (

        <li key={post.id} className="list-group-item">

          {post.title}

        </li>

      ))}

    </ul>

  );

};

export default PostList;

usePosts.ts

import { useQuery } from "@tanstack/react-query";

import axios from "axios";

interface Post {

    id: number;

    title: string;

    body: string;

    userId: number;

}

const usePosts = () => {

    const fetchPosts = () => {

        return axios

            .get('https://jsonplaceholder.typicode.com/posts')

            .then((res) => res.data);

    };

    return useQuery<Post[], Error>({

        queryKey: ["posts"],

        queryFn: fetchPosts,

        staleTime: 10 \* 1000

    });

}

export default usePosts;

**Parameterized Queries**

PostList.tsx

import { useState } from "react";

import usePosts from "./hooks/usePosts";

const PostList = () => {

  const [userId, setUserId] = useState<number>();

  const { data: posts, error, isLoading } = usePosts(userId);

  if (isLoading) return <p>Loading...</p>;

  if (error) return <p>{error.message}</p>;

  return (

    <>

      <select

        onChange={(event) => setUserId(parseInt(event.target.value))}

        value={userId}

        className="form-select mb-3"

      >

        <option value=""></option>

        <option value="1">User 1</option>

        <option value="2">User 2</option>

        <option value="3">User 3</option>

      </select>

      <ul className="list-group">

        {posts?.map((post) => (

          <li key={post.id} className="list-group-item">

            {post.title}

          </li>

        ))}

      </ul>

    </>

  );

};

export default PostList;

usePosts.ts

import { useQuery } from "@tanstack/react-query";

import axios from "axios";

interface Post {

    id: number;

    title: string;

    body: string;

    userId: number;

}

const usePosts = (userId: number | undefined) => {

    const fetchPosts = () => {

        return axios

            .get('https://jsonplaceholder.typicode.com/posts', {

                params: {

                    userId

                }

            })

            .then((res) => res.data);

    };

    return useQuery<Post[], Error>({

        queryKey: userId ? ["users", userId, "posts"] : ["posts"],

        queryFn: fetchPosts,

        staleTime: 10 \* 1000

    });

}

export default usePosts;

The hierarchy of the queryKey will be “user/1/posts”, as we would give in any REST calls. With each level, we filter data.

So anytime the userId is updated, the query will get executed.

userId is passed as params in axios query to fetch posts by userId.

**Paginated Queries**

usePosts now gets current page number and page size (which is constant)

Next button can be disabled as we wont able to fetch the total size of the posts beforehand.

PostList.tsx

import { useState } from "react";

import usePosts from "./hooks/usePosts";

const PostList = () => {

  const pageSize = 10;

  const [page, setPage] = useState(1);

  const { data: posts, error, isLoading } = usePosts({ page, pageSize });

  if (isLoading) return <p>Loading...</p>;

  if (error) return <p>{error.message}</p>;

  return (

    <>

      <ul className="list-group">

        {posts?.map((post) => (

          <li key={post.id} className="list-group-item">

            {post.title}

          </li>

        ))}

      </ul>

      <button

        disabled={page === 1}

        className="btn btn-primary my-3"

        onClick={() => setPage(page - 1)}

      >

        Previous

      </button>

      <button

        className="btn btn-primary my-3 ms-1"

        onClick={() => setPage(page + 1)}

      >

        Next

      </button>

    </>

  );

};

export default PostList;

usePosts.ts

import { useQuery } from "@tanstack/react-query";

import axios from "axios";

interface Post {

    id: number;

    title: string;

    body: string;

    userId: number;

}

interface PostQuery {

    page: number;

    pageSize: number;

}

const usePosts = (query: PostQuery) => {

    const fetchPosts = () => {

        return axios

            .get('https://jsonplaceholder.typicode.com/posts', {

                params: {

                    \_start: (query.page - 1) \* query.pageSize,

                    \_limit: query.pageSize

                }

            })

            .then((res) => res.data);

    };

    return useQuery<Post[], Error>({

        queryKey: ["posts", query],

        queryFn: fetchPosts,

        staleTime: 10 \* 1000,

        keepPreviousData: true

    });

}

export default usePosts;

\_start, \_limit params for pagination

keepPreviousData for better user experience as the page would scroll to top when Next/Previous button click.