

Step 1 for TanStack

The screenshot shows a code editor on the left with the file `main.jsx` open. The code imports React, ReactDOM, and App from the react-dom/client package, and QueryClient and QueryClientProvider from @tanstack/react-query. It creates a new `QueryClient`, renders the `App` component with a `QueryClientProvider`, and uses `ReactDOM.createRoot` to render it into the `root` element. A browser window on the right shows the `Vite + React` logo with a lightning bolt icon and an atom icon. Below the logo, there is a button labeled "Edit src/App.jsx and save to test HMR".

```
src > main.jsx > ...
1 import React from "react"; 6.9k (gzipped: 2.7k)
2 import ReactDOM from "react-dom/client"; 511 (gzipped: 319)
3 import App from "./App.jsx";
4 import "./index.css";
5 import { QueryClient, QueryClientProvider } from "@tanstack/react-query"; 23
6
7 const client = new QueryClient();
8
9 ReactDOM.createRoot(document.getElementById("root")).render(
10   <React.StrictMode>
11     <QueryClientProvider client={client}>
12       | <App />
13     </QueryClientProvider>
14   </React.StrictMode>
15 );
16
```

Fetch Data

The screenshot shows a code editor on the left with the file `App.jsx` open. The code defines a `fetchPosts` function that fetches data from `https://jsonplaceholder.typicode.com/posts`. It then uses the `useQuery` hook to query the posts, mapping the results to a list of titles. The browser window on the right displays a series of Latin placeholder text blocks, likely used for testing the data fetching logic.

```
src > App.jsx > [o] fetchPosts
5 const fetchPosts = async () => {
6   const response = await fetch("https://jsonplaceholder.typicode.com/posts");
7   if (!response.ok) throw new Error("Error fetching data");
8   return response.json();
9 };
10
11 function App() {
12   const { data, isLoading, error } = useQuery({
13     queryKey: ["posts"],
14     queryFn: fetchPosts,
15   });
16
17   if (isLoading) return <p> Loading...</p>;
18
19   if (error) return <p> Error occurred: {error.message}</p>;
20
21   return (
22     <>
23       {" "}
24       {data.map((post) => (
25         | <p> {post.title}</p>
26       )));
27     </>

```

Keep fetching data after a certain time period (In this example, data is fetched every 5 seconds (5000ms))

```
src > components > Posts.jsx > Posts
1 import { useQuery } from "@tanstack/react-query"; 13.5k (gzipped: 4.7k)
2
3 const fetchPosts = async () => {
4   const response = await fetch("https://jsonplaceholder.typicode.com/posts");
5   if (!response.ok) throw new Error("Error fetching data");
6   return response.json();
7 };
8
9 export const Posts = () => [
10   const { data, isLoading, error } = useQuery({
11     queryKey: ["posts"],
12     queryFn: fetchPosts,
13     staleTime: 5000,
14   });
15
16   if (isLoading) return <p> Loading...</p>;
17
18   if (error) return <p> Error occurred: {error.message}</p>;
19
20   return (
21     <>
22       {" "}
23       {data.map((post) => (
24         <p> {post.title}</p>
25       )));
26     </>
27   );
28 };
29
```

Fetch data using ID

```
src > components > PostById.jsx > PostById
1 import { useQuery } from "@tanstack/react-query"; 13.5k (gzipped: 4.7k)
2
3 const fetchPosts = async (id) => {
4   const response = await fetch(
5     `https://jsonplaceholder.typicode.com/posts/${id}`
6   );
7   if (!response.ok) throw new Error("Error fetching data");
8   return response.json();
9 };
10
11 export const PostById = ({ id }) => {
12   const { data, isLoading, error } = useQuery({
13     queryKey: ["posts", id],
14     queryFn: () => fetchPosts(id),
15     staleTime: 10000,
16   });
17
18   if (isLoading) return <p> Loading...</p>;
19
20   if (error) return <p> Error occurred: {error.message}</p>;
21
22   return <> {data.title}</>;
23 };

```

Toggle

ea molestias quasi exercitationem repellat qui ipsa sit aut

Posting data

```
src > components > CreatePost.jsx > [x] CreatePost
1 import { useMutation } from "@tanstack/react-query"; 4.5k (gzipped: 2k)
2 import { useState } from "react"; 4.2k (gzipped: 1.8k)
3
4 const createPost = async (newPost) => {
5   const response = await fetch("https://jsonplaceholder.typicode.com/posts",
6     method: "POST",
7     headers: { "Content-Type": "application/json" },
8     body: JSON.stringify(newPost),
9   );
10
11   return response.json();
12 };
13
14 export const CreatePost = () => {
15   const [title, setTitle] = useState("");
16
17   const { mutate } = useMutation([ mutationFn: createPost ]);
18
19   const handleSubmit = (e) => {
20     e.preventDefault();
21     mutate({ title, body: "This is a new post" });
22   };
23 }
```

Implementing optimistic updates

```
src > components > CreatePost.jsx > [x] CreatePost
14 export const CreatePost = () => {
15   const queryClient = useQueryClient();
16
17   const { mutate } = useMutation([
18     mutationFn: createPost,
19     onSuccess: () => {
20       queryClient.invalidateQueries(["posts"]);
21     },
22   ],
23   onMutate: async (newPost) => {
24     await queryClient.cancelQueries(["posts"]);
25     const previousPosts = queryClient.getQueryData(["posts"]);
26     queryClient.setQueryData(["posts"], (old) => [
27       ...old,
28       { id: Date.now(), ...newPost },
29     ]);
30
31     return { previousPosts };
32   },
33 );
34
35 const handleSubmit = (e) => {
36   e.preventDefault();
37   mutate({ title, body: "This is a new post" });
38 };
```

Referring to other queries

The screenshot shows a development environment with two tabs open: 'App.jsx' and 'CreatePost.jsx'. The 'CreatePost.jsx' tab is active, displaying the following code:

```
src > components > CreatePost.jsx > CreatePost
4  const createPost = async (newPost) => {
5    const response = await fetch("https://jsonplaceholder.typicode.com/
6      headers: { "Content-Type": "application/json" },
7      body: JSON.stringify(newPost),
8    );
9  }
10
11  return response.json();
12};

13
14 export const CreatePost = () => {
15   const [title, setTitle] = useState("");
16
17   const queryClient = useQueryClient();
18
19   const { mutate } = useMutation({
20     mutationFn: createPost,
21     onSuccess: () => {
22       | queryClient.invalidateQueries(["posts"]);
23     },
24   });
25
26   const handleSubmit = (e) => {
27     e.preventDefault();
28     | mutate({ title, body: "This is a new post" });
29   };
30}
```

The browser window on the right displays a simple form with a 'New Post' button and a 'Create' button. The page content is a Latin placeholder text from jsonplaceholder.typicode.com.

localhost:5173

New Post

Create

sunt aut facere repellat provident occaecati excepturi optio reprehenderit
qui est esse
ea molestias quasi exercitationem repellat qui ipsa sit aut
eum et est occaecati
nesciunt quas odio
dolorem eum magni eos aperiam quia
magnam facilis autem
dolorem dolore est ipsam
nesciunt iure omnis dolorem tempora et accusantium
optio molestias id quia eum