ToiletPaper #137

API calls with React Query

Author: Benjamin Hogl / Software Engineer / Office Stuttgart

✕ Problem

Your React front-end needs to talk to a back-end? You don't want to set up a Redux store with countless actions and reducers just to store the back-end's responses and the status of the requests somewhere? You don't want to worry about when to start which request, but simply access the data? You don't want to develop complicated logic to keep the displayed data up-to-date or to repeat failed requests?

✓ Solution

React Query handles almost everything related to HTTP queries, including for example:

- Caching and request deduplication via unique queryKeys (in the example below `["users", 3]`)
- Automatic background refresh of data (configurable)
- Automatic retry if a request fails (also configurable)
- Provision of the current request status (`isLoading`, `isError`, `isSuccess`, etc.)

If you don't want a request to start immediately when mounting a component, you can mark it as disabled.

Rendering content, loading indicators or error messages, and the implementation of the actual HTTP request (e.g. with axios or via the Fetch API) is up to you. In these points, React Query is not imposing any constraints on you.

Using React Query will most likely make your code clearer and less error-prone. And in case something doesn't work the way you want it to, React Query even provides its own devtools. These are automatically removed in a production build, so you don't have to take care of that.

➔ Example

```tsx
import { QueryClient, QueryClientProvider, useQuery } from "react-query";
import { ReactQueryDevtools } from "react-query/devtools";
type ApiResponse = { data: { first_name: string } };

const queryClient = new QueryClient();
export default function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <Example />
      <ReactQueryDevtools />
    </QueryClientProvider>
  );
}
function Example() {
  const query = useQuery<ApiResponse>(["users", 3], () => {
    return fetch("https://reqres.in/api/users/3?delay=1").then((res) => res.json());
  });
  if (query.isLoading) return <p>Loading...</p>;
  if (query.isError) return <button onClick={() => query.refetch()}>Try again</button>;
  return <p>Hello {query.data?.data.first_name}</p>;
}
```

+ Further Aspects

- https://react-query.tanstack.com/guides/important-defaults
  The carefully chosen default settings can be confusing if you don't know about them.
- https://epicreact.dev/my-state-management-mistake by Kent C. Dodds
  Relevant quote from it: *"Server cache is not the same as UI state, and should be handled differently."*