Tanstack Query

Tanstack manages server state, and state managers like Zustand manage app state. But I think Tanstack still helps us on the client side for things like infinite-loading and optimistic updates. Tanstack doesn't fetch data for you, you do that yourself with any tool you want.

The starting useEffect data fething for categories has 3 bugs:

1. Race conditions due to delays.
2. Momentary loading state that says No Posts
3. When you increase error chances to something like 90% then back down, the likelihood stays high

How do you fix these bugs while still in useEffect?

1. Race conditions: use 'ignore' flag. Set it to false, then within all blocks of try-catch, only execute if ignore is false. The at the bottom of the useEffect, return a clean-up function that sets ignore equal to true.
2. No Posts momentarily: this depends on the loading state, where if it's false and you have posts.length === 0 and no error. I think react renders the jsx then after runs the effects, so that's why. The fix is just to initialize the loading state to true at first.
3. The error case spawns two problems that need distinction. The other problem is that because its not handled properly, we can end up viewing the wrong previously fetched data, because we didn't reset it properly and it shoulda been empty. Anothet thing that needs to be handled is the two cases of states for the posts– returned empty array from backend, and not loaded yet; so you initialize posts with an empty array. The real fix though is to properly set the error state and posts state. The way you do that is in the success cases of either one, posts or error, reset the other one.

Tanstack for Searching

When we have a variable that we use in our query, we always put it in the cache key– goes for every variable that you send to the backend. For example in this searching case, it will be the search-term. Every search term will have its own entry in the cache– useQuery({ queryKey: ["posts", "search", "searchTerm"}). The search-term used should actually be the debounced-search-term in both the query-key and fetching parameter, because we only wanna start searching once the debounce value changes, and Tanstack automatically re-fetches data when the query-key changes.

You can have the query not run automatically on page render and wait for the search-term to not be empty using the 'enabled' property of useQuery.

Infinity Loading

You use useInfiniteQuery hook for this.

Data Mutations

You use React's useMutation hook. Notice the pattern here, how you return the returns of the default hooks you're using when making your custom hooks.

No query-key, because nothin is fetched. On success of the mutation, you can setup a refetch by having the cached query invalidated, but you must use a queryClient. There is a problem

with using this invalidation though– it refetches everything starting from the beginning, there after it shows the new updates. It invalidates all the queries you fetched.
The better method is to not call the invalidation at all, instead mutate the cache ourselves– go into the list in our cache, and put the new comment in there ourselves– when you post something, if the backend is configured to return that thing, that's what you put into the cache. Inside the onSuccess, you can access whatever is returned.
Remember whenever you mutate your cache reserve, you wanna start by cancelling running queries.