SHIP FASTER WEBSITES WITH REACT QUERY

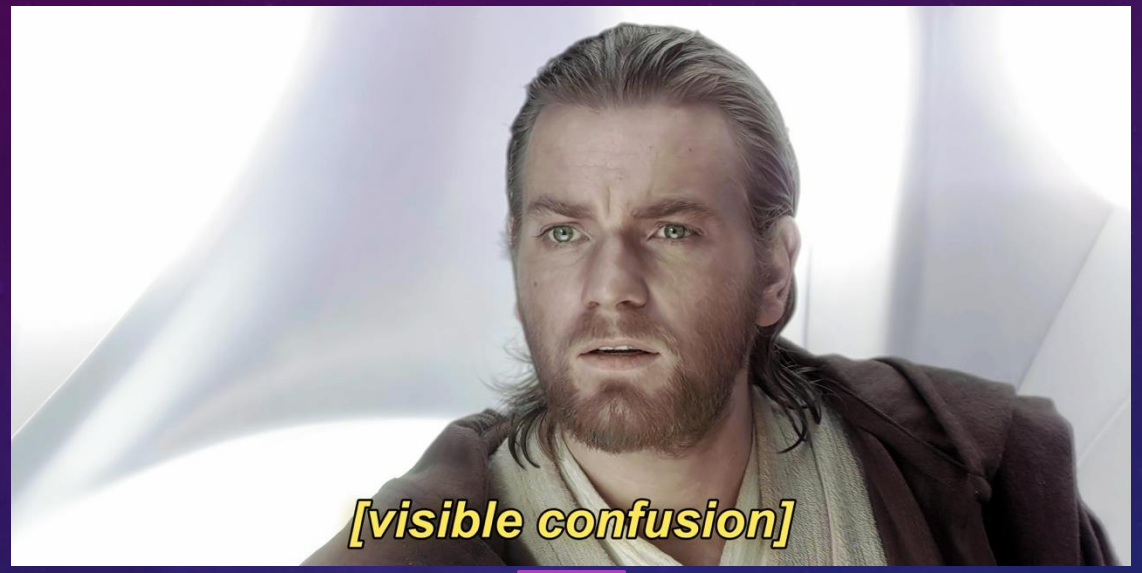
(TANSTACK QUERY)

By Kevin Van Cott

WHO AM I?

- Senior Software Engineer / OSS Maintainer
- I currently work at RentVision in Lincoln, NE
- Other companies I've worked for:
  Manifest Cyber, Fusion Medical Staffing, ALLO Fiber, Talent Plus, Nebraska DHHS
- Open Source Contributor – TanStack Table, Material React Table
- TanStack Consultant - Dedicated Support | TanStack

TANSTACK QUERY



WHAT'S WITH THE NAME?

IS IT "REACT QUERY" OR "TANSTACK QUERY"?

THE VANILLA WAY TO FETCH IN REACT

```
function Bookmarks({ category }) {
  const [data, setData] = useState([])
  const [error, setError] = useState()

  useEffect(() => {
    fetch(`${endpoint}/${category}`)
      .then(res => res.json())
      .then(d => setData(d))
      .catch(e => setError(e))
  }, [category])

  // Return JSX based on data and error state
}
```

Can you actually write code exactly like this?

Why You Want React Query | TkDodo's blog

WHY USE REACT QUERY?

- Automatic Data State Management for all GET requests
- Automatic Loading and Reloading states and logic
- Automatic Error Handling and error handling states
- Dedicated Mutation logic (POST, UPDATE, DELETE, PATCH requests)
- CACHING! (and everything to do with caching – state time, gc/cache time, cache invalidation/busting)
- PREFETCHING!
- Polling and Re-fetching features
- Pagination and Infinite Scrolling features
- Better memoization and re-rendering optimizations via structural sharing
- Can work with SSR/SSG
- Can work with RSCs
- Offline features
- Persistence to session or local storage (or IndexedDB or SQLite)
- Awesome Devtools
- Comparison | React Query vs SWR vs Apollo vs RTK Query vs React Router | TanStack Query React Docs
- @reduxjs/toolkit vs @tanstack/query-core vs react-query vs react-relay vs swr | npm trends

useQuery

[useQuery | TanStack Query React Docs]

```
const { data, isLoading, isError } = useQuery({
  queryKey: ["posts"],
  queryFn: async () => fetch(`http://localhost:3333/posts`).then((res) => res.json()),
});
```

Options

queryKey,
queryFn,
gcTime,
enabled,
networkMode,
initialData,
initialDataUpdatedAt,
meta,
notifyOnChangeProps,
placeholderData,
queryKeyHashFn,
refetchInterval,
refetchIntervalInBackground,
refetchOnMount,
refetchOnReconnect,
refetchOnWindowFocus,
retry,
retryOnMount,
retryDelay,
select,
staleTime,
structuralSharing,
throwOnError,

Returns

data,
dataUpdatedAt,
error,
errorUpdatedAt,
failureCount,
failureReason,
fetchStatus,
isError,
isFetched,
isFetchedAfterMount,
isFetching,
isInitialLoading,
isLoading,
isLoadingError,
isPaused,
isPending,
isPlaceholderData,
isRefetchError,
isRefetching,
isStale,
isSuccess,
refetch,
status,

useMutation

[useMutation | TanStack Query React Docs]

```
const { mutate, isPending, isError } = useMutation({
  mutationFn: (newTodo) =>
    fetch("/add-todo", {
      method: "POST",
      body: JSON.stringify(newTodo),
    }).then((response) => response.json()),
  onSuccess: () => {
    showNotification({ title: "Todo added", message: "Todo added successfully" });
  },
  onError: () => {
    console.error("Error adding todo");
    showNotification({ title: "Error adding todo", message: "Error adding todo" });
  },
});
```

Options

- mutationFn,
- gcTime,
- meta,
- mutationKey,
- networkMode,
- onError,
- onMutate,
- onSettled,
- onSuccess,
- retry,
- retryDelay,
- scope,
- throwOnError,

Returns

- data,
- error,
- isError,
- isIdle,
- isPending,
- isPaused,
- isSuccess,
- failureCount,
- failureReason,
- mutate,
- mutateAsync,
- reset,
- status,
- submittedAt,
- variables,

useQueries

[useQueries | TanStack Query React Docs]

```tsx
const combinedQueries = useQueries({
  queries: [
    {
      queryKey: ["post", postId],
      queryFn: () =>
        fetch(`http://localhost:3333/posts/${postId}`).then((res) =>
          res.json()
        ),
    },
    {
      queryKey: ["user", userId],
      queryFn: () =>
        fetch(`http://localhost:3333/users/${post?.userId}`).then((res) =>
          res.json()
        ),
    },
  ],
  combine: (results) => {
    return {
      post: results[0],
      user: results[1],
    };
  },
});
```

Options	Returns
queries	combined queries
combine	

useSuspenseQuery

[useSuspenseQuery | TanStack Query React Docs]

```
const { data } = useSuspenseQuery({
  queryKey: ["posts"],
  queryFn: async () => fetch(`http://localhost:3333/posts`).then((res) =>
res.json()),
```

Options

- **queryKey**,
- **queryFn**,
- gcTime,
- ~~enabled~~,
- networkMode,
- initialData,
- initialDataUpdatedAt,
- meta,
- notifyOnChangeProps,
- ~~placeholderData~~,
- queryKeyHashFn,
- refetchInterval,
- refetchIntervalInBackground,
- refetchOnMount,
- refetchOnReconnect,
- refetchOnWindowFocus,
- retry,
- retryOnMount,
- retryDelay,
- select,
- staleTime,
- structuralSharing,
- throwOnError,

Returns

- data,
- dataUpdatedAt,
- error,
- errorUpdatedAt,
- failureCount,
- failureReason,
- fetchStatus,
- isError,
- isFetched,
- isFetchedAfterMount,
- isFetching,
- isInitialLoading,
- ~~isLoading~~,
- isLoadingError,
- isPaused,
- isPending,
- ~~isPlaceholderData~~,
- isRefetchError,
- isRefetching,
- isStale,
- ~~isSuccess~~,
- refetch,
- ~~status~~,

useInfiniteQuery

[useInfiniteQuery | TanStack Query React Docs]

```tsx
const {
  data: posts,
  isError: isErrorLoadingPosts,
  isFetching: isFetchingPosts,
  isLoading: isLoadingPosts,
  fetchNextPage,
} = useInfiniteQuery({
  queryKey: ["posts"],
  queryFn: async ({ pageParam = 0 }) => {
    const fetchUrl = new URL(
      `http://localhost:3333/posts?_page=${pageParam}&_limit=10`,
    );

    const response = await fetch(fetchUrl.href);
    return response.json() as Promise<IPost[]>;
  },
  initialPageParam: 0,
  getNextPageParam: (_lastGroup, groups) => groups.length,
  refetchOnWindowFocus: false,
});

const onScroll = (e: React.UIEvent<HTMLDivElement>) => {
  const { scrollTop, clientHeight, scrollHeight } = e.currentTarget;
  if (scrollTop + clientHeight >= scrollHeight - 100 && !isFetchingPosts) {
    fetchNextPage();
  }
};
```

Options

- initialPageParam
- getNextPageParam
- getPreviousPageParam
- ...useQueryOptions

Returns

- fetchNextPage,
- fetchPreviousPage,
- hasNextPage,
- hasPreviousPage,
- isFetchingNextPage,
- isFetchingPreviousPage,
- ...useQueryResult

CODE AND SLIDES