**Installation of React Query**

npm i react-query

**React Query DevTools**

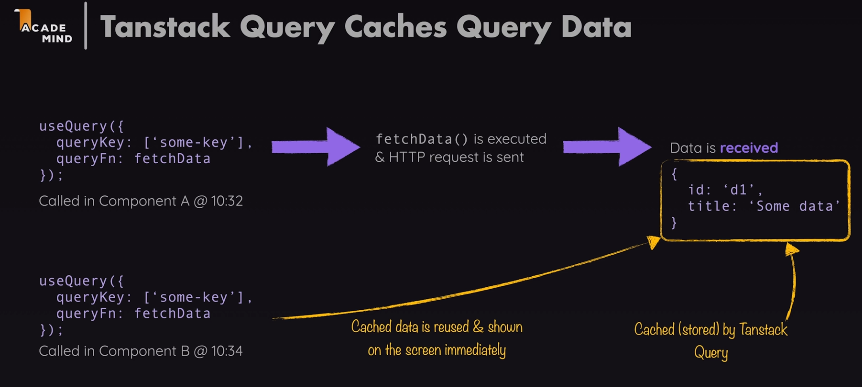
npm i react-query-devtools

Reference 🡪<https://www.youtube.com/watch?v=PJSVowvL2MU&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=5>

**Why do we need to use React Query?**

* Caching... (possibly the hardest thing to do in programming)
* Deduping multiple requests for the same data into a single request
* Updating "out of date" data in the background
* Knowing when data is "out of date"
* Reflecting updates to data as quickly as possible
* Performance optimizations like pagination and lazy loading data
* Managing memory and garbage collection of server state
* Memoizing query results with structural sharing

**How React Query performs deduping?**



React Query performs deduping using a concept called "query keys." A query key is a unique identifier that represents a specific API endpoint along with its query parameters. When a component needs to fetch data using React Query, it provides a query key to the useQuery hook or the queryClient.fetchQuery method.

Here's how deduping works in React Query:

* When a new request is made, React Query checks if there is an existing request in progress or cached data for the same query key.
* If an ongoing request for the same query key is found, React Query does not trigger a new network request. Instead, it piggybacks on the existing request and returns a promise that will resolve when the ongoing request is completed.
* If the query key is not found in the cache, a new network request is made, and the response data is stored in the cache, indexed by the query key.

By using this approach, React Query avoids redundant requests for the same data, which can happen when multiple components mount or update simultaneously and trigger the same query with the same query key.

**How React Query does Query Cache?**

There is a concept of Background refetching which is not present in RTK Query, and I didn’t like this concept. But that can be resolved with the help of staleTime field.

Reference 🡪 <https://www.youtube.com/watch?v=2TX8ojaSwF0&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=6>

**Use of staleTime and cacheTime**

staleTime helps to prevent background refetching. The default staleTime is 0s.

cacheTime is normally for 5mins (30000ms)

Reference 🡪 <https://www.youtube.com/watch?v=0BtcMLJ_Zdc&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=7>

**Refetch Defaults**

refetchOnMount, reFetchOnWindowFocus

Reference 🡪 <https://www.youtube.com/watch?v=A3gN4ji5p6E&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=8>

**Retry in React Query**

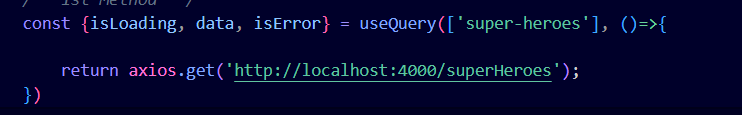
<https://tanstack.com/query/v4/docs/react/guides/query-retries>

**Queries, Query Keys and Query Functions**

To subscribe to a query in your components or custom hooks, call the useQuery hook with at least:

* A unique key for the query also called the Query Key.
* A function also called Query Function that returns a promise that:
* Resolves the data, or
* Throws an error

**1st Method**



**2nd Method**

A computer screen with text and numbers

Description automatically generated



**3rd Method**

A computer screen with text and numbers

Description automatically generated



Queries 🡺 <https://tanstack.com/query/v3/docs/react/guides/queries>

Query Keys 🡺 <https://tanstack.com/query/v3/docs/react/guides/query-keys>

Query Functions 🡺 <https://tanstack.com/query/v3/docs/react/guides/query-functions>

**isLoading and isFetching**

isLoading is a flag which gets true for the first time when the cache is empty and the api needs to be called to fil the cache and after the cache gets filled by the data then isLoading will go to false.

isFetching is a flag that will be false for the first time when the cache is empty. But from the next time the data will come from the cache and there will be background refetching which will make isFetching to true and after updating the cache isFetching goes to false.

Reference 🡪 <https://www.youtube.com/watch?v=2TX8ojaSwF0&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=6>

**The Query Configuration Object & Aborting Request**

A computer code with text

Description automatically generated with medium confidence

Here if you normally pass searchItem it will throw error. We have properly destruct the object and signal value should be preset on that signal otherwise will throw error.

Reference 🡪 <https://cognizant.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39499478#overview>

**Showing error message**

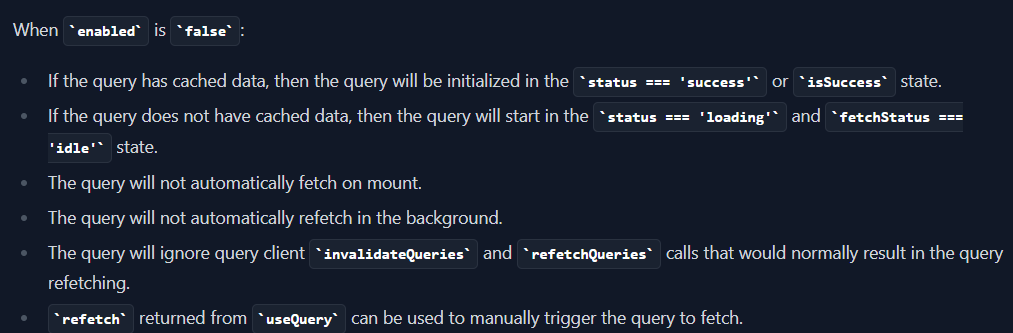


**disabled in React Query Dev Tools / enabled:true**

A close up of a text

Description automatically generated

If you ever want to disable a query from automatically running, you can use the enabled = false option.



Reference 🡪

<https://tanstack.com/query/v4/docs/react/guides/disabling-queries>

**Fetch Status**

In addition to the status field, the result object, you will also get an additional fetchStatusproperty with the following options:

fetchStatus === 'fetching' - The query is currently fetching.

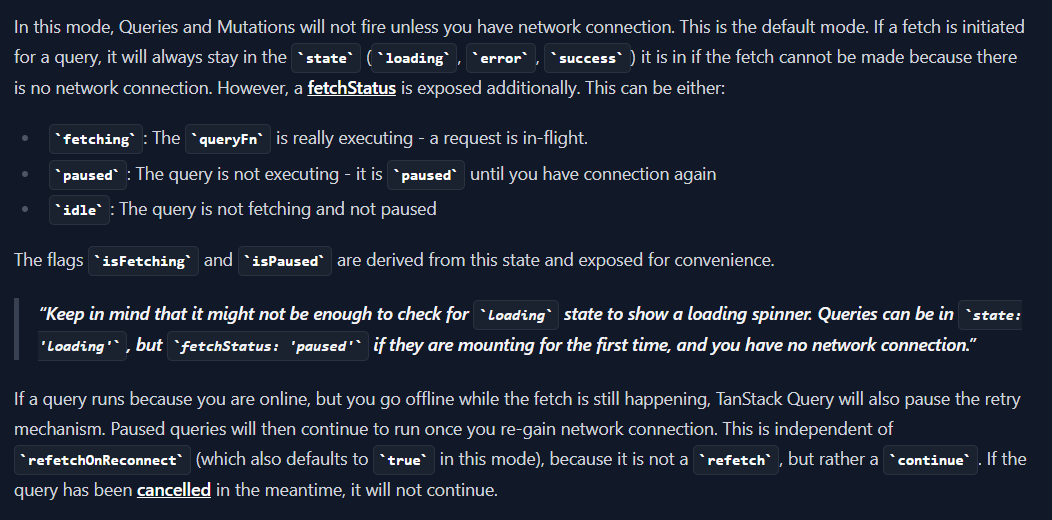
fetchStatus === 'paused' - The query wanted to fetch, but it is paused. Read more about this in the [Network Mode](https://tanstack.com/query/v4/docs/react/guides/network-mode) guide.

fetchStatus === 'idle' - The query is not doing anything at the moment.

Background refetches and stale-while-revalidate logic make all combinations for status and fetchStatus possible. For example:

* a query in success status will usually be in idle fetchStatus, but it could also be in fetching if a background refetch is happening.
* a query that mounts and has no data will usually be in loading status and fetching fetchStatus, but it could also be paused if there is no network connection.
* So keep in mind that a query can be in loading state without actually fetching data. As a rule of thumb:
  + The status gives information about the data: Do we have any or not?
  + The fetchStatus gives information about the queryFn: Is it running or not?

**Network Mode : Online affects the status**



**Lazy Queries**

A screenshot of a computer program

Description automatically generated

**Polling in react query**

Polling basically takes care of calling an api after a specified interval, say you have a stock market application and you want your data to be getting updated in the UI after every 2sec.

refetchInterval is by default false.

refetchIntervalInBackground tells that when the browser tab is not in focus and in the background it will not fetch data.

It is by default false.

Reference 🡪

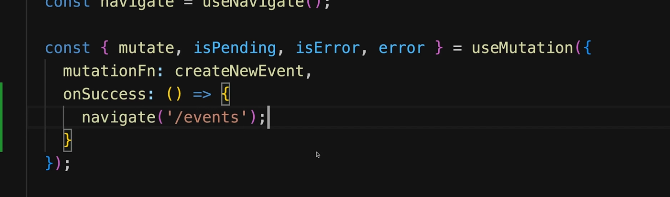
<https://www.youtube.com/watch?v=PvcRaMqtEPQ&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=9>

**Querying the data on any event**

**If you are using useQuery on event then initially during the mount the data will be undefined, and need to handle that properly in the code.**

Reference 🡪 <https://www.youtube.com/watch?v=oX3HTT5e9SQ&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=10>

**Success and Error Callback in React Query**



<https://www.youtube.com/watch?v=Z37JtB78mNo&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=11>

**Data Transformation or mapping the data coming from backend into a variable in the fronend**

<https://www.youtube.com/watch?v=fbIb0m_GhlU&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=12>

**Custom Query Hook**

<https://www.youtube.com/watch?v=Q12EVU8YpTY&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=13>

**Query By Id and giving the query in the Query Key**

const fetchSuperHero = ({queryKey})=>{

    const heroId = queryKey[1];

    return axios.get(`http://localhost:4000/superHeroes/${heroId}`)

}

export const useSuperHeroData = (heroId)=>{

*// console.log(heroId);*

    return useQuery({queryKey:['super-heroes',heroId],queryFn:fetchSuperHero})

}

<https://www.youtube.com/watch?v=2s2iJLLDwgk&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=14>

**Parallel Queries and Dynamic Parallel Queries**

Parallel Queries means when we want to query more than one endpoint simultaneously.

Parallel Queries 🡪 <https://www.youtube.com/watch?v=gBCbPpuqnRk&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=15\>

Dynamic Parallel Queries 🡪 <https://www.youtube.com/watch?v=yOjHT-oTFww&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=16>

**Dependent Queries**

One query is dependent on the result of another query.

<https://www.youtube.com/watch?v=HH68tHPq7tA&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=17>

**Paginated Queries**

keepPreviuosData

Reference 🡪 <https://www.youtube.com/watch?v=xoPguAXJmiE&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=19>

**Infinite Scrolling with Infinite Queries**

<https://www.youtube.com/watch?v=s92apk05kT4&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=20>

**Mutation**

**Normal Post request**

<https://www.youtube.com/watch?v=NYCG1o38oEQ&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=21>

**Refetching by Query Invalidation**

Refetching is actually a post/put/delete request followed by the get request.



When you do a mutation and invalidate the keyword that you pass in the queryKey will invalidate all the queries that matches that queryKey. It can be bypassed with keyword called “exact”.

Timestamp 🡪 5:30

Reference 🡪 <https://cognizant.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39499474#overview>

Now sometimes automatic refeteching lead to issues that can be solved in this way :

Reference 🡪 <https://cognizant.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39499480#overview>

This refetching will not at all work if the enabled is “false”. Written in the disabled in React Query Dev Tools

Reference 🡺 <https://www.youtube.com/watch?v=ldg3QIT53pI&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=22>

**Handling Mutation by not making a get request after the post request and just adding the data in the cache**

It should be setQueriesData().

Reference of Doc 🡪 <https://tanstack.com/query/v4/docs/react/reference/QueryClient#queryclientsetquerydata>

Reference of Video 🡪 <https://www.youtube.com/watch?v=XI0SN5AI6YA&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=23>

**Optimistic Updates**

Part 1 🡪 <https://www.youtube.com/watch?v=rnN5ng6aoAc&list=PLC3y8-rFHvwjTELCrPrcZlo6blLBUspd2&index=24>

Part 2 🡪 <https://cognizant.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39499488#overview>

Using onSettled() is very important. The function within onSettled() will run no matter the mutation failed or succeeded.

**To check how to deal with update and delete**

Check videos from 379 to 384.

Reference 🡪 <https://cognizant.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39499478#overview>

**How to perform optimized search by caching the search item as query key**

TimeStamp 🡪 4:31

Reference 🡪<https://cognizant.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39499434#overview>

It can also lead to abort request.

Reference 🡪 <https://cognizant.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39499460#overview>