React Query

⟶ React query is the library/package which helps us to fetch the data from an api in simpler way.

⟶ First of all we will create our local api using the `json-server` package.

For that create one file called `db.json` and enter some json text in array which we will get when we call the api.

For example :

```
{
 "superheros": [
     {
         "id": 1
         "name": "Ironman",
         "power": 69420
     },
     {
         "id": 2,
         "name": "Superman",
         "power": 456
     },
     {
         "id": 3,
         "name": "Spiderman",
         "power": 69
     }
 ]
}
```

⟶ after that enter this text in `package.json`'s script

```
"serve-json": "json-server --watch db.json --port 4000"
```

⟶ And start the API server with the command `npm run serve-json`

React query package

⟶ to install react query use this command `npm i react-query`

After that you have to import 2 hooks in `App.js` or the other main file.

```
import { QueryClientProvider, QueryClient } from "react-query";
```

⟶ Write your code between `<QueryClientProvider>` and `</QueryClientProvider>`

After that make new QueryClient using :

```
const queryClient = new QueryClient();
```

Then add the `client` property for `<QueryClientProvider>`

```
<QueryClientProvider client={queryClient}>
```

⟶ Then goto the component in which you want to fetch the data and add the `useQuery()` hook

```
import { useQuery } from "react-query";
```

⟶ The syntax for `useQuery` is like this :

```
const { isLoading, data } = useQuery("super-heros", () => {

    return axios.get("http://localhost:4000/superheros");

 });
```

⟶ Here `super-heros` is the key and we are generating the `GET` request with `axios` and storing the data in `data`

Here `isLoading` will be true during fetching the data so we can display loading screen.
`data.data` contains the data in the form of array so we will map it.

```
data?.data.map((ele)=>{
    return <p key={ele.name}>{ele.name}</p>
})
```

Error handling

⟶ In the traditional way we do the error handling with `.catch()` Method

```
useEffect(() => {
  axios
    .get("http://localhost:4000/superheros")
    .then((res) => {
      setData(res.data);
      setIsLoading(false);
    })
    .catch((error) => {
      setError(error.message);
      setIsLoading(false);
    });
}, []);
```

```
if (error) {
 return <h2>{error}</h2>;
}
```

⟶ In the react query we can use `isError` and `error` methods to display error

```
const { isLoading, data, isError, error } = useQuery("super-heros", (
 return axios.get("http://localhost:4000/superheros1");
});

if (isError) {
 return <h3>{error.message}</h3>;
}
```

⟶ and we can see the results

Home Traditional Super Heroes RQ Super Heroes

Request failed with status code 404

---

devtools

⟶ to use react query devtools you have to import it first :

```
import { ReactQueryDevtools } from "react-query/devtools";
```

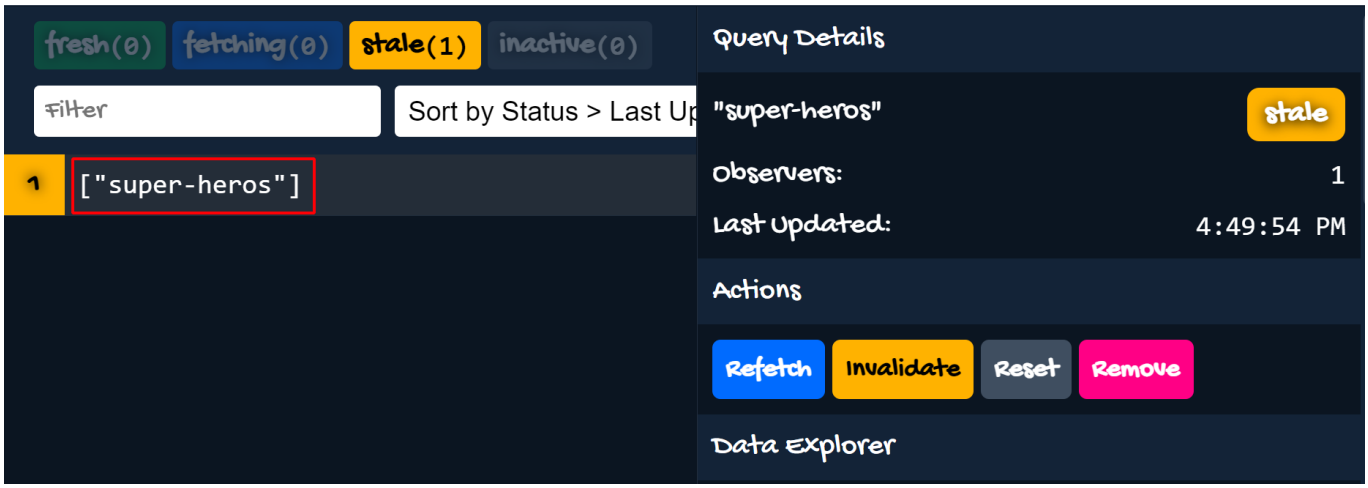
⟶ After that use the component `<ReactQueryDevtools/>` before the closing tag for `QueryClientProvider`

You can use different props in this component like :

```
<ReactQueryDevtools initialIsOpen={false} position="bottom-right" /
```

And we can see the devtools in the webpage at bottom-right

When you open it you can see one request which we made on our API



⟶ here `Super-heros` is the key which we provided in `useQuery` function

Query cache

⟶ There is a very good feature in react query which is `query cache`. in this the query stores the response in cache and if user again try to fetch the response then they will receive the same response which is stored in cache and it will don't show loading screen.

⟶ But if there are any changes in response after first fetch then the react query will show the old data first to the user without showing loading screen but it will fetch the data in the background and once it will receive the data then it will update it in DOM.

⟶ to check this you can use `isFetching` method.

the default time for cache is `5 min` but if you want to change it then you can pass one more argument to the `useQuery()` as an object and change the cache time.

Example :

```
const { isLoading, data, isError, error } = useQuery("super-heros", (
    return axios.get("http://localhost:4000/superheros"),{
        cacheTime : 5000,
    };
});
```

Stale time

⟶ if you want to fetch after certain time then you can use `staleTime` property in object in `useQuery` hook.

⟶ if you set the `staleTime` as `30000` then it will fetch the data after every 30 seconds and it will not fetch the data in background too.

The default stale time is `0 seconds`

`refetchOnMount` → this is by default `true` and if you make it false then the react will not fetch the data in the background.

`refetchOnWindowFocus` → this value is also `true` by default but this property is very useful because in traditional way if there are any changes in the response then user have to refresh the page to see the changes but in the `refetchOnWindowFocus` user don
t need to refresh the page. whenever user focus on the tab the fetch will trigger.

⟶ There is a one more property which is `always` which means it will always fetch the data if the stale time is there or not.

Polling

⟶ If you want to fetch the data at every interval of time then you can use the `refetchInterval` property. By default it's set to false but you can set any number to it (Number is in miliseconds).

⟶ but this will only work till the user focuses on the window. But if you want to fetch the data in the bg then you can use `refetchInBackground` property to `True`

Fetch on user event

⟶ If you want to fetch the data on the click of any button then first of all you have to disable the onfetchMount by doing

`enabled:false` so that it will not fetch the data.

⟶ Now just create one button

```
<button>Fetch data</button>
```

but what will be the method after `onClick` ?

⟶ For that you have to add one more property before the `useQuery` which is `refetch` and add this in button after `onClick`

```
<button onClick={refetch}>Fetch data</button>
```

---

onSuccess and onError

⟶ If you want to trigger any function on error or on success of fetching then you can use `onSuccess` and `onError` properties.

Example :

```
const success = ()⇒{
    console.log("Data fetched successfully");
}

const error = ()⇒{
    console.log("An error occured");
}

// After that just add 2 properties
{
    onSuccess:success,
    onError:error
}
```

⟹ NOTE : if there is an error then query will try to fetch the data 3 times and if it still gets error then it will show error of 404

⟶ you can pass the `data` and `error` as parameter in the functions

Example :

```
const success = (data)⇒{
    console.log("Data fetched successfully",data);
}

const error = (error)⇒{
    console.log("An error occured",error);
}
```

Data transformation

⟶ If you want the `superhero.name` directly in the data then you can use the property `select` to specify what should be in data.

Example :

```
select: (data) ⇒ {
 const superNames = data.data.map((hero) ⇒ {
    return hero.name;
 });
 return superNames;
}

// Then simply map it in DOM

{data.map((hero) ⇒ {
```

```
 return <div key={hero}>{hero}</div>;
})}
```

---

making custom hook

⟶ If you want the same query data in different components then you can make your own custom query hook

Just make one file and add this :

```
import { useQuery } from "react-query";
import axios from "axios";

export const useSuperheroData = (successFun, errorFun) => {
  return useQuery(
    "super-heros",
    () => {
      return axios.get("http://localhost:4000/superheros");
    },
    {
      onSuccess: successFun,
      onError: errorFun,
      select: (data) => {
        const superNames = data.data.map((hero) => {
          return hero.name;
        });
        return superNames;
      },
    }
  );
};
```

And call the query with the function call with needed parameters :

```
const { isLoading, data, isError, error, refetch } = useSuperheroData(
 successFun,
 errorFun
);
```

```
);
```

Query by id

⟶ If you want to Fetch the data for individual object by id then you can make a dynamic link using `react-router-dom` and `/:heroId` where `:` represents dynamic data.

```
<Route path="/rq-super-heros/:heroId">
 <RQSinglehero />
</Route>
```

Then just render it with map method.

```
{data?.data.map((ele) => {
 return (
     <div key={ele.id}>
     <Link to={`/rq-super-heros/${ele.id}`}>{ele.name}</Link>
     </div>
 );
})}
```

⟹ Json-server also provides one functionality in which we can get the data for individial object using id.

For example if the json-server is running on `http://localhost:4000/superheros` then you can get the info about individiual hero by going to the `http://localhost:4000/superheros/1` where 1 is dynamic.

⟶ For that we can create the custom hook and add this code (here i have called it `useFullData`) :

```
import { useQuery } from "react-query";
import axios from "axios";

const fetchDetails = (heroId) => {
  return axios.get(`http://localhost:4000/superheros/${heroId}`);
};

const useFulldata = (heroId) => {
  return useQuery(["super-hero", heroId], () => fetchDetails(heroId));
};

export default useFulldata;
```

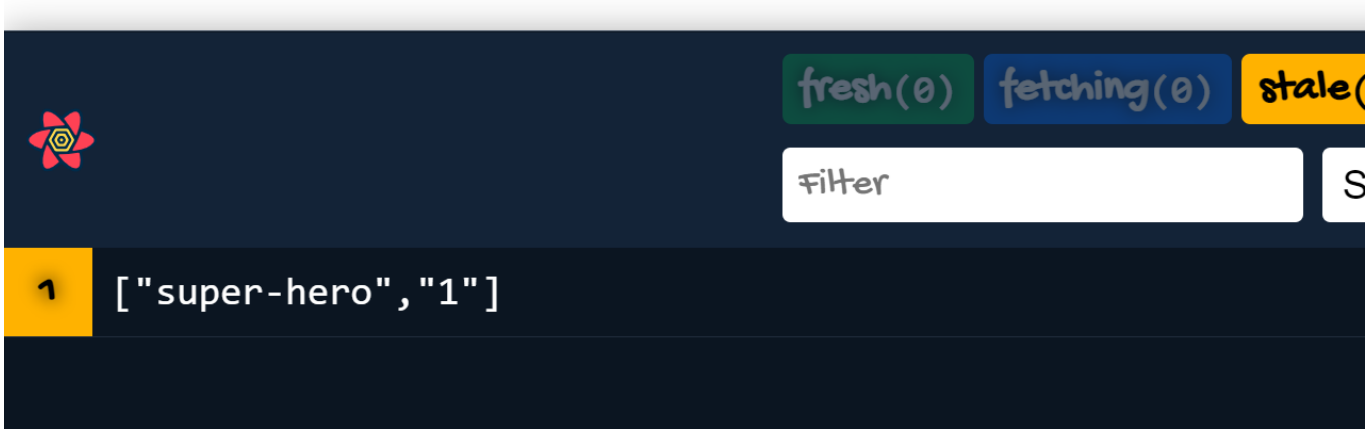
⟶ After that add this code in the component from which we want to fetch the data :

```
import React from "react";
import { useParams } from "react-router";
import useFulldata from "../Hooks/useFulldata";

function RQSinglehero() {
  const { heroId } = useParams();
  const { isLoading, data, isError, error } = useFulldata(heroId);
  if (isLoading) {
    return <h2>Loading ...</h2>;
  }

  if (isError) {
    return <h2>{error.message}</h2>;
  }
  return (
    <div>
      <p>
        {data?.data.name} - {data?.data.power}
      </p>
    </div>
  );
}
```

⟶ And we can see the output :

Home Traditional Super Heroes RQ Super Heroes

Ironman - 69420



⟹ If you don't want to pass the parameter `heroId` then you can use the `queryKey` which we have provided in `useQuery` . but here it's an array so we have to do like `queryKey[1]` because `heroId` is at index 1.

⟶ So now `fetchDetails` function will be look like this :

```
const fetchDetails = ({ queryKey }) ⇒ {
 const heroId = queryKey[1];
 return axios.get(`http://localhost:4000/superheros/${heroId}`);
};
```

And now there is no need to pass the arrow function in `useQuery`, just pass the function name instead.

Dynamic parallel query

⟶ If you want to fetch multiple data at the same time parallely then you can use `useQueries()` hook. This hook is similar to the `useQuery` hook.

I have made one component for dynamic parallel fetching and added this code :

```
import { useQueries } from "react-query";
import axios from "axios";

const fetchData = (heroid) => {
 return axios.get(`http://localhost:4000/superheros/${heroid}`);
};
const DynamicParallel = ({ heroIds }) => {
 const queryResults = useQueries(
     heroIds.map((id) => {
         return {
         queryKey: ["super-heros-dynamic", id],
         queryFn: () => fetchData(id),
         };
 })
 );
 console.log({ queryResults });
 return <div>This is a dynamic query page</div>;
};

export default DynamicParallel;
```

⟶ here `querykey` defines the key which is array of keys and `queryFn` is the fetching function.

Here we will get the array of heroIds as a prop and then fetch them one by one parallely

⟹ Code for route :

```
<Route path="/rq-dynamic-heros">
 <DynamicParallel heroIds={[1, 3]} />
</Route>
```

Dependent queries

⟶ Suppose we have this data in API

```
"users": [
    {
        "id": 1,
        "name": "Krunal",
        "email": "krunal@example.com"
    },
    {
        "id": 2,
        "name": "Furqan",
        "email": "furqan@example.com"
    },
    {
        "id": 3,
        "name": "Swapnil",
        "email": "swapnil@example.com"
    },
    {
        "id": 4,
        "name": "Parth",
        "email": "parth@example.com"
    }
 ],
 "courses": [
    {
```

```
        "id": "Furqan",
        "course": ["DS", "AI"]
    },
    {
        "id": "Krunal",
        "course": ["Java", "Assembly"]
    }
]
```

⟶ And we have to fetch the `courses` for user `Furqan`

So here we will first fetch the user id from `users` and then we will get the names from `users` also and in `courses` the id is the name of the user so first we have to fetch the users and then we will fetch the courses by that name from courses.

⟶ So here we can see that both queries are dependent on each other.

⟹ First we will create a new component and add this code :

```
import { useQuery } from "react-query";
import axios from "axios";

const fetchUsersById = (id) => {
 return axios.get(`http://localhost:4000/users/${id}`);
};

const fetchCoursesById = (id) => {
 return axios.get(`http://localhost:4000/courses/${id}`);
};

const DependentQuery = ({ userId }) => {
 const { data: user } = useQuery(["user", userId], () =>
 fetchUsersById(userId)
);
```

```
const id = user?.data.name;

 const { data, isLoading } = useQuery(
     ["courses", id],
     () => fetchCoursesById(id),
     {
         enabled: !!id, // !! converts the value into boolean becase
     }
 );

 if (isLoading) {
     return <p>Loading ...</p>;
 }
 return (
 <>
     <div>This is a DependentQuery page</div>
     <p>
         {user?.data.name} has{" "}
         {data?.data.course.map((course) => {
         return <span>{course} </span>;
         })}
     courses
     </p>
 </>
 );
};
export default DependentQuery;
```

⟶ Here we need to fetch the `users` first but if we make query for them then it will fetch both and we can't fetch because the `id` for `courses` is undefined. so we have to disable fetching for `courses` query and we can do this by using `enabled` property.

⟹ enabled property only accepts boolean value that's why i have written `!!` before `id` which converts it into boolean.