



TANSTACK QUERY



TANSTACK QUERIES

QUERY

A query is a declarative dependency on an asynchronous source of data that is tied to a unique key. A query can be used with any Promise based method (including GET and POST methods) to fetch data from a server

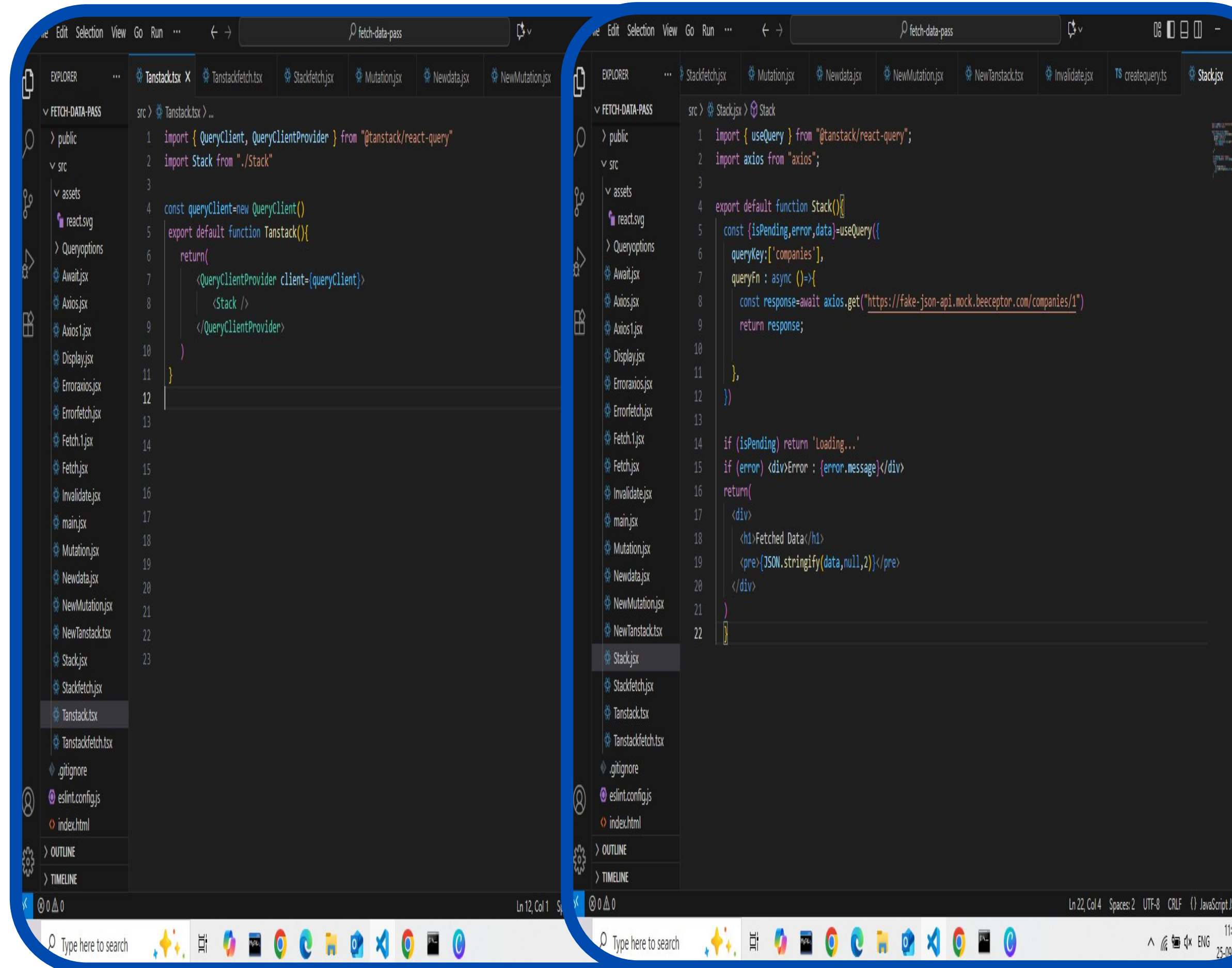
MUTATION

Mutations are typically used to create/update/delete data or perform server side-effects. For this purpose, TanStack Query exports a useMutation hook.

INVALIDATE

In React Query, invalidating a query means marking it as "stale," which tells React Query that the data associated with that query is potentially outdated and should be re-fetched.

EXAMPLE OF TANSTACK QUERY



FETCHING API

FETCH

Fetching data in React applications often involves using the fetch API, a native JavaScript interface for making HTTP requests

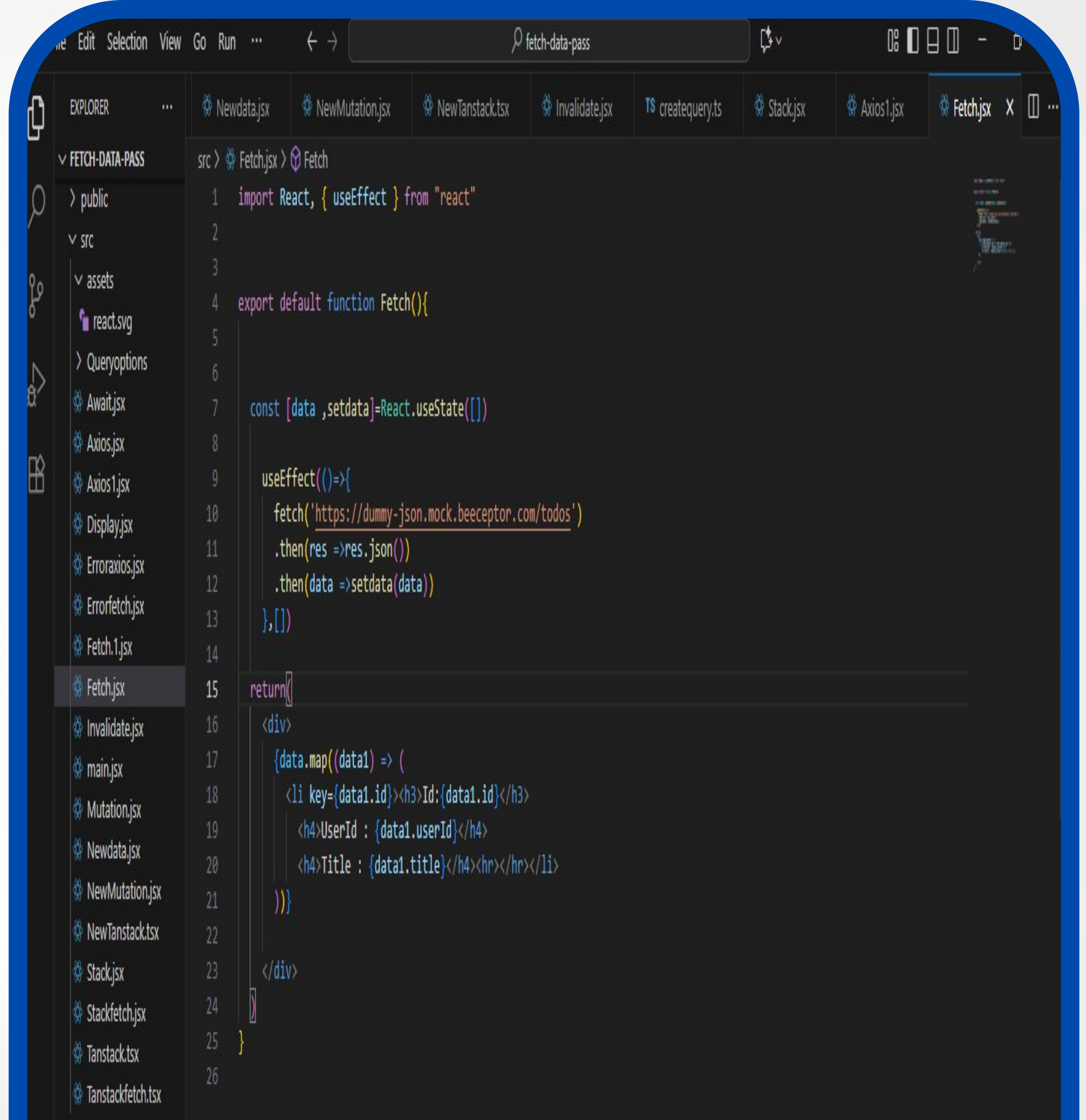
AXIOS

Axios is a popular, promise-based HTTP client used in React applications to make asynchronous requests to APIs or other backend services

TANSTACK QUERY

TanStack Query, formerly known as React Query, is a powerful data-fetching library for React applications.
The process of fetch ,caching and updating

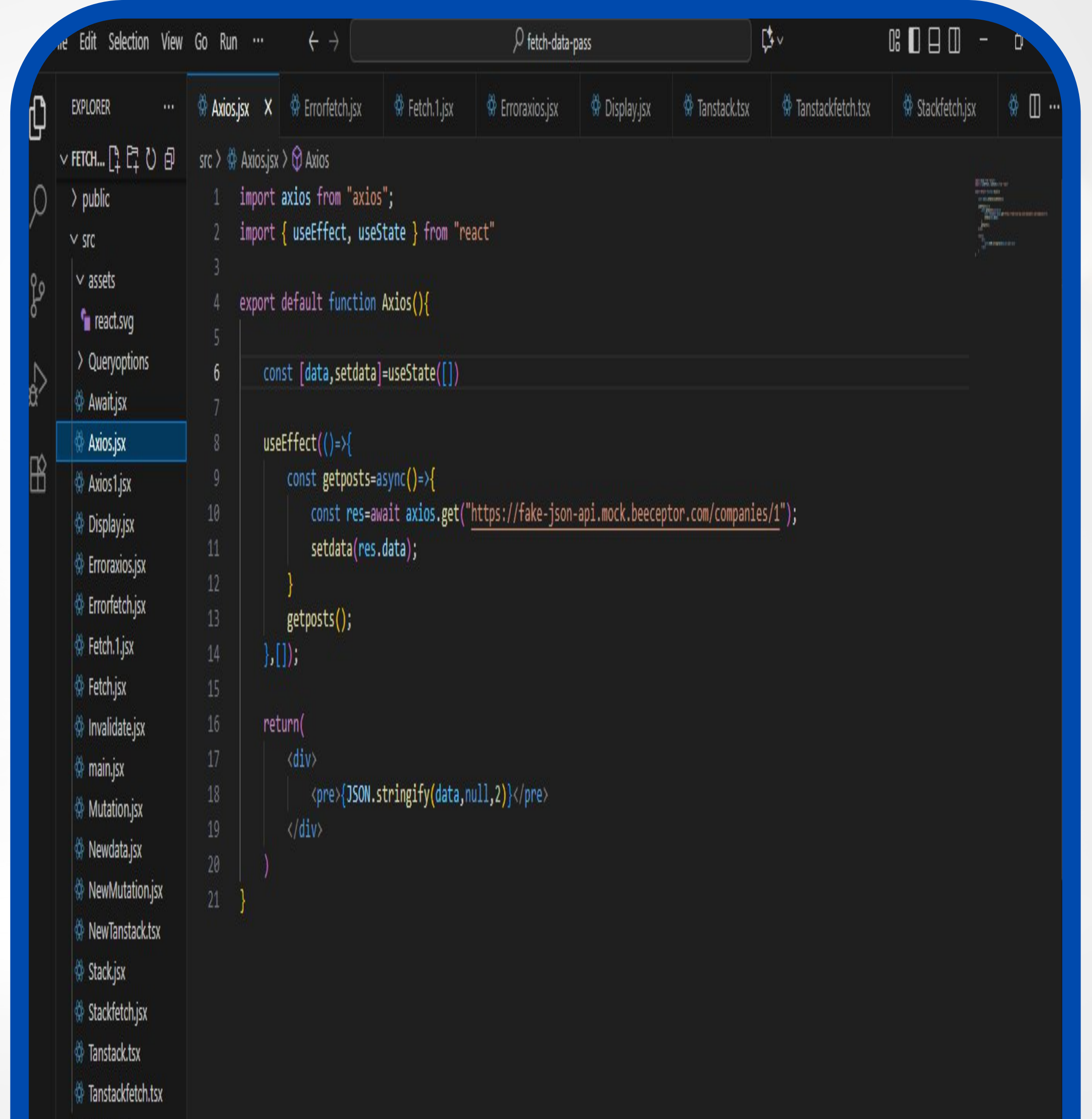
EXAMPLE OF FETCH DATA



The screenshot shows a VS Code editor window with a project named 'fetch-data-pass'. The Explorer sidebar on the left shows the file structure: 'public' and 'src' folders. The 'src' folder contains 'assets' (with 'react.svg'), 'Queryoptions', and several JavaScript files including 'Fetch.jsx'. The 'Fetch.jsx' file is selected and open in the editor. The code in 'Fetch.jsx' is as follows:


```
1 import React, { useEffect } from "react"
2
3
4 export default function Fetch(){
5
6
7   const [data ,setdata]=React.useState([])
8
9   useEffect(()=>{
10     fetch('https://dummy-json.mock.beeceptor.com/todos')
11       .then(res =>res.json())
12       .then(data =>setdata(data))
13   },[])
14
15   return(
16     <div>
17       {data.map((data1) => (
18         <li key={data1.id}><h3>Id:{data1.id}</h3>
19         <h4>UserId : {data1.userId}</h4>
20         <h4>Title : {data1.title}</h4><hr></hr></li>
21       ))}
22     </div>
23   )
24 }
25
26
```

EXAMPLE OF AXIOS



```
1 import axios from "axios";
2 import { useEffect, useState } from "react"
3
4 export default function Axios(){
5
6   const [data, setdata] = useState([])
7
8   useEffect(() => {
9     const getposts = async () => {
10       const res = await axios.get("https://fake-json-api.mock.beeceptor.com/companies/1");
11       setdata(res.data);
12     }
13     getposts();
14   }, []);
15
16   return(
17     <div>
18       <pre>{JSON.stringify(data, null, 2)}</pre>
19     </div>
20   )
21 }
```

Background sync in React, particularly within Progressive Web Apps (PWAs), enables web applications to defer actions, such as sending data to a server, until the device has a stable network connection. This ensures a more robust and offline-capable user experience.



Background Sync

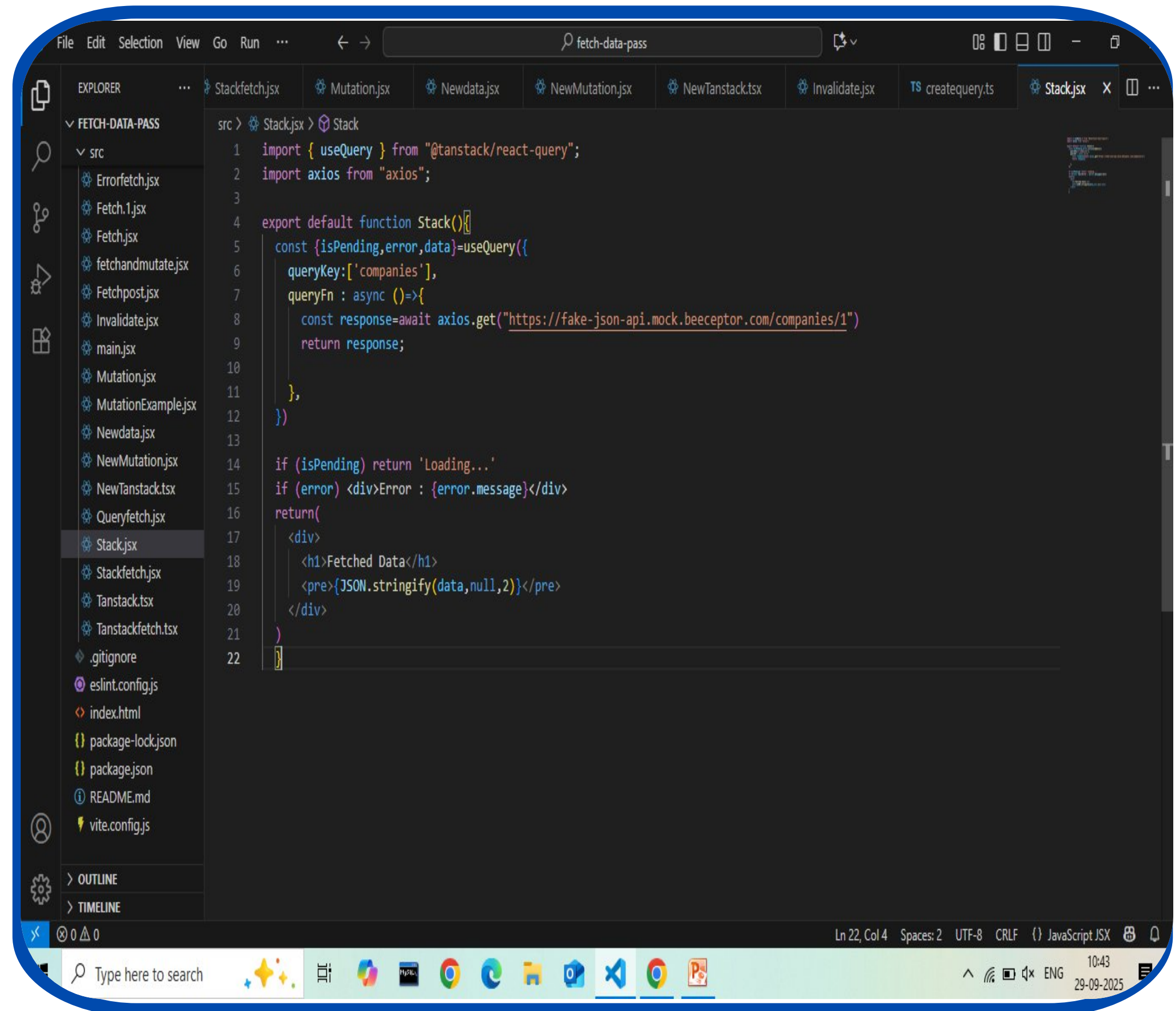
Handling loading, error, and success states in a React component using `useQuery` from React Query involves destructuring the relevant properties returned by the hook.

The most common ones for handling loading, error, and success are:

1. **isLoading** or **isPending** : A boolean indicating if the query is currently fetching data for the first time. `isPending` is the newer, preferred term.
2. **isError** : A boolean indicating if the query encountered an error.
3. **Error** : Contains the error object if `isError` is true.

Handling Error, Loading,
Success

EXAMPLE OF Handling Error, Loading Success



The screenshot shows a Visual Studio Code editor window with a project named "fetch-data-pass". The Explorer sidebar on the left shows a file tree with a "src" directory containing various files, including "Stack.jsx" which is currently selected. The main editor area displays the code for "Stack.jsx". The code uses the `useQuery` hook from `@tanstack/react-query` to fetch data from a mock API. It includes error handling and loading state management.

```
1 import { useQuery } from "@tanstack/react-query";
2 import axios from "axios";
3
4 export default function Stack() {
5   const { isPending, error, data } = useQuery({
6     queryKey: ['companies'],
7     queryFn: async () => {
8       const response = await axios.get("https://fake-json-api.mock.beeceptor.com/companies/1");
9       return response;
10     },
11   });
12
13   if (isPending) return 'Loading...'
14   if (error) <div>Error : {error.message}</div>
15   return (
16     <div>
17       <h1>Fetched Data</h1>
18       <pre>{JSON.stringify(data, null, 2)}</pre>
19     </div>
20   )
21 }
22
```

The status bar at the bottom indicates the current line and column (Ln 22, Col 4), the number of spaces (2), the encoding (UTF-8), the line ending (CRLF), and the file type (JavaScript JSX).

Stale Time	Cache Time
<p>Definition:</p> <p>Stale time defines how long data is considered "fresh" or valid before it needs to be re-fetched or revalidated from its source.</p>	<p>Definition:</p> <p>Cache time (or gcTime) determines how long data remains in the cache after it becomes unused (i.e., no longer actively observed or referenced by any part of the application).</p>
<p>Purpose:</p> <p>It dictates when a cached item should be considered potentially outdated. Once the stale time expires, the data is marked as stale, and subsequent requests for that data will trigger a re-fetch or revalidation attempt to ensure data currency.</p>	<p>Purpose:</p> <p>It manages the memory footprint of the cache by specifying when inactive data should be eligible for garbage collection and removal from memory</p>
<p>Impact:</p> <p>A shorter stale time means data is refreshed more frequently, ensuring higher data accuracy but potentially leading to more network requests. A longer stale time reduces network requests but increases the risk of serving outdated information.</p>	<p>Impact:</p> <p>A shorter cache time frees up memory more quickly but might require re-fetching data if it's needed again shortly after becoming inactive. A longer cache time keeps data readily available for longer but consumes more memory.</p>
<p>Behavior:</p> <p>Serves cached data, and if stale, triggers an immediate background refetch upon a new query.</p>	<p>Behavior:</p> <p>Deletes inactive data completely from the cache after the time limit expires.</p>

A top-down view of a desk with a laptop, a cup of coffee, a pen, glasses, paper clips, and a large monstera leaf.

THANK YOU