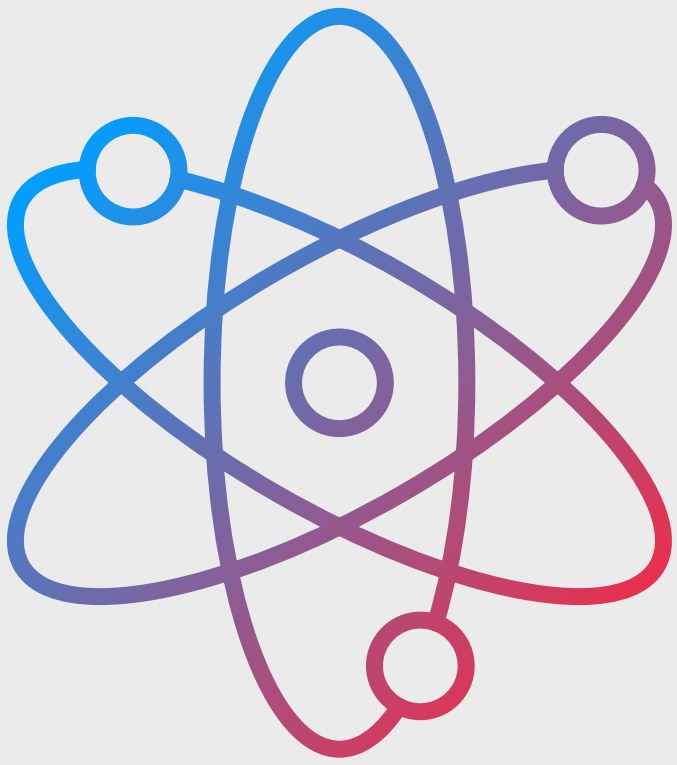
DIFFERENT WAYS TO FETCH DATA IN REACT JS



Ibtissam Oulahchoum

Swipe to know

FETCH API

The Fetch API is a built-in JavaScript feature used to send requests to a server and load the returned data into web pages dynamically. It can request data from any API, and the response is typically in JSON or XML format. The Fetch method returns a Promise, which resolves when the request completes, providing access to the response data. This method also allows for powerful options like specifying HTTP methods (GET, POST, etc.), handling headers, and even sending data in the body of the request. With the Fetch API, you have full control over handling success or failure, making it a versatile and modern way to manage data retrieval in web applications.

```
function App() {

  useEffect(() => {
    fetch('https://site.com/')
    .then(response => response.json())
    .then(json => console.log(json))
    }, []);

  return (
    <div> Different ways to fetch Data </div>
  );
}
```

AXIOS LIBRARY

Axios is a powerful JavaScript library used to make HTTP requests from the browser or Node.js. It simplifies the process of interacting with APIs by providing an intuitive API and additional features beyond what's offered by the native Fetch API. Axios automatically transforms JSON data, making it easy to work with API responses. It also supports features like request cancellation, timeouts, interceptors for requests and responses, and default headers, making it more flexible and user-friendly.

Additionally, Axios provides better support for older browsers and can be used with async/await for cleaner asynchronous code. Whether you need to send GET, POST, or other types of HTTP requests, Axios is a reliable and robust choice for modern web developm

```
function App() {

  useEffect(() => {
    axios.get("https://site.com")
      .then((response) => console.log(response.data));
    }, []);

  return (
    <div>
        Different ways to fetch Data
    </div>
  );
}
```

REACT QUERY

React Query is a library for managing server state in React applications. It simplifies data fetching, caching, and synchronization by providing powerful features like automatic caching, background data updates, and query invalidation. With hooks like useQuery, React Query handles loading, error states, and data management, making it easier to work with asynchronous data and improving performance and developer experience.

```
import axios from 'axios'
import { useQuery } from 'react-query'

function App() {
  const { isLoading, error, data } =
  useQuery('posts', () ⇒ axios('https://site.com'))

  console.log(data)

  return <div>Different ways to fetch data</div>
}
```

USING ASYNC/AWAIT

Using Async/Await with Fetch simplifies handling asynchronous operations in React components. The fetch API allows you to make HTTP requests, and using async/await makes the code cleaner and more readable compared to traditional promise chaining.

```
import useFetch from "./useFetch";

const App = () => {
  const { isLoading, serverError, apiData } = useFetch('https://site.com')

  return (
    <div>
      <h2>API data</h2>
      {isLoading && <span>Loading.....</span>}
      {!isLoading && serverError ? (
        <span>Error in fetching data ... </span>
      ) : (
        <span>{JSON.stringify(apiData)}</span>
      )}
    </div>
  )
}
```

CUSTOM HOOK

Custom Hooks for Data Fetching in React simplify data retrieval by encapsulating the logic into reusable functions. For example, useFetch manages API requests, loading states, and errors, keeping your component code clean and focused on rendering.

```
const useFetch = (url) => {
  const [isLoading, setIsLoading] = useState(false)
  const [apiData, setApiData] = useState(null)
  const [serverError, setServerError] = useState(null)

  useEffect(() => {
    setIsLoading(true)
    const fetchData = async () => {
      try {
        const resp = await axios.get(url)
        const data = await resp?.data

        setApiData(data)
        setIsLoading(false)
      } catch (error) {
        setServerError(error)
        setIsLoading(false)
      }
    }

    fetchData()
  }, [url])

  return { isLoading, apiData, serverError }
}
```

USAGE IN THE COMPONENT

The useFetch hook is a custom React hook designed to simplify the process of retrieving data from an API. By passing the API endpoint URL to the hook, it automatically handles the process of fetching data, as well as managing states like loading, success, and error. You can then use it within your component just like any other React hook, making it an efficient way to integrate API calls into your application.

```
import useFetch from "./useFetch";

const App = () => {
  const { isLoading, serverError, apiData } = useFetch('https://site.com')

  return (
    <div>
      <h2>API data</h2>
      {isLoading && <span>Loading.....</span>}
      {!isLoading && serverError ? (
        <span>Error in fetching data  ... </span>
      ) : (
        <span>{JSON.stringify(apiData)}</span>
      )}
    </div>
  )
}
```

THANK YOU FOR YOUR TIME !

KEEP CODING ,
KEEP GROWING

Ibtissam Oulahchoum