How Does Data Fetching Work in React?

freecodecamp.org/learn/front-end-development-libraries-v9/lecture-working-with-data-fetching-and-memoization-in-react/how-does-data-fetching-work-in-react

React apps often rely on external APIs and databases for dynamic content. To access the data from those APIs and databases, you need to use some data fetching techniques.

Let's take a look at how data fetching works in React and the different options available to you for fetching data.

React is not opinionated about how you fetch your data, this means on a basic level, you can use the built-in Fetch API, which all modern browsers support.

You can also use Axios and SWR. Axios is promise-based HTTP request library built on top of the XMLHttpRequest object, and SWR is a React hook for data fetching created by the Vercel team.

Let's start with an example. You first need to import the `useState` and `useEffect` hooks:

```
import { useState, useEffect } from "react";
```

Then you will need to create three state variables called `loading`, `data`, and `error`:

```
const [data, setData] = useState(null);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);
```

The `loading` variable will track whether the data is still being fetched. The `data` variable represents the data itself, and the `error` variable will capture any errors that might occur during the data fetching process.

Since data fetching is a side effect, it's best to use the Fetch API inside of a `useEffect` hook.

Here's an example of that:

```
useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/posts")
    .then((res) => res.json())
    .then((data) => {
      setData(data);
      setLoading(false);
    })
    .catch((err) => {
      setError(err);
      setLoading(false);
    });
}, []);
```

This `useEffect` fetches the data with the Fetch API and sets all the states.

You can make things better by using `async`/`await` instead of the `.then()` syntax. That means you have to have a separate function inside the `useEffect` because you cannot prefix `useEffect` with the `async` keyword:

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const res = await fetch("https://jsonplaceholder.typicode.com/posts");

      if (!res.ok) {
        throw new Error("Network response was not ok");
      }

      const data = await res.json();
      setData(data);
    } catch (err) {
      setError(err);
    } finally {
      setLoading(false);
    }
  };

  fetchData();
}, []);
```

You can then go ahead and use all of those states to render the data from the API.

Here's the full code:

```
import { useState, useEffect } from "react";

const FetchPosts = () => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await fetch("https://jsonplaceholder.typicode.com/posts");

        if (!res.ok) {
          throw new Error("Network response was not ok");
        }

        const data = await res.json();
        setData(data);
      } catch (err) {
        setError(err);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  if (loading) {
    return <p>Loading...</p>;
  }

  if (error) {
    return <p>{error.message}</p>;
  }

  return (
    <ul>
      {data.map((post) => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
};

export default FetchPosts;
```

In the UI, you would see `Loading...` on the screen when the data is being fetched, and then the data or error would show depending on if the data fetch was successful.

Remember we talked about data fetching with Axios and SWR too. Let's take a look at an example using Axios.

You will first need to install Axios from the command line like this:

```
npm i axios
```

Then you will need to import Axios like this:

```
import axios from "axios";
```

Then you can use the same state variables from earlier and fetch data from the API using `axios.get`:

```
const [data, setData] = useState(null);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);

useEffect(() => {
  const fetchData = async () => {
    try {
      const res = await axios.get(
        "https://jsonplaceholder.typicode.com/users"
      );
      setData(res.data);
    } catch (err) {
      setError(err);
    } finally {
      setLoading(false);
    }
  };

  fetchData();
}, []);
```

You might have noticed that there is no `await res.json()` line in this example. That's because Axios automatically parses JSON, so there's no need for that.

The last example we will look at is to use the `useSWR` hook to fetch data.

Just like with Axios, you will need to install SWR like this:

```
npm install swr
```

Then you will need to import the `useSWR` hook into the file like this:

```
import useSWR from "swr";
```

In comparison to the previous examples, the SWR syntax is way shorter. What you need to do is to create a fetcher function and pass it into the `useSWR` hook as its second parameter (the endpoint is the first parameter).

You also get to destructure both the data and error states from the `useSWR` hook, so you don't need the `useState` hook.

Here is the syntax:

```
const fetcher = (url) => fetch(url).then((res) => res.json());
const { data, error } = useSWR(endpoint, fetcher);
```

Note that the "fetcher" name here is only a convention, so you're free to name the variable whatever you want.

Here's a component fetching todos from the JSON Placeholder API:

```
import useSWR from "swr";

const fetcher = (url) => fetch(url).then((res) => res.json());

const FetchTodos = () => {
  const { data, error } = useSWR(
    "https://jsonplaceholder.typicode.com/todos",
    fetcher
  );

  if (!data) {
    return <h2>Loading...</h2>;
  }
  if (error) {
    return <h2>Error: {error.message}</h2>;
  }

  return (
    <>
      <h2>Todos</h2>
      <div>
        {data.map((todo) => (
          <h3 key={todo.id}>{todo.title}</h3>
        ))}
      </div>
    </>
  );
};

export default FetchTodos;
```

As you learned in a previous lesson on custom hooks, data fetching is a logic you can extract into a custom hook. So, if you're fetching data in multiple components and pages, it is best to create a `useFetch` hook.

Here's a `useFetch` hook that uses SWR for data fetching:

```
import useSWR from "swr";

const fetcher = (url) => fetch(url).then((res) => res.json());

const useFetch = (url) => {
  const { data, error } = useSWR(url, fetcher);

  return {
    data,
    loading: !data && !error,
    error,
  };
};

export default useFetch;
```

And here's how to use the `useFetch` hook to rewrite the first example that fetches posts from the JSON Placeholder API:

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

const FetchPosts = () => {
  const { data, loading, error } = useFetch(
    "https://jsonplaceholder.typicode.com/posts"
  );

  if (loading) {
    return <h2>Loading...</h2>;
  }

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

  return (
    <>
      <h2>Posts</h2>
      <ul>
        {data.map((post) => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </>
  );
};

export default FetchPosts;
```

Questions

What are the two parameters of the `useSWR` hook in the given example?

URL and cache policy.

API route and fetcher function.

Endpoint and fetcher function.

Key and configuration object.

Why do you have to handle data fetching logic inside a `useEffect`?

Because data fetching should only run once.

Because data fetching should not be part of the rendering process.

Because `useEffect` executes before the component renders.

Because `useEffect` runs synchronously with the render cycle.

What is Axios built upon?

The Fetch API.

The XMLHttpRequest object.

The WebSocket API.

The DOM API.