Recap: Querying APIs in React

Overview

In this recap, you'll revisit some of the concepts you covered in the Advanced React course. You need to have a solid grasp of these concepts to be able to fetch remote data and work with it in your project and future apps.

JSON data in your apps

JSON, which stands for JavaScript Object Notation, is the most popular data transfer format. It's a file extension format and a data interchange format. For an example of a JSON file, you can inspect any React app built using the `create-react-app npm` package. Each such app has a `package.json` file written in JSON format. Feel free to open this `package.json` file to quickly remind yourself of the basic syntax rules of JSON. It's important to remember that React projects are just some of the ones to use the `package.json` file. Any Node.js application initialized using the `npm init` command will show several prompts in the console or terminal. Based on your answers, `npm` will populate the contents of the `package.json` file that gets built in the process.

Fetching JSON data

So, how do you obtain this JSON data in our apps? Well, generally, the JSON data exists somewhere on the web, at a specific URL. You can access that data in several ways, but one of the common ways to do it is using the `fetch()` function, which is built into JavaScript.

The fetch() function

Using the Fetch API, you access the JSON data on the web - either from your own server or from a third-party JSON API. However, it's important to remember that React allows you to use the `fetch()` method, which is considered a **side effect** functionality.

Side effects

A side effect is anything that is happening outside of React itself. There are many examples of side effects, such as:

- Using `console()`
- Using `document.title`
- Using `fetch()`

As a result, to add a `fetch()` call to your React app, you need to use the `fetch()` function call inside a `useEffect` hook.

Working with retrieved JSON data

Once you've received this data, you need to:

- Update the responsible component's state with it.
- Loop over the data received and saved to that component's state or loop over the data received and passed from that component to its child component or components.

Looping over the data received usually involves using the `map()` method available on arrays, which is why it's convenient if your data comes in as an array of objects.

Simpler approaches

Note that when you are coding your apps locally, especially if you are focused on building functionality on the front end, you can simulate this data retrieval by simply placing a JSON file inside your project. In that case, all you need to do to access the data is to parse the JSON string from that `.json` file into the component that's supposed to receive it. This can be even further simplified by saving your data as a plain JavaScript array of objects - effectively, it's the same thing as fetching data, but it saves time and effort! An additional advantage of this approach is that you already have the data in the form ready to be used by your React app, so you'd just need to import this data module into the appropriate component and work with it. This has the added benefit of not having to deal with side effects and data fetching from a remote server, so it's ideal for quick application prototypes.

Conclusion

With this overview on querying APIs in React, you are better prepared for your next exercise where you will connect the Bookings page using live data. For more information on querying APIs in React, you can revisit the following lesson items from the **Advanced React** course:

- What are the rules of hooks?
- What you need to know before fetching data
- Data fetching using hooks
- Fetching data - Putting it all together