matt readout

39 Followers About Follow

Using JavaScript's Async/Await Syntax to Fetch Data in a React App

matt readout Jul 8, 2019 · 3 min read

Synchronous Fetches

A very common question when building a React app is how and when data fetching will happen. When a user first logs in or accesses the app, there's a certain amount of data that should be available to them — for instance, a list of available cities to populate a dropdown menu, or keywords to start a search. Depending on how your API

However, performing multiple fetch requests — calling `fetch()` — isn't ideal because fetching is asynchronous, meaning that your code will continue executing without waiting for the fetch to complete. If your code involves storing data returned from the fetch requests in the component's state object (another asynchronous action), some of the data might not be available by the time you call `setState()`. This results in returning the promise from the `fetch()` call, rather than a response object.

Using JavaScript's async/await feature solves these potential problems! It also allows for writing much clearer and more concise code, without the need to chain `.then()` calls, etc.

Basically, when calling `fetch()` with the `await` keyword, we're telling the async function to stop executing until the promise is resolved, at which point it can resume execution and return the resolved value. Rather than getting promises, we will get back the parsed JSON data that we expect.

In this example, I'm using async/await to fetch data for a character generator tool I built for players of the TRPG Dungeons & Dragons. Using the D&D 5e API to supply the data, I need to have all of the options for character classes and races available when a user first accesses the app. Because this requires hitting two different API endpoints, I'll make two `fetch()` calls, but (a)waiting until they've both resolved before I get what I need from the responses and then store those in the component state. I'm placing all of this within React's `componentDidMount()` to make sure everything is available on page load.

```
import React from 'react'
import { RACES, CLASSES } from '../apiEndpoints'

class Generator extends React.Component {

 state = {
   races: [],
   classes: [],
   result: null
 }
```

```
    res.json())

  const races = racesResponse.results
  const classes = classesResponse.results

  this.setState({ races, classes })

  }

...

}
```

Using the `await` keyword lets me be sure that both `racesResponse` and `classesResponse` will point to parsed JSON data — rather than promises — that I can then use to set the component state. It's almost like you're forcing asynchronous actions to execute synchronously, allowing you to accurately predict what information will be available when and where you want it in your app.

The very important thing to keep in mind is that the `await` keyword can only be used in `async` functions! These are functions that are defined with the syntax:

```
async function name() {
    // statements
}
```

In my example, I'm using the `componentDidMount()` method from `React.Component` but defining it as an `async` function — this allows my use of `await` for each fetch. Using `await` outside of an `async` function definition will result in a syntax error!

References

async function

developer.mozilla.org

await

The await operator is used to wait for a Promise. It can only be used inside an async function.

developer.mozilla.org

JavaScript