Juliet Onyekaoha

Follow 15 Followers About

React Cancel all axios request in componentWillUnmount.

Juliet Onyekaoha Oct 20, 2020 · 4 min read

The code for this article is in this **codesandbox,** If you don't need explanations as to why things are happening, head over there, otherwise keep reading, I promise you will learn a thing or two.

Why Cancel Requests?

It is one thing to write an asynchronous call to an API to fetch data, and it is another thing to cancel that call when it is not beneficial to our users. The single action of cancelling API request when it is no longer needed leads to a better performant app devoid of bugs. A common scenario where an API request needs to be cancelled is when a user navigates away from a page.

Let's say you have a React app that makes several calls once the component mounts inside the componentDidMount lifecycle of a component, if a user navigates to that page, the API requests starts, however, if the user navigates away from that page before the request is completed, by default the API requests still runs in the background and if possible updates different parts of your application that need that data even though at that point the user no longer needs (because they are on another page) the data that is returned from the request.

At the point the user navigates away from the page, all API requests made in it to update the state and render a view with the data is unnecessary and if not cancelled leads to memory leaks in an application. React alerts us to this problem by outputting these popular lines in the console

Warning: Can't perform a React state update on an unmounted component. This is a no-op, but it indicates a memory leak in your application. To fix, cancel all subscriptions and asynchronous tasks in the componentWillUnmount method.
in Landing (created by Context.Consumer)

How cancelling requests works in axios

To start the process of cancelling an axios request, the following steps are involved:

1. Create a variable that will hold the cancel token source

   `let source = axios.CancelToken.source();`

2. Inside the axios request config, pass the token of the created `source` variable as the value of the cancelToken key/property.

   `let config = { cancelToken: source.token}`

   `axios.get(endpointUrl, config).then((res) => {})`

   or

   `axios.get(endpointUrl, { cancelToken: source.token}).then((res) => {})`

3. Trigger the cancel request by calling `source.cancel()` where and when (in a react component this can be on the *componentWillUnmount* lifecycle method or on the click of a button) you need to cancel the request.

4. Catch the error being thrown (an error is thrown in every cancel request) inside the catch block of the axios request.

The entire axios cancel request API is built on top of the DOM AbortController API that is used to abort/terminate all asynchronous actions in a program. To use it:

- Instantiate a new controller object

  `const controller = new AbortController();`

- Then create a signal that alerts the async action on when to terminate/abort its process. In case of a network request, this will signal the request to cancel.

  `const signal = controller.signal;`

- Pass the signal to the action. For e.g, a fetch request has a signal property on its config object that accepts an AbortSignal.

  `fetch(url, { signal: signal}).then(res => {})`

- Call the `abort` property on the AbortController object to abort the request.

  `controller.abort();`

For practical code examples of how to use the AbortController API refer to this article.

How to Cancel Requests where the Axios request is inside the component.

```
import React from "react";
import axios from "axios";
import { Link } from "react-router-dom";
import "./styles.css";

let source;
let usersEndpoint = "https://jsonplaceholder.typicode.com/users";

let source = axios.CancelToken.source();
class Landing extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      data: [],
      isLoading: false
    };

    source = axios.CancelToken.source();
  }

  componentDidMount() {
    this.fetchUser();
  }

  fetchUser = () => {
    setTimeout(() => {
      this.setState({ isLoading: true });
      axios
        .get(usersEndpoint, {
          cancelToken: source.token
        })
        .then((res) => {
          this.setState({ data: res.data, isLoading: false });
        })
        .catch((e) => {
          console.log(e.message);
        });
    }, 2000);
  };

  componentWillUnmount() {
    if (source) {
      source.cancel("Landing Component got unmounted");
    }
  }

  render() {
    const { data, isLoading } = this.state;
    return (
      <div>
        {isLoading && <h2>Loading...</h2>}
        {!isLoading && (
          <div>
            <Link className="links" to="/todos">
              Go to Todos
            </Link>
            <Link className="links" to="/pictures">
              Go to Pictures
            </Link>
            <h4>The axios request is made from inside this component</h4>
            {data.length > 0 &&
              data.map((item) => {
                return <p key={item.id}>{item.name}</p>;
              })}
          </div>
        )}
      </div>
    );
  }
}

export default Landing;
```

landing.js hosted with ❤ by **GitHub** **view raw**

Both the axios request and cancel request happens is handled inside this component

Steps involved:

- A variable called `source` was initialized outside the component

  `let source;`

- Inside the constructor method, we assigned the cancel token source property to the source variable.

  `source = axios.CancelToken.source();`

- Inside the `fetchUser` was method, we instantiated an axios request and set the `cancelToken` inside the config object as the token property of the source variable

  `axios.get(usersEndpoint, { cancelToken: source.token}).then((res) => {...})`

- We triggered the abort/cancel process by calling source.cancel() inside the componentWillUnmount lifecycle method.
  `source.cancel("Landing component got unmounted);`

. . .

How to cancel requests where the axios requests are extracted out to different files/folders.

In most large production applications, API requests are handled in separate folders, extracting the core logic from the components.

For this article, we have an *api-service.js* file (handles the calls made to various API services) and an *axios-requests.js* file(handles the actual configuration of the axios requests).

Inside the *axios-request.js* file, we're exporting out an object called `apiRequestsFormat` , that holds a key `getRequest` which is a function that takes the `url` and `cancelToken`. On line 5, the `cancelToken` is passed into the `config` object which is further passed into the axios get call along with the `url`. On line 14, we capture the error thrown by the aborted/cancelled requests. In this case, the error is just logged into the console. You might want to do something else with it.

Handles requests to different API's.

Inside the *api-service.js* file, we write the functions that make the calls to our different API's, passing the token we will be generating from our components as arguments.

The *todos.js* component is responsible for displaying a list of todos that are fetched from an API. Just like in *landing.js*, we instantiate the source variable and assign it the cancel token source property.

On line 19, the fetchTodos function calls the fetchTodos function in *api-service.js* and pass the token from the source variable into the function as an argument. This token argument also gets passed into the `config` object (which is the second parameter of the actual axios call) in *axios-requests.js.*

The same process in *todos.js* is repeated in *pictures.js* below.

Things to Note

- The token property is passed into the config object not the entire source object. `source.token` not `source`
- One signal from the *AbortController()* object can be used to abort/cancel multiple API requests. This means that if we have more than one API request in our component(regardless of the method, POST, GET, DELETE e.t.c), we can use one source variable to cancel all the requests.
- We need to catch the error thrown by the cancelled asynchronous request. Just like it was done in *landing.js* line 34 and *axios-requests.js* line 14.

You can find the entire code for this article in the codesandbox or clone it from this github repo.

Axios Cancel Request Abortcontroller React Componentwillunmount