Axios instead of Fetch

In this lesson, you will learn how to shift from Fetch API to other APIs like Axios in your React app.

Before, we introduced the native fetch API to perform a request to the Hacker News platform. The browser allows you to use this native fetch API. However, not all browsers support this, older browsers especially. Once you start testing your application in a headless browser environment (where there is no browser, it is mocked), there can be issues with the fetch API. There are a couple of ways to make fetch work in older browsers (polyfills) and in tests (isomorphic-fetch), but these concepts are bit off-task for the purpose of learning React.

One alternative is to substitute the native fetch API with a stable library such as axios, which performs asynchronous requests to remote APIs. In this chapter, we will discover how to substitute a library, a native API of the browser in this case, with another library.

Below, the native fetch API is substituted with axios. First, we install axios on the command line:

```
npm install axios
```

Second, import axios in your App component's file:

```
import React, { Component } from 'react';
import axios from 'axios';
import './App.css';

...
```

You can use axios instead of `fetch()`, and its usage looks almost identical to the native fetch API. It takes the URL as argument and returns a promise. You don't have to transform the returned response to JSON anymore, since axios wraps the result into a `data` object in JavaScript. Just make sure to adapt your

code to the returned data structure:

```
class App extends Component {

  ...

  fetchSearchTopStories(searchTerm, page = 0) {
    axios(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}&${PAR
      .then(result => this.setSearchTopStories(result.data))
      .catch(error => this.setState({ error }));
  }

  ...

}
```

In this code, we call `axios()`, which uses an HTTP GET request by default. You can make the GET request explicit by calling `axios.get()`, or you can use another HTTP method such as HTTP POST with `axios.post()`. With these examples alone, we can see that axios is a powerful library to perform requests to remote APIs. I recommend you use it instead of the native fetch API when requests become complex, or you have to deal with promises.

Now we'll introduce another improvement for the Hacker News request in the App component. Imagine the component mounts when the page is rendered for the first time in the browser. In `componentDidMount()` the component starts to make the request, but then the user navigates away from the page with this rendered component. Then the App component unmounts, but there is still a pending request from `componentDidMount()` lifecycle method. It will attempt to use `this.setState()` eventually in the `then()` or `catch()` block of the promise. You will likely see the following warning on your command line, or in your browser's developer output: *Warning: Can only update a mounted or mounting component. This usually means you called setState, replaceState, or forceUpdate on an unmounted component. This is a no-op.*

You can handle this issue by aborting the request when your component unmounts or preventing `this.setState()` on an unmounted component. It is considered a best practice in React to preserve a clean application without warnings. However, the current promise API doesn't implement aborting a request, so we add a workaround, introducing a class field that holds the lifecycle state of your component. It can be initialized as `false` when the component initializes, changed to `true` when the component mounted, and then reset to `false` when the component unmounted. This way, you can keep

track of your component's lifecycle state. It doesn't affect the local state stored

and modified with `this.state` and `this.setState()`, because you can access it directly on the component instance without relying on React's local state management. Moreover, it doesn't lead to any re-rendering of the component when the class field is changed.

```
class App extends Component {

  _isMounted = false;
  constructor(props) {
    ...
  }

  ...

  componentDidMount() {
    this._isMounted = true;

    const { searchTerm } = this.state;
    this.setState({ searchKey: searchTerm });
    this.fetchSearchTopStories(searchTerm);
  }

  componentWillUnmount() {
    this._isMounted = false;
  }

  ...

}
```

Finally, you can use this knowledge not to abort the request itself, but avoid calling `this.setState()` on your component instance, even though the component already unmounted. It will prevent the warning.

```
class App extends Component {

  ...

  fetchSearchTopStories(searchTerm, page = 0) {
    axios(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}&${PAR
      .then(result => this._isMounted && this.setSearchTopStories(result.data))
      .catch(error => this._isMounted && this.setState({ error }));
  }

  ...

}
```

This chapter has shown you how you can replace one library with another in

React. If you run into any issues, you can use the vast library ecosystem in JavaScript to find a solution. Also, we've learned how to avoid calling `this.setState()` in React on an unmounted component. If you dig deeper into the axios library, you will find a way to cancel the request beforehand.

Further Reading:

- Read about why frameworks matter