Retrieve Data from an API with React

Last updated March 25, 2020

Theming Module Development 8.9.x/9.0.x

React excels at displaying lists of data. But that data needs to come from somewhere. In most cases this happens by making a network request to retrieve data from one or more APIs, processing the response, and then displaying the data. In the context of working with a Drupal site there are two possible options: Drupal core's JSON API module, or the `drupalSettings` JavaScript API.

In this tutorial we'll focus on the high-level overview and:

- Explain the difference between content and configuration data
- Introduce the JavaScript Fetch API, and where to find information about using it
- Get an overview of the ES6 array functions we'll use to parse the complex data structures returned from an API

In the remaining tutorials in this series we'll provide examples of real world use-cases.

By the end of this tutorial you should have a firm understanding of how to get started making API requests using React.

Goal

Explain how to make API requests to retrieve data from a React application.

Prerequisites

- React Basics

Content versus configuration

In the context of a Drupal application there are two types of data we can use:

- **Content**: User-generated content like a list of blog posts, calendar events, or taxonomy terms.
- **Configuration**: Administrator-configurable settings that influence how our application works. For example, what content to list, or the number of items to display in the list.

When dealing with **content** the best path is to enable the Drupal core JSON API module and make API requests to Drupal, just like you would if your application was decoupled from Drupal.

Learn more about this approach in Use React to List Content from Drupal.

For **configuration**, there are a couple of options. If your React code is embedded in a Drupal theme or module you can Use Server-Side Settings with drupalSettings to pass configuration data. This is by far the easiest approach. If your JavaScript code is decoupled from Drupal you'll need to find a way to expose the required configuration as JSON:API data. This could mean creating a new JSON API resource or hanging your configuration on an existing entity type like Consumers.

Get data from an API via HTTP requests

The most common way to make network requests with modern JavaScript is via the Fetch API, or one of the libraries that wrap `fetch` with additional features. We're discussing this in the context of a React application. But React is *just* JavaScript, and anything we do here will work in most JavaScript applications.

Why not jQuery?

In Drupal 8, jQuery is included with core. Its `.get` and `.ajax` methods are powerful utilities for making network requests with JavaScript. However, there are good reasons to not use jQuery for HTTP requests.

First, jQuery is no longer loaded on every page by default. One of the main reasons to use jQuery in your Drupal JavaScript code is the fact that it's there already. But if you don't need it you can avoid the overhead, and potential performance penalty, of loading it.

The JavaScript language has evolved significantly since jQuery was created, and now includes built-in functionality for handling asynchronous network requests in the form of promises. There are now more efficient ways of writing *closures*, which tidy up your JavaScript so that you do not wind up with complicated "trees" of nested asynchronous calls.

`fetch` is easier and cleaner to deal with than the longstanding XHR (XMLHttpRequest) format of the past. Abstractions like jQuery's `.get` are no longer necessary in most cases.

What is `fetch`?

The Fetch API provides a native JavaScript interface for retrieving data over a network. It's built into ES6.

The `fetch()` function lets you send network requests and get responses. It uses promises to allow for asynchronous requesting and processing of data.

- Fetch - living standard
- Introduction to fetch()

What are promises?

Promises are a way that you can pass an asynchronous function, and if the request is successful, do certain behaviors after the response is complete. If the function is not successful, say that it failed and do something else.

- Using Promises and Promise documentation
- JavaScript promises for dummies

Anatomy of a `fetch()` request

Consider the following example JavaScript code:

```
fetch(url, options)
  .then(response => response.json())
  .then(result => console.log('success', result))
  .catch(error => console.log('error', error));
```

Here's what's happening in this request:

1. Send the request with `fetch(url, options)` along with options, which can contain your authentication header information, timeouts, and CORS settings.
2. Get a response from the server and convert it to JSON. Note: This fails if it's a 500 error or there's a network connection failure.
3. Do something with the data. Since fetch passes a promise, you will either need to pass a method that knows how to update your code or process the results and run the interface changes you want to make *inside* the promise.
4. Catch any errors: `.catch(error => console.log('error', error));`. This is the handling of the 500 errors from the original `fetch` request, or if something else breaks in the sequence.

Note: `.then` methods can take 2 functions: one for handling success and another for handling errors. Depending on what you are doing, you may need to reject your promise in a `.then`. For example, you might retrieve some data, parse the response, and notice that the response itself contains an error.

Libraries for AJAX requests

Fetch is a standardized and well-documented native JavaScript utility for making asynchronous network requests. We will use `fetch` in these tutorials. It's not 100% supported, but close enough for our needs.

One commonly used alternative is Axios. Axios is like `fetch` but gives you a little bit more control over your network requests. `Superagent` and `request` also provide similar functions, but have Node.js packages that you can include for isomorphic (i.e. server-side AND client-side) React applications.

For this set of tutorials `fetch` will work just fine. But it's worth at least familiarizing yourself with the other available options.

Using ES6 functions to iterate through data

Once you've made a request and retrieved some data, you'll need to traverse the data and do something with it. ES6 comes with functions for mapping (`.map`), filtering(`.filter`) and iterating over your data (`.each`). For this tutorial, we will use the `.map` feature, but the others are similar. We will go over this with `jsonapi` data.

On the Mozilla Developer Network, read more about ES6 functions you can use to iterate through date:

- Array.prototype.map()
- Array.prototype.filter()
- Array.prototype.each()
- Loops and iteration

It's also a good idea to be familiar with `arrow functions` and how they work. That syntax is commonly used when working the above array functions. Learn more about arrow functions.

Recap

In this tutorial we learned about the difference between content and configuration data in the context of a Drupal backend. Then we looked at `fetch` and introduced `axios`, two modern ways to get data from an API into a JavaScript application. We also mentioned some commonly used ES6 features for iterating over data sets.

Further your understanding

- Use the `fetch()` function in your browser's developer console to practice retrieving data over the network. Can you use `fetch()` to access your Drupal endpoints?
- Learn about working with promises in JavaScript, using links provided in this tutorial.
- Practice `.map` with test data, using a tool like JSfiddle or with snippets in Console in your web developer's toolkit.

Additional resources

- What is the difference between the Fetch API and XMLHttpRequest? (stackoverflow.com)
- axios -- Promise-based HTTP client for the browser and Node.js (npmjs.com)

- Why I won't be using Fetch API in my apps (medium.com)
- Fetch vs. Axios.js for making http requests (medium.com)
- You-Dont-Need-Lodash-Underscore (github.com)
- Fundamentals of ES6 (es6.io)
- The Best Tutorials To Learn ECMAScript (ES6) For Beginners (medium.com)

Was this helpful? Yes No

◀ Previous tutorial

Next tutorial ▶

Get Started Using React and Drupal Together

10 **Build an Interface to Edit Nodes with React**

11 **Create a Fully Decoupled React Application** Free

12 **Use create-react-app to Start a Decoupled React Application**

13 **Make API Requests with OAuth**

14 **Use Fetch and OAuth to Make Authenticated Requests**

STAY INFORMED

Sign up for our mailing list to get Drupal tips and tricks in your inbox!

Subscribe

STAY CONNECTED

Powered by: