Create, Update, and Delete Drupal Content with JavaScript

Last updated March 30, 2020

Theming Module Development 8.9.x/9.0.x

To perform create, update, and delete (CRUD) operations with Drupal core's JSON:API via React there are a few things you'll need to understand. First, how to format the `POST`, `PATCH`, and `DELETE` requests necessary to add, edit, and delete Drupal entities. Next, how to handle authentication, and cross-site request forgery (CSRF) tokens. Over the next few tutorials we will create a simple but powerful React application that can add, edit, and delete Drupal node content.

This tutorial contains:

- An overview of the application we're building
- Information about making secure authenticated requests to Drupal's JSON:API
- An overview of the API requests we'll use to create, update, and delete nodes

By the end of this tutorial, you should have a picture of the application we're going to build, and know how to make the API requests we'll use in our application.

Goal

Understand how React can communicate with Drupal to add, edit, and delete nodes and other content.

Prerequisites

- Drupal 8.5.x installed, with some *article* nodes created. If you have Drupal 8.7 or higher installed, JSON:API module is already included.
- Install JSON:API Module
- The application builds on the previous tutorials in this series. At a minimum, you'll need to know how to Connect React to a Drupal Theme or Module, and configure a JavaScript build toolchain to compile ES6/React code.

What we will build

The goal is to build a React application that will allow a user to edit Drupal node content from within a user interface provided by React. This exercise demonstrates the fundamental concepts you'll need to understand. Our example takes shortcuts with regards to error handling and UI niceties, but we'll try and call those out when they happen.

Here's an example of what we're building:

Initially we'll build this application embedded inside a Drupal theme to show how progressively decoupled applications might work. Then, we'll refactor our code to a fully decoupled application that works by accessing Drupal via the backend API.

The methods we use in this tutorial apply to any content entity in Drupal and are intended to familiarize you with how to perform CRUD operations with React. In addition to editing node content, you can modify user data, taxonomy terms, and other content entities in Drupal using methods like those presented here.

Note: If you're using the Drupal core REST module instead of JSON:API you can also use similar techniques to manage configuration entities. The request syntax isn't exactly the same for core REST, but the concepts are close enough that you should be able to use what you learn here.

Dealing with authorized requests

Doing things like creating and deleting content in Drupal requires that you're signed in as a user with permission to perform those operations. This is also true when using JSON:API to perform these tasks.

It's worth mentioning that initially, you will *not* need to set up OAuth, CORS or Http Basic Authentication. This is because our React code gets embedded on a Drupal page, and inherits the permissions of the user viewing that page.

This works because using the JavaScript `fetch` function, and its `same-origin` setting, while logged in to Drupal, lets us use our current session to make authenticated requests to Drupal. Whenever a `fetch` request gets issued to the same domain that the JavaScript code is served from, the browser will send your Cookies along with the request. In the background, Drupal will use the content of those Cookies to validate your session. If you are logged in as a user with permissions to add, edit or delete node content, the `fetch` request will succeed, and update the data.

This code will *only* work for React applications that get embedded on a Drupal site. Many React applications are stand-alone applications, and the tools used when writing more complicated React applications often require cross-origin network requests. These types of requests *do* require handling the authentication and authorization in our code. We will dive into this more when we refactor our application into a decoupled React front-end. If you want to learn more, read Make API Requests with OAuth and Use Fetch and OAuth to Make Authenticated Requests.

Cross-site Request Forgery (CSRF) tokens

To protect from Cross-Site Request Forgery (CSRF) attacks Drupal requires an `X-CSRF-Token` HTTP header in all requests that could result in a state change. In this case, those requests are POST, PATCH, and DELETE. You can retrieve a CSRF token for the current user's active session by making a GET request to `/session/token?_format=json`. Then include the retrieved token as an HTTP header in requests that require it.

Remember, this works using `fetch()` when our React code is embedded in the Drupal page because it uses your existing session when accessing `/sessions/token?_format=json`.

HTTP requests

With a few different HTTP methods, we can update and modify content within Drupal. This is made possible in our case by the JSON:API module. Check out the JSON:API documentation to learn more about using these different HTTP methods.

Below are some examples that we'll use in our application.

POST requests

POST requests are used to create new content. We POST to the endpoint for the content-type we would like to create.

Example HTTP request:

```
POST: `http://localhost:8888/jsonapi/node/article`
Content-Type: application/json
Accept: application/vnd.api+json
Cache: no-cache

{DATA}
```

When making requests we will set the headers as a constant variable, so that each of our fetch requests can use the headers. The `Accept` and `Content-Type` refer to the format of what we are sending and receiving.

The data we pass to the API endpoint should match the structure returned from a GET request.

Example:

```
{
  "data": {
    "type": "node--article",
```

```
    "attributes": {
      "title": "My New Node",
      "body": {
        "value": "Content of new node",
        "format": "plain_text"
      }
    }
  }
}
```

Example using `fetch`:

```
const headers = new Headers({
  'Accept': 'application/vnd.api+json',
  'Content-Type': 'application/vnd.api+json',
  'Cache': 'no-cache',
  'X-CSRF-Token': '{TOKEN}'
});

const options = {
  method: 'POST',
  credentials: 'same-origin',
  headers,
  body: JSON.stringify(data)
}

fetch('https://example.com/jsonapi/node/article', options)
  .then(response => response.json())
  .then((data) => {
    console.log(data);
  })
  .catch(error => console.log('API error', error));
```

POST requests will return an object, formatted to the JSON:API specification, which contains the data for the new node content, or an error if there is a problem.

Learn more in JSON:API POST Requests: Create an Entity.

PATCH requests

PATCH is used to update existing content. PATCH works like POST, except that the endpoint is the specific resource you want to update.

Example HTTP request:

```
PATCH http://localhost:8888/jsonapi/node/article/48c2cf4b-3782-4b00-a44f-19305796e2eb
Content-Type: application/json
Accept: application/vnd.api+json
Cache: no-cache

{DATA}
```

Note the UUID included in the path.

Make sure you include the `id` of the resource to update in the request body. Note that when making PATCH requests you need to include only the fields that you want to change.

Example:

```
{
  "data": {
    "type": "node--article",
    "id": "48c2cf4b-3782-4b00-a44f-19305796e2eb",
    "attributes": {
      "title": "My Node, edited",
    }
  }
}
```

Example using `fetch`:

```
const uuid = 'ID-OF-RESOURCE-TO-UPDATE';

const headers = new Headers({
  'Accept': 'application/vnd.api+json',
  'Content-Type': 'application/vnd.api+json',
  'Cache': 'no-cache',
  'X-CSRF-Token': '{TOKEN}'
});

const options = {
  method: 'PATCH',
  credentials: 'same-origin',
  headers,
  body: JSON.stringify(data)
}

fetch(`https://example.com/jsonapi/node/article/${uuid}`, options)
  .then(response => response.json())
  .then((data) => {
    console.log(data);
  })
  .catch(error => console.log('API error', error));
```

PATCH requests will return an object, formatted to the JSON:API specification, which contains the data for the edited node content, or an error if there is a problem.

Learn more in JSON:API PATCH Requests: Update an Entity.

DELETE requests

DELETE is used to delete an existing resource.

Example HTTP request:

```
DELETE http://localhost:8888/jsonapi/node/article/48c2cf4b-3782-4b00-a44f-19305796e2eb
```

Set the method to `DELETE`. You do not need to include anything in the body of the request.

Example using `fetch`:

```
const uuid = 'ID-OF-RESOURCE-TO-UPDATE';

const headers = new Headers({
  'Accept': 'application/vnd.api+json',
  'Content-Type': 'application/vnd.api+json',
  'Cache': 'no-cache',
  'X-CSRF-Token': '{TOKEN}'
});

const options = {
  method: 'DELETE',
  credentials: 'same-origin',
  headers
}

fetch(`https://example.com/jsonapi/node/article/${uuid}`, options)
  .then(response => response.json())
  .then((data) => {
    // Empty unless there is an error.
    console.log(data);
  })
  .catch(error => console.log('API error', error));
```

DELETE requests will return an HTTP 204 (No content) with an empty response body, or an error if there is a problem.

Learn more in JSON:API DELETE Requests: Delete an Entity.

Recap

In this tutorial we got a look at the application we'll be building throughout the next few tutorials. Then we covered background information needed to write React code that can create, update, and delete Drupal content via JSON:API. We looked at how user authentication works, what CSRF tokens are, and an overview of the types of HTTP requests we'll use.

Further your understanding

- Use a tool like Postman or `cURL` to experiment with the different requests above. Can you get them all to work?

- Review the JSON:API module API overview.
- Learn more about the HTTP methods used.

Additional resources

- HTTP Headers (developer.mozilla.org)
- What JSON:API doesn't do (drupal.org)
- API Authentication and Authorization (Drupalize.Me)
- Access an API from the Browser with Cross-Origin Resource Sharing (Drupalize.Me)

◀ Previous tutorial

Next tutorial ▶

8 Use React to List Content from Drupal

9 Create, Update, and Delete Drupal Content with JavaScript

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