@wordpress/api-fetch

- Installation
- Usage
  - GET
  - GET with Query Args
  - POST
  - Options
    - path (string)
    - url (string)
    - parse (boolean, default true)
    - data (object)
  - Requests to other sites
  - Aborting a request
  - Middlewares
  - Removing middlewares
  - Built-in middlewares
  - Custom fetch handler
- Contributing to this package

Utility to make WordPress REST API requests. It's a wrapper around `window.fetch`.

Installation

Install the module

```
npm install @wordpress/api-fetch --save
```

*This package assumes that your code will run in an **ES2015+** environment. If you're using an environment that has limited or no support for such language features and APIs, you should include*

the polyfill shipped in `@wordpress/babel-preset-default` in your code.

Usage

GET

```
import apiFetch from '@wordpress/api-fetch';

apiFetch( { path: '/wp/v2/posts' } ).then( ( posts ) => {
    console.log( posts );
} );
```

GET with Query Args

```
import apiFetch from '@wordpress/api-fetch';
import { addQueryArgs } from '@wordpress/url';

const queryParams = { include: [1,2,3] }; // Return posts with ID = 1,2,3.

apiFetch( { path: addQueryArgs( '/wp/v2/posts', queryParams ) } ).then( ( posts ) => {
    console.log( posts );
} );
```

POST

```
apiFetch( {
    path: '/wp/v2/posts/1',
    method: 'POST',
    data: { title: 'New Post Title' },
} ).then( ( res ) => {
    console.log( res );
} );
```

Options

`apiFetch` supports and passes through all options of the `fetch` global.

Additionally, the following options are available:

`path` (`string`)

Shorthand to be used in place of `url`, appended to the REST API root URL for the current site.

`url` (`string`)

Absolute URL to the endpoint from which to fetch. The request still goes through the registered middlewares, so see Requests to other sites before pointing it at another site.

`parse` (`boolean`, default `true`)

Unlike `fetch`, the `Promise` return value of `apiFetch` will resolve to the parsed JSON result. Disable this behavior by passing `parse` as `false`.

`data` (`object`)

Sent on `POST` or `PUT` requests only. Shorthand to be used in place of `body`, accepts an object value to be stringified to JSON.

Requests to other sites

`apiFetch` is meant for the REST API of the WordPress site the script runs on. Whenever WordPress loads the package it registers the root URL and nonce middlewares, so with the default fetch handler every request, including one made with a full `url`, is sent with the current user's `X-WP-Nonce` header and with cookies (`credentials: 'include'`). A WordPress site on another origin cannot validate that nonce and rejects the request with a `403` `rest_cookie_invalid_nonce` error, even for public endpoints, and other origins may refuse a credentialed cross-origin request altogether.

Use `window.fetch` for requests to other sites, or make them from the server. A browser request to another site still needs that site to allow the cross-origin request; a WordPress site does so through the CORS headers its REST API sends by default.

```
import apiFetch from '@wordpress/api-fetch';

// Same site: goes through the middlewares.
apiFetch( { path: '/wp/v2/posts' } ).then( ( posts ) => {
    console.log( posts );
} );
```

```
// Another site: use fetch directly.
window
    .fetch( 'https://example.com/wp-json/wp/v2/posts' )
    .then( ( response ) => response.json() )
    .then( ( posts ) => {
        console.log( posts );
    } );
```

Aborting a request

Aborting a request can be achieved through the use of `AbortController` in the same way as you would when using the native `fetch` API.

For legacy browsers that don't support `AbortController`, you can either:

- Provide your own polyfill of `AbortController` if you still want it to be abortable.
- Ignore it as shown in the example below.

Example

```
const controller =
    typeof AbortController === 'undefined' ? undefined : new AbortController();

apiFetch( { path: '/wp/v2/posts', signal: controller?.signal } ).catch(
    ( error ) => {
        // If the browser doesn't support AbortController then the code below will never log.
        // However, in most cases this should be fine as it can be considered to be a
progressive enhancement.
        if ( error.name === 'AbortError' ) {
            console.log( 'Request has been aborted' );
        }
    }
);

controller?.abort();
```

Middlewares

the `api-fetch` package supports middlewares. Middlewares are functions you can use to wrap the `apiFetch` calls to perform any pre/post process to the API requests.

Example

```
import apiFetch from '@wordpress/api-fetch';

apiFetch.use( ( options, next ) => {
    const start = Date.now();
    const result = next( options );
    result.then( () => {
        console.log( 'The request took ' + ( Date.now() - start ) + 'ms' );
    } );
    return result;
} );
```

Removing middlewares

`apiFetch.unregister` removes a middleware by reference, and returns whether it was registered. Built-in middlewares can be removed too, as long as they are exposed on `apiFetch`.

```
import apiFetch from '@wordpress/api-fetch';

// Send `DELETE` as `DELETE`, rather than as a `POST` carrying an `X-HTTP-Method-Override`
header.
apiFetch.unregister( apiFetch.httpV1Middleware );
```

Removal is global. Every `apiFetch` call on the page loses the middleware, so only remove a middleware you registered yourself, or a built-in whose behavior the whole page can do without. `httpV1Middleware` in particular exists so that requests keep working on servers and firewalls that reject `PATCH`, `PUT` and `DELETE`; removing it can break saving on those sites.

Built-in middlewares

The `api-fetch` package provides built-in middlewares you can use to provide a `nonce` and a custom `rootURL`.

Nonce middleware

```
import apiFetch from '@wordpress/api-fetch';

const nonce = 'nonce value';
apiFetch.use( apiFetch.createNonceMiddleware( nonce ) );
```

The function returned by `createNonceMiddleware` includes a `nonce` property corresponding to the actively used nonce. You may also assign to this property if you have a fresh nonce value to use.

Root URL middleware

```
import apiFetch from '@wordpress/api-fetch';

const rootURL = 'http://my-wordpress-site/wp-json/';
apiFetch.use( apiFetch.createRootURLMiddleware( rootURL ) );
```

Custom fetch handler

The `api-fetch` package uses `window.fetch` for making the requests but you can use a custom fetch handler by using the `setFetchHandler` method. The custom fetch handler will receive the `options` passed to the `apiFetch` calls.

Example

The example below uses a custom fetch handler for making all the requests with `axios`.

```
import apiFetch from '@wordpress/api-fetch';
import axios from 'axios';

apiFetch.setFetchHandler( ( options ) => {
    const { url, path, data, method } = options;

    return axios( {
        url: url || path,
        method,
        data,
    } );
} );
```

The default handler remains available, so it can be restored later.

```
apiFetch.setFetchHandler( apiFetch.defaultFetchHandler );
```

Contributing to this package

This is an individual package that’s part of the Gutenberg project. The project is organized as a monorepo. It’s made up of multiple self-contained software packages, each with a specific purpose.

The packages in this monorepo are published to npm and used by WordPress as well as other software projects.

To find out more about contributing to this package or Gutenberg as a whole, please read the project’s main contributor guide.

□□□ #2
□ □□□ □□□ 2026-10-31 18:17:17 UTC
□ □□□ □□□ 2026-10-31 18:17:19 UTC