Middleware

- Add middleware
- Chain middleware

Middleware is reusable logic that can be run before your `onRequest` function. Middlewares are typically utility functions. Error handling, user authentication, and logging are typical candidates for middleware within an application.

Add middleware

Middleware is similar to standard Pages Functions but middleware is always defined in a `_middleware.js` file in your project's `/functions` directory. A `_middleware.js` file exports an `onRequest` function. The middleware will run on requests that match any Pages Functions in the same `/functions` directory, including subdirectories. For example, `functions/users/_middleware.js` file will match requests for `/functions/users/nevi`, `/functions/users/nevi/123` and `functions/users`.

If you want to run a middleware on your entire application, including in front of static files, create a `functions/_middleware.js` file.

In `_middleware.js` files, you may export an `onRequest` handler or any of its method-specific variants. The following is an example middleware which handles any errors thrown in your project's Pages Functions. This example uses the `next()` method available in the request handler's context object:

```
export async function onRequest(context) {
  try {
    return await context.next();
  } catch (err) {
    return new Response(`${err.message}\n${err.stack}`, { status: 500 });
  }
}
```

Chain middleware

You can export an array of Pages Functions as your middleware handler. This allows you to chain together multiple middlewares that you want to run. In the following example, you can handle any errors generated from your project's Functions, and check if the user is authenticated:

```
async function errorHandling(context) {
  try {
    return await context.next();
  } catch (err) {
    return new Response(`${err.message}\n${err.stack}`, { status: 500 });
  }
}

function authentication(context) {
  if (context.request.headers.get("x-email") != "admin@example.com") {
    return new Response("Unauthorized", { status: 403 });
  }

  return context.next();
}

export const onRequest = [errorHandling, authentication];
```

In the above example, the `errorHandling` function will run first. It will capture any errors in the `authentication` function and any errors in any other subsequent Pages Functions.

□□□ #2
□ □□□ □□□ 2026-11-01 03:02:12 UTC
□ □□□ □□□ 2026-11-01 03:02:14 UTC