Middleware

Version History

Version	Changes
`v12.0.9`	Enforce absolute URLs in Edge Runtime (PR)
`v12.0.0`	Middleware (Beta) added.

Middleware enables you to use code over configuration. This gives you full flexibility in Next.js, because you can run code before a request is completed. Based on the user's incoming request, you can modify the response by rewriting, redirecting, adding headers, or even streaming HTML.

Usage

1. Install the latest version of Next.js:

```
npm install next@latest
```

2. Then, create a `_middleware.ts` file under your `/pages` directory.
3. Finally, export a middleware function from the `_middleware.ts` file.

```
// pages/_middleware.ts

import type { NextFetchEvent, NextRequest } from 'next/server'

export function middleware(req: NextRequest, ev: NextFetchEvent) {
  return new Response('Hello, world!')
}
```

In this example, we use the standard Web API Response (MDN).

API

Middleware is created by using a `middleware` function that lives inside a `_middleware` file. Its API is based upon the native `FetchEvent`, `Response`, and `Request` objects.

These native Web API objects are extended to give you more control over how you manipulate and configure a response, based on the incoming requests.

The function signature:

```
import type { NextFetchEvent } from 'next/server'
import type { NextRequest } from 'next/server'

export type Middleware = (
  request: NextRequest,
```

```
  event: NextFetchEvent
) => Promise<Response | undefined> | Response | undefined
```

The function can be a default export and as such, does **not** have to be named `middleware`. Though this is a convention. Also note that you only need to make the function `async` if you are running asynchronous code.

Read the full Middleware API reference.

Examples

Middleware can be used for anything that shares logic for a set of pages, including:

- Authentication
- Bot protection
- Redirects and rewrites
- Handling unsupported browsers
- Feature flags and A/B tests
- Advanced i18n routing requirements

Execution Order

If your Middleware is created in `/pages/_middleware.ts`, it will run on all routes within the `/pages` directory. The below example assumes you have `about.tsx` and `teams.tsx` routes.

```
- package.json
- /pages
    _middleware.ts # Will run on all routes under /pages
    index.tsx
    about.tsx
    teams.tsx
```

If you *do* have sub-directories with nested routes, Middleware will run from the top down. For example, if you have `/pages/about/_middleware.ts` and `/pages/about/team/_middleware.ts`, `/about` will run first and then `/about/team`. The below example shows how this works with a nested routing structure.

```
- package.json
- /pages
    index.tsx
    - /about
      _middleware.ts # Will run first
      about.tsx
      - /teams
        _middleware.ts # Will run second
        teams.tsx
```

Middleware runs directly after `redirects` and `headers`, before the first filesystem lookup. This excludes `/_next` files.

Deployment

Middleware uses a strict runtime that supports standard Web APIs like `fetch`. This works out of the box using `next start`, as well as on Edge platforms like Vercel, which use Edge Functions.

Related

Middleware API Reference Learn more about the supported APIs for Middleware.

Edge Runtime Learn more about the supported Web APIs available.