Middleware

Middleware allows you to run code before a request is completed. Then, based on the incoming request, you can modify the response by rewriting, redirecting, modifying the request or response headers, or responding directly.

Middleware runs before cached content and routes are matched. See Matching Paths for more details.

Convention

Use the file `middleware.ts` (or `.js`) in the root of your project to define Middleware. For example, at the same level as `pages` or `app`, or inside `src` if applicable.

Example

middleware.ts

```ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

// This function can be marked `async` if using `await` inside
export function middleware(request: NextRequest) {
  return NextResponse.redirect(new URL('/home', request.url));
}

// See "Matching Paths" below to learn more
export const config = {
  matcher: '/about/:path*',
};
```

Matching Paths

Middleware will be invoked for **every route in your project**. The following is the execution order:

1. `headers` from `next.config.js`
2. `redirects` from `next.config.js`
3. Middleware (`rewrites`, `redirects`, etc.)
4. `beforeFiles` (`rewrites`) from `next.config.js`
5. Filesystem routes (`public/`, `_next/static/`, `pages/`, `app/`, etc.)
6. `afterFiles` (`rewrites`) from `next.config.js`
7. Dynamic Routes (`/blog/[slug]`)
8. `fallback` (`rewrites`) from `next.config.js`

There are two ways to define which paths Middleware will run on:

1. Custom matcher config
2. Conditional statements

Matcher

`matcher` allows you to filter Middleware to run on specific paths.

middleware.js

```
export const config = {
  matcher: '/about/:path*',
};
```

You can match a single path or multiple paths with an array syntax:

middleware.js

```
export const config = {
  matcher: ['/about/:path*', '/dashboard/:path*'],
};
```

The `matcher` config allows full regex so matching like negative lookaheads or character matching is supported. An example of a negative lookahead to match all except specific paths can be seen here:

middleware.js

```js
export const config = {
  matcher: [
    /*
     * Match all request paths except for the ones starting with:
     * - api (API routes)
     * - _next/static (static files)
     * - _next/image (image optimization files)
     * - favicon.ico (favicon file)
     */
    '/((?!api|_next/static|_next/image|favicon.ico).*)',
  ],
};
```

Note: The `matcher` values need to be constants so they can be statically analyzed at build-time. Dynamic values such as variables will be ignored.

Configured matchers:

1. MUST start with `/`
2. Can include named parameters: `/about/:path` matches `/about/a` and `/about/b` but not `/about/a/c`
3. Can have modifiers on named parameters (starting with `:`): `/about/:path*` matches `/about/a/b/c` because `*` is *zero or more*. `?` is *zero or one* and `+` *one or more*
4. Can use regular expression enclosed in parenthesis: `/about/(.*)` is the same as `/about/:path*`

Read more details on path-to-regexp↗ documentation.

Note: For backward compatibility, Next.js always considers `/public` as `/public/index`. Therefore, a matcher of `/public/:path` will match.

Conditional Statements

middleware.ts

```ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

export function middleware(request: NextRequest) {
  if (request.nextUrl.pathname.startsWith('/about')) {
    return NextResponse.rewrite(new URL('/about-2', request.url));
  }
  if (request.nextUrl.pathname.startsWith('/dashboard')) {
    return NextResponse.rewrite(new URL('/dashboard/user', request.url));
  }
}
```

NextResponse

The `NextResponse` API allows you to:

- `redirect` the incoming request to a different URL
- `rewrite` the response by displaying a given URL
- Set request headers for API Routes, `getServerSideProps`, and `rewrite` destinations
- Set response cookies
- Set response headers

To produce a response from Middleware, you can:

1. `rewrite` to a route (Page or Edge API Route) that produces a response
2. return a `NextResponse` directly. See Producing a Response

Using Cookies

Cookies are regular headers. On a `Request`, they are stored in the `Cookie` header. On a `Response` they are in the `Set-Cookie` header. Next.js provides a convenient way to access and manipulate these cookies through the `cookies` extension on `NextRequest` and `NextResponse`.

1. For incoming requests, `cookies` comes with the following methods: `get`, `getAll`, `set`, and `delete` cookies. You can check for the existence of a cookie with `has` or remove all cookies with `clear`.
2. For outgoing responses, `cookies` have the following methods `get`, `getAll`, `set`, and `delete`.

middleware.ts

```ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

export function middleware(request: NextRequest) {
  // Assume a "Cookie:nextjs=fast" header to be present on the incoming request
  // Getting cookies from the request using the `RequestCookies` API
  let cookie = request.cookies.get('nextjs')?.value;
  console.log(cookie); // => 'fast'
  const allCookies = request.cookies.getAll();
  console.log(allCookies); // => [{ name: 'nextjs', value: 'fast' }]

  request.cookies.has('nextjs'); // => true
  request.cookies.delete('nextjs');
  request.cookies.has('nextjs'); // => false

  // Setting cookies on the response using the `ResponseCookies` API
  const response = NextResponse.next();
  response.cookies.set('vercel', 'fast');
  response.cookies.set({
    name: 'vercel',
    value: 'fast',
    path: '/',
  });
  cookie = response.cookies.get('vercel');
  console.log(cookie); // => { name: 'vercel', value: 'fast', Path: '/' }
  // The outgoing response will have a `Set-Cookie:vercel=fast;path=/test` header.

  return response;
}
```

Setting Headers

You can set request and response headers using the `NextResponse` API (setting *request* headers is available since Next.js v13.0.0).

middleware.ts

```ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

export function middleware(request: NextRequest) {
  // Clone the request headers and set a new header `x-hello-from-middleware1`
  const requestHeaders = new Headers(request.headers);
  requestHeaders.set('x-hello-from-middleware1', 'hello');

  // You can also set request headers in NextResponse.rewrite
  const response = NextResponse.next({
    request: {
      // New request headers
      headers: requestHeaders,
    },
  });
  // Set a new response header `x-hello-from-middleware2`
  response.headers.set('x-hello-from-middleware2', 'hello');
  return response;
}
```

Note: Avoid setting large headers as it might cause 431 Request Header Fields Too Large ↗ error depending on your backend web server configuration.

Producing a Response

You can respond from Middleware directly by returning a `Response` or `NextResponse` instance. (This is available since Next.js v13.1.0 ↗)

middleware.ts

```ts
import { NextRequest, NextResponse } from 'next/server';
import { isAuthenticated } from '@lib/auth';

// Limit the middleware to paths starting with `/api/`
export const config = {
  matcher: '/api/:function*',
};

export function middleware(request: NextRequest) {
  // Call our authentication function to check the request
  if (!isAuthenticated(request)) {
    // Respond with JSON indicating an error message
    return new NextResponse(
      JSON.stringify({ success: false, message: 'authentication failed' }),
      { status: 401, headers: { 'content-type': 'application/json' } },
    );
  }
}
```

Advanced Middleware Flags

In `v13.1` of Next.js two additional flags were introduced for middleware, `skipMiddlewareUrlNormalize` and `skipTrailingSlashRedirect` to handle advanced use cases.

`skipTrailingSlashRedirect` allows disabling Next.js default redirects for adding or removing trailing slashes allowing custom handling inside middleware which can allow maintaining the trailing slash for some paths but not others allowing easier incremental migrations.

next.config.js

```js
module.exports = {
  skipTrailingSlashRedirect: true,
};
```

middleware.js

```js
const legacyPrefixes = ['/docs', '/blog'];

export default async function middleware(req) {
  const { pathname } = req.nextUrl;

  if (legacyPrefixes.some((prefix) => pathname.startsWith(prefix))) {
    return NextResponse.next();
  }

  // apply trailing slash handling
  if (
    !pathname.endsWith('/') &&
    !pathname.match(/((?!\.well-known(?:\/.*)?)(?:[^/]+\/)*[^/]+\.\w+)/)
  ) {
    req.nextUrl.pathname += '/';
    return NextResponse.redirect(req.nextUrl);
  }
}
```

`skipMiddlewareUrlNormalize` allows disabling the URL normalizing Next.js does to make handling direct visits and client-transitions the same. There are some advanced cases where you need full control using the original URL which this unlocks.

next.config.js

```js
module.exports = {
  skipMiddlewareUrlNormalize: true,
};
```

middleware.js

```js
export default async function middleware(req) {
  const { pathname } = req.nextUrl;

  // GET /_next/data/build-id/hello.json

  console.log(pathname);
  // with the flag this now /_next/data/build-id/hello.json
  // without the flag this would be normalized to /hello
}
```

Version History

Version	Changes
`v13.1.0`	Advanced Middleware flags added
`v13.0.0`	Middleware can modify request headers, response headers, and send responses
`v12.2.0`	Middleware is stable, please see the upgrade guide
`v12.0.9`	Enforce absolute URLs in Edge Runtime (PR ↗)
`v12.0.0`	Middleware (Beta) added