

[> Menu](#)

middleware.js

The `middleware.js|ts` file is used to write [Middleware](#) and run code on the server 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 executes before routes are rendered. It's particularly useful for implementing custom server-side logic like authentication, logging, or handling redirects.

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



middleware.ts

TypeScript ▾



```
1  import { NextResponse, NextRequest } from 'next/server'
2
3  // This function can be marked `async` if using `await` inside
4  export function middleware(request: NextRequest) {
5    return NextResponse.redirect(new URL('/home', request.url))
6  }
7
8  export const config = {
9    matcher: '/about/:path*',
10 }
```

Exports

Middleware function

The file must export a single function, either as a default export or named `middleware`. Note that multiple middleware from the same file are not supported.

```
JS middleware.js

1 // Example of default export
2 export default function middleware(request) {
3   // Middleware logic
4 }
```

Config object (optional)

Optionally, a config object can be exported alongside the Middleware function. This object includes the `matcher` to specify paths where the Middleware applies.

Matcher

The `matcher` option allows you to target specific paths for the Middleware to run on. You can specify these paths in several ways:

- For a single path: Directly use a string to define the path, like `'/about'` .
- For multiple paths: Use an array to list multiple paths, such as `matcher: ['/about', '/contact']`, which applies the Middleware to both `/about` and `/contact` .

Additionally, `matcher` supports complex path specifications through regular expressions, such as `matcher: ['/((?!api|_next/static|_next/image|.*\\.png$).*)']`, enabling precise control over which paths to include or exclude.

The `matcher` option also accepts an array of objects with the following keys:

- `source`: The path or pattern used to match the request paths. It can be a string for direct path matching or a pattern for more complex matching.
- `regex` (optional): A regular expression string that fine-tunes the matching based on the source. It provides additional control over which paths are included or excluded.

- `locale` (optional): A boolean that, when set to `false`, ignores locale-based routing in path matching.
- `has` (optional): Specifies conditions based on the presence of specific request elements such as headers, query parameters, or cookies.
- `missing` (optional): Focuses on conditions where certain request elements are absent, like missing headers or cookies.

JS middleware.js



```
1  export const config = {
2    matcher: [
3      {
4        source: '/api/*',
5        regexp: '^/api/(.*)',
6        locale: false,
7        has: [
8          { type: 'header', key: 'Authorization', value: 'Bearer Token' },
9          { type: 'query', key: 'userId', value: '123' },
10       ],
11       missing: [{ type: 'cookie', key: 'session', value: 'active' }],
12     },
13   ],
14 }
```

Params

request

When defining Middleware, the default export function accepts a single parameter, `request`. This parameter is an instance of `NextRequest`, which represents the incoming HTTP request.

TS middleware.ts

TypeScript ▾



```
1  import type { NextRequest } from 'next/server'
2
3  export function middleware(request: NextRequest) {
```

```
4    // Middleware logic goes here
5  }
```

Good to know:

- `NextRequest` is a type that represents incoming HTTP requests in Next.js Middleware, whereas `NextResponse` is a class used to manipulate and send back HTTP responses.

NextResponse

Middleware can use the `NextResponse` object which extends the [Web Response API](#). By returning a `NextResponse` object, you can directly manipulate cookies, set headers, implement redirects, and rewrite paths.

Good to know: For redirects, you can also use `Response.redirect` instead of `NextResponse.redirect`.

Runtime

Middleware only supports the [Edge runtime](#). The Node.js runtime cannot be used.

Version History

Version	Changes
v13.1.0	Advanced Middleware flags added
v13.0.0	Middleware can modify request headers, response headers, and send responses

Version	Changes
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

Learn more about Middleware

App Router > ... > Routing

Middleware





Learn how to use Middleware to run code before a request is completed.


Previous

< mdx-components.js

Next

not-found.js >

Was this helpful?    

	Resources	More	About Vercel	Legal
	Docs	Next.js Commerce	Next.js + Vercel	Privacy Policy
	Learn	Contact Sales	Open Source Software	
	Showcase	GitHub	GitHub	
	Blog	Releases	X	
	Analytics	Telemetry		

Subscribe to our newsletter

Stay updated on new releases and features, guides, and case studies.

you@domain.com

Subscribe

© 2024 Vercel, Inc.

