Getting Started > Session Management > Protecting Resources

Protecting Resources

Protecting routes can be done generally by checking for the session and taking an action if an active session is not found, like redirecting the user to the login page or simply returning a `401: Unauthenticated` response.

Pages

Next.js Next.js (Client) Qwik SvelteKit Express

You can use the `auth` function returned from `NextAuth()` and exported from your `auth.ts` or `auth.js` configuration file to get the session object.

app/server/page.tsx

```tsx
import { auth } from "@/auth"

export default async function Page() {
  const session = await auth()
  if (!session) return <div>Not authenticated</div>

  return (
    <div>
      <pre>{JSON.stringify(session, null, 2)}</pre>
    </div>
  )
}
```

API Routes

Protecting API routes in the various frameworks can also be done with the `auth` export.

Next.js Next.js (Client) Qwik SvelteKit Express

In Next.js, you can use the `auth` function to wrap an API route handler. The request parameter will then have an `auth` key on it which you can check for a valid session.

./app/api/admin/route.ts

```ts
import { auth } from "@/auth"
import { NextResponse } from "next/server"

export const GET = auth(function GET(req) {
  if (req.auth) return NextResponse.json(req.auth)
  return NextResponse.json({ message: "Not authenticated" }, { status: 401 })
})
```

Next.js Middleware

With Next.js 12+, the easiest way to protect a set of pages is using the middleware file. You can create a `middleware.ts` file in your root pages directory with the following contents.

middleware.ts

```ts
export { auth as middleware } from "@/auth"
```

Then define `authorized` callback in your `auth.ts` file. For more details check out the reference docs.

auth.ts

```ts
import NextAuth from "next-auth"

export const { auth, handlers } = NextAuth({
  callbacks: {
    authorized: async ({ auth }) => {
      // Logged in users are authenticated, otherwise redirect to login page
      return !!auth
    },
  },
})
```

You can also use the `auth` method as a wrapper if you'd like to implement more logic inside the middleware.

middleware.ts

```
import { auth } from "@/auth"

export default auth((req) => {
  if (!req.auth && req.nextUrl.pathname !== "/login") {
    const newUrl = new URL("/login", req.nextUrl.origin)
    return Response.redirect(newUrl)
  }
})
```

You can also use a regex to match multiple routes or you can negate certain routes in order to protect all remaining routes. The following example avoids running the middleware on paths such as the favicon or static images.

middleware.ts

```
export const config = {
  matcher: ["/((?!api|_next/static|_next/image|favicon.ico).*)"],
}
```

Middleware will protect pages as defined by the `matcher` config export. For more details about the matcher, check out the Next.js docs.

💡 You should not rely on middleware exclusively for authorization. Always ensure that the session is verified as close to your data fetching as possible.

Last updated on January 9, 2025

About Auth.js

Introduction

Security

Discord Community

Download

GitHub

NPM

Acknowledgements

Contributors

Sponsors