Version: v4

TypeScript

NextAuth.js has its own type definitions to use in your TypeScript projects safely. Even if you don't use TypeScript, IDEs like VSCode will pick this up to provide you with a better developer experience. While you are typing, you will get suggestions about what certain objects/functions look like, and sometimes links to documentation, examples, and other valuable resources.

Check out the example repository showcasing how to use `next-auth` on a Next.js application with TypeScript:
https://github.com/nextauthjs/next-auth-example

Adapters

If you're writing your own custom Adapter, you can take advantage of the types to make sure your implementation conforms to what's expected:

```
import type { Adapter } from "next-auth/adapters"

function MyAdapter(): Adapter {
  return {
    // your adapter methods here
  }
}
```

When writing your own custom Adapter in plain JavaScript, note that you can use **JSDoc** to get helpful editor hints and auto-completion like so:

```
/** @return { import("next-auth/adapters").Adapter } */
function MyAdapter() {
  return {
    // your adapter methods here
```

```
  }
}
```

> **NOTE**
>
> This will work in code editors with a strong TypeScript integration like VSCode or WebStorm. It might not work if you're using more lightweight editors like VIM or Atom.

Module Augmentation

`next-auth` comes with certain types/interfaces that are shared across submodules. Good examples are `Session` and `JWT`. Ideally, you should only need to create these types at a single place, and TS should pick them up in every location where they are referenced. Luckily, Module Augmentation is exactly that, which can do this for us. Define your shared interfaces in a single place, and get type-safety across your application when using `next-auth` (or one of its submodules).

Main module

Let's look at `Session`:

pages/api/auth/[...nextauth].ts

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

export default NextAuth({
  callbacks: {
    session({ session, token, user }) {
      return session // The return type will match the one returned in
`useSession()`
    },
  },
})
```

pages/index.ts

```
import { useSession } from "next-auth/react"

export default function IndexPage() {
  // `session` will match the returned value of `callbacks.session()` from
`NextAuth()`
  const { data: session } = useSession()

  return (
    // Your component
  )
}
```

To extend/augment this type, create a `types/next-auth.d.ts` file in your project:

types/next-auth.d.ts

```
import NextAuth from "next-auth"

declare module "next-auth" {
  /**
   * Returned by `useSession`, `getSession` and received as a prop on the
`SessionProvider` React Context
   */
  interface Session {
    user: {
      /** The user's postal address. */
      address: string
    }
  }
}
```

Extend default interface properties

By default, TypeScript will merge new interface properties and overwrite existing ones. In this case, the default session user properties will be overwritten, with the new one defined above.

If you want to keep the default session user properties, you need to add them back into the newly declared interface:

types/next-auth.d.ts

```ts
import NextAuth, { DefaultSession } from "next-auth"

declare module "next-auth" {
  /**
   * Returned by `useSession`, `getSession` and received as a prop on the
`SessionProvider` React Context
   */
  interface Session {
    user: {
      /** The user's postal address. */
      address: string
    } & DefaultSession["user"]
  }
}
```

Popular interfaces to augment

Although you can augment almost anything, here are some of the more common interfaces that you might want to override in the `next-auth` module:

```ts
/**
 * The shape of the user object returned in the OAuth providers' `profile`
callback,
 * or the second parameter of the `session` callback, when using a database.
 */
interface User {}
/**
 * Usually contains information about the provider being used
 * and also extends `TokenSet`, which is different tokens returned by OAuth
Providers.
 */
interface Account {}
/** The OAuth profile returned from your provider */
interface Profile {}
```

Make sure that the `types` folder is added to `typeRoots` in your project's `tsconfig.json` file.

Submodules

The `JWT` interface can be found in the `next-auth/jwt` submodule:

types/next-auth.d.ts

```ts
import { JWT } from "next-auth/jwt"

declare module "next-auth/jwt" {
  /** Returned by the `jwt` callback and `getToken`, when using JWT sessions */
  interface JWT {
    /** OpenID ID Token */
    idToken?: string
  }
}
```

Useful links

1. TypeScript documentation: Module Augmentation
2. Digital Ocean: Module Augmentation in TypeScript

Contributing

Contributions of any kind are always welcome, especially for TypeScript. Please keep in mind that we are a small team working on this project in our free time. We will try our best to give support, but if you think you have a solution for a problem, please open a PR!

> **NOTE**
>
> When contributing to TypeScript, if the actual JavaScript user API does not change in a breaking manner, we reserve the right to push any TypeScript change in a minor release. This ensures that we can keep on a faster release cycle.

Edit this page

*Last updated on **May 16, 2024** by **Codie Newark***