Version: v4

Options

Environment Variables

NEXTAUTH_URL

When deploying to production, set the `NEXTAUTH_URL` environment variable to the canonical URL of your site.

```
NEXTAUTH_URL=https://example.com
```

If your Next.js application uses a custom base path, specify the route to the API endpoint in full. More information about the usage of custom base path here.

e.g. `NEXTAUTH_URL=https://example.com/custom-route/api/auth`

> **TIP**
>
> When you're using a custom base path, you will need to pass the `basePath` page prop to the `<SessionProvider>`. More information here.

> **NOTE**
>
> Using System Environment Variables we automatically detect when you deploy to Vercel so you don't have to define this variable. Make sure **Automatically expose System Environment Variables** is checked in your Project Settings.

NEXTAUTH_SECRET

Used to encrypt the NextAuth.js JWT, and to hash email verification tokens. This is the default value for the `secret` option in NextAuth and Middleware.

NEXTAUTH_URL_INTERNAL

If provided, server-side calls will use this instead of `NEXTAUTH_URL`. Useful in environments when the server doesn't have access to the canonical URL of your site. Defaults to `NEXTAUTH_URL`.

```
NEXTAUTH_URL_INTERNAL=http://10.240.8.16
```

Options

Options are passed to NextAuth.js when initializing it in an API route.

providers

- **Default value**: `[]`
- **Required**: *Yes*

Description

An array of authentication providers for signing in (e.g. Google, Facebook, Twitter, GitHub, Email, etc) in any order. This can be one of the built-in providers or an object with a custom provider.

See the providers documentation for a list of supported providers and how to use them.

secret

- **Default value**: `string` (*SHA hash of the "options" object*) in development, no default in production.
- **Required**: *Yes, in production!*

Description

A random string is used to hash tokens, sign/encrypt cookies and generate cryptographic keys.

If you set `NEXTAUTH_SECRET` as an environment variable, you don't have to define this option.

If no value is specified in development (and there is no `NEXTAUTH_SECRET` variable either), it uses a hash for all configuration options, including OAuth Client ID / Secrets for entropy.

> **DANGER**
>
> Not providing any `secret` or `NEXTAUTH_SECRET` will throw an error in production.

You can quickly create a good value on the command line via this `openssl` command.

```
$ openssl rand -base64 32
```

> **TIP**
>
> If you rely on the default secret generation in development, you might notice JWT decryption errors, since the secret changes whenever you change your configuration. Defining an explicit secret will make this problem go away. We will likely make this option mandatory, even in development, in the future.

session

- **Default value**: `object`
- **Required**: *No*

Description

The `session` object and all properties on it are optional.

Default values for this option are shown below:

```
session: {
  // Choose how you want to save the user session.
  // The default is `"jwt"`, an encrypted JWT (JWE) stored in the session cookie.
  // If you use an `adapter` however, we default it to `"database"` instead.
```

```
  // You can still force a JWT session by explicitly defining `"jwt"`.
  // When using `"database"`, the session cookie will only contain a
`sessionToken` value,
  // which is used to look up the session in the database.
  strategy: "database",

  // Seconds - How long until an idle session expires and is no longer valid.
  maxAge: 30 * 24 * 60 * 60, // 30 days

  // Seconds - Throttle how frequently to write to database to extend a session.
  // Use it to limit write operations. Set to 0 to always update the database.
  // Note: This option is ignored if using JSON Web Tokens
  updateAge: 24 * 60 * 60, // 24 hours

  // The session token is usually either a random UUID or string, however if you
  // need a more customized session token string, you can define your own
generate function.
  generateSessionToken: () => {
    return randomUUID?.() ?? randomBytes(32).toString("hex")
  }
}
```

jwt

- **Default value**: `object`
- **Required**: *No*

Description

JSON Web Tokens can be used for session tokens if enabled with `session: { strategy: "jwt" }` option. JSON Web Tokens are enabled by default if you have not specified an adapter. JSON Web Tokens are encrypted (JWE) by default. We recommend you keep this behaviour. See the **Override JWT `encode` and `decode` methods** advanced option.

JSON Web Token Options

```
jwt: {
  // The maximum age of the NextAuth.js issued JWT in seconds.
  // Defaults to `session.maxAge`.
```

```
  maxAge: 60 * 60 * 24 * 30,
  // You can define your own encode/decode functions for signing and encryption
  async encode() {},
  async decode() {},
}
```

An example JSON Web Token contains a payload like this:

```
{
  name: 'Iain Collins',
  email: 'me@iaincollins.com',
  picture: 'https://example.com/image.jpg',
  iat: 1594601838,
  exp: 1597193838
}
```

JWT Helper

You can use the built-in `getToken()` helper method to verify and decrypt the token, like this:

```
import { getToken } from "next-auth/jwt"

const secret = process.env.NEXTAUTH_SECRET

export default async function handler(req, res) {
  // if using `NEXTAUTH_SECRET` env variable, we detect it, and you won't
actually need to `secret`
  // const token = await getToken({ req })
  const token = await getToken({ req, secret })
  console.log("JSON Web Token", token)
  res.end()
}
```

For convenience, this helper function is also able to read and decode tokens passed from the `Authorization: 'Bearer token'` HTTP header.

Required

The getToken() helper requires the following options:

- `req` - (object) Request object
- `secret` - (string) JWT Secret. Use `NEXTAUTH_SECRET` instead.

You must also pass *any options configured on the* `jwt` *option* to the helper.

e.g. Including custom session `maxAge` and custom signing and/or encryption keys or options

Optional

It also supports the following options:

- `secureCookie` - (boolean) Use secure prefixed cookie name

  By default, the helper function will attempt to determine if it should use the secure prefixed cookie (e.g. `true` in production and `false` in development, unless NEXTAUTH_URL contains an HTTPS URL).

- `cookieName` - (string) Session token cookie name

  The `secureCookie` option is ignored if `cookieName` is explicitly specified.

- `raw` - (boolean) Get raw token (not decoded)

  If set to `true` returns the raw token without decrypting or verifying it.

> **TIP**
>
> When using a custom session token `cookieName`, the same name should also be provided to `getToken`. If you are using the Next.js `withAuth` middleware, you will also need to configure this using the same `cookieName`.

> **NOTE**
>
> The JWT is stored in the Session Token cookie, the same cookie used for tokens with database sessions.

pages

- **Default value**: `{}`
- **Required**: *No*

Description

Specify URLs to be used if you want to create custom sign in, sign out and error pages.

Pages specified will override the corresponding built-in page.

For example:

```
pages: {
  signIn: '/auth/signin',
  signOut: '/auth/signout',
  error: '/auth/error', // Error code passed in query string as ?error=
  verifyRequest: '/auth/verify-request', // (used for check email message)
  newUser: '/auth/new-user' // New users will be directed here on first sign in
(leave the property out if not of interest)
}
```

> **NOTE**
>
> When using this configuration, ensure that these pages actually exist. For example `error: '/auth/error'` refers to a page file at `pages/auth/error.js`.

See the documentation for the pages option for more information.

callbacks

- **Default value**: `object`
- **Required**: *No*

Description

Callbacks are asynchronous functions you can use to control what happens when an action is performed.

Callbacks are extremely powerful, especially in scenarios involving JSON Web Tokens as they allow you to implement access controls without a database and to integrate with external databases or APIs.

You can specify a handler for any of the callbacks below.

```
callbacks: {
  async signIn({ user, account, profile, email, credentials }) {
    return true
  },
  async redirect({ url, baseUrl }) {
    return baseUrl
  },
  async session({ session, token, user }) {
    return session
  },
  async jwt({ token, user, account, profile, isNewUser }) {
    return token
  }
}
```

See the callbacks documentation for more information on how to use the callback functions.

events

- **Default value**: `object`
- **Required**: *No*

Description

Events are asynchronous functions that do not return a response, they are useful for audit logging.

You can specify a handler for any of these events below - e.g. for debugging or to create an audit log.

The content of the message object varies depending on the flow (e.g. OAuth or Email authentication flow, JWT or database sessions, etc). See the events documentation for more information on the form of each message object and how to use the events functions.

```
events: {
  async signIn(message) { /* on successful sign in */ },
  async signOut(message) { /* on signout */ },
  async createUser(message) { /* user created */ },
  async updateUser(message) { /* user updated - e.g. their email was verified */
},
  async linkAccount(message) { /* account (e.g. Twitter) linked to a user */ },
  async session(message) { /* session is active */ },
}
```

adapter

- **Default value**: none
- **Required**: *No*

Description

By default NextAuth.js does not include an adapter any longer. If you would like to persist user / account data, please install one of the many available adapters. More information can be found in the adapter documentation.

debug

- **Default value**: `false`
- **Required**: *No*

Description

Set debug to `true` to enable debug messages for authentication and database operations.

logger

- **Default value**: `console`
- **Required**: *No*

Description

Override any of the logger levels (`undefined` levels will use the built-in logger), and intercept logs in NextAuth.js. You can use this to send NextAuth.js logs to a third-party logging service.

The `code` parameter for `error` and `warn` are explained in the Warnings and Errors pages respectively.

Example:

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

```
import log from "logging-service"

export default NextAuth({
  ...
  logger: {
    error(code, metadata) {
      log.error(code, metadata)
    },
    warn(code) {
      log.warn(code)
    },
    debug(code, metadata) {
      log.debug(code, metadata)
    }
  }
  ...
})
```

NOTE

If the `debug` level is defined by the user, it will be called regardless of the `debug: false` option.

theme

- **Default value**: `object`
- **Required**: *No*

Description

Changes the color scheme theme of pages as well as allows some minor customization. Set `theme.colorScheme` to `"light"`, if you want to force pages to always be light. Set to `"dark"`, if you want to force pages to always be dark. Set to `"auto"`, (or leave this option out) if you want the pages to follow the preferred system theme. (Uses the prefers-color-scheme media query.)

In addition, you can define a logo URL in `theme.logo` which will be rendered above the main card in the default signin/signout/error/verify-request pages, as well as a `theme.brandColor` which will affect the accent color of these pages.

The sign-in button's background color will match the `brandColor` and defaults to `"#346df1"`. The text color is `#fff` by default, but if your brand color gives a weak contrast, correct it with the `buttonText` color option.

```
theme: {
  colorScheme: "auto", // "auto" | "dark" | "light"
  brandColor: "", // Hex color code
  logo: "", // Absolute URL to image
  buttonText: "" // Hex color code
}
```

Advanced Options

Advanced options are passed the same way as basic options, but may have complex implications or side effects. You should try to avoid using advanced options unless you are very comfortable using them.

useSecureCookies

- **Default value**: `true` for HTTPS sites / `false` for HTTP sites
- **Required**: *No*

Description

When set to `true` (the default for all site URLs that start with `https://`) then all cookies set by NextAuth.js will only be accessible from HTTPS URLs.

This option defaults to `false` on URLs that start with `http://` (e.g. `http://localhost:3000`) for developer convenience.

> **NOTE**
>
> Properties on any custom `cookies` that are specified override this option.

> **DANGER**
>
> Setting this option to *false* in production is a security risk and may allow sessions to be hijacked if used in production. It is intended to support development and testing. Using this option is not recommended.

cookies

- **Default value**: `{}`
- **Required**: *No*

Description

Cookies in NextAuth.js are chunked by default, meaning that once they reach the 4kb limit, we will create a new cookie with the `.{number}` suffix and reassemble the cookies in the correct order when parsing / reading them. This was introduced to avoid size constraints which can occur when users want to store additional data in their sessionToken, for example.

You can override the default cookie names and options for any of the cookies used by NextAuth.js.

This is an advanced option and using it is not recommended as you may break authentication or introduce security flaws into your application.

You can specify one or more cookies with custom properties, but if you specify custom options for a cookie you must provide all the options for that cookie.

If you use this feature, you will likely want to create conditional behaviour to support setting different cookies policies in development and production builds, as you will be opting out of the built-in dynamic policy.

> **TIP**
>
> An example of a use case for this option is to support sharing session tokens across subdomains.

Example

```
cookies: {
  sessionToken: {
    name: `__Secure-next-auth.session-token`,
    options: {
      httpOnly: true,
      sameSite: 'lax',
      path: '/',
      secure: true
    }
  },
  callbackUrl: {
    name: `__Secure-next-auth.callback-url`,
    options: {
      sameSite: 'lax',
      path: '/',
      secure: true
    }
  },
  csrfToken: {
    name: `__Host-next-auth.csrf-token`,
    options: {
      httpOnly: true,
      sameSite: 'lax',
      path: '/',
```

```
      secure: true
    }
  },
  pkceCodeVerifier: {
    name: `${cookiePrefix}next-auth.pkce.code_verifier`,
    options: {
      httpOnly: true,
      sameSite: 'lax',
      path: '/',
      secure: true,
      maxAge: 900
    }
  },
  state: {
    name: `${cookiePrefix}next-auth.state`,
    options: {
      httpOnly: true,
      sameSite: "lax",
      path: "/",
      secure: true,
      maxAge: 900
    },
  },
  nonce: {
    name: `${cookiePrefix}next-auth.nonce`,
    options: {
      httpOnly: true,
      sameSite: "lax",
      path: "/",
      secure: true,
    },
  },
}
```

DANGER

Using a custom cookie policy may introduce security flaws into your application and is intended as an option for advanced users who understand the implications. Using this option is not recommended.

Override JWT `encode` and `decode` methods

NextAuth.js uses encrypted JSON Web Tokens (JWE) by default. Unless you have a good reason, we recommend keeping this behaviour. Although you can override this using the `encode` and `decode` methods. Both methods must be defined at the same time.

IMPORTANT: If you use middleware to protect routes, make sure the same method is also set in the `_middleware.ts` options

```
jwt: {
  async encode(params: {
    token: JWT
    secret: string
    maxAge: number
  }): Promise<string> {
    // return a custom encoded JWT string
    return
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gR
  },
  async decode(params: {
    token: string
    secret: string
  }): Promise<JWT | null> {
    // return a `JWT` object, or `null` if decoding failed
    return {}
  },
}
```

Edit this page

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