



Providers

Email

Version: v4

Email

Overview

The Email provider uses email to send "magic links" that can be used to sign in, you will likely have seen these if you have used services like Slack before.

Adding support for signing in via email in addition to one or more OAuth services provides a way for users to sign in if they lose access to their OAuth account (e.g. if it is locked or deleted).

The Email provider can be used in conjunction with (or instead of) one or more OAuth providers.

How it works

On initial sign in, a **Verification Token** is sent to the email address provided. By default this token is valid for 24 hours. If the Verification Token is used within that time (i.e. by clicking on the link in the email) an account is created for the user and they are signed in.

If someone provides the email address of an *existing account* when signing in, an email is sent and they are signed into the account associated with that email address when they follow the link in the email.

**TIP**

The Email Provider can be used with both JSON Web Tokens and database sessions, but you **must** configure a database to use it. It is not possible to enable email sign in without using a database.

Options

The **Email Provider** comes with a set of default options:

- [Email Provider options](#)

You can override any of the options to suit your own use case.

Configuration

NextAuth.js lets you send emails either via HTTP or SMTP.

HTTP

Check out our [HTTP-based Email Provider](#) guide.

SMTP

1. NextAuth.js does not include `nodemailer` as a dependency, so you'll need to install it yourself if you want to use the Email Provider. Run `npm install nodemailer` or `yarn add nodemailer`.
2. You will need an SMTP account; ideally for one of the [services known to work with nodemailer](#).
3. There are two ways to configure the SMTP server connection.

You can either use a connection string or a `nodemailer` configuration object or transport.

2.1 Using a connection string

Create an `.env` file to the root of your project and add the connection string and email address.

`.env`

```
EMAIL_SERVER=smtp://username:password@smtp.example.com:587
EMAIL_FROM=noreply@example.com
```

Now you can add the email provider like this:

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

```
import EmailProvider from "next-auth/providers/email";
...
providers: [
  EmailProvider({
    server: process.env.EMAIL_SERVER,
    from: process.env.EMAIL_FROM
  }),
],
```

2.2 Using a configuration object

In your `.env` file in the root of your project simply add the configuration object options individually:

`.env`

```
EMAIL_SERVER_USER=username
EMAIL_SERVER_PASSWORD=password
EMAIL_SERVER_HOST=smtp.example.com
EMAIL_SERVER_PORT=587
EMAIL_FROM=noreply@example.com
```

Now you can add the provider settings to the NextAuth.js options object in the Email Provider.

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

```
import EmailProvider from "next-auth/providers/email";
...
providers: [
  EmailProvider({
    server: {
      host: process.env.EMAIL_SERVER_HOST,
      port: process.env.EMAIL_SERVER_PORT,
      auth: {
        user: process.env.EMAIL_SERVER_USER,
        pass: process.env.EMAIL_SERVER_PASSWORD
      }
    },
    from: process.env.EMAIL_FROM
```

```
    }},  
  ],
```

3. Do not forget to setup one of the database [adapters](#) for storing the Email verification token.

4. You can now sign in with an email address at `/api/auth/signin`.

A user account (i.e. an entry in the Users table) will not be created for the user until the first time they verify their email address. If an email address is already associated with an account, the user will be signed in to that account when they use the link in the email.

Customizing emails

You can fully customize the sign in email that is sent by passing a custom function as the `sendVerificationRequest` option to `EmailProvider()`.

e.g.

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

```
import EmailProvider from "next-auth/providers/email";  
...  
providers: [  
  EmailProvider({  
    server: process.env.EMAIL_SERVER,  
    from: process.env.EMAIL_FROM,  
    sendVerificationRequest({  
      identifier: email,  
      url,  
      provider: { server, from },  
    }) {  
      /* your function */  
    },  
  }),  
],
```

The following code shows the complete source for the built-in `sendVerificationRequest()` method:

```
import { createTransport } from "nodemailer"

async function sendVerificationRequest(params) {
  const { identifier, url, provider, theme } = params
  const { host } = new URL(url)
  // NOTE: You are not required to use `nodemailer`, use whatever you want.
  const transport = createTransport(provider.server)
  const result = await transport.sendMail({
    to: identifier,
    from: provider.from,
    subject: `Sign in to ${host}`,
    text: text({ url, host }),
    html: html({ url, host, theme }),
  })
  const failed = result.rejected.concat(result.pending).filter(Boolean)
  if (failed.length) {
    throw new Error(`Email(s) (${failed.join(", ")}) could not be sent`)
  }
}

/**
 * Email HTML body
 * Insert invisible space into domains from being turned into a hyperlink by
email
 * clients like Outlook and Apple mail, as this is confusing because it seems
 * like they are supposed to click on it to sign in.
 *
 * @note We don't add the email address to avoid needing to escape it, if you do,
remember to sanitize it!
 */
function html(params: { url: string, host: string, theme: Theme }) {
  const { url, host, theme } = params

  const escapedHost = host.replace(/\.\/g, "&#8203;.")

  const brandColor = theme.brandColor || "#346df1"
  const color = {
    background: "#f9f9f9",
    text: "#444",
    mainBackground: "#fff",
    buttonBackground: brandColor,
    buttonBorder: brandColor,
    buttonText: theme.buttonText || "#fff",
  }
```

```

    }

    return `
<body style="background: ${color.background};">
  <table width="100%" border="0" cellspacing="20" cellpadding="0"
    style="background: ${color.mainBackground}; max-width: 600px; margin: auto;
border-radius: 10px;">
    <tr>
      <td align="center"
        style="padding: 10px 0px; font-size: 22px; font-family: Helvetica, Arial,
sans-serif; color: ${color.text};">
        Sign in to <strong>${escapedHost}</strong>
      </td>
    </tr>
    <tr>
      <td align="center" style="padding: 20px 0;">
        <table border="0" cellspacing="0" cellpadding="0">
          <tr>
            <td align="center" style="border-radius: 5px;"
bgcolor="${color.buttonBackground}"><a href="${url}"
              target="_blank"
              style="font-size: 18px; font-family: Helvetica, Arial, sans-
serif; color: ${color.buttonText}; text-decoration: none; border-radius: 5px;
padding: 10px 20px; border: 1px solid ${color.buttonBorder}; display: inline-
block; font-weight: bold;">Sign
                in</a></td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td align="center"
        style="padding: 0px 0px 10px 0px; font-size: 16px; line-height: 22px;
font-family: Helvetica, Arial, sans-serif; color: ${color.text};">
        If you did not request this email you can safely ignore it.
      </td>
    </tr>
  </table>
</body>
`
  }

```

```

/** Email Text body (fallback for email clients that don't render HTML, e.g.
feature phones) */

```

```
function text({ url, host }: { url: string, host: string }) {  
  return `Sign in to ${host}\n${url}\n\n`  
}
```

**TIP**

If you want to generate great looking email client compatible HTML with React, check out <https://mjml.io>

Customizing the Verification Token

By default, we are generating a random verification token. You can define a `generateVerificationToken` method in your provider options if you want to override it:

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

```
providers: [  
  EmailProvider({  
    async generateVerificationToken() {  
      return "ABC123"  
    }  
  })  
],
```

Normalizing the email address

By default, NextAuth.js will normalize the email address. It treats values as case-insensitive (which is technically not compliant to the [RFC 2821 spec](#), but in practice this causes more problems than it solves, eg. when looking up users by e-mail from databases.) and also removes any secondary email address that was passed in as a comma-separated list. You can apply your own normalization via the `normalizeIdentifier` method on the `EmailProvider`. The following example shows the default behavior:

```
EmailProvider({  
  // ...
```

```

normalizeIdentifier(identifier: string): string {
  // Get the first two elements only,
  // separated by `@` from user input.
  let [local, domain] = identifier.toLowerCase().trim().split("@")
  // The part before "@" can contain a ",",
  // but we remove it on the domain part
  domain = domain.split(",")[0]
  return `${local}@${domain}`

  // You can also throw an error, which will redirect the user
  // to the error page with error=EmailSignin in the URL
  // if (identifier.split("@").length > 2) {
  //   throw new Error("Only one email allowed")
  // }
},
})

```

DANGER

Always make sure this returns a single e-mail address, even if multiple ones were passed in.

Sending Magic Links To Existing Users

You can ensure that only existing users are sent a magic login link. You will need to grab the email the user entered and check your database to see if the email already exists in the "User" collection in your database. If it exists, it will send the user a magic link but otherwise, you can send the user to another page, such as "/register".

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


```

import User from "../../../models/User";
import db from "../../../utils/db";
...
callbacks: {
  async signIn({ user, account, email }) {
    await db.connect();
    const userExists = await User.findOne({
      email: user.email, //the user object has an email property, which contains
      the email the user entered.
    });
  }
}

```



```
    });  
    if (userExists) {  
      return true; //if the email exists in the User collection, email them a  
magic login link  
    } else {  
      return "/register";  
    }  
  },  
  ...
```

 [Edit this page](#)

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