

- What Is Email Armor?
 - # Overview
 - # Features In v1
 - # More Features To Be Added Later
 - # Getting Started
 - Installation:-
 - # Usage
 - 1. Sign Up:-
 - 2. Sign Up Verify:-
 - 3. Sign In:-
 - 4. Sign-in Verify:-
 - 5. Auto User Session Check:-
 - 6. Forgot Password:-
 - 7. Resend OTP:-
 - Function 1 (For New Users):-
 - Function 2 (For Old Users):-
 - Function 3 (For Forgot Password):-
 - 8. Custom Email Template:-
 - Feel free to raise an issue if you find any bugs. Thanks in advance! 😊



Email Armor

 Stars  1 license MIT downloads 19/month minified size 1.07 MB

What Is Email Armor?

Note:- Currently, this is under development, so please check beta versions for the latest updates, or refer to latest documentation on [email-armor-github-repo](#).











Note:- Please refer to the documentation on my GitHub repository in case I missed or inaccurately mentioned something here. Documentation for [beta-0.0.2](#).

Overview






Email Armor is a Node.js module that empowers you to create a robust user **sign-up** and **sign-in** system with **two-step verification** using **RESEND**(freemium). It's also equipped with a **built-in referral system** to enhance user engagement and growth.

Note:- It only supports MongoDB as database for now.

Features In v1

-  Compatible with Next.js.
-  Custom Mail Template*
-  Sign-Up With Two-Step Verification.
-  Sign-In With Two-Step Verification.
-  Passwords Are Encrypted With Crypto.
-  Referral System.
-  No Disposable E-Mails Are Allowed To Signup.
-  Resend OTP With Limited Requests.
-  Forgot Password With Two-Step Verification.
-  Auto User Session Checking.

More Features To Be Added Later

-  Lock User After N-Times Failed Login Attempts & Send Notification Email To The User.
-  Unlock The Locked User Account (User + Auto).
-  Add Phone Number In Accounts Model With 2 Step Verification.
-  Change/Update User Info.
-  Delete Account But Make Sure User Don't Get Referral Points Again Once He Sign Up With Any Referral Code.

Getting Started

Installation:-

1. Begin by installing the packages:-

In back-end/front-end, depends on your use, install **mail-paasify**:-

```
npm i email-armor
```

Whereas, in front-end, for fetching cookies, install **cookies-next**:- **Note**:- You can use your own method for fetching cookies. For eg.:- You can also set/fetch cookies via server-side in Next.js, [READ HERE](#).

```
npm i cookies-next
```

2. Configure & Include the following values in your .env file:

```
MONGODB_URI = YOUR_MONGODB_URI (mongodb://127.0.0.1:27017/DB-NAME)
RESEND_API_KEY = YOUR_RESEND_API_KEY
RESEND_EMAIL_ID = YOUR_RESEND_EMAIL_ID
SECRET_KEY = YOUR_SECRET_KEY_FOR_ENCRYPTION_OF_LENGTH_32_OR_GREATER
SECRET_IV = YOUR_SECRET_IV_FOR_ENCRYPTION_OF_LENGTH_32_OR_GREATER
ALLOWED_EMAIL_DOMAINS=@gmail.com,@hotmail.com
{YOU_CAN_ADD_MORE_BY_SEPERATING_WITH_(comma)}
```

3. Generate the configuration file in server by using the command:-

```
npx email-armor init
```

4. This will generate 2 files **email-armor.json** & **email-template.html** files. In **email-armor.json** file, you can configure your data. Please ensure that you maintain the variables in the JSON file as specified below.

Name	Type	Usage
RESEND_SIGN_UP_MAIL_TITLE	String	Custom title for sign-up confirmation.

Name	Type	Usage
RESEND_SIGN_IN_MAIL_TITLE	String	Custom title for sign-in confirmation.
RESEND_FORGOT_PASSWORD_MAIL_TITLE	String	Custom-Forgot-Password-Title.
REFERRED_POINTS	Integer	Points awarded to the referrer.
REFERRED_PERSON_POINTS	Integer	Points awarded to the referred person.
OTP_LIMITS	Integer	Max Times User Can Request For OTP.

5. Once you update these values, again run this command to update your referral points values in your MongoDB database:-

```
npx email-armor init
```

Usage

1. Sign Up:-

To get started, set up the sign-up module data in the Front-End first and pass it to the Back-End **(you can use your preferred method to send the data):-**

```
const data = {fullName, userName, emailID, password, referralCode};  
// You can use fetch or any method you are comfortable with.  
const response = await axios.post('YOUR_URL', data);
```

Next, configure the sign-up module on the Back-End:-

```
import signup from "email-armor";
const response = await signup(fullName, userName, emailID, password, referralCode);
console.log(response);
```

After the user signs up, they will receive an OTP on their registered email. Consequently, you will receive a response similar to this:-

```
return {
  status: 201,
  message: "Account Created Successfully",
  userName: username,
};
```

Following that, in your front-end code, use cookies-next to store the userName (**which we obtained from the response above**) in the browser's cookies:

```
import { setCookie } from 'cookies-next';
const setUserNameCookies = setCookie('userName', getUserNamFromResponse);
```

After sending the OTP, redirect the user to the account verification page and follow the steps provided.

2. Sign Up Verify:-

To start, in your front-end code, use **cookies-next** to extract the userName from cookies, as well as the **OTP** entered by the user. Then, send this data to the Back-End:-

```
import { getCookie } from 'cookies-next';
const userNameCookie = getCookie('userName');
const data = {userNameCookie, OTP};
const response = await axios.post('YOUR_URL', data);
```

Set up the sign-up verify module in Back-End. Make sure to fetch userName from **cookies** as we stored it above.

```
import { signUpVerify } from "email-armor";
const response = await signUpVerify(userNameCookie, OTP);
console.log(response);
```

After the user verifies their account in the **MongoDB accounts model**, the **userVerified** section in their document will change from **false** to **true**. If they have been **referred**, they will also receive **referral points**. As a result, you will receive a response similar to this:-

```
return {
  status: 200,
  message: "Account Verified"
}
```

3. Sign In:-

To get started, set up the sign-in module data in the Front-End first and pass it to the Back-End (**you can use your preferred method to send the data**):-

```
const data = {userName, userPassword};
// You can use fetch or any method you are comfortable with.
const response = await axios.post('YOUR_URL', data);
```

Next, configure the sign-in module on the Back-End:-

```
import { signin } from "email-armor";
const response = await signin(userName, userPassword)
console.log(response);
```

After the user signs in with correct details, if the user is registered & has verified their account, they will receive an OTP on their email. You will receive this response, and you should then redirect them to the sign-in verification page:-

```
return {
  status: 200,
  message: "Sign In Successful, OTP Sent To Mail",
  userName: username,
  token: userTokenAddress,
```

```
id: savedData.id
};
```

Note:- If the user is registered but hasn't verified their account, you will receive this response, and you should redirect them to the verification/signUpVerify page:-

```
return {
  status: 401,
  message: "Please Verify Your Account",
  userName: username,
}
```

As we did above, store the userName, token, & Id in cookies that we received from the response above (*similar like this*):-

```
import { setCookie } from 'cookies-next';
const setUserNameCookies = setCookie('userName', getUserNameFromResponse);
const setToken = setCookie('token', getTokenFromResponse);
const setId = setCookie('id', getIdFromResponse);
```

4. Sign-in Verify:-

As mentioned above, the user has signed in with their details, and they are verified, then, you have redirected them to the sign-in verification page. To proceed, use the following functions in the front-end to pass the data to the Back-End:-

```
import { getCookie } from 'cookies-next';
const userNameCookie = getCookie('userName');
const userIdCookie = getCookie('userId');
const data = {userNameCookie, OTP, userId}
const response = await axios.post('YOUR_URL', data)
```

Once the data is sent to the Back-End, use this method to verify the user:-

```
import { signInVerify } from "email-armor";
const response = await signInVerify(userNameCookie, OTP, userId);
console.log(response);
```

If the user enters the correct OTP, in the **MongoDB Session Model**, the user document **OTP field** will be removed, and the document's expiry will be changed to 10 days. In return, you will receive this response:-

```
return {
  status: 202,
  message: "Account Verified"
}
```

5. Auto User Session Check:-

What if the user's session has expired, and they are still logged in, or if they attempt to manipulate cookies and perform unauthorized actions? You know that's not good, right? So, use the `sessionCheck()` function to verify if the user's session is legitimate and active. Follow these steps:-

```
import { sessionCheck } from "email-armor";
const userNameCookie = getCookie('userName');
const userTokenCookie = getCookie('userToken');
const userIdCookie = getCookie('userId');
const response = await sessionCheck(userNameCookie, userTokenCookie, userId);
// Note:- IP Will Be Automatically Fetched.
```

If the user is legitimate, you will receive this response, and their session will remain logged in:-

```
return {
  status: 202,
  message: "Session Exist"
}
```

Else, if there are no session found, then, redirect them to the login page. The response you will receive is:-

```
return {
  status: 400,
  message: "Session Don't Exist"
}
```


6. Forgot Password:-

To begin, get **userName** in the Front-End, then pass them to the Back-End, similar like this:-

```
const data = { userName }
const response = await axios.post('YOUR_URL', data)
```

Once the data is passed to the Back-End, use the **forgotPassword** function to reset/update the password in MongoDB like this:-

```
import { forgotPassword } from "email-armor";
const response = await forgotPassword(userName);
```

After this, it will first verify whether the user exists in MongoDB or not. If the user exists, you will receive this response:-

```
return {
  status: 201,
  message: "OTP Sent To Mail",
  userName: userName,
};
```

Kindly save the **userName** to cookies as we did above. After that, pass your **OTP** and **newPassword** to the Back-End via the Front-End similar like this:-

```
const userNameCookie = getCookie('userName');
const data = { userNameCookie, OTP, newPassword }
const response = await axios.post('YOUR_URL', data)
```

Once the data is received in the back-end, please perform the following actions:-

```
const response = await forgotPassword(userNameCookie, OTP, newPassword)
```

Now, firstly, we will check if the **OTP** is correct or not. If the **OTP** is correct, we will update the new password. Once the password is updated, you will receive a response

like this:-

```
return {  
  status: 200,  
  message: "Password Updated."  
}
```

To resend OTP for the **forgot password** functionality, use these values:-

```
const response = await resendOTP(userNameCookie, 'forgotPassword')
```

7. Resend OTP:-

There are **3 functions** to resend OTP to the user:-

1. [Resend OTP For New/Unverified User.](#)
2. [Resend OTP For Old/Verified User.](#)
3. [Resend OTP For Forgot Password.](#)

Function 1 (For New Users):-

Once the user is on **signup verify page**, & if he requests to resend the OTP, use this below syntax in your front-end:-

```
const userNameCookie = getCookie('userName');  
const method = 'newUserVerification'; //This Helps Module To Know That Resend OTP  
For The unverified User  
const data = { userNameCookie, method };  
const response = await axios.post('YOUR_URL', data)
```

Once the data is received in the back-end, please perform the following actions:-

```
const response = await resendOTP(data.userNameCookie, data.method)
```

Now it will find the document in the DB, & update the new OTP in the document, & will also increment the OTPCount by +1. Once the OTP is sent to the user, & updated in the DB, then, you will receive a response like this:-

```
return {
  status: 201,
  message: "OTP Resent To The User.",
};
```

If the OTPCount === OTP_LIMITS(mailpassify.json), then, it will not send OTP to the user, and you will receive a response like this:-

```
return {
  status: 403,
  message: "Max OTP Limit Reached, Please Try After 10 Minutes."
};
```

Note:- Once the OTP limits are reached, the user can try again after waiting for 5-10 minutes, as the OTP document from the database will be automatically deleted after this period.

Function 2 (For Old Users):-

Once the user is on **signin verify page**, & if he requests to resend the OTP, use this below syntax in your front-end:-

```
const userNameCookie = getCookie('userName');
const userTokenCookie = getCookie('userToken');
const userIdCookie = getCookie('userId');
const method = 'oldUserVerification'; //This Helps Module To Know That Resend OTP
For The verified User
const data = { userNameCookie, method, userTokenCookie, userIdCookie };
const response = await axios.post('YOUR_URL', data)
```

Now it will find the document in the DB, & update the new OTP in the document, & will also increment the OTPCount by +1. Once the OTP is sent to the user, & updated in the DB, then, you will receive a response like this:-

```
return {
  status: 201,
  message: "OTP Resent To The User.",
};
```

If the `OTPCount === OTP_LIMITS(mailpassify.json)`, then, it will not send OTP to the user, and you will receive a response like this:-

```
return {
  status: 403,
  message: "Max OTP Limit Reached, Please Try After 10 Minutes."
};
```

Note:- If a user reaches the maximum OTP request limit, they can still attempt to sign in again, which will generate different values.

Function 3 (For Forgot Password):-

Once the user is on **forgot password page**, & if he requests to resend the OTP, use this below syntax in your front-end:-

```
const userNameCookie = getCookie('userName');
const method = 'forgotPassword'; //This Helps Module To Know That Resend OTP For
The forgotPassword User
const data = { userNameCookie, method };
const response = await axios.post('YOUR_URL', data)
```

Once the data is received in the back-end, please perform the following actions:-

```
const response = await resendOTP(data.userNameCookie, data.method)
```

Now it will find the document in the DB, & update the new OTP in the document, & will also increment the `OTPCount` by +1. Once the OTP is sent to the user, & updated in the DB, then, you will receive a response like this:-

```
return {
  status: 201,
  message: "OTP Resent To The User.",
};
```

If the `OTPCount === OTP_LIMITS(mailpassify.json)`, then, it will not send OTP to the user, and you will receive a response like this:-

```
return {  
  status: 403,  
  message: "Max OTP Limit Reached, Please Try After 10 Minutes."  
};
```

Note:- Once the OTP limits are reached, the user can try again after waiting for 5-10 minutes, as the OTP document from the database will be automatically deleted after this period.

8. Custom Email Template:-

To create custom template, update the `email-template.html` file. Currently the template only support **plain html with in-line css**, you can checkout the pre-installed template in the file. You can use ChatGPT to convert your template to **plain html with in-line css**.

Feel free to raise an issue if you find any bugs. Thanks in advance! 😊