Flow Swiftmailer Documentation

Release 2.1.3

The Neos Team

Apr 07, 2022

Contents

This package provides helper methods to build double opt-in mechanisms in Flow applications.

This version of the documentation covering release 2.1.3 has been rendered at: Apr 07, 2022

CHAPTER 1

Overview

The package contains a helper class that can be used to

- generate tokens with attached metadata
- create validation links for these token
- send emails with validation instructions
- validate tokens
- log the validation of tokens

This can be used to build double opt-in for newsletter registrations, user sign-up or anything else that needs such a way of checking the users intention. The flow is assumed to be like this:

- create a new token associated with some identifier
- send an activation link containing this token
- check an incoming token against the stored tokens
- if a token is verified successfully, take the intended action

CHAPTER 2

Installation

The package can be installed via composer:

```
composer require flownative/flow-doubleoptin
```

CHAPTER 3

Configuration

To set up the token generation, you can override the settings in your package. To allow for different use cases in the same application, presets are used. If no preset is specified, the following *default* preset is used:

```yaml
Flownative:
  DoubleOptIn:
    presets:
      # the default preset is used if no other preset is requested
      'default':
        # the length of the generated token, in bytes, before hex encoding
        tokenLength: 32
        # how long a token should be valid after generation, in seconds
        lifetime: 86400
        # if set to true, tokens are not removed from cache automatically when the␣
↪activation link is called
        preserveToken: true
        # configuration for building an activation link for a token
        activation:
          # if uri is given as a string, it is used as given.
          # the placeholder -tokenhash- will be replaced with the token hash
          # uri: 'http://acme.com/activate/-tokenhash-'

          # if uri is a hash it is used as UriBuilder input (like in Routes.yaml)
          # the placeholder -tokenhash- will be replaced with the token hash in␣
↪the built uri
          uri:
            '@package': ~
            '@subpackage': ~
            '@controller': ~
            '@format': ~
            '@action': ~
            arguments: ~
        mail:
          # name and email address to use for activation mails
          from:
            name: ~
            address: ~
          # subject to use for activation mails
          subject: 'Double Opt-In Activation Needed'
          # the message for the activation mail must contain a plain text part. an␣
↪html part is optional
```

(continues on next page)

(continued from previous page)

```
          # the settings must point to a Fluid template file. placeholders␣
↪available in the templates when
          # being rendered: activationLink, recipientAddress, meta
          message:
            plaintext: 'resource://Flownative.DoubleOptIn/Private/Templates/
↪ActivationMail.txt'
            html: ~
```

To adjust this default preset, override as usual:

```
Flownative:
  DoubleOptIn:
    presets:
      'default':
        tokenLength: 8
        activation:
          uri: 'http://acme.com/?validate=-tokenhash-'
```

To create a custom preset, simply specify the needed differences to the *default* preset, the settings are merged with the defaults before use.

```
Flownative:
  DoubleOptIn:
    presets:
      'registration':
        mail:
          from:
            name: 'Membership department'
            address: 'noreply@acme.com'
          subject: 'Your membership registration needs activation'
          message:
            plaintext: 'resource://Acme.AcmeCom/Private/Templates/ActivationMail.
↪txt'
```

CHAPTER 4

Using double opt-in

This example assumes you have some registration form on your website and have collected an email address in `$recipientAddress`. To generate a Token and send an activation mail, two lines are needed:

```
$token = $this->doubleOptInHelper->generateToken($recipientAddress);
$this->doubleOptInHelper->sendActivationMail($recipientAddress, $token);
```

The parameter passed to `generateToken()` can be an arbitrary identifier string, even though in this example the email address of the use user is used, it can be anything that allows you to identify the subject of the double opt-in in the activation step.

When the user clicks the link, your code needs to pass the received token to the validation method and can process the result, in this example by calling `activateRegistration()`:

```
$token = $this->doubleOptInHelper->validateTokenHash($tokenHash);
if ($token === NULL) {
     // token was no valid
} else {
     // token was valid, $result contains a Token instance
     $this->activateRegistration($token->getIdentifier());
}
```

4.1 Using a custom preset

To use a custom preset, pass the name when generating a Token:

```
$token = $this->doubleOptInHelper->generateToken($recipientAddress, 'registration
↪');
```

4.2 Storing metadata with a token

When generating a Token, you can pass data in the `$meta` argument. It is stored with the token and can be retrieved when the token is validated:

```
$token = $this->doubleOptInHelper->generateToken(
     $recipientAddress,
     'registration',
     ['customerNumber' => 12345]
);

// token hash has been sent and is now coming in

$token = $this->doubleOptInHelper->validateTokenHash($tokenHash);
if ($result === FALSE) {
     // token was not valid
} else {
     // token was valid, $result contains a Token instance
     $customerNumber = $token->getMeta()['customerNumber'];
}
```

4.3 Sending activation links manually

You can of course send activation links manually or distribute the token hash in any way you need. If you just want to fetch the activation link, call `getActivationLink()`

```
$link = $this->doubleOptInHelper->getActivationLink($token);
```

To implement your own logic completely, fetching the hash from a token can be like:

```
$hash = $token->getHash();
```

CHAPTER 5

Generated log entries

To be able to prove correct use of double opt-in, having logs of token generation and validation is needed. The package logs to `DoubleOptIn.log` by default and produces the following log entries:

```
Token with hash 67...a5 generated for identifier foo@bar.com (valid until 2015-06-
↪26 20:21:35) [newsletter]
Activation link built for token with hash 67...a5
Activation mail sent to foo@bar.com for token with hash 67...a5

Validated token hash 67...a5 for identifier foo@bar.com

Validation of token hash 79...5f failed
```

In this case the identifier used for token generation was "foo@bar.com". Of course the entries are prefixed as usual with the timestamp, log level and the key "DoubleOptIn". The value in square brackets at the end of the token generation log entry is the preset name that was used.

The logging parameters can be changed by adjusting the `Flownative.DoubleOptIn.logger` settings:

```
Flownative:
  DoubleOptIn:
    logger:
      backend: Neos\Flow\Log\Backend\FileBackend
      backendOptions:
        logFileURL: '%FLOW_PATH_DATA%Logs/DoubleOptIn.log'
        createParentDirectories: TRUE
        severityThreshold: '%LOG_INFO%'
        maximumLogFileSize: 10485760
        logFilesToKeep: 10
```