Flask-EasyJWT Documentation

Release 0.2.2

Bastian Meyer

Jan 01, 2021

CONTENTS

Flask-EasyJWT provides a simple interface to creating and verifying JSON Web Tokens (JWTs) in Python. It allows you to once define the claims of the JWT, and to then create and accept tokens with these claims without having to check if all the required data is given or if the token actually is the one you expect.

Flask-EasyJWT is a simple wrapper around EasyJWT for easy usage in Flask applications. It provides configuration options via Flask's application configuration for common settings of all tokens created in a web application. For detailed information on how to use EasyJWT, see its documentation.

```python
from flask_easyjwt import FlaskEasyJWT
from flask import Flask

# Define the claims of your token.
class MySuperSimpleJWT(FlaskEasyJWT):

    def __init__(self, key):
        super().__init__(key)

        # Define a claim `name`.
        self.name = None

# Define the default configuration options for FlaskEasyJWT
# in the configuration of your Flask app.
app = Flask(__name__)
app.config.from_mapping(
    # The default key for encoding and decoding tokens.
    EASYJWT_KEY='Super secret key',

    # Tokens will be valid for 15 minutes after creation by default.
    EASYJWT_TOKEN_VALIDITY=15 * 60
)

@app.route('/token/<name>')
def get_token(name):
    """ This view returns a token with the given name as its value. """
    token_object = MySuperSimpleJWT()
    token_object.name = name
    return token_object.create()

@app.route('/verify/<token>')
def verify_token(token):
    """ This view verifies the given token and returns the contained name. """
    verified_token_object = MySuperSimpleJWT.verify(token)
    return verified_token_object.name
```

Contents

CHAPTER

ONE

FEATURES

- Integrates EasyJWT into Flask for easy configuration of default options for creating and verifying JWTs.
- Define the claims of your token once as a class, then use this class to easily create and verify multiple tokens.
- No worries about typos in dictionary keys: the definition of your claim set as a class enables IDEs to find those typos for you.
- Multiple tokens may have the same claims, but different intentions. Flask-EasyJWT will take care of this for you: you can define a token for account validation and one for account deletion, both with the account ID as a claim, and you don't need to worry about accidentally deleting a newly created account instead of validating it, just because someone mixed up the tokens.
- All registered JWT claims are supported: `aud`, `exp`, `iat`, `iss`, `jti`, `nbf`, and `sub`.

For a full list of features, see the features of EasyJWT.

CHAPTER

TWO

SYSTEM REQUIREMENTS & INSTALLATION

Flask-EasyJWT requires Python 3.6 or newer.

Flask-EasyJWT is available on PyPI. You can install it using your favorite package manager.

- PIP:

```
python -m pip install flask_easyjwt
```

- Pipenv:

```
pipenv install flask_easyjwt
```

CHAPTER

THREE

USAGE

Flask-EasyJWT is used exactly as EasyJWT. Therefore, this section only describes the specific features of Flask-EasyJWT and the basic usage. For detailed explanations on how to use EasyJWT (for example, optional claims, registered claims such as `aud`, `iat`, and `sub`, or verifying third-party tokens), see its documentation.

3.1 Application Setup

You do not need to initialize Flask-EasyJWT with your Flask application. All you have to do (although even this is, strictly speaking, not required), is to specify some default settings for all of your tokens in the configuration of your Flask application. These settings are:

Configuration Key	Description
`EASYJWT_KEY`	The key that will be used for encoding and decoding all tokens. If `EASYJWT_KEY` is not specified, Flask-EasyJWT will fall back to Flask's `SECRET_KEY` configuration value.
`EASYJWT_TOKEN_VALIDITY`	The validity of each token after its creation. This value can be given as a string (that is parsable to an integer), an integer, or a `timedelta` object. The former two are interpreted in seconds.

You can specify these configuration values as any other configuration values in your Flask application, for example, using a mapping in your code:

```
from datetime import timedelta
from flask import Flask

app = Flask(__name__)
app.config.update(
    EASYJWT_KEY='Super secret key',
    EASYJWT_TOKEN_VALIDITY=timedelta(minutes=7)
)
```

In this example, all tokens will (by default) be encoded using the (not so secure) string `Super secret key` and will be valid for seven minutes after they have been created (i.e., after the `create()` method has been called on the token object).

Of course, any other way of specifying the configuration values will work as well (see Flask's documentation).

3.2 Token Specification & Usage

Tokens are specified and used exactly as with EasyJWT:

```
from flask_easyjwt import FlaskEasyJWT

# Define the claims of your token.
class MySuperSimpleJWT(FlaskEasyJWT):

    def __init__(self, key):
        super().__init__(key)

        # Define a claim `name`.
        self.name = None

# Assuming we are within a Flask app context.

# Create a token with some values.
token_object = MySuperSimpleJWT()
token_object.name = 'Zaphod Beeblebrox'
token = token_object.create()

# Verify the created token.
verified_token_object = MySuperSimpleJWT.verify(token)
assert verified_token_object.name == 'Zaphod Beeblebrox'
```

The only difference is that you do not have to pass the key for encoding or decoding the token to the constructor and `verify()` method, respectively (you still can do so if you do not want to use the default key defined in your application's configuration).

Additionally, if the configuration value `EASYJWT_TOKEN_VALIDITY` is set, the token will be valid for the amount specified in this configuration value after it has been created with `create()`. If this configuration value is not set tokens will not expire. If you explicitly set the expiration date on a token object this value will always take precedence (if it is not `None`):

```
import datetime

from flask_easyjwt import FlaskEasyJWT
from flask import Flask

# Define the claims of your token.
class MySuperSimpleJWT(FlaskEasyJWT):

    def __init__(self, key):
        super().__init__(key)

        # Define a claim `name`.
        self.name = None

# Define the default configuration options for FlaskEasyJWT
# in the configuration of your Flask app.
app = Flask(__name__)
app.config.from_mapping(
    EASYJWT_KEY='Super secret key',
    EASYJWT_TOKEN_VALIDITY=datetime.timedelta(minutes=7)
)

# Assuming we are within a Flask app context.

token_object = MySuperSimpleJWT()
token_object.name = 'Zaphod Beeblebrox'
```

(continues on next page)

(continued from previous page)

```
# This token will expire in 15 minutes, even though the default token validity is␣
↪set to 7 minutes.
token_object.expiration_date = datetime.datetime.utcnow() + datetime.
↪timedelta(minutes=15)
```

Initializing token objects and creating and verifying tokens must be executed within a Flask application context if you want to use the configuration values from the application's configuration.

CHAPTER

FOUR

ACKNOWLEDGEMENTS

Flask-EasyJWT is just an easy-to-use abstraction layer around José Padilla's PyJWT library that does the actual work of creating and verifying the tokens according to the JWT specification. Without his work, Flask-EasyJWT would not be possible.

CHAPTER

FIVE

TABLE OF CONTENTS

5.1 API

Flask-EasyJWT provides a simple interface to creating and verifying JSON Web Tokens (JWTs) in Python. It allows you to once define the claims of the JWT, and to then create and accept tokens with these claims without having to check if all the required data is given or if the token actually is the one you expect.

Flask-EasyJWT is a simple wrapper around EasyJWT for easy usage in Flask applications. It provides configuration options via Flask's application configuration for common settings of all tokens created in a web application.

See the included README file or the documentation for details on how to use Flask-EasyJWT.

Contents

5.1.1 Classes

This section lists all classes defined by Flask-EasyJWT.

class FlaskEasyJWT(*key: Optional[str] = None*)

Bases: `easyjwt.easyjwt.EasyJWT`

The base class for representing JSON Web Tokens (JWT).

To use a JWT, you have to create a subclass inheriting from *`FlaskEasyJWT`*. All public instance variables of this class (that is, all instance variables not starting with an underscore) will make up the claim set of your token (there will be a few meta claims in the token as well that *`FlaskEasyJWT`* needs to verify the token). For details, see the documentation of EasyJWT.

`FlaskEasyJWT` simplifies the usage of *EasyJWT* in Flask applications by allowing to specify a few common settings in the application's configuration:

- The key used for encoding and decoding a token can be specified in the configuration key `EASYJWT_KEY`.
- The validity of a token can be specified in the configuration key `EASYJWT_TOKEN_VALIDITY`. The expiration date will be set at the time of creation of a token to the current time plus the specified duration (in seconds).

Parameters **key** – If set, the given string will be used to encrypt tokens when they are created. If not given, the key defined in the application's configuration will be used. Defaults to *None*.

Raises ***EasyJWTError*** – If no key is given and there is no key defined in the application's configuration.

create(*issued_at: Optional[datetime.datetime] = None*) → str

Create the actual token from the `EasyJWT` object. Empty optional claims will not be included in the token. Empty non-optional claims will cause a *MissingRequiredClaimsError*.

Parameters **issued_at** – The date and time at which this token was issued. If not given, the current date and time will be used. Must be given in UTC. Defaults to *None*.

Returns The token represented by the current state of the object.

Raises

- ***IncompatibleKeyError*** – If the given key is incompatible with the algorithm used for encoding the token.
- ***MissingRequiredClaimsError*** – If instance variables that map to non-optional claims in the claim set are empty.

classmethod verify(*token: str*, *key: Optional[str] = None*, *issuer: Optional[str] = None*, *audience: Optional[Union[Iterable[str], str]] = None*) → FlaskEasyJWT-Class

Verify the given JSON Web Token.

Parameters

- **token** – The JWT to verify.
- **key** – The key used for decoding the token. This key must be the same with which the token has been created. If left empty, the key set in the application's configuration will be used.
- **issuer** – The issuer of the token to verify.
- **audience** – The audience for which the token is intended.

Returns The object representing the token. The claim values are set on the corresponding instance variables.

Raises

- ***EasyJWTError*** – If no key is given and there is no key defined in the application's configuration.
- ***ExpiredTokenError*** – If the claim set contains an expiration date claim `exp` that has passed.
- ***ImmatureTokenError*** – If the claim set contains a not-before date claim `nbf` that has not yet been reached.
- ***IncompatibleKeyError*** – If the given key is incompatible with the algorithm used for decoding the token.
- ***InvalidAudienceError*** – If the given audience is not specified in the token's audience claim, or no audience is given when verifying a token with an audience claim, or the given audience is not a string, an iterable, or *None*.
- ***InvalidClaimSetError*** – If the claim set does not contain exactly the expected (non-optional) claims.
- ***InvalidClassError*** – If the claim set is not verified with the class with which the token has been created.

- ***InvalidIssuedAtError*** – If the claim set contains an issued-at date `iat` that is not an integer.
- ***InvalidIssuerError*** – If the token has been issued by a different issuer than given.
- ***InvalidSignatureError*** – If the token's signature does not validate the token's contents.
- ***UnspecifiedClassError*** – If the claim set does not contain the class with which the token has been created.
- ***UnsupportedAlgorithmError*** – If the algorithm used for encoding the token is not supported.
- ***VerificationError*** – If a general error occurred during decoding.

static get_validity() → Optional[datetime.timedelta]

Get the token's validity from the configuration of the current Flask application.

The token's validity is read from the configuration key *EASYJWT_TOKEN_VALIDITY*. The value can either be a string castable to an integer, an integer (both interpreted in seconds), or a *datetime.timedelta* object.

This method must be executed within the application context.

Returns *None* if no token validity is defined in the application's configuration or if the value has a wrong type.

5.1.2 Enumerations

This section lists all enumerations defined by Flask-EasyJWT.

class Algorithm(*value*)

Bases: `enum.Enum`

The supported algorithms for cryptographically signing the tokens.

HS256 = 'HS256'

HMAC using the SHA-256 hash algorithm.

HS384 = 'HS384'

HMAC using the SHA-384 hash algorithm.

HS512 = 'HS512'

HMAC using the SHA-512 hash algorithm.

5.1.3 Errors

This section lists all error classes defined by Flask-EasyJWT.

exception EasyJWTError(*message: str*)

Bases: `Exception`

A base class for all errors raised by `EasyJWT`.

Parameters **message** – A user-readable description of this error.

Creation Errors

This section lists all error classed defined by Flask-EasyJWT that may be raised during the creation of a token. Note that some error classes may also listed below *Verification Errors*.

exception CreationError(*message: str*)
Bases: easyjwt.errors.EasyJWTError

A base class for all errors raised during the creation of a token.

Parameters **message** – A user-readable description of this error.

exception IncompatibleKeyError(*message: str*)
Bases: easyjwt.errors.EasyJWTError

Raised if the creation or verification of a token fails because the given key is incompatible with the used algorithm.

Parameters **message** – A user-readable description of this error.

exception MissingRequiredClaimsError(*missing_claims: Iterable[str]*)
Bases: easyjwt.errors.CreationError

Raised if the creation of a token fails because non-optional claims are empty.

Parameters **missing_claims** – The names of claims that are required but empty.

missing_claims: Set[str]
A set of the names of claims that are expected but missing in the claim set.

Verification Errors

This section lists all error classed defined by Flask-EasyJWT that may be raised during the verification of a token. Note that some error classes may also listed below *Creation Errors*.

exception VerificationError(*message: str*)
Bases: easyjwt.errors.EasyJWTError

A base class for all errors raised during the verification of a token.

Parameters **message** – A user-readable description of this error.

exception ExpiredTokenError
Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the included expiration date has passed.

Parameters **message** – A user-readable description of this error.

exception ImmatureTokenError
Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the included not-before date has not yet been reached.

Parameters **message** – A user-readable description of this error.

exception IncompatibleKeyError(*message: str*)
Bases: easyjwt.errors.EasyJWTError

Raised if the creation or verification of a token fails because the given key is incompatible with the used algorithm.

Parameters **message** – A user-readable description of this error.

exception InvalidAudienceError
Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the audience with which the application tries to verify a token is not included in the token's audience claim, or the audience given in the verify method is not a string, an iterable, or None.

Parameters **message** – A user-readable description of this error.

exception InvalidClaimSetError(*missing_claims: Optional[Iterable[str]] = None*, *unexpected_claims: Optional[Iterable[str]] = None*)

Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the claim set is invalid due to missing or unexpected claims.

Parameters

- **missing_claims** – The names of claims that are expected but missing in the claim set.
- **unexpected_claims** – The names of claims that are given in the claim set but are not specified in the class.

missing_claims: Set[str]

A set of the names of claims that are expected but missing in the claim set.

If no missing claims are given, this will be an empty set.

unexpected_claims: Set[str]

A set of the names of claims that are given in the claim set but are not specified in the class.

If no unexpected claims are given, this will be an empty set.

exception InvalidClassError(*expected_class: str*, *actual_class: str*)

Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the EasyJWT class with which it has been created is not the one with which it is being verified.

Parameters

- **expected_class** – The class with which the token is being verified.
- **actual_class** – The class with which the token has been created.

actual_class: str

The name of the class with which the token has been created.

expected_class: str

The name of the class with which the token has been verified.

exception InvalidIssuedAtError

Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the issued-at date specified in a token is not an integer.

Parameters **message** – A user-readable description of this error.

exception InvalidIssuerError

Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the given issuer is not the issuer of the token.

Parameters **message** – A user-readable description of this error.

exception InvalidSignatureError

Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the token's signature does not validate the token's content.

Parameters **message** – A user-readable description of this error.

exception UnspecifiedClassError

Bases: easyjwt.errors.VerificationError

Raised if the verification of a token fails because the `EasyJWT` class with which it has been created is not specified in the claim set.

Parameters **message** – A user-readable description of this error.

exception UnsupportedAlgorithmError
Bases: `easyjwt.errors.VerificationError`

Raised if the verification of a token fails because the algorithm used for encoding the token is not supported.

Parameters **message** – A user-readable description of this error.

5.1.4 Types

This section lists the types defined by Flask-EasyJWT.

FlaskEasyJWTClass
The type of the *FlaskEasyJWT* class, allowing subclasses.

alias of TypeVar('FlaskEasyJWTClass', covariant=True)

5.2 Changelog

This project follows semantic versioning.

Possible log types:

- `[added]` for new features.
- `[changed]` for changes in existing functionality.
- `[deprecated]` for once-stable features removed in upcoming releases.
- `[removed]` for deprecated features removed in this release.
- `[fixed]` for any bug fixes.
- `[security]` to invite users to upgrade in case of vulnerabilities.

5.2.1 0.2.2 (January 1, 2021)

- `[fixed]` Dependencies in `setup.py`.

5.2.2 0.2.1 (January 1, 2021)

- `[fixed]` Updated dependency of EasyJWT.

5.2.3 0.2.0 (September 30, 2019)

- `[added]` Method `get_validity()` to return the validity of a token.

5.2.4 0.1.0 (September 29, 2019)

- Initial release of Flask-EasyJWT
- `[added]` Support defining the token key in the Flask application configuration (`EASYJWT_KEY`).
- `[added]` Support defining the token validity in the Flask application configuration (`EASYJWT_TOKEN_VALIDITY`).

5.3 License

Flask-EasyJWT is licensed under the MIT License.

The full license can be found below (*License Text*).

5.3.1 Author

Flask-EasyJWT is developed by Bastian Meyer <bastian@bastianmeyer.eu>.

5.3.2 Remarks

The *license text* applies to the entire source code shipped as part of Flask-EasyJWT, including, but not limited to, all examples, tests, and this documentation. Exceptions are stated where necessary.

5.3.3 License Text

MIT License

Copyright (c) 2019 Bastian Meyer

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Developed by Bastian Meyer. Flask-EasyJWT is licensed under the MIT License. See the license page for details.

PYTHON MODULE INDEX

f

INDEX