Falcon Auth2

Federico Caselli

Dec 08, 2021

CONTENTS:

Falcon authentication middleware that supports multiple authentication schemes.

CHAPTER

ONE

USER GUIDE

1.1 Install

```
$ pip install falcon-auth2[jwt]
```

The above will install *falcon-auth2* and also the dependencies to use the `JWT` authentication backend.
If you plan to use async falcon with ASGI run:

```
$ pip install falcon-auth2[jwt, async]
```

1.2 Usage

This package provides a falcon middleware to authenticate incoming requests using the selected authentication backend. The middleware allows excluding some routes or method from authentication. After a successful authentication the middleware adds the user identified by the request to the `request.context`. When using falcon v3+, the middleware also supports async execution.

See below *WSGI example* and *ASGI example* for complete examples.

```
import falcon
from falcon_auth2 import AuthMiddleware
from falcon_auth2.backends import BasicAuthBackend

def user_loader(attributes, user, password):
    if authenticate(user, password):
        return {"username": user}
    return None

auth_backend = BasicAuthBackend(user_loader)
auth_middleware = AuthMiddleware(auth_backend)
# use falcon.API in falcon 2
app = falcon.App(middleware=[auth_middleware])

class HelloResource:
    def on_get(self, req, resp):
        # req.context.auth is of the form:
```

(continues on next page)

(continued from previous page)

```
        #
        #   {
        #       'backend': <instance of the backend that performed the authentication>,
        #       'user': <user object retrieved from the user_loader callable>,
        #       '<backend specific item>': <some extra data that may be added by the␣
↪backend>,
        #       ...
        #   }
        user = req.context.auth["user"]
        resp.media = {"message": f"Hello {user['username']}"}

app.add_route('/hello', HelloResource())
```

1.2.1 Overriding authentication for a resource

The middleware allows each resource to customize the backend used for authentication or the excluded methods. A resource can also specify that does not need authentication.

```
from falcon_auth2 import HeaderGetter
from falcon_auth2.backends import GenericAuthBackend

class OtherResource:
    auth = {
        "backend": GenericAuthBackend(
            user_loader=lambda attr, user_header: user_header, getter=HeaderGetter("User
↪")
        ),
        "exempt_methods": ["GET"],
    }

    def on_get(self, req, resp):
        resp.media = {"type": "No authentication for GET"}

    def on_post(self, req, resp):
        resp.media = {"info": f"User header {req.context.auth['user']}"}

app.add_route("/other", OtherResource())

class NoAuthResource:
    auth = {"auth_disabled": True}

    def on_get(self, req, resp):
        resp.media = "No auth in this resource"

    def on_post(self, req, resp):
        resp.media = "No auth in this resource"

app.add_route("/no-auth", NoAuthResource())
```

1.3 Examples

1.3.1 WSGI example

```
# flake8: noqa
import random

import falcon

from falcon_auth2 import AuthMiddleware
from falcon_auth2 import HeaderGetter
from falcon_auth2.backends import BasicAuthBackend
from falcon_auth2.backends import GenericAuthBackend
from falcon_auth2.backends import JWTAuthBackend
from falcon_auth2.backends import MultiAuthBackend

# To run this application with waitress (or any other wsgi server)
# waitress-serve --port 8080 readme_example:app
# You can then use httpie to interact with it. Example
# http :8080/hello -a foo:bar
# http :8080/hello 'Authorization:Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJzdWIiOiJiYXIiLCJpc3MiOiJhbiBpc3N1ZXIiLCJhdWQiOiJhbiBhdWQiLCJpYXQiOjE2MDk0NTkyMDAsIm5iZiI6MTYwOTQ1O
↪FDpE_-jL-reHrheIYVGbwdVf8g1HWFAoGJet_6zC2Tk'
# http :8080/no-auth
# http POST :8080/generic User:foo


def authenticate(user, password):
    # Check if the user exists and the password match.
    # This is just for the example
    return random.choice((True, False))


def basic_user_loader(attributes, user, password):
    if authenticate(user, password):
        return {"username": user, "kind": "basic"}
    return None


def jwt_user_loader(attributes, payload):
    # Perform additional authentication using the payload.
    # This is just an example
    if "sub" in payload:
        return {"username": payload["sub"], "kind": "jwt"}
    return None


# NOTE: this is just an example. A key should be propertly generated, like using:
# key=secrets.token_bytes(256)
key = "not-a-secret-key"
basic_backend = BasicAuthBackend(basic_user_loader)
jwt_backend = JWTAuthBackend(jwt_user_loader, key)
```

(continues on next page)

(continued from previous page)

```
auth_backend = MultiAuthBackend((basic_backend, jwt_backend))
auth_middleware = AuthMiddleware(auth_backend)
# use falcon.API in falcon 2
app = falcon.App(middleware=[auth_middleware])


class HelloResource:
    def on_get(self, req, resp):
        # req.context.auth is of the form:
        #
        #   {
        #       'backend': <instance of the backend that performed the authentication>,
        #       'user': <user object retrieved from the user_loader callable>,
        #       '<backend specific item>': <some extra data that may be added by the␣
↪backend>,
        #       ...
        #   }
        user = req.context.auth["user"]
        resp.media = {"message": f"Hello {user['username']} from {user['kind']}"}


app.add_route("/hello", HelloResource())


def user_header_loader(attr, user_header):
    # authenticate the user with the user_header
    return user_header


class GenericResource:
    auth = {
        "backend": GenericAuthBackend(user_header_loader, getter=HeaderGetter("User")),
        "exempt_methods": ["GET"],
    }

    def on_get(self, req, resp):
        resp.media = {"type": "No authentication for GET"}

    def on_post(self, req, resp):
        resp.media = {"info": f"User header {req.context.auth['user']}"}


app.add_route("/generic", GenericResource())


class NoAuthResource:
    auth = {"auth_disabled": True}

    def on_get(self, req, resp):
        resp.text = "No auth in this resource"

    def on_post(self, req, resp):
```

(continues on next page)

(continued from previous page)

```
        resp.text = "No auth in this resource"


app.add_route("/no-auth", NoAuthResource())


def make_token():
    # the token above was generated by calling this example function
    from authlib.jose import jwt

    payload = {
        "sub": "bar",
        "iss": "an issuer",
        "aud": "an aud",
        "iat": 1609459200,
        "nbf": 1609459200,
        "exp": 1924992000,
    }
    print(jwt.encode({"alg": "HS256"}, payload, key))
```

1.3.2 ASGI example

```
# flake8: noqa
import random

import falcon.asgi

from falcon_auth2 import AuthMiddleware
from falcon_auth2 import HeaderGetter
from falcon_auth2.backends import BasicAuthBackend
from falcon_auth2.backends import GenericAuthBackend
from falcon_auth2.backends import JWTAuthBackend
from falcon_auth2.backends import MultiAuthBackend

# To run this application with uvicorn (or any other asgi server)
# uvicorn --port 8080 readme_example_async:app
# You can then use httpie to interact with it. Example
# http :8080/hello -a foo:bar
# http :8080/hello 'Authorization:Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJzdWIiOiJiYXIiLCJpc3MiOiJhbiBpc3N1ZXIiLCJhdWQiOiJhbiBhdWQiLCJpYXQiOjE2MDk0NTkyMDAsIm5iZiI6MTYwOTQ1O
↪FDpE_-jL-reHrheIYVGbwdVf8g1HWFAoGJet_6zC2Tk'
# http :8080/no-auth
# http POST :8080/generic User:foo


async def authenticate(user, password):
    # Check if the user exists and the password match.
    # This is just for the example
    return random.choice((True, False))
```

(continues on next page)

(continued from previous page)

```
async def basic_user_loader(attributes, user, password):
    if await authenticate(user, password):
        return {"username": user, "kind": "basic"}
    return None


async def jwt_user_loader(attributes, payload):
    # Perform additional authentication using the payload.
    # This is just an example
    if "sub" in payload:
        return {"username": payload["sub"], "kind": "jwt"}
    return None


# NOTE: this is just an example. A key should be properly generated, like using:
# key=secrets.token_bytes(256)
key = "not-a-secret-key"
basic_backend = BasicAuthBackend(basic_user_loader)
jwt_backend = JWTAuthBackend(jwt_user_loader, key)
auth_backend = MultiAuthBackend((basic_backend, jwt_backend))
auth_middleware = AuthMiddleware(auth_backend)
app = falcon.asgi.App(middleware=[auth_middleware])


class HelloResource:
    async def on_get(self, req, resp):
        # req.context.auth is of the form:
        #
        #   {
        #       'backend': <instance of the backend that performed the authentication>,
        #       'user': <user object retrieved from the user_loader callable>,
        #       '<backend specific item>': <some extra data that may be added by the␣
↪backend>,
        #       ...
        #   }
        user = req.context.auth["user"]
        resp.media = {"message": f"Hello {user['username']} from {user['kind']}"}


app.add_route("/hello", HelloResource())


async def user_header_loader(attr, user_header):
    # authenticate the user with the user_header
    return user_header


class GenericResource:
    auth = {
        "backend": GenericAuthBackend(user_header_loader, getter=HeaderGetter("User")),
        "exempt_methods": ["GET"],
```

(continues on next page)

(continued from previous page)

```
    }

    async def on_get(self, req, resp):
        resp.media = {"type": "No authentication for GET"}

    async def on_post(self, req, resp):
        resp.media = {"info": f"User header {req.context.auth['user']}"}


app.add_route("/generic", GenericResource())


class NoAuthResource:
    auth = {"auth_disabled": True}

    async def on_get(self, req, resp):
        resp.text = "No auth in this resource"

    async def on_post(self, req, resp):
        resp.text = "No auth in this resource"


app.add_route("/no-auth", NoAuthResource())


def make_token():
    # the token above was generated by calling this example function
    from authlib.jose import jwt

    payload = {
        "sub": "bar",
        "iss": "an issuer",
        "aud": "an aud",
        "iat": 1609459200,
        "nbf": 1609459200,
        "exp": 1924992000,
    }
    print(jwt.encode({"alg": "HS256"}, payload, key))
```

CHAPTER

TWO

API REFERENCE

2.1 Middleware

class falcon_auth2.**AuthMiddleware**(*backend:* falcon_auth2.backends.base.AuthBackend, *, *exempt_templates: Iterable[str] = ()*, *exempt_methods: Iterable[str] = ('OPTIONS',)*, *context_attr: str = 'auth'*)

Falcon middleware that can be used to authenticate a request.

The authentication backend returns an authenticated user which is then set by default in `request.context.auth["user"]`. In case of errors `falcon.HTTPUnauthorized` is raised. In addition to the `"user"`, the authenticating backend is returned in the `"backend"` key. A backend may also store additional information in this dict.

This middleware supports a global authentication configuration using provided *AuthBackend*, as well as per resource configuration. To override the authentication configuration a resource can specify an optional `auth` attribute the override properties. The `auth` attribute is a dict that can specify the keys:

- `auth_disabled` boolean. `True` disables the authentication on the resource.
- `exempt_methods` iterable that overrides the global `exempt_methods` for the resource.
- `backend` backend instace that overrides the globally configured backend used to handle the authentication of the request.

Parameters **backend** (AuthBackend) – The default auth backend to be used to authenticate requests. A resource can override this value by providing a `backend` key in its `auth` attribute

Keyword Arguments

- **exempt_templates** (*Iterable[str], optional*) – A list of paths templates to be excluded from the authentication. This value cannot be overridden by a resource. Defaults to `()`.
- **exempt_methods** (*Iterable[str], optional*) – A list of http methods to be excluded from the authentication. A resource can override this value by providing a `exempt_methods` key in its `auth` attribute. Defaults to `("OPTIONS",)`.
- **context_attr** (*str, optional*) – The attribute of the `req.context` object that will store the authentication information after a successful precessing. Defaults to `"auth"`.

process_resource(*req: falcon.request.Request*, *resp: falcon.response.Response*, *resource: Any*, *params: dict*)

Called by falcon when processing a resource.

It will obtain the configuration to use on the resource and, if required, call the provided backend to authenticate the request.

async process_resource_async(*req: falcon.request.Request*, *resp: falcon.response.Response*, *resource: Any*, *params: dict*)

Called by async falcon when processing a resource.

It will obtain the configuration to use on the resource and, if required, call the provided backend to authenticate the request.

2.2 Backends implementations

2.2.1 Base classes

AuthBackend

class falcon_auth2.backends.**AuthBackend**

Base class that defines the signature of the *authenticate()* method.

Backend must subclass of this class to be used by the *AuthMiddleware* middleware.

abstract authenticate(*attributes:* falcon_auth2.utils.classes.RequestAttributes) → dict

Authenticates the request and returns the authenticated user.

If a request cannot be authenticated a backed should raise:

- *AuthenticationFailure* to indicate that the request can be handled by this backend, but the authentication fails.
- *BackendNotApplicable* if the provided request cannot be handled by this backend. This is usually raised by the *Getter* used by the backend to process the request.
- *UserNotFound* when no user could be loaded with the provided credentials.

Parameters **attributes** (RequestAttributes) – The current request attributes. It's a named tuple which contains the falcon request and response objects, the activated resource and the parameters matched in the url.

Returns *dict* – A dictionary with a required "user" key containing the authenticated user. This dictionary may optionally contain additional keys specific to this backend. If the "backend" key is specified, the middleware will not override it.

BaseAuthBackend

class falcon_auth2.backends.**BaseAuthBackend**(*user_loader: Callable*, *, *challenges: Optional[Iterable[str]] = None*)

Utility class that handles calling a provided callable to load an user from the authentication information of the request in the *load_user()* method.

Parameters **user_loader** (*Callable*) – A callable object that is called with the *RequestAttributes* object as well as any relevant data extracted from the request by the backend. The arguments passed to user_loader will vary depending on the *AuthBackend*. It should return the user identified by the request, or None if no user could be not found. When using falcon in async mode (asgi), this function may also be async.

Note: An error will be raised if an async function is used when using falcon in sync mode (wsgi).

Note: Exceptions raised in this callable are not handled directly, and are surfaced to falcon.

Keyword Arguments **challenges** (*Optional[Iterable[str]], optional*) – One or more authentication challenges to use as the value of the `WWW-Authenticate` header in case of errors. Defaults to `None`.

load_user(*attributes:* falcon_auth2.utils.classes.RequestAttributes, **args*, ***kwargs*) → Any

Invokes the provided `user_loader` callable to allow the app to retrieve the user record. If no such record is found, raises a *UserNotFound* exception.

Parameters

- **attributes** (RequestAttributes) – The request attributes.
- ***args** – Positional arguments to pass to the `user_loader` callable.
- ****kwargs** – Keyword arguments to pass to the `user_loader` callable.

Returns *Any* – The loaded user object returned by `user_loader`.

2.2.2 Authentication Backends

BasicAuthBackend

class falcon_auth2.backends.**BasicAuthBackend**(*user_loader: Callable*, *, *auth_header_type: str = 'Basic'*, *getter: Optional[*falcon_auth2.getter.Getter*] = None*)

Implements the 'Basic' HTTP Authentication Scheme.

Clients should authenticate by passing the credential in the format `username:password` encoded in `base64` in the `Authorization` HTTP header, prepending it with the type specified in the setting `auth_header_type`. For example, the user `"Aladdin"` would provide his password, `"open sesame"`, with the header:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Parameters **user_loader** (*Callable*) – A callable object that is called with the *RequestAttributes* object and the username and password credentials extracted from the request using the provided `getter`. It should return the user identified by the credentials, or `None` if no user could be not found. When using falcon in async mode (asgi), this function may also be async.

Note: An error will be raised if an async function is used when using falcon in sync mode (wsgi).

Note: Exceptions raised in this callable are not handled directly, and are surfaced to falcon.

Keyword Arguments

- **auth_header_type** (*string, optional*) – The type of authentication required in the `Authorization` header. This value is added to the `challenges` in case of errors. Defaults to `"Basic"`.

Note: When passing a custom `getter` this value is only used to generate the `challenges`, since the provided getter will be used to obtain the credentials to authenticate.

- **getter** (*Optional[Getter]*) – Getter used to extract the authentication information from the request. When using a custom getter, the returned value must be a `base64` encoded string with the credentials in the format `username:password`. Defaults to *AuthHeaderGetter* initialized with the provided `auth_header_type`.

authenticate(*attributes:* falcon_auth2.utils.classes.RequestAttributes) → dict
Authenticates the request and returns the authenticated user.

JWTAuthBackend

class falcon_auth2.backends.**JWTAuthBackend**(*user_loader: Callable*, *key: Union[str, bytes, dict, Callable[[dict, dict], Union[str, bytes]]]*, ***, *auth_header_type: str = 'Bearer'*, *getter: Optional[falcon_auth2.getter.Getter] = None*, *algorithms: Optional[Union[str, List[str]]] = 'HS256'*, *claims_options: Optional[dict] = None*, *leeway: int = 0*)

Implements the JSON Web Token (JWT) standard.

Clients should authenticate by passing the token key in the *Authorization* HTTP header, prepending it with the type specified in the setting `auth_header_type`.

This backend uses the Authlib library to handle the validation of the tokens. See also its JSON Web Token (JWT) documentation for additional details on the authentication library features.

Parameters

- **user_loader** (*Callable*) – A callable object that is called with the *RequestAttributes* object and the token payload obtained after a successful validation. It should return the user identified by the token, or `None` if no user could be not found. The token is extracted from the request using the provided `getter`. When using falcon in async mode (asgi), this function may also be async.

  Note: An error will be raised if an async function is used when using falcon in sync mode (wsgi).

  Note: Exceptions raised in this callable are not handled directly, and are surfaced to falcon.

- **key** (*str, bytes, dict, Callable*) – The key to use to decode the tokens. This parameter is passed to the `JsonWebToken.decode()` method and is used to verify the signature of the token. A key can be passed as string or bytes. Dynamic keys are also supported by passing a "JWK set" dict or a callable that is called with the token header and payload and returns the key to use to validate the current token signature. See Use dynamic keys for additional details on the supported values.

Keyword Arguments

- **auth_header_type** (*string, optional*) – The type of authentication required in the `Authorization` header. This value is added to the `challenges` in case of errors. Defaults to `"Bearer"`.

Note: When passing a custom `getter` this value is only used to generate the `challenges`, since the provided getter will be used to obtain the credentials to authenticate.

- **getter** (*Optional[Getter]*) – Getter used to extract the authentication token from the request. When using a custom getter, the returned value must be a valid jwt token in string form (ie not yet parsed). Defaults to *AuthHeaderGetter* initialized with the provided `auth_header_type`.
- **algorithms** (*str, List[str]*) – The signing algorithm(s) that should be supported. Using a list multiple values may be provided. Defaults to `"HS256"`. Allowed values are listed at RFC7518: JSON Web Algorithms.
- **claims_options** (*dict*) – The claims to validate in the token. By default the value returned by *JWTAuthBackend.default_claims()* is used.
- **leeway** (*int*) – Leeway in seconds to pass to the `JWTClaims.validate()` call to account for clock skew. Defaults to 0.

authenticate(*attributes:* falcon_auth2.utils.classes.RequestAttributes) → dict
Authenticates the request and returns the authenticated user.

default_claims()
Returns the default claims to verify in the tokens.

The default claims check that the 'iss', 'sub', 'aud', 'exp', 'nbf', 'iat' are present in the token.

Subclasses can choose to override this method. The claims may also be passed using the `claims_options` parameter when instanciating this class.

See JWT Payload Claims Validation for additional details on the `claims_options` format.

GenericAuthBackend

class falcon_auth2.backends.**GenericAuthBackend**(*user_loader: Callable*, *getter:* falcon_auth2.getter.Getter, ***, *payload_key: Optional[str] = None*, *challenges: Optional[Iterable[str]] = None*)

Generic authentication backend that delegates the verification of the authentication information retried from the request by the provided `getter` to the `user_loader` callable.

This backend can be used to quickly implement custom authentication schemes or as an adapter to other authentication libraries.

Depending on the `getter` provided, this backend can be used to authenticate the an user using a session cookie or using a parameter as token.

Parameters

- **user_loader** (*Callable*) – A callable object that is called with the *RequestAttributes* object and the information extracted from the request using the provided `getter`. It should return the user identified by the request, or `None` if no user could be not found. When using falcon in async mode (asgi), this function may also be async.

Note: An error will be raised if an async function is used when using falcon in sync mode (wsgi).

Note: Exceptions raised in this callable are not handled directly, and are surfaced to falcon.

- **getter** (Getter) – Getter used to extract the authentication information from the request. The returned value is passed to the `user_loader` callable.

Keyword Arguments

- **payload_key** (*Optional[str], optional*) – It defines a key in the dict returned by the `authentication()` method that will contain data obtained from the request by the `getter`. Use `None` to disable this functionality. Defaults to `None`.
- **challenges** (*Optional[Iterable[str]], optional*) – One or more authentication challenges to use as the value of the `WWW-Authenticate` header in case of errors. Defaults to `None`.

authenticate(*attributes:* falcon_auth2.utils.classes.RequestAttributes) → dict
Authenticates the request and returns the authenticated user.

NoAuthBackend

class falcon_auth2.backends.**NoAuthBackend**(*user_loader: Callable*, *, *challenges: Optional[Iterable[str]] = None*)

No authentication backend.

This backend does not perform any authentication check. It can be used with the *MultiAuthBackend* in order to provide a fallback for an unauthenticated user or to implement a complitely custom authentication workflow.

Parameters **user_loader** (*Callable*) – A callable object that is called with the *RequestAttributes* object and returns a default unauthenticated user (alternatively the user identified by a custom authentication workflow) or `None` if no user could be not found. When using falcon in async mode (asgi), this function may also be async.

Note: An error will be raised if an async function is used when using falcon in sync mode (wsgi).

Note: Exceptions raised in this callable are not handled directly, and are surfaced to falcon.

Keyword Arguments **challenges** (*Optional[Iterable[str]], optional*) – One or more authentication challenges to use as the value of the `WWW-Authenticate` header in case of errors. Defaults to `None`.

authenticate(*attributes:* falcon_auth2.utils.classes.RequestAttributes) → dict
Authenticates the request and returns the authenticated user.

2.2.3 Meta Backends

CallBackBackend

class falcon_auth2.backends.**CallBackBackend**(*backend:* falcon_auth2.backends.base.AuthBackend, *, *on_success: Optional[Callable] = None*, *on_failure: Optional[Callable] = None*)

Meta-Backend used to notify when another backend has success and/or fails to authenticate a request.

This backend delegates all the authentication actions to the provided `backend`.

Parameters **backend** (*AuthBackend*) – The backend that will be used to authenticate the requests.

Keyword Arguments

- **on_success** (*Optional[Callable], optional*) – Callable object that will be invoked with the *RequestAttributes*, the `backend` and the authentication result (the dict that will be placed in the request context by the middleware) after a successful request authentication. When using falcon in async mode (asgi), this function may also be async. Defaults to `None`.

  Note: An error will be raised if an async function is used when using falcon in sync mode (wsgi).

- **on_failure** (*Optional[Callable], optional*) – Callable object that will be invoked with the *RequestAttributes*, the `backend` and the raised exception after a failed request authentication. When using falcon in async mode (asgi), this function may also be async. Defaults to `None`.

  Note: An error will be raised if an async function is used when using falcon in sync mode (wsgi).

  Note: This method cannot be used to suppress an exception raised by the `backend` because it will be propagated after `on_failure` invocation ends. The callable may choose to raise a different exception instead.

authenticate(*attributes:* falcon_auth2.utils.classes.RequestAttributes) → dict

Authenticates the request and returns the authenticated user.

MultiAuthBackend

class falcon_auth2.backends.**MultiAuthBackend**(*backends: Iterable[*falcon_auth2.backends.base.AuthBackend*]*, *, *continue_on: Optional[Callable] = None*)

Meta-Backend used to combine multiple authentication backends.

This backend successfully authenticates a request if one of the provided backends can authenticate the request or raises *BackendNotApplicable* if no backend can authenticate it.

This backend delegates all the authentication actions to the provided `backends`.

Parameters **backends** (*Iterable[AuthBackend]*) – The backends to use. They will be used in order.

Keyword Arguments **continue_on** (*Callable*) – A callable object that is called when a backend raises an exception. It should return `True` if processing should continue to the next backend or `False` if it should stop by re-raising the backend exception. The callable takes the backend and the raised exception as parameters. The default implementation continues processing if any backend raises a *BackendNotApplicable* exception.

Note: This callable is only invoked if a backend raises an instance of `HTTPUnauthorized` or one of it's subclasses. All other exception types are propagated.

---

authenticate(*attributes:* falcon_auth2.utils.classes.RequestAttributes)
Authenticates the request and returns the authenticated user.

2.3 Getter

A "Getter" is an instance used by a backend to extract the authentication information from a falcon `Request`.

2.3.1 Getter

class falcon_auth2.**Getter**
Represents a class that extracts authentication information from a request.

Note: Subclasses that wish to only support the *load_async()* method are also required to override the *load()* method since it is defined as abstract. In these cases the sync version may just raise an exception.

---

async_calls_sync_load = None
Indicates if this Getter has an async load implementation that is not just a fallback to sync *load()* method, like the default *load_async()* method.

This property is automatically set by the *Getter* when a subclass is defined (using `__init_subclass__`) if not specified directly by a subclass (by setting it to a valued different than `None`).

abstract load(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str
Loads the specified attribute from the provided request.

If a getter cannot be used with the current request, a *BackendNotApplicable* is raised. The `challenges`, when provided, will be added to `WWW-Authenticate` header in case of error.

Parameters **req** (*Request*) – The current request. This may be a wsgi or an asgi falcon request.

Keyword Arguments **challenges** (*Optional[Iterable[str]], optional*) – One or more authentication challenges to use as the value of the `WWW-Authenticate` header in case of errors.

Returns *str* – The loaded data, in case of success.

async load_async(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str
Async version of *load()*. The default implementation simply calls *load()*, but subclasses may override this implementation to provide an async version.

2.3.2 HeaderGetter

class falcon_auth2.**HeaderGetter**(*header_key: str*)
Returns the specified header from a request.

Parameters **header_key** (*str*) – the name of the header to load.

load(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str
Loads the header from the provided request

2.3.3 AuthHeaderGetter

class falcon_auth2.**AuthHeaderGetter**(*auth_header_type: str*, *, *header_key: str = 'Authorization'*)
Returns the auth header from a request, checking that it in the form `<auth_header_type> value`.

Parameters **auth_header_type** (*str*) – The type of the auth header. Common values are `"Basic"`, `"Bearer"`.

Keyword Arguments **header_key** (*str, optional*) – The name of the header to load. Defaults to `"Authorization"`.

load(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str
Loads the auth header from the provided request

2.3.4 ParamGetter

class falcon_auth2.**ParamGetter**(*param_name: str*)
Returns the specified parameter from the request.

If the parameter appears multiple times an error will be raised.

Note: When the falcon `Request` option `RequestOptions.auto_parse_form_urlencoded` is set to `True`, this getter can also retrieve parameter in the body of a `form-urlencoded` request.

Parameters **param_name** (*str*) – the name of the param to load.

load(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str
Loads the parameter from the provided request

2.3.5 CookieGetter

class falcon_auth2.**CookieGetter**(*cookie_name: str*)
Returns the specified cookie from the request.

If the cookie appears multiple times an error will be raised.

Parameters **cookie_name** (*str*) – the name of the cookie to load.

load(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str
Loads the cookie from the provided request

2.3.6 MultiGetter

class falcon_auth2.**MultiGetter**(*getters: Iterable[*falcon_auth2.getter.Getter*]*)

Combines multiple getters. This is useful if a value can be passed in multiple ways to the server, like using an header or a query parameter.

Will use the first value successfully returned, ignoring all *BackendNotApplicable* exceptions raised by the previously tried getters. If no getter can return a valid value an exception will only be raised.

Parameters **getters** (*Iterable[Getter]*) – The getters to use. They will be tried in order and the first value successfully returned is used.

load(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str

Loads the value from the provided request using the provided getters

async load_async(*req: falcon.request.Request*, *, *challenges: Optional[Iterable[str]] = None*) → str

Async version of *load()*.

Makes sure `load` is called inside a greenlet spawn context

2.4 Exceptions

All exceptions are subclasses of falcon `HTTPUnauthorized`.

class falcon_auth2.**AuthenticationFailure**(*title=None*, *description=None*, *headers=None*, *challenges=None*, ***kwargs*)

Raised when an authentication backend fails to authenticate a syntactically correct request.

This will terminate the request with status 401 if no other logic is present.

class falcon_auth2.**BackendNotApplicable**(*title=None*, *description=None*, *headers=None*, *challenges=None*, ***kwargs*)

Raised when a request is not understood by an authentication backend. This may indicate that the request is intended for another backend.

This will terminate the request with status 401 if no other logic is present.

class falcon_auth2.**UserNotFound**(*title=None*, *description=None*, *headers=None*, *challenges=None*, ***kwargs*)

Raised when the `user_loader` callable of an authentication backend cannot load an user with the received payload. This may indicate that the request is intended for another backend.

This will terminate the request with status 401 if no other logic is present.

2.5 Utils

class falcon_auth2.**RequestAttributes**(*req: falcon.request.Request*, *resp: falcon.response.Response*, *resource: Any*, *params: dict*, *is_async: bool*)

Named tuple that is passed to the backend *authenticate()* when a request is performed.

req: falcon.request.Request

The falcon request.

resp: falcon.response.Response

The falcon response.

resource: Any
The falcon responder resource.

params: dict
The parameters of passed in the url.

is_async: bool
Indicates that authenticate is running in async mode.

2.5.1 Async utilities

async falcon_auth2.utils.**greenlet_spawn**(*fn: Callable*, **args*, ***kwargs*) → Any
Runs a sync function `fn` in a new greenlet.

The sync function can then use *await_()* to wait for async functions.

Parameters

- **fn** (*Callable*) – The sync callable to call.
- ***args** – Positional arguments to pass to the `fn` callable.
- ****kwargs** – Keyword arguments to pass to the `fn` callable.

Returns *Any* – The return value of `fn` or raises an exception if it raised one.

falcon_auth2.utils.**await_**(*awaitable: Coroutine*) → Any
Awaits an async function in a sync method.

The sync method must be insice a *greenlet_spawn()* context. *await_()* calls cannot be nested.

Parameters **awaitable** (*Coroutine*) – The coroutine to call.

Raises **RuntimeError** – If `await_` was called outside a *greenlet_spawn()* context or nested in another `await_` call.

Returns *Any* – The return value of `awaitable` or raises an exception if it raised one.

falcon_auth2.utils.**call_maybe_async**(*support_async: bool*, *function_is_async: Optional[bool]*, *err_msg: str*, *function: Callable*, **args*, ***kwargs*) → Tuple[Any, bool]
Calls a function and waits for the result if it is async.

Parameters

- **support_async** (*bool*) – Can run async cuntions.
- **function_is_async** (*Optional[bool]*) – If the function is async. This function will determine if `function` is async when this parameter is `None`.
- **err_msg** (*str*) – Name of the function. Used in case of error.
- **function** (*Callable*) – The function to call.
- ***args** – Positional arguments to pass to the `function` callable.
- ****kwargs** – Keyword arguments to pass to the `function` callable.

Raises **TypeError** – if `function` is async and `support_async=False`.

Returns *Tuple[Any, bool]* – Returns the result and whatever the function is async.

INDEX

U