# aioauth

*Release 0.1.6*

**Ali Aliyev**

**Mar 12, 2021**

# CONTENTS

aioauth is a spec-compliant OAuth 2.0 asynchronous Python module. aioauth works out-of-the-box with asynchronous server frameworks like FastAPI, Starlette, aiohttp, and others, as well as asynchronous database modules like Motor (MongoDB), aiopg (PostgreSQL), aiomysql (MySQL), or ORMs like Gino, sqlalchemy, or Tortoise.

The magic of aioauth is its plug-and-play methods that allow the use of virtually any server or database framework.

CHAPTER

ONE

# INSTALLING

To install aioauth at the command line:

```
$ pip install aioauth
```

To install pre-releases:

```
$ pip install git+https://github.com/aliev/aioauth
```

CHAPTER

TWO

# SUPPORTED RFC

aioauth supports the following RFCs:

- RFC 6749 - The OAuth 2.0 Authorization Framework
- RFC 7662 - OAuth 2.0 Token Introspection
- RFC 7636 - Proof Key for Code Exchange by OAuth Public Clients

CHAPTER

# THREE

# PAGES

- Github Project
- Issues
- Discussion

CHAPTER

# FOUR

# SECTIONS

## 4.1 Plug-and-Play

aioauth was designed to be as flexible as possible to allow developers to choose their own server framework, as well as database provider. Since aioauth is written as an asynchronous module it would be to the advantage of the developer to write their applications asynchronously.

## 4.2 Configuration

## 4.3 Server & Database

## 4.4 Aiohttp

## 4.5 FastAPI

## 4.6 Base

### 4.6.1 Database

**class BaseDB**

**async create_token**(*request:* aioauth.requests.Request, *client_id: str*, *scope: str*) → *aioauth.models.Token*

Generates Token model instance.

Generated Token MUST be stored in database.

Method is used by all core grant types. Method is used by response types:

- ResponseTypeToken

**async get_token**(*request:* aioauth.requests.Request, *client_id: str*, *access_token: Optional[str] = None*, *refresh_token: Optional[str] = None*) → Optional[*aioauth.models.Token*]

Gets existing token from the database

**Method is used by:**

- create_token_introspection_response

**Method is used by grant types:**

- RefreshTokenGrantType

**async create_authorization_code**(*request:* aioauth.requests.Request, *client_id: str*, *scope: str*, *response_type:* aioauth.types.ResponseType, *redirect_uri: str*, *code_challenge_method:* aioauth.types.CodeChallengeMethod, *code_challenge: str*) → *aioauth.models.AuthorizationCode*

Generates AuthorizationCode model instance.

Generated AuthorizationCode MUST be stored in database.

**Method is used by response types:**

- ResponseTypeAuthorizationCode

**async get_client**(*request:* aioauth.requests.Request, *client_id: str*, *client_secret: Optional[str] = None*) → Optional[*aioauth.models.Client*]

Gets existing Client from database.

If client doesn't exists in database this method MUST return None to indicate to the validator that the requested `client_id` does not exist or is invalid.

Method is used by all core grant types. Method is used by all core response types.

**async authenticate**(*request:* aioauth.requests.Request) → bool

Authenticate user.

**Method is used by grant types:**

- PasswordGrantType

**async get_authorization_code**(*request:* aioauth.requests.Request, *client_id: str*, *code: str*) → Optional[*aioauth.models.AuthorizationCode*]

Gets existing AuthorizationCode from database.

If authorization code doesn't exists it MUST return None to indicate to the validator that the requested authorization code does not exist or is invalid.

**Method is used by grant types:**

- AuthorizationCodeGrantType

**async delete_authorization_code**(*request:* aioauth.requests.Request, *client_id: str*, *code: str*) → None

Deletes authorization code from database.

**Method is used by grant types:**

- AuthorizationCodeGrantType

**async revoke_token**(*request:* aioauth.requests.Request, *refresh_token: str*) → None

Revokes token in database.

This method MUST set *revoked* in True for existing token record.

**Method is used by grant types:**

- RefreshTokenGrantType

### 4.6.2 Server

**class BaseAuthorizationServer**(*db:* aioauth.base.database.BaseDB)

**response_type: Dict[Optional[*aioauth.types.ResponseType*], Type[*aioauth.response_type.*

**grant_type: Dict[Optional[*aioauth.types.GrantType*], Type[*aioauth.grant_type.GrantType*

**register**(*endpoint_type:* aioauth.types.EndpointType, *server: Union[*aioauth.types.ResponseType, aioauth.types.GrantType*]*, *endpoint_cls: Union[Type[*aioauth.response_type.ResponseTypeBase*],* *Type[*aioauth.grant_type.GrantTypeBase*]]*)

**unregister**(*endpoint_type:* aioauth.types.EndpointType, *server: Union[*aioauth.types.ResponseType, aioauth.types.GrantType*]*)

### 4.6.3 Request Validator

**class BaseRequestValidator**(*db:* aioauth.base.database.BaseDB)

**allowed_methods = [<RequestMethod.GET: 'GET'>, <RequestMethod.POST: 'POST'>]**

**async validate_request**(*request:* aioauth.requests.Request)

## 4.7 Config

```
from aioauth import config
```

Configuration settings for aioauth server instance.

---

**class Settings**

Configuration options that is used by the Server class.

**TOKEN_EXPIRES_IN: int = 86400**

Access token lifetime in seconds. Defaults to 24 hours.

**AUTHORIZATION_CODE_EXPIRES_IN: int = 300**

Authorization code lifetime in seconds. Defaults to 5 minutes.

**INSECURE_TRANSPORT: bool = False**

Allow connections over SSL only.

---

**Note:** When this option is disabled server will raise "HTTP method is not allowed" error when attempting to access the server without a valid SSL tunnel.

---

**ERROR_URI: str = ''**

URI to redirect resource owner when server encounters error.

**AVAILABLE: bool = True**

Boolean indicating whether or not the server is available.

## 4.8 Constances

```
from aioauth import constances
```

Constants that are used throughout the project.

---

**default_headers = {'content-type': 'application/json', 'cache-control': 'no-store', 'prag**

The authorization server **must** include the HTTP `Cache-Control` response header field, as per RFC2616, with a value of `no-store` in any response containing tokens, credentials, or other sensitive information, as well as the `Pragma` response header field, as per RFC2616, with a value of `no-cache`.

## 4.9 Errors

```
from aioauth import errors
```

Errors used throughout the project.

---

**exception MethodNotAllowedError**(*request:* aioauth.requests.Request, *description: Optional[str] = None*, *headers: Optional[*aioauth.structures.CaseInsensitiveDict*] = None*)

The request is valid, but the method trying to be accessed is not available to the resource owner.

**description: str = 'HTTP method is not allowed.'**

**status_code: http.HTTPStatus = 405**

**error: *aioauth.types.ErrorType* = 'method_is_not_allowed'**

**exception InvalidRequestError**(*request:* aioauth.requests.Request, *description: Optional[str] = None*, *headers: Optional[*aioauth.structures.CaseInsensitiveDict*] = None*)

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.

**error: *aioauth.types.ErrorType* = 'invalid_request'**

**exception InvalidClientError**(*request:* aioauth.requests.Request, *description: Optional[str] = None*, *headers: Optional[*aioauth.structures.CaseInsensitiveDict*] = None*)

Client authentication failed (e.g. unknown client, no client authentication included, or unsupported authentication method). The authorization server **may** return an `HTTP 401` (Unauthorized) status code to indicate which HTTP authentication schemes are supported. If the client attempted to authenticate via the `Authorization` request header field, the authorization server **must** respond with an `HTTP 401` (Unauthorized) status code, and include the `WWW-Authenticate` response header field matching the authentication scheme used by the client.

**error: *aioauth.types.ErrorType* = 'invalid_client'**

**status_code: http.HTTPStatus = 401**

**exception InsecureTransportError**(*request:* aioauth.requests.Request, *description: Optional[str] = None*, *headers: Optional[*aioauth.structures.CaseInsensitiveDict*] = None*)

An exception will be thrown if the current request is not secure.

**description: str = 'OAuth 2 MUST utilize https.'**

**error: *aioauth.types.ErrorType* = 'insecure_transport'**

**exception UnsupportedGrantTypeError**(*request: aioauth.requests.Request, description: Optional[str] = None, headers: Optional[aioauth.structures.CaseInsensitiveDict] = None*)

The authorization grant type is not supported by the authorization server.

**error: *aioauth.types.ErrorType* = 'unsupported_grant_type'**

**exception UnsupportedResponseTypeError**(*request: aioauth.requests.Request, description: Optional[str] = None, headers: Optional[aioauth.structures.CaseInsensitiveDict] = None*)

The authorization server does not support obtaining an authorization code using this method.

**error: *aioauth.types.ErrorType* = 'unsupported_response_type'**

**exception InvalidGrantError**(*request: aioauth.requests.Request, description: Optional[str] = None, headers: Optional[aioauth.structures.CaseInsensitiveDict] = None*)

The provided authorization grant (e.g. authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, does not match the redirection URI used in the authorization request, or was issued to another client.

See RFC6749 section 5.2.

**error: *aioauth.types.ErrorType* = 'invalid_grant'**

**exception MismatchingStateError**(*request: aioauth.requests.Request, description: Optional[str] = None, headers: Optional[aioauth.structures.CaseInsensitiveDict] = None*)

Unable to securely verify the integrity of the request and response.

**description: str = 'CSRF Warning! State not equal in request and response.'**

**error: *aioauth.types.ErrorType* = 'mismatching_state'**

**exception UnauthorizedClientError**(*request: aioauth.requests.Request, description: Optional[str] = None, headers: Optional[aioauth.structures.CaseInsensitiveDict] = None*)

The authenticated client is not authorized to use this authorization grant type.

**error: *aioauth.types.ErrorType* = 'unauthorized_client'**

**exception InvalidScopeError**(*request: aioauth.requests.Request, description: Optional[str] = None, headers: Optional[aioauth.structures.CaseInsensitiveDict] = None*)

The requested scope is invalid, unknown, or malformed, or exceeds the scope granted by the resource owner.

See RFC6749 section 5.2.

**error: *aioauth.types.ErrorType* = 'invalid_scope'**

**exception ServerError**(*request: aioauth.requests.Request, description: Optional[str] = None, headers: Optional[aioauth.structures.CaseInsensitiveDict] = None*)

The authorization server encountered an unexpected condition that prevented it from fulfilling the request. (This error code is needed because a `HTTP 500` (Internal Server Error) status code cannot be returned to the client via a HTTP redirect.)

**error: *aioauth.types.ErrorType* = 'server_error'**

**exception TemporarilyUnavailableError**(*request:* aioauth.requests.Request, *description: Optional[str] = None*, *headers: Optional[*aioauth.structures.CaseInsensitiveDict*] = None*)

The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server. (This error code is needed because a `HTTP 503` (Service Unavailable) status code cannot be returned to the client via a HTTP redirect.)

**error:** *aioauth.types.ErrorType* **= 'temporarily_unavailable'**

## 4.10 Grant Type

```
from aioauth import grant_type
```

Different OAuth 2.0 grant types.

---

**class GrantTypeBase**(*db:* aioauth.base.database.BaseDB)

Base grant type that all other grant types inherit from.

**allowed_methods = [<RequestMethod.POST: 'POST'>]**

**grant_type:** **Optional[*aioauth.types.GrantType*] = None**

**async validate_request**(*request:* aioauth.requests.Request) → *aioauth.models.Client*

Validates the client request to ensure it is valid.

**async create_token_response**(*request:* aioauth.requests.Request) → *aioauth.responses.TokenResponse*

Creates token response to reply to client.

**get_client_credentials**(*request:* aioauth.requests.Request) → Tuple[str, str]

**class AuthorizationCodeGrantType**(*db:* aioauth.base.database.BaseDB)

The Authorization Code grant type is used by confidential and public clients to exchange an authorization code for an access token. After the user returns to the client via the redirect URL, the application will get the authorization code from the URL and use it to request an access token.

It is recommended that all clients use RFC 7636 Proof Key for Code Exchange extension with this flow as well to provide better security. Note that `aioauth` implements RFC 7636 out-of-the-box.

See RFC 6749 section 1.3.1.

**grant_type:** *aioauth.types.GrantType* **= 'authorization_code'**

**async validate_request**(*request:* aioauth.requests.Request) → *aioauth.models.Client*

Validates the client request to ensure it is valid.

**class PasswordGrantType**(*db:* aioauth.base.database.BaseDB)

The Password grant type is a way to exchange a user's credentials for an access token. Because the client application has to collect the user's password and send it to the authorization server, it is not recommended that this grant be used at all anymore.

See RFC 6749 section 1.3.3.

The latest OAuth 2.0 Security Best Current Practice disallows the password grant entirely.

**grant_type:** *aioauth.types.GrantType* **= 'password'**

**async validate_request**(*request:* aioauth.requests.Request) → *aioauth.models.Client*

Validates the client request to ensure it is valid.

**class RefreshTokenGrantType**(*db:* aioauth.base.database.BaseDB)

The Refresh Token grant type is used by clients to exchange a refresh token for an access token when the access token has expired. This allows clients to continue to have a valid access token without further interaction with the user.

See RFC 6749 section 1.5.

**grant_type:** *aioauth.types.GrantType* **= 'refresh_token'**

**async validate_request**(*request:* aioauth.requests.Request) → *aioauth.models.Client*

Validates the client request to ensure it is valid.

**async create_token_response**(*request:* aioauth.requests.Request) → *aioauth.responses.TokenResponse*

Creates token response to reply to client.

**class ClientCredentialsGrantType**(*db:* aioauth.base.database.BaseDB)

The Client Credentials grant type is used by clients to obtain an access token outside of the context of a user. This is typically used by clients to access resources about themselves rather than to access a user's resources.

See RFC 6749 section 4.4.

**grant_type:** *aioauth.types.GrantType* **= 'client_credentials'**

## 4.11 Models

```
from aioauth import models
```

Memory objects used throughout the project.

---

**class Client**

OAuth2.0 client model object.

**client_id: str**

Public identifier for the client. It must also be unique across all clients that the authorization server handles.

**client_secret: str**

Client secret is a secret known only to the client and the authorization server. Used for secure communication between the client and authorization server.

**grant_types: List[*aioauth.types.GrantType*] = []**

The method(s) in which an application gets an access token from the provider. Each grant type is optimized for a particular use case, whether that's a web app, a native app, a device without the ability to launch a web browser, or server-to-server applications.

**redirect_uris: List[str] = []**

After a user successfully authorizes an application, the authorization server will redirect the user back to the application with either an authorization code or access token in the URL. Because the redirect URL will contain sensitive information, it is critical that the service doesn't redirect the user to arbitrary locations.

**scope: str = ''**

Scope is a mechanism that limit an application's access to a user's account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted.

**response_types: List[*aioauth.types.ResponseType*] = []**

A list containing the types of the response expected.

**check_redirect_uri**(*redirect_uri: str*) → bool
Verifies passed `redirect_uri` is part of the Clients's `redirect_uris` list.

**check_grant_type**(*grant_type:* aioauth.types.GrantType) → bool
Verifies passed `grant_type` is part of the client's `grant_types` list.

**check_response_type**(*response_type:* aioauth.types.ResponseType) → bool
Verifies passed `response_type` is part of the client's `response_types` list.

**get_allowed_scope**(*scope: str*) → str
Returns the allowed `scope` given the passed `scope`.

> **Note:** Note that the passed `scope` may contain multiple scopes seperated by a space character.

**check_scope**(*scope: str*) → bool
Checks if passed `scope` is allowed for the client.

**class AuthorizationCode**

**code: str**
Authorization code that the client previously received from the authorization server.

**client_id: str**
Public identifier for the client. It must also be unique across all clients that the authorization server handles.

**redirect_uri: str**
After a user successfully authorizes an application, the authorization server will redirect the user back to the application with either an authorization code or access token in the URL. Because the redirect URL will contain sensitive information, it is critical that the service doesn't redirect the user to arbitrary locations.

**response_type: *aioauth.types.ResponseType***
A string containing the type of the response expected.

**scope: str**
Scope is a mechanism that limit an application's access to a user's account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted.

**auth_time: int**
JSON Web Token Claim indicating the time when the authentication occurred.

**code_challenge: Optional[str] = None**
Only used when RFC 7636, Proof Key for Code Exchange, is used.

PKCE works by having the app generate a random value at the beginning of the flow called a Code Verifier. The app hashes the Code Verifier and the result is called the Code Challenge. The app then kicks off the flow in the normal way, except that it includes the Code Challenge in the query string for the request to the Authorization Server.

**code_challenge_method: Optional[*aioauth.types.CodeChallengeMethod*] = None**
Only used when RFC 7636, Proof Key for Code Exchange, is used.

Method used to transform the code verifier into the code challenge.

**nonce: Optional[str] = None**
Only used when RFC 7636, Proof Key for Code Exchange, is used.

Random piece of data.

**check_code_challenge**(*code_verifier: str*) → bool
Verifies the code challenge.

**is_expired**(*request:* aioauth.requests.Request) → bool
Checks if the authorization time has expired.

**class Token**

**access_token: str**
Token that clients use to make API requests on behalf of the resource owner.

**refresh_token: str**
Token used by clients to exchange a refresh token for an access token when the access token has expired.

**scope: str**
Scope is a mechanism that limit an application's access to a user's account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted.

**issued_at: int**
Time date in which token was issued at.

**expires_in: int**
Time delta in which token will expire. *token_expires_in()* will give the date time for which the token is to expire.

**client_id: str**
Public identifier for the client. It must also be unique across all clients that the authorization server handles.

**token_type: str = 'Bearer'**
Type of token expected.

**revoked: bool = False**
Flag that indicates whether or not the token has been revoked.

**is_expired**(*request:* aioauth.requests.Request) → bool
Checks if the token has expired.

**property refresh_token_expires_in**
Refreshes the 'expires_in' parameter.

**property token_expires_in**
Time date in which the token will expire in.

**property refresh_token_expired**
Checks if refresh token has expired.

## 4.12 Requests

```
from aioauth import requests
```

Request objects used throughout the project.

---

**class Query**
Object that contains a client's query string portion of a request. Read more on query strings here.

**client_id: Optional[str] = None**

**redirect_uri:** str = ''

**response_type:** Optional[*aioauth.types.ResponseType*] = None

**state:** str = ''

**scope:** str = ''

**code_challenge_method:** Optional[*aioauth.types.CodeChallengeMethod*] = None

**code_challenge:** Optional[str] = None

**class Post**

Object that contains a client's post request portion of a request. Read more on post requests here.

**grant_type:** Optional[*aioauth.types.GrantType*] = None

**client_id:** Optional[str] = None

**client_secret:** Optional[str] = None

**redirect_uri:** Optional[str] = None

**scope:** str = ''

**username:** Optional[str] = None

**password:** Optional[str] = None

**refresh_token:** Optional[str] = None

**code:** Optional[str] = None

**token:** Optional[str] = None

**code_verifier:** Optional[str] = None

**class Request**

Object that contains a client's complete request.

**method:** *aioauth.types.RequestMethod*

**headers:** *aioauth.structures.CaseInsensitiveDict* = {}

**query:** *aioauth.requests.Query* = Query()

**post:** *aioauth.requests.Post* = Post()

**url:** str = ''

**user:** Optional[Any] = None

**settings:** *aioauth.config.Settings* = Settings()

## 4.13 Response Type

```
from aioauth import responses
```

Response objects used throughout the project.

---

**class ResponseTypeBase** (*db:* aioauth.base.database.BaseDB)

Base response type that all other exceptions inherit from.

**response_type:** Optional[*aioauth.types.ResponseType*] = None

**allowed_methods = [<RequestMethod.GET: 'GET'>]**

**code_challenge_methods = [<CodeChallengeMethod.PLAIN: 'plain'>, <CodeChallengeMethod.S**

**async validate_request**(*request:* aioauth.requests.Request) → *aioauth.models.Client*
Validates the client request to ensure it is valid.

**async create_authorization_response**(*request:* aioauth.requests.Request) → *aioauth.models.Client*
Validate authorization request and create authorization response.

**class ResponseTypeToken**(*db:* aioauth.base.database.BaseDB)
Response type that contains a token.

**response_type: *aioauth.types.ResponseType* = 'token'**

**async create_authorization_response**(*request:* aioauth.requests.Request) → *aioauth.responses.TokenResponse*
Validate authorization request and create authorization response.

**class ResponseTypeAuthorizationCode**(*db:* aioauth.base.database.BaseDB)
Response type that contains an authorization code.

**response_type: *aioauth.types.ResponseType* = 'code'**

**async create_authorization_response**(*request:* aioauth.requests.Request) → *aioauth.responses.AuthorizationCodeResponse*
Validate authorization request and create authorization response.

## 4.14 Responses

```
from aioauth import responses
```

Response objects used throughout the project.

---

**class ErrorResponse**
Response for errors.

**error: *aioauth.types.ErrorType***

**description: str**

**error_uri: str = ''**

**class AuthorizationCodeResponse**
Response for `authorization_code`.

**code: str**

**scope: str**

**class TokenResponse**
Response for token.

**expires_in: int**

**refresh_token_expires_in: int**

**access_token: str**

**refresh_token: str**

**scope:** **str**

**token_type:** **str = 'Bearer'**

**class TokenActiveIntrospectionResponse**
Response for a valid access token.

**scope:** **str**

**client_id:** **str**

**exp:** **int**

**active:** **bool = True**

**class TokenInactiveIntrospectionResponse**
For an invalid, revoked or expired token.

**active:** **bool = False**

**class Response**
General response class.

**content:** **Optional[Union[*aioauth.responses.ErrorResponse*, *aioauth.responses.TokenRespo***

**status_code:** **http.HTTPStatus = 200**

**headers:** ***aioauth.structures.CaseInsensitiveDict*** **= {'content-type': 'application/json**

## 4.15 Server

```
from aioauth import server
```

Memory object and interface used to initialize an OAuth2.0 server instance.

**Warning:** Note that *aioauth.server.AuthorizationServer* is not depedent on any server framework, nor serves at any specific endpoint. Instead, it is used to create an interface that can be used in conjunction with a server framework like `FastAPI` or `aiohttp` to create a fully functional OAuth 2.0 server.

Check out the *Examples* portion of the documentation to understand how it can be leveraged in your own project.

---

**class AuthorizationServer**(*db:* aioauth.base.database.BaseDB)
Interface for initializing an OAuth 2.0 server.

**async create_token_response**(*request:* aioauth.requests.Request) → *aioauth.responses.Response*
Endpoint to obtain an access and/or ID token by presenting an authorization grant or refresh token.

Validates a token request and creates a token response.

For more information see RFC6749 section 4.1.3.

**Note:** The API endpoint that leverages this function is usually `/token`.

### Example

Below is an example utilizing FastAPI as the server framework.

```
@app.post("/token")
async def token(request: fastapi.Request) -> fastapi.Response:

    # Converts a fastapi.Request to an aioauth.Request.
    oauth2_request: aioauth.Request = await to_oauth2_request(request)

    # Creates the response via this function call.
    oauth2_response: aioauth.Response = await server.create_token_
↪response(oauth2_request)

    # Converts an aioauth.Response to a fastapi.Response.
    response: fastapi.Response = await to_fastapi_response(oauth2_response)

    return response
```

**Parameters** **request** – An `aioauth` request object.

**Returns** An `aioauth` response object.

**Return type** response

**async create_authorization_response**(*request: aioauth.requests.Request*) → *aioauth.responses.Response*

Endpoint to interact with the resource owner and obtain an authorization grant.

Validate authorization request and create authorization response.

For more information see RFC6749 section 4.1.1.

---

**Note:** The API endpoint that leverages this function is usually `/authorize`.

---

### Example

Below is an example utilizing FastAPI as the server framework.

```
@app.post("/authorize")
async def authorize(request: fastapi.Request) -> fastapi.Response:

    # Converts a fastapi.Request to an aioauth.Request.
    oauth2_request: aioauth.Request = await to_oauth2_request(request)

    # Creates the response via this function call.
    oauth2_response: aioauth.Response = await server.create_authorization_
↪response(oauth2_request)

    # Converts an aioauth.Response to a fastapi.Response.
    response: fastapi.Response = await to_fastapi_response(oauth2_response)

    return response
```

**Parameters** **request** – An `aioauth` request object.

**Returns** An `aioauth` response object.

**Return type** response

**async create_token_introspection_response**(*request: aioauth.requests.Request*) → *aioauth.responses.Response*

Returns a response object with introspection of the passed token.

For more information see RFC7662 section 2.1.

**Note:** The API endpoint that leverages this function is usually `/introspect`.

### Example

Below is an example utilizing FastAPI as the server framework.

```
@app.get("/introspect")
async def introspect(request: fastapi.Request) -> fastapi.Response:

    # Converts a fastapi.Request to an aioauth.Request.
    oauth2_request: aioauth.Request = await to_oauth2_request(request)

    # Creates the response via this function call.
    oauth2_response: aioauth.Response = await server.create_token_
↪introspection_response(oauth2_request)

    # Converts an aioauth.Response to a fastapi.Response.
    response: fastapi.Response = await to_fastapi_response(oauth2_response)

    return response
```

**Parameters** **request** – An `aioauth` request object.

**Returns** An `aioauth` response object.

**Return type** response

## 4.16 Structures

```
from aioauth import structures
```

Structures that are used throughout the project.

**class CaseInsensitiveDict**(*dict=None*, */*, ***kwargs*)

A case-insensitive `dict`-like object.

#### Example

```
from aioauth.structures import CaseInsensitiveDict

d = CaseInsensitiveDict({"hello": "world"})
d['hello'] == 'world' # >>> True
d['Hello'] == 'world' # >>> True
d['hElLo'] == 'world' # >>> True
```

## 4.17 Types

```
from aioauth import types
```

Containers that contain constants used throughout the project.

---

**class ErrorType**(*value*)

Error types.

**INVALID_REQUEST = 'invalid_request'**

**INVALID_CLIENT = 'invalid_client'**

**INVALID_GRANT = 'invalid_grant'**

**INVALID_SCOPE = 'invalid_scope'**

**UNAUTHORIZED_CLIENT = 'unauthorized_client'**

**UNSUPPORTED_GRANT_TYPE = 'unsupported_grant_type'**

**UNSUPPORTED_RESPONSE_TYPE = 'unsupported_response_type'**

**INSECURE_TRANSPORT = 'insecure_transport'**

**MISMATCHING_STATE = 'mismatching_state'**

**METHOD_IS_NOT_ALLOWED = 'method_is_not_allowed'**

**SERVER_ERROR = 'server_error'**

**TEMPORARILY_UNAVAILABLE = 'temporarily_unavailable'**

**class GrantType**(*value*)

Grant types.

**TYPE_AUTHORIZATION_CODE = 'authorization_code'**

**TYPE_PASSWORD = 'password'**

**TYPE_CLIENT_CREDENTIALS = 'client_credentials'**

**TYPE_REFRESH_TOKEN = 'refresh_token'**

**class ResponseType**(*value*)

Response types.

**TYPE_TOKEN = 'token'**

**TYPE_CODE = 'code'**

**class EndpointType**(*value*)
Endpoint types.

**GRANT_TYPE = 'grant_type'**

**RESPONSE_TYPE = 'response_type'**

**class RequestMethod**(*value*)
Request types.

**GET = 'GET'**

**POST = 'POST'**

**class CodeChallengeMethod**(*value*)
Code challenge types.

**PLAIN = 'plain'**

**S256 = 'S256'**

## 4.18 Utils

```
from aioauth import utils
```

Contains helper functions that is used throughout the project that doesn't pertain to a specific file or module.

---

**is_secure_transport**(*request:* aioauth.requests.Request) → bool
Verifies the request was sent via a protected SSL tunnel.

> **Note:** This method simply checks if the request URL contains `https://` at the start of it. It does **not** ensure if the SSL certificate is valid.

> **Parameters** **request** – A request object.
>
> **Returns** Flag representing whether or not the transport is secure.

**get_authorization_scheme_param**(*authorization_header_value: str*) → Tuple[str, str]
Retrieves the authorization schema parameters from the authorization header.

> **Parameters** **authorization_header_value** – Value of the authorization header.
>
> **Returns** Tuple of the format `(scheme, param)`.

**list_to_scope**(*scope: Optional[Union[str, Set, Tuple, List]] = None*) → str
Converts a list of scopes to a space separated string.

> **Note:** If a string is passed to this method it will simply return an empty string back. Use *scope_to_list()* to convert strings to scope lists.

> **Parameters** **scope** – An iterable or string that contains a list of scope.
>
> **Returns** A string of scopes seperated by spaces.
>
> **Raises** **TypeError** – The `scope` value passed is not of the proper type.

**scope_to_list**(*scope: Optional[Union[str, Set, Tuple, List]] = None*) → List
Converts a space separated string to a list of scopes.

---

**Note:** If an iterable is passed to this method it will return a list representation of the iterable. Use *list_to_scope()* to convert iterables to a scope string.

---

**Parameters** **scope** – An iterable or string that contains scopes.

**Returns** A list of scopes.

**Raises** **TypeError** – The `scope` value passed is not of the proper type.

**generate_token**(*length: int = 30*, *chars: str = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'*) → str
Generates a non-guessable OAuth token.

OAuth (1 and 2) does not specify the format of tokens except that they should be strings of random characters. Tokens should not be guessable and entropy when generating the random characters is important. Which is why SystemRandom is used instead of the default random.choice method.

**Parameters**

- **length** – Length of the generated token.
- **chars** – The characters to use to generate the string.

**Returns** Random string of length `length` and characters in `chars`.

**build_uri**(*url: str*, *query_params: Optional[Dict] = None*, *fragment: Optional[Dict] = None*) → str
Builds an URI string from passed `url`, `query_params`, and `fragment`.

**Parameters**

- **url** – URL string.
- **query_params** – Paramaters that contain the query.
- **fragment** – Fragment of the page.

**Returns** URL containing the original `url`, and the added `query_params` and `fragment`.

**encode_auth_headers**(*client_id: str*, *client_secret: str*) → *aioauth.structures.CaseInsensitiveDict*
Encodes the authentication header using base64 encoding.

**Parameters**

- **client_id** – The client's id.
- **client_secret** – The client's secret.

**Returns** A case insensitive dictionary that contains the `Authorization` header set to `basic` and the authorization header.

**decode_auth_headers**(*request:* aioauth.requests.Request) → Tuple[str, str]
Decodes an encrypted HTTP basic authentication string.

Returns a tuple of the form `(client_id, client_secret)`, and raises a *aioauth.errors.InvalidClientError* exception if nothing could be decoded.

**Parameters** **request** – A request object.

**Returns** Tuple of the form `(client_id, client_secret)`.

**Raises** ***aioauth.errors.InvalidClientError*** – Could not be decoded.

**create_s256_code_challenge**(*code_verifier: str*) → str
Create S256 code challenge with the passed `code_verifier`.

**Note:** This function implements `base64url(sha256(ascii(code_verifier)))`.

**Parameters** **code_verifier** – Code verifier string.

**Returns** Representation of the S256 code challenge with the passed `code_verifier`.

**catch_errors_and_unavailability**(*f: Callable*) → Callable
Decorator that adds error catching to the function passed.

**Parameters** **f** – A callable.

**Returns** A callable with error catching capabilities.

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## a

# INDEX

## D



## E



## G



## H



## I

## L



## M



## N



## P



## Q



## R



## S

## T



## U



## V