Animalia SSO

Implementation guide

*Please contact us if you have any bug reports, questions, or suggestions. Cooperation with our partners is important to us, and we greatly appreciate any feedback. Animalia IT can be reached at* [*itdrift@animalia.no*](mailto:itdrift@animalia.no)*.*

Animalia SSO builds on Oauth 2.0 and a subset of OpenID Connect. The system uses a stateless JWT which is delivered by our authentication service. This access token acts as a key for accessing Animalia services for the user through the client. Access tokens must be renewed periodically for continued use of the service. All grants and restrictions follow this token and are valid and static for the whole period. New or updated privileges will not take effect until a new token is requested from Animalia SSO.

Every client will get a client id and client secret. Animalia SSO supports three of the authentication grants described in OAuth 2.0: *authorization code*, *client credentials* and *refresh token*. Authorization code is used when you have users logging into systems. This grant verifies both the user and the client, and is often referred to as the standard OAuth 2.0 flow, or user-to-system. Client credentials are only used in situations where there are no users involved. In this case, only the client is verified, and receives a user-less access token. This is a system-to-system approach, and a typical example may be an external service that polls an API every night.

The implementation part of this document will be split into three parts. One describing the authentication code grant, one describing the client credentials grant, and one describing the refresh token grant. If you are unsure about which of these are right for you, please do not hesitate to contact your representative at Animalia, or Animalia IT at [*itdrift@animalia.no*](mailto:itdrift@animalia.no)*.*

# Content

[Content 1](#_Toc145667770)

[Using the access token 2](#_Toc145667771)

[Implementation 2](#_Toc145667772)

[Authorization code (user-to-system) 2](#_Toc145667773)

[Client implementation 5](#_Toc145667774)

[Request (Animalia SSO authorization service, login box) 5](#_Toc145667775)

[Request (Animalia SSO authorization service, logout) 6](#_Toc145667776)

[Request (Client authorization service example) 6](#_Toc145667777)

[Request (Animalia SSO token service) 7](#_Toc145667778)

[Request (Animalia SSO token service, refresh token) 8](#_Toc145667779)

[OpenID Connect 9](#_Toc145667780)

[Scopes 9](#_Toc145667781)

[Animalia SSO accepts the following general scopes: 9](#_Toc145667782)

[Client credentials (system-to-system) 10](#_Toc145667783)

[Request (Animalia SSO token service) 10](#_Toc145667784)

[Responses (errors) 11](#_Toc145667785)

[Verifying the access token 12](#_Toc145667786)

[Request (Animalia SSO public key endpoint) 12](#_Toc145667787)

# Using the access token

With every request to an Animalia service, the access token must be set in the authorization header, prefixed with “Bearer” like this:

**Authorization:** “Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnQiOiJkaHA…

The receiving service will read the token and decide if the request is allowed and respond in a fitting manner. Usually, an invalid token generates a 403 Forbidden response, while the lack of a token will result in a 401 Unauthorized. This may vary between the different services, and you should consult the documentation for the relevant service for the actual implementation of these responses.

# Implementation

## Authorization code (user-to-system)

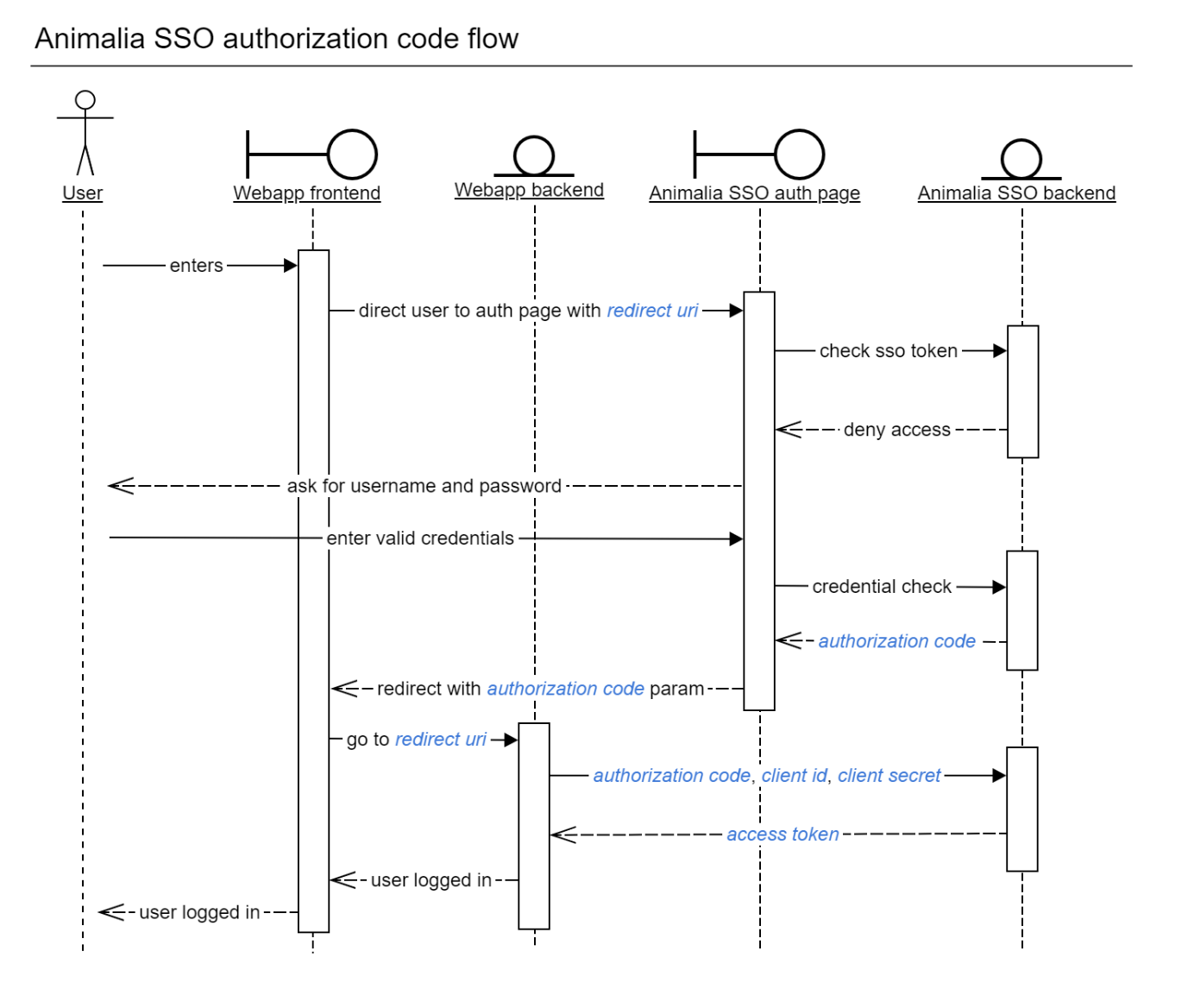
This is the grant that is most commonly associated with OAuth 2.0 and is used to grant a verified user access to an API through a verified client. The purpose of this grant is to create a secure verification process for the user, independently from the client, and in turn verify the client before an access token is dispatched. The client has no access to the user credentials at any time.

The authentication flow is described here as an interaction between the webapp in the browser, the embedded login box and the different backend endpoints. The webapp, typically the backend part, is commonly referred to as the client in the OAuth 2.0 flow, and we will use this term throughout this documentation.

The terms *client id* and *client secret* describe the credentials of the client. This must not be confused with the user credentials, like username and password from the user. The client credentials are only used to verify the client from which the user connects to the different Animalia services. The client secret is a confidential key and should never be disclosed to the user. In other words, it shall never be sent to the browser as part of the client frontend, and only kept as part of the client backend configuration.

The client backend must implement a service that collects the *authorization code* from the user, and send it to the Animalia SSO authorization service, together with the *client id* and *client secret*. The response will return an *access token* that should be used to authenticate the user in all Animalia services.

This client backend service must have a public endpoint. The endpoint uri should be specified as a parameter to the embedded login box in the webapp frontend, called *redirect uri*.

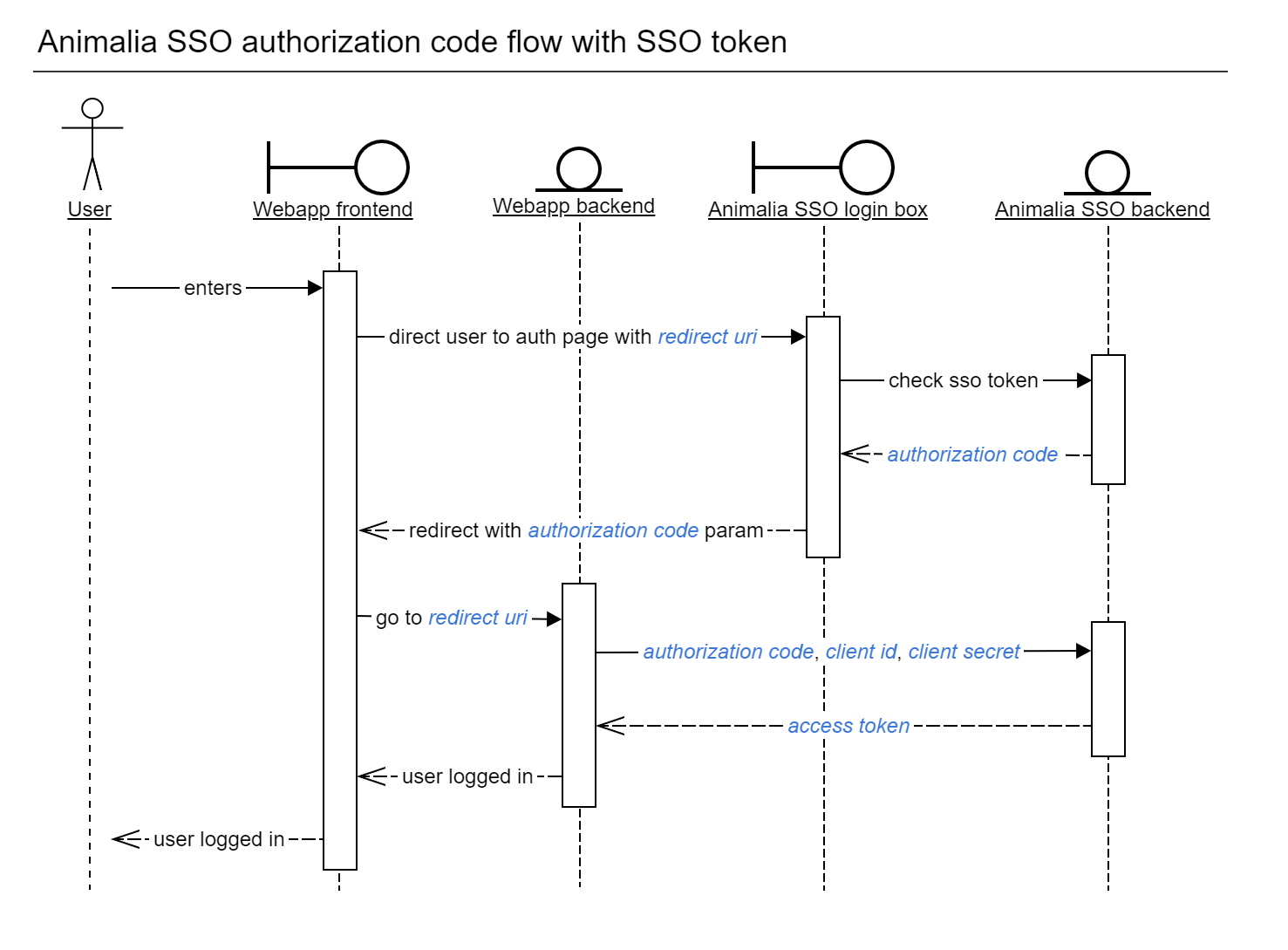


The authentication process can be broken down into the following steps, as seen from the client perspective.:

1. The client redirects the browser to the Animalia SSO login page.
2. The user authenticates, and the browser window redirects to an authentication endpoint that the client has implemented, with an authorization code set as an url parameter.
3. The client backend receives the authorization code and sends it to the Animalia SSO token service alongside the client id and client secret.
4. The response from the token service includes an access token with a lifespan of 30 minutes, granting access to all services the user and client has access to for the whole period.

After the initial authentication, the browser contains a SSO token that the login page can use to automate step 2. In this case, the login page will just do the request to the client backend immediately upon loading, giving an authorization code. The lifespan of the SSO token varies and is set by Animalia for each client individually. There is both an inactivity timeout and an absolute timeout for each client. It also depends on whether the user choosing the “remember me / husk meg” option on the login page. If “remember me” is not chosen on the login page, the SSO token will only last 1 hour and extend if the user is active until it reaches an absolute timeout.

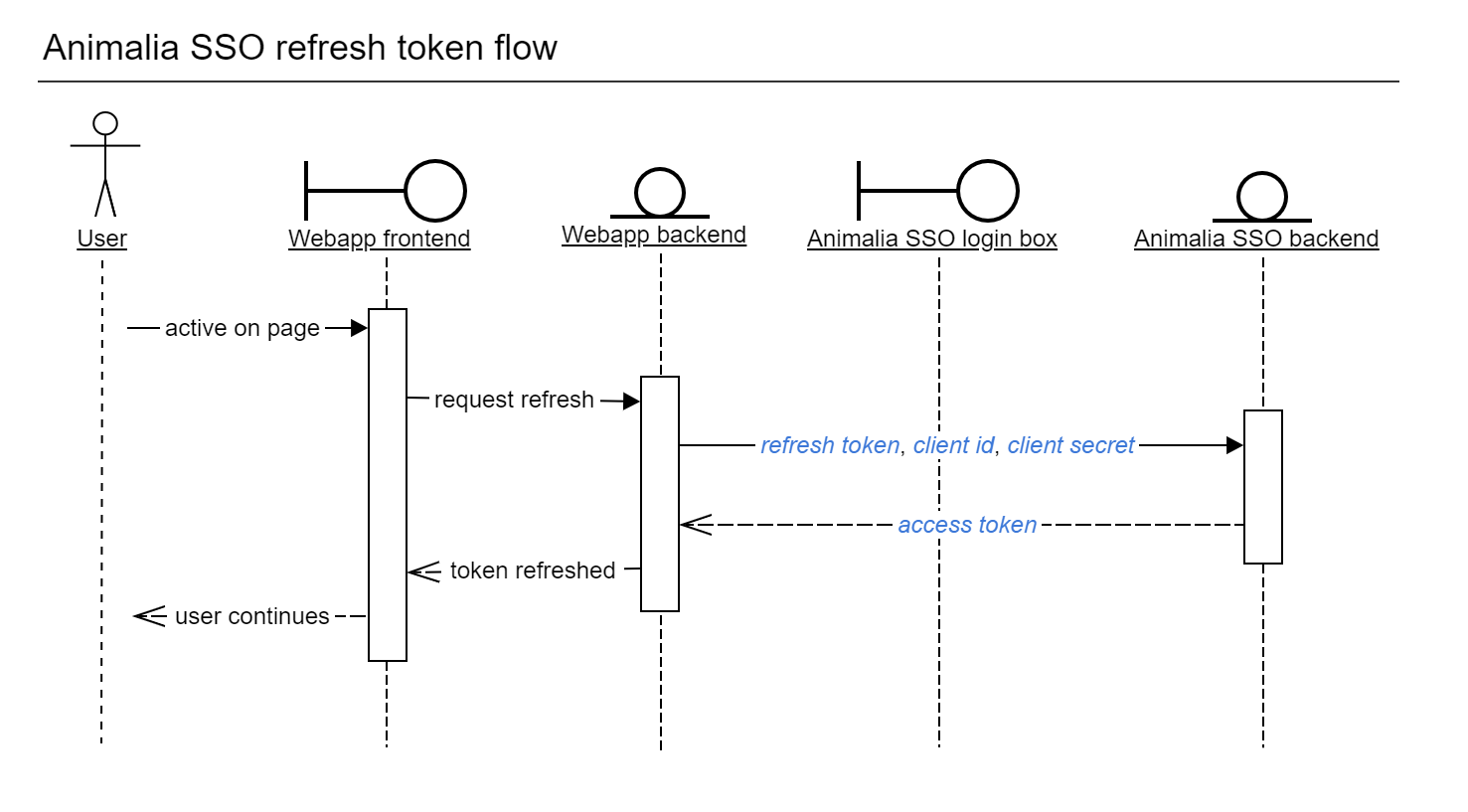
The following sequence diagram describes the authentication flow with a valid SSO token:



As long as the SSO token is valid for a given client, the user will be able to proceed through the authentication without entering credentials. The only interaction required is that the user confirms the logged in username, but this step can be completely automated by setting the query parameter “auto\_login=true” on the /authorization-page.

#### Token refresh

For applications that rely on long uninterrupted sessions, the need to redirect the user back to the auth page every 30 minutes may result in a suboptimal experience. Animalia SSO supports refresh tokens as a way for clients to keep sessions alive uninterrupted. In order to receive refresh tokens, the scope “offline\_access” must be set in the authorization request.



### Client implementation

The steps needed to implement the authorization grant for any client can be roughly broken down into the following steps:

1. Direct the user to the Animalia SSO authorization service, which contains the login box. Include the necessary parameters, like the redirect\_uri and client\_id parameter that points to the service described in step 2.
2. Implement a client authorization service, with an accessible endpoint, in the client backend. This endpoint must accept a GET request from the browser, which will contain a parameter called authorization\_code, initiated as a redirect from the login box. This service must in turn call the Animalia SSO token service with the authorization code, client id and client secret. The response contains an access token.
3. The user is now logged in. The service can respond to the browser request in whatever way fitting for the client. The access token should be put into some kind of storage for its 30-minute lifespan, like a cookie or the browser local storage, and the response can choose to redirect to the “logged in” page or respond with fitting content.

### Request (Animalia SSO authorization service, login box)

* **Staging environment:** [**https://staging-sso.animalia.no/authorize**](https://staging-sso.animalia.no/authorize)
* **Production environment:** [**https://sso.animalia.no/authorize**](https://sso.animalia.no/authorize)

**Allowed-methods:** GET

**Accept:**  text/html

**Query parameters:**

* **redirect\_uri** required string: uri to the client backend auth service
* **client\_id** required string: the client id
* **state** optional string: CSRF anti-forgery token
* **auto\_login** optional boolean: set to true to auto login
* **scope**  optional string: scopes, whitespace separated

The redirect uri parameter tells the login box where to redirect the main browser window when the user has been authenticated. This redirect will include the authorization code, and the state if given.

The state parameter can be set to mitigate CSRF attack vectors. It is highly recommended that the client backend generates a new state, often called an anti-forgery token, that is delivered with the webapp front page on load. This should in turn be set as a parameter to the login page. The login page will return this state key with the request to the redirect uri, and the client backend can then verify that the state key has not changed, and discard requests that do not match. This way, no rouge requests will be allowed.

The auto login parameter tells the login box to automatically process refresh tokens without asking the user. In most cases you would want to set this to true, so that expired access tokens can be refreshed without user interaction.

**Response (success)**

**Status code:** 200 Created

**Content-type:** text/html

**Body:** HTML response containing the login page

### Request (Animalia SSO authorization service, logout)

* **Staging environment:** [**https://staging-sso.animalia.no/logout**](https://staging-sso.animalia.no/logout)
* **Production environment:** [**https://sso.animalia.no/logout**](https://sso.animalia.no/logout)

**Allowed-methods:** GET

**Parameters:**

* **redirect\_uri** optional string: uri to client logout success page

In order for the user to be logged out, the browser must visit this service as part of the logout process. This invalidates the sso token and the refresh token, so that a new user can enter their credentials into the login box. The service accepts an optional parameter redirect\_uri. When used, this allows the client to redirect the user automatically to any page after the logout has been successful.

**Response (success)**

**Status code:** 200 OK

**Content-type:** (matching the Accept header from the request)

**Body:** “SSO-utlogging vellykket”

**Response (redirect)**

**Status code:** 301 Moved permanently

**Location:**  (The redirect\_uri query param from the request)

### Request (Client authorization service example)

This is the service that you need to implement in the client backend. The login box will execute this request after a successful authentication of the user, in order to authenticate the client.

* **My environment:** [**https://my-example.app/authorize**](https://my-example.app/authorize) **(redirect uri)**

**Allowed-methods:** GET

**Accept:**  (not relevant)

**Parameters:**

* **code** required string: the authorization code
* **state** optional string: CSRF anti-forgery token

This service should in turn call the Animalia SSO token service with the authorization code and exchange it for an access token.

**Response (success)**

**Status code:** 302 Found

**Location:** Your “logged in” page

**Cookie:** The access token can be bound to a cookie

How you want to keep the access token is up to you, but a common way to do it is to save it in a cookie. This way you can have a stateless web application that has no information about the user logged in, and only use the given token to authenticate the user on each subsequent request.

### Request (Animalia SSO token service)

* **Staging environment:** [**https://staging-sso.animalia.no/token**](https://staging-sso.animalia.no/token)
* **Production environment:** [**https://sso.animalia.no/token**](https://sso.animalia.no/token)

**Allowed-methods:** POST

**Accept:**  application/json, application/edn

**Parameters:**

* **grant\_type** required string: “authorization\_code”
* **client\_id** requiredstring: the client id
* **client\_secret** required string: the client secret (UUIDv4)
* **code** required string: the authorization code
* **redirect\_uri** required string: redirect uri, given to auth service

The grant type should be set to “authorization\_code” when using this grant.

As this request always should be initiated by the client backend, the client id and client secret are available, and must be added to the request.

The authorization code is delivered from the client frontend, who initiates this chain of requests by calling the client backend with the authorization code.

The redirect uri is used to verify that the authorization code delivered has the correct origin. The authorization code includes a signed fingerprint of the original redirect uri used by the login box, and the token service will check this parameter up against this fingerprint. In other words, this has to be the same as the redirect uri used by the login box.

**Response (success)**

**Status code:** 201 Created

**Content-type:** (matching the Accept header from the request)

**Body:**

* access\_token
* token\_type
* expires\_in

|  |  |  |  |
| --- | --- | --- | --- |
| access\_token | string | 0 .. \* chars | Base64-encoded signed token including the access rights granted to the client and the expiry time |
| refresh\_token | string | 0 .. \* chars | Base64-encoded signed token that can be used to refresh the access token. Only available when the scope “offline\_access” is used. |
| token\_type | string | 0 .. \* chars | Token type. Returns “bearer” in current implementation. |
| expires\_in | integer | 0 .. 231-1 | The lifespan of the token, in seconds. |

### Request (Animalia SSO token service, refresh token)

* **Staging environment:** [**https://staging-sso.animalia.no/token**](https://staging-sso.animalia.no/token)
* **Production environment:** [**https://sso.animalia.no/token**](https://sso.animalia.no/token)

**Allowed-methods:** POST

**Accept:**  application/json, application/edn

**Parameters:**

* **grant\_type** required string: “refresh\_token”
* **client\_id** requiredstring: the client id
* **client\_secret** required string: the client secret (UUIDv4)
* **refresh\_token** required string: the refresh token
* **redirect\_uri** required string: redirect uri, given to the auth service

The grant type should be set to “refresh\_token” when renewing access tokens with a refresh token.

As this request always should be initiated by the client backend, the client id and client secret are available, and must be added to the request.

The refresh token is delivered together with the access token, if the scope “offline\_access” was used as in the initial authentication.

**Response (success)**

**Status code:** 201 Created

**Content-type:** (matching the Accept header from the request)

**Body:**

* access\_token
* token\_type
* expires\_in

|  |  |  |  |
| --- | --- | --- | --- |
| access\_token | string | 0 .. \* chars | Base64-encoded signed token including the access rights granted to the client and the expiry time |
| refresh\_token | string | 0 .. \* chars | Base64-encoded signed token that can be used to refresh the access token again |
| token\_type | string | 0 .. \* chars | Token type. Returns “bearer” in current implementation. |
| expires\_in | integer | 0 .. 231-1 | The lifespan of the token, in seconds. |

**Example response (JSON)**

**{**

**"access\_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnQiOiJkaHAiLCJ0eXBlIjoiY2xpZW50IiwiZXhwIjoxNDc0OTU5MzY0fQ.Jy2OIsybFafAhDnPAu7RTUtoUEFAy-KabaBuZDp94Sb8xIVFKXz1igDiQ1V5gFSKibPEblyNamb5YftKabhHXwJYI\_fcai7JDKzrMXXauvmys7DuxZA9gvS9HgWrvbqVBVR\_s1EOh-q5d5RVLtlR-0Tl4WIsZaG931XXh2UKPJbjPW5uC4rJAHhhil6vqgrNWFcS2vbN\_TYKVBzXVuYW0-rEI\_33NTVJ0WXTeH8t9oMlntRWRY381EpDnMgjqTLUgoKT4rt9Wer7Hw7ltxTSIIU3IWBndF6kIFZEd5vt2KvQL1E3EbOn0IZgQks4\_SxCMgoSu2Dtj1bB1yBvUrofbw",**

**"token\_type": "bearer",**

**"expires\_in": 1800**

**}**

### OpenID Connect

Animalia SSO has basic support for OpenID Connect. Retrieve the configuration from the following service endpoints:

**Staging environment:** [**https://staging-sso.animalia.no/.well-known/openid-configuration**](https://staging-sso.animalia.no/.well-known/openid-configuration)

**Production environment:** [**https://sso.animalia.no/.well-known/openid-configuration**](https://sso.animalia.no/.well-known/openid-configuration)

### Scopes

## Animalia SSO accepts the following general scopes:

|  |  |
| --- | --- |
| openid | Use this scope in order to receive an OpenID Connect compliant ID token together with the access token. |
| offline\_access | This scope enables the use of refresh tokens to renew the access token directly from the client. |
| profile | Adds the name of the user, if available, to the ID token (requires the openid scope) |
| email | Adds the email address of the user, if available, to the ID token (requires the openid scope) |
| address | Adds the postal address of the user including street address, postal code and country, if available, to the ID token (requires the openid scope) |
| phone | Adds the phone number of the user, if available, to the ID token (requires the openid scope) |
| pid | Adds the user PID number from Landbrukets Dataflyt, if available, to the ID token (requires the openid scope) |

## Client credentials (system-to-system)

This grant is only to be used in cases where there is no user interaction involved at all, and the communication is strictly initiated by an external system and does not by default involve data from any one particular user.

The following endpoint is used to exchange a client id and a client secret for an access token lasting 30 minutes.

Please note that the production and staging environments do not share client secrets. If you have a client id and secret for staging, please contact your representative in Animalia or Animalia IT at [*itdrift@animalia.no*](mailto:itdrift@animalia.no) for access to the production environment.

### Request (Animalia SSO token service)

* **Staging environment:** [**https://staging-sso.animalia.no/token**](https://staging-sso.animalia.no/token)
* **Production environment:** [**https://sso.animalia.no/token**](https://sso.animalia.no/token)

**Allowed-methods:** POST

**Accept:**  application/json, application/edn

**Parameters:**

* **grant\_type** required string: “client\_credentials”
* **client\_id** requiredstring: the client id
* **client\_secret** required string: the client secret (UUIDv4)

**Response (success)**

**Status code:** 201 Created

**Content-type:** (matching the Accept header from the request)

**Body:**

* access\_token
* token\_type
* expires\_in

|  |  |  |  |
| --- | --- | --- | --- |
| access\_token | string | 0 .. \* chars | Base64-encoded signed token including the access rights granted to the client and the expiry time |
| token\_type | string | 0 .. \* chars | Token type. Returns “bearer” in current implementation. |
| expires\_in | integer | 0 .. 231-1 | The lifespan of the token, in seconds. |

### 

**Example response (JSON)**

**{**

**"access\_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnQiOiJkaHAiLCJ0eXBlIjoiY2xpZW50IiwiZXhwIjoxNDc0OTU5MzY0fQ.Jy2OIsybFafAhDnPAu7RTUtoUEFAy-KabaBuZDp94Sb8xIVFKXz1igDiQ1V5gFSKibPEblyNamb5YftKabhHXwJYI\_fcai7JDKzrMXXauvmys7DuxZA9gvS9HgWrvbqVBVR\_s1EOh-q5d5RVLtlR-0Tl4WIsZaG931XXh2UKPJbjPW5uC4rJAHhhil6vqgrNWFcS2vbN\_TYKVBzXVuYW0-rEI\_33NTVJ0WXTeH8t9oMlntRWRY381EpDnMgjqTLUgoKT4rt9Wer7Hw7ltxTSIIU3IWBndF6kIFZEd5vt2KvQL1E3EbOn0IZgQks4\_SxCMgoSu2Dtj1bB1yBvUrofbw",**

**"token\_type": "bearer",**

**"expires\_in": 1800**

**}**

## Responses (errors)

**Status code:** 401 Unauhorized

**Content-type:** (matching the Accept header from the request)

**Body:** error: (feilkode)

The client id, client secret or grant type did not match.

**Status code:** 404 Not Found

**Content-type:** (matching the Accept header from the request)

**Body:** error: Tjenesten du forsøker å aksessere, finnes ikke

The URL given did not match any resource

**Status code:** 406 Not Acceptable

**Content-type:** text/plain

**Body:** (List of valid content types)

The client asked for a content type in the Accept header that does not match any of our implemented response content types.

**Status code:** 500 Server Error

**Content-type:** (matching the Accept header from the request)

**Body:** (feilmelding)

In a wonderful, perfect world, there would be no errors. The truth is that the world is not perfect, and neither are our services. If you ever encounter an error in our service, please tell us, and we will fix it as soon as possible!

**Status code:** 502 Bad Gateway

**Content-type:** text/html

**Body:** (generic HTML error page from our load balancer)

If something is wrong in our service platform, the load balancer will report this as a bad gateway.

**Status code:** 503 Service Unavailable

**Content-type:** text/html

**Body:** (generic HTML error page from our service platform)

If the Animalia SSO or Animalia Authentication services are down, the service platform will respond with 503.

## Verifying the access token

If you want to verify the integrity of tokens issued by Animalia SSO to ensure they have not been manipulated, you need to verify the signature of the token.

Every token has a key identification (KID) in the header section of the token:

**{**

**"alg": "RS256",**

**"kid": "animalia\_sso\_0000\_00\_00\_0"**

**}**

**...**

In this example the KID is *“animalia\_sso\_0000\_00\_00\_0”*, which identifies the key used to sign this token. The corresponding public key can be used to verify its integrity.

All active public keys for Animalia SSO are available at this service endpoint:

### Request (Animalia SSO public key endpoint)

* **Staging environment:** [**https://staging-sso.animalia.no/keys**](https://staging-sso.animalia.no/keys)
* **Production environment:** [**https://sso.animalia.no/keys**](https://sso.animalia.no/keys)

**Allowed-methods:** GET

**Accept:**  application/json

**Response (success)**

**Status code:** 200 OK

**Content-type:** (matching the Accept header from the request)

**Headers:**

* Cache-Control: max-age=?

**Body:**

* keys

|  |  |  |  |
| --- | --- | --- | --- |
| keys | list | 0 .. \* JWK-objects | A list of all active public keys in the [JWK format](https://tools.ietf.org/html/rfc7517). |

### 

The keys are delivered according to the [RFC 7517](https://tools.ietf.org/html/rfc7517) definition of the JWK format, and should be compatible with standard JWK parsers. Each JWK has a KID that should match the KID in the access token that was signed with the corresponding private key. Use that key to verify the access token integrity.

Please note that calls to this service could and should be cached for the amount of time defined by the Cache-Control header max-age value. All tokens issued within this duration should always have a corresponding public key in the cached response.