connecting customers

ANVA

Creating ANVA API based Customer Apps

Partner Documentation

Versie 1.14
Wijzigingsdatum 10-06-2024

Table of Contents

Aims

This document provides insights to third party partners and integrators to integrate the ANVA APIs to their customer centric custom applications. For authentication and Authorization of users and proper access of the designated endpoints- the partners and integrators are expected to follow the prescribed ANVA OpenID flows- and this document also provides the required information for the partners to integrate the ANVA OpenID flows into their apps.

Integration prerequisites

Before starting the process of integration with the ANVA APIs, it is necessary to have the following processes completed:

Registering a New OpenID client in ANVA Hub

All access to ANVA APIs is restricted only to registered valid client applications. It's necessary for the customer to create a ticket for ANVA Customer Support with an OpenID request. A Customer Support employee then can register (create) a new client in the hub platform for the customer.

Once the OpenID client is registered, the hub platform would display a ***Client ID*** and ***Client Secret*** for the newly created OpenID client. These two values will be required in the implementation along with the provided redirect URLs. For security, you will receive this information directly from the customer and not from an ANVA employee. It is recommended to immediately reset the Client Secret upon receiving it for security reasons, ensuring that it is only known by the customer.

Authentication and Authorization

The Authentication and Authorization follows the OpenID standards and on successful completion of the authentication process, a valid ID Token is returned to the provided redirect URL in the request. For Partner app clients- the recommendation is to use the Authorization Code Flow.

The Authorization Request

The Authorization request can be made at the designated authorization endpoint.

Endpoint : ***/identity/authorize***
Method : **GET**
Query Parameters : The following parameters are to be passed in the Query String

1. **client_id**- The ***Client ID*** of the registered client
2. **redirect_uri**- A valid redirect URL associated the client
3. **response_type**- use ***code*** for authorization code Flow
4. **scope**- use ***openid Customer*** for client apps

5. **state**- any custom value that needs to be fetched back in the response
6. **nonce-** any custom value that needs to be present as a claim in JWT.
7. **max_age-** an optional value to specify the longevity of the generated token in seconds
8. **response_mode -** an optional value to specify the method that should be used to send the resulting Authorization Endpoint Response. Use response_mode as 'query' for encoding Authorization Response parameters in the query string or Use response_mode as 'fragment' for encoding Authorization Response parameters in the fragment string.
9. **prompt-** It is an optional parameter.

   Defined values in Prompt Parameter are:

   - **none-** The Authorization Server MUST NOT display any authentication or consent user interface pages. An error is returned if an End-User is not already authenticated or the Client does not have pre-configured consent for the requested Claims or does not fulfil other conditions for processing the request.
   - **login-** The Authorization Server SHOULD redirect the end-user to the login page. If it cannot reauthenticate the End-User, it MUST return an error, typically **login_required.**
   - **consent-** The Authorization Server SHOULD redirect the end-user to the consent page. If it cannot obtain consent, it MUST return an error, typically **consent_required.**

   login and **consent** can be used together as a prompt parameter. **none** can not be used with other values, otherwise an error is returned.

NOTE : If ***openid*** is not passed as a scope, the system will not return an ID Token at the end of a successful Authorization process, but will only return OAuth 2.0 compliant access tokens.

Example Requests

For ***Authorization Code Flow*** an example request would be:

```
/identity/authorize?response_type=code
                    &scope=openid Customer
                    &client_id=<Your_Client_ID>
                    &state=test_state
                    &redirect_uri=<Your_Redirect_URL>
                    &nonce=<Your_Nonce_String>
                    &max_age=<Your_Desired_Longivity_Of_Token>
                    &response_mode=<Your_Desired_Response_Mode>
                    &prompt=<Your_Desired_Prompt_Value>
```

The Authorization Process

Every successful request to the Authorization endpoint redirects the calling client's browser to present the login page - where the customer user has to login with the provided account credentials (refer to creation on customer accounts section in the prerequisites). After the user credentials are validated - a change password screen is presented (as all customer accounts are assigned a temporary system generated password) , where the user has to enter the assigned password and the desired new password and save the new password. Once that is done and validated by the system, the user is again sent to the login page to login with the changed password. Once the login is successful, the user is presented with a consent screen informing the user about the user information that will be passed on from the ANVA Hub Platform to the client app and asks the user to provide a consent or decline. Once the user gives consent, the response is redirected to the redirect URL provided in the Authorization request. The response contains an Authorization code.

The Authorization Response

Authorization Code Flow (response_type= code)

The Authorization code flow response returns an Authorization Code along with the scope value passed in the request. A sample response is as follows if no response_mode is specified in the Authorize Request (i.e default as" **query**"):-

```
<Redirect_URL>?code=<Your_Authorization_Code>
               &state=<Your_Original_State_value>
```

The response is as follows if response_mode as **"fragment"** is specified in the Authorize Request

```
<Redirect_URL>?code=<Your_Authorization_Code>
               &state=<Your_Original_State_value>
```

It is important that the users use the *access_token* for accessing resources and not the *id_token*. Only use the id_token value to get information about the authenticated user.

Getting the token from the Authorization Code (Authorization Code Flow)

The Authorization response for Authorization code flow returns an Authorization code. This authorization code has to be used to fetch the ID token. This can be done by making a request to the token endpoint as follows:

Endpoint : ***/identity/token***
Method : **POST**
Headers : The following request headers are to be added:

1. **Authorization**: Basic Authorization by creating a Base64 encoded string of the Client ID and Client Secret in the format ***<Your_Client_ID>:<Your_Client_Secret>***. The

Authorization header value should be in the following format to be valid
`'Authorization': 'Basic <Your_Base64_Code>'`

2. **Content Type** : should be ***application/x-www-form-urlencoded***. The header should be
`'Content-Type': 'application/x-www-form-urlencoded'`

Request Body : The following parameters are to be passed in the Request Body:

1. **grant_type** - should be set to ***authorization_code***
2. **redirect_uri** - A valid redirect URL associated the client
3. **code** - The ***Authorization code*** in the Authorization response
4. **client_id** - The **Client Id** for the Client. [If not in the Authorization Header.]
5. **client_secret** - The **Client Secret** for the client. [If not in the Authorization Header.]

In response to a valid token request with the proper authorization code, the token endpoint returns an **ID Token** to the provided redirect URL similar to the response and **refresh token** that is used to generate a new access token.

NOTE : The ***Authorization code*** is only valid for 10 mins only from consent.

Refresh Token

The Refresh Token grant type is used by clients to exchange a refresh token for an access token when the access token has expired.

The Refresh Token Request

To refresh an Access Token, the OpenID Client must authenticate to the Token Endpoint using the authentication method.

Endpoint: /identity/token
Method: POST
Headers:
The following request headers are to be added:

1. **Authorization:** Basic Authorization by creating a Base64 encoded string of the OpenID Client ID and OpenID Client Secret in the format ***<Your_Client_ID>:<Your_Client_Secret>*** The Authorization header value should be in the following format to be valid `'Authorization': 'Basic <Your_Base64_Code>'`
2. **Content Type :** should be **application/x-www-form-urlencoded**. The header should be
`'Content-Type': 'application/x-www-form-urlencoded'`

Request Body: The following parameters are to be passed in the Request Body :

1. **grant_type** - should be set to **refresh_token**
2. **refresh_token** - Same as the value of refresh token during last generated access token.

3. **client_id** - The OpenID **Client ID** for the Client. [If not in the Authorization Header.]
4. **client_secret** - The OpenID **Client Secret** for the client. [If not in the Authorization Header.]

Successful Refresh Token Response

For every valid request to the Token endpoint, the identity component issues an access token (id_token) and refresh_token along with token_type and expires_in parameters in the response body.

The response is as follows :

Headers: The following response header fields are to be added:

1. **Content-Type :** Response body content should be in 'application/json' format, with a character encoding of UTF-8.

```
'Content-Type': 'application/json;charset=UTF-8'
```

2. **Cache-Control :** HTTP 'Cache-Control' response header field, with a value of 'no-store'.

```
'Cache-Control': 'no-store'
```

3. **Pragma :** HTTP 'Pragma' response header field, with a value of 'no-cache'.

```
'Pragma': 'no-cache'
```

Response Body: The following parameters are passed in the Response Body :

1. **id_token -** ID Token value associated with the authenticated session.
2. **access_token -** The access token issued by the authorization server.
3. **token_type -** The type of the token as Bearer.
4. **refresh_token -** The refresh token issued by the authorization server every time an access_token is requested.. This refresh token can be used to generate a new access token when the previous access_token has expired.
5. **expires_in -** expiry time of the ID token

Users are required to use the *access_token* for accessing resources from the server.

NOTE : Users will get a new refresh token each time a new access token is requested. Once the refresh token is used it is invalidated.

Client Credentials Flow

The Client Credentials Flow follows OAuth 2.0 standards and on successful completion of the client authentication process, a valid Access Token is returned to the client. In ANVA Identity Component, Client Credentials Flow is used by Internal Clients.

The Access Token Request

The Access Token request can be made at the designated token endpoint.

Endpoint: *identity/token*
Method: POST
Headers: The following request headers are to be added:

1. **Authorization**: Basic Authorization by creating a Base64 encoded string of the OpenID Client ID and OpenID Client Secret in the format ***<Your_Client_ID>:<Your_Client_Secret>***. The Authorization header value should be in the following format to be valid `'Authorization': 'Basic <Your_Base64_Code>'`
2. **Content Type**: Should be ***application/x-www-form-urlencoded***. The header should be `'Content-Type': 'application/x-www-form-urlencoded'`

Request Body: The following parameters are to be passed in the Request Body :

1. **grant_type** - Should be ***client_credentials*** `'grant_type': 'client_credentials'`
2. **client_id** - The OpenID **Client Id** for the Client. [If not in the Authorization Header.]
3. **client_secret** - The OpenID **Client Secret** for the client. [If not in the Authorization Header.]
4. **scope or scopes** - The scopes of the access request along with organisation code as **orgCode:<orgCode>** or organisation GUID as **orgId:<orgGUID>**; optionally, on behalf of username field to be added for adding the username in the access token as **onBehalfOfUsername:<username>.** The use of scopes parameter will be replaced by the scope in the near future.
   `'scope': 'Basic orgCode:<orgCode>/ orgId:<orgGUID> {either orgCode or orgId to be used} onBehalfOfUsername:<username>'`

In response to a valid token request with the proper grant type and scope, the token endpoint returns an Access Token to the client.

The Client Authentication Process

The OpenID Client retrieved from the encoded header is validated against the database. Once that is successful, the following validations are done with the fetched client.

- Once a valid OpenID Client is fetched the OpenID Client Secret is matched.
- If both the above steps are successful then it is checked if the scopes specified are present with the client.
- And finally it is checked if the client has access to the requested organisation.

The Token Response

Once the client authentication is successful. The process of generating the token begins. The required claims are put in the token. The type of the token is “System”. The scope claim contains the intersection of the scopes provided and the scopes available with the client. Then finally the token is sent to the user via the response body.

Successful Token Response

For every valid request to the Token endpoint, the identity component issues an access token (id_token) along with token_type and expires_in parameters in the response body. The response is as follows :

Headers : The following response header fields are to be added:

1. Content-Type: Response body content should be in ‘application/json’ format, with a character encoding of UTF-8.

```
'Content-Type': 'application/json;charset=UTF-8'
```

2. Cache-Control: HTTP ‘Cache-Control’ response header field, with a value of ‘no-store’.

```
'Cache-Control': 'no-store'
```

3. Pragma : HTTP ‘Pragma’ response header field, with a value of ‘no-cache’.

```
'Pragma': 'no-cache'
```

Response Body: The following parameters are passed in the Response Body:

1. **access_token** - The access token issued by the authorization server
2. **token_type** - The type of the token as ***Bearer***.
3. **expires_in** - Lifetime in seconds of the access token.

Token Error Response
If the token request fails client authentication or is invalid, the authorization server returns an error response.

Headers: The following response header fields are added:

1. **Content-Type** : Response body content should be in 'application/json' format, with a character encoding of UTF-8.

```
'Content-Type': 'application/json;charset=UTF-8'
```

2. **Cache-Control** : HTTP 'Cache-Control' response header field, with a value of 'no-store'.

```
'Cache-Control': 'no-store'
```

3. **Pragma** : HTTP 'Pragma' response header field, with a value of 'no-cache'.

```
'Pragma': 'no-cache'
```

Response Body : The following parameters are passed in the Response Body :

1. **error**- A single error code. For details, check the Error Messages table below.

Error Messages For Client Credentials Flow

Endpoint	Error	Details
/identity/token	invalid_grant	The OpenID Client does not support client credentials flow.
/identity/token	invalid_scopes	The OpenID client does not have access to one or more of the scopes specified in the request.
/identity/token	invalid_organization	The provided organisation code does not exist for this OpenID client.
/identity/token	invalid_client	The given OpenID Client ID does not exist.

Using the token for API access

Once the OpenID client receives an ID Token in JWT format, it can be used to access the Customer API endpoints, by passing it in the Request Headers in the following way:

```
'Authorization': 'Bearer <Your_JWT_Token>'
```

Requests without the Authorization header will be considered as 12nauthorized access requests and will get an 12nauthorized access error response.

Error Messages

All API requests are validated to check if all the required input parameters have been provided. In case of an error, the API returns the following error messages

Endpoint	Error Code	Details
`/identity/authorize`	invalid_scope	The OpenID Client does not have one or scopes specified in the request.
`/identity/authorize`	invalid_client	The given OpenID Client ID does not exist.
`/identity/authorize`	request_uri_not_supported	When the specified request uri is not associated with the OpenID client.
`/identity/token`	invalid_grant	The OpenID client does not support auth code flow.
`/identity/token`	invalid_code	If the *code* has been used or is invalid.
`<redirect uri>?error=`	client_scopes_does_not_match_with_the_user_scopes	The scopes of the user and the scopes requested during authorization have none in common.

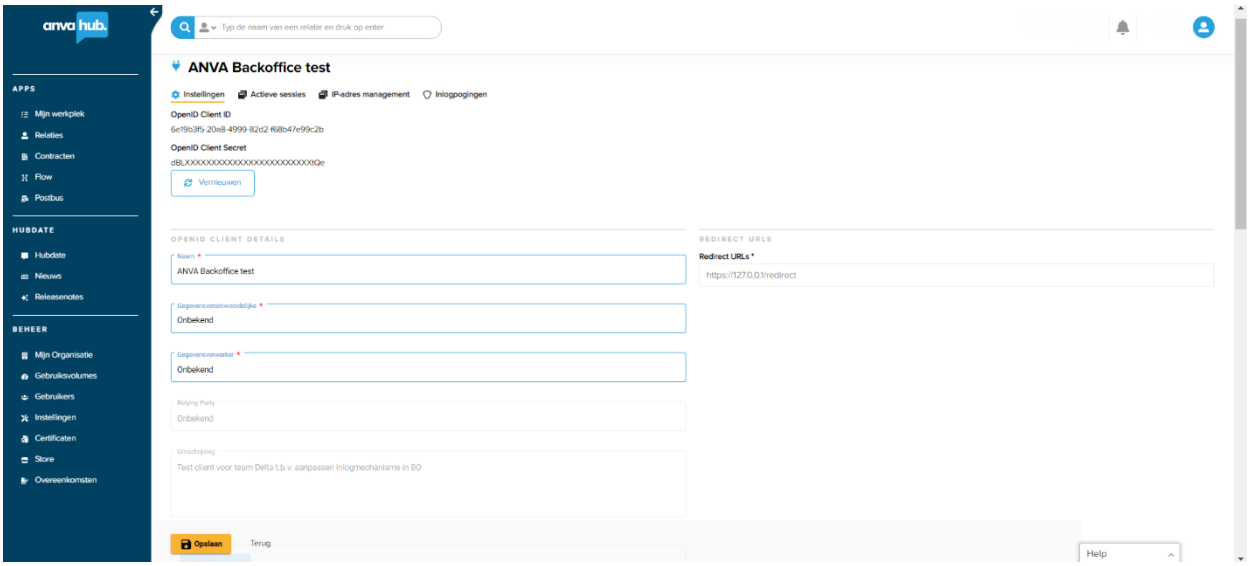
Hub Activities

As discussed earlier in the Integration Prerequisites section- the ANVA Hub Platform has to be used for Creation of the Clients and Customer Accounts. This section gives a detailed overview of the process of these activities.

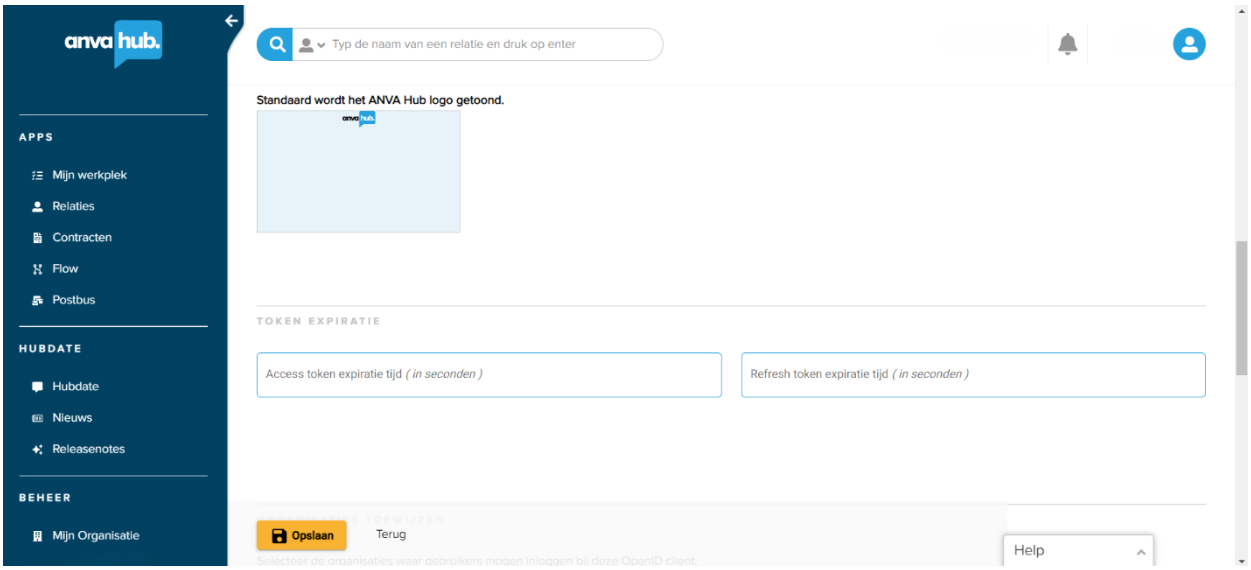
Client Creation and tokens

As mentioned earlier in this document, only an ANVA Customer Support employee can create an OpenID Client after they receive an OpenID request through a customer's ticket.

Once an OpenID Client is created, newly created client information shows here.



When you scroll down, you can see two fields: Access Token Expiry Time (Optional field) and Refresh Token Expiry Time (Optional field). These two fields can customize the lifetime access of tokens. The values should be entered in seconds.



For Example- Access Token Expiry Time is 1200 (in seconds) then the maximum expiry time of access token will be 1200 (20 mins).
The range of Access Token Expiry Time - **900 (15 mins) to 36,000 (10 hrs)**
The range of RefreshToken Expiry Time - **900 (15 mins) to 3,153,600 (365 days)**

If no value is passed in these two fields then **by default** Access Token Expiry Time will be **36,000 (10 hrs)** and Refresh Token Expiry Time will be **36,600 (10 hrs 10 mins).**

Discovery (Well-Known) Endpoint

OpenID Connect defines a discovery mechanism, called OpenID Connect Discovery, where an OpenID server publishes its metadata at a "well-known" URL. This URL returns a JSON listing of the OpenID/OAuth endpoints, supported scopes and claims, keys used to sign the tokens, and other details. The clients can use this information to construct a request to the OpenID server, i.e., the Identity component.

OpenID Provider Configuration Request

An OpenID Provider Configuration endpoint MUST be queried using an HTTP GET request.

Endpoint : *`/identity/.well-known/openid-configuration`*
Method : GET

Successful OpenID Provider Configuration Response

A successful response MUST use the 200 OK HTTP status code and return a JSON object using the application/json content type that contains a set of Claims as its members that are a subset of the Metadata.

Response Body : The following parameters are passed in the Response Body-

1. **issuer** - The URL that the OpenID Provider, i.e., the Identity component asserts as its Issuer Identifier. (*https://api.anva.live/identity*)
2. **authorization_endpoint**- The URL of the Identity component's OAuth 2.0 Authorization Endpoint. (*https://api.anva.live/identity/authorize*)
3. **token_endpoint**- The URL of the Identity component's OAuth 2.0 Token Endpoint. (*https://api.anva.live/identity/token*)
4. **Userinfo_endpoint** - The URL of the Identity component's UserInfo Endpoint. (*https://api.anva.live/identity/userinfo*)
5. **jwks_uri**- The URL of the Identity component's JSON Web Key Set [JWK] document. (*https://api.anva.live/identity/.well-known/jwks*)
6. **scopes_supported**- JSON array containing a list of the OAuth 2.0 scope values that the Identity component supports. (*openid, profile, email, Basic, Customer*)

7. **response_types_supported**- JSON array containing a list of the OAuth 2.0 response_type values that the Identity component supports. (*code and token*)
8. **response_modes_supported**- JSON array containing a list of the OAuth 2.0 response_mode values that the Identity component supports. (*query*)
9. **grant_types_supported**- JSON array containing a list of the OAuth 2.0 Grant Type values that the Identity component supports. (*authorization_code and client_credentials)*
10. **subject_types_supported**- JSON array containing a list of the Subject Identifier types that the Identity component supports. (*public*)
11. **id_token_signing_alg_values_supported** - JSON array containing a list of the JWS signing algorithms (alg values) supported by the Identity component for the ID Token to encode the Claims in a JWT [JWT]. (*RS256*)
12. **token_endpoint_auth_methods_supported**- JSON array containing a list of Client Authentication methods supported by the Token Endpoint. (*client_secret_basic*)
13. **token_endpoint_auth_signing_alg_values_supported**- JSON array containing a list of the JWS signing algorithms (alg values) supported by the Token Endpoint for the signature on the JWT [JWT] used to authenticate the Client at the Token Endpoint for the private_key_jwt and client_secret_jwt authentication methods. (*RS256*)
14. **claim_types_supported**- JSON array containing a list of the Claim Types that the Identity component supports. (*normal*)
15. **claims_supported**- JSON array containing a list of the Claim Names of the Claims that the Identity component shall be able to supply values for. (*sub, iss, aud, jti, iat, exp, nonce*)

Sample Successful Response :-


```
{
  "issuer": "http://localhost:8129/identity",
  "authorization_endpoint": "http://localhost:8129/identity/authorize",
  "token_endpoint": "http://localhost:8129/identity/token",
  "userinfo_endpoint": "http://localhost:8129/identity/userinfo",
  "jwks_uri": "http://localhost:8129/identity/.well-known/jwks",
  "scopes_supported": [
    "openid",
    "Profile",
    "Email",
    "Basic",
    "Customer"
  ],
  "response_types_supported": [
    "code",
    "token",
    "id_token",
```

```
        “id_token token”
    ],
    "response_modes_supported": [
        "query"
    ],
    "grant_types_supported": [
        "authorization_code",
        "client_credentials",
        “refresh_token”

    ],
    "subject_types_supported": [
        "public"
    ],
    "id_token_signing_alg_values_supported": [
        "RS256"
    ],
    "token_endpoint_auth_methods_supported": [
        "client_secret_basic",
        “client_secret_post”
    ],
    "token_endpoint_auth_signing_alg_values_supported": [
        "RS256"
    ],
    "claim_types_supported": [
        "normal"
    ],
    "claims_supported": [
        "iss",
        "sub",
        "aud",
        "jti",
        "iat",
        "exp",
        "nonce",
        “auth_time”,
        “at_hash”
    ]
}
```

OpenID Provider Configuration Error Response
An error response uses the 404 Not Found HTTP status code value.

JWKS URI Endpoint

This endpoint renders the Identity component's JSON Web Key Set [JWK] document. This contains the signing key(s) the Relying Party uses to validate signatures from the Identity component.

JWKS Request

An OpenID Provider JWKS URI endpoint MUST be queried using an HTTP GET request.

Endpoint: *identity/.well-known/jwks*
Method: GET

Successful JWKS Response

A successful response MUST use the 200 OK HTTP status code and return a JSON object using the application/json content type that contains a set of Claims as its members that are a subset of the Metadata.

Response Body : The following parameters are passed in the Response Body-

1. **keys**- The value of the "keys" parameter is an array of JWK values.

i. **kty**- Identifies the cryptographic algorithm family used with the key. (*RSA*)
ii. **use**- Identifies the intended use of the public key. (*sig*)

iii. **alg**- Identifies the algorithm intended for use with the key.(*RS256*)
iv. **n**- Modulus of the public key.
v. **e**- Exponent of the public key

Sample Successful Response :-

```
{
   "keys": [
      {
         "kty": "RSA",
         "use": "sig",
         "alg": "RSA",
         "n":  <Public_Key_Modulus>,
         "e": <Public_Key_Exponent>
      }
   ]
}
```

JWKS Error Response
An error response uses the 404 Not Found HTTP status code value.

UserInfo Endpoint

The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User.

To obtain the requested Claims about the End-User, the Client makes a request to the UserInfo Endpoint using an Access Token obtained through OpenID Connect Authentication. These Claims are normally represented by a JSON object that contains a collection of name and value pairs for the Claims.

The UserInfo Endpoint MUST accept Access Tokens as **Bearer** tokens.

UserInfo Request

An OpenID Provider UserInfo endpoint MUST be queried using an HTTP GET request.

Endpoint: *identity/userinfo*
Method: GET
Header: Authorization: Bearer <id_token>
Permission Required: 1. openid (Must) 2. Profile/Email

Successful UserInfo Response

A successful response MUST use the 200 OK HTTP status code and return a JSON object using the application/json content type that contains a set of Claims about the Authenticated End-User.

Response Body:

1. **With Both Profile and Email Scopes in the access_token**
   The following parameters are passed in the Response Body-
   a. **sub: The account_id of the end user.**
   b. **name: Full_Name of the end user.**
   c. **given_name: First_name of the end user**
   d. **family_name: Last_name of the end user.**
   e. **middle_name: Middle_name of the end user.**
   f. **preferred _username: anva_username**
   g. **email: email of the end User**

h. **email_verified: true , as Anva Always validate the email_address before User Account Creation**
i. **updated_at: last time the end user account is updated.**

Sample Response:


```
{
  "sub": "account_id_of_End_user",
  "name": "full_name",
  "given_name": "first_name",
  "family_name": "last_name",
  "middle_name": "middle_name",
  "preferred_username": "anva_username",
  "email": "email_of_the_end_user",
  "email_verified": true,
  "updated_at": "last_time_end_user_updated"
}
```


2. **With Profile Scope Only in the access_token**

The following parameters are passed in the Response Body:

a. **sub: The account_id of the end user.**
b. **name: Full_Name of the end user.**
c. **given_name: First_name of the end user**
d. **family_name: Last_name of the end user.**
e. **middle_name: middle_name**
f. **preffered_username: anva_username**
g. **updated_at: last time the end user account is updated.**

Sample Response:


```
{
  "sub": "account_id_of_End_user",
  "name": "full_name",
  "given_name": "first_name",
  "family_name": "last_name",
  "middle_name": "middle_name",
  "preferred_username": "anva_username",
  "updated_at": "last_time_end_user_updated"
}
```

3. **With Email Scope Only in the access_token**
   The following parameters are passed in the Response Body-
   a. **sub: The account_id of the end user.**
   b. **email: email of the end User**
   c. **email_verified: true , as Anva Always validates the email_address before User Account Creation.**
   d. **updated_at: last time the end user account is updated.**

Sample Response:


```
{
   "sub “": "account_id_of_End_user",
   "email": "email_of_the_end_user",
   "email_verified": true,
   "updated_at": "last_time_end_user_updated"
}
```


UserInfo Error Response

An error response uses the 404 Not Found HTTP status code value.

error	errorCode	error_Description
HTTP/1.1 401 Unauthorized WWW-Authenticate: error="invalid_token", error_description="The Access Token expired		
invalid_request	400 (Bad Request)	The request is missing a required parameter, includes an unsupported parameter or parameter value, repeats the same parameter, uses more than one method for including an access token, or is otherwise malformed.
invalid_token	401 (Unauthorized)	The access token provided is expired, revoked, malformed, or invalid for other reasons.
insufficient_scope	403 (Forbidden)	The request requires higher privileges than provided by the access token.

JWT Token Signing

The JWT token is signed using a private key.

The signature can be verified by using the corresponding public key. The private key is unique to each domain (.live, .me)

To verify the signature, just use the key from the ***keys*** array of the jwks uri response, which has the use value as "***sig".***