N26 - PSD2 Dedicated Interface - Sandbox Access documentation

General information

Our sandbox environment provides the means by which to perform limited testing of our API calls without requiring real user accounts. (e.g. to test formatting, errors etc)

Endpoints and interactions are as described in the general API documentation, except where otherwise discussed here.

This is a minimum viable product; we value feedback and suggestions for improvements. We don't currently recommend to go directly to a full public rollout after testing with the sandbox alone, although it should be a useful starting point.

Access & Identification of TPP

Sandbox URL

```
https://xs2a.tech26.de/sandbox
```

On-boarding of new TPPs

This is as described for the real interface, and all valid QWAC certificates are accepted. If you are facing issues accessing this interface using a test certificate, please reach out to us with the following:

- The certificate used to access the interface
- Details of the request and response received

OAuth as a Pre-step

This may be simulated as follows:

1. Make a request to /sandbox/oauth2/authorize providing a redirectUrl and a hashed code verifier.
2. Make a request to /sandbox/psu-interaction/generate-auth-code to generate an authorisation code (this simulates the user interaction).
3. Make a request to /sandbox/oauth2/token with the auth code and the code verifier to get an access token. In reality, the auth code should be appended to your URL by a redirect from N26, but we do not cover that case here.

Initiate authorization

This begins the authorization process. In reality, users should be redirected to the URL supplied in the response, but we provide a dummy URL for the time being.

Sample request

```
GET /sandbox/oauth2/authorize?client_id=PSDDE-BAFIN-000001&
                      scope=DEDICATED_AISP&
                      code_challenge=w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI&
                      redirect_uri=https://tpp.com/redirect&
                      response_type=CODE&
                      state=1fL1nn7m9a
HTTP/1.1
```

Sample Response

```
HTTP/1.1 302 Found
location: https://app.n26.com/open-banking?requestId=0daa152a-651a-4592-8542-47ff60799deb&state=1fL1nn7m9a&authType=XS2A
```

Generate authorization code

For the sandbox, this endpoint replaces the following user interaction on the real API:

1. The TPP redirects the PSU to the URL returned by the authorize endpoint.
2. We communicate directly with the PSU to give them an authorization code.
3. The PSU redirects back to the TPP, providing the authorization code.

Because we don't have equivalent UI for the sandbox (and because it's easier for automated testing), we replace this entire flow with a single call to the following endpoint, which returns an authorisation code directly without requiring a redirect to and from N26.

Sample request

```
POST /sandbox/psu-interaction/generate-auth-code/{requestId}?role=DEDICATED_AISP
HTTP/1.1
```

Sample Response

```
{
        "code": "3a4918e7-9277-4df6-a56f-fbd2b32fbacb"
}
```

Retrieve Token

Sample Request

```
POST    /sandbox/oauth2/token?role=DEDICATED_AISP HTTP/1.1
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=dbtF5AqOApjjSnNF5TK3w3gaEPdwtV2&
code_verifier=foobar&
redirect_uri=https://tpp.com/redirect
```

Response

Successful

```
HTTP/1.1 200 OK
{
    "access_token": "{{access_token}}",
    "token_type": "bearer",
    "refresh_token": "{{refresh_token}}",
    "expires_in": {{expires_in_seconds}}
}
```

Consent endpoints

Consent interactions are the same as for the real API, except that some edge cases may not be covered identically. We also have an additional endpoint allowing TPPs to simulate the SCA flow (described below).

All endpoints are prefixed with `/sandbox` compared to the live system, so for example for the root URL for consents would be:

`https://xs2a.tech26.de/sandbox/v1/berlin-group/v1/consents`

SCA user interaction

As described in our AISP documentation, we currently use the `DECOUPLED` SCA flow to establish consent, which means that the normal sequence of events from a TPP perspective is:

1. POST /consents to create the consent
2. Instruct the user to confirm with N26 directly (in our mobile app).
3. In the meantime, poll the consent status

For the sandbox, step 2 is replaced by an endpoint which simulates the PSU interaction without requiring UI, similar to what is done for the authorisation code above, so the flow changes to:

1. POST /consents to create the consent
2. POST /psu-interaction/consents/{consentId}/scas to simulate SCA
3. In the meantime, poll the consent status

You can pass “APPROVED” or “REJECTED” to this endpoint, which is equivalent to the user confirming or cancelling in the N26 app. The consent status will update accordingly.

Sample request

```
POST /sandbox/psu-interaction/consents/{consentId}/scas
HTTP/1.1
{
        "result": "APPROVED"
}
```

Sample Response

```
HTTP/1.1 204 No Content
```

AIS endpoints

These endpoints currently respond with dummy data. You can expect requests to be validated for correct syntax and valid responses to be returned.

All endpoints are prefixed with `/sandbox` compared to the live system, so for example for the root URL for accounts would be:

`https://xs2a.tech26.de/sandbox/v1/berlin-group/v1/accounts`

PIS endpoints

These endpoints currently respond with dummy data. You can expect requests to be validated for correct syntax and valid responses to be returned.

All endpoints are prefixed with `/sandbox` compared to the live system, so for example for the root URL for payments would be:

`https://xs2a.tech26.de/sandbox/v1/berlin-group/v1/payments`

Get payment status

As described in our PISP documentation, we currently use the `DECOUPLED` SCA flow to confirm payment initiation, which means that the normal sequence of events from a TPP perspective is:

1. POST /payments to create the payment
2. Instruct the user to confirm with N26 directly (in our mobile app).
3. In the meantime, poll the payment status

For the PISP sandbox, these endpoints are currently stateless, which means the API cannot be tested in exactly the same way we do it for AIS consents. Instead - by default - as soon as you query payment status you will receive that the payment was accepted, meaning that the flow changes to:

1. POST /payments to create the payment
2. Payment is immediately considered to be accepted without any other action required.
3. Poll the payment status; you’ll get back that it’s been accepted.

If you want to test the polling case more thoroughly, you can store state on the client side and use an additional header when calling the GET payment/payment status endpoints, as shown in the following sample request.

Request

```
GET    /sandbox/v1/berlin-group/v1/payments/sepa-credit-transfers/{{paymentstId}}/status HTTP/1.1
Authorization: bearer {{access_token}}
X-Request-ID: {{Unique UUID}}
X-Sandbox-Transaction-Status: RCVD

Content-Type: application/json
```

Response

```
{
    "transactionStatus": "RCVD"
}
```