DeVolksbank Psd2 sandbox

Version 1.1 - 5 July 2019

Introduction

This document describes the test environment that the *DeVolksbank* makes available to test the Psd2 API, inside a 'sandbox' environment.

What is the difference between Sandbox and Production?

The production environment that enables the Psd2 functionality is complex. It consists of many different systems in which each system takes care of a portion of the entire business logic.

In contrast the Sandbox looks identical to the production environment, but instead of returning live data (or live error responses), the Sandbox always returns static data. But that does not mean it always returns exactly the same data for each invocation. Based on the given input (in particular the consentId) a different response, including error responses may be returned.

The Sandbox environment enables you to develop and test your application.

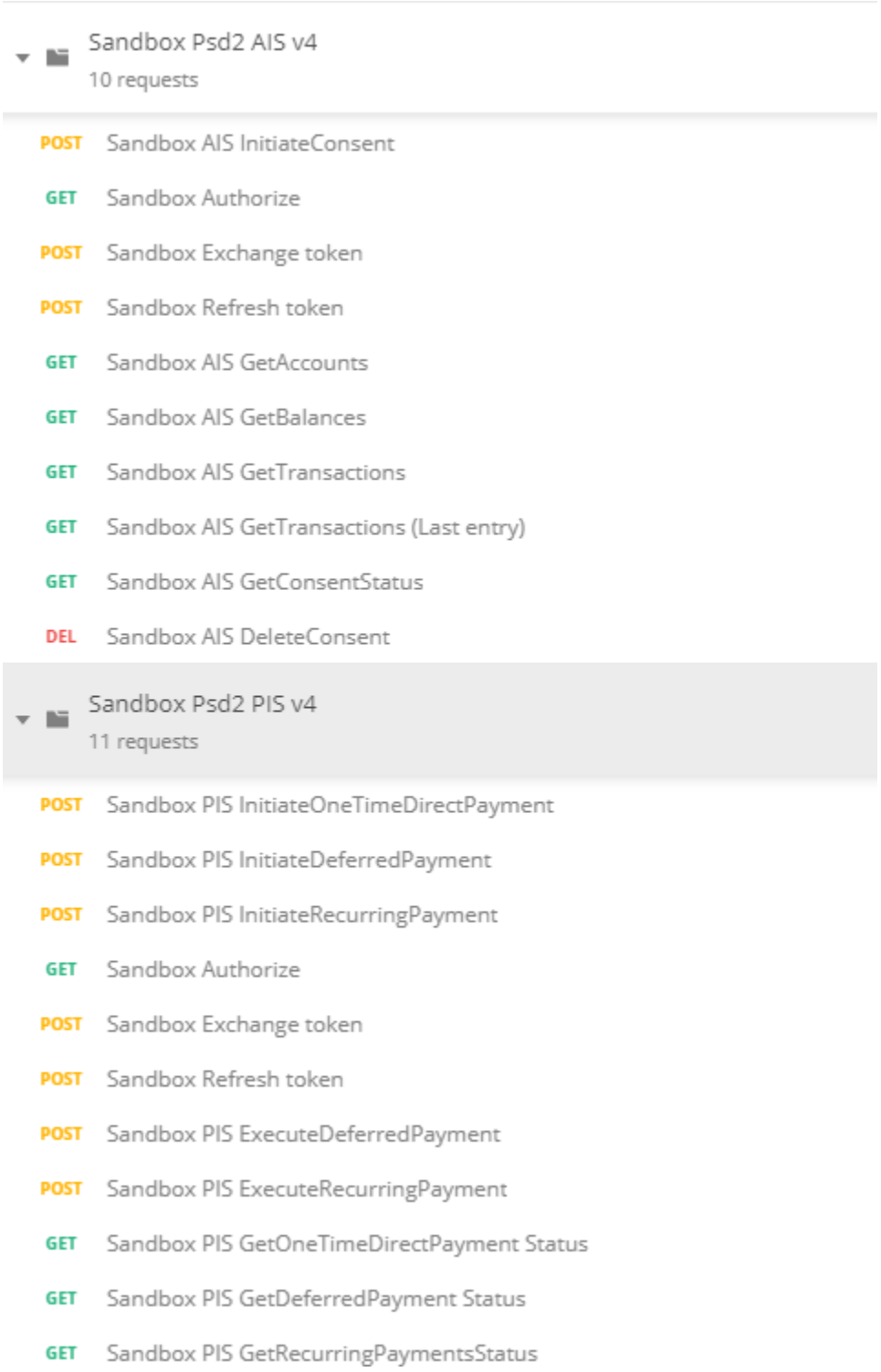
- Sandbox simulates all interactions with the *DeVolksbank* api, similar to the production environment.
- Sandbox allows you to fully test the OAuth2 process without needing an actual *DeVolksbank* account. This includes the interaction with our so-called production WebServerGateway (WSG), which requires a valid client certificate (see our https://openbanking.devolksbank.nl/apis.html).
- Sandbox APIs describes how to simulate specific error scenario.

Get started

To give you a kick-start, we provided a zip file that contains two Postman collections and a Postman environment file, that can be used to test all available Authorize and Psd2 AIS resp. PIS Api. The Postman files, not only contain the correct url 's to the various endpoint, but also all the (required) header argument and body (where applicable) are provided.
Inside *sandbox-devolksbank.postman_environment.json* there are a number of environment variable values that start with "*your-*". You should replace these with a valid value.
In particular, the following environment variables are critical: *clientId , clientSecret, redirectUrl, accessToken , consentId, paymentId* and *resourceId.*
The first three variables are fixed and are part of the initial sign-up process. The latter variables are returned in the form of json response messages by various requests.

The zipfile can be found at: https://openbanking.devolksbank.nl/documentation.html

Currently the following Postman requests are supported:



Below the main flows that are available in the Sandbox are explained more in detail.

Happy flow scenario

The happy flow consists largely of 3 flows

- API flow to initiate a Consent or to initiate a payment. The first API call to “generate” / retrieve an AIS or PIS resource are respectively:
  - Sandbox AIS InitiateConsent.
  - Sandbox PIS Initiate (OneTimeDirect or Deferred or Recurring) Payment
- The Authorization flow to retrieve an authorization code of the PSU and to exchange this code for access & refresh tokens;
- API flow to retrieve (get) account information, manage the consent or to execute a deferred/recurring payment (and to retrieve the payment status). For these calls you first need as input a (new) access token.
  Note that for getting the consent status only a client id/secret is necessary as input (not an access token)

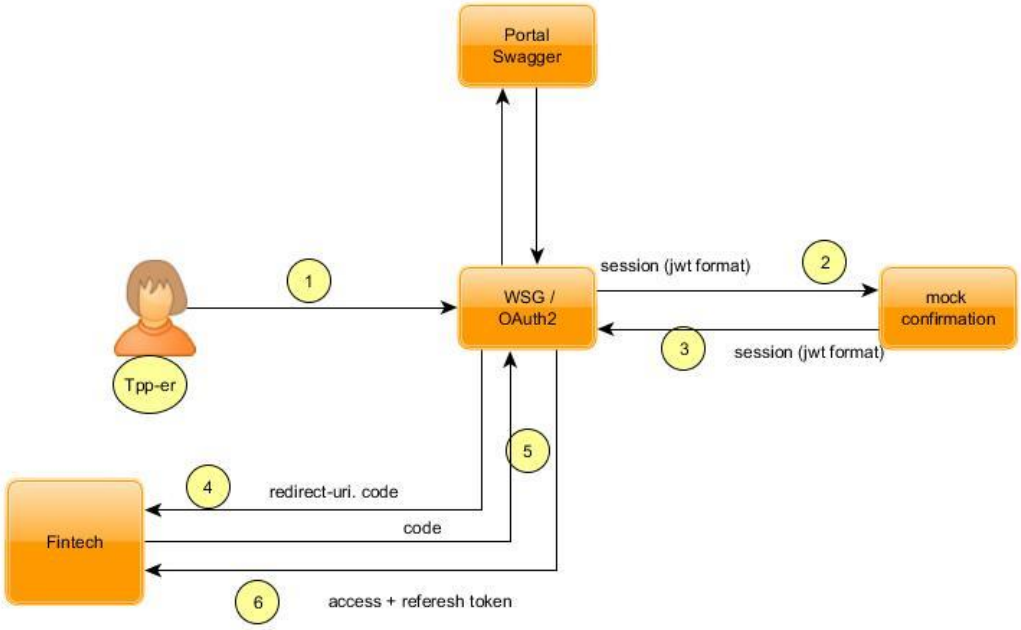
The flows above are in detail described in the Pdf files ‘API AIS’ respectively ‘API PIS’ on our Open banking website: https://openbanking.devolksbank.nl/documentation.html

Authorization flow

Within the sandbox, the flow (described below) is used, to simulate the process for authorizing the fintech app, to access data of a deVolksbank customer.

Note that the api authorization flow described below is not part of the Berlin group api, and can **not** be found in a swagger file on the development portal. This api, is the deVolksbank specific api on its “Webservice Gateway” that is used for authorization. This “Webservice Gateway” is connected with the DeVolksbank specific OAuth2 server, which is used for authentication.

DeVolksbank 'Webservice Gateway" api for authentication + authorization.



Explanation:

1. The Tpp (that is you), initiates the authorization flow
   see Postman request: '**Sandbox Authorize'**.

2. All requests are routed through the **W**eb**S**erver**G**ateway (WSG), including the request above. This particular request is forwarded to *'mock confirmation'*, the Sandbox implementation that mimics the behavior of the production login and consents flow. Here a screen (see *1) is presented, with a button to grant access.

3. The 'm*ock confirmation,* returns the sessionData in (the form of a Jwt token) to the WSG.

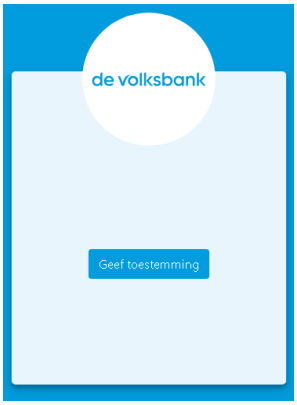
4. The WSG verifies this Jwt token and redirects an authorization **code** to the redirect url, that you provided.

5. With this **code** (from above) plus the **client-id** and **client-secret** (that you received from deVolksbank) , the app of the Tpp can then make a request for an access and refresh token.
   see Postman request:' Sandbox Exchange Token'.

6. If all data from above is valid, the WSG returns an access + refresh token.

Note *1

In production the DeVolksbank client, is redirected to the login page, where he/she need to logging, select an account and finally grant access by hitting the 'Granting access' button.
In the Sanbox, the following screen is presented, where you only have to hit the [*Geef toestemming*] button:

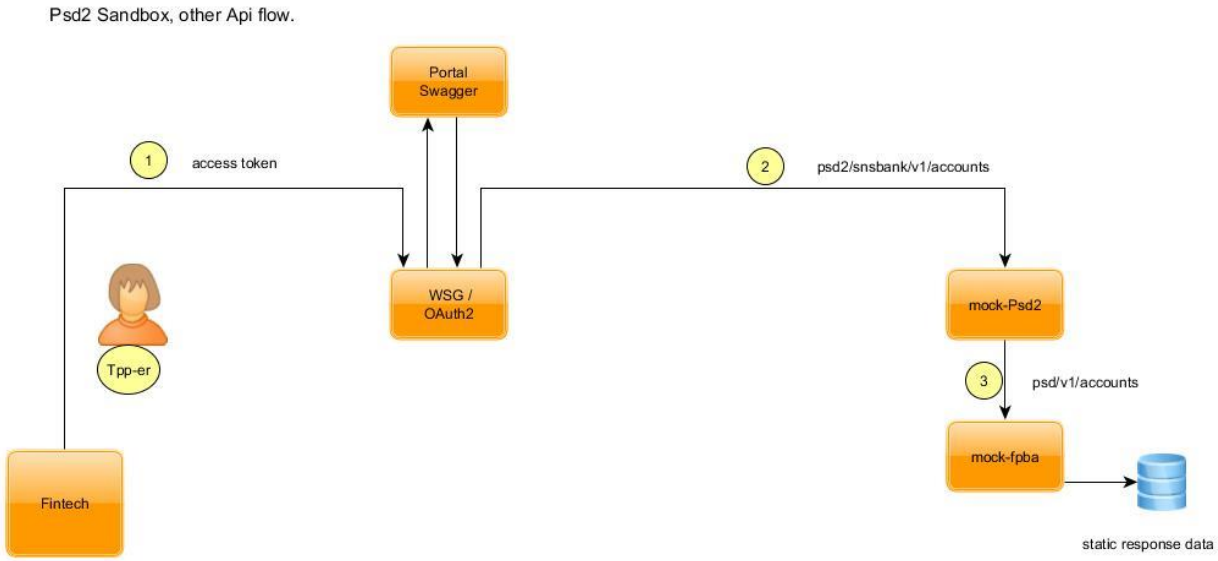


Refresh access token flow

The access token from above is valid for 10 minutes. After 10 minutes, this access cannot be used anymore for other operations, but with the refresh-token, that was also returned in the flow above, a new access-token can be obtained with:
see Postman request: 'Sandbox Exchange token'

API flow

Once the TPP App obtained a valid access-token, it can execute the Api Rest calls, as described in the Sandbox swagger files, on the developer portal. For all these calls the following flow is used:



Explanation:

1. With the access-token obtained in the previous flow, the TPP app can then execute other operations from the deVolksbank PSD2 API.
   For example:
   see Postman request: **'Sandbox AIS GetAccounts'** or **"Sandbox PIS Execute(Deferred or Recurring) Payment"**

2. The request above is forwarded via the WSG to *'mock-Psd2'.* Here validations on all input field, including header and request parameters take place. Note that these validations are production like.

3. If all validations from above are passed, the request is then 'processed', by simply returning static data. What static data is returned exactly, depends on the input consentId, paymentId **,** or (in some scenario) the resourceId.
   See also paragraph 'Unhappy test scenario', what response is returned.

Test the API.

The easiest way to test the Api, is via the Postman collection and environment file that we prepared for the Sandbox.

Note that the returned response http code and response message for the happy flow as well as all unhappy flows, comes from static data. By default all the available Postman requests will return with a happy response. By providing a special consentId or paymentId different error scenario can be simulated, see the table below what consentId/paymentId corresponds to what error scenario.
The validation however on input request and/or header parameters, is production like. Hence in some use-cases you may get an error response, if the input validation fails. In addition, the Sandbox also tries to simulate production like behavior to some extent. For example providing an illegal consentId/paymentId will result in a http 401 response with the response message that this mandate cannot be found. And the AIS GetTransactions flow is simulated in such a way that depending on the input a (fixed) number of transactions or a response indicating that no more transactions are available, is returned (see Postman request: Sandbox AIS GetTransactions (Last entry)).

For example, the Postman request 'Sandbox GetTransactions', looks like this:

```
{{host}}/psd2/sandbox/v1/accounts/SNS7642002867101/transactions
?bookingStatus=booked&dateTo=2018-10-31
```

If you replace for example: bookingStatus=booked with bookingStatus=XXX you will get a production like response: response code = 400: Bad Request.

Unhappy test scenario

In the production environment, errors may occur even if all input is valid. These error scenarios can be simulated by using on the following consentId or paymentId in a Postman request. This consentId /

paymentId should be set in the Postman environment file.
Note for those requests, that don't need a consentId or paymentID as input ('Initiate consent' and 'Initiate payment'),- the xRequestId can be used.

consentId	Unhappy scenario
SNS1313131313000	Mandate (resource) not found Note this response is returned when the consentId or paymentId is invalid.
SNS1313131313001	Mandate (Consent/payment) is expired
SNS1313131313002	Mandate (Consent/payment) is revoked by the PSU
SNS1313131313004	AIS GetTransactions, input validation error. For example endDate is before startDate.
SNS1313131313007	Iban is not in contract
SNS1313131313019	Generic error from backend system.
SNS1313131313021	AIS GetConsentStatus, mandate is revoked
SNS1313131313022	AIS GetConsentStatus, mandate is expired
SNS1313131313023	AIS GetConsentStatus, mandate is terminated
SNS1313131313030	Resource not found Note this response is returned when the resourceId is invalid.