DeVolksbank Psd2 sandbox

Version 1.0 March 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.

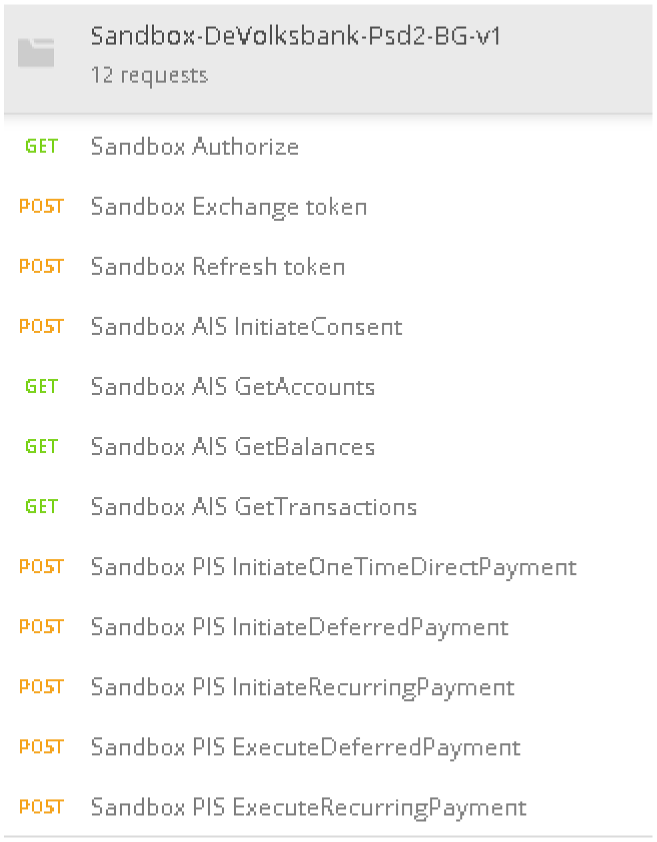
The Sandbox environment enables you to develop and test your application.

- 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.
- Sandbox APIs describes how to simulate specific error scenario.

Get started

To give you a kick-start, we provided a zip file that contains a Postman collection and a Postman environment file, that can be used to test all available Authorize and Psd2 API.
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.
This zipfile can be found at: https://openbanking.devolksbank.nl/documentation.html

Currently the following Postman requests are supported:



Sandbox-DeVolksbank-Psd2-BG-v1
12 requests

GET Sandbox Authorize
POST Sandbox Exchange token
POST Sandbox Refresh token
POST Sandbox AIS InitiateConsent
GET Sandbox AIS GetAccounts
GET Sandbox AIS GetBalances
GET Sandbox AIS GetTransactions
POST Sandbox PIS InitiateOneTimeDirectPayment
POST Sandbox PIS InitiateDeferredPayment
POST Sandbox PIS InitiateRecurringPayment
POST Sandbox PIS ExecuteDeferredPayment
POST Sandbox PIS ExecuteRecurringPayment

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

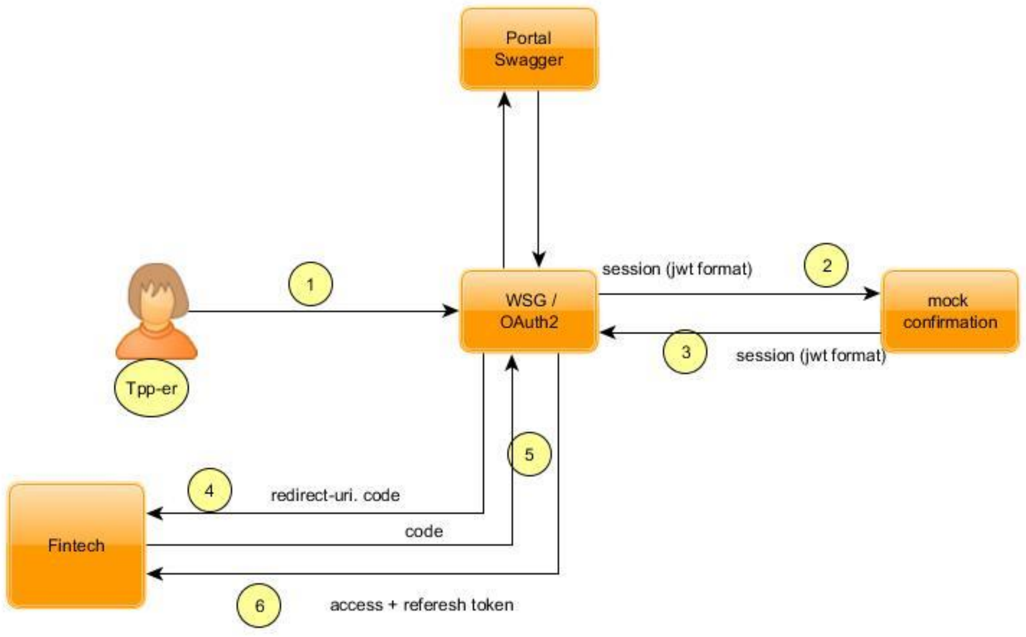
Authorization flow

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

Note that the api describes 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.

Note: In https://openbanking.devolksbank.nl/documentation.html the Pdf files ‘API AIS’ respectively ‘API PIS’ provide more detail information.

DeVolksbank 'Webservice Gateway" api for authentication + authorization.



Explanation:

1. The Tpp-er (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 confiration'*, 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.

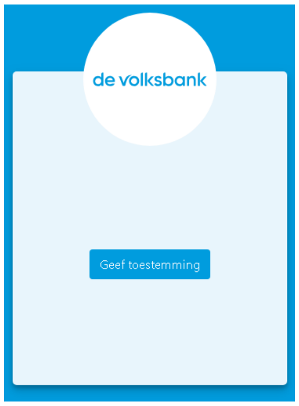
4. The WSG verifies this Jwt token and redirects a **code** to the redirect url, that you provided.

5. With this **code** (from above) plus the **client-id** and **client-secret** (that you received from DeVolksbank) , the Tpp app 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 ‘Allow access’ button.
In the Sanbox, the following screen is presented, where you only have to hit the [*Geef toestemming*] button:

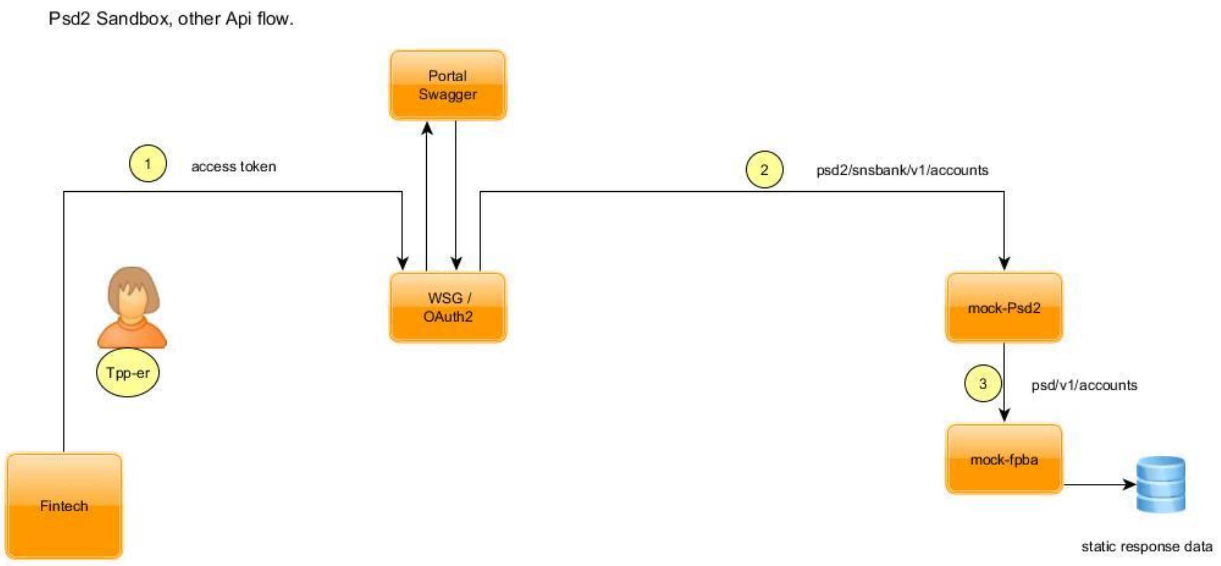


Refresh access token flow

The access token from above is valid for 10 minutes. After 10 minutes, this access cannot be used anymore, 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 methods from the DeVolksbank Psd2 API.
   For example:
   see Postman request: **'Sandbox InitiateConsent'**

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 ,** or (in some scenario) the **xRequestId**.
   See paragraph 'Unhappy test scenario', what response is returned

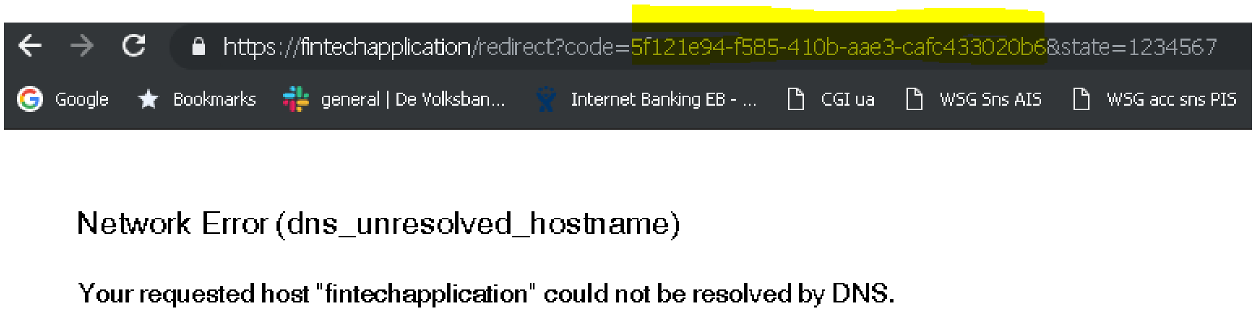
Test the API.

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

The first step is to retrieve a **code** from the Wsg. This can be achieved via the 'Sandbox Authorize' request. The Sandbox Wsg is production like, hence you have to set the correct values in the Postman environment file, specifically the following fields:

- clientId
- clientSecret
- redirectUri

If you supplied a redirectUri, that is not (yet) fully functional, you may grab the **code** from the redirect uri in a browser like this:

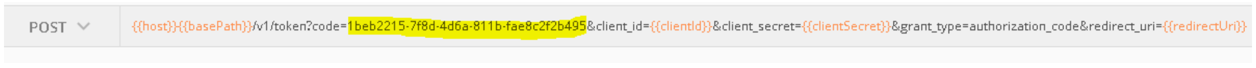


https://fintechapplication/redirect?code=5f121e94-f585-410b-aae3-cafc433020b6&state=1234567

Network Error (dns_unresolved_hostname)

Your requested host "fintechapplication" could not be resolved by DNS.

This **code** can then be used in the next Postman request: ' Sandbox Exchange token', to exchange the **code** with a access and refresh token. Note that you have to set the correct code in the Postman request:



```
POST  {{host}}{{basePath}}/v1/token?code=1beb2215-7f8d-4d6a-811b-fae8c2f2b495&client_id={{clientId}}&client_secret={{clientSecret}}&grant_type=authorization_code&redirect_uri={{redirectUri}}
```

When a valid **code** is provided, an **access-token** and **refresh-token** are returned. This **access-token,** should be set in the Postman environment property: **accessToken**.
After that, the other Postman request that contain 'AIS' or 'PIS' can be executed, in any order.

This **accessToken** is valid 10 minutes. With the Postman request: 'Sandbox Refresh token', this can be refreshed.

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** different error scenario can be simulated, see the table below what **consentId** 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.

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

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

If you replace for example: bookingStatus=booked with bookingStatus=WrongStatus, 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' s in a Postman request. This consentId is set in Postman environment file.
Note for those requests, that don't provide a consentId ('Initiate consent' and 'Get Accounts'), the xRequestId can be used.

consentId	Unhappy scenarion
SNS1313131313001	Mandate is expired
SNS1313131313002	Mandate is revoked
SNS1313131313003	Mandate is not permitted
SNS1313131313004	Operation not allowed for this account
SNS1313131313005	No active digipass tokens available
SNS1313131313006	Invalid input
SNS1313131313007	Account not in contract
SNS1313131313008	Internal server from the frontend system
SNS1313131313009	Jwt token expired
SNS1313131313010	Jwt token invalid
SNS1313131313011	Input validation error
SNS1313131313012	Parameter not allowed for read transaction
SNS1313131313014	Validation error from backend system, invalid payment amount

SNS1313131313015	Validation error from backend system, invalid endDate
SNS1313131313016	Validation error from backend system, endDate required for oneTimePlanned.
SNS1313131313017	Validation error from backend system, for oneTimeDirect no endDate should be supplied
SNS1313131313018	Validation error from backend system, endDate too far in the future
SNS1313131313019	Generic error from backend system.