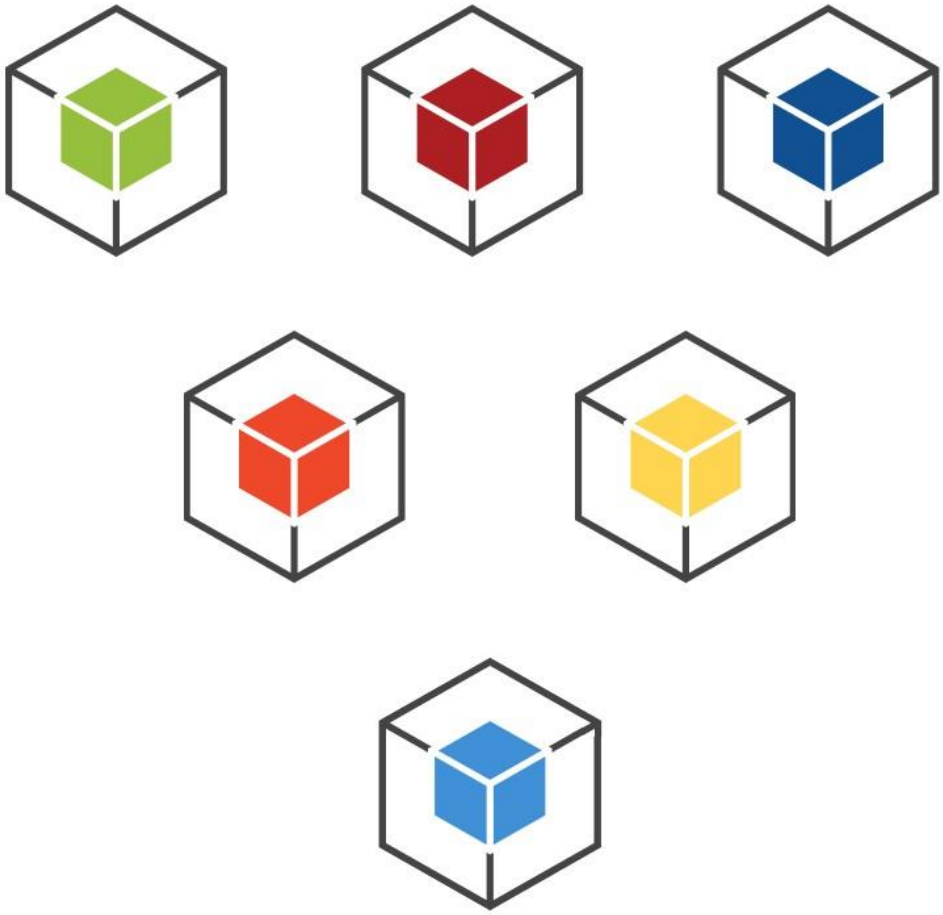
API

PSD2 Accounts API

Version 1.3 ● 17 August 2021



BANQUE BANORIENT FRANCE S.A. SUCURSALA ROMANIA

Address: 66 Unirii Blvd.K3 Block, 3rd District Bucharest, Romania

Email: psd2.sandbox@banorientfrance.com

Trademarks

EUBank are registered trademarks of Advahoo SRL Company. All other trademarks or registered trademarks are the property of their respective owners.

Disclaimer

The information provided in this document is provided "as is" without warranty of any kind. BANORIENT Bank disclaims all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall BANORIENT Bank be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if BANORIENT Bank or its suppliers have been advised of the possibility of such damages.

Document Lifetime

BANORIENT Bank may occasionally update online documentation between releases of the related software. Consequently, if this document was not downloaded recently, it may not contain the most up-to-date information. Please refer to https://www.banorientfrance.com for the most current information.

From the Web site, you may also download and refresh this document if it has been updated, as indicated by a change in this date: 14-03-2019.

Where to get help

BANORIENT Bank support, product, and licensing information can be obtained as follows.

Product information

For documentation, release notes, software updates, or for information about BANORIENT Bank products, licensing, and service, go to the following website: https://www.banorientfrance.com/romanian/romania **Technical support**

For technical support, use the email address psd2.sandbox@banorientfrance.com.

Note that to open a service request, you must have a valid support agreement.

Your comments

Your suggestions will help us continue to improve the accuracy, organization, and overall quality of the user publications. Please send your opinion of this document to: psd2.sandbox@banorientfrance.com

If you have issues, comments, or questions about specific information or procedures, please include the title and, if available, the part number, the revision, the page numbers, and any other details that will help us locate the subject that you are addressing.

Preface

Intended Audience

This guide is part of the PSD2 Accounts API documentation set. It is intended for use by System Administrator, Application Developers from Third Party Provider during integration of the PSD2 services offered by BANORIENT Bank.

Readers should be familiar with the following API specifications defined by Berlin Group.

01. NextGenPSD2 Access to Account Interoperability Framework - General Introduction Paper V2_20181120.pdf

02. NextGenPSD2 Access to Account Interoperability Framework - Operational Rules V1_20180208.pdf

03. NextGenPSD2 Access to Account Interoperability Framework - Implementation Guidelines V1.3_20181019.pdf

04. NextGenPSD2 Access to Account Interoperability Framework - ChangeLog V12 V13 20181019.pdf

Style Conventions

The following style conventions are used in this document:

Bold

- Names of commands, options, programs, processes, services, and utilities
- Names of interface elements (such windows, dialog boxes, buttons, fields, and menus)
- Interface elements the user selects, clicks, presses, or types

Italic

- Publication titles referenced in text
- Emphasis (for example a new term)
- Variables

Courier

- System output, such as an error message or script
- URLs, complete paths, filenames, prompts, and syntax

Courier italic

- Variables on command line

User input variables

- < > Angle brackets enclose parameter or variable values supplied by the user
- [ ] Square brackets enclose optional values
- | Vertical bar indicates alternate selections - the bar means “or”
- { } Braces indicate content that you must specify (that is, x or y or z)

Table of Contents

Document History

Paper copies are valid only on the day they are printed. Contact the author if you are in any doubt about the accuracy of this document.

Revision History

This document has been revised by:

Revision Number	Revision Date	Summary of Changes	Author
v1.1	26 March 2020	Initial version	BANORIENT Bank
v1.2	14 July 2020	Second version	BANORIENT Bank
v1.3	17 August 2021	Third version	BANORIENT Bank

1. Overview

This guide presents the PSD2 Accounts API services offered by BANORIENT Bank.

The services are protected by OAuth2 protocol. Order of presentation for the services will follow the logic access, including authentication step, token exchanges, and status.

This will help users to standalone test the services without developing a specific application for this purpose.

All services are documented using Open API 3.0 version.

For details please follow https://sandbox.banorientfrance.com/eubank/openapi-accounts

1.1. Conventions

We use the following conventions in this document:

- Responses are listed under 'Responses' for each method.
- Responses are in JSON format.
- Request parameters are mandatory unless explicitly marked as Optional.
- The type of values accepted for a request parameter are shown the **values** column.
- The | symbol means OR.

1.2. Current Version

Version specification follow Berlin Group recommendations and are present in the URL form [/v1/].

1.3. Schema

All API access is over HTTPS, and accessed from the address: https://sandbox.banorientfrance.com/DVHPSD2AccountsAPI/

All data is sent and received as JSON.

All timestamps are returned in ISO 8601 format: YYYY-MM-DDTHH:MM:SS

Summary Representations - When you fetch a list of resources, the response includes a subset of the attributes for that resource. This is the "summary" representation of the resource.

Detailed Representations - When you fetch an individual resource, the response typically includes all attributes for that resource. This is the "detailed" representation of the resource.

1.4. HTTP Redirects

If necessary, describe if the API uses HTTP redirection. Help the reader understand the purpose the redirect and status code information.

“Receiving an HTTP redirection is not an error and clients should follow that redirect. Redirect responses will have a Location header field which contains the URI of the resource to which the client should repeat the requests.”

Status Code	Description
301	Permanent redirection. The URI used to make the request has been superseded by the one specified in the Location header field. Direct this and all future requests to this resource to the new URI.

1.5. Communication security

PSD2 Directive defines requirements on communication among payment service providers and account servicing institutions.

The Regulatory Technical Standards defines requirements on the use of qualified certificates (as defined in eIDAS) for website authentication and qualified certificates for electronic seal for communication among payment and bank account information institutions.

The ETSI TS 119 495 defines a standard for implementing the requirements of the RTS for use of qualified certificates as defined in eIDAS (Regulation (EU) No 910/2014) to meet the regulatory requirements of PSD2.

Regulatory Technical Standards mandates the use of certificates according to Article 34. The article restricts the use of certificates to "qualified certificates for electronic seals as referred to in Article 3(30) of Regulation (EU) No 910/2014 or for website authentication as referred to in Article 3(39) of that Regulation".

EUBank will encrypt the communication between Bank and TPP by using a SSL extended validation certificate. No mutual TLS authentication and encryption will be used.

The TPP request and Bank responses will be authenticated and protected by the usage of QSealC certificates. Both TPP and Banks will sign the corresponding requests and responses using qualified seal certificates.

Message Signing

Each request initiated by a TPP must contain a JSON Web Signature as a header. This header signs the payload of the request, using the private key of the TPP’s compliant certificate. The responses are also signed using the Bank’s certificate, using the same technique. Both the Bank and the TPP must validate requests and responses using the appropriate public keys.

The present documentation details the signing procedure for the TPP requests. The bank will sign responses using the same methodology.

We assume TPP software will compute and attach the header signature for each API requests.

For procedure verification only this document includes detailed examples for building signatures on Windows Operating System.

References

In order to build and check the signature on requests please considers the following references:

JSON Web Signature Documentation

Base64URL Documentation

OpenSSL

Certificates

The sandbox contains for testing a pre-registered TPP SC_EXEMPLU_SRL. The certificate and the private key for SC_EXEMPLU_SRL are available for download on sandbox page.

The Bank's public certificate is available for download on Bank website and sandbox page.

OAuth2 Protocol Implementation

Authorization of the consent for accounts access is controlled using Authorization code grant type protocol. For SC_EXEMPLU_SRL TPP an application was previously registered with following parameters:

Client Id: LrcL4ywuHuLtyf34g40LNf14RFfDJ4SL

Client Secret: N9Vt3Jm9Bx3MCDByyclwXcbliyqxXzGk

Request header parameter signature format

Each request coming from the TPP will include a special header parameter x-jws-signature.

The signature includes three sections:

1. JWS Header
2. JWS Payload
3. JWS Signature

The three sections are finally assembled in the parameter x-jws-signature.

x-jws-signature= Base64URL (JWS Header) '.'Base64URL (JWS Payload) '.'Base64URL (JWS Signature)

1. The JWS Header

The JWS Header will contain specific information:

- alg: the algorithm to sign - RS256
- typ: type of the encoded object –JOSE
- kid: certificate thumbprint for SC_EXEMPLU_SRL the value is:

  133c11470740d7ed33c86c3501e3ac8221fece03

Consequently, the JWS Header will be:

JWS Header = {"alg":"RS256","typ":"JOSE","kid":"133c11470740d7ed33c86c3501e3ac8221fece03"}

For obtaining Base64URL(JWSHeader) the steps are: -

Compute base64 for JWS Header

- Replace any occurrences of '+' character with '-' and any occurrences of '/' character with '_'. Also, delete every '=' from the resulted string.

The result in our test case is:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjMzUwMWUzYW
M4MjIxZmVjZTAzIn0
```

2. *The JWS Payload*

The JWS Payload is specific for each request; an example is provided for each request type within the document, starting from the general template of the JSON request which includes the headers information and the body information, altogether on a single line, trim spaces:

```
{"headers":{all not null headers properties as they occur in request },"payload":{request body}}
```

3. *The JWS Signature*

The process of computing the *JWS Signature* component includes the following steps:

1. Concatenate the first two parts separated by a *'.'*:

```
Base64URL(JWS Header) '.' Base64URL(JWS Payload)
```

2. Sign the resulted string using the TPP private key and then apply Base64 encoding.

3. Replace any occurence of “+" character with “-" and any occurence of "/" character with “_". Also, delete every “=" from the resulted string.

2. API Reference Documentation

PSD2 Accounts API

Under the Account Information Service, the following set of methods is available (Figure 1):

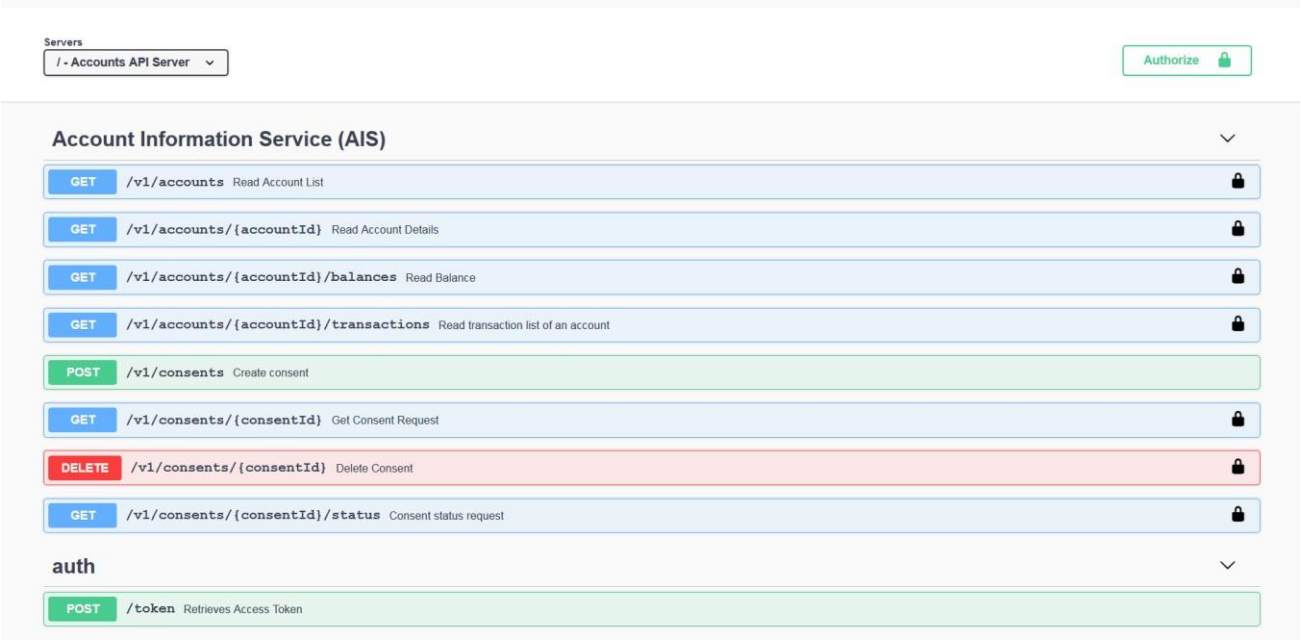


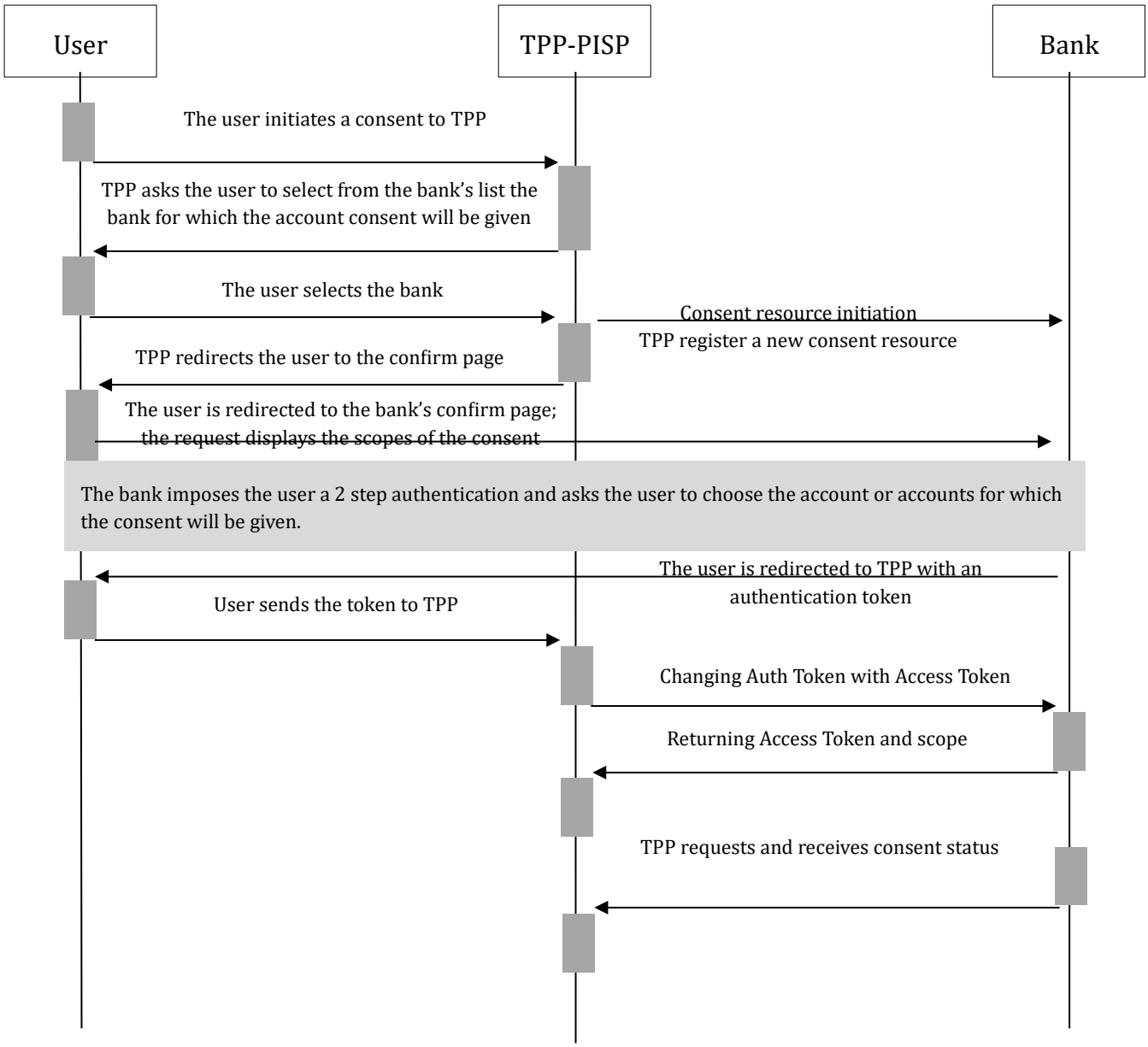
Figure 1

2.1. Overview

Method	Endpoint	Purpose
GET	/v1/accounts	Read accounts list.
GET	/v1/accounts/{accountId}	Read account details.
GET	/v1/accounts/{accountId}/balances	Read account balance.
GET	/v1/accounts/{accountId}/transactions	Read account transactions.
POST	/v1/consents	Create consent.
GET	/v1/consents/{consentId}	Returns the content of a consent object.
DELETE	/v1/consents/{consentId}	Delete consent.
GET	/v1/consents/{consentId}/status	Returns the consent status.
POST	/token	Retrieve access token for a specific account resource.

2.2. Specific flow

Under PSD2 rules creating an account information consent follow a specific flow of API calls. The calls are protected by OAuth2 authentication and authorization protocol using authorization code flow.



3. Testing a consent flow example

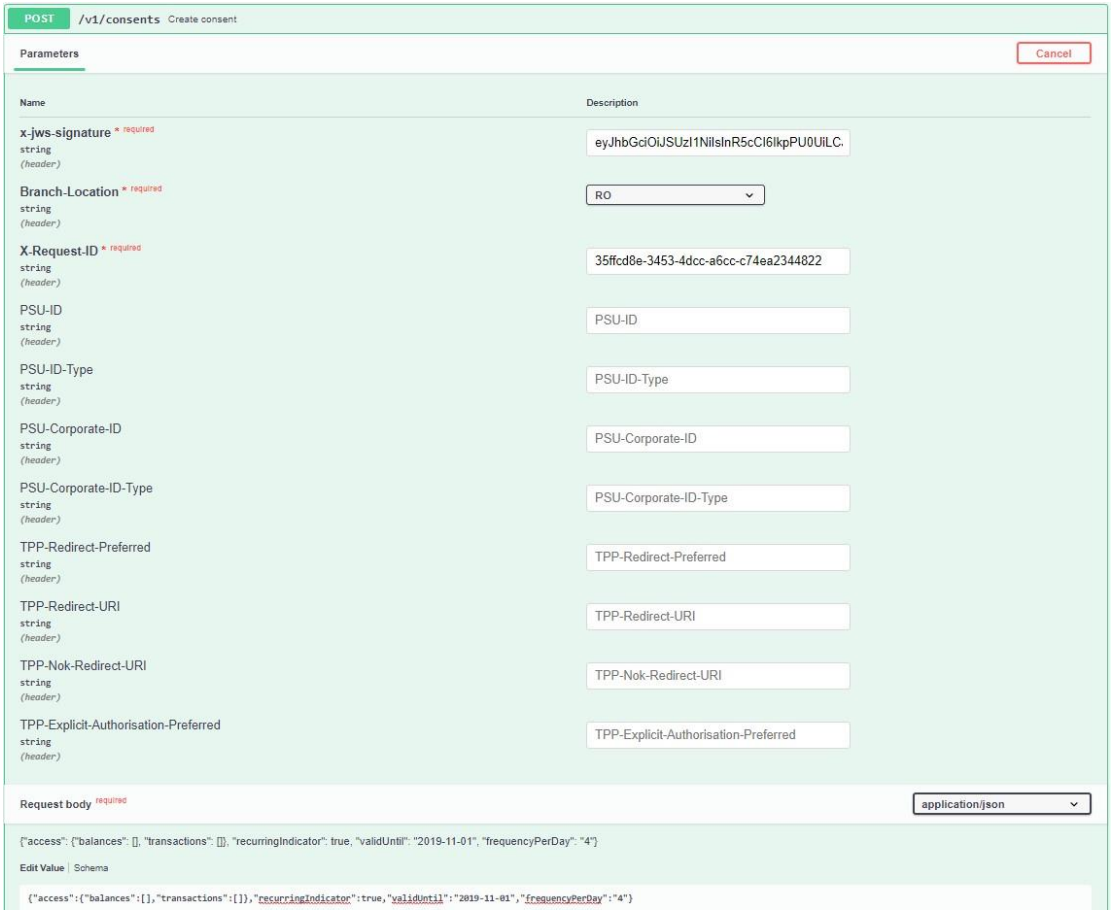


Figure 2

This section presents an example of a complete consent authorization. The scenario assumes that one customer [PSU] initiated an account consent request on a site [TPP]. In order to authorize the consent using PSD2 standard the user is asked to select the bank where he owns the account for which the consent will be done and the scopes of the consent. The scopes can be one or many of the following: accounts, balances and transactions.

- The accounts scope is used to view the details of the account for which the consent will be given.
- The balances scope is used to view the balance of the account for which the consent will be given.
- The transactions scope is used to view the transactions of the account for which the consent will be given.

After the bank and scope selection, the site initiates a consent resource registration. This is done through API call Consent resource initiation on bank side [ASPSP].

For testing this service please launch the call from the BANORIENT Bank sandbox UI having the endpoint **/v1/consents**.
For this request the body used is the same on any request.

Steps

The *Request Body must be* on a single line without spaces:

```
{"access": {"balances": [], "transactions": []}, "recurringIndicator": true, "validUntil": "201911-01", "frequencyPerDay": "4"}
```

Compute the ***Base64URL (JWS Payload):***

1. Create the following JSON on a single line without spaces:

```
{"headers":{"Branch-Location":"RO","X-Request-ID":"35ffcd8e-3453-4dcc-a6ccc74ea2344822"},"payload":{"access":{"balances":[],"transactions":[]},"recurringIndicator":tr ue,"validUntil":"2019-11-01","frequencyPerDay":"4"}}
```

2. Apply SHA-256 on the JSON from step 1 using the following command (on Windows OS):

```
echo|set /p="{"headers":{"Branch-Location":"RO","X-Request-ID":"b7d96357-b320-4f54bb02-1c4511e4b772","PSU-IP-Address":"127.0.0.1"},"payload":{"endToEndIdentification":"test","instructedAmount":{"currency":"RON","amount":"101"},"creditorAccount":{"iban":"RO61TREZ27A660404200109X"},"creditorName":"PaySafe"}}" | openssl dgst -sha256
```

The result will be:

```
b31299e351a3a5519e19b71de9fe34c9e417504d85cf499d840e373ce6587a32
```

3. Create the following JSON with the result:

```
{"SHA256":" b31299e351a3a5519e19b71de9fe34c9e417504d85cf499d840e373ce6587a32"}
```

4. Compute Base64 encoding on the later JSON using the following command (on Windows OS):

```
echo|set /p="{"SHA256":"b31299e351a3a5519e19b71de9fe34c9e417504d85cf499d840e373ce6587a32"}" | openssl base64 -e –A
```

5. Replace any occurence of “+" character with “-" and any occurence of "/" character with “_". Also, delete every “=" from the resulted string. The result for our test case is:

```
eyJTSEEyNTYiOiJiMzEyOTllMzUxYTNhNTUxOWUxOWI3MWRlOWZlMzRjOWU0MTc1MDRkODVjZjQ5OWQ4NDBlMzczY2U2NTg3YTMyIn0
```

Compute the ***JWS-signature:***

1. Compute the following string: *Base64URL(JWS Header) '.' Base64URL(JWS Payload)*

   The result will be:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJiMzEyOTllMzUxYTNhNTUxOWUxOWI3MWRlOWZlMzRjOWU0MTc1MDRkODVjZjQ5OWQ4NDBlMzczY2U2NTg3YTMyIn0
```

2. Sign the string using the TPP private key and apply Base64 encoding using the following command (on Windows OS):

```
echo|set
/p="eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJiMzEyOTllMzUxYTNhNTUxOWUxOWI3MWRlOWZlMzRjOWU0MTc1MDRkODVjZjQ5OWQ4NDBlMzczY2U2NTg3YTMyIn0" |
openssl dgst -sha256 -sign SC_EXEMPLU_SRL.key | openssl base64 -e -A
```

   The result will be:

```
NwcWzExKfx_zvGDnttR9atBGmhLaVJoK3yZZnacXpyPLUz7Etw6Jebmt_JeGjw8wr8xrHyCmX22HURQaKZsSKGzOdlNB7CDVRoiG-I_ckdiL5Z7ONjOae20aMwLWs7_XaBEdhKMMl4Gbi6LvRynfQfE-TIp97UtLBx_Bu59XGdrYDH8BUorHBSBEWYYDcNMx-ciftKSR4yU_nhCrongK72Jl5jlHimZ3ZFFvwqdNRW4jC7Jm6YoUYZz4AzKobS_LXj7MFFllhCr47pw_etKCmo1erp3Z_m5Cne7MeEwwRd_W51U9l04xYag7_2uNJ8MHduMwLLm2h3C9dej-kCg
```

3. Finally, the Header JWS-signature will be:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJiMzEyOTllMzUxYTNhNTUxOWUxOWI3MWRlOWZlMzRjOWU0MTc1MDRkODVjZjQ5OWQ4NDBlMzczY2U2NTg3YTMyIn0.NwcWzExKfx_zvGDnttR9atBGmhLaVJoK3yZZnacXpyPLUz7Etw6Jebmt_JeGjw8wr8xrHyCmX22HURQaKZsSKGzOdlNB7CDVRoiG-I_ckdiL5Z7ONjOae20aMwLWs7_XaBEdhKMMl4Gbi6LvRynfQfETIp97UtLBx_Bu59XGdrYD-H8BUorHBSBEWYYDcNMxciftKSR4yU_nhCrongK72Jl5jlHimZ3ZFFvw-qdNRW4jC7Jm6YoUYZz4AzKobS_LXj7MFFllhCr47pw_etKCmo1erp3Z_m5Cne7MeEwwRd_W51U9l04xYag7_2uNJ8MHduMwLLm2h3C9dej-kCg
```

The successful response from the bank to the site returns the consentId and the link for authentication and authorization of the customer.

```
{
 "consentStatus": "received",
 "consentId": "f6242571-ca1e-4988-a325-7916212c0420",
 "_links": {
  "scaOAuth": "https://sandbox.banorientfrance.com/services/startAuthorize"
 }
}
```


Request Response

In the next step the e-Commerce site redirects user to the SCA authentication and authorization page according to OAuth2 authorization code flow.

In order to test this step from sandbox please press Authorize button (Figure 3). From dialog box choose OAuth2 authorizationCode method and fill client_id and client_secret fields (with the values from 1.5 Message Signing – Oauth2) and choose the following consent scopes:

- accounts
- balances
- transactions

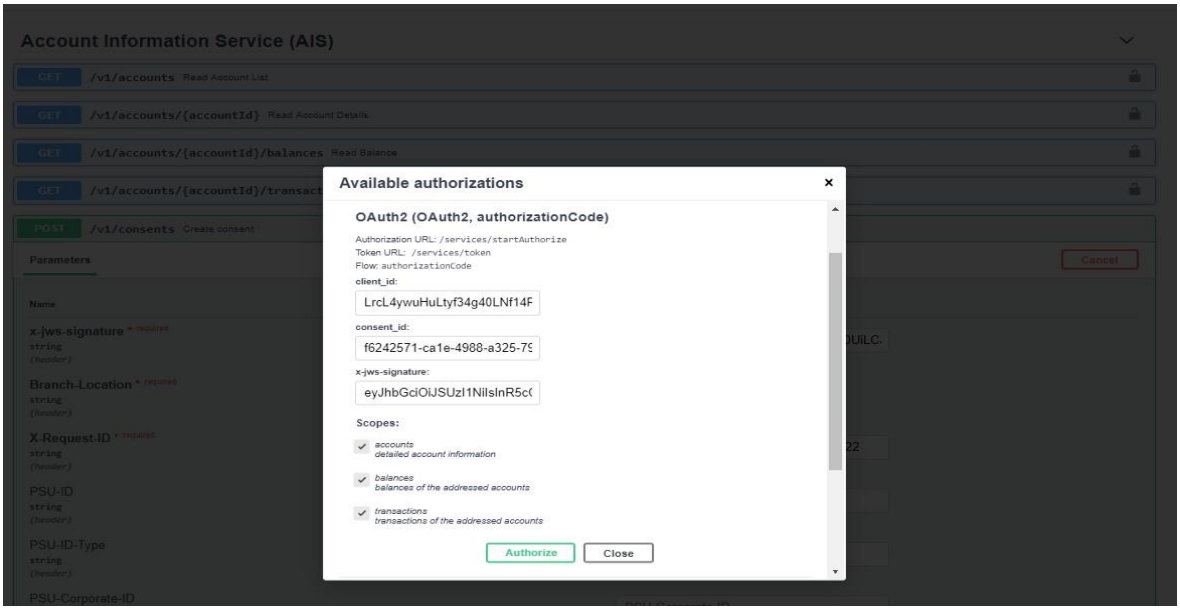


Figure 3

Steps

Compute the ***Base64URL (JWS Payload):***

1. Create the following JSON on a single line without spaces:

```
{"headers":{},"payload":{"consent_id":"f6242571-ca1e-4988-a325-7916212c0420","scope":"accounts_balances_transactions","response_type":"code","state":"state","client_id":"LrcL4ywuHuLtyf34g40LNf14RFfDJ4SL"}}
```

2. Apply SHA-256 on the JSON from step 1 using the Windows command:

```
echo|set /p="{"headers":{},"payload":{"consent_id":"f6242571-ca1e-4988-a325-7916212c0420","scope":"accounts_balances_transactions","response_type":"code","state":"state","client_id":"LrcL4ywuHuLtyf34g40LNf14RFfDJ4SL"}}" | openssl dgst -sha256
```

The result will be:

```
c9aa38878bc4ed42485acc6b8ece025d1c7601b21aa168d74f608faf948c2d50
```

3. Create the following JSON with the result:

```
{"SHA256":"c9aa38878bc4ed42485acc6b8ece025d1c7601b21aa168d74f608faf948c2d50"}
```

4. Compute Base64 encoding on the last JSON using the following command (on Windows OS):

```
echo|set /p="{"SHA256":"c9aa38878bc4ed42485acc6b8ece025d1c7601b21aa168d74f608faf948c2d50"}" | openssl base64 -e –A
```

5. Replace any occurence of “+" character with “-" and any occurence of "/" character with “_". Also, delete every “=" from the resulted string. The result for our test case is:

```
eyJTSEEyNTYiOiJjOWFhMzg4NzhiYzRlZDQyNDg1YWNjNmI4ZWNlMDI1ZDFjNzYwMWIyMWFhMTY4ZDc0ZjYwOGZhZjk0OGMyZDUwIn0
```

Compute the ***JWS-signature:***

1. Compute the following string: *Base64URL(JWS Header) '.' Base64URL(JWS Payload)*

   The result will be:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJjOWFhMzg4NzhiYzRlZDQyNDg1YWNjNmI4ZWNlMDI1ZDFjNzYwMWIyMWFhMTY4ZDc0ZjYwOGZhZjk0OGMyZDUwIn0
```

2. Sign the string using the TPP private key and apply Base64 encoding using the following command:

```
echo|set
/p="eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZ
jMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJjOWFhMzg4NzhiYzRlZDQyNDg1YWNjN
mI4ZWNlMDI1ZDFjNzYwMWIyMWFhMTY4ZDc0ZjYwOGZhZjk0OGMyZDUwIn0" | openssl dgst -
sha256 -sign SC_EXEMPLU_SRL.key | openssl base64 -e -A
```

The result will be:

```
XYK9OySSQqwQvpPjDgKEqbHylCNv69yeEMauEVW3_6Yb8SxjuMDfoicrcj9VcnNPjJygo8Yf
2dodqZAKv-
pm7niChDS9YjG6JMh6LLIyTUnIkfZTblrj0IqwjMHh5Dde9bFHt2A687C4U96RirgbJV8hS0ya
VUT63eGvnAHpRrznRdhDDm78CSk_yXa2oAwddnRbmNPxQZdPlTzzczfDtS9a93CsJya5hAl
a52YFGM6yNOJ5-ZXXFATLCvne5XvLPiuPuHWE1kIT6d3q8G-
k5xGDMZYgtx5icTKol8IOx_8TfSoi7OXCPbGhUK1uK7wSTEFNiGrzAfBUAkTl7ClUQ
```

3. Finally, the Header JWS-signature will be:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjOD
ZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJjOWFhMzg4NzhiYzRlZDQyNDg1
YWNjNmI4ZWNlMDI1ZDFjNzYwMWIyMWFhMTY4ZDc0ZjYwOGZhZjk0OGMyZDUwIn0.XY
K9OySSQqwQvpPjDgKEqbHylCNv69yeEMauEVW3_6Yb8SxjuMDfoicrcj9VcnNPjJygo8Yf2d
odqZAKv-
pm7niChDS9YjG6JMh6LLIyTUnIkfZTblrj0IqwjMHh5Dde9bFHt2A687C4U96RirgbJV8hS0ya
VUT63eGvnAHpRrznRdhDDm78CSk_yXa2oAwddnRbmNPxQZdPlTzzczfDtS9a93CsJya5hAl
a52YFGM6yNOJ5-ZXXFATLCvne5XvLPiuPuHWE1kIT6d3q8G-
k5xGDMZYgtx5icTKol8IOx_8TfSoi7OXCPbGhUK1uK7wSTEFNiGrzAfBUAkTl7ClUQ
```

Based on SCA [strong customer authentication] the user is will be redirected to the Sign in page (Figure 4). The value for username is “user1”, the value for password is “Parola1234” and the value for consent Id is the one from the request response of Create Consent service.

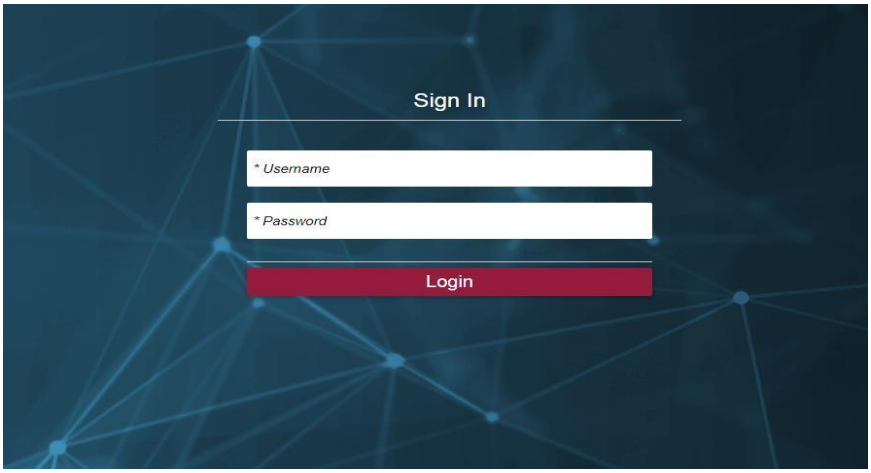


Figure 4

Finally, the user is asked to input the OTP – in this example a SMS code. Please use 123456 to test this scenario (Figure 6).

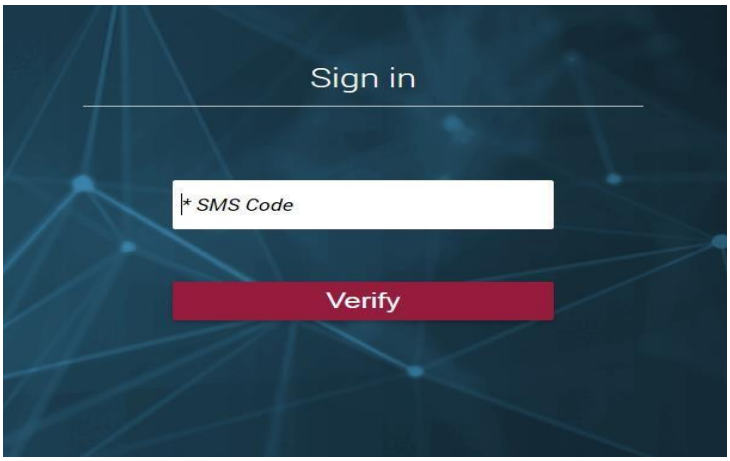


Figure 5

Following a successful SCA for consent intent, the EUBank Auth/Authz Server will present to the user the scopes (in this case accounts and balances) and the available accounts and, if the user agrees, he is asked to choose the accounts and confirm the intent (Figure 7).

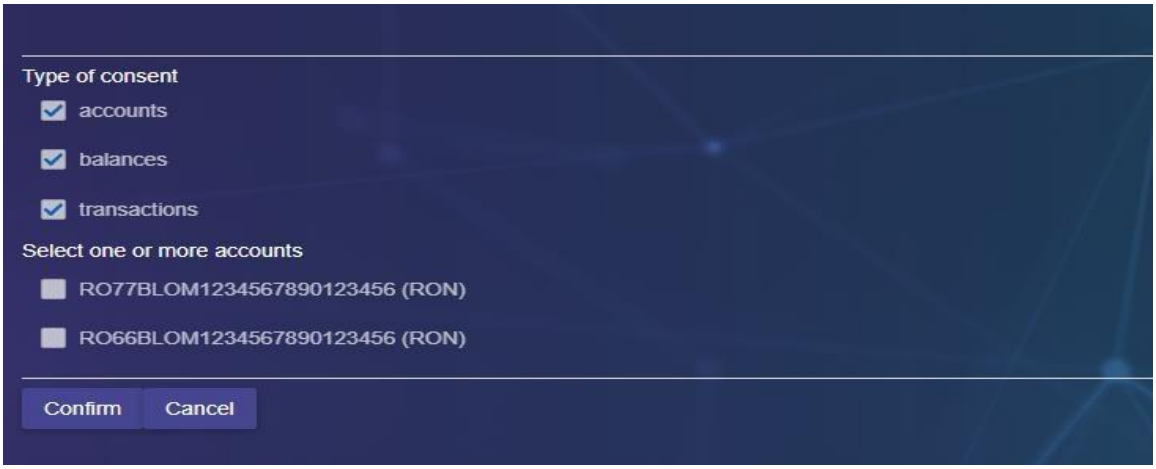


Figure 6

Following a user confirmation, the Auth/Authz Server will update the status of the consent on the bank side and the user will be redirected to the e-Commerce site with authorization code.

In our testing scenario the e-Commerce site is not present. That's why the redirected action will not succeed, but in our case will allow us to copy the authorization code from the browser URL (Figure 8).

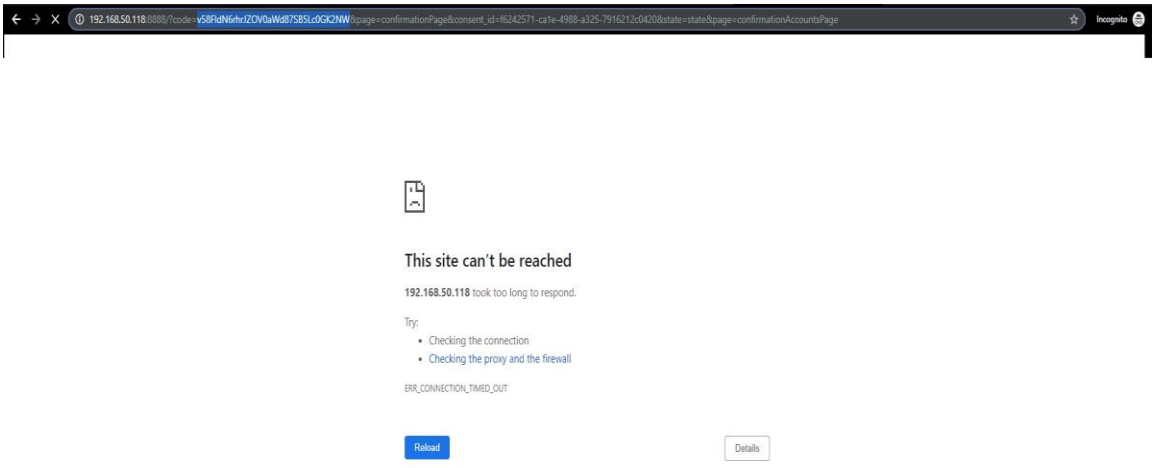


Figure 7

The next step in the OAuth2 flow is to exchange the *authorization code* for *access token*.

For this operation the e-Commerce site application will call Retrieves Access Token specific API on the bank side.

In order to test the service from sandbox first step is to select it (Figure 9).

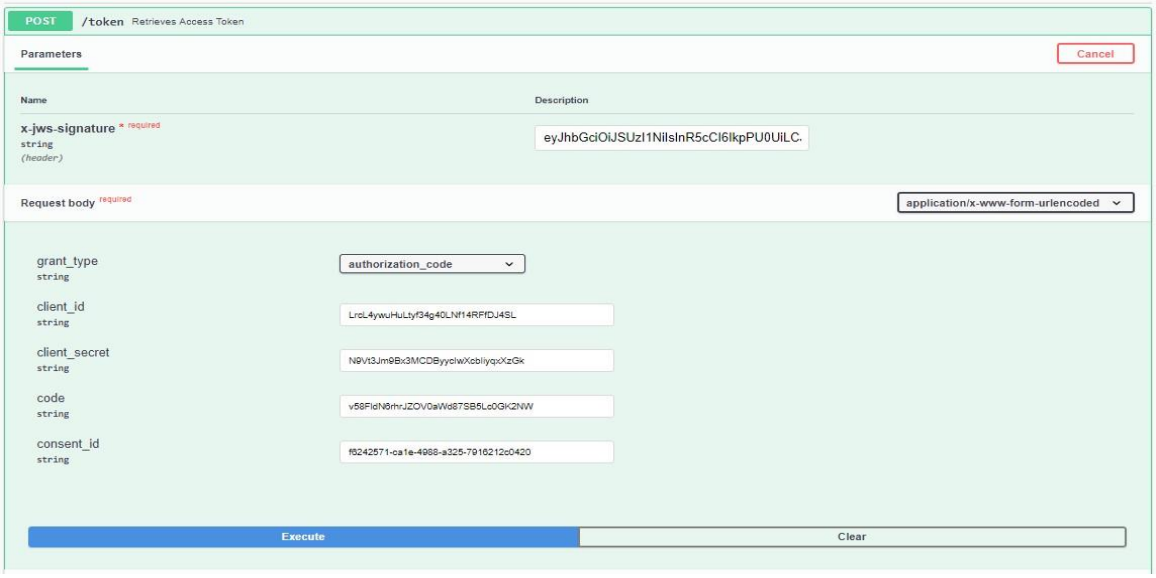


Figure 8

It is necessary to fill client Id and client secret (see the constant values in 1.5 Message Signing - Oauth2), authorization code (retrieved from the previous step) and the consent id (retrieved from the response body of Payment Initiation Request service).

Steps

Compute the ***Base64URL (JWS Payload):***

1. Create the following JSON on a single line without spaces:

```
{"headers":{},"payload":{"grant_type":"authorization_code","client_id":"LrcL4ywuHuLtyf34g40LNf14RFfDJ4SL","client_secret":"N9Vt3Jm9Bx3MCDByycIwXcbIiyqxXzGk","code":"v58FldN6rhrJZOV0aWd87SB5Lc0GK2NW","consent_id":"f6242571-ca1e-4988-a3257916212c0420"}}
```

2. Apply SHA-256 on the JSON from step 1 using the following command (on Windows OS):

```
echo|set /p="{"headers":{},"payload":{"grant_type":"authorization_code","client_id":"LrcL4ywuHuLtyf34g40LNf14RFfDJ4SL","client_secret":"N9Vt3Jm9Bx3MCDByycIwXcbIiyqxXzGk","code":"v58FldN6rhrJZOV0aWd87SB5Lc0GK2NW","consent_id":"f6242571-ca1e-4988-a3257916212c0420"}}" | openssl dgst -sha256
```

The result will be:

```
df441dff105ebef9057282baf87ecedba6b1f4b909e17016f5181fea4ca4254e
```

3. Create the following JSON with the result:

```
{"SHA256":"df441dff105ebef9057282baf87ecedba6b1f4b909e17016f5181fea4ca4254e"}
```

4. Compute Base64 encoding on the last JSON using the following command (on Windows OS):

```
echo|set
/p="{"SHA256":"df441dff105ebef9057282baf87ecedba6b1f4b909e17016f5181fea4ca4254e
"}" | openssl base64 -e –A
```

5. Replace any occurence of “+" character with “-" and any occurence of "/" character with “_". Also, delete every “=" from the resulted string. The result in our test case is:

```
eyJTSEEyNTYiOiJkZjQ0MWRmZjEwNWViZWY5MDU3MjgyYmFmODdlY2VkYmE2YjFmNGI5M
DllMTcwMTZmNTE4MWZlYTRjYTQyNTRlIn0
```

Compute the ***JWS-signature:***

1. Compute the following string: *Base64URL(JWS Header) '.' Base64URL(JWS Payload)*

   The result will be:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjM
zUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJkZjQ0MWRmZjEwNWViZWY5MDU3Mjg
yYmFmODdlY2VkYmE2YjFmNGI5MDllMTcwMTZmNTE4MWZlYTRjYTQyNTRlIn0
```

2. Sign the string using the TPP private key and apply Base64 encoding using the following command:

```
echo|set
/p="eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjOD
ZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJkZjQ0MWRmZjEwNWViZWY5MDU3
MjgyYmFmODdlY2VkYmE2YjFmNGI5MDllMTcwMTZmNTE4MWZlYTRjYTQyNTRlIn0" | openssl
dgst -sha256 -sign SC_EXEMPLU_SRL.key | openssl base64 -e -A
```

   The result will be:

```
XTk2vmLJBt1eaGy6FJQ2dqZBrq2mo8u2HDfEDy54TeYsyxqnVdlHLiJvCBSGG-
BR6ufJxPYVOxiLN9DdSMzGbRBqnU66qKDN-_ZB6xRnWiNk-LlTbH_aNjeIHAeMSEllNaPXOV-
L4gIhoiPzgEWlVZz1PnKZePhZpC1xrWr6nMVERq9DyYWOeNrgCGoLWeIP3lIQpPuI7Qv8fOdhnh
oi
2Osr9GrrWgklKq1qqIgE3KLo2Eo2HR_mZGpJnXge1IIAx5BMg3Ho_dG7a6kdvaNxy_Iy6Swe-
```

XyeBSQgeNTVGetdt8GkLHonJJ6VYBWvY0O6QWSNWIu2Y3Wv8GVAlBLg

3. Finally, the Header JWS-signature will be:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJkZjQ0MWRmZjEwNWViZWY5MDU3MjgyYmFmODdlY2VkYmE2YjFmNGI5MDllMTcwMTZmNTE4MWZlYTRjYTQyNTRlIn0.XTk2vmLJBt1eaGy6FJQ2dqZBrq2mo8u2HDfEDy54TeYsyxqnVdlHLiJvCBSGG-BR6ufJxP-YVOxiLN9DdSMzGbRBqnU66qKDN-_ZB6xRnWiNk-LlTbH_aNjeIHAeMSEIlNaPXOV-L4gIhoiPzgEWlVZz1PnKZePhZpC1xrWr6nMVERq9DyYWOeNrgCGoLWeIP3lIQpPuI7Qv8fOdhnhoi
2Osr9GrrWgklKq1qqIgE3KLo2Eo2HR_mZGpJnXge1IIAx5BMg3Ho_dG7a6kdvaNxy_Iy6Swe-XyeBSQgeNTVGetdt8GkLHonJJ6VYBWvY0O6QWSNWIu2Y3Wv8GVAlBLg


```
{
 "refresh_token": "slQPQ6G207vrqXbJMRJwj0kzrrxorqM2",
 "token_type": "bearer",
 "access_token": "9lgteBOPc4HG2ncyOulUo8DZeFctEk1I",
 "expires_in": 7776000
}
```


Request Response

Upon successful exchange of authorization code for access token the e-Commerce site will be able to call the API for checking the status of a consent or verify accounts information.

The e-Commerce application should build the next API request with the presence of access token in the header of HTTP request.

For testing with the sandbox it is necessary to use the bearer Auth token service (Figure 10). From the available authorization list we choose the bearer Auth service which will have the value of the access_token from the request response (9lgteBOPc4HG2ncyOulUo8DZeFctEk1I).

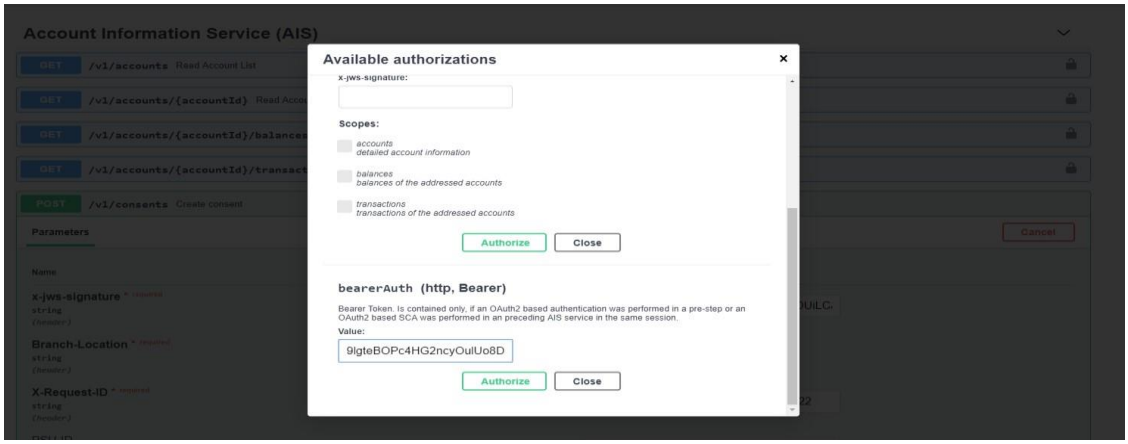


Figure 9

Fill the value with the access token value received in the previous call. This operation will ensure the presence of access token in the HTTP header of the subsequent API requests (Figure 11).

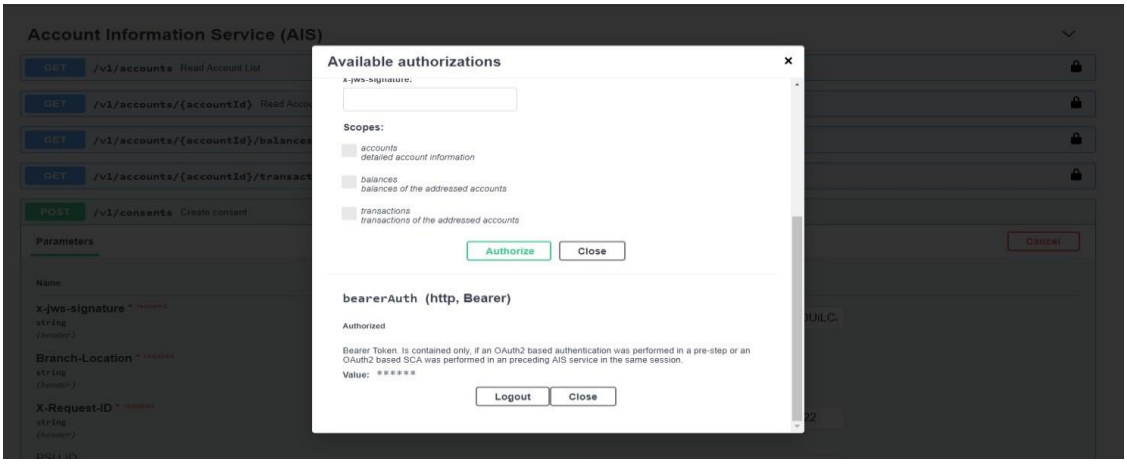


Figure 10

Checking the status of consent

Service Example (Consent status request)

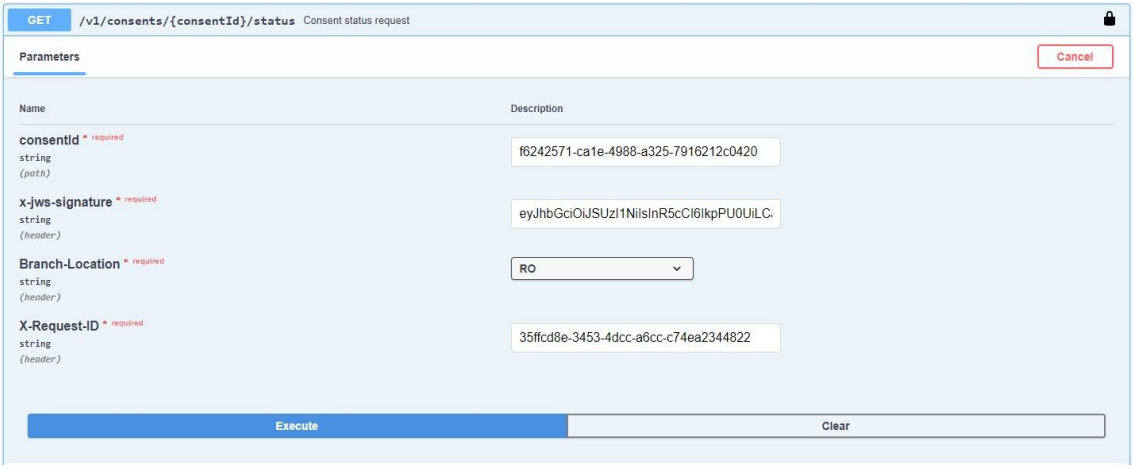


Figure 11

Values:

- "access_token": "9lgteBOPc4HG2ncyOulUo8DZeFctEk1I"
- "X-Request-ID":"35ffcd8e-3453-4dcc-a6cc-c74ea2344822"
- "consentId": "f6242571-ca1e-4988-a325-7916212c0420"
- "resourceId": "5ce3b48b-9ff0-4f79-9d93-c91d811a3ab2"

Steps

Compute the ***Base64URL (JWS Payload):***

1. Create the following JSON on a single line without spaces:

```
{"headers":{"Branch-Location":"RO","X-Request-ID":"35ffcd8e-3453-4dcc-a6ccc74ea2344822"},"payload":{"consent_id":"f6242571-ca1e-4988-a325-7916212c0420"}}
```

2. Apply SHA-256 on the JSON from step 1 using the following command (on Windows OS):

```
echo|set /p="{"headers":{"Branch-Location":"RO","X-Request-ID":"35ffcd8e-3453-4dcc-a6ccc74ea2344822"},"payload":{"consent_id":"f6242571-ca1e-4988-a325-7916212c0420"}}" | openssl dgst -sha256
```

The result will be:

```
b60ff0b5ae4871425442156523f451f771379e7b618e294d324e70570833af1c
```

3. Create the following JSON with the result:

```
{"SHA256":" b60ff0b5ae4871425442156523f451f771379e7b618e294d324e70570833af1c"}
```

4. Compute Base64 encoding on the later JSON using the following command (on Windows OS):

```
echo|set
/p="{"SHA256":"b60ff0b5ae4871425442156523f451f771379e7b618e294d324e70570833af1c"}"
| openssl base64 -e –A
```

5. Replace any occurrence of “+" character with “-" and any occurrence of "/" character with “_". Also, delete every “=" from the resulted string. The result in our test case is:

```
eyJTSEEyNTYiOiJiNjBmZjBiNWFlNDg3MTQyNTQ0MjE1NjUyM2Y0NTFmNzcxMzc5ZTdiNjE4ZTI5N
GQzMjRlNzA1NzA4MzNhZjFjIn0
```

Compute the ***JWS-signature:***

1. Compute the following string: *Base64URL(JWS Header) '.' Base64URL(JWS Payload)*

   The result will be:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjM
zUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJiNjBmZjBiNWFlNDg3MTQyNTQ0MjE1Nj
UyM2Y0NTFmNzcxMzc5ZTdiNjE4ZTI5NGQzMjRlNzA1NzA4MzNhZjFjIn0
```

2. Sign the string using the TPP private key and apply Base64 encoding using the following command:

```
echo|set
/p="eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjOD
ZjMzUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJiNjBmZjBiNWFlNDg3MTQyNTQ0MjE
1NjUyM2Y0NTFmNzcxMzc5ZTdiNjE4ZTI5NGQzMjRlNzA1NzA4MzNhZjFjIn0" | openssl dgst
sha256 -sign SC_EXEMPLU_SRL.key | openssl base64 -e -A
```

   The result will be:

```
emGS3PGTQkC8Mb8Oj2xAgEe8CjGNOkf67nlMdXT4KyMmE146_kAK_SMS2dAkFVnYIaHvadu
Oe4ShcWBcwOSYr0V4yTggXRTW2pIEqdF2 qbxxU3HwKrB-
```

DIueXCh6Yq88CoHfePQSYIMVKjjIetWAvkMXuvBqtaCoQdtyh7T2pKygWF4RpC03iPxLXVczKLj
DUXsIb8bXMfa_b4bnDn_rjZ0l_ekO2eS88Vu8gHGHRNcigbePfelGdMONUccAimJBY9otGeg3bP
n8732GLolTe1qtSqp44sNGIvKdsdtQELcKwV62tYv3Zkg3VLjHWJ5R_Hp4TMLBvJM3AE3KvtU w

3. Finally, the Header JWS-signature will be:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiIxMzNjMTE0NzA3NDBkN2VkMzNjODZjM
zUwMWUzYWM4MjIxZmVjZTAzIn0.eyJTSEEyNTYiOiJiNjBmZjBiNWFlNDg3MTQyNTQ0MjE1Nj
UyM2Y0NTFmNzcxMzc5ZTdiNjE4ZTI5NGQzMjRlNzA1NzA4MzNhZjFjIn0.emGS3PGTQkC8Mb8
Oj2xAgEe8CjGNOkf67nlMdXT4KyMmE146_kAK_SMS2dAkFVnYIaHvaduOe4ShcWBcwOSYr0V
4yTggXRTW2pIEqdF2 qbxxU3HwKrB-
DIueXCh6Yq88CoHfePQSYIMVKjjIetWAvkMXuvBqtaCoQdtyh7T2pKygWF4RpC03iPxLXVczKLj
DUXsIb8bXMfa_b4bnDn_rjZ0l_ekO2eS88Vu8gHGHRNcigbePfelGdMONUccAimJBY9otGeg3bP
n8732GLolTe1qtSqp44sNGIvKdsdtQELcKwV62tYv3Zkg3VLjHWJ5R_Hp4TMLBvJM3AE3KvtU w

{
"consentStatus": "valid"
}

Request Response

Also the following services are available for calling:

- Read accounts list
- Read account details
- Read account balance
- Read account transactions
- Read the content of a consent object
- Delete consent

4. Consent resource initiation

Register a consent resource

4.1. Resource Information

The resource information is as follows:

Method	Purpose
Response formats	JSON
Requires authentication	No
Rate limited	Yes
Requests	15

4.2. Request

The HTTP method and URL is as follows:

Method	URL
POST	https://sandbox.banorientfrance.com/v1/consent

4.1. Parameters

Header Parameter	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory
X-Request-ID	Mandatory
PSU-ID	Optional
PSU-ID-Type	Optional
PSU-Corporate-ID	Optional
PSU-Corporate-ID-Type	Optional
TPP-Redirect-Preferred	Optional
TPP-Redirect-URI	Optional
TPP-Nok-Redirect-URI	Optional
TPP-Explicit-Authorisation-Preferred	Optional

```
{
```

```
"access": {
        "balances": [],
        “transactions”: []
        },
 "recurringIndicator": true,
        "validUntil": “2019-11-01”,
 “frequencyPerDay”: “4”
}
```

Request Body

```
{
 "consentStatus": "received",
 "consentId": " f6242571-ca1e-4988-a325-7916212c0420",
 "_links": {
  "scaOAuth": "https://sandbox.banorientfrance.com/services/startAuthorize"
 }
}
```

Request Response

The response includes the consentId resource created and the URL link for the user authentication/authorization step.

5. Retrieves access token

This service exchanges the authorization code for access token and is the final step of OAuth2 authorization code flow.

For complete description of the OAuth2 flow please follow section 4 Testing a consent flow example.

5.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes
Rate limited	Yes
Requests	15

5.2. Request

Method	URL
POST	https://sandbox.banorientfrance.com/token

5.1. Parameters

Header Parameter	Required
x-jws-signature	Mandatory

5.2. Request Body

Parameter	Required
grant_type	Mandatory
client_id	Mandatory
client_secret	Mandatory
code	Mandatory
consent_id	Mandatory


```
{
 "refresh_token": "slQPQ6G207vrqXbJMRJwj0kzrrxorqM2",
 "token_type": "bearer",
 "access_token": "9lgteBOPc4HG2ncyOulUo8DZeFctEk1I",
 "expires_in": 7776000
}
```


Request Response

The response includes the access token, the refresh token and the expiration period.

6. Read accounts list

This service will return a list that contains the information of the accounts for which the consent has been given.

6.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes
Rate limited	Yes
Requests	15

6.2. Request

Method	URL
GET	https://sandbox.banorientfrance.com/v1/accounts

6.3. Parameters

Query Parameter	Required
With Balance	Optional

Header Parameter	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory
X-Request-ID	Mandatory
Consent-ID	Mandatory

No Request Body


```
{
 "accounts": [
  {
   "resourceId": "5ce3b48b-9ff0-4f79-9d93-c91d811a3ab2",
   "iban": "RO49AAAA1B31007593840000",
   "name": "Popescu Ion",
   "accountType": "*",
   "currency": "EUR",
```

```
    "_links": {
     "balances": {
      "href": "/v1/accounts/5ce3b48b-9ff0-4f79-9d93-c91d811a3ab2/balances"
     },
     "transactions": {
      "href": "/v1/accounts/5ce3b48b-9ff0-4f79-9d93-c91d811a3ab2/transactions"
     }
    }
   },
   {
    "resourceId": "a0edd571-eb32-4d53-a937-02086c02eea4",
    "iban": "RO49AAAA1B31007593840000",
    "name": "Popescu Ion",
    "accountType": "*",
    "currency": "RON",
    "_links": {
     "balances": {
      "href": "/v1/accounts/a0edd571-eb32-4d53-a937-02086c02eea4/balances"
     },
     "transactions": {
      "href": "/v1/accounts/a0edd571-eb32-4d53-a937-02086c02eea4/transactions"
     }
    }
   }
  ]
}
```


Request Response

7. Read account details

This service will return details of an account.

7.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes
Rate limited	Yes
Requests	15

7.2. Request

Method	URL
GET	https://sandbox.banorientfrance.com/v1/accounts/{accountId}

7.3. Parameters

Query Parameter	Required
With Balance	Optional

Path Parameter	Required
accountId	Mandatory

Header Parameter	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory
X-Request-ID	Mandatory
Consent-ID	Mandatory

No Request Body


```
{
 "account": {
  "resourceId": "5ce3b48b-9ff0-4f79-9d93-c91d811a3ab2",
  "iban": "RO49AAAA1B31007593840000",
  "name": "Popescu Ion",
  "accountType": "*",
  "currency": "EUR",
  "_links": {
   "balances": {
    "href": "/v1/accounts/5ce3b48b-9ff0-4f79-9d93-c91d811a3ab2/balances"
   },
   "transactions": {
```

```
    "href": "/v1/accounts/5ce3b48b-9ff0-4f79-9d93-c91d811a3ab2/transactions"
   }
  }
 }
}
```


Request Response

8. Read account balances

This service will return the balances of an account.

8.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes
Rate limited	Yes
Requests	15

8.2. Request

Method	URL
GET	https://sandbox.banorientfrance.com/v1/accounts/{accountId}/balances

8.3. Parameters

Path Parameters

Parameter	Required
accountId	Mandatory

Header Parameters

Name	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory

X-Request-ID	Mandatory
Consent-ID	Mandatory

No Request Body


```
{
 "account": {
  "iban": "RO49AAAA1B31007593840000"
 },
 "balances": [
  {
   "balanceAmount": {
    "currency": "RON",
    "amount": "163079.18"
   },
   "balanceType": "closingBooked",
   "date": "11-05-2018"
  }
 ]
}
```


Request Response

9. Read account transactions

This service will return the transactions of an account.

9.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes
Rate limited	Yes

Requests	15

9.2. Request

Method	URL
GET	https://sandbox.banorientfrance.com/v1/accounts/{accountId}/transactions

9.3. Parameters

Path Parameter	Required
accountId	Mandatory

Query Parameter	Required
Date From	Mandatory
Date To	Mandatory
Entry Reference From	Optional
Booking Status	Mandatory
Delta List	Optional
With Balance	Optional

Header Parameter	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory
X-Request-ID	Mandatory
Consent-ID	Mandatory
Accept	Optional

No Request Body


```
{
 "account": {
  "iban": "RO49AAAA1B31007593840000",
  "bban": null,
  "pan": null,
  "maskedPan": null,
  "msisdn": null,
  "currency": null
 },
 "transactions": {
  "booked": [
   {
    "transactionId": "5656930",
    "creditorName": "John Miles",
    "creditorAccount": {
     "iban": "RO49AAAA1B31007593840000",
     "bban": null,
     "pan": null,
     "maskedPan": null,
     "msisdn": null,
     "currency": null
    },
    "transactionAmount": {
     "currency": "EUR",
     "amount": "-200"
    },
```

"bookingDate": "25-01-2019",

```
      "valueDate": "25-01-2019",
      "remittanceInformationUnstructured": "Example booked 89674532 1"
    },
    {
      "transactionId": "4782038",
      "creditorName": "",
      "creditorAccount": {
        "iban": "",
        "bban": null,
        "pan": null,
        "maskedPan": null,
        "msisdn": null,
        "currency": null
      },
      "transactionAmount": {
        "currency": "EUR",
"amount": "300.9"
      },
      "bookingDate": "30-01-2019",
      "valueDate": "30-01-2019",
      "remittanceInformationUnstructured": "Example booked 89674532 2"
    }
  ],
  "pending": [
    {
      "transactionId": "478200",
      "creditorName": "John Miles",
      "creditorAccount": {
        "iban": "RO49AAAA1B31007593840000",
        "bban": null,
        "pan": null,
        "maskedPan": null,
        "msisdn": null,
        "currency": null
      },
      "transactionAmount": {
        "currency": "EUR",
        "amount": "-600"
      },
      "bookingDate": "",
      "valueDate": "12-02-2019",
      "remittanceInformationUnstructured": "Example pending 89674532 3"
    },
    {
      "transactionId": "369145",
```

```
      "creditorName": "",
      "creditorAccount": {
        "iban": "",
        "bban": null,
        "pan": null,
        "maskedPan": null,
        "msisdn": null,
        "currency": null
      },
      "transactionAmount": {
        "currency": "EUR",
        "amount": "470"
      },
      "bookingDate": "",
      "valueDate": "11-02-2019",
      "remittanceInformationUnstructured": "Example pending 89674532 4"
    }
  ],
  "_links": {
    "account": {
      "href": "/v1/accounts/c828504b-8a79-4d95-a2b1-7fbe0cdcbfc7"
    }
  }
 }
}
```


Request Response

10. Retrieve the consent request

This service will return the consent request.

10.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes
Rate limited	Yes

Requests	15

10.2. Request

10.3.

Method	URL
GET	https://sandbox.banorientfrance.com/v1/consents/{consentId}

Parameters

Path Parameter	Required
consentId	Mandatory

Header Parameter	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory
X-Request-ID	Mandatory

No Request Body

```
{
 "access": {
```

"accounts": [

```
  {
    "iban": "RO49AAAA1B31007593840000",
    "bban": null,
    "pan": null,
    "maskedPan": null,
    "msisdn": null,
    "currency": null
  },
  {
    "iban": "RO49AAAA1B31007593840000",
    "bban": null,
    "pan": null,
    "maskedPan": null,
    "msisdn": null,
    "currency": null
  }
 ],
 "balances": [
  {
    "iban": "RO49AAAA1B31007593840000",
    "bban": null,
    "pan": null,
    "maskedPan": null,
    "msisdn": null,
    "currency": null
  },
  {
    "iban": "RO49AAAA1B31007593840000",
    "bban": null,
    "pan": null,
    "maskedPan": null,
    "msisdn": null,
    "currency": null
  }
 ],
 "transactions": [
  {
    "iban": "RO49AAAA1B31007593840000",
    "bban": null,
    "pan": null,
    "maskedPan": null,
    "msisdn": null,
    "currency": null
  },
  {
```

```
      "iban": "RO49AAAA1B31007593840000",
      "bban": null,
      "pan": null,
      "maskedPan": null,
      "msisdn": null,
      "currency": null
    }
  ],
  "availableAccounts": null,
  "availableAccountsWithBalance": null,
  "allPsd2": null
 },
 "recurringIndicator": true,
 "validUntil": "2019-11-01",
 "frequencyPerDay": 4,
 "lastActionDate": "2019-03-08",
 "consentStatus": "valid"
}
```


Request Response

11. Delete consent

This service will delete the consent.

11.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes
Rate limited	Yes
Requests	15

11.2. Request

Method	URL
DELETE	https://sandbox.banorientfrance.com/v1/consents/{consentId}

11.3. Parameters

Path Parameter	Required
consentId	Mandatory

Header Parameter	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory
X-Request-ID	Mandatory

The *Request Response* in case of HTTP code 204 *is No content.*

The *Request Status* will be indicated by the status of the response.

12. Retrieve consent status

This service will return the status of the consent.

12.1. Resource Information

Method	Purpose
Response formats	JSON
Requires authentication	Yes

Rate limited	Yes
Requests	15

12.2. Request

Method	URL
GET	/v1/consents/{consentId}/status

12.3. Parameters

Path Parameter	Required
consentId	Mandatory

Header Parameter	Required
x-jws-signature	Mandatory
Branch-Location	Mandatory
X-Request-ID	Mandatory


```
{
    “consentStatus”: “valid”
}
```


Request Response