PSD2 Fallback API - PISP access documentation

Access & Identification of TPP

Base URL

```
https://pisp.tech26.de
```

On-boarding of new TPPs

1. A TPP shall connect to the N26 PSD2 API by using an eIDAS valid certificate (QWAC) issued
2. N26 shall check the QWAC certificate in an automated way and allow the TPP to identify themselves with the subsequent API calls
3. As the result of the steps above, the TPP should be able to continue using the API without manual involvement from the N26 side

Certificates can be renewed by making an API call **using the new certificate**, which will then be onboarded automatically.

Characteristics of this solution

Identification of TPP through QWAC

Please use your QWAC certificate when making any request on `pisp.tech26.de`.

Data which needs to be stored & provided

Third parties will need to store following data in order to access the API:

- device token

Additionally, Third parties are obliged to send following data in every request to our API:

- device token : `device-token: {{device_token}}`
- real user ip: `x-tpp-userip: {{userip}}`

TPP should provide a unique `device_token` per client/device connection. `device_token` must be a valid UUID v4 as per RFC 4122.

Importance of `x-tpp-userip`

All payment initiation-related API calls must include the IP address of the end-user in the `x-tpp-userip` header. From a technical point of view, providing this IP address **guarantees us to identify proper user calls** and to **protect the integrity of our services**.

Since our Fallback API is based on our original API for our mobile and web apps, there are a lot of security measures built-in. Many of these security measures track the end-user IP addresses and prevent frequent calls from different IP addresses for the same user (amongst other features, that we do not disclose for security reasons). If a TPP doesn't provide an end-user's IP address with API requests requiring IP specification, or tries to manipulate or obfuscate IP addresses, such cases will be treated in accordance to our established security policies applied to the original API.

According to PSD2 (Level1) Art. 94 (1) we require the end-user IP to be specified with every end-user generated request. Monitoring end-user IPs on the N26 side is necessary to safeguard the prevention, investigation and detection of payment fraud. In case N26 security measures applied to the API detect unusual user-related activity, such cases will be processed in accordance with the established security policies, including (but not limited to) rate limiting, blocking request by IP and reporting such cases to the regulatory authorities.

Data which should not be stored

As per Art 22 (1), (2b) and Art 33(5a) of Directive (EU) 2015/2366 of the European Parliament and of the Council with regard to regulatory technical standards for strong customer authentication and common and secure open standards of communication TPPs should **NOT store passwords (incl. PINs)** of users!

> *CHAPTER IV*
>
> *CONFIDENTIALITY AND INTEGRITY OF THE PAYMENT SERVICE USERS' PERSONALISED SECURITY CREDENTIALS*
>
> *Article 22*
>
> *1.*
>
> *Payment service providers shall ensure the confidentiality and integrity of the personalised security credentials of the payment service user, including authentication codes, during all phases of the authentication.*
>
> *2.*
>
> *For the purpose of paragraph 1, payment service providers shall ensure that each of the following requirements is met:*
>
> *a) […]*
>
> *b) personalised security credentials in data format, as well as cryptographic materials related to the encryption of the personalised security credentials are not stored in plain text;*

> *Section 2*
>
> *Specific requirements for the common and secure open standards of communication*
>
> *Article 33*
>
> *Contingency measures for a dedicated interface*
>
> *[…]*
>
> *5.*
>
> *For this purpose, account servicing payment service providers shall ensure that the payment service providers referred to in Article 30(1) can be identified and can rely on the authentication procedures provided by the account servicing payment service provider to the payment service user. Where the payment service providers referred to in Article 30(1) make use of the interface referred to in paragraph 4 they shall:*
>
> *a) take the necessary measures to ensure that they do not access, store or process data for purposes other than for the provision of the service as requested by the payment service user;*

If we identify the TPP is doing this, we reserve the right to block those accesses.

Validity of access tokens

	Access Token
Purpose	Access for API calls in **one session**
How to get	1. Email & password 2. **Push** (OOB) or SMS (OTP)
Validity	15 min
Storage	NEVER

⚠ Access tokens are supposed to be used only for **1 session (sequence of calls)**.

If a user requests initiation of a new payment a new access token has to be requested **EVEN** if the original access token is still valid.

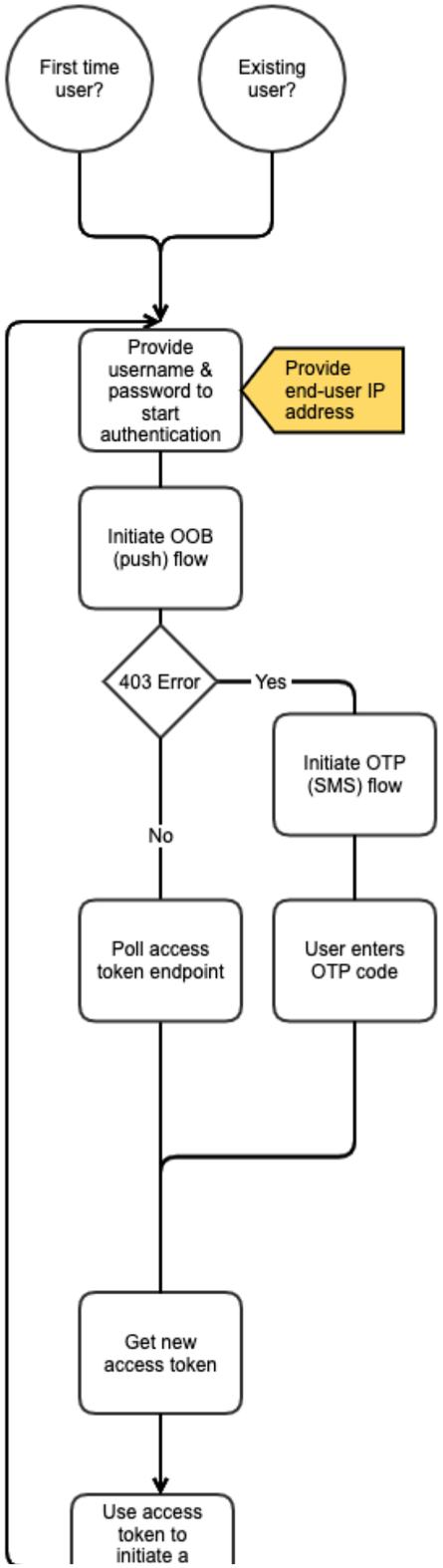
For this reason the TPP should **NEVER** store the access token.

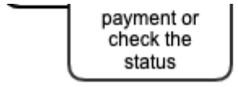
⚠ The TPP should not use these access tokens on other base URLs than `pisp.tech26.de`.

⛔ If those policies above are not respected, there is no guarantee you will not be rate-limited.

User Authentication

Overview


First time user?
Existing user?
Provide username & password to start authentication
Provide end-user IP address
Initiate OOB (push) flow
403 Error
Yes
No
Initiate OTP (SMS) flow
Poll access token endpoint
User enters OTP code
Get new access token
Use access token to initiate a



Step 1 - Start authentication with username & password

Request

```
POST /oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
device-token: {{device_token}}
x-tpp-userip: {{userip}}

username={{username}}&password={{password}}&grant_type=password
```

Parameters

`device_token` is a unique identifier of device. Has to be the same per installation.

For syncs from the backend, this has to be unique and persisted per user.

`userip` has be populated with the real user ip.

Only the headers mentioned here are necessary

Responses

Successful

```
HTTP/1.1 403 Forbidden
{
  "status": 403,
  "error": "mfa_required",
  "mfaToken": {{mfaToken}},
  "hostUrl": "{{hostUrl}}",
  "detail": "mfa_required",
  "userMessage": {
    "title": "MFA token is required",
    "detail": "MFA token is required"
  }
}
```

Parameters

mfaToken

`mfaToken` is unique token of this login try. If is required for the subsequent Authentication requests.

Invalid credentials/username does not exist

```
HTTP/1.1 400 Forbidden
{
  "error": "invalid_grant",
  "error_description": "Bad credentials",
  "status": 400,
  "detail": "Bad credentials",
  "userMessage": {
    "title": "Login failed",
    "detail": "Incorrect user name or password! Please, try again"
  }
}
```

User was rate-limited

```
HTTP/1.1 429 Too Many Requests
{
  "error": "too_many_requests",
  "error_description": "Too many log-in attempts. Please try again in 30 minutes.",
  "status": 429,
  "detail": "Too Many Requests",
  "userMessage": {
    "title": "Too Many Requests",
    "detail": "Too many log-in attempts. Please try again in 30 minutes."
  }
}
```

No x-tpp-userip header was provided

```
HTTP/1.1 451 Unavailable For Legal Reasons
{
  "error": "Oops!",
  "status": 451,
  "detail": "Please try again later.",
  "userMessage": {
    "title": "Oops!",
    "detail": "Please try again later."
   },
}
```

Step 2a - Recommended - Continue authentication with Two-Step Certification (OOB)

This type of authentication triggers a second factor authentication notification on a user mobile device.

Request

```
POST /api/mfa/challenge HTTP/1.1
Content-Type: application/json
device-token: {{device_token}}
x-tpp-userip: {{userip}}

{
   "mfaToken": "{{mfaToken}}",
   "challengeType": "oob"
}
```

Responses

Successful

```
HTTP/1.1 200 OK
{
    "challengeType": "oob"
}
```

User received a push notification with the request to authorise this connection.

Unsuccessful

In case `device_token` is incorrect or `mfa_token` is incorrect.

```
HTTP/1.1 400 Bad Request
{
  "error": "invalid_grant",
  "error_description": "Bad credentials",
  "status": 400,
  "detail": "Bad credentials",
  "userMessage": {
    "title": "Login failed",
    "detail": "Session has expired or is not valid! Please, try again"
  }
}
```

User does not have a paired device

```
HTTP/1.1 403 Forbidden
{
  "error": "invalid_state",
  "error_description": "Invalid state to start the challenge",
  "status": 403,
  "detail": "Invalid state to start the challenge",
  "userMessage": {
    "title": "Login failed",
    "detail": "Invalid state to start the challenge"
  }
}
```

In this case, please proceed to the OTP flow.

Step 2b - Not Recommended - Continue authentication with Two-Step SMS (OTP)

⚠️ Use this step only as a fallback to Step 2a

Reasons why SMS is not recommended:

- SMS delivery rate is not 100%
- SMS delivery takes time
- We limit how many SMS a user can use
- Users might have changed the phone number and not informed us

Request

```
POST /api/mfa/challenge HTTP/1.1
Content-Type: application/json
device-token: {{device_token}}
x-tpp-userip: {{userip}}

{
   "mfaToken": "{{mfaToken}}",
   "challengeType": "otp"
}
```

Responses

Successful

```
HTTP/1.1 201 Created
{
    "challengeType": "otp",
    "remainingResendCodeCount": {{remainingResendCodeCount}},
    "waitingTimeInSeconds": {{waitingTimeInSeconds}},
    "obfuscatedPhoneNumber": "{{obfuscatedPhoneNumber}}"
}
```

An SMS was sent to the user with the request to authorise this connection.

Parameters

`remainingResendCodeCount` (int) - amount of remaining resend attempts

`waitingTimeInSeconds` (int) - amount of time the client needs to wait before the next SMS send request

`obfuscatedPhoneNumber` - the phone number where the SMS has been sent to, e.g. "+49******0285"

Successful resent the SMS

```
HTTP/1.1 200 Ok
{
    "challengeType": "otp",
    "remainingResendCodeCount": {{remainingResendCodeCount}},
    "waitingTimeInSeconds": {{waitingTimeInSeconds}},
    "obfuscatedPhoneNumber": "{{obfuscatedPhoneNumber}}"
}
```

An SMS was resent to the user with the request to authorise this connection.

Parameters

`remainingResendCodeCount` (int) - amount of remaining resend attempts

`waitingTimeInSeconds` (int) - amount of time the client needs to wait before the next SMS send request

`obfuscatedPhoneNumber` - the phone number where the SMS has been sent to, e.g. "+49******0285"

SMS sent successfully less than 30 seconds ago

```
HTTP/1.1 204 No content
```

Unsuccessful

In case `device_token` is incorrect or `mfa_token` is incorrect.

```
HTTP/1.1 400 Forbidden
{
  "error": "invalid_grant",
  "error_description": "Bad credentials",
  "status": 400,
  "detail": "Bad credentials",
  "userMessage": {
    "title": "Login failed",
    "detail": "Session has expired or is not valid! Please, try again"
  }
}
```

SMS retry limit is reached

```
HTTP/1.1 429 Too Many Requests
{
  "error": "too_many_sms",
  "error_description": "Too many SMS have been sent. Please try again in 1 day.",
  "status": 429,
  "detail": "Too Many SMS",
  "userMessage": {
    "title": "Too Many SMS",
    "detail": "Too many SMS have been sent. Please try again in 1 day."
  }
}
```

Step 3 - Get access token

The TPP should poll endpoints in this section not more than every 2 seconds.

After a successful, expired or unauthorised response the polling should stop.

Request for OOB

```
POST /oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
device-token: {{device_token}}
x-tpp-userip: {{userip}}

mfaToken={{mfaToken}}&grant_type=mfa_oob
```

Responses for OOB

Successful

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

User has not yet provided authorisation

```
HTTP/1.1 400 Bad Request
{
  "error": "authorization_pending",
  "error_description": "MFA token was not yet confirmed",
  "status": 400,
  "detail": "MFA token was not yet confirmed",
  "userMessage": {
    "title": "Login failed",
    "detail": "Authorisation request is not confirmed. Please, confirm it on your device and try again."
  }
}
```

Request for OTP

```
POST /oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
device-token: {{device_token}}
x-tpp-userip: {{userip}}

mfaToken={{mfaToken}}&otp={{otp}}&grant_type=mfa_otp
```

Parameters

`otp` is the one-time password that user will received as an SMS and has to enter in the flow.

Responses for OTP

Successful

```
HTTP/1.1 200 OK
{
    "access_token": "{{access_token}}",
    "token_type": "bearer",
    "expires_in": {{expires_in}},
    "scope": "trust",
    "host_url": "{{host_url}}"
}
```

User has provided the wrong code

```
HTTP/1.1 400 Bad Request
{
  "error": "invalid_otp",
  "error_description": "OTP is invalid",
  "status": 400,
  "detail": "OTP is invalid",
  "userMessage": {
    "title": "Invalid code",
    "detail": "Provided code is invalid. Please, try again."
  }
}
```

Unsuccessful

If `device_token` is incorrect, `mfa_token` is incorrect or expired in 5 minutes.

```
HTTP/1.1 400 Bad Request
{
  "error": "invalid_grant",
  "error_description": "Bad credentials",
  "status": 400,
  "detail": "Bad credentials",
  "userMessage": {
    "title": "Login failed",
    "detail": "Session has expired or is not valid! Please, try again"
  }
}
```

Too many code attempts (SMS must be resent)

```
HTTP/1.1 429 Too Many Requests
{
  "error": "too_many_attempts",
  "error_description": "Amount of the attempts has been exceeded. Please resend the SMS.",
  "status": 429,
  "detail": "Amount of the attempts has been exceeded. Please resend the SMS.",
  "userMessage": {
    "title": "Too many attempts",
    "detail": "Amount of the attempts has been exceeded. Please resend the SMS."
  }
}
```

Payment Initiation

Overview

In order to make requests to the payment initiation endpoints such as SEPA DT or Standing Orders it is necessary to supply **encrypted-secret** and **encrypted-pin** headers. These parameters can be obtained by encrypting the user PIN which TPPs shall ask from the user in the same manner as email and password during authentication.

⚠ A TPP shall not store the PIN provided by a user and should treat it the same way as a user's password. Please refer to the Data which should not be stored section for more details.

We provide an additional endpoint `/api/encryption/key` which should be used to obtain an encryption key which in turn is used to encrypt the user PIN.

In the following sections we provide the step-by-step guide on how to obtain the encryption key and use it to generate **encrypted-secret** and **encrypted-pin**.

User PIN Encryption

Step 1 - Getting User PIN

As the first step before initiating any payment a TPP needs to request a user to provide a PIN. PIN is the four digit passcode for confirming payment initiation. The PIN code should be treated the same way as user password and shouldn't be stored on the TPP side for future usage. In other words, whenever a TPP initiates a transactions, they should ask for the user input providing the PIN code every time.

Step 2 - Getting Encryption key

In order to generate `encrypted-secret` and `encrypted-pin` headers for payment initiation endpoints, the next step should be getting a public key to be used for PIN encryption.

Request

```
GET    /api/encryption/key HTTP/1.1
Authorization: bearer {{access_token}}
x-tpp-userip: {{userip}}
device-token: {{device_token}}
Content-Type: application/json
```

Response

```
{
  "publicKey": "..." # Base-64 encoded public key.
}
```

Example (Bash)

```
public_key=$(curl -k --key $CLIENT_KEY_FILE --cert $CLIENT_CERT_FILE -s -H "Authorization:Bearer $access_token" https://pisp.tech26.de/api/encryption/key | jq -r '.
publicKey')
```

Step 3 - Encrypting User PIN

After you obtained the public key from the previous step, the next step would be to encrypt the user PIN and get **encrypted-secret** and **encrypted-pin** as a result. A short walkthrough follows (with example code in Bash).

1. Generate an AES-256 key/IV of your choice, for example:

```
aes_secret_raw=$(openssl enc -nosalt -aes-256-cbc -pbkdf2 -k $AES_SECRET_KEY -P)
aes_key_hex=$(echo "$aes_secret_raw" | grep "key\s*=" | sed -e "s/.*=//")
aes_iv_hex=$(echo "$aes_secret_raw" | grep "iv\s*=" | sed -e "s/.*=//")
```

2. Encode this key and IV as a JSON string (collectively, this data is the secret) with the following format:

```
{
  "secretKey": "{key encoded as Base-64}",
  "iv": "{iv encoded as base-64}"
}
```

Bash:

```
aes_key_base64=$(echo $aes_key_hex | xxd -r -p | base64)
aes_iv_base64=$(echo $aes_iv_hex | xxd -r -p | base64)
unencrypted_aes_secret=$(jq -n --arg key "$aes_key_base64" --arg iv "$aes_iv_base64" '{secretKey: $key, iv: $iv}')
```

3. Encrypt the secret JSON with RSA using the provided public key. Note: Use PKCS#1 v1.5 padding; this is the default for openssl in the sample below, but may not be with other technologies. Then convert the encrypted secret to base-64:

```
rsa_public_key=$(echo -e "-----BEGIN PUBLIC KEY-----\n$public_key\n-----END PUBLIC KEY-----" | sed -r 's/(.{67})/\1\n/g')
mkdir -p $TEMP_FOLDER
echo "$rsa_public_key" > $TEMP_FOLDER/rsa_public_key.pem
encrypted_secret=$(echo $unencrypted_aes_secret | openssl rsautl -encrypt -inkey $TEMP_FOLDER/rsa_public_key.pem -pubin | base64 | tr -d '\n')
rm $TEMP_FOLDER/rsa_public_key.pem
```

4. Encrypt the user's PIN, as a string, with AES-256, using the key/IV generated in step 1, and convert this to base-64:

```
encrypted_pin=$(echo $PIN | tr -d '\n' | openssl enc -nosalt -aes-256-cbc -K $aes_key_hex -iv $aes_iv_hex | base64 | tr -d '\n')
```

As a result, you get `encrypted_secret` on Step 3 and `encrypted_pin` on Step 4. Starting from this point you can use the payment initiation endpoints described below.

We provide the full Bash script to encrypt the user PIN for your convenience in Appendix A. Feel free to use this sample script or come up with your alternative implementation in the language of your choice.

Step 4 - Initiate a payment

At this step you should have everything necessary to make a call to one of the payment initiation endpoints listed below. Please make sure to repeat the process of getting the user PIN, the new encryption key and generating the `encrypted_secret` and `encrypted_pin` with every new transaction, even for the same user.

Initiate SEPA (Debit Transfer) DT Transactions

Please make sure you've completed the steps from the User PIN encryption section before proceeding with the request.

⚠ SEPA transfers are available only for EU users. In order to identify if a user's legal entity is EU or UK, please refer to the Get Main Account information endpoint

```
POST    /api/transactions HTTP/1.1
Authorization: bearer {{access_token}}
x-tpp-userip: {{userip}}
device-token: {{device_token}}
encrypted-secret: {{encrypted-secret}}
encrypted-pin: {{user-pin}}
Content-Type: application/json


{
   "transaction":{
```

```
      "amount":"12.0",
      "partnerBic":"COBADEFXX", # transaction details as partner bic, iban, name
      "partnerIban":"DE12500105170648489890",
      "partnerName":"McDonalds",
      "referenceText":"McMenu",
      "type":"DT" # debit transfer type,
   }
}
```

Responses

Successful

```
{
  "id": "bc7170a7-725e-11e9-80f4-0242ac110004" # created Transaction id
}
```

Incorrectly formatted payload (e.g. missing field etc)

```
HTTP/1.1 400 Bad Request
{
        "timestamp": 1582910546847,
        "status": 400,
        "error": "Bad Request",
        "message": "Bad Request",
        "detail": "Bad Request"
}
```

Error validating PIN

```
HTTP/1.1 400 Bad Request
{
        "timestamp": 1582911454774,
        "status": 400,
        "error": "Bad Request",
        "message": "PIN validation failure",
        "detail": "Bad Request"
}
```

Other validation failures

```
HTTP/1.1 400 Bad Request
{
```

```
        "title": "Error",
        "message": "{Explanation}" # See below for examples.
}
```

Some example message values for validation errors include:

```
"The IBAN you've entered is not valid."
"The transaction amount should be greater than zero."
```

Server-side errors

```
HTTP/1.1 500 Internal Server Error
{
        "title": "Error",
        "message": "An unexpected error happened"
}
```

Initiate SEPA Standing Order (Recurring/Future Payments)

Please make sure you've completed the steps from the User PIN encryption section before proceeding with the request.

⚠ SEPA standing orders are available only for EU users. In order to identify if a user's legal entity is EU or UK, please refer to the Get Main Account information endpoint

Request

```
POST    /api/transactions/so HTTP/1.1
Authorization: bearer {{access_token}}
x-tpp-userip: {{userip}}
device-token: {{device_token}}
encrypted-secret: {{encrypted-secret}}
encrypted-pin: {{user-pin}}
Content-Type: application/json

{
   "standingOrder":{
      "amount":"12.0",
      "partnerIban":"ES2015632626323268851568",
      "partnerName":"Pancho Villa",
      "referenceText":"standing order for Dio",
      "nextExecutingTS":"1583452800000", # exact day in epoch millis, should be whole day in UTC
      "executionFrequency":"WEEKLY",
      "stopTS":"1593129600000", # optional, last execution timestamp in epoch millis, should be whole day in UTC
   }
}
```

Responses

Successful

```
{
  "id": "bc7170a7-725e-11e9-80f4-0242ac110004" # created Transaction id
}
```

Incorrectly formatted payload (e.g. missing field etc)

```
HTTP/1.1 400 Bad Request
{
        "timestamp": 1582910546847,
        "status": 400,
        "error": "Bad Request",
        "message": "Bad Request",
        "detail": "Bad Request"
}
```

Error validating PIN

```
HTTP/1.1 400 Bad Request
{
        "title": "Invalid confirmation PIN",
        "message": "Invalid confirmation PIN"
}
```

All other errors

```
HTTP/1.1 500 Internal Server Error
{
        "title": "Error",
        "message": "An unexpected error happened"
}
```

Get Status of Initiated Transactions

Overview

Access to lists of *user transactions* and *standing orders* is provided as a means of identifying whether a transaction has been successfully initiated. Since every transaction needs to be certified by a user via a mobile app, payment initiation endpoints listed in the previous section, return only a transaction ID and can't guarantee that the transaction will be booked. The result of the user confirmation (certification) of a transaction can be queried by making the GET transaction lists requests.

If a transaction is present on a list of transactions, it means that a user has accepted the certification and all validations have been completed.

- To check the **SEPA DT** transaction statuses, please use **Main Account Transaction List**: Each N26 user has only one payment account, which is the Main Account. All transactions successfully initiated and certified by a user appear on the Main Account Transaction List. Transaction list of the main account is provided via `/api/smrt/transactions`
- To check the **Standing Order** transaction status, please use **Standing Order Transaction List**: Approved Standing Orders don't appear on the list of Main Account Transaction List, but on the separate list of Standing orders provided via `/api/transactions/so` with a not null `"userCertified"`field.

For **Standing Orders**, don't forget to check the `"userCertified"`field

Not yet confirmed Standing Orders have `"userCertified": null`

Confirmed Standing Orders have not null `"userCertified"`

Get (Main) Account Transactions List

Request

```
GET    /api/smrt/transactions HTTP/1.1
Authorization: bearer {{access_token}}
x-tpp-userip: {{userip}}
device-token: {{device_token}}
```

Parameters

`limit` number of transactions, defaults to 20.

`lastId` the last transactionId, exclusive.

`from` from timestamp, Unixtime

`to` to timestamp, Unixtime

For instance:

```
GET /api/smrt/transactions?limit=20&lastId=c690c24d-9a19-4400-0001-6db5542c82d5&from=1570312800000&to=1570312800001
```

Response


```
[
  {
    "id": "b6255a9a-97bd-4453-b332-701ac576bd10",
    "userId": "fdd2d3eb-f16f-4aa1-9292-eac88ee356d5",
    "type": "PT",
    "amount": -5432.0,
    "currencyCode": "EUR",
    "originalAmount": -5432.0,
    "originalCurrency": "EUR",
    "exchangeRate": 1.0,
    "visibleTS": 1570705239000,
    "mcc": 6011,
    "mccGroup": 18,
    "recurring": false,
    "partnerAccountIsSepa": false,
    "accountId": "4badce07-0de0-420d-a648-d3ae3e2d54d5",
    "category": "micro-v2-atm",
    "cardId": "096b54f1-f9ca-4d7a-8e78-7b1cebd2edd0",
    "userCertified": 1570705240081,
    "pending": false,
    "transactionNature": "NORMAL",
    "transactionTerminal": "ATM",
```

```
    "createdTS": 1570705240090,
    "smartLinkId": "b6255a9a-97bd-4453-b332-701ac576bd10",
    "linkId": "b6255a9a-97bd-4453-b332-701ac576bd10",
    "confirmed": 1570705240081
  },
  {...}
]
```

Get (Main) Account Transactions Details

Request

```
GET    api/smrt/transactions/{transactionId} HTTP/1.1
Authorization: bearer {{access_token}}
x-tpp-userip: {{userip}}
device-token: {{device_token}}
```

Response

```
{
    "id": "b6255a9a-97bd-4453-b332-701ac576bd10",
    "userId": "fdd2d3eb-f16f-4aa1-9292-eac88ee356d5",
    "type": "PT",
    "amount": -5432.0,
    "currencyCode": "EUR",
    "originalAmount": -5432.0,
    "originalCurrency": "EUR",
    "exchangeRate": 1.0,
    "visibleTS": 1570705239000,
    "mcc": 6011,
    "mccGroup": 18,
    "recurring": false,
    "partnerAccountIsSepa": false,
    "accountId": "4badce07-0de0-420d-a648-d3ae3e2d54d5",
    "category": "micro-v2-atm",
    "cardId": "096b54f1-f9ca-4d7a-8e78-7b1cebd2edd0",
    "userCertified": 1570705240081,
    "pending": false,
    "transactionNature": "NORMAL",
    "transactionTerminal": "ATM",
    "createdTS": 1570705240090,
    "smartLinkId": "b6255a9a-97bd-4453-b332-701ac576bd10",
    "linkId": "b6255a9a-97bd-4453-b332-701ac576bd10",
    "confirmed": 1570705240081
}
```

Get Standing Order Transactions List

Request

```
GET    /api/transactions/so HTTP/1.1
Authorization: bearer {{access_token}}
x-tpp-userip: {{userip}}
device-token: {{device_token}}
```

Response

Not yet confirmed Standing Orders have `"userCertified": null`

Confirmed Standing Orders have not null `"userCertified"`


```
{
  "paging": {
    "previous": null,
    "next": null,
    "totalResults": 2
  },
  "data": [
    {
      "id": "172078ba-2c20-5351-bedd-2e40a5ee648d",
      "created": 1575888863557,
      "updated": 1577750403118,
      "amount": 0.99,
      "currencyCode": {
        "currencyCode": "EUR"
      },
      "partnerIban": "DE31100110012628943987",
      "partnerBic": "NTSBDEB1XXX",
      "partnerAccountIsSepa": true,
      "partnerAccountBan": "2628943987",
      "partnerBcn": "10011001",
      "userCertified": 1575888863554, # Check this field. 'null' means SO is not confirmed by user
      "userCanceled": null,
      "n26Iban": "DE91100110012625635983",
      "bankTransferTypeText": null,
      "firstExecutingTS": 1577750400000,
      "nextExecutingTS": null,
      "stopTS": 1577836800000,
      "referenceText": "Southside Order: NJ4EE",
      "partnerBankName": "N26 Bank",
      "partnerName": "Southside",
      "initialDayOfMonth": 31,
      "linkId": null,
      "internal": false,
      "referenceToOriginalOperation": null,
      "executionFrequency": "MONTHLY",
      "executionCounter": 1,
      "userId": "96baf720-441f-430e-8b7d-2e78b8712d59",
      "accountId": "d0ef2c77-a81e-443f-8a02-1ff44e88f135",
```

```
      "openBankingId": "cwr4vc2ss1p81di1hb",
      "tokenSubmissionId": "ss:68782ycyPgUCvY7FHddnkynukZ1dUmRhLMSWRLfYokkV:qXTTjgzAQHgC3Hyrn"
    }
  ]
}
```

Get (Main) Account Information

This is the main method for identifying if a user account is based on the EU or UK.

Request

```
GET    /api/accounts HTTP/1.1
Authorization: bearer {{access_token}}
x-tpp-userip: {{userip}}
device-token: {{device_token}}
```

Response for EU users

EU users include all countries in Europe with the exception of the UK, no matter if it is a German Iban or not. Countries such as Switzerland will also have EU as legal entity and thus support SEPA.

```
{
    "id": "4badce07-0de0-420d-a648-d3ae3e2d54d5",
    "physicalBalance": null,
    "availableBalance": 1044970.94,
    "usableBalance": 1044970.94,
    "bankBalance": 1044970.94,
    "iban": "DE15100110012627633320",
    "bic": "NTSBDEB1XXX",
    "bankName": "N26 Bank",
    "seized": false,
    "currency": "EUR",
    "legalEntity": "EU",
    "users": [
        {
            "userId": "fdd2d3eb-f16f-4aa1-9292-eac88ee356d5",
            "userRole": "OWNER"
        }
    ],
    "externalId": {
        "iban": "DE15100110012627633320"
    }
}
```

Response for UK users

Notice that for these users currency is also set as GBP and the IBAN is british.

```
{
    "id":"80ad5484-1d66-4922-96e3-9861405c8c3e",
    "physicalBalance":null,
    "availableBalance":99960.00,
    "usableBalance":99960.00,
    "bankBalance":99960.00,
    "iban":"GB56NTSB04002600001392",
    "bic":"NTSBDEB1XXX",
    "bankName":"N26 Bank",
    "seized":false,
    "currency":"GBP",
    "legalEntity":"UK",
    "users":[
        {
            "userId":"e4af5220-e9f5-4449-98cb-eff9f980d46c",
            "userRole":"OWNER"
        }
    ],
    "externalId":{
        "iban":"GB56NTSB04002600001392",
        "accountNumber":"00001392",
        "sortCode":"040026"
    }
}
```


Appendix A

Bash Script for User PIN Encryption and Initiating Transaction

```
#!/usr/bin/env bash

[ -z "$CLIENT_KEY_FILE" ] && echo "Please specify CLIENT_KEY_FILE (this is the location of your client cert's private key" && exit 1
[ -z "$CLIENT_CERT_FILE" ] && echo "Please specify CLIENT_CERT_FILE (this is the location of your client cert file" && exit 1
[ -z "$USER_NAME" ] && echo "Please specify USER_NAME (this is the email address of the N26 user)" && exit 1
[ -z "$PASSWORD" ] && echo "Please specify PASSWORD (this is the password of the N26 user)" && exit 1
[ -z "$AES_SECRET_KEY" ] && echo "Please specify AES_SECRET_KEY (this is used to generate an example AES secret key/iv)" && exit 1
[ -z "$PISP_HOST" ] && echo "PISP_HOST not specified (this is the location of the PISP access host), defaulting to pisp.tech26.de" && export PISP_HOST=pisp.tech26.
de
[ -z "$TEMP_FOLDER" ] && echo "Please specify TEMP_FOLDER (this is a folder which this script can create and destroy, and use to store temporary files)" && exit 1
[ -z "$PIN" ] && echo "Please specify PIN (this is the user's PIN, as a plain string)" && exit 1

# Log in and get an access token

token_json=$(curl -k --key $CLIENT_KEY_FILE --cert $CLIENT_CERT_FILE -s -X POST --data-urlencode 'grant_type=password' --data-urlencode "username=$USER_NAME" --
data-urlencode "password=$PASSWORD" https://${PISP_HOST}/oauth2/token)

access_token=$(echo $token_json | jq .access_token | sed "s/^null$//" | tr -d '"')

if [ -z "$access_token" ]; then
```

```
# Assume MFA if token not immediately available.

mfa_token=$(echo $token_json | jq .mfaToken | sed "s/^null$//" | tr -d '"')

echo "MFA token: $mfa_token"

curl -k --key $CLIENT_KEY_FILE --cert $CLIENT_CERT_FILE -s -X POST -H 'Authorization:Basic YW5kcm9pZDpzZWNyZXQ=' -H "Content-Type: application/json" --data
'{"mfaToken": "'$mfa_token'", "challengeType":"oob"}' https://$PISP_HOST/api/mfa/challenge
echo
echo "Please confirm on your paired phone.."

read -n 1 -p "Press enter when done:" mainmenuinput

access_token=$(curl -k --key $CLIENT_KEY_FILE --cert $CLIENT_CERT_FILE -s -X POST --data-urlencode 'grant_type=mfa_oob' --data-urlencode "mfaToken=$mfa_token"
https://$PISP_HOST/oauth2/token | jq .access_token | tr -d '"' )

fi

echo "Access token: $access_token"

# Get a public key from the encryption key endpoint.

public_key=$(curl -k --key $CLIENT_KEY_FILE --cert $CLIENT_CERT_FILE -s -H "Authorization:Bearer $access_token" https://${PISP_HOST}/api/encryption/key | jq -r '.
publicKey')

echo "Public key: $public_key"

# Generate an AES secret of your choice (here it is just based on the $AES_SECRET_KEY variable).

aes_secret_raw=$(openssl enc -nosalt -aes-256-cbc -pbkdf2 -k $AES_SECRET_KEY -P)
aes_key_hex=$(echo "$aes_secret_raw" | grep "key\s*=" | sed -e "s/.*=//")
aes_iv_hex=$(echo "$aes_secret_raw" | grep "iv\s*=" | sed -e "s/.*=//")

echo "AES key: $aes_key_hex, AES IV: $aes_iv_hex"

# Encode this encryption key as JSON; this is the secret that we will pass.

aes_key_base64=$(echo $aes_key_hex | xxd -r -p | base64)
aes_iv_base64=$(echo $aes_iv_hex | xxd -r -p | base64)
unencrypted_aes_secret=$(jq -n --arg key "$aes_key_base64" --arg iv "$aes_iv_base64" '{secretKey: $key, iv: $iv}')

echo "AES secret: $unencrypted_aes_secret"

# Encrypt the secret using the supplied public key.

rsa_public_key=$(echo -e "-----BEGIN PUBLIC KEY-----\n$public_key\n-----END PUBLIC KEY-----" | sed -r 's/(.{67})/\1\n/g')
mkdir -p $TEMP_FOLDER
echo "$rsa_public_key" > $TEMP_FOLDER/rsa_public_key.pem
encrypted_secret=$(echo $unencrypted_aes_secret | openssl rsautl -encrypt -inkey $TEMP_FOLDER/rsa_public_key.pem -pubin | base64 | tr -d '\n')
rm $TEMP_FOLDER/rsa_public_key.pem

echo "Encrypted AES secret: $encrypted_secret"
```

```
# Encrypt the PIN using your encryption key.

encrypted_pin=$(echo $PIN | tr -d '\n' | openssl enc -nosalt -aes-256-cbc -K $aes_key_hex -iv $aes_iv_hex | base64 | tr -d '\n')

echo "Encrypted PIN: $encrypted_pin"

# Post the headers as encrypted-secret and encrypted-pin to e.g. initiate a transaction.

curl -X POST -H "encrypted-secret: $encrypted_secret" -H "encrypted-pin: $encrypted_pin" -H "Authorization: Bearer ${access_token}" -H "Content-Type: application
/json" --data '{"standingOrder":{"amount":"12.0","partnerIban":"ES2015632626323268851568","partnerName":"Pancho Villa","nextExecutingTS":"1583452800000","
executionFrequency":"WEEKLY"}}' -k --key $CLIENT_KEY_FILE --cert $CLIENT_CERT_FILE https://${PISP_HOST}/api/transactions/so
```

payment_initiation_test.sh