User Lifecycle & Card Management API & SDK

User registration and lifecycle management. Server-2-server integration.

- Introduction
- Overview

Introduction

The main purpose of Verestro Lifecycle API & SDK web service is to add and manage user and payment cards in a safe way. All data are transfered to Verestro Data Core system which is placed in PCI compliant zone and is fully secure. By registering in the database, the partner can use various services provided by the Verestro i.e. Card Issuing, Card tokenization, Money Transfers and more.

The API allows to mass import of data files to Verestro systems, such as: user, user with a card, or cards. Using the HTTPS REST protocol, Partner can add a new resource (user or cards), update their status, e.g. lock, unlock or completely remove a resource from system.

Once connected to the partner's application, the SDK enables full user management and securing the connection to all Verestro products. Our SDK was develop for iOS and Android platforms.

Main responsibilities Lifecycle API:

- user - end user management in system,
- cards - card management in system,
- addresses - module related to user resource.

Main responsibilities Lifecycle SDK:

- user - end user registration, password and pin management,
- cards - card management,
- addresses - addresses management,
- device - device and session token management.

Overview

Lifecycle API The main purpose of a web service is to add a user and payment cards in a safe way to dedicated database PCI DSS compliant systems, which are provided by Verestro. By registering in the database, the partner can use various services provided by the Verestro company, e.g. Card Issuing, Card tokenization, Money Transfers and more. The API allows to mass import of data files to Verestro systems, such as: user, user with a card, or cards. Using the HTTPS REST protocol, you can add a new resource (user or cards), update their status, e.g. lock, unlock or completely remove a resource from system.

LIfecycle API is an internal service secured by x509 certificate, which increases the safety of transported data. The API communicates with the data storehouse called DataCore. DataCore is internal service and one of crucial components of Verestro's product line-up. Its main responsibility is to provide secure, PCI-DSS compliant storage for cardholder data. DataCore manages the status of the user and their aggregates. All other product in implementation connect to DataCore which returns information about the user and his aggregates.

Security

Data Storage

Lifecycle API is a part of PCI zone of Verestro platform so it meets all standards and restrictions of secure data storage. All fragile data is secured and encrypted.

We are using HashiCorp Vault as a Software HSM.

The master key isn't stored anywhere. It is reconstructed in unsealing process. It is used to encrypt encryption key. The data stored by Vault is stored encrypted. Therefore, to decrypt the data, Vault must decrypt the encryption key which requires the master key. Unsealing is the process of reconstructing this master key.

Instead of distributing this master key as a single key to an operator, Vault uses an algorithm known as Shamir's Secret Sharing to split the key into key shards. A certain threshold of shards is required to reconstruct the master key.

More information

When the Vault is initialized it generates an encryption key which is used to protect all the data. The encryption key is also stored with the data, but encrypted with another encryption key known

as the *master key*. Once Vault retrieves the encryption key, it is able to decrypt the data in the storage backend, and enters the *unsealed state.* Vault uses 256-bit AES to encrypt Encryption Key.

All sensitive data is encrypted in this way.

Encryption of fragile data

LC api allows to encrypt fragile card data. Detailed description is provided below.

JWE Standard

To encrypt fragile card data you should use JWE. If you are unfamiliar with this kind of standard please look at links below:

Wiki,

RFC,

Example.

The setup for Lifecycle JWE is presented below:

- in headers:
  - alg: RSA-OAEP-256 - keyEncryptionAlgo,
  - enc: A128GCM/A256GCM - contentEncryptionAlgo,
  - zip: DEF,
  - iat: this field should contain current timestamp,
  - kid: SHA1 of thumbprint of public key used to generate JWE (Static values is: Pdk08OtjTS6-I7H_E96XKme0BOY),
- in body: plaintext json data. Please see example below.

Public key used to generate JWE can be found here. //TODO: add file with public key for dev.

JWE Examples

During development you can use test methods that allows to generate and check your implementation of JWE:

PAYLOAD - json string used to generate JWE, for example:

```
{"pan":"5555444433331234","expiryDate":"2040-11-30"}
```

ISSUER - string provided by DC Team during integration
EXAMPLE_JWE - JWE to be validated and decoded

Generate JWE

```
curl --location --request GET 'https://datacore.upaidtest.pl/test/generate-jwe-token/PAYLOAD' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Basic dGVzdDE6dGVzdDEyMw==' \
--header 'ISSUER-CODE: ISSUER' \
--header 'COLLECTION: internal'
```

Read JWE

```
curl --location --request POST 'https://datacore.upaidtest.pl/test/read-jwe-token' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Basic dGVzdDE6dGVzdDEyMw==' \
--header 'ISSUER-CODE: ISSUER' \
--header 'COLLECTION: internal' \
--data-raw '{
    "token" : "EXAMPLE_JWE"
}'
```

Hash HMAC

Lifecycle also provides configuration called idType. It is simple tool that can be used if you can't store LC internal id's.
There are two main id types: useridType and cardIdType. All possible options for both are presented in LC API specification.
Here we are describing case when cardIdType is set to hash.
Hash value is as SHA-256 HMAC, please see links below for more details:

RFC,

Wiki,

Java example.

Test value of key used to calculate HMAC in HEX

59c6d62dde38d8a2c32105a53336b8ef

To validate your implementation please check plain and hashed values below:

```
"5555444433332222" "4f64c445c859f7e53209e0091a5faef7e8b3ebbad899fbf8c74df09a6bfe5646"
"698457689763489576394857" "4b2eab65ab16183fa6ac8a8b12ad690890db98c5ce20e6d56aa037b723bbe842"
"someTestValue398048096859607" "9596a78a7382e90159d8ec78a8d37baff57d05f676c0607dd7fb24b0396270ce"
```

Lifecycle Import

Info

Lifecycle file import is a mechanism that allows you to create a file with multiple instructions for api. Each instruction is like an request. Instead of sending 10.000 requests to API, which may take ages, you can create single file with all those instructions and let our system handle it asynchronously. Once import is completed an report is generated which will contain errors and information about processed rows. Files with instructions should be uploaded onto SFTP. To get access details, a new integration needs to be set up for bank.

Requirements

File storage with directories:

/**input** for input files
/**output** for output files (reports)
/**processed** for files that were processed

Input

Input files should be uploaded onto storage into 'input' directory. Everyday at 12:07 am a cron job will run and process those files.

File example

File format is json line (jsonl). In short this is a file that contains valid json in every line (separated by line break, not a comma etc.).

Each line is a single instruction for our import mechanism. Example file below:


```
{"method":"addUser","data":{"externalId":"48111111111","firstName":"First","lastName":"User","
phone":"48111111111","email":"you@post.com","birthDate":"1979-10-
06","wPIN":"1234","state":"VERIFIED"}}
{"method":"addUser","data":{"externalId":"48222222222","firstName":"Second","lastName":"User",
"phone":"48222222222","email":"me@post.com","birthDate":"1979-10-
06","wPIN":"1234","state":"VERIFIED"}}
{"method":"addUser","data":{"externalId":"48333333333","firstName":"Third","lastName":"User","
```

```
phone":"48333333333","email":"validemail@post.com","birthDate":"1979-10-
06","wPIN":"1234","state":"VERIFIED"}}
```


Line explaination

Each line is a JSON with two keys:

method - name of method you're calling (supported methods below),
data - request body.

For example result of importing first line of example file is similar to making a request to **/lifecycle/v1/users** with body:


```
{"externalId":"48111111111","firstName":"First","lastName":"User","phone":"48111111111","email":
```


Supported methods

At the moment the only one supported method is **addUser** (
https://datacore.upaidtest.pl/documentation-lifecycle/#api-Lifecycle-user_add).

Processing

Processing is handled asynchronously. It means that once file is read, every line will be changed into a "job" and then processed by our "workers". After reading whole file it is moved to **processed** directory. PS. This does not mean that the import is completed.

If workers encounter any troubles during handling their job an error message will be inserted into report file. Structure of error line is "**X, ERROR_MESSAGE**" where ***X*** is the line number from import file and **ERROR_MESSAGE** is just the error message. Examples presented in Output section.

Output

Once import is completed an line with rows processed information is inserted into the report file. You can find report files inside **output** directory. Report file name is $input_file_name_without_extension-report.csv. Example input/output file with names below.

Example input file

input_file.jsonl

```
{invalid json :(()
{"method":"addUser","data":{"externalId":"48222222222","firstName":"Second","lastName":"User",
"phone":"48222222222","email":"me@post.com","birthDate":"1979-10-
06","wPIN":"1234","state":"VERIFIED"}}
{"method":"addUser","data":{"externalId":"48333333333","firstName":"Third","lastName":"User","
phone":"48333333333","email":"validemail@post.com","birthDate":"1337","wPIN":"1234","state":"V
ERIFIED"}}
{"method":"thisWillNotWork","data":{"externalId":"4833"}}
```

Example output file

input_file-report.csv

```
1,INVALID_JSON
3,{"errors":{"phone":["VALUE_HAS_TO_BE_UNIQUE"],"birthDate":["DATE_IS_INVALID","DATE_FORMAT_IS
_INVALID"]}}
4,INVALID_METHOD

Total rows processed: 4
```

Typical error messages:

INVALID_JSON - there was an error while trying to decode line (first line of example input file).
INVALID_METHOD - invalid/unsupported method (fourth line of example input file).
UNKNOWN_ERROR - unhandled error (contact DC team for more info).

As you may have noticed there is also an error message encoded in json format. This is the same response that you would receive in a normal api call.

PS. Successfully processed line doesn't produce any output in report file (that's why there is no status for 2nd line).

Lifecycle API usage

URL construction

/users/{id}/cards/{cardId}?userIdType={userIdType}&cardId=internalId

id - value defines user identifier.
userIdType - type of value that identifies user. Supported options:

- phone,

- email,
- externalId - provided by partner,
- internalId - provided by Verestro (default option),
- hash – card hash provided by bank.

Headers

Field	Description
Authorization	Basic Access Authentication token.
Accept	application/json .
Content-Type	application/json .
Accept-Language	Options: en, pl, ...
Issuer-Code	Wallet name. This value is provided by datacore team.
Collection	Collection name. Options: lifecycle .
Application-Id	The unique name of the application, e.g. "admin-panel" .
X-B3-SpanId	This is the id of the specific operation. Link .
X-B3-TraceId	This is a global id that is used to identify a sequence of requests / responses in the whole system. This means that it should be passed between all websites in requests / responsives. In case it is null then of course you have to generate a new one, because it means that this is the first request from an external client.