trustos Documentation

Release master

hyperledger

Jun 07, 2023

Contents

Warning: Hey, these docs has been deprecated.

Please visit trustos.telefonica.com/docs to read our new documentation and keep up to date with all the new enhancements.

A complete Blockchain solution which abstracts all the complexity of blockchain technology.

CHAPTER 1

Introduction

TrustOS is both an SDK and a powerful, easy-to-use set of decentralized network modules deployed in several blockchain networks. It makes easier to anyone to get the benefits of blockchain technology without facing the inherent complexity of it. If you want to enter the Blockchain space, it does not matter the underlying technology, the consensus mechanisms, the network, the consortia or the infrastructure anymore. Just get the better of blockchain invoking the **TrustOS.**

1.1 Our vision

As a telco, we really believe **Blockchain adds a trust layer to operations.** So, we must make accessible this trust without adding additional complexity, and the best way to do that is to embed it in the network. So, we aim to design a **"Blockchain for ALL" proposition** to change the way companies and entrepreneurs build their digital services in a decentralized world. Nobody must validate, certificate or audit what the services are doing, the info they handle, its authorship or how they are working, the services themselves will do it through a Blockchain layer embedded in the networks.

1.2 Problem statement

Companies are interested in Blockchain and are constantly asking themselves what technology is for and if there is any way to take advantage of it in their business. They feel both interest and frustration at the complexity of choosing a technology, developing their Smart Contracts in specific languages, deploying their own network, and so on. The result in most cases is that for now the possible advantages are totally out of reach.

However, the use cases just benefit from technology in the same way and we can offer and bundle these benefits to the community without having to really understand the technology or deploying their own network.

1.3 Target audience

TrustOS is initially designed to enable developers who wish to take advantage of Blockchain to easily integrate it into their products and services by invoking a set of JSON APIs, without investing in understanding the technology or startings ophisticated ad-hoc developments. They need us to offer them a simple, flexible, affordable and fast way to add the advantages of Blockchain to their products and services without having to set up or operate their network. They need to record information immutably, create and manage tokens or reach consensus on information gathered from different sources. We build the ecosystem of trust they want through TrustOS so they can use Blockchain and add their advantages to their value proposition without dealing directly with the technology. **They want trust as a service.**

1.4 Solution

TrustOS is a network-deployed middleware that makes the customer's business systems independent from the Blockchain networks:

- Enables use of Blockchain without knowledge of the technology
- Simplifies integration
- Facilitates adaptation to new Blockchain technologies and networks
- Allows third party verification of information

How can TrustOS help you?

INMUTABILITY: The underlying cryptography allows the existence, authorship and integrity of the information that is generated and transmitted to be attributed with absolute certainty

TRACEABILITY: The succession of transactions makes it possible to reconstruct without a doubt the complete history of events without the ability to repudiate

TRANSPARENCY The technology itself allows the information to be notarised and verified autonomously without the intervention of trusted third parties

CHAPTER 2

Architecture

TrustOS software is deployed on any Hyperledger Fabric based blockchain network. Support for Quorum/Ethereum and CORDA based networks will be added soon.

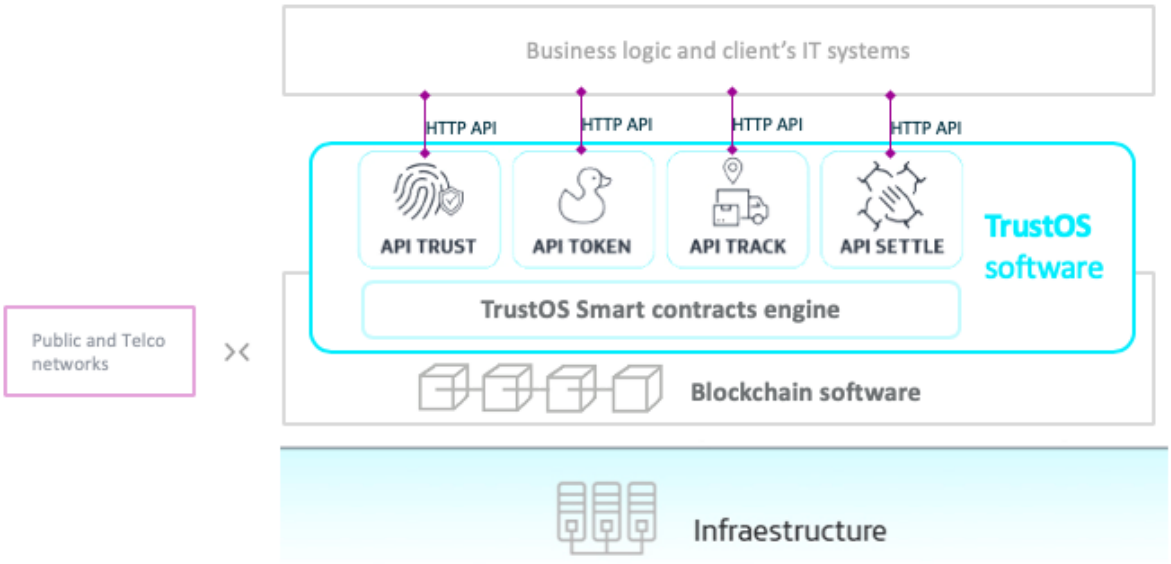
Once deployed, an HTTP API is published so that it can be used from any programming language. The API hides much of the complexity of working with a Blockchain, but its direct use still requires low-level knowledge. For that reason, TrustOS also includes some libraries to be consumed by applications and other Smart Contracts. Therefore, TrustOS commands can be invoked from outside Blockchain through its APIs or from other Smart Contracts inside the Blockchain itself.

Since the installation and deployment of TrustOS requires considerable efforts, TrustOS is perfectly designed to allow the majority of business logics and client systems take advantage of its benefits without having knowledge of Hyperledger Fabric neither deploying anything, just using simple APIs. Moreover, for those who are deepening into the field, doors will be open to deploy TrustOS software over its own network.

TrustOS also includes some administration features and interfaces that allow the developer community to add new modules to TrustOS.

2.1 How does it work

The following figure describes a layer-based architecture:



APIs do not speak the **Blockchain language** (send, validate or verify a transaction, query a block, provision or obtain gas at an address, compile, deploy or execute contracts, manage and safeguard cryptographic material, etc.) but the language a customer understands (create an asset and update its status or position, add account activity, verify an identity, create or transfer a token, etc.).

Initially, **TrustOS** is available for deployment on any Hyperledger Fabric-based network and interacts with Ethereum (mainnet and testnet networks), Polygon (mainnet and testnet networks) and Alastria's Network B (built on Hyperledger Besu). In the coming months support will be added for implementation in other technologies. We put a lot of effort into evolving the product

CHAPTER 3

Getting Started

Why worrying about the low-level of blockchain technology, and having to worry about building your network when you can leverage the benefits of the technology without having to know the details of its operation. **TrustOS** abstracts all the complexity of blockchain technology implementing the basic operations you need to leverage the power of blockchain technology.

We have created several APIs for tracking the whole lifecycle of assets, for creating and managing transferable tokens, for offering agreement between diferent parts and ensuring confidence and integrity in the all the generated information.

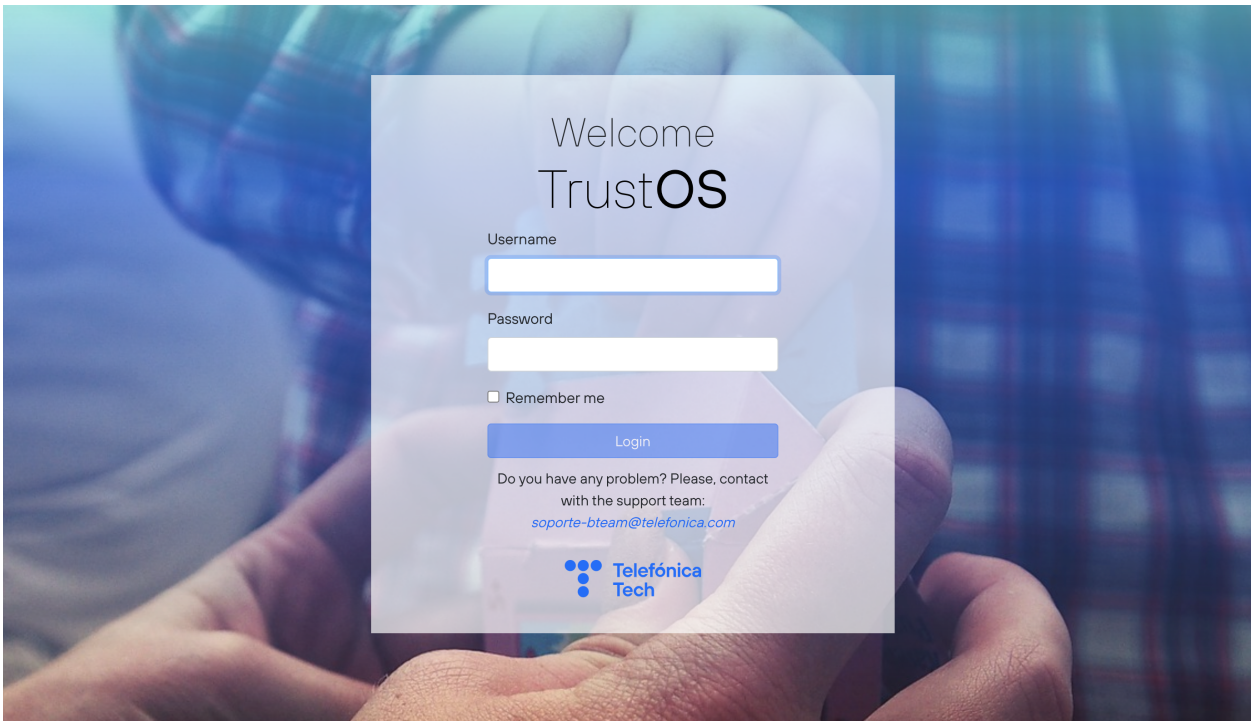
Before starting, it may be interesting for you to know about the architecture and the different modules that compound TrustOS.

3.1 Login

In order to use the APIs, you need to have an active user and login to use the system. Every API has a login method, which asks for a user and password.

3.1.1 Login UI

To facilitate the first interactions with TrustOS there is a login website so that the user can easily login and authenticate the next use of the website through cookies that contains the JWT and that expires after the JWT becomes invalid.



3.1.2 Login request

Once your solution has matured, it would be nice to integrate the login process through API requests.

A call to the login method will return a JWT token, of this form:

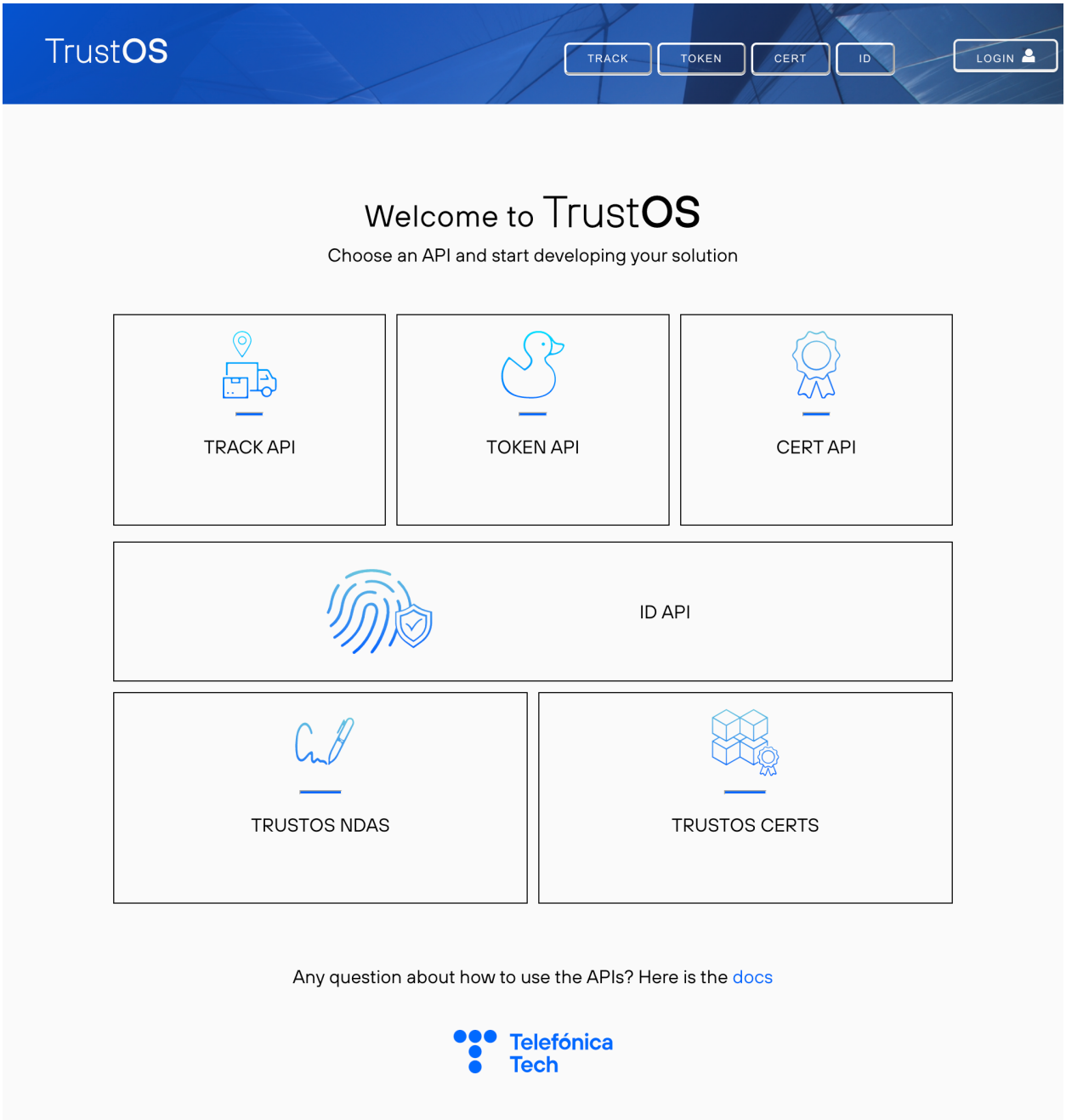
```
{
  "message": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJ1c2VyIjoidGVzdCIsImV4cCI6MTU2MDAwMDE5MH0.M4PBSslERUImcOpWgg--N-
↪2ZNW306BzWXTZVJgtdXWE"
}
```

3.2 Choose an API and start developing

Once the login is sucessfully done it is time to start developing your own solution based on one of the TrustOS modules.

3.2.1 Swagger UI

We have created a simple website that aggregates all the accesible modules so that the user can easily reach and test all the functionalities.



If you have login using the Login UI, it is simple to see and copy the JWT Token just clicking the header green button `JWT`.

Choose an API to see what there is inside and then you have to click the button **Authorize, on the right** of the screen.

From there, in the "value" field of ApiKeyAuth, you should write "Bearer" followed by the JWT Token already copied:

```
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJ1c2VyIjoidGVzdCIsImV4cCI6MTU2MDAwMDE5MH0.M4PBSslERUImcOpWgg--N-
↪2ZNW306BzWXTZVJgtdXWE
```

You should now see that all the locks in swagger are closed, meaning that you are now authenticated!

3.2.2 API request

Every call to the API, has to be authenticated, so the caller must provide this message as proof of his identity.

Add the following header to your HTTP request in order to authenticate your call:

```
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJ1c2VyIjoidGVzdCIsImV4cCI6MTU2MDAwMDE5MH0.M4PBSslERUImcOpWgg--N-
↪2ZNW306BzWXTZVJgtdXWE
```

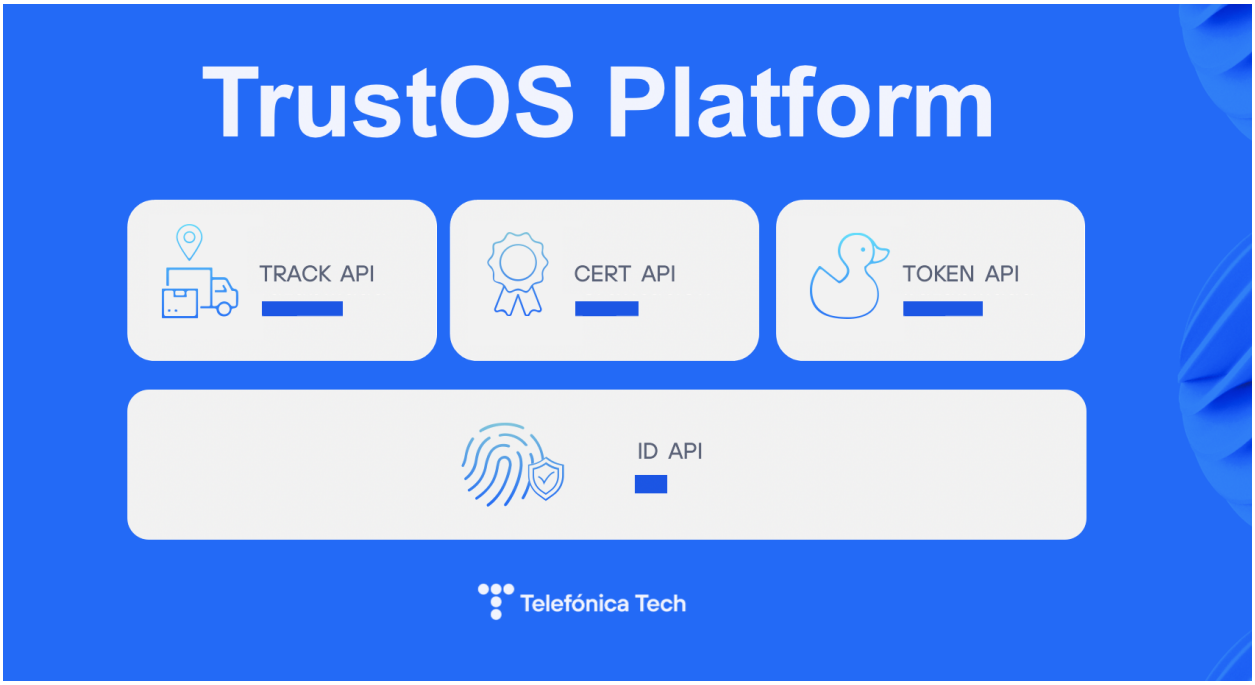
If you are calling the APIs from Postman, you can set manually the headers (you can see an example in tutorials section):

```
{
"Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJ1c2VyIjoidGVzdCIsImV4cCI6MTU2MDAwMDE5MH0.M4PBSslERUImcOpWgg--N-
↪2ZNW306BzWXTZVJgtdXWE"
}
```

Now it is time to read more about the TrustOS modules.

CHAPTER 4

Modules



4.1 Track

Creation and tracking the life cycle of the assets in the Blockchain. An asset is a digital representation of a real asset in the physical world. Using this module you can record immutable information that uniquely characterizes the asset and update its attributes inherently linked to the instant it was created/updated, so you do not need a trusted third party to certify the existence of that information at any time since you recorded it.

4.2 Token

Creation and management of transferable tokens to build new markets, gamification strategies and easily create transferable value on the network. Applications can issue tokens, transfer them, lock them, etc., keeping at any time an absolute and certain control of the property or rights of use over it. ERC20 and ERC721 standards are implemented to make easier any kind of integration or migration with/from external environments.

4.3 Cert

Creation, management and sign of digital certificates on blockchain. A certificate is a tamper-proof and verifiable collection of data that represents a process, a file/document, an accomplishment or any activity that can benefit from the inmmutability and authenticity.

4.4 ID

Identity management and decentralised services. Through this module it is possible to create and manage identities. TrustOS acts as the custodian of the identity keys, but it is also possible to operate with external identities and even import them. In addition, it has integration with OpenID to operate with identities generated in the different identity providers.

CHAPTER 5

Track

Track API is used to create, manage, and follow life cycle of **digital assets** on the blockchain. An asset is a digital representation of a real asset in the physical world. Through this API tracking the whole state changes of every asset can be implemented in an easy way, taking advantage of the inherit benefits from blockchain.

5.1 API Specification

An abstraction API with all the asset functionalities.

5.1.1 Asset

An asset is a digital representation of a real asset in the physical world. An asset records every single state or data change (f.e. the update of metadata, the transfer of ownership, etc.) This allow us to track the whole transactions since its creation in an inmutable and transparent way.

Every asset has the following structure:

- `assetid` : `<string>` Unique identifier of the asset
- `data` : `<json>` JSON of **inmutable** data. Can have as many fields as required by the use case
- `metadata`: `<json>` JSON of **mutable** data. Can have as many fields as required by the use case
- `timestamp` : `<string>` UNIX date of asset creation
- `userOwner`: `<string>` Owner of the asset
- `hash` : `<string>` Hash of the asset

Sample structure (Click to expand)

```
{
    "assetid": "exampleAsset",
    "data": {
```

(continues on next page)

(continued from previous page)

```
      "id":"A2839RP",
      "version":"1"
    },
    "metadata": {
      "color": "red"
      "position": { "x": "53", "y": "22"}
    },
    "timestamp": 1558009289,
    "userOwner": "test:telefonicaMSP",
    "hash": "oCZygxQBp5HBVm+SSUCCrgJfV3+CeghOzV9m+UxDsY8=",
}
```

5.1.2 Authorised assets

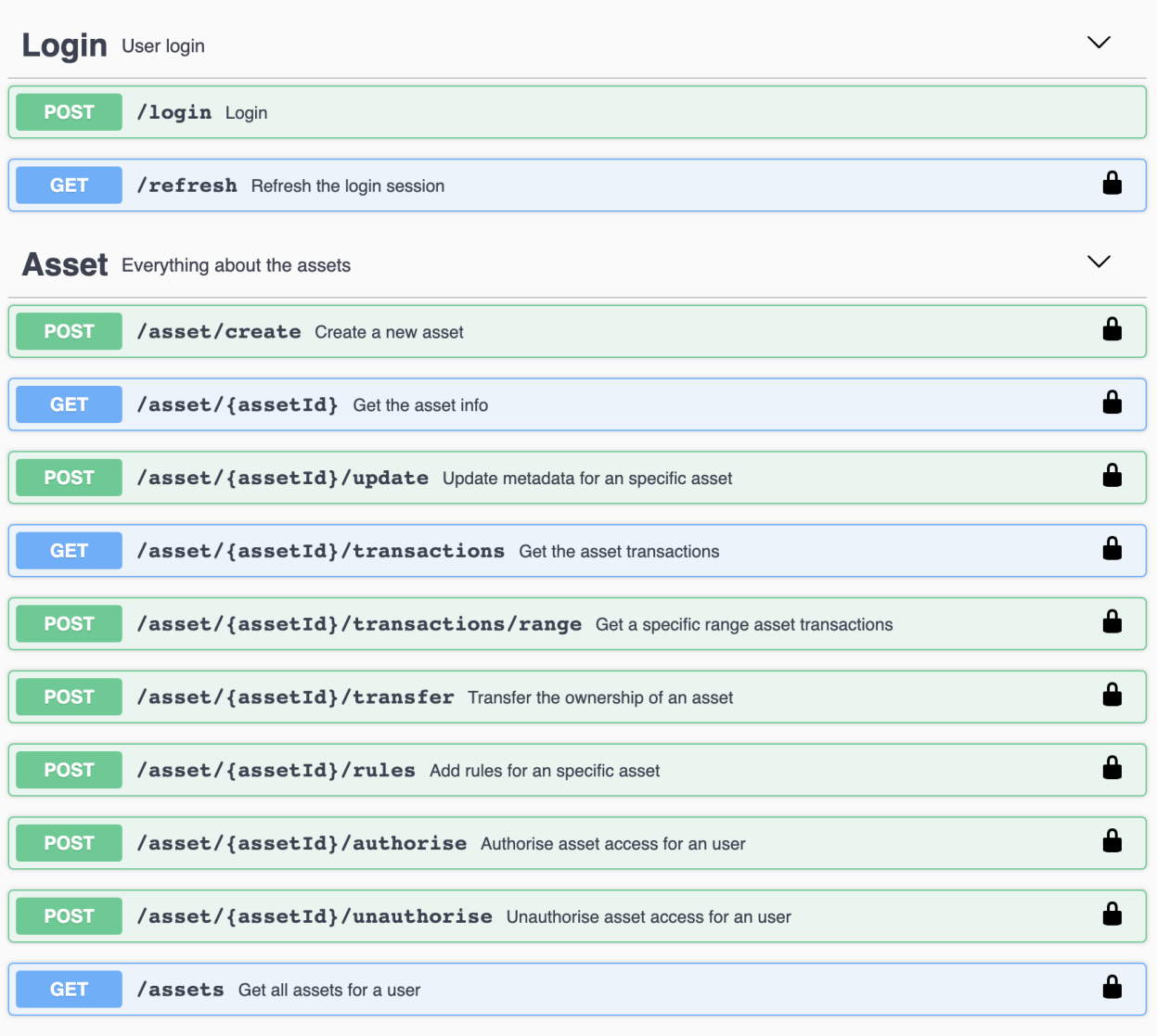
There are two types of assets: owned assets and authorised assets. These last ones, the `authorised assets` are assets created by other users that have granted you access allowing you to consult and update it.

In order to interact with both assets in some functions it is necessary to set a flag formely known as `isAuthorised`. In order to interact with authorised, only it is necessary to set the `isAuthorised` flag to `true` as a query parameter in the URL (`...?isAuthorised=true`) as it is shown in the examples below:

- GET - `/asset/{assetId}?isAuthorised=boolean`
- GET - `/asset/{assetId}/transactions?isAuthorised=boolean`
- POST - `/asset/{assetId}/transactions/range?isAuthorised=boolean`
- GET - `/assets?isAuthorised=boolean`
- POST - `/asset/{assetId}/update?isAuthorised=boolean`
- POST - `/asset/{assetId}/batch/array?isAuthorised=boolean`
- POST - `/asset/{assetId}/batch/range?isAuthorised=boolean`
- POST - `/asset/{assetId}/batch/updateArray?isAuthorised=boolean`
- POST - `/asset/{assetId}/batch/updateRange?isAuthorised=boolean`
- POST - `/asset/{assetId}/evidence?isAuthorised=boolean`
- POST - `/asset/{assetId}/getEvidence?isAuthorised=boolean`
- GET - `/asset/{assetId}/getEvidences?isAuthorised=boolean`

To manage your own assets you must set the parameter to false (`...?isAuthorised=false`).

5.2 API Methods



Asset methods (Click to expand)

5.2.1 POST - `/asset/create`

Ceate a digital asset.

Input

- `assetid` : `<string>` Unique identifier of the asset.
- `data` : `<json>` JSON of **inmutable** data. It can have as many field as required.
- `metadata`: `<json>` JSON of **mutable** data. It can have as many field as required.
- `metadata`: `<bool>` Boolean value to set if it is a batch asset or not.

Sample structure (Click to expand)

```
{
    "assetid": "",
    "data": {
      "id":"A2839RP",
      "version":"1"
    },
    "metadata" : {
      "color": "red",
      "position": { "x": 23.34, "y": -24.22}
    }
}
```

Output

- `asset : <json>`

Sample structure (Click to expand)

```
{
    "assetid": "exampleAsset",
    "data": {
      "id":"A2839RP",
      "version":"1"
    },
    "metadata": {
      "color": "red"
      "position": { "x": "53", "y": "22"}
    },
    "timestamp": 1558009289,
    "userOwner": "test:telefonicaMSP",
    "hash": "oCZygxQBp5HBVm+SSUCCrgJfV3+CeghOzV9m+UxDsY8=",
}
```

5.2.2 GET - `/asset/{assetId}?isAuthorised=boolean`

Get the asset identified by assetId.

Input

- `assetid : <string>` Unique identifier of the asset.
- `isAuthorised: <boolean>` Flag to get own or authorised assets.

(*) Please navigate to the following *section* for isAuthorised query param details.

Output

- `asset : <json>`

Sample structure (Click to expand)

```
{
    "assetid": "exampleAsset",
    "data": {
      "id":"A2839RP",
      "version":"1"
```

(continues on next page)

(continued from previous page)

```
    },
    "metadata": {
      "color": "red"
      "position": { "x": "53", "y": "22"}
    },
    "timestamp": 1558009289,
    "userOwner": "test:telefonicaMSP",
    "hash": "oCZygxQBp5HBVm+SSUCCrgJfV3+CeghOzV9m+UxDsY8=",
}
```

5.2.3 POST /asset/{assetId}/update?isAuthorised=boolean

Updates the **mutable** ("metadata") of an asset.

Input

- `assetid` : <string> Unique identifier of the asset.
- `isAuthorised`: <boolean> Flag to update own (false) or authorised (true) assets.
- `metadata`: <json> JSON of **mutable** data. It can have as many field as required.

(*) Please navigate to the following *section* for isAuthorised query param details.

Sample structure (Click to expand)

```
{
  "metadata": {
    "color": "blue",
    "position": { "x": 98.35, "y": -12.32}
  }
}
```

Output

- `asset` : <json>

Sample structure (Click to expand)

```
{
    "output": {
        "assetId": "test1",
        "data": {
            "color": "yellow",
            "size": "big"
        },
        "metadata": {
            "color": "blue",
            "position": {
                "x": 98.35,
                "y": -12.32
            }
        },
        "timestamp": 1647953653,
        "userOwner":
↪"did:vtn:trustid:ed770703f65656e5b689a047d1cee645b7ad119610a1d31a63f5be0e45c6e0d9",
```

(continues on next page)

(continued from previous page)

```
        "hash": "F5SzRyp4ELhbtcDEsPm8a+2XjyI5w4uoLkAb5yO9C0E="
    }
}
```

5.2.4 GET - /asset/{assetId}/transactions?isAuthorised=boolean

Get all transactions for the whole lifecycle of the asset.

Input

- `assetid` : <string> Unique identifier of the asset.
- `isAuthorised`: <boolean> Flag to get own or authorised assets.

(*) Please navigate to the following *section* for isAuthorised query param details.

Output

- `asset` : <json> A list of all transactions.

Sample structure (Click to expand)

```
{
    "output": {
        "assetId": "test1",
        "data": {
            "color": "yellow",
            "size": "big"
        },
        "transactions": [
            {
                "metadata": {
                    "color": "red",
                    "size": "medium"
                },
                "timestamp": 1647953221,
                "userOwner":
↪"did:vtn:trustid:ed770703f65656e5b689a047d1cee645b7ad119610a1d31a63f5be0e45c6e0d9",
                "hash": "a20Reot68bbYEap+RfN4EmtEbrKoE0UO9rgn205jln0="
            },
            {
                "metadata": {
                    "color": "blue",
                    "size": "big"
                },
                "timestamp": 1647951890,
                "userOwner":
↪"did:vtn:trustid:ed770703f65656e5b689a047d1cee645b7ad119610a1d31a63f5be0e45c6e0d9",
                "hash": "RB/vC1wSwS2hhbttvmtMehWROqmcwlPL9+tkdODLVGI="
            }
        ]
    }
}
```

5.2.5 POST - /asset/{assetId}/transactions/range?isAuthorised=boolean

Get all transactions within a range for the whole lifecycle of the asset.

Input

- `assetid` : <string> Unique identifier of the asset.
- `isAuthorised`: <boolean> Flag to get own or authorised assets.
- `init` : <string> Transactions low limit.
- `end` : <string> Transactions upper limit.

(*) Please navigate to the following *section* for isAuthorised query param details.

Sample structure (Click to expand)

```
{
  "init": "0",
  "end": "1575975331"
}
```

Output

- `asset` : <json> A list of all transactions.

Sample structure (Click to expand)

```
{
    "output": {
        "assetId": "test1",
        "data": {
            "color": "yellow",
            "size": "big"
        },
        "transactions": [
            {
                "metadata": {
                    "color": "red",
                    "size": "medium"
                },
                "timestamp": 1647953221,
                "userOwner":
↪"did:vtn:trustid:ed770703f65656e5b689a047d1cee645b7ad119610a1d31a63f5be0e45c6e0d9",
                "hash": "a20Reot68bbYEap+RfN4EmtEbrKoE0UO9rgn205jln0="
            },
            {
                "metadata": {
                    "color": "blue",
                    "size": "big"
                },
                "timestamp": 1647951890,
                "userOwner":
↪"did:vtn:trustid:ed770703f65656e5b689a047d1cee645b7ad119610a1d31a63f5be0e45c6e0d9",
                "hash": "RB/vC1wSwS2hhbttvmtMehWROqmcwlPL9+tkdODLVGI="
            }
        ]
    }
}
```

5.2.6 POST - /asset/{assetId}/transfer

Transfer the ownership of the asset. The user has to be the owner of the asset.

Input

- `assetid : <string>` Unique identifier of the asset.
- `destinationId : <string>` The destination owner.

Sample structure (Click to expand)

```
{
  "destinationId": "bteam",
}
```

Output

- `asset : <json>`

Sample structure (Click to expand)

```
{
    "assetid": "exampleAsset",
    "data": {
      "id":"A2839RP",
      "version":"1"
    },
    "metadata": {
      "color": "red"
      "position": { "x": "53", "y": "22"}
    },
    "timestamp": 1558009289,
    "userOwner": "bteam",
    "hash": "oCZygxQBp5HBVm+SSUCCrgJfV3+CeghOzV9m+UxDsY8=",
}
```

5.2.7 POST - /asset/{assetId}/rules

Add rules to monitor asset parameters.

Input

- `assetId : <string>` Unique identifier of the asset.
- `rules: <json>` JSON of rules. It can have at least two fields: value & range, to specify a constant value or range of values that has to accomplish a parameter. Every rule (value, range) can contain as many conditions for different parameters as necessary. However it's noted that a use of quite many conditions affects the performance of the asset udpates.

Sample structure (Click to expand)

```
{
  "rules": {
    "value": [
      {
        "param": "a",
        "value": "b"
      },
      {
        "param": "aa",
        "value": "bb"
      }
    ],
    "range": [
      {
        "param": "b",
        "min": 0,
        "max": 100
      }
    ]
  }
}
```

Output

- `rules : <json>`

Sample structure (Click to expand)

```
{
{
  "rules": {
    "value": [
      {
        "param": "a",
        "value": "b"
      },
      {
        "param": "aa",
        "value": "bb"
      }
    ],
    "range": [
      {
        "param": "b",
        "min": 0,
        "max": 100
      }
    ]
  }
}
```

5.2.8 POST - `/asset/{assetId}/authorise`

Authorise user access for an asset. Only the asset owner can do this.

Input

- `assetId` : `<string>` Unique identifier of the asset.
- `userId` : `<string>` The authorised user.

Sample structure (Click to expand)

```
{
  "userId": "did:bteam"
}
```

Output

- `asset` : `<json>`

Sample structure (Click to expand)

```
{
  "output": {
    "message": "Successfully authorised user did:bteam for asset XXXXX",
  }
}
```

5.2.9 POST - `/asset/{assetId}/unauthorise`

Unauthorise user access for an asset. Only the asset owner can do this.

Input

- `assetId` : `<string>` Unique identifier of the asset.
- `userId` : `<string>` The unauthorised user.

Sample structure (Click to expand)

```
{
  "userId": "did:bteam"
}
```

Output

- `asset` : `<json>`

Sample structure (Click to expand)

```
{
  "output": {
    "message": "Successfully unauthorised user did:bteam for asset XXXXX",
  }
}
```

5.2.10 GET - `/assets?isAuthorised=boolean`

Get all assets for user

Input

- `isAuthorised` : <bool> Flag to get own or authorised assets.

(*) Please navigate to the following *section* for isAuthorised query param details.

Output

- `assetList` : <json>

Sample structure (Click to expand)

```
{
  "output": [
    "exampleAsset1",
    "exampleAsset2",
    "exampleAsset3"
  ]
}
```

5.2.11 POST - `/assets/create`

Creates assets from file

Input

- `fileInput` : <string> File from which the asset will be generated.

Output

- `output` : <json>

Sample structure (Click to expand)

```
{
  "output": [
    {
      "message": "The asset with assetId x has been created successfully"
    }
  ]
}
```

5.2.12 POST - `/assets/update`

Updates assets from file

Input

- `fileInput` : <string> File from which the asset will be updated.

Output

- `output` : <string>

Sample structure (Click to expand)

```
{
  "output": [
    {
      "message": "The asset with assetId x has been updated successfully"
    }
  ]
}
```

5.2.13 POST - /asset/{assetId}/batch/array?isAuthorised=boolean

Creates batch info for an specific asset. This is a list with the IDs of the assets that will belong to the batch.

Input

- `assetid` : <string> Unique identifier of the asset.
- `isAuthorised`: <boolean> Flag to get own or authorised assets.
- `batchInfo`: <json> Array that will represent the list of the IDs of the assets that will belong to the batch.

Output

- `output` : <string>

Sample structure (Click to expand)

```
{
  "output": "Batch stored successfully"
}
```

5.2.14 POST - /asset/{assetId}/batch/range?isAuthorised=boolean

Creates batch info for an specific asset. This is a range of the IDs of the assets that will belong to the batch.

Input

- `assetid` : <string> Unique identifier of the asset.
- `isAuthorised`: <boolean> Flag to get own or authorised assets.
- `init` : <string> Range of the IDs of the assets that will belong to the batch.
- `end` : <string> Range of the IDs of the assets that will belong to the batch.

Output

- `output` : <string>

Sample structure (Click to expand)

```
{
  "output": "Batch stored successfully"
}
```

5.2.15 POST - /asset/{assetId}/batch/updateArray?isAuthorised=boolean

Update batch info for an specific asset. This is a list with the IDs of the assets that will be added to the batch.

Input

- `assetid` : `<string>` Unique identifier of the asset.
- `isAuthorised`: `<boolean>` Flag to get own or authorised assets.
- `batchArray`: `<json>` Array that will represent the list of the IDs of the assets that will be added to the batch.

Output

- `output` : `<string>`

Sample structure (Click to expand)

```
{
  "output": "Batch stored successfully"
}
```

5.2.16 POST - /asset/{assetId}/batch/updateRange?isAuthorised=boolean

Update batch info for an specific asset. This is a range of the IDs of the assets that will belong to the batch.

Input

- `assetid` : `<string>` Unique identifier of the asset.
- `isAuthorised`: `<boolean>` Flag to get own or authorised assets.
- `init` : `<string>` Range of the IDs of the assets that will belong to the batch.
- `end` : `<string>` Range of the IDs of the assets that will belong to the batch.

Output

- `output` : `<string>`

Sample structure (Click to expand)

```
{
  "output": "Batch stored successfully"
}
```

5.2.17 POST - /asset/{assetId}/admin/create

Creates an admin user that is going to be able to authorise other users. Only the asset owner can do this. There can be more than one admin user and the admin can be admin from different assets of different owners.

Input

- `assetId` : `<string>` Unique identifier of the asset.
- `userId` : `<string>` The user that is going to manage the asset access.

Sample structure (Click to expand)

```
{
  "userId": "did:bteam"
}
```

Output

- `output : <string>`

Sample structure (Click to expand)

```
{
  "output": "Authorisation has been sucessfully done for asset: example1 and user:␣
↪did:vtn:trustid:bob"
}
```

5.2.18 POST - `/asset/{assetId}/admin/delete`

Delete an admin user that is not going to be able to authorise other users. Only the asset owner can do this.

Input

- `assetId : <string>` Unique identifier of the asset.
- `userId : <string>` The user that is going to manage the asset access.

Sample structure (Click to expand)

```
{
  "userId": "did:bteam"
}
```

Output

- `output : <string>`

Sample structure (Click to expand)

```
{
  "output": "Authorisation has been sucessfully done for asset: example1 and user:␣
↪did:vtn:trustid:bob"
}
```

5.2.19 POST - `/asset/{assetId}/admin/authorise`

Authorise user access for an asset. Only the asset admin can do this.

Input

- `assetId : <string>` Unique identifier of the asset.
- `userId : <string>` The authorised user.
- `ownerId : <string>` The asset's owner.

Sample structure (Click to expand)

```
{
  "userId": "did:bteam",
  "ownerId": "did:bteam"
}
```

Output

- `output : <string>`

Sample structure (Click to expand)

```
{
  "output": "Authorisation has been sucessfully done for asset: example1 and user:␣
↪did:vtn:trustid:bob"
}
```

5.2.20 POST - `/asset/{assetId}/admin/unauthorise`

Unauthorise user access for an asset. Only the asset owner can do this.

Input

- `assetId : <string>` Unique identifier of the asset.
- `userId : <string>` The unauthorised user.
- `ownerId : <string>` The asset's owner.

Sample structure (Click to expand)

```
{
  "userId": "did:bteam",
  "ownerId": "did:bteam"

}
```

Output

- `output : <json>`

Sample structure (Click to expand)

```
{
  "output": "Authorisation has been sucessfully done for asset: example1 and user:␣
↪did:vtn:trustid:bob"
}
```

5.2.21 POST - `/asset/{assetId}/evidence?isAuthorised=boolean?networkId=int`

Create an asset evidence in a public network.

Input

- `assetId : <string>` Asset identifier.

- `isAuthorised`: <boolean> Flag to get own or authorised assets.
- `networkId` : <number> Network identifier (Ethereum = 1, Goerli = 5, Polygon = 137, Mumbai = 80001)
- `init` : <string> Transactions low limit to generate a public evidence.
- `end` : <string> Transactions upper limit to generate a public evidence.

Sample structure (Click to expand)

```
{
  "init": "0",
  "end": "1575975331"
}
```

Output

- `evidence` : <json>

Sample structure (Click to expand)

```
{
  "output": {
    "networkId": 1,
    "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c=",
    "timestamp": 1647522920,
    "init": 0,
    "end": 1592568489,
    "smartContract": "0x1B646bc6C3465Fa8171F7171097A7d8e37b43D6B",
    "transaction": "0x3d3d63714b62db4f28ef6d46911e864520db0645985dce250a80dc8bf6d35f6f
↪",
    "includedTransactions": [
      {}
    ]
  }
}
```

5.2.22 POST - `/asset/{assetId}/getEvidence?isAuthorised=boolean`

Get a specific asset evidence from the creation timestamp

Input

- `assetId` : <string> Asset identifier.
- `isAuthorised`: <boolean> Flag to get own or authorised assets.
- `timestamp` : <string> Timestamp when the public evidence was generated.

Sample structure (Click to expand)

```
{
  "timestamp": "1575975331"
}
```

Output

- `evidence` : <json>

Sample structure (Click to expand)

```
{
  "output": {
    "networkId": 1,
    "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c=",
    "timestamp": 1647522920,
    "init": 0,
    "end": 1592568489,
    "smartContract": "0x1B646bc6C3465Fa8171F7171097A7d8e37b43D6B",
    "transaction": "0x3d3d63714b62db4f28ef6d46911e864520db0645985dce250a80dc8bf6d35f6f
↪",
    "includedTransactions": [
      {}
    ]
  }
}
```


5.2.23 GET - `/asset/{assetId}/getEvidences?isAuthorised=boolean`

Get all asset evidences from public networks.

Input

- `assetId` : <string> Asset identifier.
- `isAuthorised`: <boolean> Flag to get own or authorised assets.

Output

- `evidences` : <json>

Sample structure (Click to expand)


```
{
  "output": {
    [
      "networkId": 1,
      "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c=",
      "timestamp": 1647522920,
      "init": 0,
      "end": 1592568489,
      "smartContract": "0x1B646bc6C3465Fa8171F7171097A7d8e37b43D6B",
      "transaction":
↪"0x3d3d63714b62db4f28ef6d46911e864520db0645985dce250a80dc8bf6d35f6f",
      "includedTransactions": [
        {}
      ]
    ],
    [
      "networkId": 2,
      "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c=",
      "timestamp": 1647522922,
      "init": 0,
      "end": 1592568489,
      "smartContract": "0x1B646bc6C3465Fa8171F7171097A7d8e37b43D6B",
      "transaction":
↪"0x3d3d63714b62db4f28ef6d46911e864520db0645985dce250a80dc8bf6d35f6f",
```


(continues on next page)

(continued from previous page)

```
      "includedTransactions": [
        {}
      ]
    ]
  }
}
```

5.3 How we run the application

As you could see in the Architecture module, all the applications are running on cloud. Through Kubernetes orchestration system the application deployment, scaling and management is an easy and automated task.

5.4 Testing the Application

In postman folder there are the collection and environment to interact and test with the API methods. It is only needed to import them into postman application and know to use the coren-trackapi module.

Also you can download the files in the links below:

- Postman collection

- Postman environment

5.5 Errors management

Track API errors are managed through the following JSON:

```
{
  "error": {
    "code": "HTTP status code",
    "function": "function in which the error was generated",
    "message": "error description"
  }
}
```

CHAPTER 6

Token

Token API allows developers to easily create, transfer and get transferable tokens on the Hyperledger Fabric network. Through this API giving transferable value to simple assets can generate new markets and gamification strategies.

6.1 API Specification

An abstraction API with all the token functionalities.

6.1.1 Token

A token is an asset that gets a transferable value on the blockchain network and can represent whatever it can be imagined, either an abstract or real/physic thing. The term is a little bit confusing at first, since a token can represent the whole class, in which the total supply is declarated, or individual tokens (balances) transferable between users.

Every token has the following structure:

- `name` : `<string>` Name of the generic token.
- `symbol` : `<string>` Shortname for the token.
- `owner` : `id<string>:company<string>` Token owner. It has to be specified the ID of the owner and the ID of the organization it belongs to inside the organization.
- `ethereumAddress`: `<string>` Address in the Ethereum blockchain, in case we want to link it to a public blockchain token.
- `totalSupply` : `<integer>` Total number of individual tokens issued.

Sample structure (Click to expand)

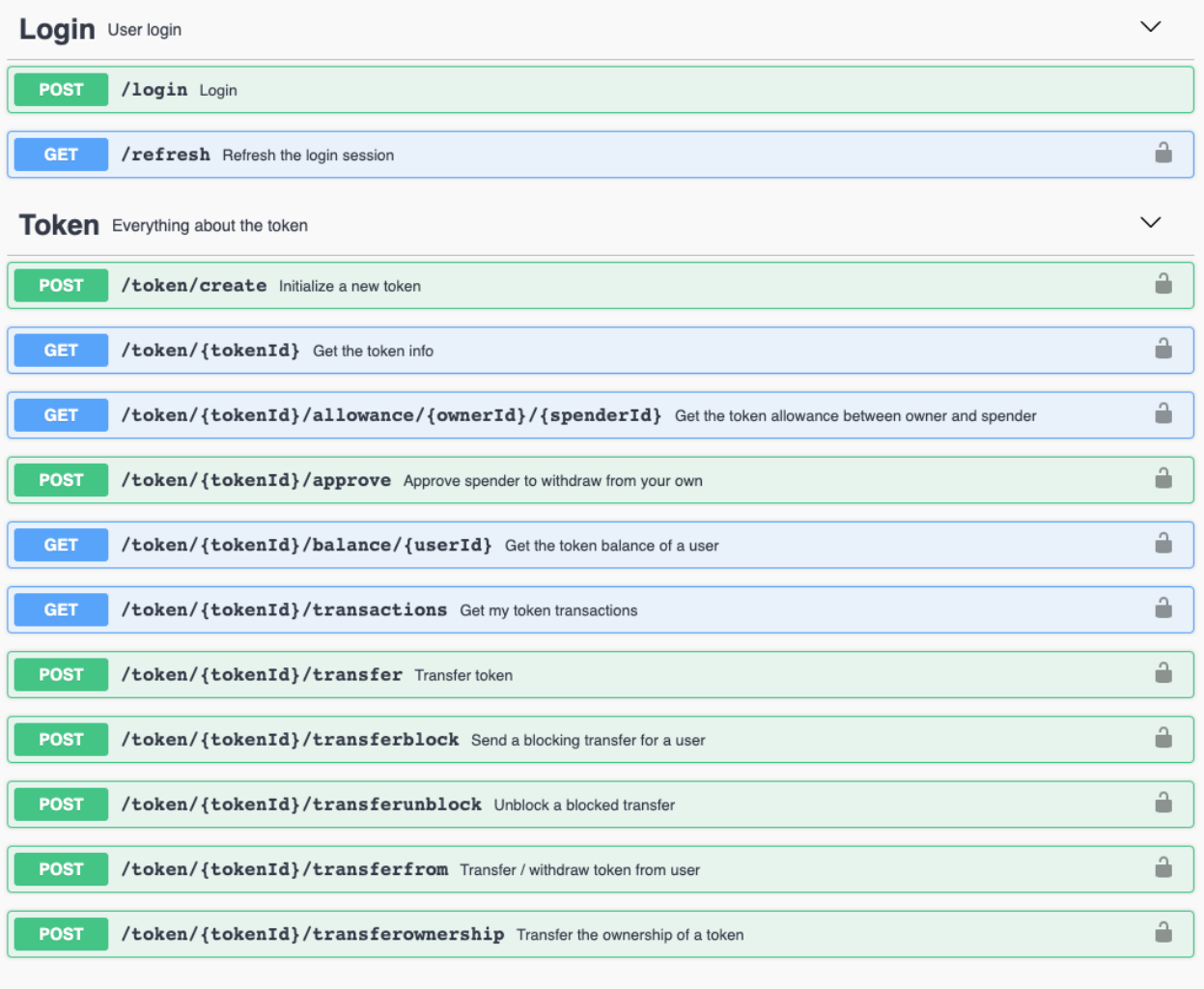
```
{
      "name": "KARMA",
      "symbol": "KRM",
      "owner": "bteam:org1MSP",
```

(continues on next page)

(continued from previous page)

```
    "ethereumAddress": "0x320aty492ua90suf9a0veuf903as3q82",
    "totalSupply": 99999999999999
}
```

6.2 API Methods



Token methods (Click to expand)

6.2.1 POST - `/token/create`

Initialize a new token in the network with some specific information.

Input

- `name` : `<string>` Token name.
- `symbol` : `<string>` Token shortname.

- `owner : identifier<string>:company<string>` Token owner contract.
- `ethereumAddress : <string>` This address represents the smart contract in Ethereum associated to the token. At the moment is given as an input, but in future releases it will be created by the chaincode and given to the developer as a response.
- `totalSupply : <integer>` Total amount of individual tokens that will be created.

Sample structure (Click to expand)

```
{
  "name": "KARMA",
  "symbol": "KRM",
  "owner": "bteam:org1MSP",
  "ethereumAddress": "0x0",
  "totalSupply": 99999999999999
}
```

Output

- `message : <string>`

Sample structure (Click to expand)

```
{
  "output": {
    "name": "KARMA",
    "symbol": "KRM",
    "owner": "bteam:org1MSP",
    "ethereumAddress": "0x0",
    "totalSupply": 99999999999999
  }
}
```

6.2.2 GET - `/token/{tokenId}`

Gets all the token information.

Input

- `tokenId : <string>` Token identifier..

Output

- `token : <string>`

Sample structure (Click to expand)

```
{
  "output": {
    "name": "KARMA",
    "symbol": "KRM",
    "owner": "bteam:org1MSP",
    "ethereumAddress": "0x0",
    "totalSupply": 99999999999999
  }
}
```

6.2.3 GET - /token/{tokenId}/allowance/{owner}/{spender}

This call tells if some specific spender is allowed by some owner to performs actions over the token.

Input

- `tokenId` : <string> Token name.
- `ownerId` : <string> Token owner.
- `spenderId` : <string> Person from whom we want to know how much he is allowed to spend.

Output

- `allowed` : <integer> Quantity of tokens he is allowed to spend.

Sample structure (Click to expand)

```
{
  "output": {
    "allowed": 230
  }
}
```

6.2.4 POST - /token/{tokenId}/approve

Approve a different spender for a amount of token you own.

Input

- `tokenId` : <string> Token name.
- `spender` : <string> Name of the spender user.
- `value` : <int> Amount that is allowed to spend.

Sample structure (Click to expand)

```
{
  "spender": "Satoshi:org1MSP",
  "value": "20"
}
```

Output

- `id` : <string> Transaction id.
- `message` : <string> Message of the approve transaction.

Sample structure (Click to expand)

```
{
  "output": {
    "id": "eaff9d6d289ca1894cffb4bbx0e540de4f58f69eb067a23efba2b7581c77e398",
    "message": "Approve 230 from bteam:org1MSP to Satoshi:org1MSP"
  }
}
```

6.2.5 GET - /token/{tokenId}/balance/{userID}

Gets the token balance of a user.

Input

- `tokenId` : <string> Token name.
- `userId` : <string> User did.

Output

- `balance` : <integer> User's balance.
- `blocked_balance` : <integer> User's blocked balance.

Sample structure (Click to expand)

```
{
  "output": {
    "balance": 950,
    "blocked_balance": 50
  }
}
```

6.2.6 GET - /token/{tokenId}/transactions

Get the token transaction history.

Input

- `tokenId` : <string> Token identifier.

Output

- `list` : <json> List of transactions of the token.

Sample structure (Click to expand)

```
{
  "output": [
    {
      "id": "eaff9d6d289ca4894cffb4bbb0e540de4f58f69eb067a23efba2b7581c77e398",
      "message": "Approve 20 from bteam:org1MSP to Satoshi:org1MSP"
    },
    {
      "id": "5aa6239bb3c45647ab4ffa52759fc1fc981c28f8beb0724765b0c4505fca5a13",
      "message": "Transfer 40 from bteam:org1MSP to Bob:org1MSP"
    },
    {
      "id": "bb7f630b44a42b67abd437d4d9afb0b515fcd4b5794dd8cd6e3bf4f3510c5146",
      "message": "Transfer 40 from bteam:org1MSP to Alice:org1MSP"
    }
  ]
}
```

6.2.7 POST - /token/{tokenId}/transfer

Transfers individual tokens (balances of a token class).

Input

- `tokenId` : <string> Token name.
- `to` : <string> Destination user.
- `value` : <string> Balance to transfer.

Sample structure (Click to expand)

```
{
  "to": "Alice:org1MSP",
  "value": "40"
}
```

Output

- `id` : <string> Transaction id.
- `message` : <string> Message of the approve transaction.

Sample structure (Click to expand)

```
{
  "output": {
    "id": "bb7f630b44a42b67abd437d4d9afb0b515fcd4b5794dd8cd6e3bf4f3510c5146",
    "message": "Transfer 40 from bteam:org1MSP to Alice:org1MSP"
  }
}
```

6.2.8 POST - /token/{tokenId}/transferfrom

Transfer / withdraw from a user. The user has to be approved to spend individual tokens.

Input

- `from` : <string> Person who approves to spend. He has to have a positive balance.
- `to` : <string> Destination user of the funds.
- `value`: <string> Amount of tokens.

Sample structure (Click to expand)

```
{
  "from": "bteam:org1MSP",
  "to": "Satoshi:org1MSP",
  "value": "20"
}
```

Output

- `id` : <string> Transaction id.
- `message` : <string> Message of the approve transaction.

Sample structure (Click to expand)

```
{
  "output": {
    "id": "7876322dbfe61fd2c293f63fd148875d36b3d4fbcec6635d55351bfab2e9e713",
    "message": "TransferFrom 20 from bteam:org1MSP to Satoshi:org1MSP"
  }
}
```

6.2.9 POST - /token/{tokenId}/transferownership

Transfer the ownership of a generic token.

Input

- `tokenId` : <string> Token name.
- `to` : <string> New owner.

Sample structure (Click to expand)


```
{
  "to": "Bob:org1MSP",
}
```


Output

- `token` : <json> Token with updated parameters.

Sample structure (Click to expand)


```
{
  "output": {
    "ethereumAddress": "0x0",
    "name": "KARMA",
    "owner": "Bob:org1MSP",
    "symbol": "KRM",
    "totalSupply": 99999999999999
  }
}
```

6.2.10 POST - /token/{tokenId}/transferblock

Transfer individual tokens as a blocked balance for a user.

Input

- `tokenId` : <string> Token name.
- `to` : <string> Destination user of the blocked funds.
- `value` : <string> Amount of tokens to be blocked.

Sample structure (Click to expand)

```
{
    "to":"Alice:org1MSP",
    "value":"50"
}
```

Output

- `blocked_id` : <string> Transaction id.
- `id` : <string> Transaction id.
- `message` : <string> Message of the approve transaction.

Sample structure (Click to expand)

```
{
  "output": {
    "blocked_id": "7c76500379b86967c04490baa1d25ecc52adfc9df9340d561805b548d8c87e72",
    "id": "7c76500379b86967c04490baa1d25ecc52adfc9df9340d561805b548d8c87e72",
    "message": "Blocking transfer 50 from bteam:org1MSP to Alice:org1MSP"
  }
}
```

6.2.11 POST - `/token/{tokenId}/transferunblock`

Unblocks a blocked transfer being true equivalent to accept the blocked balance and false equivalent to returning the blocked balance to origin user.

Input

- `tokenId` : <string> Token name.
- `blocked_id` : <string> Id of the blocked transaction.
- `accept` : <string> Flag to determine the acceptance or not of the transaction.

Sample structure (Click to expand)

```
{
  "blocked_id": "b1cdd2e5df1d7565edb2c54ee39c0ab9931ea647d04627062cb52500ee82c0af",
  "accept": "true"
}
```

Output

- `id` : <string> Transaction id.
- `message` : <string> Message of the approve transaction.

Sample structure (Click to expand)

```
{
  "output": {
    "id": "e309a55be8b1c842c3aaf475bb7b7b2b759101b09f622fbb5f8dc56aab3d3795",
    "message": "Accepting blocked transaction␣
↪b1cdd2e5df0d7565bdb2c54ee39c0ab9931ea647d04627062cb52500ee82c0af from bteam:org1MSP␣
↪and token amount goes to bteam:org1MSP"
  }
}
```

6.3 How we run the application

As you could see in the Architecture module, all the applications are running on cloud. Through Kubernetes orchestration system the application deployment, scaling and management is an easy and automated task.

6.4 Testing the Application

In postman folder there are the collection and environment to interact and test with the API methods. It is only needed to import them into postman application and know to use the coren-tokenapi module.

Also you can download the files in the links below:

- Postman collection

- Postman environment

6.5 Errors management

Token API errors are managed through the following JSON:

```
{
  "error": {
    "code": "HTTP status code",
    "function": "function in which the error was generated",
    "message": "error description"
  }
}
```

CHAPTER 7

Cert

Cert API is used to create, sign and verify **digital certificates** on the blockchain. The entire functionalities are based on a Smart Contract that handles the lifecycle of every certificate in the most autonomous and secure way.

By using Blockchain, we create trust, ensure security and reduce the risk of fraud, forgery or information leakage. reduce the risk of fraud, forgery or information leakage.

Blockchain adds immutability to business process information:

- Everything is recorded forever.
- Anyone can verify it irrefutably and without repudiation.
- No one can alter it or therefore doubt it.

A certificate is a digital document that is issued at the request of a user and through cryptographic techniques and blockchain storage ensures the existence at a given time of the content being certified and guarantees its integrity and immutability. Each certificate combines 3 evidences to establish with that the user was in possession of the certified content on the date of issuance. content on the date of issuance of the certificate.

- User digital signature
- Digital fingerprint of the content to certificate
- A blockchain timestamp

Any information exchange or registration process can benefit from certificates to provide guarantees of the immutability over time of that information and that none of the parties that may have had access to it have modified it for their own benefit. We can think of certificates as the simple way that, thanks to Blockchain, allows us to create evidence about the existence of a certain content without the involvement of a third party to guarantee it.

Is it possible to create the following certificate types:

- Asset: Certification of an asset. This asset must be created first in the Track API.
- Data: Certification of any kind of information. The data could be documents or free text in string o JSON format.
- NFTs: Certification of Non Fungible Tokens to guarantee its authenticity.

7.1 API Specification

An abstraction API with all the certificate functionalities.

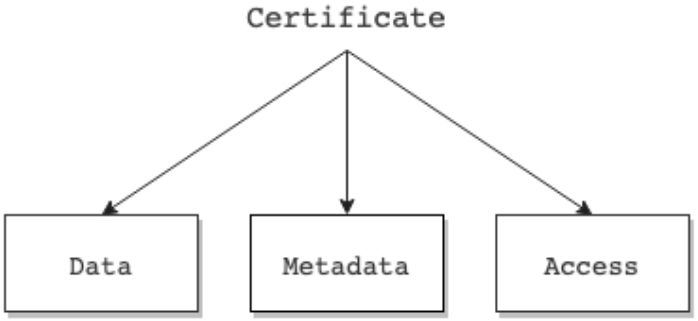
7.1.1 Certificate

A certificate is a tamper-proof and verifiable collection of data that represents a process, a file/document, an accomplishment or any activity that can benefit from the inmmutability and authenticity.

Thanks to certificates, it allows you to **assure any fact in your business case recording the information along with parameters that everyone can verify** (e.g. signatures, issuance/expiration time, network evidences, etc..).

Every certificate is identified by a unique ID and is composed of three complementary but different parts: Data, Metadata and Access.

IMPORTANT: Every certificate can be also identified by a custom ID (`externalId`) that is user's choice. Due to this, the certificate can be consulted both by its id and by its pesonalized id.



Thus a certificate has the following structure:

- `certID` : `<string>` Unique identifier of the certificate.
- `data` : `<json>` JSON of certificate data that is inmutable.
- `metadata`: `<json array>` Array of transactions that feed the certificate (e.g. verification instructions, signatures, revocation and public registration).
- `access` : `<json>` JSON of granted accesses to interact with the certificate (e.g. admin, sign, read access).

Sample Certificate structure (Click to expand)

```
{
  "certID": "...",
  "data": {...},
  "metadata": [...],
  "access": {...}
}
```

Certificate data

Data contains the **inmutable information about the certificate**: who issued it, when it was issued, when it will expire and of course what it is certified and what does it mean.

The certificate `data` contains the following fields:

- `badge` : <json> JSON of **badge information** that characterises the certificate (e.g. name, description, type, content, issuer).
- `issuedOn` : <date> Datetime of when the certificate is issued.
- `expires` : <date> Datetime of when the certificate should no longer be considered valid.
- `hash` : <string> Hash of the data information.

Sample Data structure (Click to expand)

```
{
  "badge":{
      "certID":
↪"did:vtn:c1:certid:c3ff3f3c055ae9eb4884123f0627cd8a4496d8ad381766c963ae496a6cd4669f
↪",
      "name": "Certificate example",
      "description": "Description for the Certificate example",
      "type": "content/asset",            // Type should be: content or asset
      "content": [{...}],           // Content to certify
      "issuer": "did:vtn:trustid:admin"
  },
  "issuedOn": "2020-11-12 12:00:34 +0000 UTC",
  "expires": "2021-12-31 23:59:59 +0000 UTC",
  "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c="
}
```

Certificate metadata

Metadata contains the **additional information that feeds the certificate** in order to update the status of the certificate (e.g. revocation, evidences) or to add required data to be valid and know how to verify it (e.g. verification, signers, signatures).

The certificate `metadata` contains the following fields:

- `certID` : <string> Unique identifier of the certificate.
- `type` : <string> Type of transaction (Signature/Revocation/Evidence/Adding signers).
- `verification` : <json> JSON of **verification information**. It contains the required signers.
- `signatures` : <json> JSON of **signatures** added for the verification of the certificate.
- `public_evidences` : <json> JSON of **certificate evidences** registered in public networks, private databases and other sites.
- `revoked` : <bool> Flag to determine that the certificate is revoked and is not longer valid.
- `datetime` : <string> Datetime of when the transaction (signature, evidence or revocation) is done.
- `hfTxId` : <string> Current transaction identifier in Hyperledger Fabric.
- `hash` : <string> Hash of the transaction.

Sample Metadata structure (Click to expand)

```
{
  "certID":
↪"did:vtn:c1:certid:c3ff3f3c055ae9eb4884123f0627cd8a4496d8ad381766c963ae496a6cd4669f
↪",
  "type": "Signature/Revocation/Evidence/Adding signers",
  "verification": {
```

(continues on next page)

(continued from previous page)

```
      "type":"SignedBadge",
      "signers": [...]              // List of required signers
  },
  "signatures": {...},              // Collection of signatures
  "public_evidences": {...},        // Collection of evidences
  "revoked": false,
  "datetime": "1592568489",
  "hfTxId": "3ae1d6b0f914aee4ce7105ddd65c4cf2dcf160ca398297a13032aaf33b50ed291",
  "hash": "oRj6yKH4EDGCGiKRjHzv3yuqX5wAEzwZFgLnE9jwRIs="
}
```

Certificate access

Access contains **list of authorised users depending on their role** (i.e. admin, sign, read) for a certificate in order to control the possible interaction of every user.

The certificate `access` contains the following fields:

- `admin` : <json> Json of **admin**. An admin can manage, sign and revoke the certificate.
- `sign` : <json> Json of **signers**. A signer can sign only once and read the certificate.
- `read` : <json> Json of **readers**. A reader can only read and verify the certificate.
- `public` : <bool> Flag to determine whether the certificate is public and readable for all users or not.

Sample Access structure (Click to expand)

```
{
  "admin":{
    "did:vtn:admin": 1
  },
  "sign":{
    "did:vtn:signer1": 1,  // Signer has not signed yet
    "did:vtn:signer2": 0   // Signer has already signed
  },
  "read":{
    "did:vtn:reader1": 1,
    "did:vtn:reader2": 1
  },
  "public": false         // The certificate is not visible for all
}
```

7.2 API Methods

API methods (Click to expand)

7.2.1 POST - `/certificate/content/create`

Create a certificate from a specific content like file/document/collection of files on Blockchain. A unique and irrevocable identifier (`certID`) is generated for every certificate.

Input

- `name` : `<string>` Name of the certificate
- `description` : `<string>` Short description of the certificate
- `content` : `<json>` Content to certify (*)
- `public` : `<bool>` Flag to determine whether the certificate is public and readable for all users or not
- `readers` : `<string array>` List of readers, in case it is not public
- `signers` : `<string array>` List of required signers
- `externalId` : `<string>` Optional custom identifier for the certificate
- `expires` : `<string>` Optional parameter to determine the expiration of the certificate in `2020-12-18 09:59:13 +0000 UTC` format, similar to ISO 8601

Sample structure (Click to expand)

```
{
  "name": "ABC Certificate",
  "description": "This certificate is a tamper-proof and valid record of the ABC␣
↪document file",
  "content": {...},
  "public": false,
  "readers": [
    "did:vtn:reader1",
    "did:vtn:reader2"
  ],
  "signers": [
    "did:vtn:signer1",
    "did:vtn:signer2"
  ]
}
```

(*) **Content** field is JSON format and open for every use case. An example is shown below:

Sample Content structure (Click to expand)

```
{
  "content":{
    "file_name": "example.pdf",
    "file_hash": "3aAFa39ho53589gbxCSkFj239y90tiFAa78xZAuo=",
    "file_size": "10KB"
  }
}
```

Output

- `certificate` : `<json>`

Sample structure (Click to expand)

```
{
  "output": {
```

(continues on next page)

(continued from previous page)


```
    "certID":
↪"did:vtn:c1:certid:6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248
↪",
    "data": {
        "badge": {
            "certID":
↪"did:vtn:c1:certid:6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248
↪",
            "name": "ABC Certificate",
            "description": "This certificate is a tamper-proof and valid record of␣
↪the ABC document file",
            "type": "content",
            "content": [
              {
                "file_hash": "3aAFa39ho53589gbxCSkFj239y90tiFAa78xZAuo=",
                "file_name": "example.pdf",
                "file_size": "10KB"
              }
            ],
            "issuer":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
        },
        "issuedOn": "2021-04-28 14:25:06 +0000 UTC"
        "expires": "",
        "hash": "Bs2nFa30Ghu84uwBnjs2aOi53qe6r6YTpjk+o/HB2c="
    },
    "metadata": [
        {
          "type":"Creation",
          "verification": {
            "signers":["did:vtn:signer1","did:vtn:signer2"],
            "type": "SignedBadge"
          },
          "signatures": null,
          "public_evidences": null,
          "revoked": false,
          "datetime": "2021-04-28 14:25:06 +0000 UTC",
          "hfTxId": "3f52c32ed0751d4b77bd5859333761c5180abe151605ff4a7546408ecbe8045c"
        }
    ],
    "access": {
        "admin": {
            "did:vtn:admin": 1
        },
        "sign": {
            "did:vtn:signer1": 1,
            "did:vtn:signer2": 1
        },
        "read": {
            "did:vtn:reader1": 1,
            "did:vtn:reader2": 1
        },
        "public": false
    }
  }
}
```

7.2.2 POST - /certificate/file/hash

Create a unique hash from a file. A unique and irrevocable identifier (certID) is generated for every certificate.

Input

- fileInput: <formData> File from which the hash will be generated.

Output

- File hash : <json>

Sample structure (Click to expand)

```
{
"output": {
    "file_hash": "f05131b2230066bf3b2bed28829bba890cea4fe48abd56e1c9a96aa212315c6h"
  }
}
```

7.2.3 POST - /certificate/file/create

Create a certificate from a specific hash file on Blockchain. A unique and irrevocable identifier (certID) is generated for every certificate.

Input

- name : <string> Name of the certificate
- description : <string> Short description of the certificate
- content : <json> Content to certify (*). An existing file hash previously generated with /certificate/file/hash method
- public : <bool> Flag to determine whether the certificate is public and readable for all users or not
- readers : <string array> List of readers, in case it is not public
- signers : <string array> List of required signers
- externalId : <string> Optional custom identifier for the certificate
- expires : <string> Optional parameter to determine the expiration of the certificate in 2020-12-18 09:59:13 +0000 UTC format, similar to ISO 8601

Sample structure (Click to expand)

```
{
  "name": "ABC Certificate",
  "description": "This certificate is a tamper-proof and valid record of the ABC␣
↪document file",
  "content": {...},
  "public": true
}
```

(*) **Content** field is JSON format and open for every use case. An example is shown below:

Sample Content structure (Click to expand)

```
{
  "content":{
    "file_hash": "2132d2a5fcf50d988fea87d257f0c69f51bb771196856cec9255242661f8d5e6"
  }
}
```

Output

- `certificate : <json>`

Sample structure (Click to expand)

```
{
  "output": {
    "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
    "data": {
        "badge": {
            "certID":
↪"6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
            "name": "ABC Certificate",
            "description": "This certificate is a tamper-proof and valid record of␣
↪the ABC document file",
            "type": "content",
            "content": {
               "file_hash":
↪"2132d2a5fcf50d988fea87d257f0c69f51bb771196856cec9255242661f8d5e6"
              }
            ,
            "issuer":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
        },
        "issuedOn": "2021-04-28 14:25:06 +0000 UTC"
        "expires": "",
        "hash": "Bs2nFa30Ghu84uwBnjs2aOi53qe6r6YTpjk+o/HB2c="
    },
    "metadata": [
        {
          "type":"Creation",
          "verification": {
            "signers": null,
            "type": "SignedBadge"
          },
          "signatures": null,
          "public_evidences": null,
          "revoked": false,
          "datetime": "2021-04-28 14:25:06 +0000 UTC",
          "hfTxId": "3f52c32ed0751d4b77bd5859333761c5180abe151605ff4a7546408ecbe8045c"
        }
    ],
    "access": {
        "admin": {
            "did:vtn:admin": 1
        },
        "public": true
    }
  }
}
```

7.2.4 POST - /certificate/file/full

Generate a hash from a file and create a new certificate from this file hash A unique and irrevocable identifier (certID) is generated for every certificate.

Input

- fileInput: <binary - multipart/form-data> File is sent in binary format as a multipart/form-data content type
- body: <object - multipart/form-data> The rest of the parameters are sent in am object body as a multipart/form-data content type
  - name : <string> Name of the certificate
  - description : <string> Short description of the certificate
  - public : <bool> Flag to determine whether the certificate is public and readable for all users or not
  - readers : <string array> List of readers, in case it is not public
  - signers : <string array> List of required signers
  - externalId : <string> Optional custom identifier for the certificate
  - expires : <string> Optional parameter to determine the expiration of the certificate in 2020-12-18 09:59:13 +0000 UTC format, similar to ISO 8601

Sample structure: Body (Click to expand)

```
{
  "name": "ABC Certificate",
  "description": "This certificate is a tamper-proof and valid record of the ABC␣
↪document file",
  "public": true
}
```

Output

- certificate : <json>

Sample structure (Click to expand)

```
{
  "output": {
    "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
    "data": {
        "badge": {
            "certID":
↪"6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
            "name": "ABC Certificate",
            "description": "This certificate is a tamper-proof and valid record of␣
↪the ABC document file",
            "type": "content",
            "content": {
               "file_hash":
↪"2132d2a5fcf50d988fea87d257f0c69f51bb771196856cec9255242661f8d5e6"
              }
            ,
            "issuer":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
        },
```

(continues on next page)

(continued from previous page)


```
        "issuedOn": "2021-04-28 14:25:06 +0000 UTC"
        "expires": "",
        "hash": "Bs2nFa30Ghu84uwBnjs2aOi53qe6r6YTpjk+o/HB2c="
    },
    "metadata": [
        {
          "type":"Creation",
          "verification": {
            "signers": null,
            "type": "SignedBadge"
          },
          "signatures": null,
          "public_evidences": null,
          "revoked": false,
          "datetime": "2021-04-28 14:25:06 +0000 UTC",
          "hfTxId": "3f52c32ed0751d4b77bd5859333761c5180abe151605ff4a7546408ecbe8045c"
        }
    ],
    "access": {
        "admin": {
            "did:vtn:admin": 1
        },
        "public": true
    }
  }
}
```


7.2.5 POST - `/certificate/asset/create`

Create a certificate from a file/document/collection of files on Blockchain. A unique and irrevocable identifier (`certID`) is generated for every certificate.

Input

- `name` : `<string>` Name of the certificate.
- `description` : `<string>` Short description of the certificate.
- `assetID` : `<string>` Asset to certify (*).
- `public` : `<bool>` Flag to determine whether the certificate is public and readable for all users or not.
- `readers` : `<string array>` List of readers, in case it is not public.
- `signers` : `<string array>` List of required signers.

Sample structure (Click to expand)


```
{
  "name": "Asset Certificate",
  "description": "This certificate is a tamper-proof and valid record of the process",
  "assetID": "asset_example1",
  "public": false,
  "readers": [
    "did:vtn:reader1",
    "did:vtn:reader2"
```


(continues on next page)

(continued from previous page)

```
  ],
  "signers": [
    "did:vtn:signer1",
    "did:vtn:signer2"
  ]
}
```

(*) **assetID** field is used for certify assets created with Track API. If you have already created some assets (f.e. asset_example1, asset_example2, asset_example3) you can choose what asset to certify and also what range of its transactions.

If wanted to certify only transactions within a period or range of time, only you have to determine `init` and `end` params as depicted below:

Sample of asset range certificate (Click to expand)

```
{

  "name": "Asset Certificate",
  "description": "This certificate is a tamper-proof and valid record of the process",
  "assetID": "asset_example1",
  "init": "1600000000",
  "end":"1615555555"
  "public": true
}
```

Output

- `certificate:<json>`

Sample structure (Click to expand)

```
{
  "output": {
    "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
    "data": {
        "badge": {
            "name": "Asset Certificate",
            "description": "This certificate is a tamper-proof and valid record of␣
↪the process",
            "type": "asset",
            "content": [ array of asset transactions ]
        },
        "issuedOn": "2021-04-28 14:25:06 +0000 UTC",
        "expires": "",
        "hash": "Bs2nFa30Ghu84uwBnjs2aOi53qe6r6YTpjk+o/HB2c="
    },
    "metadata": [
        {
          "verification": {
            "signers":["did:vtn:signer1","did:vtn:signer2"]
          },
          "signatures": null,
          "public_evidences": null,
          "revoked": false,
          "datetime": "2021-04-28 14:25:06 +0000 UTC",
          "hfTxId": "3f52c32ed0751d4b77bd5859333761c5180abe151605ff4a7546408ecbe8045c"
        }
```

(continues on next page)

(continued from previous page)

```
    ],
    "access": {
        "admin": {
            "did:vtn:admin": 1
        },
        "read": {
            "did:vtn:reader1": 1,
            "did:vtn:reader2": 1
        },
        "sign": {
            "did:vtn:signer1": 1,
            "did:vtn:signer2": 1
        },
        "public": false
    }
  }
}
```

7.2.6 POST - /certificate/nft

Create a certificate from a specific NFT. The NFT is managed with a smartcontract that maps an owner an a tokenId of the collection. A unique and irrevocable identifier (`certID`) is generated for every certificate.

Input

- `name` : `<string>` Name of the certificate
- `description` : `<string>` Short description of the certificate
- `content` : `<json>` Content to certify (*)
- `public` : `<bool>` Flag to determine whether the certificate is public and readable for all users or not
- `readers` : `<string array>` List of readers, in case it is not public
- `signers` : `<string array>` List of required signers
- `externalId` : `<string>` Optional custom identifier for the certificate
- `expires` : `<string>` Optional parameter to determine the expiration of the certificate in `2020-12-18 09:59:13 +0000 UTC` format, similar to ISO 8601

Content

- `contractAddress` : `<string>` Smartcontract address that manages the nft collection
- `networkId` : `<string>` Blockchain network where the smartcontract is deployed. The networkId can be found in https://chainlist.org/
- `tokenId` : `<string>` Unique id that represents the nft
- 'url': `<string>` Url of the blockexplorer with the smartcontract info (optional)

Sample structure (Click to expand)

```
{
  "name": "Certificate name",
  "description": "Certificate short description",
```

(continues on next page)

(continued from previous page)

```
  "content": {
    "contractAddress": "0x6751D366820d9022624cF0a543A734467b8A6a73",
    "networkId": 1,
    "tokenId": 3,
    "url": "https://polygonscan.com/address/0x6751d366820d9022624cf0a543a734467b8a6a73
↪"
  },
  "public": false,
  "readers": [
    "did:vtn:reader1",
    "did:vtn:reader2"
  ],
  "signers": [
    "did:vtn:signer1",
    "did:vtn:signer2"
  ],
  "externalId": "External certificate identifier"
}
```

(*) **Content** field is JSON format and open for every use case. An example is shown below:

Sample Content structure (Click to expand)

```
{
    "contractAddress": "0x6751D366820d9022624cF0a543A734467b8A6a73",
    "networkId": 1,
    "tokenId": 3,
    "url": "https://polygonscan.com/address/0x6751d366820d9022624cf0a543a734467b8a6a73
↪"
}
```

Output

- `certificate:<json>`

Sample structure (Click to expand)

```
{
  "output": {
    "certID":
↪"did:vtn:channel1:certid:c3ff3f3c055ae9eb4884123f0627cd8a4496d8ad381766c963ae496a6cd4669f
↪",
    "data": {
      "badge": {
        "name": "Certificate name",
        "description": "Certificate description",
        "type": "content",
        "content": [
          {}
        ]
      },
      "issuer":
↪"did:vtn:trustid:afa1f9ad8e593ced39051d1f909b33b0852f18c16e89613bc7e3d2b5ef43a878",
      "issuedOn": "2025-12-31T23:59:59+00:00",
      "expires": "",
      "hash": "Bs2nFa30Ghu84uwBnjs2aOi53qe6r6YTpjk+o/HB2c="
    },
    "metadata": [
```

(continues on next page)

(continued from previous page)


```
      {
        "type": "Creation/Signature/Revocation/Evidence",
        "verification": {},
        "signatures": {},
        "public_evidences": {},
        "revoked": false,
        "datetime": "1592568489",
        "hfTxId": "3ae1d6b0f914aee4ce7105ddd65c4cf2dcf160ca398297a13032aaf33b50ed291",
        "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c="
      }
    ],
    "access": {}
  }
}
```


7.2.7 POST - `/certificate/nft/collection`

Create a certificate of a smart contract collection.

Input

- `name` : `<string>` Name of the certificate
- `description` : `<string>` Short description of the certificate
- `content` : `<json>` Content to certify (*)
- `public` : `<bool>` Flag to determine whether the certificate is public and readable for all users or not
- `readers` : `<string array>` List of readers, in case it is not public
- `signers` : `<string array>` List of required signers
- `externalId` : `<string>` Optional custom identifier for the certificate
- `expires` : `<string>` Optional parameter to determine the expiration of the certificate in `2020-12-18 09:59:13 +0000 UTC` format, similar to ISO 8601

Content

- `contractAddress` : `<string>` Smartcontract address that manages the nft collection
- `networkId` : `<string>` Blockchain network where the smartcontract is deployed. The id can be found in https://chainlist.org/.
- 'url': `<string>` Url of the blockexplorer with the smartcontract info (optional)

Sample structure (Click to expand)


```
{
  "name": "Certificate name",
  "description": "Certificate short description",
  "content": {
    "contractAddress": "0x6751D366820d9022624cF0a543A734467b8A6a73",
    "networkId": 1,
    "tokenId": 3,
    "url": "https://polygonscan.com/address/0x6751d366820d9022624cf0a543a734467b8a6a73
↪"
```


(continues on next page)

(continued from previous page)

```
  },
  "public": false,
  "readers": [
    "did:vtn:reader1",
    "did:vtn:reader2"
  ],
  "signers": [
    "did:vtn:signer1",
    "did:vtn:signer2"
  ],
  "externalId": "External certificate identifier"
}
```

(*) **Content** field is JSON format and open for every use case. An example is shown below:

Sample Content structure (Click to expand)

```
{
    "contractAddress": "0x6751D366820d9022624cF0a543A734467b8A6a73",
    "networkId": 1,
    "tokenId": 3,
    "url": "https://polygonscan.com/address/0x6751d366820d9022624cf0a543a734467b8a6a73
↪"
}
```

Output

- `certificate : <json>`

Sample structure (Click to expand)

```
{
  "output": {
    "certID":
↪"did:vtn:channel1:certid:c3ff3f3c055ae9eb4884123f0627cd8a4496d8ad381766c963ae496a6cd4669f
↪",
    "data": {
      "badge": {
        "name": "Certificate name",
        "description": "Certificate description",
        "type": "content",
        "content": [
          {}
        ]
      },
      "issuer":
↪"did:vtn:trustid:afa1f9ad8e593ced39051d1f909b33b0852f18c16e89613bc7e3d2b5ef43a878",
      "issuedOn": "2025-12-31T23:59:59+00:00",
      "expires": "",
      "hash": "Bs2nFa30Ghu84uwBnjs2aOi53qe6r6YTpjk+o/HB2c="
    },
    "metadata": [
      {
        "type": "Creation/Signature/Revocation/Evidence",
        "verification": {},
        "signatures": {},
        "public_evidences": {},
        "revoked": false,
```

(continues on next page)

(continued from previous page)


```
        "datetime": "1592568489",
        "hfTxId": "3ae1d6b0f914aee4ce7105ddd65c4cf2dcf160ca398297a13032aaf33b50ed291",
        "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c="
      }
    ],
    "access": {}
  }
}
```


7.2.8 GET - /certificate/{certID}

Get certificate from the blockchain.

Input

- `certID` : `<string>` Unique identifier of the certificate

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}`)

Output

- `certificate` : `<json>`

Sample structure (Click to expand)


```
{
  "output": {
    "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
    "data": {
      "badge": {
        "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
        "content": [
          {
            "file_hash": "3aAFa39ho53589gbxCSkFj239y90tiFAa78xZAuo=",
            "file_name": "example.pdf",
            "file_size": "10KB"
          }
        ],
        "description": "This certificate is a tamper-proof and valid record of the␣
↪ABC document file",
        "issuer":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
        "name": "ABC Certificate",
        "type": "content"
      },
      "expires": "",
      "hash": "I6fI6JPNS9y3QGfACbZkAVTOGr5060Y3JF2b/Tl2TCY=",
      "issuedOn": "2021-04-28 14:25:06 +0000 UTC"
    },
    "metadata": [
      {
        "public_evidences": {
          "ethereum": {
            "evidenceHash": "I6fI6JPNS9y3QGfACbZkAVTOGr5060Y3JF2b/Tl2TCY=",
```


(continues on next page)

(continued from previous page)


```
            "smartContract": "0x0C9114b9Ec58d1fbF9FC650BE4B7Aefe481597A0",
            "timestamp": 1619621204,
            "transaction":
↪"0x331bf9fd1514cb41434c594b18f4b848783b3807004ebd68c9ca73a77c7ee48b"
          },
          "besu": {
            "evidenceHash": "I6fI6JPNS9y3QGfACbZkAVTOGr5060Y3JF2b/Tl2TCZ=",
            "smartContract": "0x0C8114b9Ec58d1fbF9FC650BE4B7Aefe481597A1",
            "timestamp": 1619621267,
            "transaction":
↪"0x441bf9fd1514cb41434c594b18f4b848783b3807004ebd68c9ca73a77c7ee48b"
          }
        },
        "revoked": false,
        "signatures": null,
        "verification": {
          "signers": [
            "did:vtn:signer1",
            "did:vtn:signer2"
          ],
          "type": "SignedBadge"
        }
      }
    ],
    "access": {
      "admin": {

↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1":␣
↪1
      },
      "read": {
        "did:vtn:reader1": 1,
        "did:vtn:reader2": 1
      },
      "sign": {
        "did:vtn:signer1": 1,
        "did:vtn:signer2": 1
      }
    }
  }
}
```


7.2.9 GET - `/certificate/file/{file_hash}`

Get a certificate from the file hash

Input

- `file_hash` : `<string>` Certificate file hash

Output

- `certificate` : `<json>`

Sample structure (Click to expand)

The following certificate has been queried using the `file_hash`: `3aAFa39ho53589gbxCSkFj239y90tiFAa78xZAuo=`

```
{
  "output": {
    "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
    "data": {
      "badge": {
        "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
        "content": [
          {
            "file_hash": "3aAFa39ho53589gbxCSkFj239y90tiFAa78xZAuo=",
            "file_name": "example.pdf",
            "file_size": "10KB"
          }
        ],
        "description": "This certificate is a tamper-proof and valid record of the␣
↪ABC document file",
        "issuer":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
        "name": "ABC Certificate",
        "type": "content"
      },
      "expires": "",
      "hash": "I6fI6JPNS9y3QGfACbZkAVTOGr5060Y3JF2b/Tl2TCY=",
      "issuedOn": "2021-04-28 14:25:06 +0000 UTC"
    },
    "metadata": [
      {
        "public_evidences": {
          "ethereum": {
            "evidenceHash": "I6fI6JPNS9y3QGfACbZkAVTOGr5060Y3JF2b/Tl2TCY=",
            "smartContract": "0x0C9114b9Ec58d1fbF9FC650BE4B7Aefe481597A0",
            "timestamp": 1619621204,
            "transaction":
↪"0x331bf9fd1514cb41434c594b18f4b848783b3807004ebd68c9ca73a77c7ee48b"
          },
          "besu": {
            "evidenceHash": "I6fI6JPNS9y3QGfACbZkAVTOGr5060Y3JF2b/Tl2TCZ=",
            "smartContract": "0x0C8114b9Ec58d1fbF9FC650BE4B7Aefe481597A1",
            "timestamp": 1619621267,
            "transaction":
↪"0x441bf9fd1514cb41434c594b18f4b848783b3807004ebd68c9ca73a77c7ee48b"
          }
        },
        "revoked": false,
        "signatures": null,
        "verification": {
          "signers": [
            "did:vtn:signer1",
            "did:vtn:signer2"
          ],
          "type": "SignedBadge"
        }
      }
    ],
    "access": {
      "admin": {

↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1":␣
↪1
```


(continues on next page)

(continued from previous page)

```
      },
      "read": {
        "did:vtn:reader1": 1,
        "did:vtn:reader2": 1
      },
      "sign": {
        "did:vtn:signer1": 1,
        "did:vtn:signer2": 1
      }
    }
  }
}
```

7.2.10 POST - /certificate/{certID}/sign

Sign a certificate.

Input

- `certID` : <string> Unique identifier of the certificate.

(*) You can also use the method with the custom ID:

- `externalId` : <string> Custom identifier of the certificat

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/sign`)

Output

- `certificate` : <json>

Sample structure (Click to expand)

```
{
  "output": {
    "datetime": "2021-04-28 14:42:42 +0000 UTC",
    "hfTxId": "0a55b28d171bfdbadc2f3e92cae89a998ed9531267c658dd1bd658004fa98df7",
    "signature":
↪"eyJjdHkiOiJqd2sranNvbiIsImFsZyI6IlJTMjU2Iiwia2lkIjoidS1MWnp5RnprNEMyOUNkdjh2T1pVd0tXSlo4WUxJdmZxbX
↪eyJwYXlsb2FkIjp7ImJhZGdlIjp7ImNlcnRJRCI6IjY0MDRiMjU0ZjAwOGFjZGE2ZDU1ZjY4ZGVlNDgzMDRmY2YzNmM3M2NmODc
↪kt1yX2MRxMGL4rZqokxxFQy5AcUNP4824VFTEFf-W7DoExYNeh-
↪xKl1oIwiJKRdO6WIGMIiDPPwd2l0WT3m8jdsy6KMyHMGzimG3GoQUTBLd4lTPjQL-
↪93f5OkLQJqut77iX2lVblI3HmvLdV_Vx7-IqFwIFszaULHh8TJzxNCFXOy9MpJhUUfk0abExK_
↪2NaYlvif6y6WeQ1pCrjoOi7fnO0mSzyVXhvlRB2WTXLjJ4cD-lsljdg91KRcnFucoWZu9sQYpt6Gv_
↪rvKPJaejeGCGGjqUEifXwEHBEP6soRLyvcM1TF5TDql03xTd4chP4GDC0uTMXJwNpiX9GyTe6g",
    "signer": {
      "controller":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
      "id":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
      "publicKey": "-----BEGIN PUBLIC KEY-----
↪MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtxqRLjAv2spWbBzjqi8uM+mgNo0EbYSbulfUM2BwcAuCi3+lD+Z8Te
↪ut+ArK45l1ZhyEKURYQ92rQBZMwWkRw4G0bTrSS7f/8OH2V+/
↪9LO8SpeZKhOoY15wGBZbO+OTk2nl03SkbcLe2QugnZ3CXXYU6LPfU4P2HgeugZXF+HwGBWJWMG6mwpKGRX9ba8qzyZMJ0xKUx6
↪+EprVXIdVdtizbxbgNH/PomFqgfyObJMSNaNw8odj52NEFwvfy7UbM90Sm693IiVE7wIDAQAB-----END␣
↪PUBLIC KEY-----"
```

(continues on next page)

(continued from previous page)

```
    },
    "type": "SignedBadge"
  }
}
```

7.2.11 POST - /certificate/{certID}/sign/external

Sign a certificate with external identity and keys.

Input

- `certID` : <string> Unique identifier of the certificate.
- `signature` : <string> Signature.
- `publicKey` : <string> External public key to verify the signature.
- `did` : <string> Signer DID identifier or internal identifier.

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/sign/external`)

Sample structure (Click to expand)

```
{
  "signature": "eyJSosd289ap389fa8uf3u8u4er8912rz....",
  "publicKey": "--------BEGIN PUBLIC KEY-------- ... --------END PUBLIC KEY--------",
  "did": "did:external:signer3"
}
```

Output

- `certificate` : <json>

Sample structure (Click to expand)

```
{
  "output": {
    "datetime": "2021-04-28 14:45:34 +0000 UTC",
    "hfTxId": "42dab150121afe3db840101e5b648fa0f7e3413248314a018ef28ebef63e82d8",
    "signature": "eyJSosd289ap389fa8uf3u8u4er8912rz...",
    "signer": {
        "controller":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
        "id": "did:external:signer3",
        "publicKey": "-----BEGIN PUBLIC KEY----- ... -----END PUBLIC KEY-----"
    },
    "type": "SignedBadge"
  }
}
```

7.2.12 POST - /certificate/{certID}/register?networkId=integer

Register a certificate evidence in a public network.

Input

- certID : <string> Unique identifier of the certificate.
- networkId: <integer> Flag to identify the public network on which the trustpoint will be registered. Currently the following network ID are available:
  - Ethereum mainnet: 1
  - Goerli network (ETHEREUM): 5
  - Polygon maninnet: 137
  - Mumbai network (POLYGON): 80001
  - Alastria network (HYPERLEDGER BESU): 101
  - LACChain network: 102

(*) You can also use this method replacing the certID in the URL with the custom identifier externalId like this (/certificate/{externalId}/register?networkId=integer)

Sample structure (Click to expand)

```
`/certificate/{certID}/register?networkId=5 // Goerli network (Ethereum)
```

Output

- certificate : <json>

Sample structure (Click to expand)

```
{
  "output": {
      "datetime": "2021-04-28 14:46:53 +0000 UTC",
      "hash": "kKI+m663XKazuZ+Orbt/oCQp5rAQJQ7kjYwpqNMnMy4=",
      "hfTxId": "0b9743f8ae6529b4c3bb051202568598facb29227c8d819047db4a63452df37c",
      "type":"Evidence",
      "verification": {...},
      "signatures": {...},
      "public_evidences": {
        "1": {
            "evidenceHash": "I6fI6JPNS9y3QGfACbZkAVTOGr5060Y3JF2b/Tl2TCY=",
            "smartContract": "0x0C9114b9Ec58d1fbF9FC650BE4B7Aefe481597A0",
            "timestamp": 1619621204,
            "transaction":
↪"0x331bf9fd1514cb41434c594b18f4b848783b3807004ebd68c9ca73a77c7ee48b"
        }
      },
      "revoked": false
    }
}
```

7.2.13 GET - /certificate/register/providers

Get all the available providers to register the certificate

Output

- `ouput` : <json> A list with all the available providers

Sample structure (Click to expand)


```
{
  "output": {
    "ethereum": "1",
    "goerli": "5",
    "polygon": "137",
    "mumbai": "80001",
    "besu": "101",
    "lacchain": "102"
  }
}
```

7.2.14 POST - /certificate/{certID}/revoke

Revoke a certificate.

Input

- `certID` : <string> Unique identifier of the certificate.

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/revoke`)

Output

- `certificate` : <json>

Sample structure (Click to expand)


```
{
  "output": {
      "datetime": "2021-04-28 14:48:51 +0000 UTC",
      "hash": "NeeF0whDLcqkq4I8e5p9Yuiz7UZ2UVu68LZvIsBJhg8=",
      "hfTxId": "20b750b8b54dd81d95177048d185ecc17a5ba42ac5351b8915d59d99c2c77967",
      "type": "Revocation",
      "signatures": {...},
      "public_evidences": {...},
      "revoked": true
      }
}
```

7.2.15 GET - /certificate/{certID}/history

Get all transactions for the whole lifecycle of the certificate.

Input

- `certID` : <string> Unique identifier of the certificate.

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/history`)

Output

- `Certificate transactions` : <string> A list of all transactions.

Sample structure (Click to expand)


```
{
  "output": {
    "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
    "data": {
      "badge": {
        "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
        "name": "ABC Certificate",
        "description": "This certificate is a tamper-proof and valid record of the␣
↪ABC document file",
        "issuer":
↪"did:vtn:trustid:76ce288f169fdd9b90a2b9b6a11700fbd80123093f3296e235ab10c27eb306c1",
        "type": "content",
        "content": [...]
      },
      "issuedOn": "2025-12-31T23:59:59+00:00",
      "expires": "",
    },
    "metadata": [
      {
      "datetime": "2021-04-28 14:48:51 +0000 UTC",
      "hash": "NeeF0whDLcqkq4I8e5p9Yuiz7UZ2UVu68LZvIsBJhg8=",
      "hfTxId": "20b750b8b54dd81d95177048d185ecc17a5ba42ac5351b8915d59d99c2c77967",
      "type": "Revocation",
      "verification": {...},
      "signatures": {...},
      "public_evidences": {...},
      "revoked": true
    },
    {
      "datetime": "2021-04-28 14:46:53 +0000 UTC",
      "hash": "kKI+m663XKazuZ+Orbt/oCQp5rAQJQ7kjYwpqNMnMy4=",
      "hfTxId": "0b9743f8ae6529b4c3bb051202568598facb29227c8d819047db4a63452df37c",
      "type": "Evidence",
      "verification": {...},
      "signatures": {...},
      "public_evidences": {...},
      "revoked": false
    },

      ...

    ],
    "access": {...}
  }
}
```

7.2.16 GET - /certificates

Lists all the certificates of a user.

Input N/A. It returns all the certificates which belong to the logged user.

Output

- `Certificate list : <json>`

Sample structure (Click to expand)


```
{
  "output": {
    "certID": [
      "0520dc4c155c06c5319308f245dffb318b9c9ed3b30cb14f0ded59640d193960",
      "060c60c2b1c6adfcbbff3e3d2cdb01184910998322b5914f91d19d16e063490b",
      "084407ec6851fa7bf0753a51071ca2ba6a65d47085fdb5e47e66c9b7c7c73548",
      "0a1d198ed8a2822fc45cd4e749e4c6c1897046b0e6ccdb4dac4488cbbf71b2d7"
    ]
  }
}
```

7.2.17 GET - /certificate/{certID}/access

Get all granted accesses for the certificate.

Input

- `certID : <string>` Unique identifier of the certificate.

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/access`)

Output

- `Access : <string>` A list of all accesses.

Sample structure (Click to expand)


```
{
  "output": {
      "admin": {
        "did:vtn:admin": 1
      },
      "read": {
        "did:vtn:reader1": 1,
        "did:vtn:reader2": 1
      },
      "sign": {
        "did:vtn:signer1": 1,
        "did:vtn:signer2": 1
      },
      "public": false
  }
}
```

7.2.18 POST - /certificate/{certID}/access

Authorise the reading access for a certificate. The user has to be the owner (admin) of the certificate. Readers are overwriten in every call.

Input

- `certID` : <string> Unique identifier of the certificate.
- `public` : <bool> Flag to determine whether the certificate is public and readable for all users or not.
- `readers` : <string array> List of readers, in case it is not public.

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/access`)

Sample structure (Click to expand)

```
{
  "output": {
    "public": false,
    "readers": ["did:vtn:reader3","did:vtn:reader4"]
  }
}
```

Output

- `Access` : <json> A list of all accesses.

Sample structure (Click to expand)

```
{
  "output": {
      "admin": {
        "did:vtn:admin": 1
      },
      "read": {
        "did:vtn:reader3": 1,
        "did:vtn:reader4": 1
      },
      "sign": {
        "did:vtn:signer1": 1,
        "did:vtn:signer2": 1
      },
      "public": false
  }
}
```

7.2.19 GET - /certificate/{certID}/signers

Get the signers and their signatures for the certificate.

Input

- `certID` : <string> Unique identifier of the certificate.

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/signers`)

Output

- `Signers` : <string> A list of all signers with their signature and public key.

Sample structure (Click to expand)

```
{
  "output": {
    "did:vtn:signer1": {},
    "did:vtn:signer2": {}
  }
}
```

7.2.20 POST - `/certificate/{certID}/signers/add`

Add new signers to the certificate.

Input

- `certID` : <string> Unique identifier of the certificate.
- `signers`: <string array> List of new signers.

(*) You can also use this method replacing the `certID` in the URL with the custom identifier `externalId` like this (`/certificate/{externalId}/signers/add`)

Sample structure (Click to expand)

```
{
  "signers": ["did:vtn:signerNEW", "did:vtn:signerNEW2"]
}
```

Output

- `Metadata` : <json> A JSON of current metadata information.

Sample structure (Click to expand)

```
{
  "output": {
      "certID": "6404b254f008acda6d55f68dee48304fcf36c73cf881cdeff478b7b4a2545248",
      "type":"Adding signers",
      "verification":{
        "type":"SignedBadge",
        "signers":[
            "did:vtn:signer1",
            "did:vtn:signer2",
            "did:vtn:signerNEW",
            "did:vtn:signerNEW2"
        ]
      },
      "signatures": {...},
      "public_evidences": {...},
      "revoked": false,
      "datetime": "1592568489",
      "hfTxId": "3ae1d6b0f914aee4ce7105ddd65c4cf2dcf160ca398297a13032aaf33b50ed291",
      "hash": "Ni7JYQG6GSmlEjWoRj2xrfF6ZVFhqBDPzyjk+o/HB2c="
    }
}
```

7.2.21 POST - /certificate/{certID}/advancedsign/init

Initialise advanced signature flow for a certificate.

Input

- certID : <string> Unique identifier of the certificate.
- platform: <string array> Third party platform to use for advanced signature.
- signers: <json array> List of advanced signers that will be part of the advanced signature flow (email, name and internal identifier).

(*) You can also use this method replacing the certID in the URL with the custom identifier externalId like this (/certificate/{externalId}/advancedsign/init)

Sample structure (Click to expand)

```
{
  "platform": "VIDSigner",
  "signers": [
    {
      "email": "signer1@test.com",
      "name": "Signer1 Test",
      "did": "internalID1"
    },
    {
      "email": "signer2@test.com",
      "name": "Signer2 Test",
      "did": "internalID2"
    }
  ]
}
```

Output

- Metadata : <json> A JSON of current metadata information.

Sample structure (Click to expand)

```
{
  "output": {
    "datetime": "2021-04-28 14:20:43 +0000 UTC",
    "hash": "jpt8fQ/+Ml71jR1TO5F5aCNAhlB+j69xt1IfJsBu6AY=",
    "hfTxId": "920d649508240961096ba7a4c1a003be90c8ea5566003d1ce73e7200b0f6c57d",
    "type": "Advanced signature VIDSigner",
    "public_evidences": null,
    "revoked": false,
    "signatures": {
      "advanced": {
        "controller":
↪"did:vtn:trustid:31171f9390a4a048e00b7c4691316f362de8d3e3de69eeb0530c9b286a288509",
        "flowID": "6320b607-ae30-4968-9ef9-6a4d2742f68f",
        "platform": "VIDSigner",
        "status": "Pending",
        "type": "AdvancedSignedBadge"
      }
    },
    "verification": {
      "advancedSigners": [
```

(continues on next page)

(continued from previous page)

```
        "internalID1",
        "internalID2"
      ],
      "signers": null,
      "type": "AdvancedSignedBadge"
    }
  }
}
```

7.2.22 GET - /certificate/{certID}/advancedsign/status

Get the signers and their signatures for the certificate.

Input

- certID : <string> Unique identifier of the certificate.

(*) You can also use this method replacing the certID in the URL with the custom identifier externalId like this (/certificate/{externalId}/advancedsign/status)

Output

- Status : <json> A JSON of status information about the advanced signature flow and the respective signers.

Sample structure (Click to expand)

```
{
  "output": {
    "flowID": "97877dd3-56de-44d1-a38d-be783d01e7eb",
    "AdditionalData":
↪"a82be0e41b91f9de7d76f4974fd747522aac58d3b93e3180ab7676e3b4e33ba2",
    "DocGUI": "97877dd3-56de-44d1-a38d-be783d01e7eb",
    "DocStatus": "Unsigned",
    "Downloaded": false,
    "FileName": "certificate.pdf",
    "Signers": [
      {
        "FormInfo": null,
        "NumberID": "did.vtn.trustid.advancedsigner1",
        "OperationTime": null,
        "RejectionReason": null,
        "SignatureStatus": "Unsigned",
        "SignerGUI": "c8e6cf1b-a296-4046-920d-037b617f6740",
        "SignerName": "Signer1 Test",
        "TypeOfID": "TrustID",
        "UserNoticesInfo": null
      },
      {
        "FormInfo": null,
        "NumberID": "did.vtn.trustid.advancedsigner2",
        "OperationTime": null,
        "RejectionReason": null,
        "SignatureStatus": "Unsigned",
        "SignerGUI": "036c6673-a7e1-476f-9eaf-65b497450a8d",
        "SignerName": "Signer2 Test",
```

(continues on next page)

(continued from previous page)


```
        "TypeOfID": "TrustID",
        "UserNoticesInfo": null
      }
    ]
  }
}
```


7.2.23 GET - /certificate/{certID}/advancedsign/document

Get the signers and their signatures for the certificate.

Input

- certID : <string> Unique identifier of the certificate.

(*) You can also use this method replacing the certID in the URL with the custom identifier externalId like this (/certificate/{externalId}/advancedsign/document)

Output

- Document : <json> A JSON with the document signed/pending to be signed (DocContent) in base64 format and other useful information.

Sample structure (Click to expand)


```
{
  "output": {
    "flowID": "97877dd3-56de-44d1-a38d-be783d01e7eb",
    "AdditionalData":
↪"a82be0e41b91f9de7d76f4974fd747522aac58d3b93e3180ab7676e3b4e33ba2",
    "DocContent": "JVBERi0xLjQKJeLjz9MKNS...",
    "FileName": "certificate.pdf"
  }
}
```


7.2.24 POST - /certificate/advancedsign/notification

Receive notifications from any advanced signature flow.

This method and route requires a special authorization process, don't hesitate to ask us in order to provide you more detailed information.

7.3 How we run the Application

As you could see in the Architecture module, all the applications are running on cloud. Through Kubernetes orchestration system the application deployment, scaling and management is an easy and automated task.

7.4 Testing the Application

In postman folder there are the collection and environment to interact and test with the API methods. It is only needed to import them into postman application and know to use the coren-certapi module.

Also you can download the files in the links below:

- Postman collection

7.5 Errors management

Cert API errors are managed through the following JSON

```
{
  "error": "error description"
}
```

CHAPTER 8

ID

An abstraction TrustID API implementation. It wraps the functionality of the TrustID SDK in order to offer basic identity management services to users comfortable delegating the responsability of their keys to a custodian. ID API acts as the third-party custodian of the users keys.

8.1 API Specification

The implementation considers that the API is the third-party custodial of TrustID keys. Currently the keystore is implemented with a FileKeystore, MongoKeystore comming soon. Private keys from users are stored ciphered with a passphrase. In order to call every function and unlock the account the passphrase needs to be provided. The API has the following routes.

8.1.1 Identity Methods

POST - `/id/login`

It return a JWT to interact with services authenticated using JWT using TrustID as identity backend. This functionality is offered for every user in the system (even those for which we are not the custodials of the keys). Some services may be still authenticated with JWT, and we want to support this authentication even of TrustID users.

- `id` : `<string>` Unique identifier of the user.
- `password` : `<string>` Password of the user.

Sample structure (Click to expand)

```
{
  "id": "did:vtn:trustid:ae0213fncasdf234",
  "password": "pass",
}
```

POST - /id/refresh

It refreshes a JWT login.

POST - /id/create

Create a new identity with the key determined in type and a passphrase to lock the private key.

- id : <string> Unique identifier of the user.
- type : <string> Algorithm used, right now only RSA is supported.
- channel: <string> Identifier of the HF network, only required for admin user.

Sample structure (Click to expand)

```
{
  "password": "passphrase",
  "type": "RSA",
  "channel" "channel1"
}
```

POST - /id/get

Gets the information from a registered identity.

- id : <string> Unique identifier to query the identity.

Sample structure (Click to expand)

```
{
  "id": "did:vtn:service:012343",
  "password": "passphrase"
}
```

POST - /id/revoke

Sends a request to revoke a specific identity. You need to be the owner of the identity or its controller to perform this task.

- id : <string> Identity that is going to be revoked.

Sample structure (Click to expand)

```
{
  "id": "did:vtn:test",
  "password": "passphrase"
}
```

POST - /id/verify

Verification of a registered identity by a controller in the system.

- id : <string> Identity that is going to be verified for the user that had been logged.

Sample structure (Click to expand)

```
{
  "id": "did:vtn:test",
  "password": "passphrase"
}
```

POST - /id/update/password

Updates the old password with a new one.

- password: <string> New password selected.
- oldPassword : <string> Actual password.

Sample structure (Click to expand)

```
{
  "password": "newpass"
  "oldPassword": "test"
}
```

8.1.2 Identity Recovery Methods

POST - /id/recover/create

Creates the mechanism to recover the user password. Recovery operation needs an email to send the recovery code in case of password change and a group of custodians, in order to have a social recovery wallet. Each custodian will receive a secret that must be provided when the user wants to recover the password. The minumum number of guardians is two. Then in order to recover, you only need at least one of the secrets provided by the guardians.

- email: <string> Email that will own the user in order to recover their identity.
- guardians : <array> List of guardians.

Sample structure (Click to expand)

```
{
  "guardians": [
    {
      "value": "bob@email.com",
      "type": "email"
    }
  ],
  "email": "alice@email.com"
}
```

(*) Please navigate to the following *section* for details of the mail communication service.

GET - /id/recover/info

Gets the recovery info associated to the session. It returns the email of the user and the guardians.

POST - /id/recover/update/guardian

Updates the guardian information. It's necessary to provide the info of the oldGuardian in order to change it. It's necessary to specify the internalMailService query parameter. As result it's returned the new secret for the new guardian or an email is sent.

- `newGuardian`: <string> Email of the new guardian.
- `oldGuardian`: <string> Email of the old guardian.
- `secret` : <array> Old guardian's secret.

Sample structure (Click to expand)

```
{

  "newGuardian": "alice@email.com",
  "oldGuardian": "bob@email.com",
  "secret": "secret"
}
```

(*) Please navigate to the following *section* for details of the mail communication service.

POST - /id/recover/update/email

Updates the recovery email associated with the identity in order to recover the identity. The email must match with the one registered in /id/recover/create.

- `email`: <string> The new user mail.

Sample structure (Click to expand)

```
{
  "email": "john@email.com"
}
```

POST - /id/recover/init

Initialises the recovery process in order to recover an identity account, the user will receive an email with a sigle use code.

- `email`: <string> Email to send the verification code to the user in order to start the account recovery process.

Sample structure (Click to expand)

```
{
  "email": "john@email.com"
}
```

(*) Please navigate to the following *section* for details of the mail communication service.

POST - /id/recover/password

Finishes the process to recover the password. The params required are the single use code received in the users email and the secrets that were sent to the guardians. You need at least one of the secrets provided by the guardians.

- `code`: <string> An single use access code sent to the user mail in /id/recover/init function.

- `guardians`: `<array,objects>` Email of the guardiands and their respective secrets.

Sample structure (Click to expand)

```
{
  "code": "a12f120c912b12e12bd",
  "guardians": [
    {
      "secret": "234895738256",
      "email": "bob@email.com"
    }
  ],
  "newPassword": "testPassword"
}
```

8.1.3 OpenID Methods

GET - `/openId/authorizationUrl`

Gets the OpenID autorization URL to initiate an authentication flow.

Input

- `authCallback` : `<string>` Authorization callback endpoint where identity token is served. It should be the final application.
- `identityProvider` : `<string>` Identity provider which issues the token. Integration with Google and Microsoft is currently available.

Output

- `message` : `<json>` An authorization URL to request the identity token.

Sample structure (Click to expand)

```
{
  "message": {
    "authorizationUrl": "https://accounts.google.com/o/oauth2/v2/auth?client_
↪id=3168832523525320-4cb3nmfohpvh3c4p5pstnd6url5k80md.apps.googleusercontent.com&
↪scope=openid%20email%20profile&response_type=code&redirect_uri=http%3A%2F
↪%2Flocalhost%3A9090%2Fopenid%2FauthorizationCallback"
  }
}
```

GET - `/openId/authorizationToken`

Gets the autorization token generated for the requesting identity in the JWT standard.

Input

- `code` : `<string>` Authorization code served to the application needed to request the authorization token.

Output

- `output` : `<string>` The identity token in the JWT standard and the TrustOS user identifier.

Sample structure (Click to expand)

```
{
  "output": {
   "did":
↪"did:vtn:trustid:d1a33da017ad4d7a0edb34b5906f703fa17765cdbfb8db5f60baaa8a8681fd1f",
   "jwt":
↪"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6ImJXOFpjTWpCQ25KWlMtaWJYNVVRRE5TdHZ4NCJ9.
↪eyJ2ZXIiOiIyLjAiLCJpc3MiOiJodHRwczovL2xvZ2luLm1pY3Jvc29mdG9ubGluZS5jb20vOTE4ODA0MGQtNmM2Ny00YzViLWI
↪yhaqvrx9VUmTGJNGTZQYMv-jRIv4OP-sYb4UP4XRADY6FAn7-G5OHj_VHc96yHzPxx-
↪2wxzzrPGF4hxA9T4M5FSuuw4LqrQXT5H7oey27LYAE_gFJ8Uo8xY0fMb3yh8Q6HPbmAEuPjLHi_
↪X6roB1dZR3AZj1OnxQr4_0wZRKpvBOkvX943bJtvMKYdrewVaqlI9JfxsPX89pdGh9zNqkyU6mdqAKt_
↪BeucotUQ-Kbpj9Y_mYzXKiHxMzN_uhM_UtEeY_hbpBf87YrC9Jd9Iv3oBZ3aSTLXHHpN8tRrywRXAG-
↪f23TgldldjTuCIuTXVg2xiGJwo9tIGoH0dRmnUl_Q"
    }
  }
```


8.1.4 Signature methods

POST - `/id/sign`

Requests a signature using a key in custody.

Body

- `payload` : `<json>` Payload that needs to be signed.

Sample structure (Click to expand)

```
{
  "payload": {
    "arg1": "test",
    "arg2": "payload"
  }
}
```

POST - `/id/validate`

Validates a signature using a key in custody.

- `id`: `<string>` Identifier of the signer.
- `payload` : `<string>` Payload that needs to be validated.

Sample structure (Click to expand)

```
{
  "id"
  "payload": "eyaqwuqiiqiwiqaqiemasoiaiqiiaskqweioqweiuwe"
}
```

8.1.5 Services methods

POST - `/service/create`

Create a new service in the TrustID network.

- `serviceID` : <string> Identity that is going to be verified for the user that had been logged.
- `name` : <string> Name of the service deployed in Hyperledger Fabric.
- `access` : <json> Access policy that will have the service: PUBLIC, SAME_CONTROLLER, FINE_GRAINED.
- `channel` : <string> Identifier of the HF network.

Sample structure (Click to expand)

```
{
  "serviceID":  "track",
  "name": "trackscc",
  "access": {
    "policy": "PUBLIC"
  },
  "channel": "channel1"
}
```

POST - /service/get

Gets the registered information for a service.

- `serviceID` : <string> Identity that is going to identify the service.

Sample structure (Click to expand)

```
{
  "serivceID": "did:vtn:service:012343",
  "password": "passphrase"
}
```

POST - /service/updateAccess

Updates the access of a service.

- `serviceID` : <string> Identity that is going to identify the service.
- `access` : <json> Access policy that will have the service: PUBLIC, SAME_CONTROLLER, FINE_GRAINED.

The supported policies are the following:

- PUBLIC, everyone with access on the network can call the service.
- SAME_CONTROLLER, identities with the controller that created the service, can call the service.
- FINE_GRAINED, only specific identities can access to the service.

Sample structure (Click to expand)

```
{
  "serviceID":  "did:vtn:service:sacc",
  "access": {
    "policy": "FINE_GRAINED",
    "threshold": 0,
    "access": {

↪"did:vtn:trustid:c0fd6b4749329c4acec7f4ac273d46c2b755736e9f5cae6fc62acec8d04549c6":␣
↪2
```

(continues on next page)

(continued from previous page)

```
    }
  }
}
```

POST - /service/update

Updates the information of a service.

- serviceID : <string> Identity that is going to identify the service.
- name : <string> Name of the service deployed in Hyperledger Fabric.
- channel : <string> Identifier of the HF network.

Sample structure (Click to expand)

```
{
  "serviceID":  "did:vtn:service:sacc",
  "channel": "channel1",
  "name":"sacc-chaincode"
}
```

POST - /service/invoke

Invoke a function from a distributed service using TrustID.

- serviceID : <string> Identity that is going to identify the service.
- function : <string> Function that is going to be called.
- args : <string> Args of the function that is going to be called.

Sample structure (Click to expand)

```
{
  "serviceID": "did:vtn:trustos:123drfw",
  "password": "passphrase",
  "args": [
    "a",
    "b",
    "200"
  ],
  "channel": "channel1",
  "function": "create"
}
```

POST - /service/invoke/jsonArgs

Invoke a function from a distributed service using TrustID.

- serviceID : <string> Identity that is going to identify the service.
- function : <string> Function that is going to be called.
- args : <json> Args of the function that is going to be called in a json format.

Sample structure (Click to expand)

```
{
  "serviceID": "did:vtn:trustos:123drfw",
  "args": {
    "field": "value"
  },
  "function": "create"
}
```

POST - /service/query

Invoke a function from a distributed service using TrustID.

- `serviceID` : <string> Identity that is going to identify the service.
- `function` : <string> Function that is going to be called.
- `args` : <string> Args of the function that is going to be called.

Sample structure (Click to expand)

```
{
  "serviceID": "did:vtn:trustos:123drfw",
  "args": [ "a"],
  "function": "get"
}
```

POST - /service/query/jsonArgs

Invoke a function from a distributed service using TrustID.

- `serviceID` : <string> Identity that is going to identify the service.
- `function` : <string> Function that is going to be called.
- `args` : <json> Args of the function that is going to be called in a json format.

Sample structure (Click to expand)

```
{
  "serviceID": "did:vtn:trustos:123drfw",
  "args": {
    "field": "value"
  },
  "function": "get"
}
```

8.1.6 Signed Transactions

POST - /signed/id/import

Import an identity without custody in TrustID. Due to this, each transaction must be signed with the private key in custody by the user.

- `publicKey`: <string> The publicKey to verify the signature of the transactions
- `payload`: <string> A JWS signed with the private Key.

In order to create the JWS for the payload body, the JSON structure that is needed to be signed is the following:

```
{
   "function":"createSelfIdentity",
   "params":{
      "did":"$id",
      "publicKey":"$pubkey"
   }
}
```

Sample structure (Click to expand)

```
{
       "publicKey":  "-----BEGIN PUBLIC KEY-----
↪\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAokZtVnCJ5KnQRmYqV6j3\r\n1Oak7K0X/
↪pBDe/QS+LG1UEwcvi2rzMutvJUouMnLgroTMjErYFq0gVJAq7io2Yo/\r\n1/
↪I0suueWSfU30kNYoVikHgMKrHtWZA1iCR9uzYffeC/
↪KYBi0f+bLB5789+zJvr3\r\n8ScaPu+S0LIoYN7j6b7CqWfTBIfqM8NNxyhPiWsGBykQOB1Gl3VEiivm9dp5vFuS\r\n0OM/
↪pe/
↪+ToA4vHDIpr1NyxV0zM4TzLHbvWV70HMztsXmYp1yW8EQcLd412FOmtkU\r\nkEqqsoNK1IYX6F1ooRUjSGeybxPNKobNwkAQqa
↪----END PUBLIC KEY-----\r\n",
       "payload":
↪"eyJjdHkiOiJqd2sranNvbiIsImFsZyI6IlJTMjU2Iiwia2lkIjoidlB2MnhmLUUxcTEyVVhjUmtJMzZBcDU2blpzUktFSWpKYm
↪eyJmdW5jdGlvbiI6ImNyZWF0ZVNlbGZJZGVudGl0eSIsInBhcmFtcyI6eyJkaWQiOiJkaWQ6dnRuOnRydXN0aWQ6YjgwNmYzYm
↪"
}
```

POST - `/signed/service/invoke`

Write operation to a service signing the transaction with the private key guarded by the user.

- `publicKey`: `<string>` The did to recover the public key
- `payload`: `<string>` A JWS signed with the private key. The payload is the signature of the JSON explained below.

```
{
        "function": "invoke",
        params: {
            did: "coren-trackscc",  // did of the service
            args: ["createAsset", JSON.stringify(**args**)], // function to call the␣
↪chaincode and the respective params
            channel: "channel1" // channel where is deployed the chaincode
        }
    }
```

Sample structure (Click to expand)

```
{
  {
       "id":
↪"did:vtn:trustid:b806f3bef9967f903db9467eb809eb138c48d3d1717eda2d5a6bb178962a881f21606828938366
↪",
       "payload":
↪"eyJjdHkiOiJqd2sranNvbiIsImFsZyI6IlJTMjU2Iiwia2lkIjoidlB2MnhmLUUxcTEyVVhjUmtJMzZBcDU2blpzUktFSWpKYm
↪eyJmdW5jdGlvbiI6ImNyZWF0ZVNlbGZJZGVudGl0eSIsInBhcmFtcyI6eyJkaWQiOiJkaWQ6dnRuOnRydXN0aWQ6YjgwNmYzYm
↪"
```

(continues on next page)

(continued from previous page)

```
}
```

POST - `/signed/service/query`

Read operation to a service signing the transaction with the private key guarded by the user.

- `publicKey`: `<string>` The did to recover the public key
- `payload`: `<string>` A JWS signed with the private key. The payload is the params explained below.

```
{
        "function": "invoke",
        params: {
            did: "coren-trackscc",  // did of the service
            args: ["getAsset", JSON.stringify(**args**)], // function to call the␣
↪chaincode and the respective params
            channel: "channel1" // channel where is deployed the chaincode
        }
    }
```

Sample structure (Click to expand)

```
{
  {
       "id":
↪"did:vtn:trustid:b806f3bef9967f903db9467eb809eb138c48d3d1717eda2d5a6bb178962a881f21606828938366
↪",
       "payload":
↪"eyJjdHkiOiJqd2sranNvbiIsImFsZyI6IlJTMjU2Iiwia2lkIjoidlB2MnhmLUUxcTEyVVhjUmtJMzZBcDU2blpzUktFSWpKYn
↪eyJmdW5jdGlvbiI6ImNyZWF0ZVNlbGZJZGVudGl0eSIsInBhcmFtcyI6eyJkaWQiOiJkaWQ6dnRuOnRydXN0aWQ6YjgwNmYzYm
↪"
}
```

(*) Internal mail service for identity recovery communications

The password recovery process functions require a series of communications via email both to send the one-time code to the user and to send the different secrets to the guardians involved in the process. There are two possibilities for communication.

- TrustOS mail service: TrustOS has a mail service that allows internal communication. In this case, functions generate and send emails automatically to the user and the guardians, eliminating the need to do so externally. The response is shown bellow:

```
{
  "output": "Sent email to user"
}
```

- External communication: If on the contrary, the use case requires external communication, TrustOS just generates the codes and return them in the function response. The response for this case is shown below:

```
{
  "output": {
    "guardians": [
{
        "email": "bob@mail.com",
        "secret": "d10ff4ef162ee9fb4cd22ec8a78695520a3ed1ba809hff0495b8abf62d997224"
},
{
        "email": "alice@mail.com",
        "secret": "d10ff4ef162ee9fb4cd22sdjk78695520a3ed1ba809hff0495b8abf62sdm2333"
},
    ]
  }
}
```

To decide the type of communication there is a flag named `internalMailService` sent as a query param in the request. If the flag is set to `true` emails will be sent internally through the TrustOS mail service. On the contrary if it is set to `false` the communication will be managed externally.

The functions that require this flag are:

- POST - `/id/recover/create?internalMailService=true`
- POST - `id/recover/init?internalMailService=true`

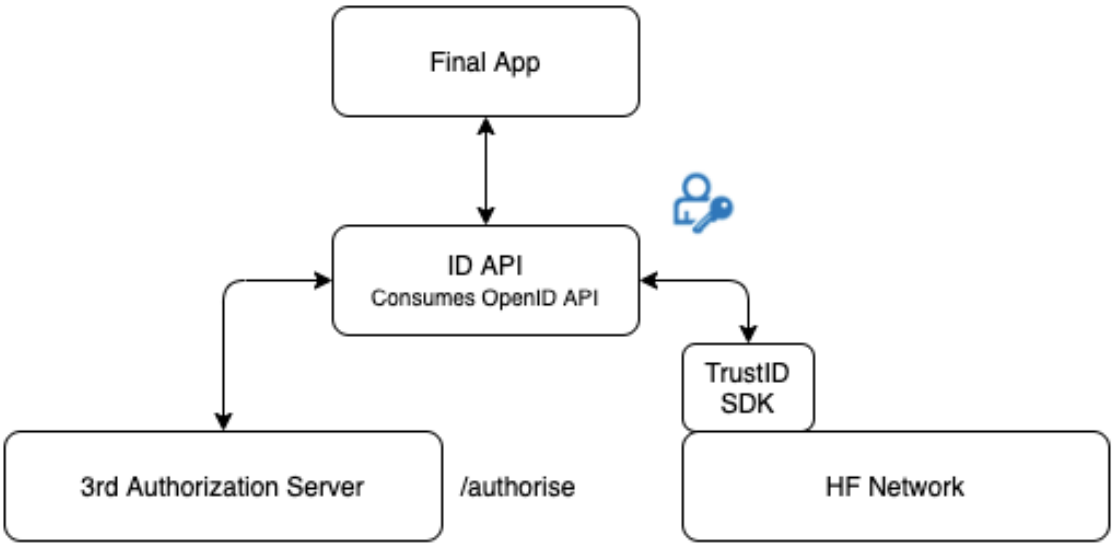
8.2 OpenID Connect integration

OAuth 2.0 is the industry-standard protocol for authorization. It is a framework for delegating access authorisation to APIs that allows the reuse of authentication mechanisms so that apps do not have to manage identities.

OpenId Connect is an identity layer built on top of the OAuth 2.0 protocol and extends it with an authentication layer. Authorization Servers allow to verify the identity of end-users and to obtain certain information associated with their profiles.

TrustID has integration with OpenID Connect enabling the onboarding and use of the TrustOS modules to identities generated in different identity providers such as Google or Microsoft.

8.2.1 OpenID integration



The OAuth 2.0 flow used by TrustID is Authorization code grant type:

- The application requests the user's express consent to access the data by opening a browser where the user specifies the scopes it wants to access.
- The user sends the access approval to the server through the browser.
- The server gives the user an access code that is redirected to the application.
- The application requests access to the authentication server by sending it the access code obtained in the previous step.
- Finally, the server gives the application the identity token.

TrustID manages the identity token to provide access to TrustOS to the end-users. The identity token is represented by a JWT signed by the Identity Provider (using RS256 algorithm), it is a claim which contains user's information as payload. A payload of an identity token generated by Google is shown below:

```
{
  "iss": "https://accounts.google.com",
  "azp": "316883325320-2il4roboraui8497s0so0vp82d1vo19h.apps.googleusercontent.com",
  "aud": "316883325320-2il4roboraui8497s0so0vp82d1vo19h.apps.googleusercontent.com",
  "sub": "112689420544152116604",
  "email": "bob_hamilton@gmail.com",
  "email_verified": true,
  "at_hash": "p4CHDlrU9yfN3WcGtzp9Uw",
  "name": "Bob",
  "picture": "https://lh4.googleusercontent.com/-5lQ3comXlRU/AAAAAAAAAAI/AAAAAAAAAAA/
↪AMZuuckL7QQpCJckUhD1AaU4lKkWuNq-uw/s96-c/photo.jpg",
  "given_name": "Bob",
  "family_name": "Hamilton",
  "locale": "es",
  "iat": 1613581956,
```

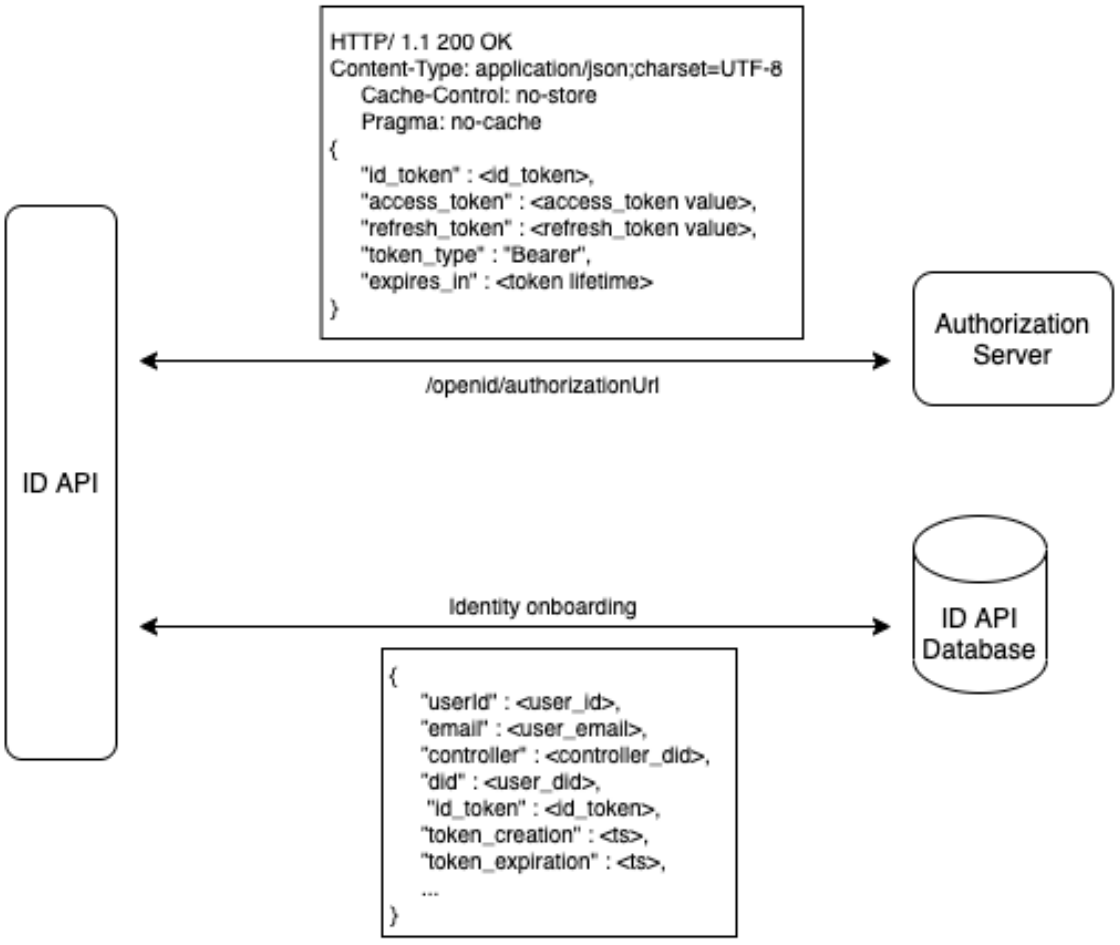
(continues on next page)

(continued from previous page)

```
  "exp": 1613585556
}
```

The first time an identity token arrives, an onboarding flow is triggered. This means that a TrustOS identity is generated for the issued token as it is necessary to generate keys for the user to be able to sign transactions within the platform.

8.2.2 General OpenID identity onboarding flow



When a request arrives authenticated by an Identity Token, TrustID checks the following:

- The issuer. Should be a recognised Identity Provider.
- The issuer signature. Verified with the public key of the Identity Provider.
- The expiration data. To ensure the token is still valid.
- The presence os a TrustOS identity that represents the user.

If all conditions are satisfied the request is validated, the end-user has access to the requested module and is able to sign transactions.

8.2.3 OpenID flow details and requirements

When a OpenID flow is initialized, an application callback endpoint should be passed as query parameter. The goal of the endpoint is to receive a code from the authorization server to authenticate the application and request the identity token. Thus, it is mandatory for the endpoint to be in a public domain and you should contact soporte-bteam@telefonica.com as we need to provision it on the OpenID clients managed by TrustOS. An example is shown below:

```
authCallback = "https://yourapp.com/authCb"
```

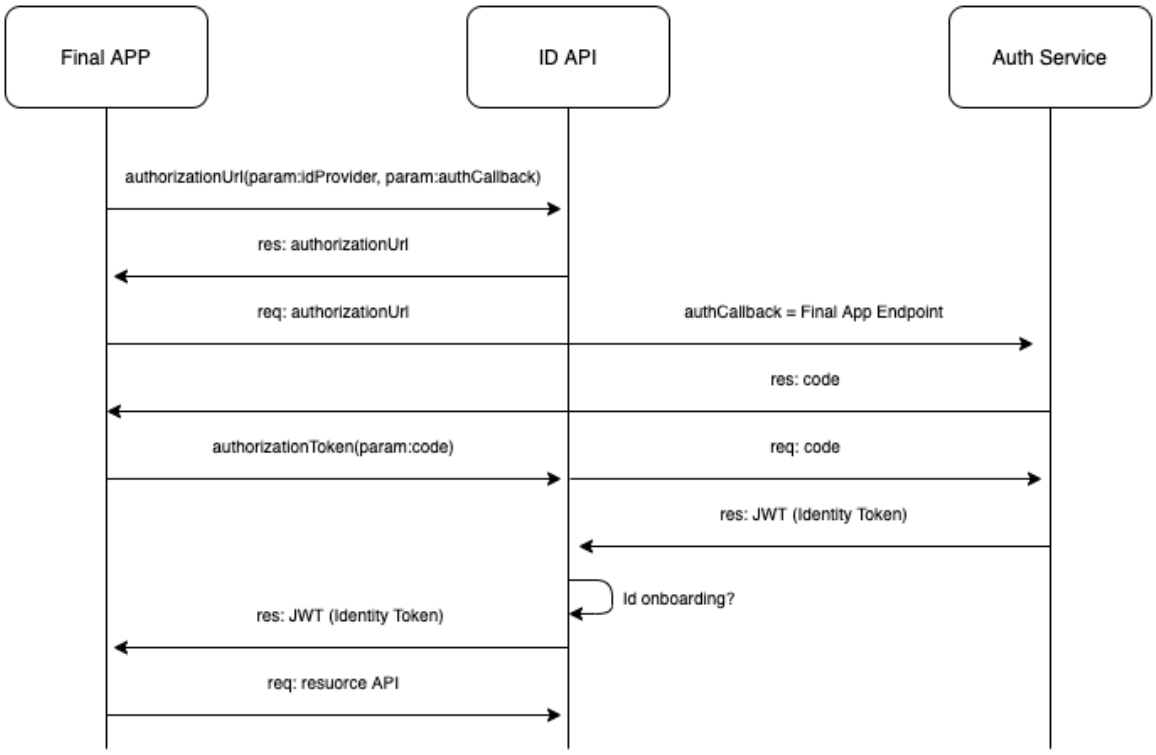
When the authorization URL is requested and the callback is received from the Authorization Server, an authorization code is served to the application. At this point the application should call the TrustID authorization token function (/authorizationToken) specifying the authorization code. This function allows the application verification on the authorization server checking its authorization code (passed as query parameter). An example of a request is shown below:

```
/openid/authorizationToken?code=M.R3_BAY.36b026d5-b621-4252-124e-e43a00aeb68b
```

Once the Authorization Server validates the code, it sends the JWT which contains the Identity Token to TrustID and, after the onboarding process, sends it to the final application. From then on, the application can send requests to the TrustOS modules by including the JWT in the authentication header in the following way:

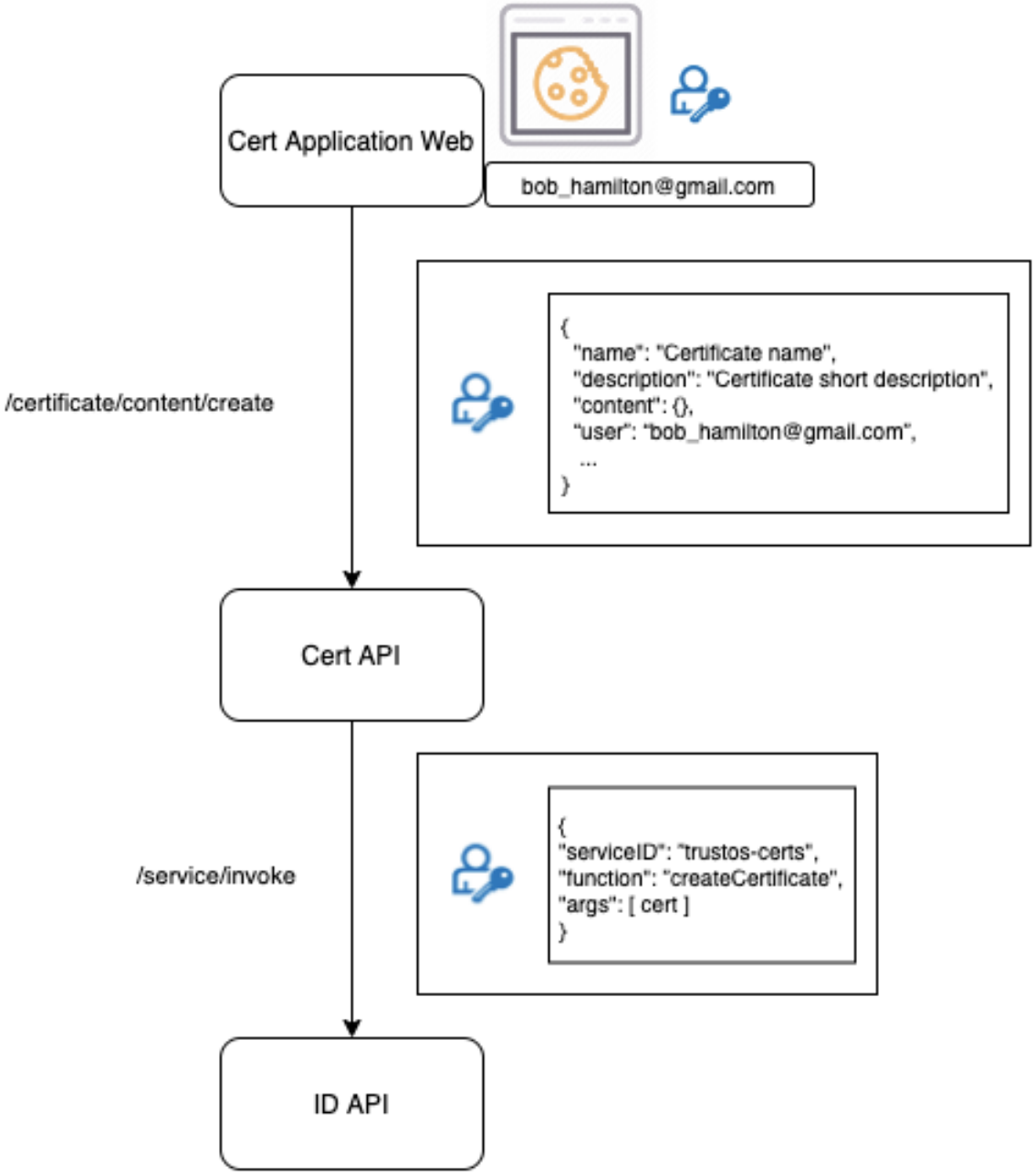
```
bearer JWT
```

The sequence diagram of the process is shown below:



8.2.4 Resource request flow

The final application should store the user's JWT. If for example it is a web application, it could store it in a session cookie:



8.3 Testing the application

In postman folder there are the collection and environment to interact and test with the API methods. It is only needed to import them into postman application and know to use the coren-certapi module

Also you can download the files in the links below:

- Postman collection

- Postman environment

CHAPTER 9

TrustID

TrustID is a standalone identity module for TrustOS. We followed a decentralized identity approach for its design, where users (and services) are identified through a DID.

TrustID is an opensource project and it was released as an opensource project under the umbrella of the Hyperledger Labs.

These DIDs follow the W3C standard, and they serve as a unique ID to identify users. DIDs aggregate all the pieces of public information required to authenticate a user (i.e., their public key or X.509 certificate).

This is the structure of a DID Identity:

- `id` : `<string>` Unique identifier of the identity using the W3C standard.
- `pubkey` : `<String>` Public key of the identity.
- `privkey` : `<string>` Private key of the identity.
- `type` : `<string>` Algorithm used in order to generate the keys.
- `controller` : `<String>` Verifier of the identity. In future, an identity should have more than one controller.

Example:

```
{
  "id":
↪"did:vtn:trustid:d830c50977703f421a0ed5a8707ecfe50559fe73dd2bf90c261b76a1379acbea",
  "type": "RSA",
  "controller": "",
  "access": 0,
  "privkey": "U2FsdGVkX1+t4HgMMEsla3Kw050Td00gS0Crs5Q/
↪D95LQ0Q8KAM3D8CIDQSbWf2fVlZZqvn2Rw5rr8nVkQRPVCAXQcLf3ufw53qLyB2pOqZh3hWWJ8T0jwc1esn6ONkDR4oyKXMWU1
↪rO4Q0yOTMDyJpQe7DsCusRYAU42Xf2ZTV+v2ZnPHMm/aN18Zmk3xRh7/
↪KH1Sz8LjJNpDgatYcETupYfIrdKES21mNAbSikn7UXzXB3gPrHIP9JucQqckgStpwgwf1qnI2P7xEXB6pZKBb9v5riCzxCygbWa
↪nthyUEJy10JlK0BCpwms+AnJ8JY5nn25z8JD4+p5i6zpq4IOuZaOf1qsja3fuGstH/
↪0tPlGHvu556gqN8vIVGMt9QwzUIE+kH+GxmyHEi/
↪TV8MRQmyGeCFtsZT9EoS5io1bP7mMC3+KvwvAXuqZXnvHBh5k7RreykIX6VpuJdUwibv5ram7DxV9wXLyVc8zbFWAWwoaFE9XVY
↪0uMlJ+4PW21GxdmV9uw8NdocpxWjusfHjsmPl5bno0nq9HcmOvKKdU5cwQKJS/
↪eaQc8GS104tDHjurxriS7KXHBLDGllzDl3QOskMw7RDb5jOLUbq2Zl0p0nznitKnLrHTc7/
↪xMfAY8d8OXD5yhzps/JQUEQDWy0OoheJ8CPmmsF22HmNce+/ylRfI1bv2/cHYN8oz+7wGO/
↪LCFW9cPv0s5Dlwvix6NsHfXHzLu0Vo9lbsn+LcPgA5+JMOBDnJ+ba2ps4QGRc/
↪i5gCF8NrFaa+tc2EXRLpAD33r1sxicCVm7CSK4jVWTg6MCkGbqGaaQTEx+x9RUp5og8Ap0IZ2OxjUVe9BaMsHwdyfcfjzG1J0sP
↪wfxw5sTC6rCoAt8n+pcdIiAXfuzyMtkthsdELXtrXJ84BeSc0YAob0/duPfHVikNDdQOcFNWWKH1f/
↪6Y5R0Qcf1QaDIkfdjX77smM2duO6jkV3g1xrGpwUyFiHM+YHYp0UsPOa/YzVEpWRJu/
↪tIivCmxiSZwTom1QNo7Y7PtkReIeW3RvA8vlhEGMYGrg6jzZOA4aXjYpYxM5O8zWTANRhQbs4DtBOSXGrw7YtKmTdC9B1waiKs
↪egKjPQi4rgqvjLYmcTkWYD27XaP2NM8L+pcBeeTN2f3MCCfhM4niuDF1l09GG9G703Z3GHmWJP8dHWCs+CVJgn06/
```

(continues on next page)

(continued from previous page)

```
  "pubkey": "-----BEGIN PUBLIC KEY-----
↪\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxfun1NH3jOlNRuX0z7cr\r\n5eFWCt/
↪xUSKwikYg7NRoaNn/NNAQln/BDSamVG3RwlmM/dvUt783Hp7YqeHscXmp\r\ngKwWiF/
↪RycT3Sx9l3qC0MfLlbmsQJ0ZqO+Iuy+vjAxq4RGTS1dzuCjipPy4yBCxB\r\nrWT/
↪q3lFboWaEAoFA+DFm0Hwt47P+lApAfcMc0JduW31gaULUNpqTZyQJBwBKfk1\r\nBZPrurutwnEuOxOOOFNXXL/
↪IUSqneC+71drGO4xA1Sq1bKtsJE9WYW3U8w4C8dHs\r\nU1rq9z/
↪MOAsUlz+DGzyIMI16ypjTC8T4BkWg9vGPgf/C8ropgsbx+9fMKBSisHG5\r\ntQIDAQAB\r\n-----END␣
↪PUBLIC KEY-----\r\n"
}
```

This is the structure that is stored in the DLT platform that belongs to an user or device identity:

- `Did` : `<string>` Unique identifier of the identity using the W3C standard.
- `PubKey` : `<String>` Public key of the identity.
- `Controller` : `<String>` Verifier of the identity. In future, an identity should have more than one controller.
- `Access` : `<integer>` Policy of the identity in order of identity operations, 0 root, 1 admin, 2 user.

Example

```
{
  "Did  ":
↪"did:vtn:trustid:d830c50977703f421a0ed5a8707ecfe50559fe73dd2bf90c261b76a1379acbea",
  "Controller": "",
  "Access": 0,
  "PubKey": "-----BEGIN PUBLIC KEY-----
↪\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxfun1NH3jOlNRuX0z7cr\r\n5eFWCt/
↪xUSKwikYg7NRoaNn/NNAQln/BDSamVG3RwlmM/dvUt783Hp7YqeHscXmp\r\ngKwWiF/
↪RycT3Sx9l3qC0MfLlbmsQJ0ZqO+Iuy+vjAxq4RGTS1dzuCjipPy4yBCxB\r\nrWT/
↪q3lFboWaEAoFA+DFm0Hwt47P+lApAfcMc0JduW31gaULUNpqTZyQJBwBKfk1\r\nBZPrurutwnEuOxOOOFNXXL/
↪IUSqneC+71drGO4xA1Sq1bKtsJE9WYW3U8w4C8dHs\r\nU1rq9z/
↪MOAsUlz+DGzyIMI16ypjTC8T4BkWg9vGPgf/C8ropgsbx+9fMKBSisHG5\r\ntQIDAQAB\r\n-----END␣
↪PUBLIC KEY-----\r\n"
}
```

This is the structure of a service identity:

- `serviceID` : `<string>` Unique identifier for the service.
- `Name` : `<string>` Chaincode or smart contract identifier at the DLT level.
- `Controller` : `<string>` Owner/admin of the service.
- `Access` : `<string>` Access policy to the service.
- `Channel` : `<string>` Network ID of the service.

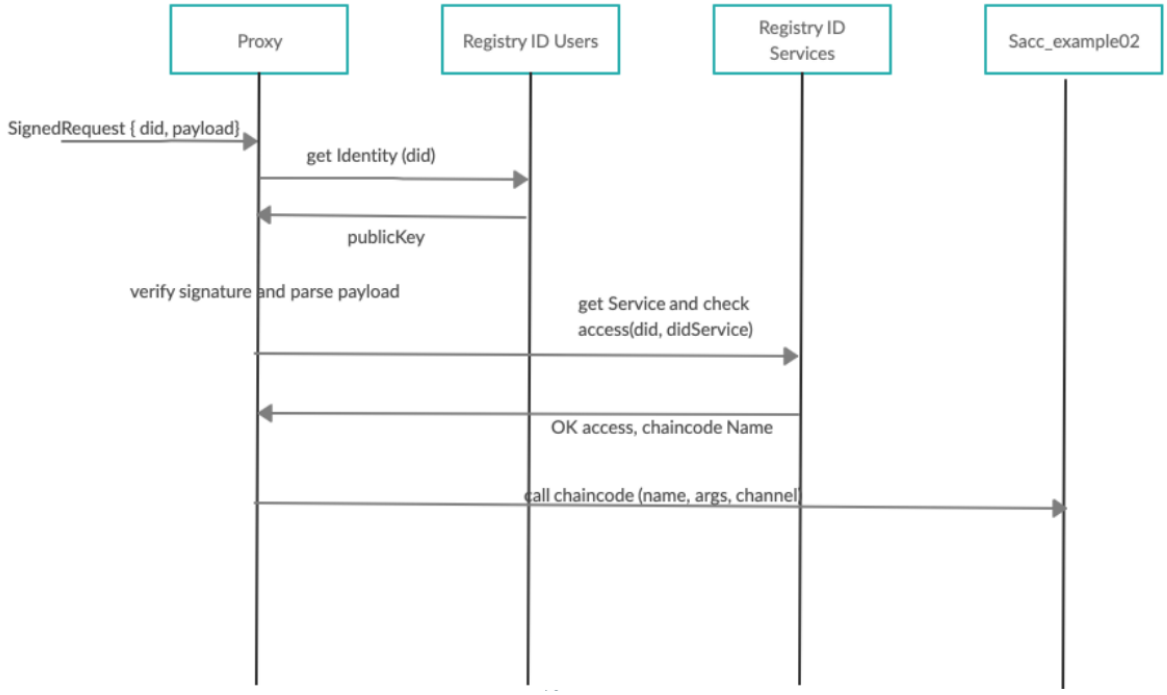
Example:

```
{
  "serviceID":  "track",
  "name": "trackscc",
  "access": {
    "policy": "PUBLIC"
  },
  "channel": "telefonicachannel"
}
```

In order to uniquely identify chaincodes and services deployed in TrustOS, we decided to also give them DIDs so that they could be seamlessly discovered and accessed even if they "live" in independent channels not shared by all the organizations of the network.

All the authentication and management of identities in the system is performed on-chain through an "Identity Chaincode".

If user A wants to start interacting with the network, he requests the generation of a new DID. The related keys to this DID could be an existing X.509 issued by a valid organization, or even an Ethereum-related public key (internally we use all the JWS, JWE, JWK, secp256k1, etc. RFCs to make our Fabric infrastructure compatible with identities of any nature for the sake of interoperability). This DID generation request has to be validated by a valid organization of the network. Once verified, every transaction signed by user A and directed through a Proxy chaincode is authenticated successfully and delegated to the corresponding chaincode.



The TrustID project is conformed by the aforementioned chaincode and a client SDK to ease the integration and interaction with TrustID-enabled networks.

TrustID is designed to ease the management of identities for the case of TrustOS. Users shouldn't need to hold a different set of credentials for each network or decentralized application they interact with. The same credentials used to access your owned Bitcoins and manage your tokens in Ethereum should let you update the state of a Fabric asset or launch a secondary market in TrustOS.

9.1 TrustID SDK

This SDK exposes all the functionalities required to interact with TrustID-based DLT networks.

9.1.1 Install

- To install this library you need access to the private repo:

```
npm install trustos-trustid-sdk
```

9.1.2 Structure

The library has the following modules:

Wallet

- `wallet.ts`: Core module of the library. It wraps all the state and logic for identity management and interaction with TrustID networks. To start using the SDK a new wallet needs to be initialized. A wallet exposes the following methods:
  - `setKeystore(keystore: Keystore): void`: Sets a type of keystore, supported: in memory, filesystem, mongodb.
  - `generateDID(type: string, controller: string, passphrase): DID`: Generates an identity.
  - `storeDID(did: DID): Promise<boolean>`: Stores the did in the keystore.
  - `updateDID(did: DID): Promise<boolean>`: Updates info from DID.
  - `listDID(): string[]`: Returns dids stored in keystore.
  - `recoverKeySSS(id: string, secrets: Buffer[], newPassword: string): Promise<void>`: Recovers the key.
  - `updatePassword(id: string, oldPassphrase:,passphrase: string=""): Promise<void>`: Updates the password to unlock the did.
  - `updateTempKeyDID(id: string, passphrase:,tempPassphrase: string=""): Promise<void>`: Unlocks the account with a temporal key.
  - `addNetwork(id: string, network: TrustID): void`: Adds a new network to interact to.

Class DID

- `class DID`: Has the following structure:
  - `id: string`: Id string that identifies the DID.
  - `pubkey: string`: PublicKey of the DID.
  - `type: string`: Key type (RSA / EC / OKP).
  - `controller: string`: Verifier of the identity.
  - `access: number`: Access level.
  - `private privkey: string`: Private Key of the DID.
  - `private recoveryKey: string`: Private Key to recover the DID.

And exposes the following functions:

- `unlockAccount(passphrase: string): void`: Unlocks private key in order to use the DID.
- `unlockAccountTemp(passphrase: string): void`: Unlocks private key with a temporal key in order to use the DID.

- `lockAccount(): any`: Locks the private key for a DID.
- `sign(payload: object, passphrase: string): string`: Sign a payload with a specific DID.
- `verify(signature: string, id: string): any`: Verifies a signature from a DID.
- `updatePassword(oldPassphrase:, passphrase:): Promise < DID >`: Updates the password.
- `generateRecoveryKey(password:string, shares: number, threshold: number): Promise <Buffer[]>`: Generates the recovery key using shamirs secrets sharing.
- `generateRecoveryKeyTemp(passwordTemp:string, shares: number, threshold: number): Promise <Buffer[]>`: Generates the recovery key using shamirs secrets sharing unlocking the account with the temporal Key.
- `recoverKey(secrets: Buffer[], newPassword: string): Promise < DID >`: Recovers the key using the secrets generated with Shamirs secrets sharing algorithm.
- `exportDID(withPrivate: boolean)` : Exports a Did stored in the keystore.
- `importDID(obj: any)`: Imports a DID and stores it in the keystore.
- `sign(payload: object): Promise < string >`: Generates a JWS from a payload using an id from the wallet.
- `verify(signature: string, did: DID): Promise < any >`: Verifies a JWS from a payload using a did.

TrustID operations

- `TrustID.ts`: Interface that enables the inteoperation between the drivers and the different functionalities of TrustID. The only component implemented currently is the `trustIDhf.ts` enabling the interaction with Hyperledger Fabric TrustID. networks.
  - `configureDriver(endpoint: string): void`: Sets the network endpoint to interact with the TrustID network.
  - `disconnectDriver(endpoint: string): void`: Disconects the network endpoint to interact with the TrustID network.
  - `createIdentity(did: DID): Promise<object>`: Create an identity in TrustID. It generates a new DID in the wallet and register it in the network.
  - `importIdentity(did: DID, controller?: DID)`: Imports an existing identity to the chaincode.
  - `verifyIdentity(adminDID: DID, id:string): Promise<object>`: Verifies an identity as an admin.
  - `getIdentity(did: DID, id: string): Promise<object>`: Gets a registered identity from TrustID.
  - `revokeIdentity(adminDID: DID, id: string): Promise<object>`: Revokes a registered identity. Only supported by the owner or controller of the DID.
  - `createService(did: DID, serviceDID: string, name: string, isPublic: boolean): Promise<object>`: Creates a new service in the TrustID network.
  - `updateService(did: DID, serviceDID: string, access: Access, isPublic: boolean): Promise<object>`: Updates the information from a service.

  - `updateServiceAccess(did: DID, serviceDID: string, access: AccessPolicy): Promise<object>`: Updates the access from a service.
  - `getService(did: DID, serviceDID: string): Promise<object>`: Gets information from a registered service.
  - `invoke (did: DID, serviceDID: string, args: string[], channel: string): Promise<object>`: Invokes a function of a registered service in the TrustID network.
  - `query(did: DID, serviceDID: string, args: string[], channel: string): Promise<object>`: Queries a function of a registered service in the TrustID network.

- `PolicyType (policy: PolicyType, threshold:?Number, registry:?object)`: It defined the policyType to be used for a service. There are currently three types of policyTypes supported (more could be easily added according to your needs) * PublicPolicy: Grants public access by any user to your service. * SameControllerPolicy: Only verified identities whose controller is the same controller who created the service has access to the service (this policy comes pretty handy when you want to define "corporate-wide" services). * FineGrainedPolicy: Grants fine-grained access to users to your service. In this policy you explicitly define the access of users to the service. There are two ways of using this policyType, you can define a threshold so every user with an access level equal or higher than the threshold is granted access to the service; or you could use fine-grained access levels defined in the registry, where you would add the following tuple: `{<did>, <access_role>}`. Thus, only users in the registry with an access level over the threshold will be granted access to the service with `access_role` permissions.

Driver operations

- `driver.ts`: Interface that enables the implementation of connection drivers with different TrustID networks. The only driver implemented currently is the `hfdriver.ts` enabling the interaction with Hyperledger Fabric TrustID networks.
  - `connect(config: object): Promise<object>;`: Connects with a DLT networks.
  - `disconnect(config: object): void`: Disconnects of a DLT network.
  - `checkConnection(channelName?:string): Promise<object>`: Check the connection with a DLT.
  - `callContractTransaction(id: string, fcn: string, args: any, channel?: string): Promise<object>`: Write operation in the DLT.
  - `getContractTransaction(id: string, fcn: string, args: any, channel?: string): Promise<object>`: Read operation in the DLT.

Keystore

- `keystore.ts`: Interface that enables the implementation of keystore storages.bThere are currently two implementations of keystore supported: `FileKeystore.ts` (to store DIDs in file keystore)and `MongoKeystore.ts` (to store DIDs in MongoDB).
  - `abstract getDID(id: string): DID`: It gets specific DID from keystore.
  - `abstract storeDID(did: DID): boolean`: It stores DID in keystore.
  - `abstract updateDID(did: DID): boolean`: It updates DID in keystore.
  - `storeInMemory(did: DID): boolean`: Stores DID inMemory for easy and performant use.
  - `listDID(): string[]`: List DIDs in memory.

- `setDefault(did: DID): boolean`: Set DID as default identity for the keystore wallet.

If you want additional information of TrustID, its SDK and its functionality check the following repo.

9.1.3 Example of use

```
// Use library
var id = require('trustos-trustid-sdk')
import { Keystore } from './keystore/keystore';

// Initialize wallet
wal = id.Wallet.Instance;

// Initialize new FileKeystore with storage at ./keystore
const ks = new sdk.FileKeystore("file", "./keystore");
wal.setKeystore(ks)
// ccp is the configu file of a Hyperledger Fabric network
const ccp = JSON.parse(fs.readFileSync("../ccp-dev-dsn.json", 'utf8'));
const config = { // additional config to connect with hyperledger fabric
    stateStore: '/tmp/statestore',
    caURL: 'https://ca:7054',
    caName: 'ca',
    caAdmin: 'admin',
    caPassword: 'password',
    tlsOptions: {
        trustedRoots: "Certificate",
        verify: false
    },
    mspId: 'telefonicaMSP',
    walletID: 'admin',
    asLocalhost: false,
    ccp: ccp,
    chaincodeName: "identitycc", //name of the trustid contract deployed on fabric
    fcn: "proxy",
    channel: "channel1"
}


// Create HF driver to interact with a hyperledger fabric network
var trustID = new sdk.TrustIdHf(config);
// Add and configure the network driver in our wallet.
wal.addNetwork("hf", trustID);
await wal.networks["hf"].configureDriver()

// IDENTITY OPERATION
 // Generate key pair locally.
const did = await wal.generateDID("RSA", "test", "test")
await did.unlockAccount("test")
    // Register in the platform.
await wal.networks.hf.createSelfIdentity(did)
 wal.setDefault(did)
// Get the registered identity.
let res = await wal.networks.hf.getIdentity(did, did.id)
console.log("[*] Get registered identity\n", res)

//SERVICES OPERATIONS
await wal.networks.hf.createService(did, "coren-trackscc", "track", "PUBLIC",
↪"channel1")
```

(continues on next page)

(continued from previous page)

```
let res = await wal.networks.hf.getService(did, "coren-trackscc")
// Create an asset in the service: Track operation via trustid
const asset = {assetId: "test"+Date.now(), data:{"a":1, "b":2}, metadata: {"c": 4}}
const assetStr = JSON.stringify(asset)
// Invoke tract
res = await wal.networks.hf.invoke(did, "coren-trackscc",["createAsset", assetStr],
↪"channel1")
console.log("[*] Asset creation:\n", res)
    // Get the created asset.
res = await wal.networks.hf.invoke(did, "coren-trackscc",["getAsset", JSON.stringify(
↪{assetId: asset.assetId})], "channel1")
console.log("[*] Asset registered\n", res)
```

9.2 Roadmap

- TrustID compatible with Hyperledger Indy, Verifiable Credentials, etc.
- Anonymous authentication with anonymous transactions.
- Integration with other Identity Providers, related with the non-blockchain world.
- Support for other Keys algorithms, now only RSA is supported.
- Compatibility with Ethereum keys.

CHAPTER 10

Use cases

TrustOS enables companies to develop new decentralised services for a variety of industries and situations.

10.1 Proof of Everything

The Trust module allows you to record and store on the network any type of information (contracts, mail, messages, content, media, etc.) that you want to be verifiable to anyone at any time from registration. Any supply chain or inventory management solution can be easily implemented by integrating the Trust module into business processes.

10.2 Issue certifications

It is possible to create an asset always verifiable for the business processes: ticket sales, training and education, certificates of origin, ownership and authenticity, etc.

10.3 Secondary markets

The Token module can be used to create and manage tokens that represent any asset in the real world (from physical assets to usage rights or intangible assets such as a person's time) that can be transmitted or validated according to ERC20 and ERC721 standards. We can think of mobility vehicles in cities (scooters, motorcycles, cars, etc.), collaborative economy resources (apartments, cars, etc.), rationalization of the use of shared resources (meeting rooms, visiting slots, medical consultations, etc.) or any physical or digital asset to be modelled.

10.4 Digital ID

The decentralized ID approach allows you to add security to any process by validating people and device identities and verifying digital signatures without sharing a large amount of unnecessary data.

CHAPTER 11

Tutorials

In this section you will find snipets of code to integrate API calls to your page

11.1 Postman

Postman is a fantastic software that lets us run requests against our APIs.

One of the many great tools of Postman are collections, sets of grouped requests that can be exported/imported, in order to be run all together for testing purposes.

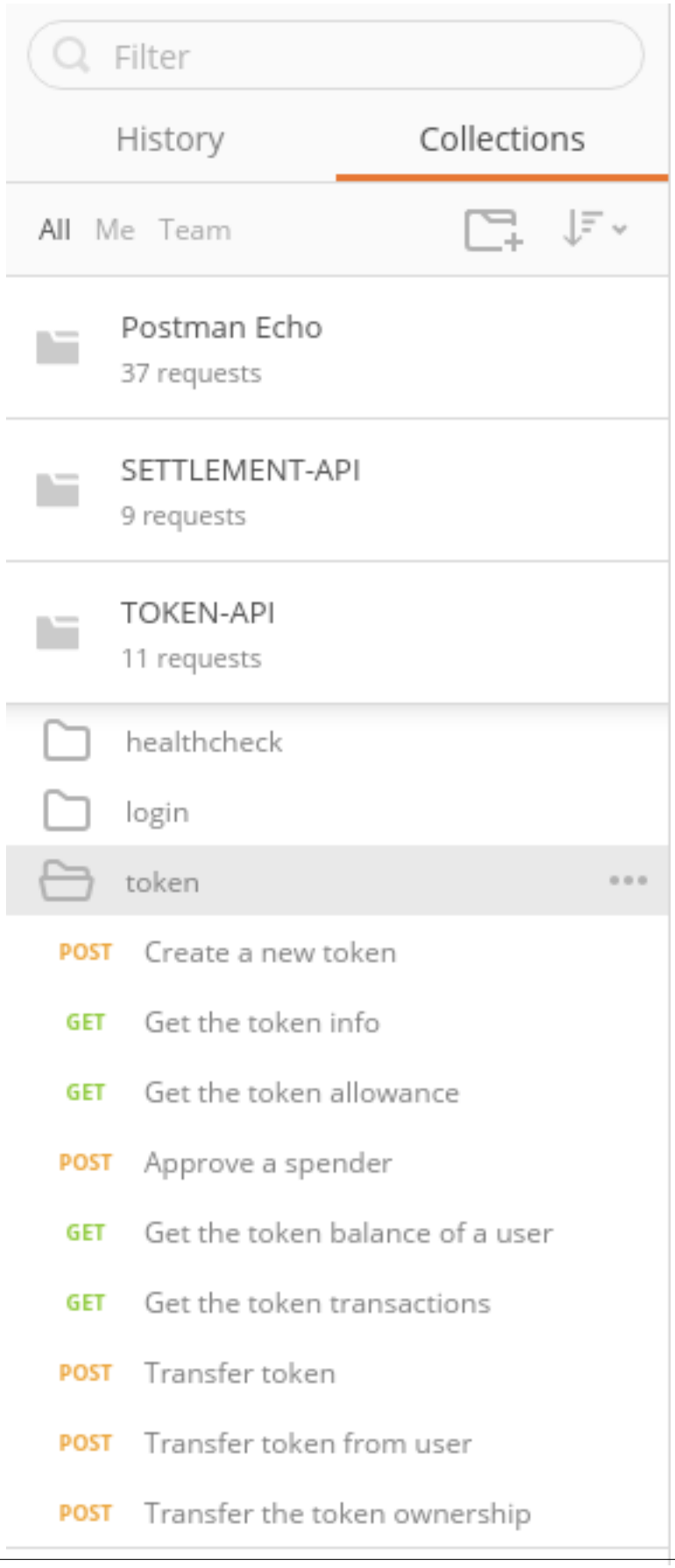
11.1.1 Postman collection

1. First, download the collections we have created on the folder **miscelanea/postman**

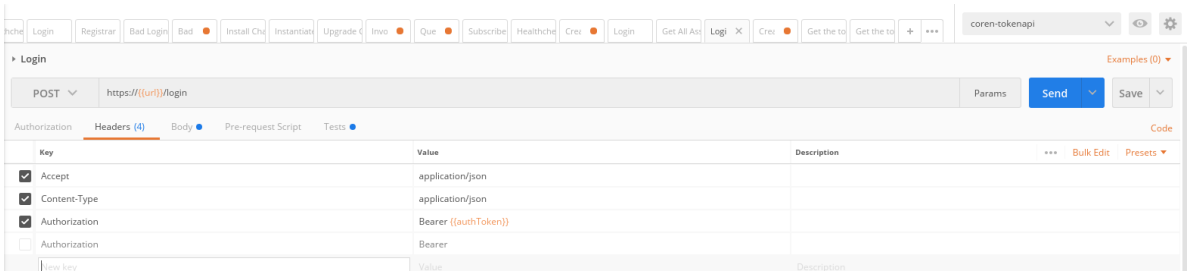
   You will see 2 diferent files: **collections** and **enviroment**. Download both, as we will explain later what are enviroments for.

1. On Postman, click on "Import" , and then select **Import file**.

   After doing so, on the left pannel you should see the collection you just imported, as shown in the image


Filter
History
Collections
All
Me
Team
Postman Echo
37 requests
SETTLEMENT-API
9 requests
TOKEN-API
11 requests
healthcheck
login
token
POST Create a new token
GET Get the token info
GET Get the token allowance
POST Approve a spender
GET Get the token balance of a user
GET Get the token transactions
POST Transfer token
POST Transfer token from user
POST Transfer the token ownership

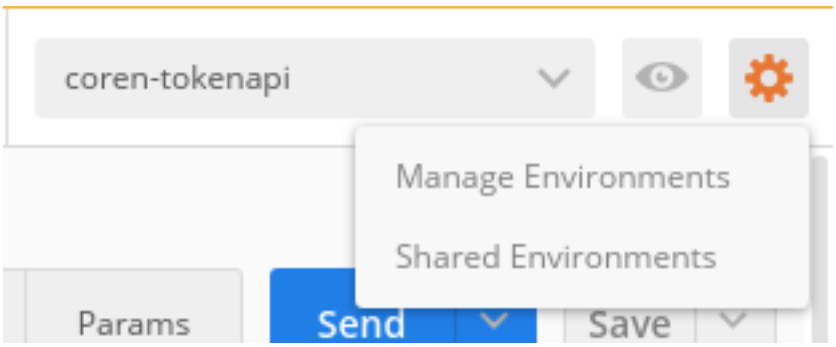
1. Now that we can see the collection, lets take alook at the URL for which the request is being made:



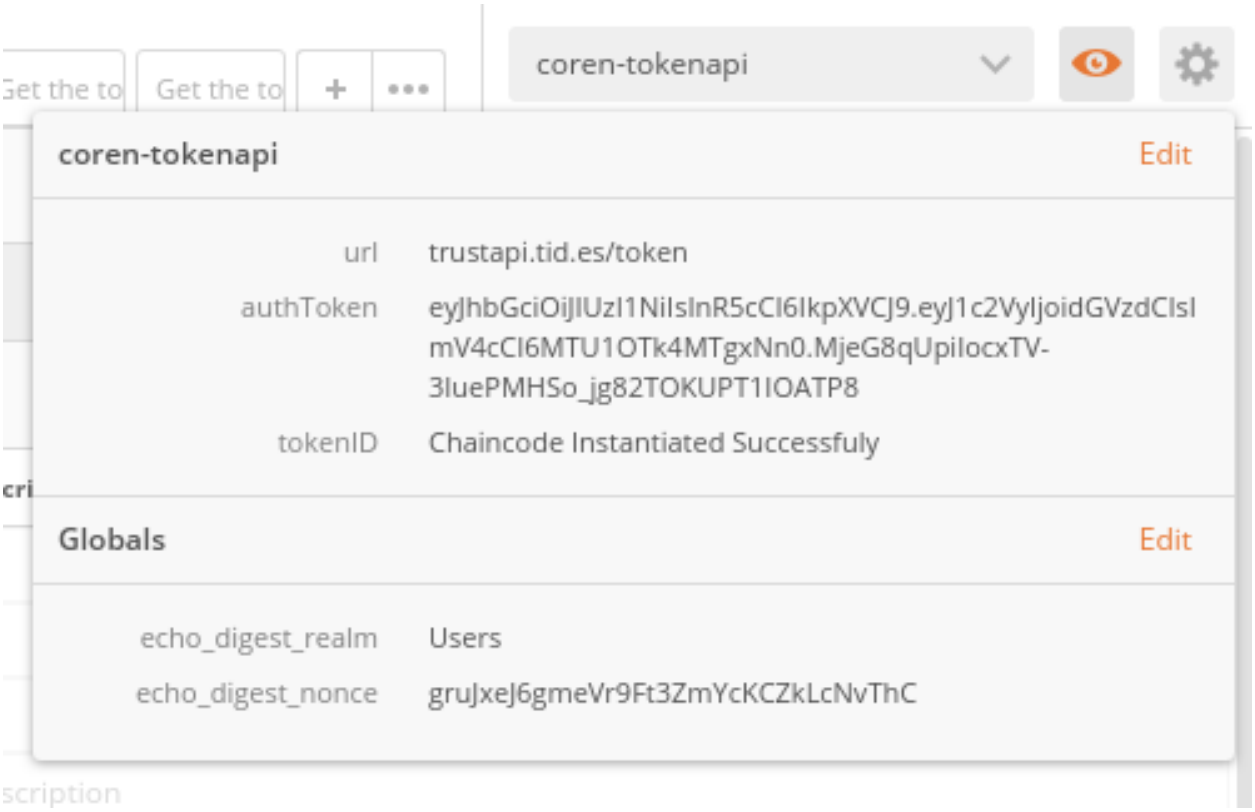
We can see that the URL contains: ” {{url}} “

This is a variable in POSTMAN. The nice things about using variables, is that we can change the endpoints of the APIs without having to modify the requests, but only the **enviroment** file, where the enviroment variables are specified.

To load a specific enviroment file, click the configuration button, manage enviroment, and select the enviroment file you want to load.



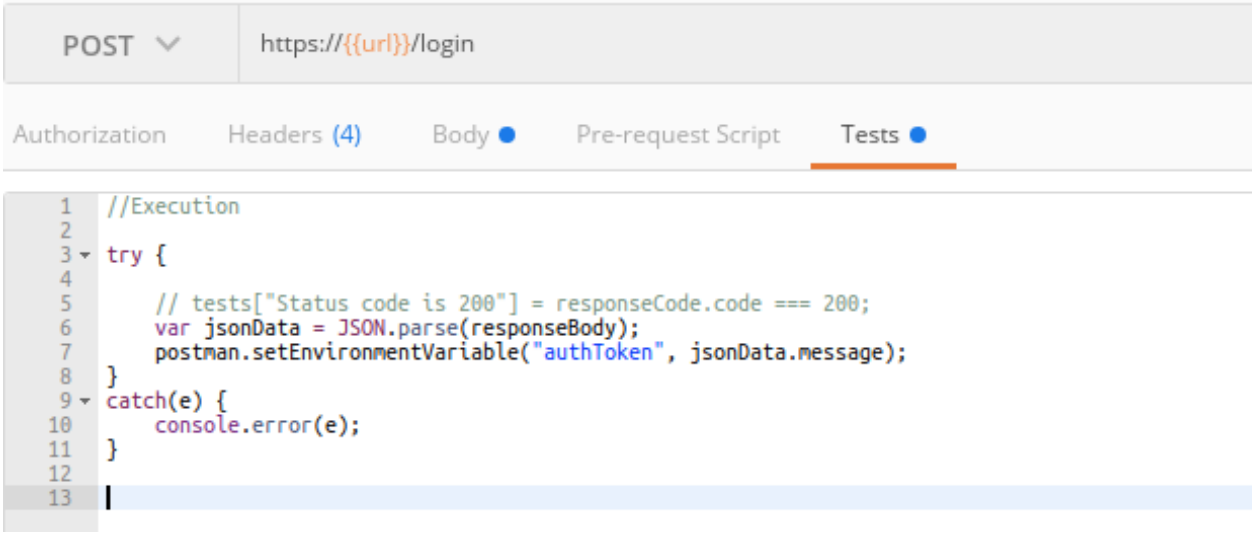
Great! Now, by clicking on the eye, you can watch the **actual values** of the enviroment values



Side note Why do we say "actual values"? Because values can change dinamically

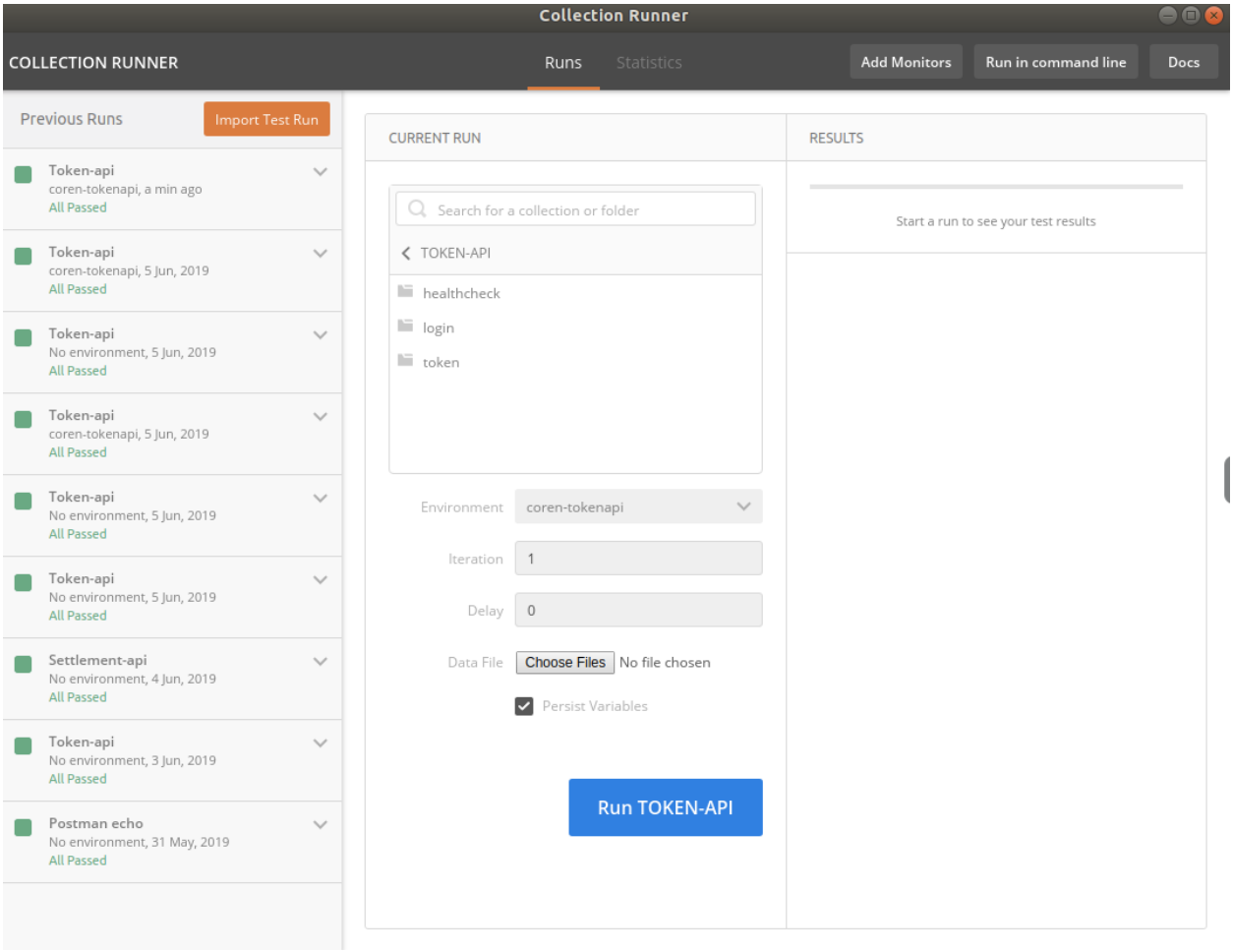
We can see that under the URL field there are some others others.

Concretely, **pre-reqest scripts** are scripts that execute right before the query is made, where **tests** are scripts that execute after the request is made.



Luckily you dont have to worry about this, since we have already taken care of it. When you execute the login request, the Auth token received is set as the value of the variable "AuthToken", which is later used in the **Header** tab (as can be seen in the second image of this tutorial)

1. Finally, now that we have everything set up, click on the "Runner" tab on the top left of the screen. Select the collection you want to run, the enviroment file you want to use for this collection, and run the tests!



11.2 Managing requests

11.2.1 NodeJS

Axios library

Axios is a promise based HTTP client for the browser. Using promises is great when dealing with code that requres chains of events.

```
const axios = require('axios');

axios.post('https://trustapi.tid.es/trust/asset', {
 headers: {
   Authorization: 'Bearer ' + token // Token is a variable that stores the JWT
},
  body: {
    'input':'whatever'
  })

  .then(response => {
    console.log(response.data.url);
```

(continues on next page)

(continued from previous page)

```
    console.log(response.data.explanation);
  })
  .catch(error => {
    console.log(error);
  });
```

HTTP

If you dont want to use third party libraries,you can use node http standard module. However, this option is a little more verbose than the preceding one

```
const https = require('https')

const data = JSON.stringify({
  input: 'whatever'
})

const options = {
  hostname: 'https://trustapi.tid.es/',

  path: '/trust/asset/create',
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': data.length,
    'Authorization': 'Bearer '+token  // Where token is the JWT
  }
}

const req = https.request(options, (res) => {
  console.log(`statusCode: ${res.statusCode}`)

  res.on('data', (d) => {
    process.stdout.write(d)
  })
})

req.on('error', (error) => {
  console.error(error)
})

req.write(data)
req.end()
```

11.2.2 Python

Requests

You dont really want to use any other module!

```
import requests

auth_token='kbkcmbkcmbkcbc9ic9vixc9vixc9v'
```

(continues on next page)

(continued from previous page)

```
hed = {'Authorization': 'Bearer ' + auth_token}
data = {'app' : 'aaaaa'}

url = 'https://trustapi.tid.es/trust/asset'
response = requests.post(url, json=data, headers=hed)
print(response)
print(response.json())
```

11.2.3 Shell

Lets not forget our old friend curl. This way can also be integrated in any other programming language calling something like exec.shell(command)

```
curl -X POST "https://trustapi.tid.es/trust/asset/create" -H "accept: application/json
↪" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJ1c2VyIjoidGVzdCIsImV4cCI6MTU2MTEyMjMwOX0.
↪Qu28A580dTOXPAX9bKsnEuHRk8NxFLGL0iPkK5RuOKg" -H "Content-Type: application/json" -d
↪"{ \"assetid\": \"1\", \"data\": {}, \"metadata\": {}, \"registerInEthereum\": \
↪"true/false\"}"
```

11.3 ReactJS

CHAPTER 12

Demos

In this section you will find snipets of code to make APIs calls to test and see the results for different use case examples. These demo applications allow to understand how easy and fast things are building a new blockchain-based solution or integrating your systems with TrustOS.

12.1 Vaccine tracker

Vaccine Tracker is an easy demo based on Track API that allows us to follow the whole transport process for a vaccine from origin to destination and to monitor constantly critical measures (e.g temperature, humidity)

Example JSON structures (Click to expand)

12.1.1 1 - Login into the platform

POST - `/login`

```
{
  "id":
↪"did:vtn:trustid:8297475fa68e64de5128f0b819fae647f128969bad64bf5ad43f1797e5733eb0",
  "password":"test"
}
```

12.1.2 2 - Create asset: Vaccine batch

POST - `/asset/create`

```
{
  "assetId":"Vaccine_batch_001",
  "data":{
      "lab":"BioNTech",
```

(continues on next page)

(continued from previous page)

```
        "date":"2020/12/12",
        "disp":"2020/08/08",
        "batch":"123111",
        "units":"20",
        "description":"ALLE DEMI CV333"
    },
    "metadata":{
        "location":"51.165691, 10.451526",
        "ref":"Marburg, Germany",
        "temperature":"15",
        "humidity":50
    }
}
```

12.1.3 3 - Add rules: Temperature and humidity measures

POST - `/asset/{assetId}/rules`

```
{
    "rules":{
        "value":[
            {
            "param":"temperature",
            "value":"15"
            }
        ],
        "range":[
            {
            "param":"humidity",
            "min":40,
            "max":60
            }
        ]
    }
}
```

12.1.4 4 - Update asset (first transport update - Lyon)

POST - `/asset/{assetId}/update`

```
{
    "metadata":{
        "location":"45.74846, 4.84671",
        "ref":"Lyon, France",
        "temperature":"15",
        "humidity":50,
        "transportCompany":"UPS",
        "transporter":"David"
    }
}
```

12.1.5 5 - Update asset (transport update with TEMPERATURE ALARM - Barcelona)

POST - `/asset/{assetId}/update`

```
{
  "metadata":{
      "location":"41.38879, 2.15899",
      "ref":"Barcelona, Spain",
      "temperature":"10",
      "humidity":50,
      "transportCompany":"SEUR",
      "transporter":"Pep"
  }
}
```

12.1.6 6 - Update asset (transport update with HUMIDITY ALARM - Zaragoza)

POST - `/asset/{assetId}/update`

```
{
  "metadata":{
      "location":"41.65606, -0.87734 ",
      "ref":"Zaragoza, Spain",
      "temperature":"15",
      "humidity":38,
      "transportCompany":"SEUR",
      "transporter":"Pep"
  }
}
```

12.1.7 7 - Update asset (last transport update - Madrid)

POST - `/asset/{assetId}/update`

```
{
  "metadata":{
      "location":"40.4165, -3.70256 ",
      "ref":"Madrid, Spain",
      "temperature":"15",
      "humidity":50,
      "transportCompany":"SEUR",
      "transporter":"Pep"
  }
}
```

12.1.8 8 - Demonstrator

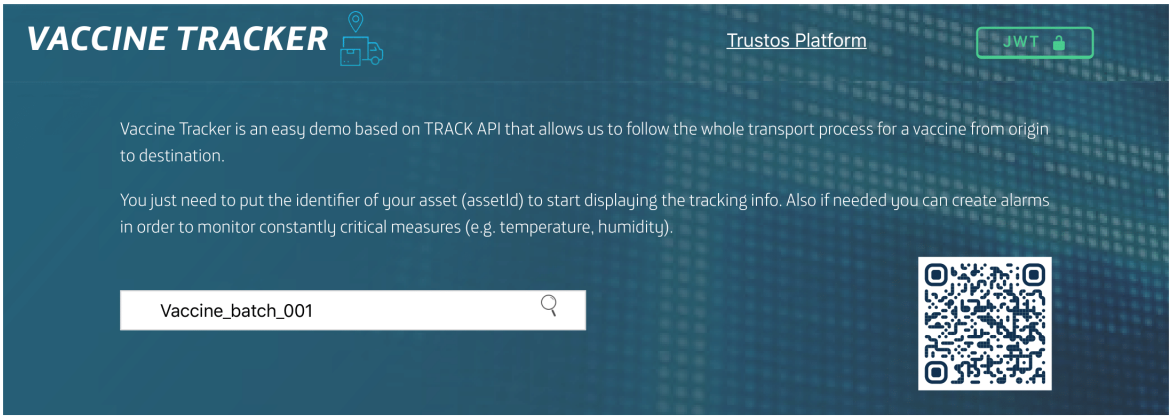
Along the transport process all the information is displayed in the Vaccine Tracker application. You just need to put the identifier of your asset (assetId) to start seeing the whole tracking information. Moreover, the application is connected to a WebSocket channel in order to listen and monitor all the critical measures that are out of the range or value stablished (e.g temperature and humidity). Every new alarm is displayed as a notification.

Endpoint: `/tracker/vaccine`

Input (assetId):

Front-end demo (Click to expand)

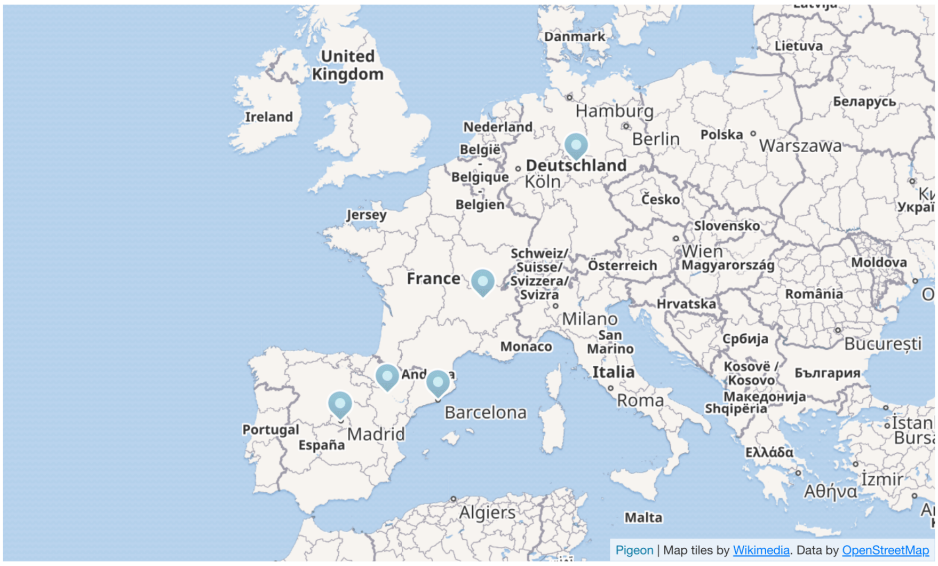
- Tracking & monitoring view



ID	LAB	PRODUCTION DATE	DISP	BATCH	UNITS	DESCRIPTION
Vaccine_batch_001	BioNTech	2020/12/12	2020/08/08	123111	20	ALLE DEMI CV333

NEW ALARM UPDATE

TRANSPORT COMPANY	TRANSPORTER	TEMPERATURE	HUMIDITY	LOCATION	
firm C	Pep	15	50	40.4165, -3.70256	MAP
firm C	Pep	15	38	41.65606, -0.87734	MAP
firm C	Pep	10	50	41.38879, 2.15899	MAP
firm B	David	15	50	45.74846, 4.84671	MAP
		15	50	51.165691, 10.451526	MAP



Zoom In Zoom Out

Video (Click to expand)

12.2 TrustOS Certs

TrustOS Certs is a demo that is based on Cert API and provides a platform to certify each document, file or digital process into Blockchain.

Example JSON structures (Click to expand)

12.2.1 1 - Login into the platform

POST - `/login`

```
{
  "id":
↪"did:vtn:trustid:8297475fa68e64de5128f0b819fae647f128969bad64bf5ad43f1797e5733eb0",
  "password":"test"
}
```

12.2.2 2 - Create certificate from a file

For this step we are going to use an API request, but also you could use the front-end demo to create the certificate in an easier way.

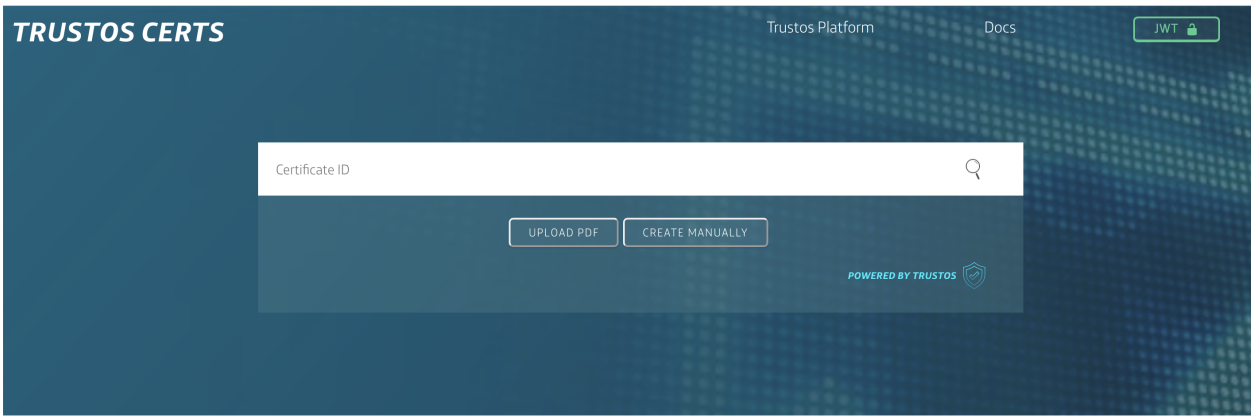
POST - `/certificate/content/create`

```
{
  "name": "Contract Certificate",
  "description": "This certificate is a tamper-proof and veriafable collection of␣
↪data that represent the Contract stored on Blockchain.",
  "content": {
    "filename":"Contract.pdf",
    "filehash":"c57c7ba270c4e67020a2944324559eb6d292068015647ec0ad112517ec05579e",
    "size":"40kb"
  },
  "public": false,
  "readers": [
    "did:vtn:reader1",
    "did:vtn:reader2"
  ],
  "signers": [
    "did:vtn:external:signer1",
    "did:vtn:trustid:db4c630673b8e8ca269149d6d611a84a20bb010f8db9b773b1a01576d5e5022b"
  ]
}
```

Now we have a unique and irrevocable `certID` that can be used for the next interactions for this certificate. Remember to copy it because it will be useful for next steps.

12.2.3 3 - Demonstrator

Now you can see the certificate in a more beautiful way though the simple demo that we've built.



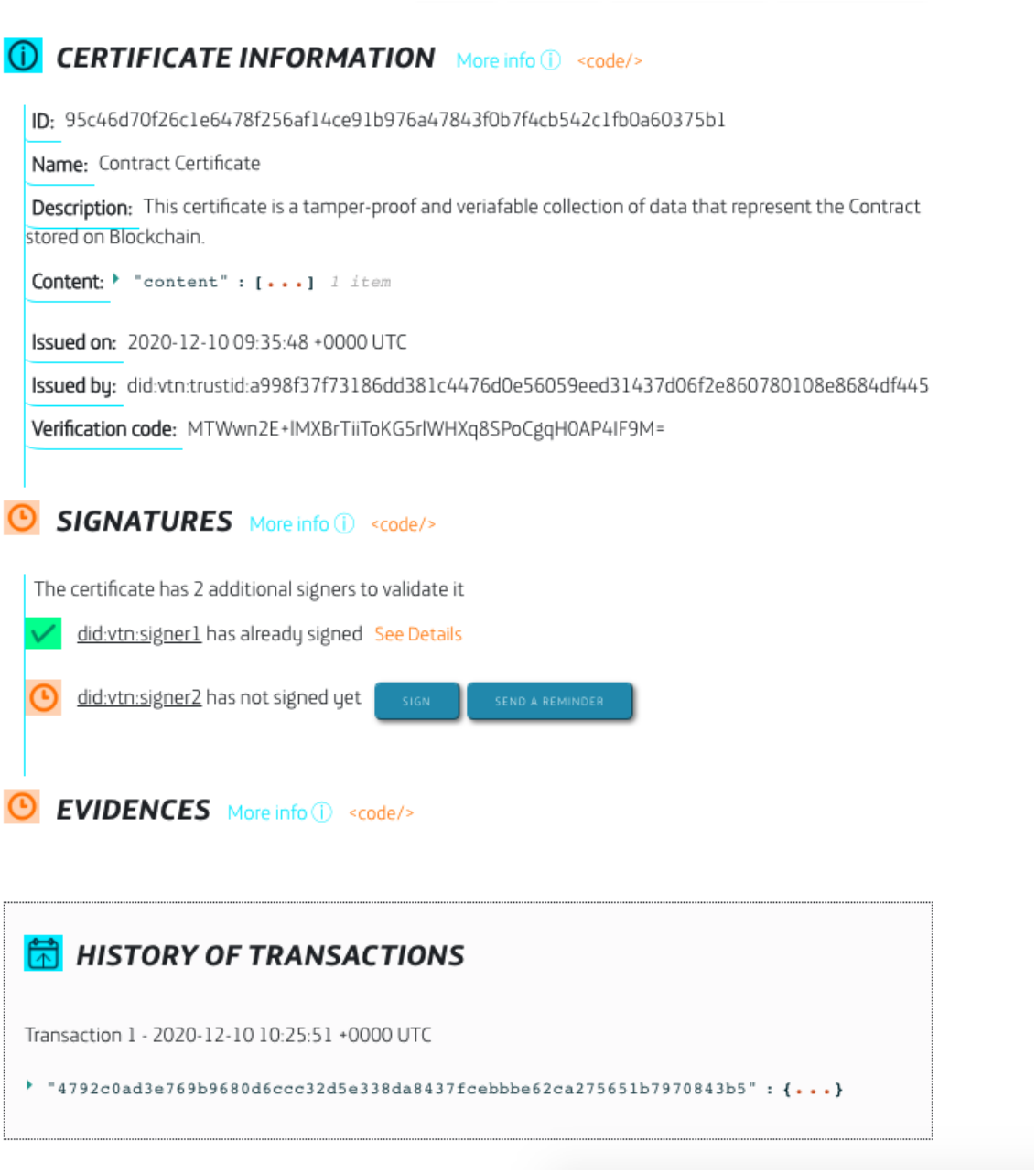
WHAT IS TRUSTOS CERTS?

TrustOS Certs is a **blockchain-based certification platform that allows to create and verify documents, actions, facts or states in a easy and fast way**. You can create, share, sign and verify every type of data as digital certificates, as same time as you legally protect your businesses and create trust links with your customers

HOW IT WORKS?

You just need to put the `certID` (e.g. `95c46d70f26c1e6478f256af14ce91b976a47843f0b7f4cb542c1fb0a60375b1`) to search for the information about the certificate: certificate information, signatures and other evidences.


TRUSTOS CERTS
Trustos Platform
JWT
Telefónica
BLOCKCHAIN
VALID CERTIFICATE
95c46d70f26c1e6478f256af14ce9...
The certificate with ID 95c46d70f26c1e6478f256af14ce91b976a47843f0b7f4cb542c1fb0a60375b1 and name Contract Certificate has been successfully registered in blockchain at 2020-12-10 09:35:48 +0000 UTC with valid format.
This certificate has been requested to be signed by 2 additional signers.
This is a tamper-proof and verifiable certificate powered by TrustOS.



12.2.4 4 - Sign certificate

To continue the process, the certificate can require one or more signers to validate and sign the certificate. This process can be done in two ways: using TrustID identities or using external identities/certificates. Every signer can only signe once, but will be able read, check and verify the certificate every time needed.

Now first we are simulating a signature from an external identity (`did:vtn:signer1`). The creator of the certificate is the responsable for this action, and it will appear as the "controller" once signed.

POST - `/certificate/{certID}/sign/external`

```
{
  "signature": "eyJSosd289ap389fa8uf3u8u4er8912rz...",
  "publicKey": "-----BEGIN PUBLIC KEY----- .... -----END PUBLIC KEY-----",
  "did": "did:vtn:external:signer1"
}
```

In case the signers were TrustID identities (created with ID API), those signers should login into the TrustOS platform and sign the certificate with their keys custodied by TrustOS.

POST - `/login`

```
{
  "id":
↪"did:vtn:trustid:db4c630673b8e8ca269149d6d611a84a20bb010f8db9b773b1a01576d5e5022b",
  "password":"test"
}
```

POST - `/certificate/{certID}/sign`

12.2.5 4 - Register and revoke certificate

In case your process need more transparency, an evidence of the certificate can be register in any of the public blockhain.

POST - `/certificate/{certID}/register`

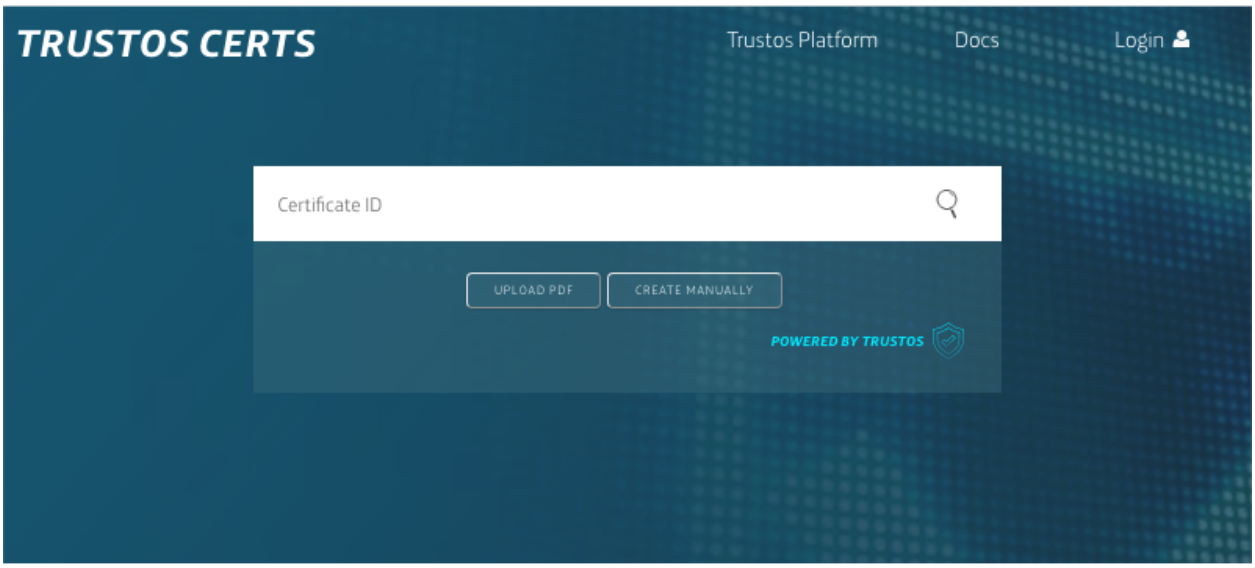
```
{
  "network": "Ethereum"
}
```

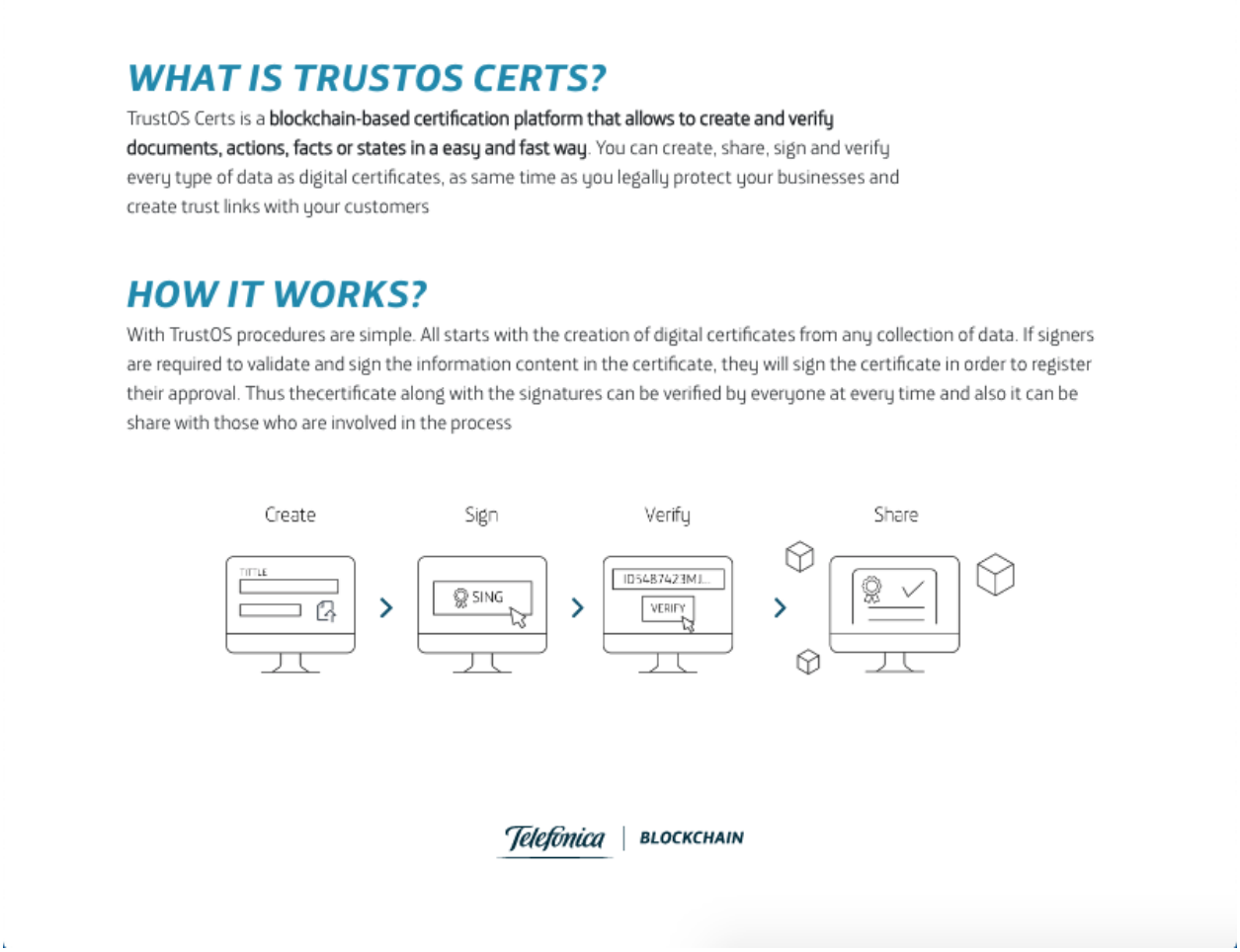
Once the issuer decides the certificate is no longer valid, it can be revoked.

POST - `/certificate/{certID}/revoke`

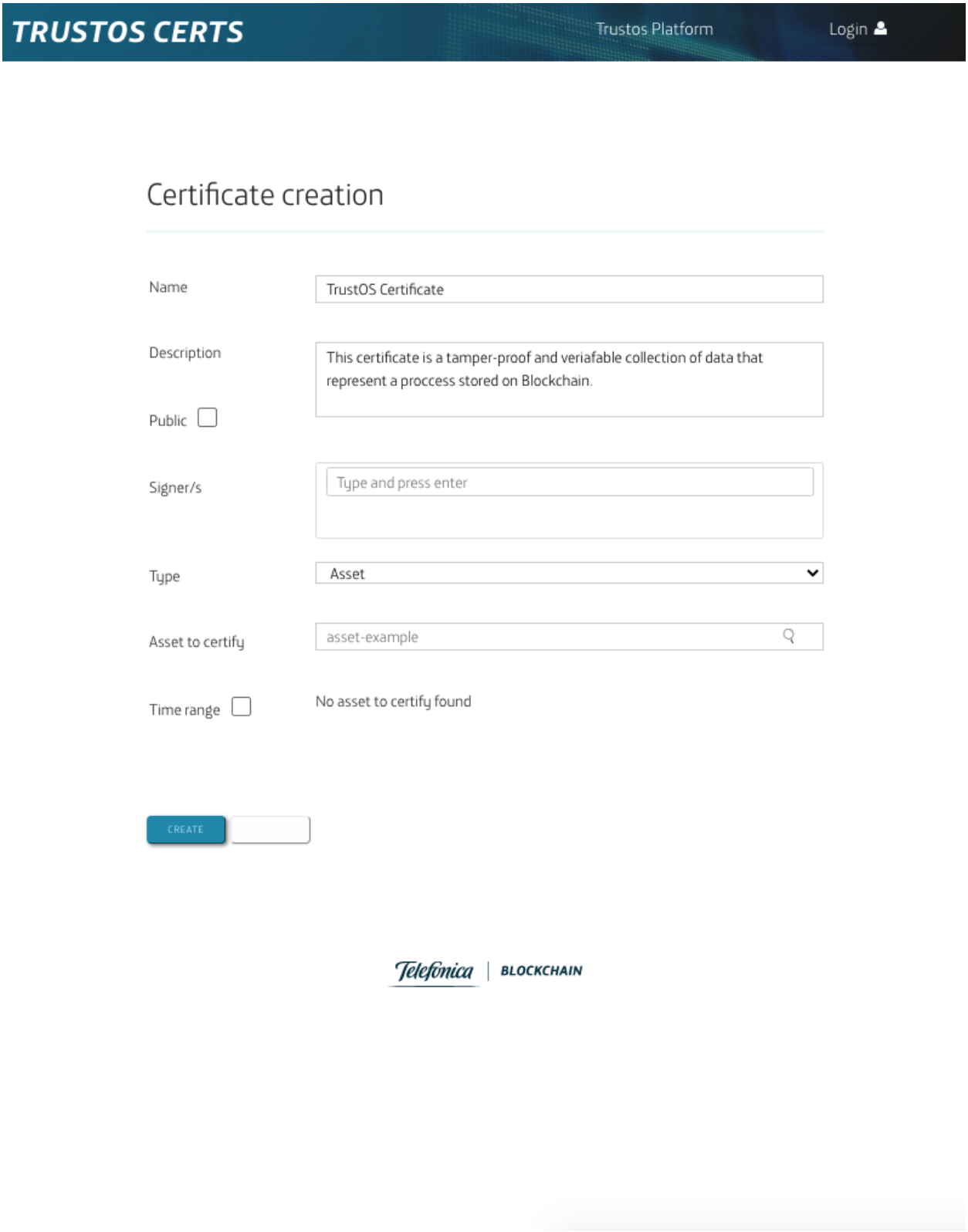
Front-end demo (Click to expand)

- Home view



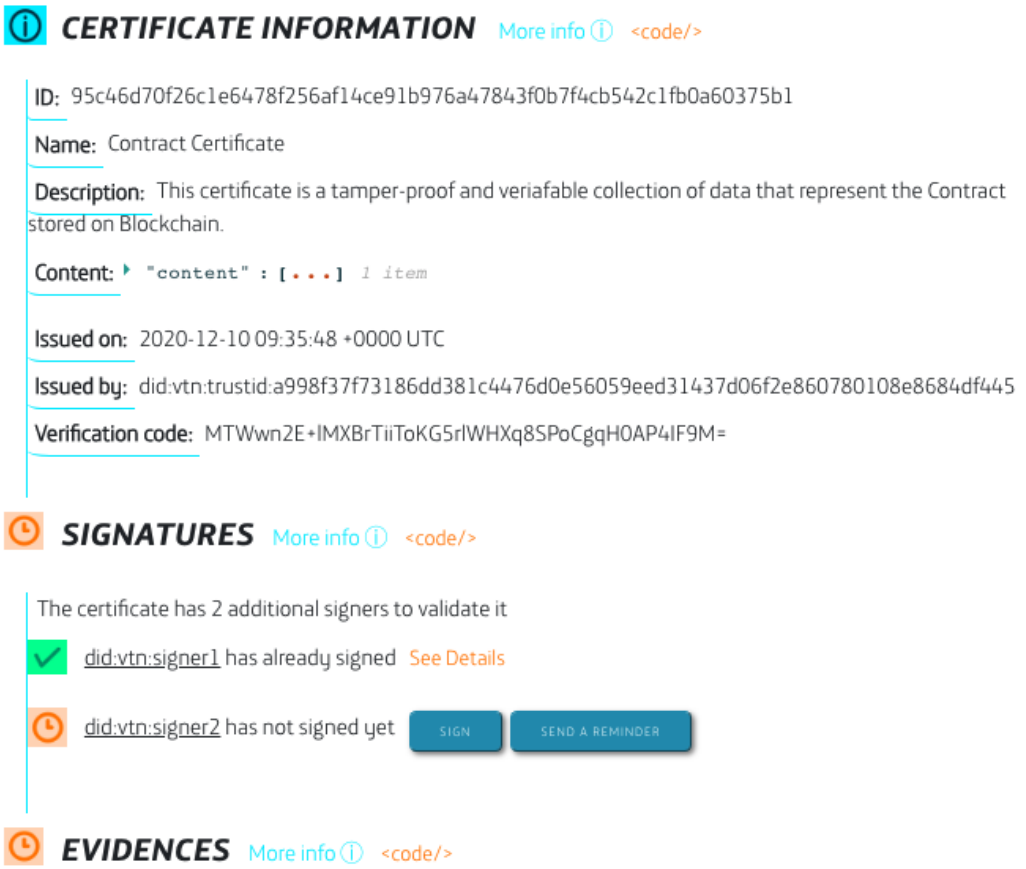


- Certificate creation view



- Certificate verification view


TRUSTOS CERTS
Trustos Platform
JWT
Telefónica
BLOCKCHAIN
VALID CERTIFICATE
95c46d70f26c1e6478f256af14ce9...
The certificate with ID 95c46d70f26c1e6478f256af14ce91b976a47843f0b7f4cb542c1fb0a60375b1 and name Contract Certificate has been successfully registered in blockchain at 2020-12-10 09:35:48 +0000 UTC with valid format.
This certificate has been requested to be signed by 2 additional signers.
This is a tamper-proof and verifiable certificate powered by TrustOS.



12.3 Karma

Karma is an easy demo based on Token API. The following steps will reproduce the real and easy implementation to carry out a success case like Karma, one of the innovative projects incubated in Telefónica to promote good actions in its employees. For this approach, the employees are awarded for participating in volunteer activities, organising interesting workshops, helping other employees and other non-profitable activities. With Karma, employees can bid for special experiences like a ticket for a concert or museum, a magazine or a coffee with a co-worker, just blocking an amount of tokens.

Example JSON structures (Click to expand)

12.3.1 1 - Login into the platform

POST - `/login`

```
{
  "id":
↪"did:vtn:trustid:8297475fa68e64de5128f0b819fae647f128969bad64bf5ad43f1797e5733eb0",
  "password":"test"
}
```

12.3.2 2 - Create customized token

POST - `/token/create`

```
{
  "name": "KARMA",
  "symbol": "KRM",
  "ethereumAddress": "0x0",
  "totalSupply": "100000"
}
```

At this point, you have created a token identified by the name: KARMA (`tokenId = KARMA`) and the total amount of KARMA tokens are transferred to the owner, that is, you.

12.3.3 3 - Transfer tokens to other user

POST - `/token/{tokenId}/transfer`

```
{
  "to": "Alice",
  "value": "40"
}
```

Now Alice received 40 KARMA tokens and you have lost it from your balance

12.3.4 4 - Block amount of tokens

POST - `/token/{tokenId}/transfer`

```
{
  "to": "Bob",
  "value": "80"
}
```

Now you have blocked 80 KARMA tokens, and Bob is the one who has to unblock it deciding whether to accept or return the total blocked amount. You can not use the blocked amount of tokens until the recipient unblocks it, in case Bob wants to return it...

12.3.5 5 - Get balance of user

GET - `/token/{tokenId}/balance/{userId}`

You will see your balance is 998880 KARMA tokens and... 80 blocked KARMA tokens

```
{
  "balance": 99880,
  "blocked_balance": 80
}
```

12.3.6 6 - Get transactions of user

GET - `/token/{tokenId}/transactions`

You will see the history of transactions you have done until now.

```
[
  {
    "id": "4f24ba1e6c8714ed89904f0e5bf9c2282e621b0fbc765655533543d29a24845f",
    "message": "Blocking transfer 80 from␣
↪did:vtn:trustid:18bbb416b43196359942918da78d5467379db8274cad2a5b207e4eb4048f4d69 to␣
↪Bob"
  },
  {
    "id": "b0e8a091817a1fb4352d69b117ae7dcd5ce85a2a05a601bd2a596f74eb64e8f1",
    "message": "Transfer 40 from␣
↪did:vtn:trustid:18bbb416b43196359942918da78d5467379db8274cad2a5b207e4eb4048f4d69 to␣
↪Alice"
  }
]
```

12.3.7 7 - Get transactions of other users (only for token owner)

GET - `/token/{tokenId}/transactions?userId=Bob`

Also you can see the history of transactions in which other users, as Bob, they are involved with. This functionality is for the owner of the token, that means, the one who created or the user who has received the token ownership once created.

```
[
    {
      "id": "4f24ba1e6c8714ed89904f0e5bf9c2282e621b0fbc765655533543d29a24845f",
      "message": "Blocking transfer 80 from␣
↪did:vtn:trustid:18bbb416b43196359942918da78d5467379db8274cad2a5b207e4eb4048f4d69 to␣
↪Bob"
    }
  ]
```

Video (Click to expand)

CHAPTER 13

Releases

TrustOS started with the initial v1.0.0 release. Since that several changes have been implemented in order to improve the functionality and performance of each API. Take a look at the last changes.

13.1 Track API

v2.1.0 [October 2020]

What's new in Track API v2.0.0:

- ***New concept of "authorised asset"***: Until now every asset belong to the user that has created it and there was not possibility of being consulted or updated by others unless it was transferred. Now two types of assets are handled: `own assets and authorised assets`. Thus it's possible for a user not only have access to its own assets but also the ones created by others. Having access to an authorised assets means that it's able to get the current state, to get the history of transactions and to update the metadata of that asset.
- ***New methods to handle authorisation and un-authorisation for an asset***: In this model two new methods are required in order to authorise or not a user to access a specific asset.`POST - .../asset/{assetId}/authorisePOST - .../asset/{assetId}/unauthorise`
- ***New flag to determine whether the request involves an authorised asset or not***: The current methods for consulting and updating an asset have to handle whether the transaction involves an own or authorised asset. For that reason and based on simplicity, a flag `...?isAuthorised=true` is turned on in query for routes: `/asset/{assetId}`, `/asset/{assetId}/transactions`, `/asset/{assetId}/transactions/range`, `/asset/{assetId}/update` to specify that it is an authorised asset. By default, with no flag it refers to an own asset. An example is shown below:`GET - .../asset/{assetId}?isAuthorised=true` <– Gets an authorised asset.`POST - .../asset/{assetId}/update?isAuthorised=true` <– Updates an authorised asset.
- ***New Smart Contract functionality to emit events based on rules***: Smart contracts have been enhanced with event handling. That means it is possible to establish a set of rules some wanted parameters have to accomplish. Smart Contract monitors the values and emit an event, or alarm, every time a value has differed a constant value or exceeded range of values.

- ***New method to establish rules for assets***: In this model a new method is required to define the rules the asset must follow. Thus it is possible to monitor the value or range of values for the parameters in every asset update. `POST - .../asset/{assetId}/rules`

Changes, future fixes and know issues:

- Every rule is applicable separately for every asset. So far it is not possible to re-use same rule for two or more assets. This would ease the usability and performance of this method. Moreover the rule can be add once the asset is created, and not during the creation that would be an interesting option. These features will be explored in future and included in a new release.
- Authorise & un-authorise methods return only a message. So far it is not possible to know what users are authorised to consult / update the asset. This feature will be explored in future and included in a new release.

v2.0.0 [June 2020]

What's new in Track API v2.0.0:

- Compatibility with Hyperledger Fabric v2.x.x
- Integration with TrustID (ID API) for interacting with the network and offer basic identity management services to users (create, import, export verify and sign)
- The Track chaincode (smart contract in Hyperledger Fabric) is now deployed as external services. It makes possible to deploy, handle and upgrade as an external container in a simple and fast way.

Changes, fixes and deprecations:

- The model of user credentials (user, password) used for `/login` method is now used with did credentials (id, password)
- Current user credentials are and looks like a DID credential: `did:vtn:trustid:289f893a9oir930pajklbx938ajzhd8` instead of: `user:org1MSP`
- The internal functions of invoking / querying a service (chaincode) are changed to new service model managed by TrustID
- Some minor fixes regarding API's HTTP response status code (2xx successful, 4xx client error, 5xx server error)

v1.1.0 [April 2020]

What's new in Track API v1.1.0:

- New trustpoint param in the Asset data model for grouping and showing information about trust points. Every call to createTrust or registerTrust functions (that is the creation/registration of the trust point in Hyperledger Fabric/Ethereum) involves an update of the trustpoint param in the asset. So this way allows to know when/where the trust point was generated.
- In order to increase the performance every update of the asset is now done in a separated and independent manner, so it doesn't keep the last state, just the asset data that is immutable. (f.e. when there is a creation/registration of a trustpoint, the new transaction just fill the trustpoint field, but not the metadata contained in the last asset transaction)
- News in Swagger UI: link to Readthedocs and data model section with proper nomenclature
- New `/refresh` method to able to refresh the JWT TOKEN without having to write again the user credentials

Changes, fixes and deprecations:

- The history/range of asset transactions now returns an array of txs also when there is only 1 transaction.
- The history/range of asset transaction is now returned from newest to oldest
- Postman collections are now updated

v1.0.0 [December 2019]

What's new in Track API v1.0.0:

- All basic tracking functionalities working: create, manage, and export digital assets on the blockchain
- API visualisation and interaction through Swagger UI
- Common errors management method implemented
- API integration testing with Hyperledger Fabric. Already successfully integrated with Hyperledger Fabric SDK and network and deployed through Kubernetes system with continuous integration and continuous deployment (CICD) mechanisms.

Changes, fixes and deprecations:

- All issues from previous releases (< v1.0.0) are solved and closed
- API code is cleaned and improved and unnecessary files are removed from the repository
- Third party dependencies are handled through Go modules

13.2 Token API

v2.0.0 [June 2020]

What's new in Token API v2.0.0:

- Compatibility with Hyperledger Fabric v2.x.x
- Integration with TrustID (ID API) for interacting with the network and offer basic identity management services to users (create, import, export verify and sign)
- The Token chaincode (smart contract in Hyperledger Fabric) is now deployed as external services. It makes possible to deploy, handle and upgrade as an external container in a simple and fast way.

Changes, fixes and deprecations:

- Owner of a token automatically set from the transaction issuer. Removed manual set of owner in `/create`
- The model of user credentials (user, password) used for `/login` method is now used with did credentials (id, password)
- Current user credentials are and looks like a DID credential: `did:vtn:trustid:289f893a9oir930pajklbx938ajzhd8` instead of: `user:org1MSP`
- The internal functions of invoking / querying a service (chaincode) are changed to new service model managed by TrustID
- Some minor fixes regarding API's HTTP response status code (2xx successful, 4xx client error, 5xx server error)

v1.1.0 [March 2020]

What's new in Token API v1.1.0:

- New token data model: a single chaincode can handle multiple tokens. Since the creation of a new token incurs sometimes in a timeout response error because of the process time while installing and instantiating a new chaincode, there was a need to implement a new data model based on a one single chaincode
- News in Swagger UI: link to Readthedocs and data model section with proper nomenclature
- New `/refresh` method to able to refresh the JWT TOKEN without having to write again the user credentials

Changes, fixes and deprecations:

- Changes in the implementation for `/token/{id}/transactions` path. Now it is allowed that a token owner can get the transactions of specific user in the way `/token/{id}/transactions?userId="test"`
- The paths `/token/initialize` and `/token/instantiate` are converged in `/token/create`
- Postman collections are now updated

v1.0.1 [January 2020]

What's new in Token API v1.0.1:

- Improvement based on writes performance. Now the token balances are handled separately as same way as the token information defined in the creation. Each one of the token transaction (f.e. token transfer transaction) are written in a separately way using completely different composite keys based on the user.

Changes, fixes and deprecations:

- FIXED: Known vulnerability that makes possible to upgrade the token replacing the token information defined in its creation.
- DEPRECATED: token implementation based on a unique token state which contains all the balances and token information in a unique key. It was deprecated because of the low performance.

v1.0.0 [December 2019]

What's new in Token API v1.0.0:

- All basic token functionalities working:
- API visualisation and interaction through Swagger UI
- Common errors management method implemented
- API integration testing with Hyperledger Fabric. Already successfully integrated with Hyperledger Fabric SDK and network and deployed through Kubernetes system with continuous integration and continuous deployment (CICD) mechanisms.

Changes, fixes and deprecations:

- All issues from previous releases (< v1.0.0) are solved and closed
- API code is cleaned and improved and unnecessary files are removed from the repository
- Third party dependencies are handled through Go modules

13.3 Settle API

v2.0.0 [June 2020]

What's new in Settle API v2.0.0:

- Compatibility with Hyperledger Fabric v2.x.x
- Integration with TrustID (ID API) for interacting with the network and offer basic identity management services to users (create, import, export verify and sign)
- The Settle chaincode (smart contract in Hyperledger Fabric) is now deployed as external services. It makes possible to deploy, handle and upgrade as an external container in a simple and fast way.

Changes, fixes and deprecations:

- The model of user credentials (user, password) used for `/login` method is now used with did credentials (id, password)

- Current user credentials are and looks like a DID credential: `did:vtn:trustid:289f893a9oir930pajklbx938ajzhd8` instead of: `user:org1MSP`
- The internal functions of invoking / querying a service (chaincode) are changed to new service model managed by TrustID
- Some minor fixes regarding API's HTTP response status code (2xx successful, 4xx client error, 5xx server error)

v1.1.0 [April 2020]

What's new in Settle API v1.1.0:

- In order to increase the performance every update of the settlement with record is now done in a separated and independent way through composite keys implementation
- New `/settlement/{id}/global` function to get global status of the settlement structure.
- New `/refresh` method to able to refresh the JWT TOKEN without having to write again the user credentials

Changes, fixes and deprecations:

- Splitted get settlement structure function into get status for the logged user and get global status.
- The function for getting the global status `/settlement/{id}` only gets the status for the logged user.
- Postman collections are now updated

v1.0.0 [December 2019]

What's new in Settle API v1.0.0:

- All basic settlement functionalities working: create, update, aggregate, and settle a settlement structure on the blockchain
- API visualisation and interaction through Swagger UI
- Ensuring the transparency through hash and merkle tree mechanisms
- Common errors management method implemented
- API integration testing with Hyperledger Fabric. Already successfully integrated with Hyperledger Fabric SDK and network and deployed through Kubernetes system with continuous integration and continuous deployment (CICD) mechanisms.

Changes, fixes and deprecations:

- All issues from previous releases (< v1.0.0) are solved and closed
- API code is cleaned and improved and unnecessary files are removed from the repository
- Third party dependencies are handled through Go modules

13.4 Cert API

v1.1.0 [April 2021]

What's new in Cert API v1.1.0:

- New methods for managing advanced signature:
  - `/certificate/{certID}/advancedsignature/init` - Initialise an advanced signature process for a certificate.
  - `/certificate/{certID}/advancedsignature/status` - Check the status of advanced signature for a certificate

  - `/certificate/{certID}/advancedsignature/document` - Get the signed document for a certificate with advanced signature
  - `/certificate/advancedsignature/notification`- Receive notifications for an advanced signature process. It requires a new way of authorization based on "Basic Authorization" in order to accept incoming request from third party notification services.

  It is the first integration with a third party platform: VIDSigner. Others will be integrated in next releases.

- Added `init` and `end` parameters in `/certificate/asset/create` to allow the certification of a range of transactions for an asset
- Now Cert API has integration with OpenID Connect, enabling the use of the functionalities with different identity providers such as Google or Microsoft.
- Added `public` parameter in the creation data model for the certificate creation
- Added `access` information in response for `/certificate/{certID}` and `/certificate/{certID/history` methods

Changes, fixes and deprecations:

- Swagger methods and responses have been updated
- Postman collection has been updated
- Fixed issue #4: checking that advanced signature is initialized in `/advancedsign/{certID}/status` and `/advancedsign/{certID}/document` methods in order to avoid possible errors
- Fixed issue #2: Handling bad input in `/advancedsign/{certID}/init` method

v1.0.0 [January 2021]

What's new in Cert API v1.0.0:

- All basic trust functionalities working:
  - Core methods: create, get, sign, register, revoke, and get history of certificates on blockchain.
  - Other useful methods: healthcheck, get certificates by user, external sign and managing access / signers.
- There are two possible digital signatures (Public Key Infrastructure):
  - Sign with a certificate generated in TrustOS
  - Sign with an external certificate (X.509 standard)
- All functionalities are applicable to two types of certificates:
  - Content certificate: Certificate with specific and customisable content like file/document/collection of files.
  - Asset certificate: Certificate based on an existing asset generated in Track module.
- All certificates follows the same data structure:
  - CertID - Unique identifier of the certificate
  - Data - JSON of certificate information that is inmutable
  - Metadata - Array of JSON transactions that feed the certificate (e.g. signatures, revocation and public evidences)
  - Access - JSON of granted accesses to interact with the certificate (e.g. admin, sign, read access)
- API visualisation and interaction through Swagger UI

- API integration testing with Hyperledger Fabric. Already successfully integrated with Hyperledger Fabric SDK and network and deployed through Kubernetes system with continuous integration and continuous deployment (CICD) mechanisms.

Changes, fixes and deprecations:

None.

13.5 Trust API

v2.0.0 [June 2020]

What's new in Trust API v2.0.0:

- Compatibility with Hyperledger Fabric v2.x.x
- Integration with TrustID (ID API) for interacting with the network and offer basic identity management services to users (create, import, export verify and sign)
- Some minor fixes regarding API's HTTP response status code (2xx successful, 4xx client error, 5xx server error)
- The Trust chaincode (smart contract in Hyperledger Fabric) is now deployed as external services. It makes possible to deploy, handle and upgrade as an external container in a simple and fast way.

Changes, fixes and deprecations:

- New body input field in `/register`and `/create` methods to allow to fill the asset's metadata once the /registration/creation of a trustpoint is done and the asset is updated with the trustpoint info (ethereum transaction & contract or hyperledger fabric transaction)
- The model of user credentials (user, password) used for `/login` method is now used with did credentials (id, password)
- Current user credentials are and looks like a DID credential: `did:vtn:trustid:289f893a9oir930pajklbx938ajzhd8` instead of: `user:org1MSP`
- The internal functions of invoking / querying a service (chaincode) are changed to new service model managed by TrustID

v1.0.0 [December 2019]

What's new in Trust API v1.0.0:

- All basic trust functionalities working: create, manage and public register trust points
- API visualisation and interaction through Swagger UI
- Ensuring the transparency through hash and merkle tree mechanisms
- Common errors management method implemented
- API integration testing with Hyperledger Fabric. Already successfully integrated with Hyperledger Fabric SDK and network and deployed through Kubernetes system with continuous integration and continuous deployment (CICD) mechanisms.

Changes, fixes and deprecations:

- All issues from previous releases (< v1.0.0) are solved and closed
- API code is cleaned and improved and unnecessary files are removed from the repository
- Third party dependencies are handled through Go modules

CHAPTER 14

Contributing

You are more than welcome to contribute with TrustOS!

Soon it will be open to the community of developers who want to take part of this amazing project. The next new module of TrustOS can be developed by yourself. Stay tuned!

Note: If you have questions not addressed by this documentation, or run into issues with any of the tutorials, please contact us to find additional help.