Title:	Document Version:
D3.7 - Permissioned Blockchain network M27	1.0

Project Number:	Project Acronym:	Project Title:
H2020-871493	DataPorts	A Data Platform for the Cognitive Ports of the Future

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security*:
M27 (March 2022)	M27 (March 2022)	O-PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other; ORDP: Open Research Data Pilot; E: Ethics.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
Fabiana Fournier	IBM	WP3

Authors (organisation):	
Fabiana Fournier (IBM)	Inna Skarbovsky (IBM)
Miguel Llop (VPF)	Pablo Gimenez (VPF)
Vicente Perales (VPF)	Daniel Garcia Latorre (EVR)
Alejandro Aparicio Blasco (EVR)	Daniel Vilas (EVR)
Alexandros Zerzelidis (CERTH)	Vasilis Siopidis (CERTH)

Abstract:

Permissioned Blockchain network deliverable provides the design details for the enhancements of the three blockchain prototypes that have been developed in the project, namely the Data Governance, Verify Gross Mass, and Container Pick-Up. These extensions have been identified while implementing the first version of the prototypes or have arisen as result of new requirements. Each new feature or capability in each of the blockchain networks is described in terms of the requirements and a detailed design. Special attention and efforts have been devoted to the integration of the prototypes into the DataPorts platform and the operational systems in the ports. The next and final version of the blockchain prototypes implementation will be based on the presented design.

Keywords:

Hyperledger Fabric, blockchain, blockchain network, data governance

Revision History

Revision	Date	Description	Author (Organisation)
V0.1	10.02.2022	TOC	Fabiana Fournier (IBM)
V0.2	20.02.2022	VGM use case section	Inna Skarbovsky (IBM), Miguel Llop (VPF), Vicente Perales (VPF), and Pablo Gimenez (VPF)
V0.3	30.02.2022	CPU use case section	Alexandros Zerzelidis (CERTH)
V0.4	05.03.2022	Introductory section	Fabiana Fournier (IBM)
V0.5	07.03.2022	Data Governance section	Alejandro Aparicio Blasco (EVR), Daniel Garcia Latorre (EVR), and Alexandros Zerzelidis (CERTH)
V0.6	09.03.2022	Conclusions and abstract	Fabiana Fournier (IBM)
V0.7	10.03.2022	Submission for internal review	Fabiana Fournier (IBM)
V1.0	30.03.2022	Submission to the EU	Fabiana Fournier (IBM)

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement № 871493.

More information available at https://DataPorts-project.eu

Copyright Statement

INDEX

LIST OF FIGURES

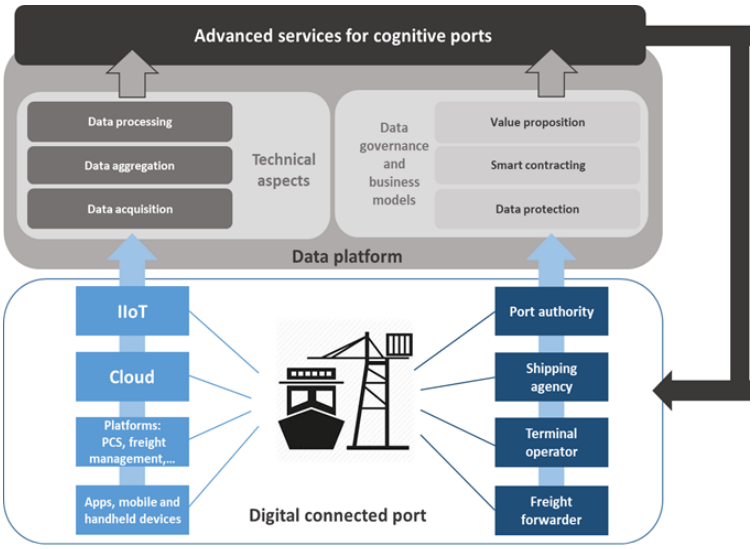
LIST OF TABLES

1 INTRODUCTION

1.1 DATAPORTS PROJECT OVERVIEW

DataPorts is a project funded by the European Commission as part of the H2020 Big Data Value PPP programme, and coordinated by the ITI - Technological Institute of Informatics. DataPorts rely on the participation of 13 partners from five different nationalities. The project involves the design and implementation of a data platform, its deployment in two relevant European seaports connecting to their existing digital infrastructures and addressing specific local constraints. Furthermore, a global use case involving these two ports and other actors and targeting inter-port objectives, and all the actions to foster the adoption of the platform at European level.

Hundreds of different European seaports collaborate with each other, exchanging different digital data from several data sources. However, to achieve efficient collaboration and benefit from AI-based technology, a new integrating environment is needed. To this end, DataPorts project is designing and implementing an Industrial Data Platform.



The DataPorts Platform aim is to connect to the different digital infrastructures currently existing in digital seaports, enabling the interconnection of a wide variety of systems into a tightly integrated ecosystem. In addition, to set the policies for a trusted and reliable data sharing and trading based on data owners' rules and offering a clear value proposition. Finally, to leverage on the data collected to provide advanced Data Analytic services based on which the different actors in the port value chain could develop novel AI and cognitive applications.

DataPorts will allow establish a future Data Space unique for all maritime ports of Europe and contribute to the EC global objective of creating a Common European Data Space.

1.2 DELIVERABLE PURPOSE AND SCOPE

This document provides a comprehensive report on the design of the second version of the three blockchain (BC) networks prototypes identified in the DataPorts project: Data Governance (DG), Verified Gross Mass (VGM), and Container Pick-Up (CPU).

Specifically, the Description of Action (DoA) states the following regarding deliverable D3.7 - Permission Blockchain Network: "This deliverable will implement the permissioned blockchain infrastructure that will define data models, authentication and authorization mechanisms, and that will integrate with the rest of components of the platform". Specifically, D3.7 relates to Task 3.5 (T3.5) – Blockchain implementation led by IBM and carried out by partners IBM, ITI, EVR, and CERTH.

The purpose of this document is to present the design work that has been carried out in each of the three blockchain networks in the project from M19 to M27. In essence, this reflects the advances of the three BC networks from their first Minimal Viable Product (MVP) detailed in D4.4 - Blockchain Based Data Governance Rules M20[1], towards their second version to be provided at M30 in the scope of D4.2 - Blockchain based data governance rules M30. Figure 1 sketches the relationships between the BC deliverables. The two deliverables

[1] https://dataports-project.eu/deliverables/

in work package 3 (WP3) are related to the design of the envisioned BC networks, while the second version (the document in hand) is a more advanced version of the first deliverable (D3.4[1]) submitted in M18. Likewise, the two deliverables in WP4 reflect the implementation work carried out for each of the articulated designs, where D4.4 (to be submitted in M30) is the second and more advanced version of the first prototypes presented in D4.2. In addition, the current document (D3.7) reflects the new requirements that arose after the implementation of the first prototypes and therefore is also fed from D4.2.

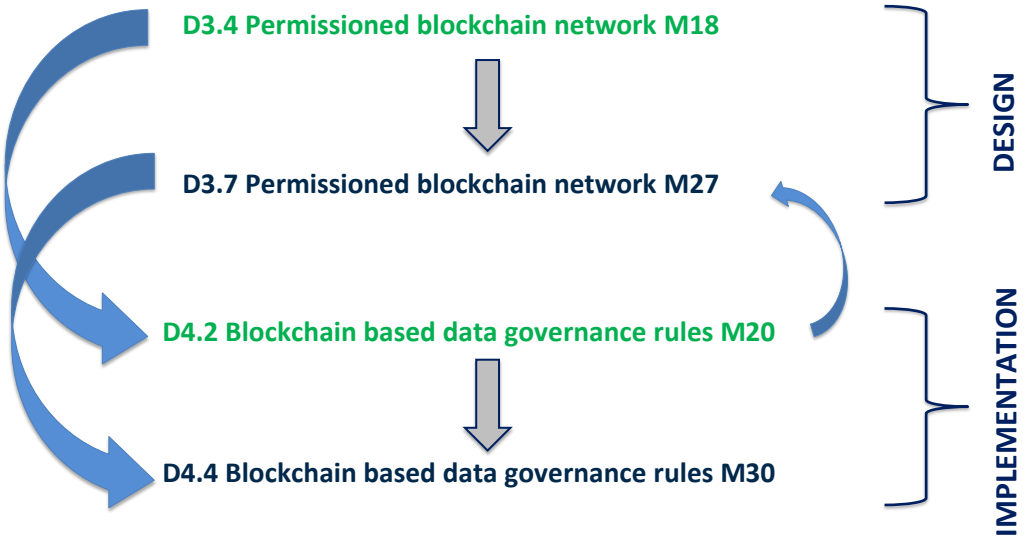


Figure 1 - Blockchain related deliverables sequencing

We assume the reader is familiar with the concepts provided in the corresponding previous deliverables and specifically technical blockchain related concepts, with emphasis on Hyperledger Fabric (simply Fabric). Therefore, description of concepts such as peers, organizations, orderers, certificate authority (CA), channels, and chaincodes (i.e., smart contracts in the Hyperledger Fabric jargon) are out of the scope of this document. For more details on Hyperledger Fabric the reader can refer to [1] and [2].

We also assume the reader is familiar with previous related documents as shown in Figure 1 and Section 1.3, as the document in hand builds upon them and merely details new requirements and features identified from M18 to M27. The design work introduced in this report guides the development of the second round of MVPs for each of our three networks. Some of the features, as indicated in Sections 2, 3, and 4 have already been implemented while others are still under development and will be completed and documented in D4.4 by M30.

1.3 DELIVERABLE CONTEXT

Its relationship to other documents is as follows:

Primary Preceding documents:

- D3.4 – Permissioned Blockchain network M18. Provided the detailed design of the three envisioned blockchain networks in the project. These deliverable complements this first design with details for the second and final versions of the developed prototypes.
- D2.4 - Platform architecture and specifications at M24. Provided the technical and functional specifications of the components composing the Dataports platform and therefore relevant to understanding the interactions between the blockchain components and their counterparts in the platform.

Primary Dependant documents:

- D4.2 - Blockchain based data governance rules M20. Provided a comprehensive presentation of the three BC networks first implementation MVP. In fact, the report in hand complements the detailed design for the subsequent version of these three MVPs.

1.4 DOCUMENT STRUCTURE

The main purpose of this deliverable in to present the advances in the design of the three BC networks supported by the DataPorts platform. To this end, for each of the networks, we relate to each of the new features/capabilities and describe the requirements of the feature and its design. The list of new features is based on the developments on the prototypes as envisioned in D4.2 next steps and new requirements that arose after M20.

This deliverable is broken down in the following sections:

- **Section 1** Intro: Summary of the deliverable objective, scope, and structure**.**
- **Section 2** Data Governance blockchain network MVP version 2 detailed design
- **Section 3** Verified Gross Mass (VGM) blockchain network MVP version 2 detailed design
- **Section 4** Container Pick-Up (CPU) blockchain network MVP version 2 detailed design
- **Section 5** Summary and conclusions

1.5 DOCUMENT DEPENDENCIES

This document is part of an iteration of living deliverables and constitutes the last variant. The first and previous variant was delivered in M18.

2 DATA GOVERNANCE NETWORK

This section describes the design of the feature additions and extensions to the Data Governance (DG) blockchain (BC) application as devised in D3.4 and D4.2 deliverables. Specifically, in the summary and next steps sections of D3.4 and D4.2 we have outlined a list of extensions proposed for the Data Governance Minimal Viable Product (MVP) second version (MVP v2). These are presented in Table 1 below, along with the status for each of these features. The latter can be: "done" (implemented at the time of the writing of this document); "in progress" (to be finalized by the submission of D4.4); and "dropped", meaning the specific feature won't be part of the Data Governance MVP v2.

Some of the entries are the natural progression of the initial MVP. Others are new features and extensions we had in mind for version 2 of the MVP. Additionally, some of the extensions appearing in the table were not initially planned, and therefore, not included in the previous deliverables, however the need for such came up during the development of other features. Those are added at the end of the table and denoted with an asterisk (*).

#	Feature/activity	Status	Comments
1	Integration with DataPorts components	In progress	Awaiting the Identity Manager (IM) integration
2	Organizational and public access	In progress	Extension of the DG frontend to support the new backend features
3	Add logging for access requests, access revocations (made by the provider) and requests to revoke access (made by the consumer)	In progress	Extension of the DG frontend to support the new backend features
4	User level access additions (*)	In progress	Extension of the DG frontend to support the new backend features
5	Register & Login additions/updates (*)	Done	
6	Definition of the roles: Data provider and data owner (*)	In progress	Extension of the DG frontend to support the new backend features

Table 1 - Data Governance MVP version 2 new features and their status

In the following sections we address each of the features and extensions and explain the reason for including the feature in v2 of the MVP, the requirements for it, and its design.

2.1 INTEGRATION WITH DATAPORTS COMPONENTS

In this section, we describe the design of the integration of Data Governance with other DataPorts components. An overall overview of the integration has already been presented in deliverable D2.4[1]. Here we give an in-depth description on the design of this integration and its functionality. This integration will be part of the updated second version of the Data Governance MVP (MVP v2).

2.1.1 Requirements

To successfully integrate Data Governance with the rest of the DataPorts platform components, we have identified the following five requirements.

1) The actions the various users of Data Governance can perform, be it users of blockchain-participant organizations, users of non-participant organizations (external users), or other components of the DataPorts platform, must be clearly specified.
2) Data Governance must satisfy the security principle of *authorisation* with respect to user access to datasets. To this end, the implementation of *external users* must be concretely specified.
3) In order for data consumers to access datasets, Data Governance must confirm their eligibility to do so. The following two scenarios are considered, under which a consumer can try to access a dataset:
   - Datasets are read from the data source on behalf of the consumer by means of an agent software provided by the Data Access component of data Processing Services, which then forwards the data to the consumer.
   - Datasets are read by the consumer directly from the data source made available by the provider, e.g., an endpoint.

   Data Governance must be consulted in both scenarios, with security principles always being satisfied.
4) For a dataset to be fully searchable through Data Governance, the Semantic Interoperability (SI) component must be able to pass its data source metadata of the dataset in question to Data Governance. Deliverable D2.4 describes this interaction between Semantic Interoperability and Data Governance in Section 5.2, where it states: "…the Semantic Interoperability layer communicates with the Data Governance component, registering the dataset metadata in the Broker, i.e., the data source metadata…"
5) The Data Analytics component of the platform will need to interact with Data Governance to query about user permissions and obtain metadata.

2.1.2 Design

For each of the above requirements we detail its design for the MVP v2 of the DG blockchain network.

2.1.2.1 Actions available to Data Governance users

The main functionality offered by the BC Data Governance component is to provide a space where authorised users can define datasets, describe them with relevant metadata, and assign access rights to those datasets for other users. These users are primarily meant to be blockchain network users, belonging to a blockchain participant organization. However, there is a need for DataPorts to also accommodate organizations, and hence users, that do not belong to the blockchain network. These would be organizations that would not feel inclined to setup and maintain the whole Fabric blockchain infrastructure (e.g., endorsing peer node and CAs), but would be still interested in participating in the exchanging of data taking place within DataPorts.

Furthermore, there is another kind of Data Governance user that we need to take into consideration. These are other DataPorts components that need to interact with Data Governance to, for example, query a particular user's permissions or to update a dataset's metadata.

The identification of the types of users is necessary to decide which of the features of Data Governance each of them can use. Table 2 shows which of the main features of Data Governance the three user types can perform. The "normal" Data Governance user, that is, a user that belongs to a blockchain participant organization can perform all available actions, except special actions, such as, updating a data source metadata. "External" users, i.e., users whose organization is not part of the blockchain network, can do everything a "normal" user can, except receiving organizational access. Finally, the "special users" representing other DataPorts components, e.g., Semantic Interoperability, cannot do anything apart from performing queries on the metadata and performing special actions, like updating a data source metadata.

User categories	Act as Provider	Act as Consumer	Can perform special actions, e.g. update data source metadata	User-level access to data			Organisational access to data			Public access to data	Query metadata
				read	write	share	read	write	share		
Data Governance user	☑	☑	☒	☑	☑	☑	☑	☑	☑	☑	☑
Special DataPorts component user	N/A	N/A	☑	N/A	N/A	N/A	N/A	N/A	N/A	N/A	☑
User whose organisation is not part of the Data Governance Fabric network	☑	☑	☒	☑	☑	☑	☒	☒	☒	☑	☑

Table 2 - Permitted user actions in the Data Governance BC component

We denote that all non-special users are able to update the Data Governance metadata for datasets they are providers of, but not data source metadata, since that comes directly from the Semantic Interoperability component (refer to 2.1.2.5).

2.1.2.2 External users

External users are those users that belong to organizations not participating in the Data Governance blockchain network. To provide authorisation, DG essentially keeps records of triplets (dataset, user, permissions). However, while considering the existence of external users, the question of how to implement them becomes paramount. There are two main alternatives:

- External users will be implemented as extraordinary users of existing blockchain participant organizations.
- External users will not exist as an entity in Data Governance, but rather permissions will be assigned based on username strings.

Choosing the first approach would mean that for every external user participating in the DataPorts ecosystem (as a data consumer and/or provider), a user will be created in Data Governance. There are two options to accomplish this:

- External users are created as full Fabric users with Fabric identity, or
- External users are created only as entities saved in the blockchain state or in an external database (which is usually preferred)

To implement the first option, external users would be created under a mock “umbrella” organization, while an extra attribute could be added to the user entity to keep track of the name of the external organization they belong to. This way, keeping user permissions and ruling on access requests will be done as per current implementation. The second option entails extra logic to be implemented to cater for the management of the new entities and the management of the external database if that option is chosen.

Nonetheless, no matter the option, the implication of having Data Governance know all external users on its own is that DG does not need to interact with the identity manager in any way, other than when it needs to prove its own identity to the DataPorts platform’s Application Programming Interface (API) Gateway to get the data source metadata or to create a new subscription for metadata events. Every call to Data Governance will be coming from user whom DG directly knowns and therefore using the user’s credentials in the call would be enough to authenticate the call and adjudicate on the user’s authorisation status.

Choosing the second solution (i.e., external users as dummy users), would lead to potential conflicts when it comes to usernames. The existing chaincode would not be able to tell whether a particular username has been used for more than one external user. To keep track of that, an external identity manager would have to be used to register all external users, but then the matter of synchronization between Data Governance and the external identity manager comes into play. Synchronization is, in general, a delicate issue that can pose unforeseen problems if not implemented correctly.

Having laid out the different approaches to the issue, we choose to implement external users as Fabric users with Fabric identity. This approach poses the least risk to the existing implementation because it is taking advantage of the way users are currently implemented.

2.1.2.3 Integration with an Identity Management platform

In order to integrate Data Governance with the rest of the DataPorts platform, it must share the same identity management as other components. For example, when Data Governance needs to make a call to another DataPorts component, it needs to prove that it is indeed who it claims to be and not some malicious external entity. To achieve that we need to integrate Data Governance with an Identity Management (IM) solution. The IM will be an identity-brokering component that will be deployed alongside the rest of the DataPorts components.

Optionally, the IM could also be used to identify other components and DataPorts users to the Data Governance component. However, this is not necessary, since Data Governance has its own identity management component and anyone wishing to use Data Governance could be assigned their own account and credentials on Data Governance. The decision on whether the IM will be used to identify components or users to Data Governance will be taken in the following weeks and certainly well before the deadline for the final implementation of the platform. The following paragraphs describe the use of the IM component, if we choose to exercise this option.

Therefore, if the Identity Manager is to be also used for API calls made by other components to Data Governance, Data Governance will have to translate the transaction invoker's identity to on-chain permissions for data access. In other words, the identity used by the calling application has to be linked to an identity in the Fabric network, and the transactions that the blockchain user performs have to be associated to an identity in the IM.

Keycloak[2] is an open-source identity management tool that provides access management for users and can generate single use tokens for authentication. JSON Web Tokens (JWTs) are encoded credentials that allow users to be identified in the platform. Keycloak can generate these tokens to be assigned to a Data Governance user. JWT have a common encoded structure that can be exchanged between parties. Keycloak can be deployed in a docker container like the rest of Data Governance components, for ease of management and maintenance.

Figure 2 describes the general structure of a JWT token. The structure of the header, i.e., `Header.Payload.Signature`, contains the encoding algorithm type and the type of token; the payload contains all the relevant information to be included in the transaction, and the signature holds the secret key that verifies the validity of the token.

[2] https://www.keycloak.org/

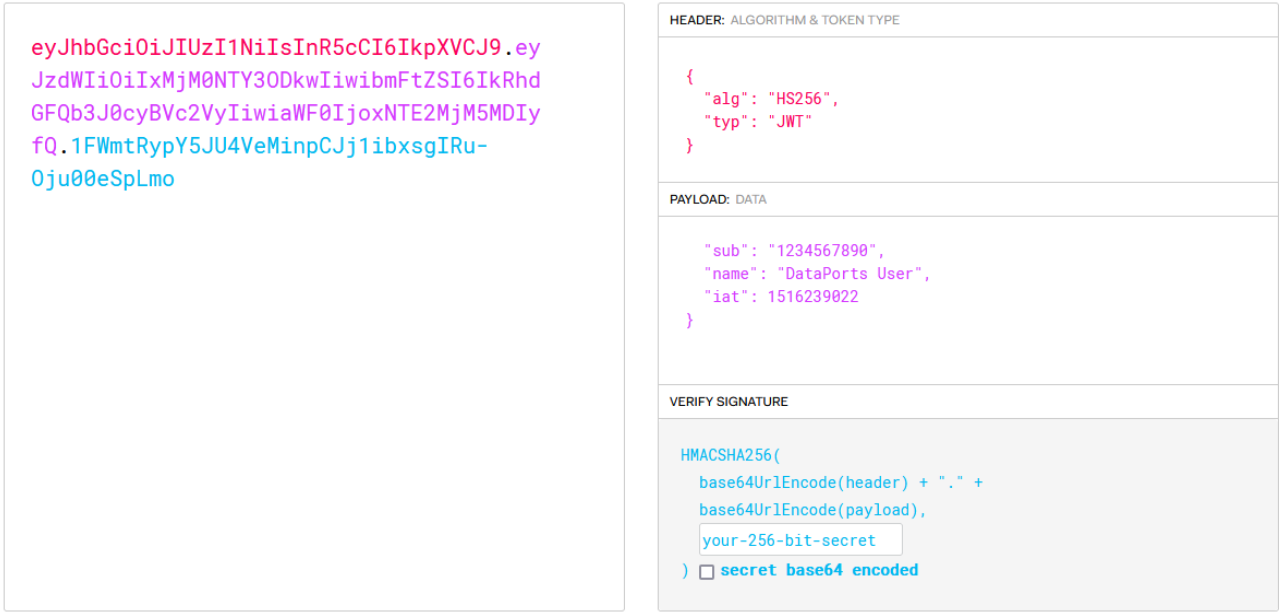


Figure 2 - General structure of a JWT token

In Keycloak, users are grouped into user realms. A realm therefore is created to host all Data Governance users, so as to provide them with a single identification for the tokens to be issued. The token will hold this information in the Signature portion of the token.

When logging into the application, a JWT is generated in Keycloak for every off-chain request, and submitted to the API Gateway. The session token is linked to the identity of the DataPorts user at the moment of minting the token. This token is then passed on to the Fabric API, which will in turn link the associated token to an identity in blockchain that is provided by the Fabric Certificate Authority. The JWT is included in the blockchain transaction proposal to be also evaluated by the chaincode.

The `getUserFromTx` method in the Data Governance chaincode is modified to check the information contained in the token and allow or deny authorization for the transaction proposal. To this end, the chaincode must receive as inputs the issuer ID and the public key of the issuer, data that is included in the token. This data is originally generated in Keycloak according to the user for which the token was issued and the Keycloak realm that the user belongs to. A detailed description on how this public ID is passed on to the Data Governance component will be provided in the coming D4.4.

Each and every off-chain transaction proposal that takes place in the application must be accompanied by a Keycloak-issued JWT token. Therefore, the tokens are used only once and only for a specific operation. Any following operation, even if performed by the same user, will need a new token freshly minted by Keycloak, with a new payload and its corresponding digital signature. Every chaincode that checks for the user identity must make a call to the modified method and receive new JWT inputs for each following operation.

Figure 3 depicts the corresponding identification process.

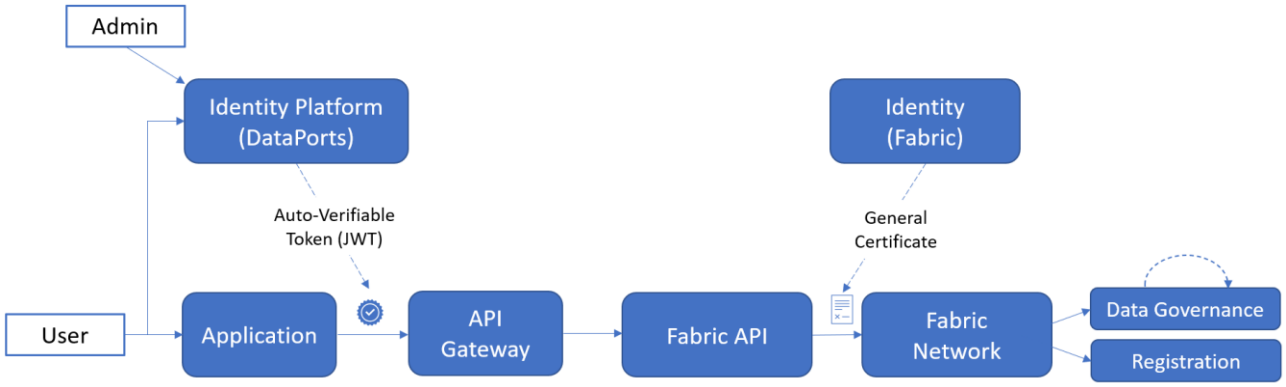


Figure 3 - Identification via Keycloak for Hyperledger Fabric integration

2.1.2.4 Data consumer access to datasets

The acquisition of a dataset by a consumer directly involves the Data Governance component, since it is the component tasked with keeping dataset access rights. Therefore, irrespective of whether an agent is being used or not, Data Governance must be consulted. Following we present the designs for both cases bringing two alternatives for each case. These are influenced by the overall integration of the components in the platform and still under discussions at the moment of writing the document in hand.

2.1.2.4.1 Dataset obtained through Data Access

Deliverable D2.4 describes this interaction between Semantic Interoperability and Data Governance, stating "…the Semantic Interoperability Component API interacts with the Data Governance components to enforce the data access policies…".

Figure 4 shows the scenario where a consumer wants to retrieve a dataset using the API Gateway of a DataPorts platform installation. The consumer first needs to obtain a JWT token from the DataPorts Identity Manager. Once available, the token is placed in the Authorization header of a REST API call to the gateway, asking for a particular dataset. All this is standard procedure, done as part of the IM's functionality, with which one authenticates oneself to the platform. The next step, however, is not. The API Gateway queries Data Governance, on behalf of Semantic Interoperability, on whether the user (the consumer) has adequate access rights on the requested dataset. This query is made through a REST call where authentication is achieved by passing along the credentials of Semantic Interoperability. These are credentials created for the SI component on the Data Governance platform, which itself is a platform that manages its own users. Once Data Governance responds affirmatively, Data Processing Services can send the dataset to the consumer. So, the key to this first approach is for Data Governance to have a dedicated user account specifically for use by Semantic Interoperabilty.

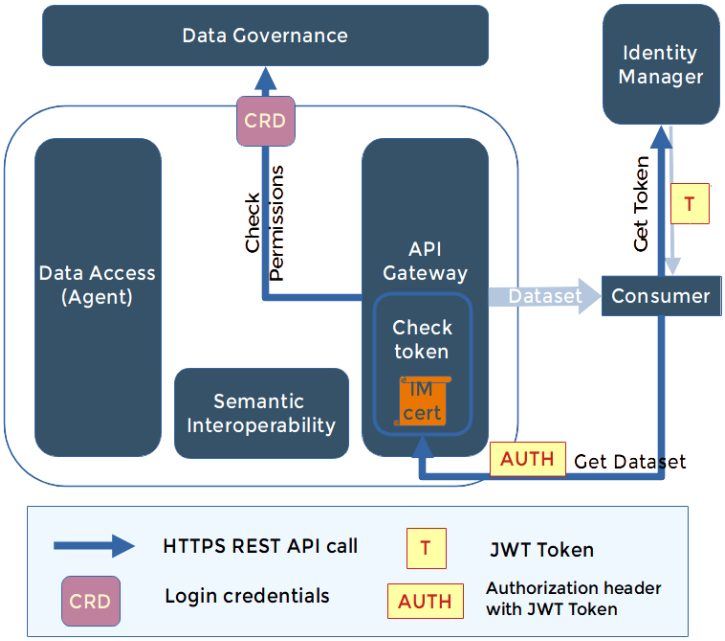


Figure 4 - Dataset obtained through Data Access (with credentials)

An alternative design for this scenario involves the API Gateway obtaining its own token from the IM, with which it authenticates its call to Data Governance. This can be seen in Figure 5.

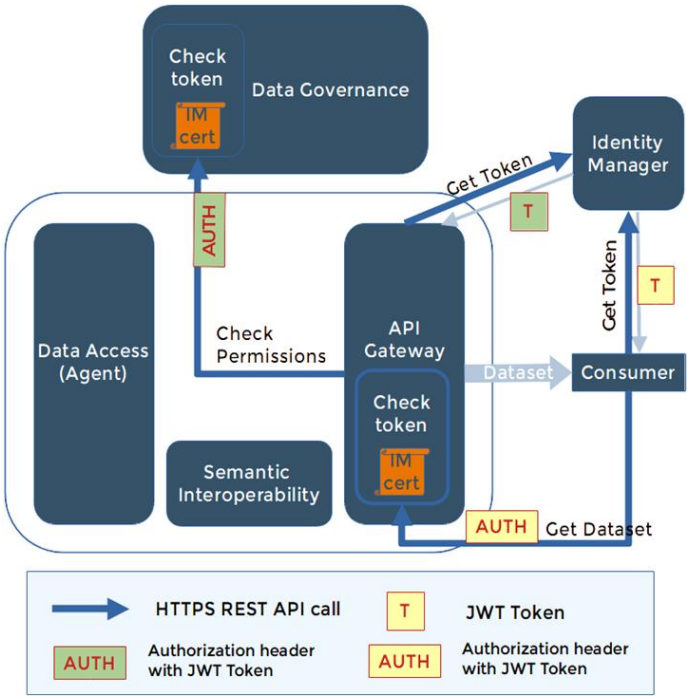


Figure 5 - Dataset obtained through Data Access (with token)

The difference for Data Governance here is that it must include the functionality of being able to verify an IM token, which in turn means that the IM certificate must always be available. This means that in order to decrypt and validate the AUTH token in DG, the certificate of the issuer (IM in this case) should be available to DG.

2.1.2.4.2 Dataset obtained directly from provider

In this case, the consumer directly calls an endpoint on the provider's infrastructure to get a dataset that they have found through searching in Data Governance. The endpoint is among the available dataset metadata.

As with the previous case, here again the consumer obtains a token from the IM to authenticate themselves to the provider. In turn, the provider uses his Data Governance credentials to authenticate their call to Data Governance (Figure 6). The idea is that the provider fills in his preference for access to the dataset once and afterwards the management of access is done automatically.

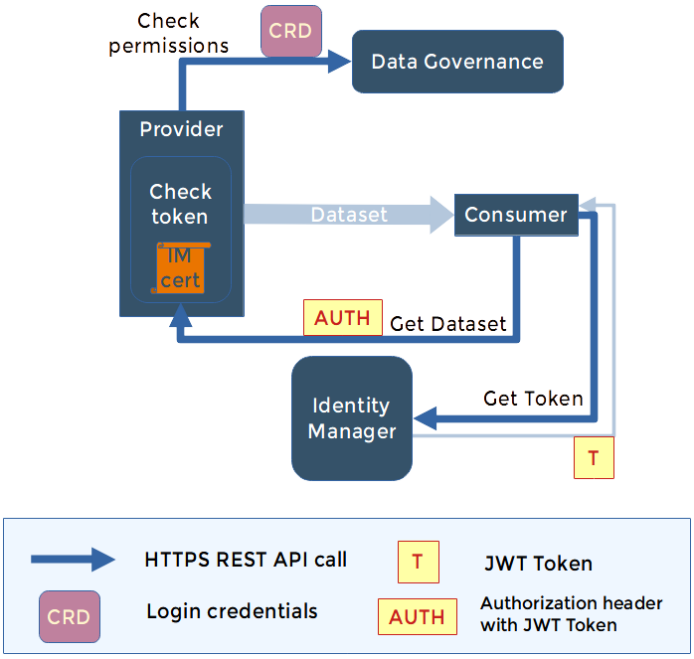


Figure 6 - Dataset obtained directly from provider

An alternative design for this scenario involves the provider obtaining a token to authenticate their call to Data Governance (Figure 7). Therefore, we again see that Data Governance needs to have the certificate of the IM available in order to verify the provider's token.

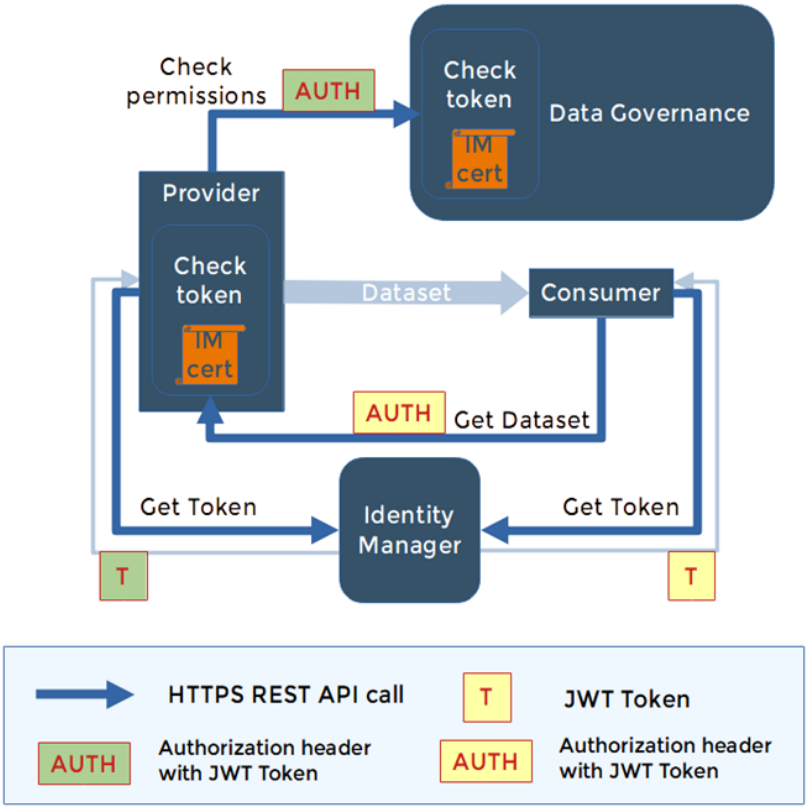


Figure 7 - Dataset obtained directly from provider

2.1.2.4.3 IM certificate

It is worth pointing that the "token" variation of the above scenarios of Data Governance use entails that Data Governance must always have a copy of the IM certificate. The obvious solution is to store this certificate in the blockchain, which is relatively easy to implement.

2.1.2.5 Data source metadata

Dataset metadata are necessary for Data Governance to be able to provide any meaningful querying capabilities. There are two sets of metadata defined:

- The *Data Governance (DG) metadata*, which are added by the provider and constitute the basis of the dataset representation in Data Governance,
- And the *Data Source (DS) metadata*. Data source metadata are produced by the Semantic Interoperability component and must be made known to Data Governance and stored together with DG metadata.

The way DS metadata are passed on to Data Governance is a combination of two approaches, by means of a REST GET request to the API Gateway, and by means of notification through a subscription service offered by the API Gateway. All interaction with the API Gateway must be authenticated with the use of IM tokens. Figure 8 shows the acquisition of DS metadata using these two ways.

- During start-up of the Data Governance component, a subscription for DS metadata events (POST request) is sent to the API Gateway. To do that, Data Governance first has to obtain a token from the IM, in order to authenticate itself to the API Gateway.
- Upon the addition of new dataset metadata in Data Governance by a provider, a GET request is sent to the API Gateway, to retrieve the DS metadata for that dataset. Similarly, here also Data Governance first has to obtain a token from the IM, in order to authenticate itself to the API Gateway.

- If the DS metadata are not available at the time of the DG metadata creation, then later, at the moment of their creation, they are sent asynchronously to Data Governance by means of the subscription mechanism, through a notification. For the notification to be acceptable to Data Governance, the API Gateway has to use one of the two mechanisms discussed earlier, i.e. Data Governance credentials or an IM token. This is shown within the dashed alternative sequence box.

Using the above mechanism, we can be assured that eventually all dataset metadata will be available for searching purposes in Data Governance.

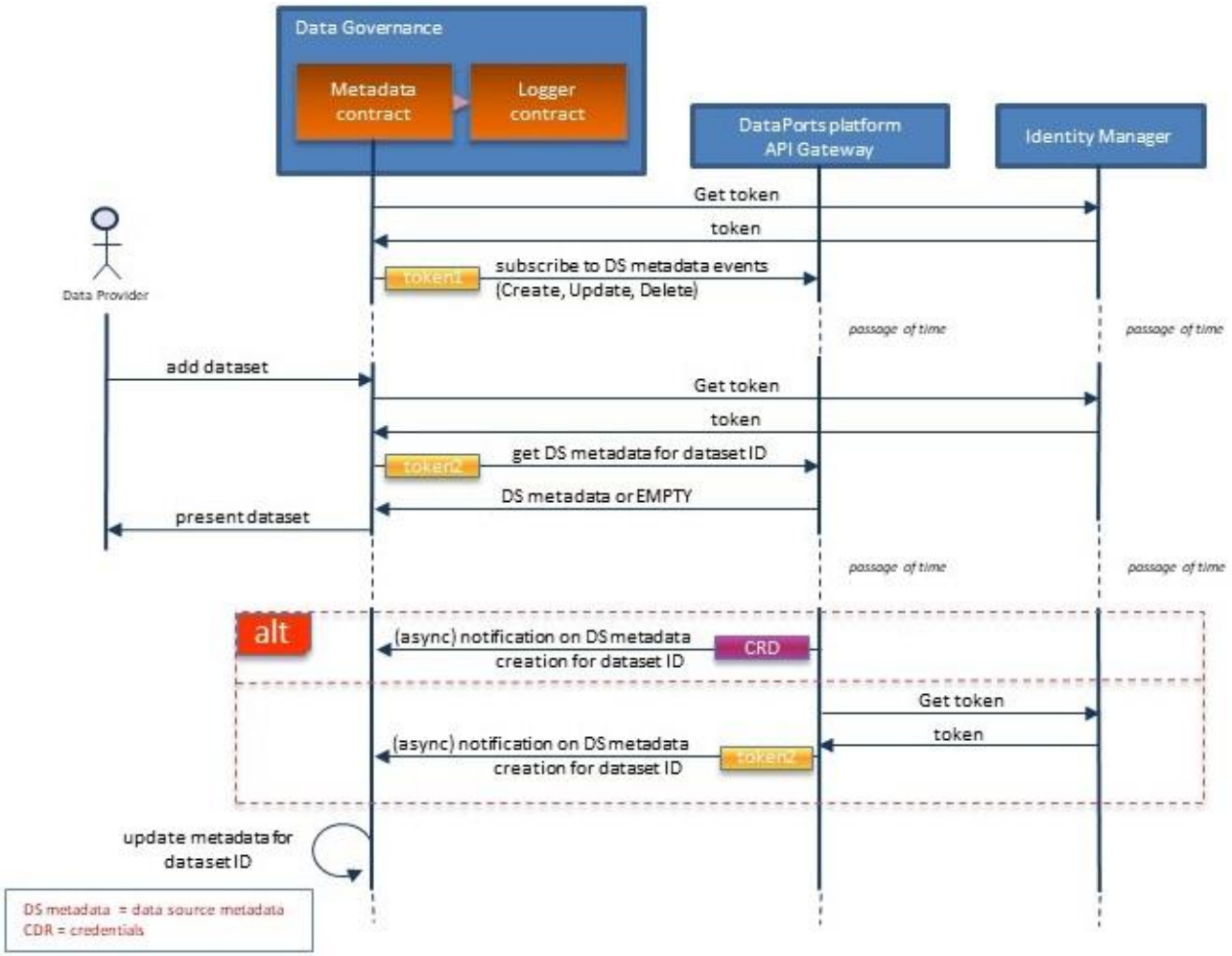


Figure 8 - Acquisition of DS metadata

The data model for DS metadata in Data Governance comprises the following data entities:

```
type DataSourceMetadata struct {
        ObjectType                  string              `json:"docType"`
        DataSourceID                string              `json:"id"`
        DataSourceType              string              `json:"dataSourceType"`
        DatasetName                 string              `json:"datasetName"`
        DataModels                  *DataModelsStruct   `json:"dataModels"`
        DataProvided                *DataProvidedStruct `json:"dataProvided"`
        Attributes                  string              `json:"attributes"`
        Service                     string              `json:"service"`
        ServicePath                 string              `json:"servicePath"`
        DataportsDataModelandFormat bool
`json:"dataportsDataModelandFormat"`
}
```

```
type DataModelsStruct struct {
        Type     string `json:"type"`
        Value    string `json:"value"`
        Metadata string `json:"metadata"`
}

type DataProvidedStruct struct {
        Type     string `json:"type"`
        Value    string `json:"value"`
        Metadata string `json:"metadata"`
}
```

2.1.2.6 Integration with Data Analytics

The Data Analytics component, and in particular the Automatic Models Training Engine, will be making use of Data Governance to ask for user authorisation and for dataset metadata that are necessary in order to correctly apply the analytics algorithms on a particular dataset. Integration with the Analytics component will be done in the same way as with Semantic Interoperability. The two possibilities are to either use specialised user credentials (Figure 9) or a token issued by the Identity Manager (Figure 10).

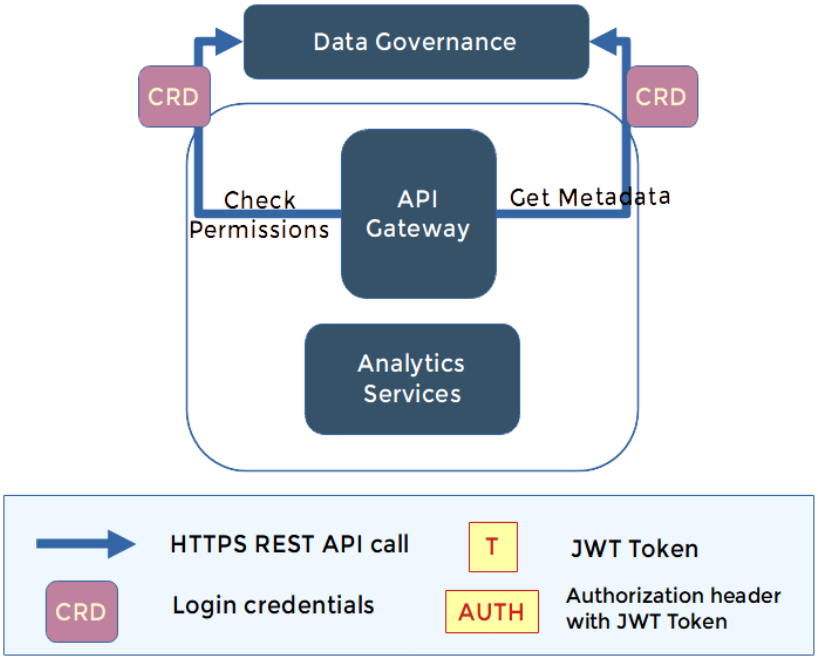


Figure 9 - Analytics services using DG with credentials

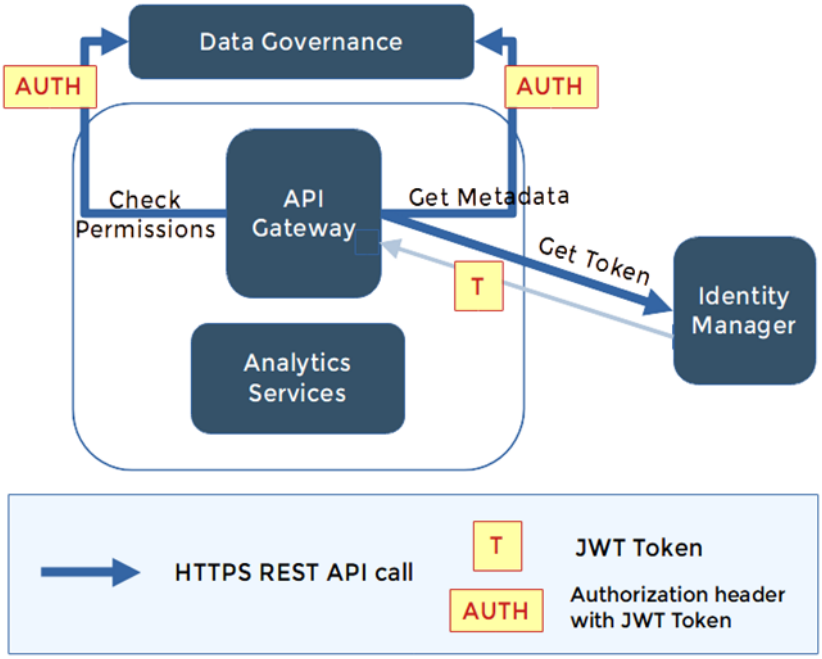


Figure 10 - Analytics services using DG with token

Figure 11 shows the interaction between the Automatic Models Training Engine and Data Governance in the more general context of the use of AI services within the platform. The relevant interactions are highlighted with a red rectangle.

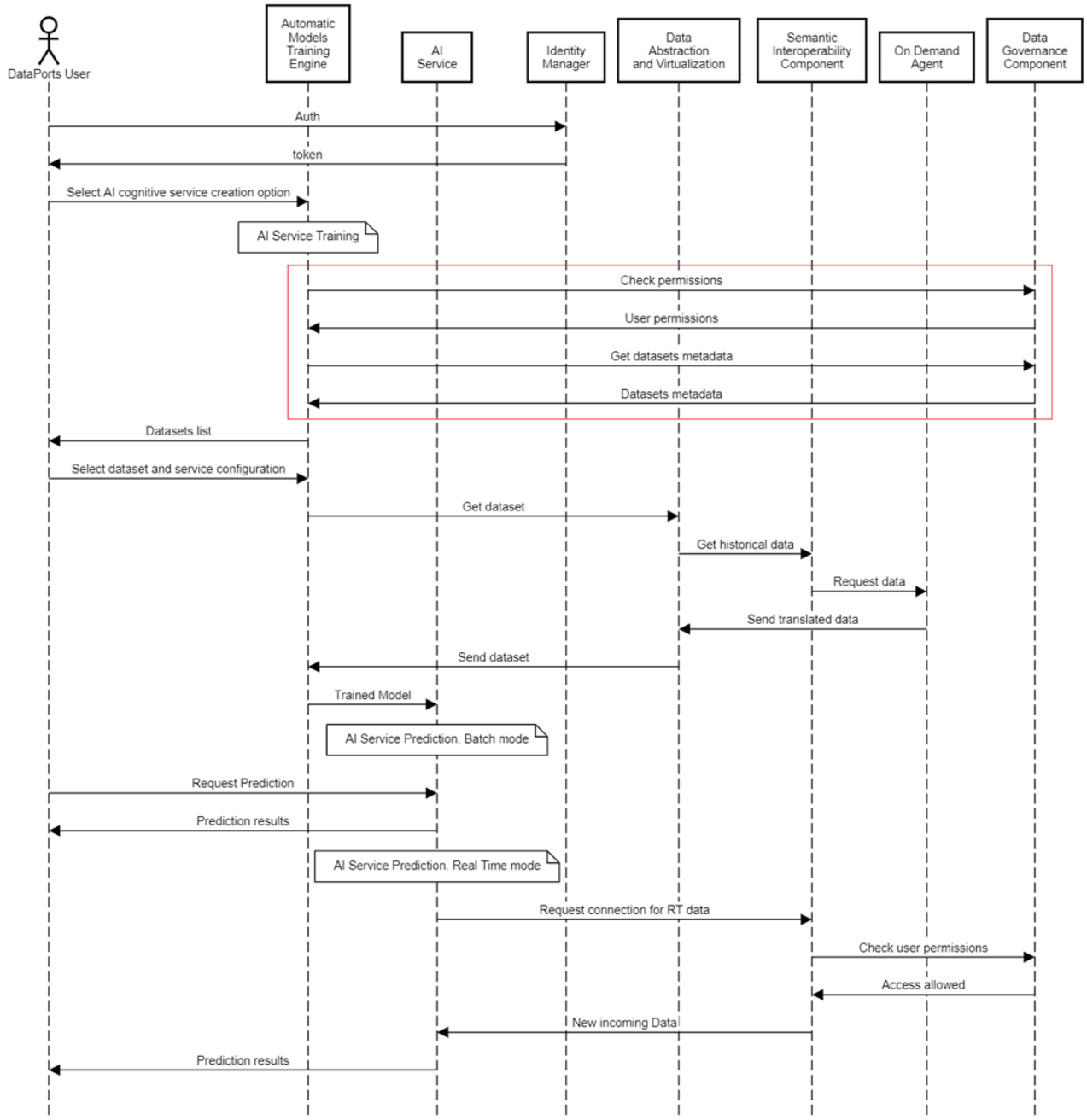


Figure 11 - Use of Data Governance by the Automatic Models Training Engine

2.2 ORGANIZATIONAL AND PUBLIC ACCESS

2.2.1 Requirements

The requirements, the user stories, and the functionalities described in D3.4 for the user-level access are enhanced with two more access levels. The first one is the organizational-level access, while the second one is the public-level access. To grant organizational-level access (or simply organizational access) to a dataset means that all users of an organization, which is participating in the Data Governance blockchain network, have the specified access to this dataset. For example, granting organization ABC read access to dataset A means that all users of ABC have read access to dataset A. On the other hand, to grant public-level access (or simply public access) to a dataset means that all Data Governance users, irrespective of organization, will have the specified access to the dataset.

Moreover, some more functionalities have been added. Below are described the requirements for a user that acts as a data provider or as a data consumer.

As a data provider, one must be able to:

- Accept/deny organizational access requests
- For each dataset, view the organizations that have been granted organizational access
- Revoke access on organizational level
- Assign public access to a dataset with certain permissions (read, write, share)
- Revoke public access to a dataset
- For each dataset, view whether public access has been granted and with what permissions

As a data consumer, one must be able to:

- (only for *organization admins*) Create request for organizational access to a dataset
- (only for *organization admins*) Create request for the revocation of organizational access to a dataset
- View list of datasets to which they have public access
- View list of datasets to which they have organizational access

2.2.2 Design

The Data Governance chaincode has been extended to manage organizational and public access. In particular, new functions have been added to the chaincode that query dataset information regarding its organizational and public access. We have also created the corresponding functions on the Hyperledger Fabric Software Development Kit (SDK) and have exported them on the Node.js server. Finally, the functionalities regarding organizational and public access have already been partially integrated in the Data Governance frontend.

This section shows the functionalities from the perspective of the user, be it a data provider or a data consumer. Figure 12 depicts a user viewing the external datasets menu, i.e., accessing the set of datasets to which the user is a consumer, and more specifically viewing the *Public* tab which lists the available public access datasets. Figure 13 depicts a data provider viewing the organizational permissions granted for a dataset, listing all organizations that have organizational access to the dataset. Moreover, the data provider is able to revoke the specific permission by clicking the *Revoke* button.

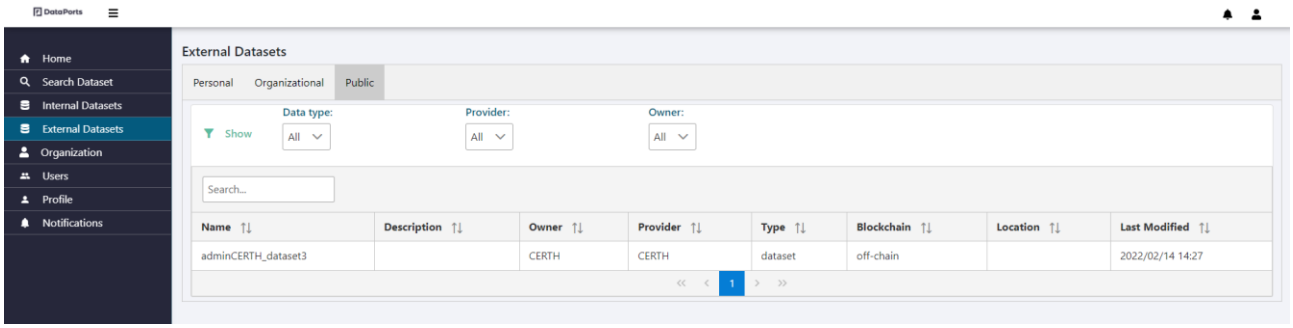


Figure 12 - Granted access view for a data consumer

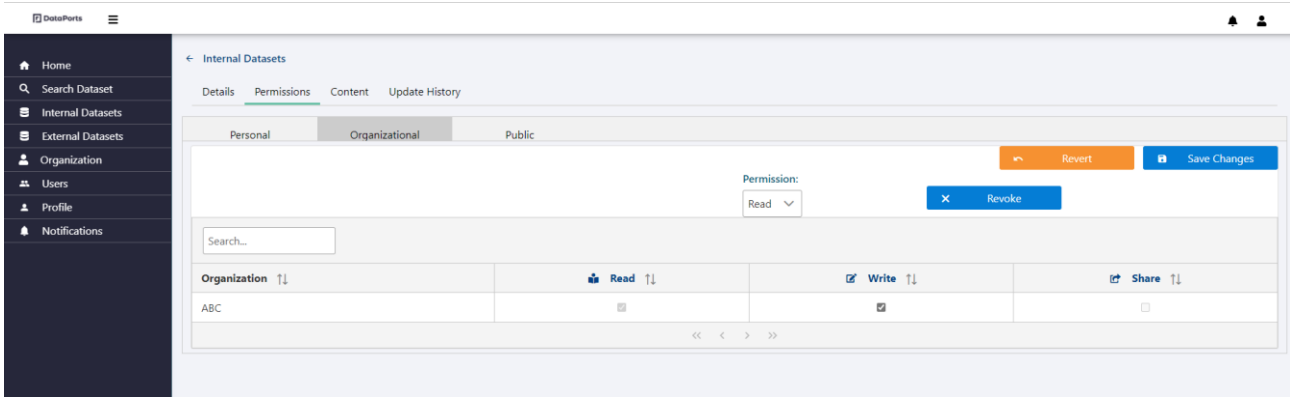


Figure 13 - Granted access view for a data provider

Finally, below we provide the data models for the organizational and public access as they appear in the Data Governance chaincode.

Data model for an organizational access request

```
type RequestAccessByOrg struct {
        ObjectType       string `json:"docType"`
        RequestID        string `json:"requestid"`
        Dataset_Name     string `json:"dataset_name"`
        Dataset_Provider string `json:"dataset_provider"`
        Inicator         string `json:"indicator"`
        Endpoint         string `json:"endpoint"`
        Permission       string `json:"permission"`
        Organization     string `json:"organization"`
        DateTime         string `json:"dateTime"`
}
```

Data model for a request to revoke organizational access

```
type RequestRevokeAccessByOrg struct {
        ObjectType       string `json:"docType"`
        RequestID        string `json:"requestid"`
        Dataset_Name     string `json:"dataset_name"`
        Dataset_Provider string `json:"dataset_provider"`
        Permission   string `json:"permission"`
        Organization string `json:"organization"`
        DateTime     string `json:"dateTime"`
}
```

Data model for a particular case of organizational access

```
type AuthorizedOrgs struct {
        ObjectType       string `json:"docType"`
        Dataset_Name     string `json:"dataset_name"`
        Dataset_Provider string `json:"dataset_provider"`
        Permission     []string `json:"permission"`
        Organization     string `json:"organization"`
}
```

Data model for the public access of a particular dataset

```
type PublicDataset struct {
        ObjectType       string `json:"docType"`
        Dataset_Name     string `json:"dataset_name"`
```

```
        Dataset_Provider string `json:"dataset_provider"`
        Permission     []string `json:"permission"`
}
```

2.3 LOGGING FOR ACCESS REQUESTS, ACCESS REVOCATIONS, AND REQUESTS TO REVOKE ACCESS

2.3.1 Requirements

In D3.4 we had specified that "the clearing house, whose function is to allow provenance and audit on all dataset access requests/granted permissions is implemented using inherent built in blockchain functionality". However, it became apparent with time that we could have more flexibility in our implementation of a clearing house, if we did the implementation based on logger entities managed purely by the Logger smart contract. That way we can log the exact information we need, in the exact format we need, without trying to piece together a collage of information taken from the blockchain. For the same reason, this way we can have better querying capability. Therefore, we implement the clearing house on the smart contract level.

Moreover, the general requirements have been that *we want to be able to retrieve the history of transactions*, so that *we are able to track the information flow*. To that end, we have expanded our logging capability to cover, for example, requests for access to datasets, both for user and organizational level access, and for the revocation of access on all three access levels (user, organizational, public).

The purpose of the *logger* chaincode is to store the history of the transactions regarding the created/updated metadata of data, the access requests for the user/organizational/public access, and the revoked access for each metadata record of data. The logger chaincode design regards a series of cross-chain communications. Each time the state is going to be created/updated regarding the metadata of data or the permissions then the logger chaincode is invoked. Moreover, the backend implementation has been completed but it is still in progress of integration with the Data Governance frontend. In summary, the Data Governance functionalities are:

- History of updates for Data Governance metadata of data
- History of updates to Data Source metadata of data
- History of access requests (user/organizational/public level)
- History of revoked access (user/organizational/public level)

The data models for the history on the Data Governance chaincodes have the following structure:
Data model for the history of user-level revoked access

```
type LoggerRevokedAccess struct {
        ObjectType         string `json:"docType"`
        Dataset            string `json:"dataset"`
        UsernameOfProvider string `json:"usernameOfProvider"`
        UsernameOfConsumer string `json:"usernameOfConsumer"`
        Organization       string `json:"organization"`
        Timestamp          string `json:"timestamp"`
        RevokedPermission  string `json:"revokedPermission"`
        Count              int    `json:"count"`
}
```

Data model for the history of organizational-level revoked access

```
type LoggerRevokedAccessOrgs struct {
        ObjectType           string `json:"docType"`
        Dataset              string `json:"dataset"`
        UsernameOfProvider   string `json:"usernameOfProvider"`
```

```
        OrganizationConsumer string `json:"organizationConsumer"`
        Timestamp            string `json:"timestamp"`
        RevokedPermission    string `json:"revokedPermission"`
        Count                int    `json:"count"`
}
```

Data model for the history of public-level revoked access:

```
type LoggerRevokedAccessPublic struct {
        ObjectType         string `json:"docType"`
        Dataset            string `json:"dataset"`
        UsernameOfProvider string `json:"usernameOfProvider"`
        Timestamp          string `json:"timestamp"`
        RevokedPermission  string `json:"revokedPermission"`
        Count              int    `json:"count"`
}
```

Data model for the history of user-level access requests

```
type LoggerAccess struct {
        ObjectType         string `json:"docType"`
        Dataset            string `json:"dataset"`
        UsernameOfProvider string `json:"usernameOfProvider"`
        UsernameOfConsumer string `json:"usernameOfConsumer"`
        Organization       string `json:"organization"`
        Timestamp          string `json:"timestamp"`
        Status             string `json:"status"`
        RequestID          string `json:"requestID"`
        Count              int    `json:"count"`
}
```

Data model for the history of organizational-level access requests

```
type LoggerAccessOrgs struct {
        ObjectType           string `json:"docType"`
        Dataset              string `json:"dataset"`
        UsernameOfProvider   string `json:"usernameOfProvider"`
        OrganizationConsumer string `json:"organizationConsumer"`
        Timestamp            string `json:"timestamp"`
        Status               string `json:"status"`
        RequestID            string `json:"requestID"`
        Count                int    `json:"count"`
}
```

Data model for the history of public access

```
type LoggerAccessPublic struct {
        ObjectType         string `json:"docType"`
        Dataset            string `json:"dataset"`
        UsernameOfProvider string `json:"usernameOfProvider"`
        Permission         string `json:"permission"`
        Timestamp          string `json:"timestamp"`
        Count              int    `json:"count"`
}
```

Data model for the history of updates of a dataset's data source metadata

```
type DataSourceMetadata struct {
        ObjectType         string   `json:"docType"`
        DataSourceMetadata string   `json:"dataSourceMetadata"`
        Status             string   `json:"status"`
```

```
        Changes             []string `json:"changes"`
        Timestamp           string   `json:"timestamp"`
}
```

2.4 USER LEVEL ACCESS ADDITIONS: REVOKE ACCESS

2.4.1 Requirements

Some more functionalities have been added that were not part of previous deliverables. We realized that a user, having user-level access, or an organization having organizational access to a dataset, may no longer be interested in retaining that access but may want to cancel it. Therefore, we added new functionality on both access levels, in which the data consumer can create a request in order to downgrade the access rights. In the case of organizational access, only an organization's admin can request the revocation of access to a dataset.

2.4.2 Design

The data consumer (on the user level - the user and on the organizational level - the organization admin) can create a revocation request for a dataset to which they already have access to. The effect of this request is immediate, i.e., revocation happens immediately. The solution regarding the grant/revoke permission on the user level is accomplished through a series of transactions on the Data Governance chaincodes. The backend functionality has already been developed while the frontend is still being integrated in the DG frontend.

Data model for user-level revoke access requests

```
type RequestRevokeAccessByUser struct {
        ObjectType       string `json:"docType"`
        RequestID        string `json:"requestid"`
        Dataset_Name     string `json:"dataset_name"`
        Dataset_Provider string `json:"dataset_provider"`
        Permission   string `json:"permission"`
        Name         string `json:"name"`
        Surname      string `json:"surname"`
        Organization string `json:"organization"`
        Username     string `json:"username"`
        Email        string `json:"email"`
        Role         string `json:"role"`
        DateTime     string `json:"dateTime"`
}
```

Data model for organizational-level revoke access requests

```
type RequestRevokeAccessByOrg struct {
        ObjectType       string `json:"docType"`
        RequestID        string `json:"requestid"`
        Dataset_Name     string `json:"dataset_name"`
        Dataset_Provider string `json:"dataset_provider"`
        Permission   string `json:"permission"`
        Organization string `json:"organization"`
        DateTime     string `json:"dateTime"`
}
```

2.5 REGISTER AND LOGIN ADDITIONS/UPDATES

2.5.1 Requirements

The register and login processes are crucial to the security of Data Governance. The registration on the Data Governance platform is almost identical to the previous version, but now the initial password is not given by the admin but is automatically randomly generated. In this way, we ensure that the initial password is strong because it always contains uppercase letters, lowercase letters, numbers, and symbols. Moreover, we added the *deactivate user* functionality, available both to organization admins as well as the network admin. Thus, both types of administrators can deactivate a user, if there is evidence of that user being malicious or having his/her account hijacked.

2.5.2 Design

The design of the registration and the login processes are still the same as described in D4.2. Figure 14 depicts an organization admin that registers a new user. The addition of the Data Governance MVP v2 in contrast to v1 is that the organization admin does not create the password but is generated automatically. Moreover, the password is known to the organization admin through a pop-up window when the registration has been completed.

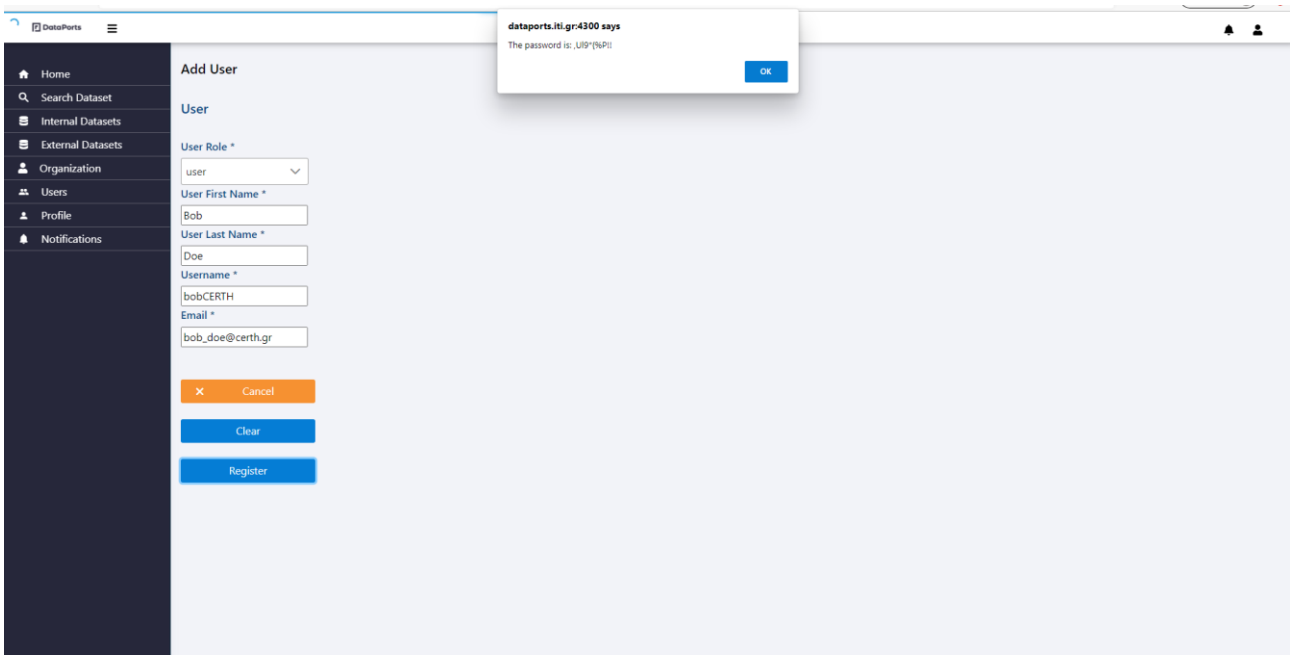


Figure 14 - Registration form

Another added value in Data Governance MVP v2 regarding users, is the ability to deactivate a user's account. Administrators can revoke a user's access to the system by deactivating their account, but they are also able to later reactivate the account. More specifically, the caller must have `hf.Registrar` blockchain native authority. Essentially, the Hyperledger Fabric account is marked as *Inactive* on the chaincode level. Figure 15 depicts the list of active users while Figure 16 depicts the list of inactive users.

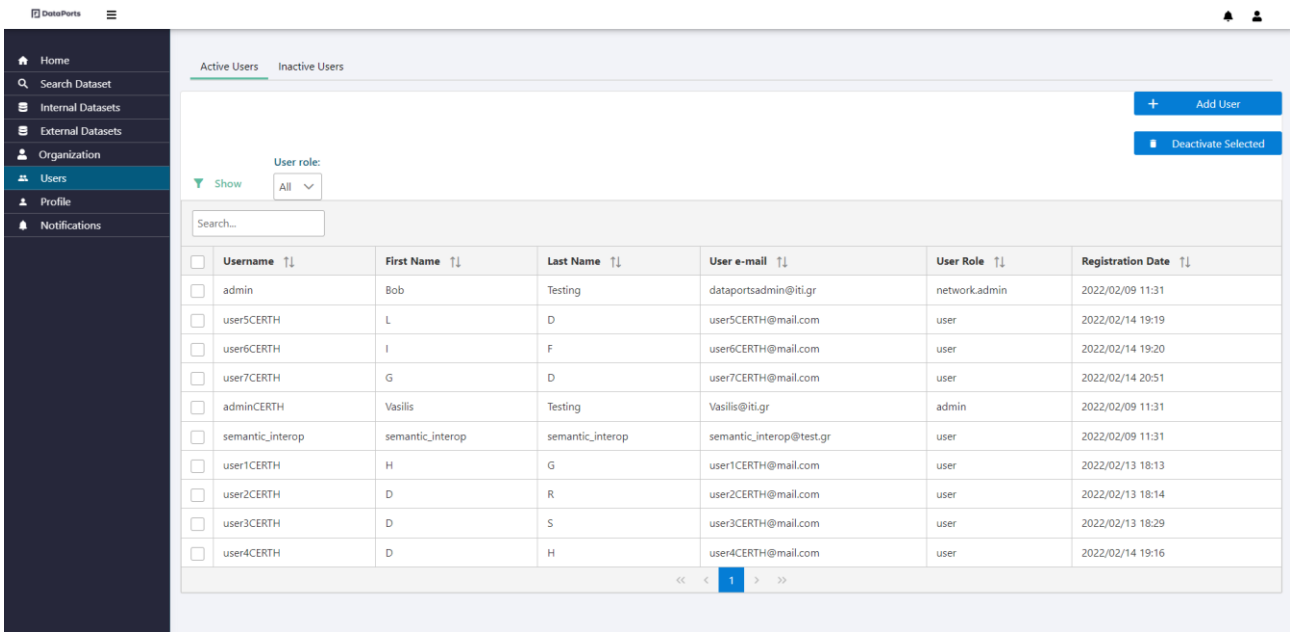


Figure 15 - Active users

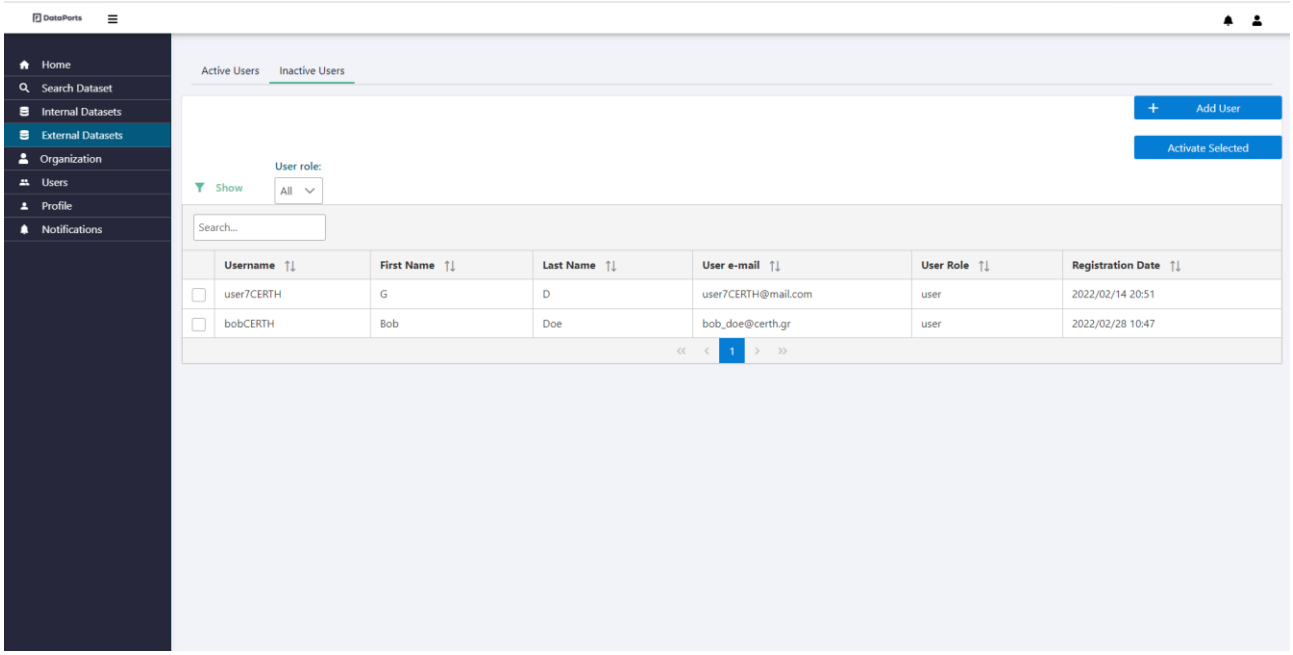


Figure 16 - Inactive users

2.6 DEFINITION OF THE ROLES: DATA PROVIDER AND DATA OWNER

2.6.1 Requirements

As stated before, the key roles in Data Governance are the data provider, the data owner, and the data consumer. It is therefore important, that the rights, obligations, and prohibitions that are in line with each role are addressed. The role of the data consumer remains the same as described in the previous deliverables. In the context of the Data Governance platform, the data provider is the one that manages the metadata and grants/revokes access to the data consumers on all three access levels. The data owner is either the provider itself, or the one that has given the data to the data provider to manage it on its behalf within the Data Governance platform. In the latter case, the owner does not have an active role within Data Governance.

2.6.2 Design

A data provider is a registered and enrolled user and owns Fabric credentials. As a result, the data provider is part of a participant organization on the Fabric network. On the other hand, the data owner is now considered as being strictly an organization and not a user. The data owner can be either a blockchain participant organization, or not. In addition, the data provider is the sole manager of the metadata and the permissions to datasets. If the data owner organization is a participant member of the Data Governance platform, its administrators can view the metadata of the datasets of which his/her organization is owner. The data owner endpoints for this functionality are described in Table 3. The data owner endpoints have yet to be integrated with the Data Governance frontend.

#	Endpoint	Description
1	`queryMetadataByOrganizationOwner`	The organization admin can query all the uploaded metadata in which his/her organization is the data owner
2	`queryMetadataByOrganizationOwnerAndDatasetName`	For a user/admin whom the organization is the data owner. Can create a query request based on the metadata's name

Table 3 - Data owner endpoints

3 DATA SHARING BLOCKCHAIN NETWORK - VERIFIED GROSS MASS USE CASE

This section describes the design of the feature additions and the extensions to the Verified Gross Mass (VGM) blockchain (BC) application in the port of Valencia as devised in D3.4 and D4.2 deliverables. Specifically, in the summary and next steps sections of D3.4 and D4.2 we have outlined a list of extensions for the VGM use case and Minimal Viable Product (MVP) developed. These will be part of an updated version of the MVP, i.e., MVP v2 and are presented in Table 4 below along with their status. The latter can be: "done" (implemented at the time of writing this document); "in progress" (to be finalized by the submission of D4.4); and "dropped", meaning the specific feature won't be part of the VGM MVP v2. Some of the entries are the natural progression of the initial MVP, such as deployment on cloud environment. Others are new features and extensions we had in mind for version 2 of the MVP. Additionally, some of the extensions appearing in the table were not initially planned and therefore not included in the previous deliverables, however the need for such came up during development of the other features. Those are added at the end of the table and denoted with an (*).

#	Feature/activity	Status	Comments
1	Integration with Valencia port systems, including events and notifications, e.g., with the Port Community System	Done	PCS, Redsys
2	Addition of invoices and payments	In progress	In process of payments implementation
3	Usage of private collections for the sake of confidentiality requirements	Done	
4	Application of MongoDB caching in the client	Done	
5	MongoDB-based wallet for BC client	Done	
6	Additions of queries, such as, company history (which also derives extension to the data schema)	Dropped	The queries are done over MongoDB for better performance
7	Updates of data model and restructuring of chaincodes to be more generic and easily extendable for other "assets" (*)	Done	
8	Configurable BC client, extensions to conPESO user interface (UI) to provide configuration options (*)	Done	

Table 4 - VGM MVP version 2 new features and their status

In the following sections we address each of the features and extensions and explain the reason for including the feature in V2 of the MVP, the requirements for the feature and the design of the feature.

3.1 INTEGRATION WITH VALENCIA PORT SYSTEMS

3.1.1 Requirements

Figure 17 describes the stakeholders and their roles in the VGM workflow. As can be seen from the figure, one of the stakeholders in the BC network is the port's Port Community System (PCS) system. In the flow it serves both as a data provider (using conPESO RESTful APIs to submit container transport data) and as a data consumer. As such, PCS needs to be notified on VGM certificate creation in order to communicate the

certificate, without which an entrance of the container into the terminal is prohibited. The notification on VGM process completion is then transferred to VERMAS (refer to D3.4) format and communicated to the relevant systems in the port (such as port container terminal) which are integrated with the PCS. Integration with Valencia's port operational systems means first of all the integration of the PCS system to the conPESO platform which serves as the frontend of the VGM BC solution and as a BC client gateway.

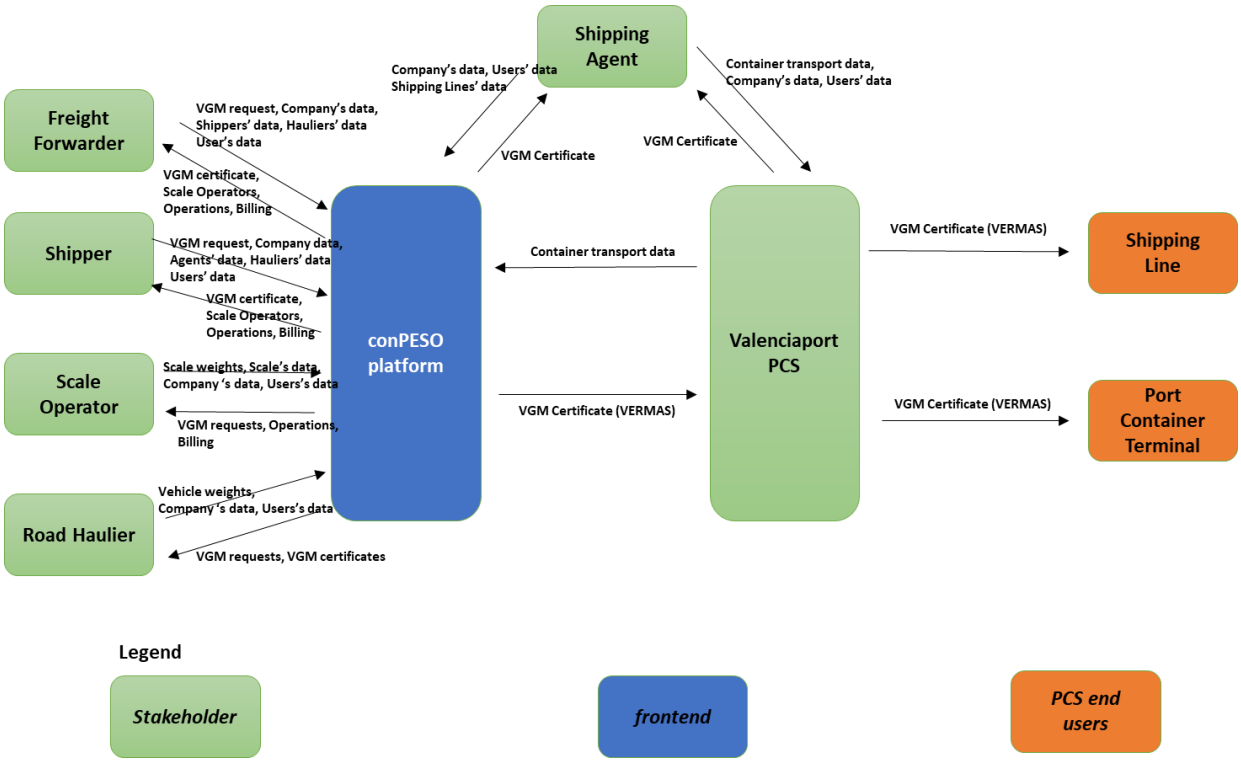


Figure 17 - Data providers and data consumers in the VGM use case (source: D3.4)

3.1.2 Design

Hyperledger Fabric (simply Fabric) supports the ability to raise events during the invocation of a chaincode transaction. Events are named objects that are recorded as part of the transaction execution and eventually bubbled up to the application clients connected to the same channel for their perusal once the transaction is committed.

Events are only raised at the transaction commitment time (therefore are raised only upon successful completion and commitment of the transaction) and are piggybacked to the peer at the end of the transaction simulation execution. When the BC client receives a successful transaction response signed by the peers, the event raised by the transaction is part of this response payload. The client can register a listener function, which will be invoked for execution when an event of a particular type will be raised.

The whole process is described in Figure 18. When a ledger state is changed and a client is interested in the notification of the change, the smart contract (a.k.a chaincode) emits a chaincode event, which is forwarded to the SDK and can be captured and acted upon by event listener function.

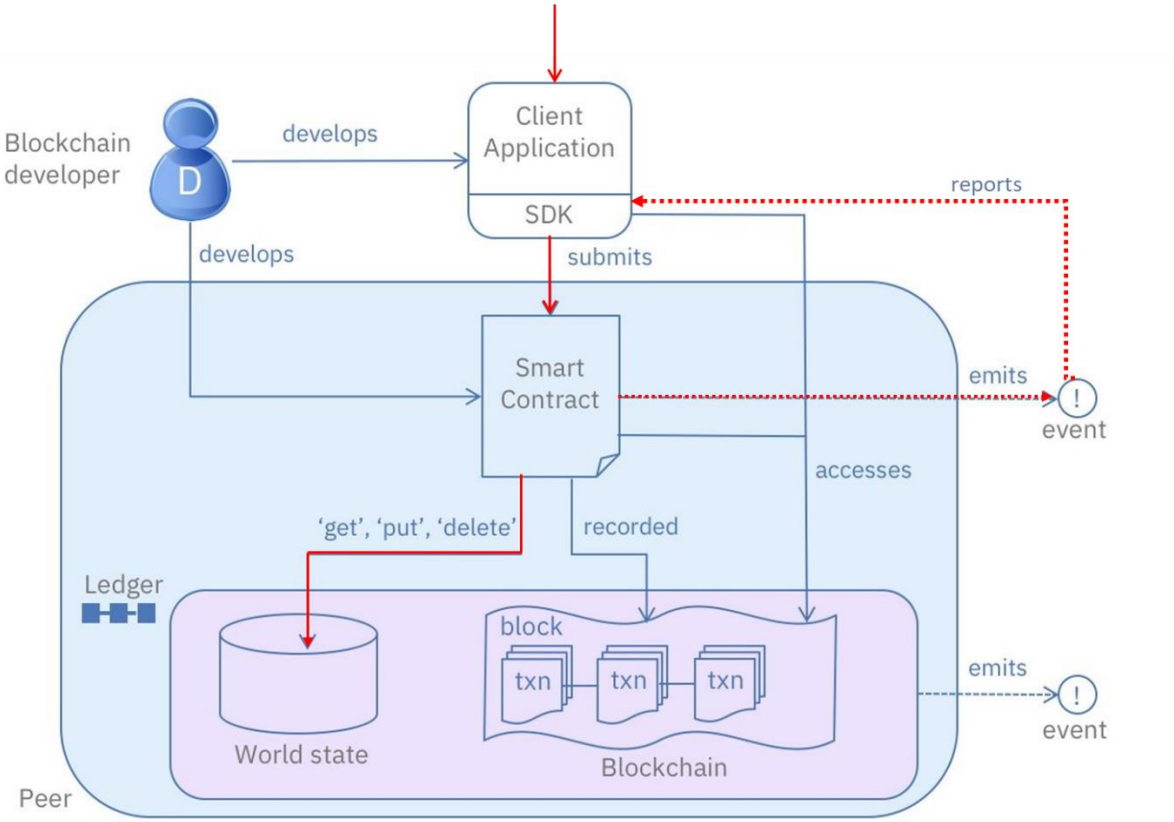


Figure 18 - Chaincode event emission on ledger state change (source "Tutorial chaincode event listener on Hyperledger fabric")

We are using this mechanism of chaincode events to notify conPESO, which envelops a BC client and defines listener for events on a completion of the process for VGM request.

On a state change of the asset we are interested in (inside creation, update, and deletion functions in the chaincode), the chaincode registers an appropriate event with the relevant payload (Table 5):

Event payload Represent an asset state change event payload		
Field	**Type**	**Description**
`ID`	String	Event identifier
`eventType`	String	Type of the event (indicates the type of state change – 'registered', 'changed', 'removed')
`performedTime`	Date	Event creation time

Table 5 - Event payload definition

As described above, this event is emitted by the chaincode by using shim function `SetEvent()`.

In conPESO client application, an event listener function is registered by using the Fabric client SDK `contract.addContractListener()` method. When the function is invoked upon event reception, the event payload is parsed, and according to the type of the event further actions are taken.

In our backend, the weighing request, which is stored in the database in JSON format according to our internal data model, is transformed into XML format, and sent to the PCS SOAP service.

Next, the PCS transforms our XML message into Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT), and sends it to the carriers.

In the frontend configuration view (Figure 19), we must indicate the access credentials, and the URLs in which the SOAP web services are available to log in to the PCS, to obtain information from it, and to provide the VERMAS in XML format:

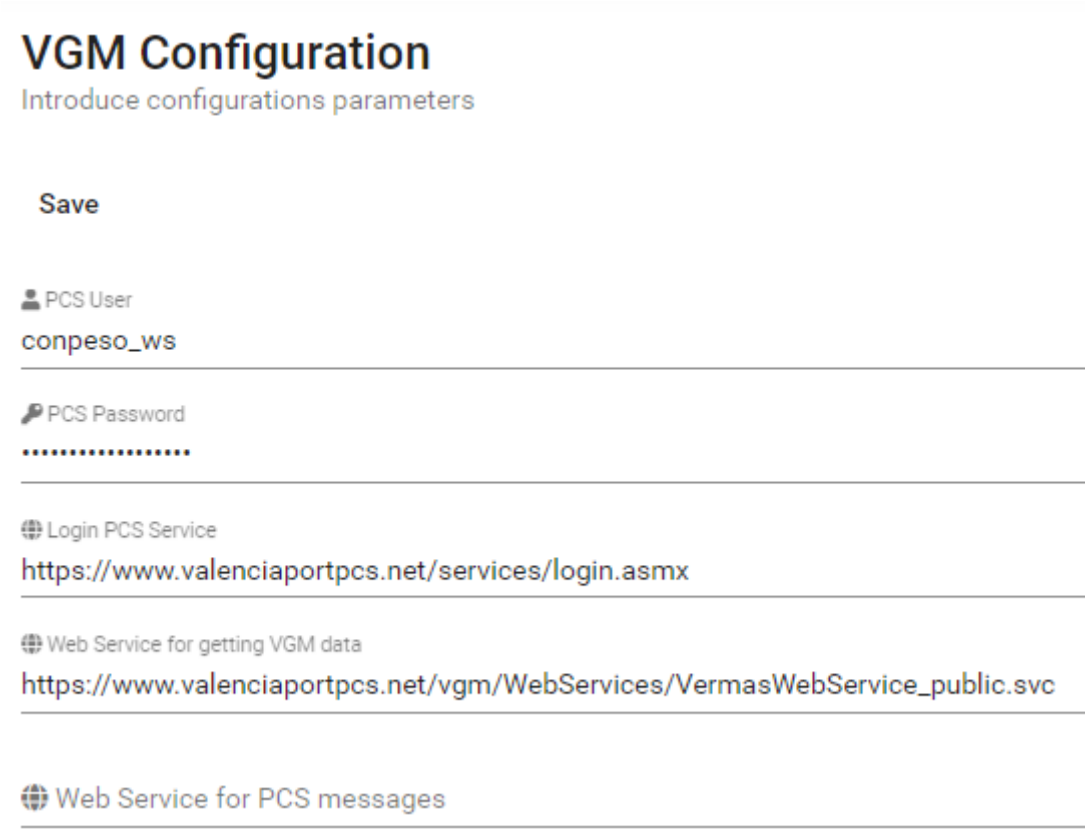


Figure 19 - VGM configuration view

3.2 INVOICES AND PAYMENTS

3.2.1 Requirements

The requirements related to payments and invoices were mentioned in the system functionalities defined in D3.4 and were delegated to V2 of the MVP. We copy below the relevant requirements per role.

Freight forwarder/shipper

- View weighting invoices and operations
- Generating payments for weighting services

Scale operators

- Register weighting invoices, viewing operations
- Receive payments

As can be seen from the defined functionalities, a way to store and view weighting invoices, and integration with payment services is required both for weighting service provider (scale operator) and for the service consumer (shipper or his representative).

The required functionality encompasses the ability to register and read weighting invoices and perform and receive payments for the weighting services.

3.2.2 Design

The solution includes a method to register the credit of the customers through credit card payments and the expenses incurred by these users for the services of weighing the containers by the scale operators and the service fees collected by the solution. These movements can be filtered and ordered by different parameters and the results can be exported to an Excel file. On a monthly basis, the expenses and fees paid by the customers are communicated to the ERP system for accounting and invoices. To this end, the solution generates an XML movement file for the month which is transferred to the ERP. With this information, the ERP system registers the movements into the account system of the company, and it generates a set of invoices in PDF format. The latter is archived in a ZIP compressed file together with an XML document containing the information to be registered about the invoices together with the PDF file. This compressed archive is processed by the solution and the invoices are registered for the users check and download. The set of invoices generated includes the invoices of the scale operators for the weighing services provided during the month and the invoices of the solution provider according to the fees paid for the use of the solution (Figure 20).

The chaincode has been extended to register the credit and payment records in the Distributed Ledger Technology (DLT) and to register the information of the invoices together with the hash of the PDF invoice. The PDF itself is not stored in the DLT.

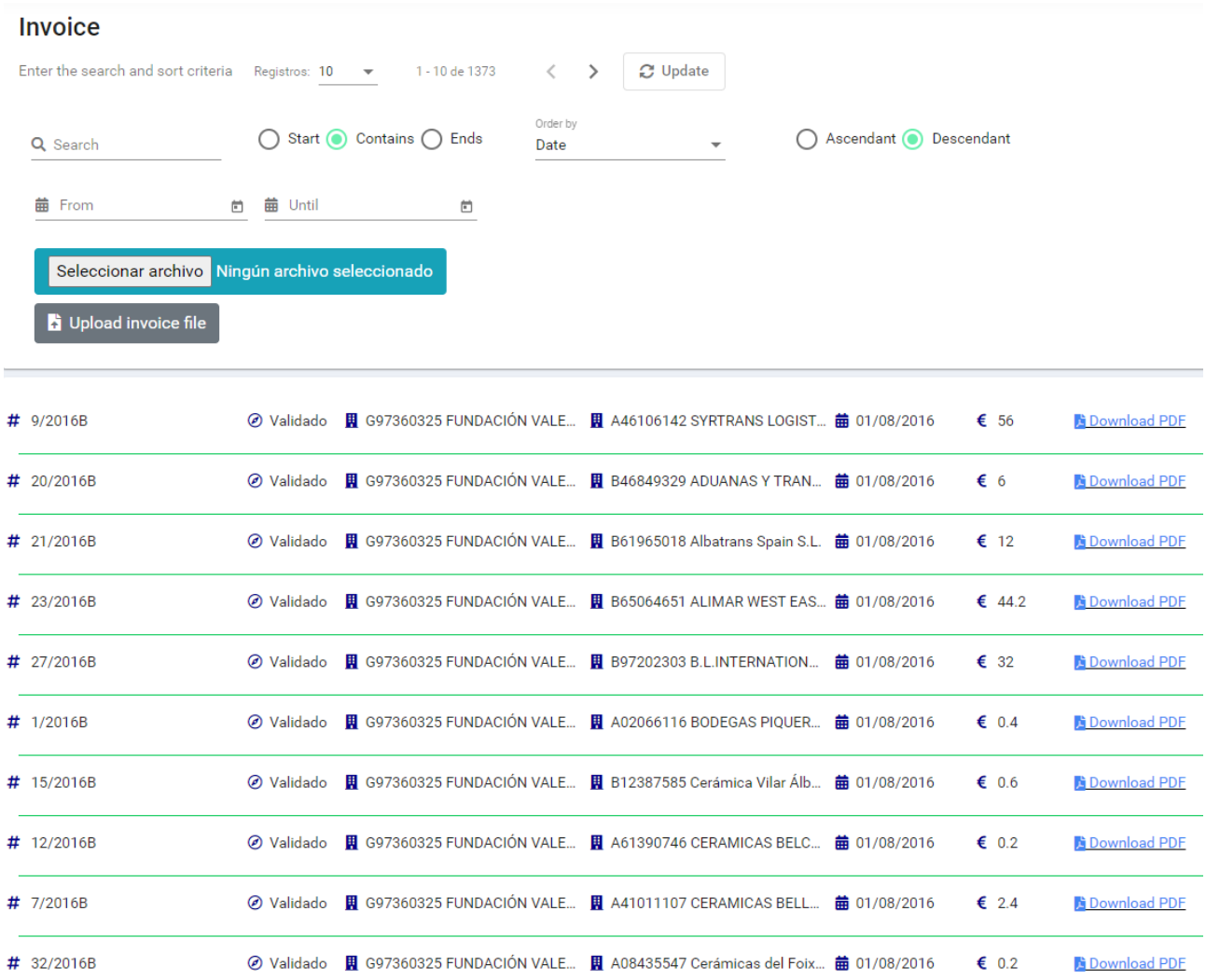


Figure 20 - Operations on invoices screenshot

The data model for the charges where we store the movements in the database and the DLT has the following structure:

```
export class Charges extends TrackableEntity {

    public chargesID?: string;
    public from?: Party;
    public to?: Party;
    public issueDate?: Date;
    public dueDate?: Date;
    public subject?: string;
    public references?: Reference[];
    public exchange?: ExchangeRate;
    public charges?: Charge[];
    public subtotal?: Amount;
    public taxes?: Charge[];
    public total?: Amount;
    public payed?: Amount;
    public pending?: Amount;
    public paymentMethod?: PaymentMethod;
    public bankAccount?: string;
    public bank?: string;
    public invoiceNumber?: string;
    public payload?: any;
}
```

The data model for the documents where we store the invoices in the database and in the DLT has the following structure:

```
class Document extends TrackableEntity {

    public documentID?: string;
    public documentType?: DocumentType;
    public filename?: string;
    public date?: Date;
    public format?: FormatType;
    public sender?: Party;
    public receiver?: Party;
    public origin?: Location;
    public destination?: Location;
    public attributes?: Attribute[];
    public hash?: string;
    public fileId?: string;
}
```

Payments use the Redsys[3] credit card gateway offered by the banks for conducting electronic payments. Redsys credit card payment system applies a redirection mechanism for conducting the payments. It consists of directly loading the Redsys card data entry page once the user has indicated the amount to be recharged. This redirection mode has the advantage that it is simple and very secure, since Redsys is in charge of managing the confidential data of the user's bank card through its own https page. Redsys offers other connection mechanisms such as web services, but in this case the solution requires to comply with the Payment Card Industry Data Security Standard (PCI-DSS) security standard (Figure 21).

[3] http://www.redsys.es/

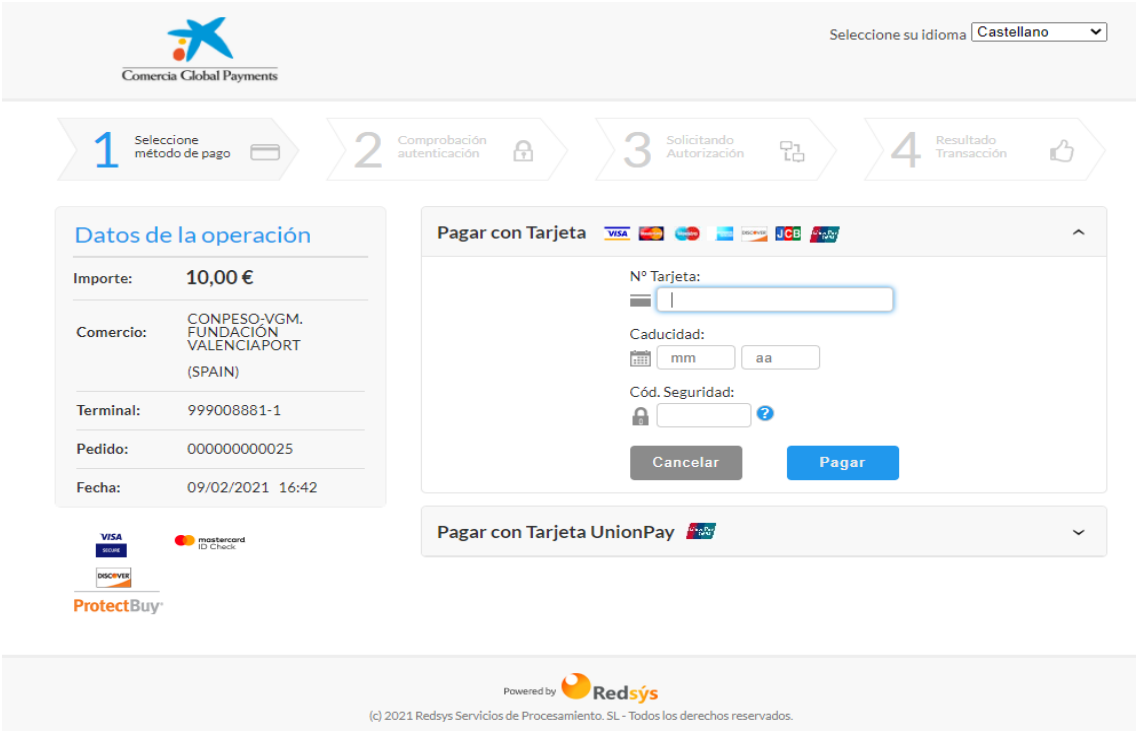


Figure 21 - Credit card payment via Redys

When the purchase process is finished, Redsys notifies the result back to the solution (OK or failure) by means of a page redirection, being able to indicate a different page for each of the two cases. In addition, we have opted for a SOAP synchronization, which implies indicating to Redsys also the URL of a SOAP web service, so that Redsys notifies that web service through a POST message before the transaction is completed, and then if the solution responds with an OK, the transaction becomes effective ends, while if we respond with KO the transaction is rejected. Thanks to this SOAP synchronization, we avoid the risk that the user closes the browser once the transaction in Redsys is finished, and consequently the redirection to the OK or failure page of the solution will not take place, and it will not know the result of the transaction (Figure 22).

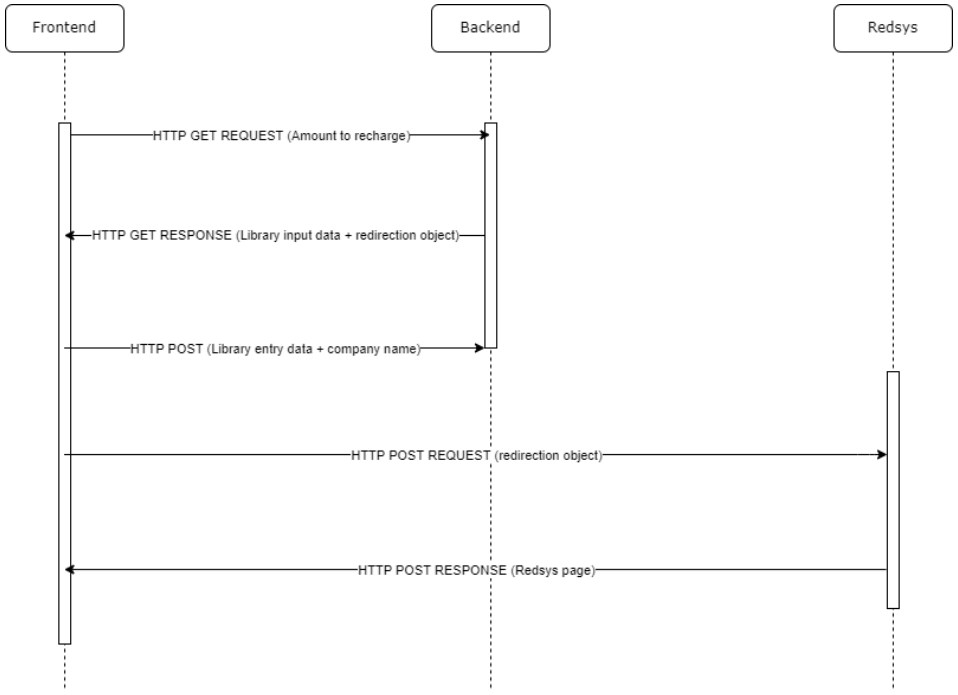


Figure 22 - Electronic payments sequence

Records are kept in the database, within the "charges" collection, with the following additional data:

- Data of the input object for the library indicated in section 4.2.2.1
- Company name

In addition, a "balance" field has been added to store the current balance of the company in the "companydocuments" collection that sums up all the credit being purchased and discounts all the charges being applied for the services used.

3.3 USAGE OF PRIVATE DATA COLLECTIONS FOR CONFIDENTIALITY

3.3.1 Requirements

As described in the VGM section in D3.4, some of the weight request and VGM data, such as the scale operator company's price for the weighting operation, is considered business sensitive and should not be shared with the company's competitors, i.e., the other scale operators on the network. This data should be shared only with "neutral" stakeholders of the network, such as VGM solution provider, PCS system, and the companies which are involved in one or another role in the weight request itself (such as shipper/freight forwarder, road hauler, and shipping line). We refer to the sensitive data from now on as "private data" or "private asset".

Scale operators are network stakeholders which may hold their own peer nodes and therefore have a copy of the ledger. Hence, theoretically, if all the data, including private data, is replicated in the ledger of all peers in equal manner, the scale operator's competitor companies could gain access to this sensitive data.

Therefore, a solution is required in this case where:

- The scale operator company can access their own data,
- The allowed parties, such as the companies participating in the weight request in question, can access and view the sensitive information and private assets, and
- The other scale operators' companies in the network have no copy of the other scale operators' private data in their ledger.

We have addressed the possible solutions for this problem in D3.4. The chosen solution was to utilize private data collections to store the private assets, which would be held only by authorized peers (the PCS and VGM solution provider) ledgers, but not by the scaling operators' peers. Non-sensitive information would be stored on the peers of all blockchain organizations (Figure 23).

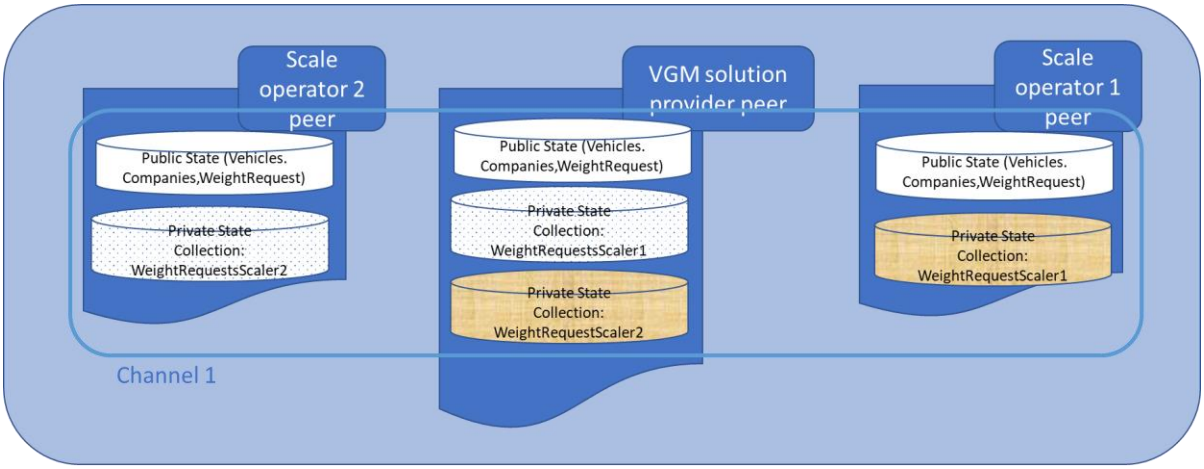


Figure 23 - Private data collection on organizational peers (source: D3.4)

A private data collection is the combination of two elements:

1. The actual private data sent peer-to-peer via gossip protocol to only the organization(s) authorized to see it. This data is stored in a private state database on the peers of authorized organizations, which can be accessed from chaincode on these authorized peers. The ordering service is not involved here and does not see the private data.
2. A hash of that data, which is endorsed, ordered, and written to the ledgers of every peer on the channel. The hash serves as evidence of the transaction and is used for state validation and can be used for audit purposes.

When a user logs in through a client application, they log in into organizational peer belonging to their blockchain organization (this is done by utilization of certification, authentication, and authorization mechanisms built-in into Fabric itself, providing CA and Membership Service Provider (MSP) services to verify that each user can interact only with their own organizational nodes and do so according to their organizational role). This means that if a user is a scale operator user, they will login into scale operators' organizational peer and will invoke transactions through this peer. In case of the scale operator peer, which doesn't have access to the private assets, the response to those transactions would not include the sensitive data stored in the private data collection.

The end users whose organizations do not have a blockchain organization and blockchain nodes belonging to their companies (road haulers, freight forwarders) will be registered by VGM solution provider CA and therefore belong to the VGM solution provider blockchain organization. Therefore, they will have access to the private access collection data as "users" of this organization and will be able to access private data collections through VGM solution provider's blockchain peers.

3.3.2 Design

To use a private data collection within BC solution the following is required:

- Private data collection definition – a JSON-based list of private data collections used in the chaincode for storing/retrieving data. A collection definition contains one or more collections, each having a policy definition listing the organizations which are allowed to access the collection, as well as properties used to control dissemination of private data at endorsement time and, optionally, whether and how the data in the collection will be purged. The private data collection definition file path is used during the chaincode instantiation command to configure the collection on the channel.
- Chaincode enhancements reading and writing to private data collection while storing or reading VGM assets. When reading or writing to a private data collection instead to world-state ledger, a different shim function is invoked from the chaincode:

  ```
  <async> putPrivateData(collection,key,value)
  ```

  where a collection name to which to write, a key which identifies the record and the value to write to the private data collection should be provided.

  The same goes for reading of private data – a `getPrivateData()` shim function should be invoked to read private asset information from the private data collection, where a name of the collection to read from and the key to read should be provided as arguments.

In case of our VGM MVP the following artifacts were added to the solution to support private data:

The collection definition JSON:

```
[
  {
        "name": "collection_service",
        "policy": "OR('PCS.member','VGM.member')",
        "requiredPeerCount": 1,
```

```
        "maxPeerCount": 3,
        "blockToLive":0,
        "memberOnlyRead": true,
       "endorsementPolicy": {"signaturePolicy":
"AND('PCS.member','VGM.member')"
    }

    }
  ]
```

The following fields are of importance:

- This private collection `name` defines the type of asset which will be stored in this collection (each VGM asset, such as vehicle, VGM request, company has its own asset type). 'service' is the asset type for VGMAsset representing the VGM request.
- `policy` indicates the organizations which have access and can store and read data from this collection. In the VGM case those are the PCS and VGM organizations member nodes.
- `requiredPeerCount` indicates the minimum number of peers (across authorized organizations) that each endorsing peer must successfully disseminate private data to before the peer signs the endorsement and returns the proposal response back to the client. This is required for data availability purposes so even if endorsing peer(s) become unavailable the data can be found in the network of other authorized organizational peers.
- `blocksToLive` indicates whether to purge private data after a certain amount of blocks is written to the collection. Value of 0 indicates never to purge.
- `memberOnlyRead/Write`**:** if true, automatically enforces that only clients belonging to the collections' membership organizations are allowed to read/write private data from chaincode (default values for this flag are true). If a client from a non-member org attempts to execute a chaincode function that performs a read/write on a private data key, the chaincode invocation is terminated with an error.
- `endorsementPolicy`**:** An optional endorsement policy to utilize for the collection that overrides the chaincode level endorsement policy. More details about this will be provided in the following paragraphs.

Implementation in the chaincode whether, when, and how to use private data collection for storing assets. Two parameters designate whether a VGM asset would be stored on world-state ledger seen to all VGM peers or on a private data collection:

- A general configuration parameter *privateCollections*, set in conPESO client application and passed to the chaincode during initialization indicating whether to use privateCollections (by default set to false).
- For each VGM asset, a *privateAsset* flag which is set to true if the designation of the asset data is to be saved in private data collection or false otherwise (in which case the asset data would be stored on the ledger).

During chaincode write or read operations implementing CRUD for VGM assets, if both of those flags are true, the "public" part of the data asset (common fields such as **id**, **creation timestamps**, **asset type**) along with the **hash** of the private asset would be stored on the public ledger, while the private part of the asset would be stored in private data collection. When retrieving the asset, if its private asset then the data is retrieved from private data collection, the public part of the asset is retrieved by ID from the world-state ledger, and the hash is compared to verify the validity of the private data.

Another important thing to mention is that private data is passed as transient information into transaction invocation function and can be retrieved from the context by `getTransient()` function call. Since all transaction data is added into a block which is then chained to the ledger's transaction log, this ensures that private data would not be placed in the log and therefore be reachable by all peers holding the ledger.

All chaincode CRUD operations (create/read/update or delete of asset) include the following logical sequence:

For read operations:

```
{
     publicAsset = getState(ID)
     if (privateCollections==true and privateAsset==true) then
     {
             privateData = getPrivateData(assetType.collectionName,ID)
             hash = getPrivateDataHash(assetType.collectionName, ID)
             if (publicAsset.hash !== hash) {
              throw Error('Invalid hash');
             publicAsset = merge(publicAsset,privateData)
             return publicAsset;
         }

     }else
     {
             return publicAsset;
     }

}
```

For write operations:

```
{
 publicAsset= assetData[commonFields]
 if (privateCollections==true and privateAsset==true) then
 {
      privateData = context.getTransientData()
      putPrivateData(assetType.collectionName,ID, privateData)
      publicAsset.hash = getPrivateDataHash(assetType.collectionName,ID)
      putState(ID,publicAsset)

 }else
 {    putState(ID,assetData)
 }

}
```

Endorsement policy (as defined in the private collection definition)

VGM solution current blockchain network is comprised from three blockchain organizations: PCS, VGM solution provider, and scale operator. The chaincode level endorsement policy used in the VGM solution is AND among the endorsing peer replies of the three organizations

```
AND(PCS.member, VGM.member, ScaleOperator.member)
```

This means that in order to endorse (approve and commit) a transaction, an endorsing peer of each of the organizations needs to run the chaincode and receive a successful result of the chaincode execution (comprising of read/write set). These 3 results should be identical among the endorsing peers.

This is the behaviour we want for shared data, so only if the three organizations agree on the outcome of the transaction this transaction should be committed and its results applied to the ledger.

However, the implication of this policy in private data collections related transactions, where less than three organizations can access the private data collection would result in failure of those transactions since at least one organization's endorsing peers (organization which is not the member of the private data collection) would fail in the chaincode execution when trying to read/write to private data collection and therefore the transaction will fail (will not be committed by either peer).

To avoid this, since we do want to commit private data transactions as long as they are approved by the private data collection member organizations, we will define an endorsing policy on the level of the private data collection which will be applicable only to operations of data read/writes to this private data collection. The new policy will be defined as agreement between endorsing peers of those organizations which can access the private data collection. Therefore, for our case the endorsing policy for access to the `collection_service` (see above) would be an **AND** of results from PCS and VGM endorsing peers execution only, and it is enough that those two organizations endorse the chaincode transaction reading/writing to this private data collection for the transaction to be successful and its results applied to the ledger.

3.4 APPLICATION OF MONGODB CACHING IN THE CLIENT

3.4.1 Requirements

VGM assets' information is stored in BC ledger for transparency, provenance, and non-repudiation purposes. The VGM frontend application (conPESO) provides end-user with easy-to-use rich UI with functionality allowing to store asset data, read current information regarding an asset, or perform rich queries to search for assets.

Blockchain main purpose is to provide a trusted and immutable environment for data provenance and non-repudiation. This is achieved by replicating and storing the same data on multiple ledger copies on each of the blockchain peer nodes. Due to the nature of blockchain, the performance of read and writes is inferior to the performance of state-of-the-art databases and NoSQL solutions for data storage.

Since VGM solution is intended to support multiple organizational end-users and port systems simultaneously, and especially to provide fast responses in an environment where the amount of queries and tuples in query responses can reach large numbers, we have designed the solution in such a way as to provide the transparency and single source of truth using BC, and to integrate MongoDB as a cache to be used on top of blockchain for fast data access and query support.

3.4.2 Design

When creating or updating assets using the conPESO frontend, the information is stored on the blockchain ledger and at the same time in the MongoDB instance serving as fast-access cache (Figure 24). The conPESO user interface (UI) provides views into the list of assets of different types by performing complex queries with different filters from MongoDB. Those queries return tuples where each tuple contains asset ID and some high-level information pertaining to the asset. When a particular asset is chosen, this asset information is read from the BC ledger since it serves as the single source of truth in the application

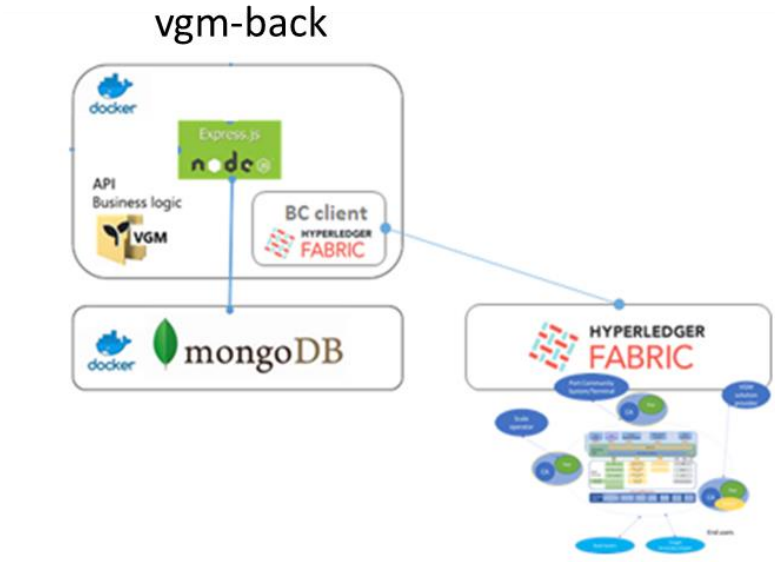


Figure 24 - VGM UI backend and connection to MongoDB and BC

The following diagrams describe the sequence of operations performed on asset update operations (such as creation, update, or deletion) and asset read operation (querying for asset information).

For the update operations, the asset information is stored both in MongoDB for fast querying, and in BC to serve for non-repudiation and single source of truth (Figure 25).

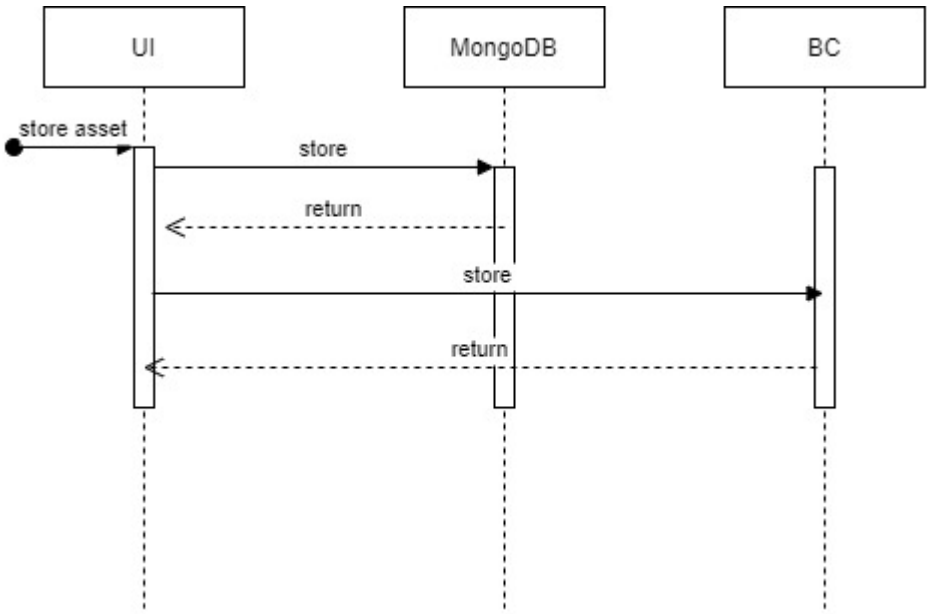


Figure 25 - Asset update sequence

For querying of the data assets in order to populate UI views, the query is performed on top of MongoDB cache. This returns a list of tuples with general information, such as assetID, and some common fields. When farther clicking on the asset to get asset's details, the read operation is performed against the BC to get the most updated and verifiable information on the asset (Figure 26).

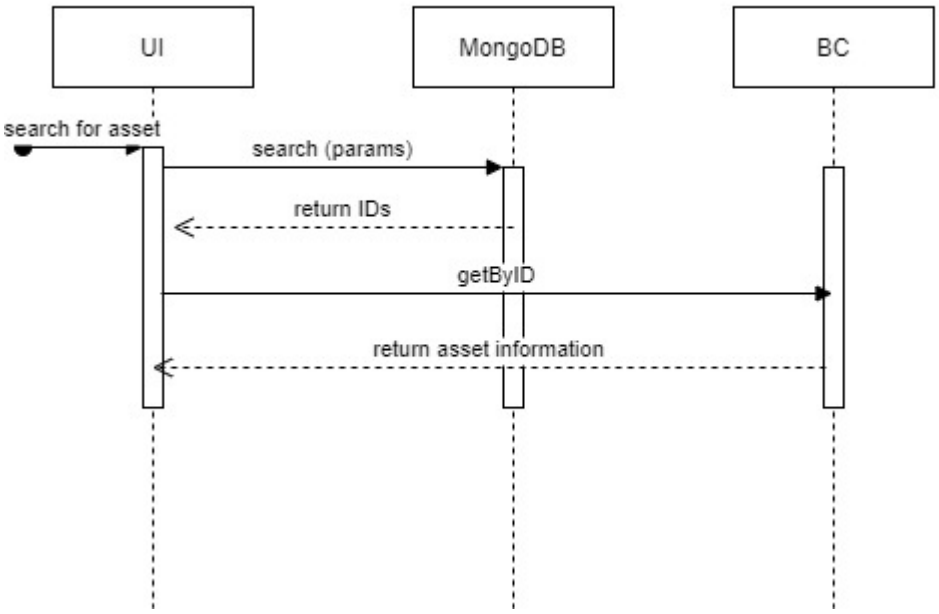


Figure 26 - Asset query sequence

3.5 MONGO-DB BASED WALLET FOR BC CLIENT

3.5.1 Requirements

For a user to be able to connect through a BC client and submit transaction, they should be certified and approved by their organizational Certification Authority service. The user's identity, when created by the certificate authority, is placed in a Fabric wallet. A wallet contains a set of such user identities. An application run by a user selects one of these identities when it connects to a channel. Access rights to channel resources, such as the ledger, are determined using this identity in combination with an MSP. Identity is comprised from certificate, private key and Fabric metadata as can be seen in Figure 27.

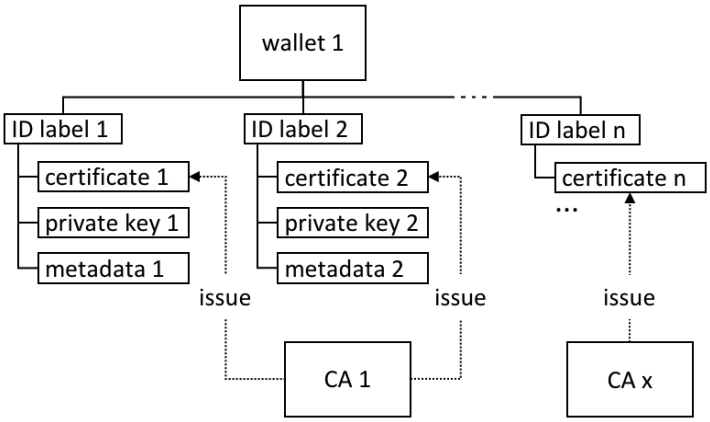


Figure 27 - Fabric wallet holding identities (source "Fabric 1.5 readthedocs: Wallet")

When several frontend apps, representing different blockchain organizations, interact with a BC, each will incorporate its own BC client with its own wallet for user identities managed by this application.

There are different types of Fabric wallets built-in into the Fabric SDK – File, In-Memory, and CouchDB wallets. The different types are different in the medium where the identities are stored – applicational file system, in-memory volatile wallet used in resource-scarce environments usually without access to

a file system, and a CouchDB wallet for those users who want to use the database back-up and restore mechanisms, which is a useful option to simplify disaster recovery.

Wallets are incorporated into the BC client when creating a gateway for communication with the blockchain network: a path/configuration of a wallet to use is specified at the time of initiating a connection with the gateway. Identities from this wallet will be used by the gateway to submit Fabric transactions to the network.

In the VGM application, since a MongoDB instance was already utilized as data storage cache for the BC-backed data assets, a MongoDB based wallet was implemented for user identities storage. Additionally, identities stored in the MongoDB are encrypted to make the wallet more secure. Last, but not least, the MongoDB wallet is backed by in-memory wallet for better performance, identities from MongoDB could be imported into in-memory wallet for faster access in subsequent calls. Identities from cached wallet are imported into the db-based wallet for persistence.

3.5.2 Design

A wallet implementation should implement a common Fabric Wallet[4] interface. It should support the following operations:

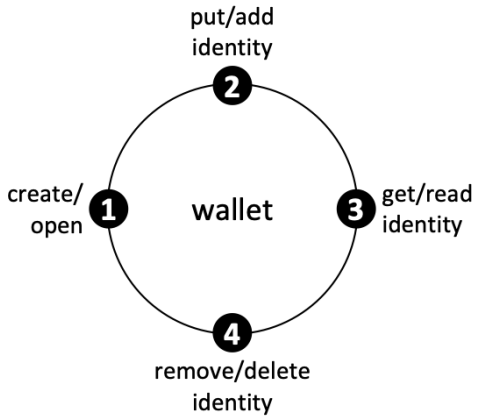


Figure 28 - Fabric Wallet operations (source "Fabric 1.5 readthedocs: Wallet")

- Create/open: wallet lifecycle management operations, creating a new wallet, or opening existing one (saved on file system or db)
- Wallet identity operations: adding a new identity, getting an existing identity, or deleting existing identity (Figure 29).

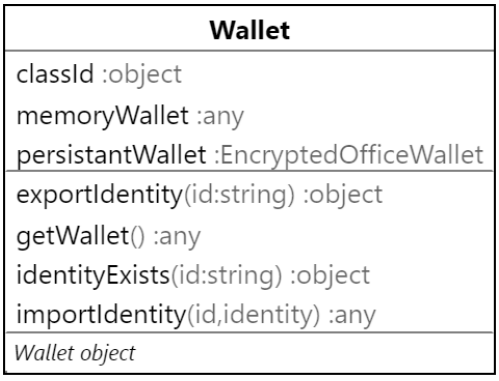


Figure 29 - Encrypted MongoDB backed wallet with in-memory cache

[4] https://hyperledger.github.io/fabric-sdk-node/release-2.2/module-fabric-network.Wallet.html

MongoDB backed wallet is comprised by a standard in-memory wallet which is provided by the Hyperledger Fabric library and a persistent encrypted wallet where the cryptographic material for connecting to the DLT is stored for each company office encrypted in MongoDB. This solution provides a secure repository for the cryptographic material of each identity in the system instead of saving this material in the file system. The in-memory wallet is synchronized with the persistent encrypted wallet through the create, put, read and delete identity interface of the wallet. When it is required to obtain the identify of a company office, it is checked the office wallet, if the identity already exists in memory, it is returned from there, otherwise it is previously retrieved from the DB persistent encrypted wallet to the in-memory wallet. In the event that we want to add, replace or delete a wallet, we remove the in-memory wallet if it exists and update the DB persistent encrypted wallet from the crypto-material returned by the Certificate Authority. These operations are made through a CA admin user configured in the system (with the credentials encrypted in the DB) to perform the operations of creating and removing the cryptomaterial with the identities of the company offices that will access to the DLT.

The database model that supports the admin user capable of accessing the CA and managing identities is stored in the DB ConfigData sheet, which stores the relevant information regarding the admin Wallet in these four fields:

```
bcAdminId: string;
bcAdminSecretEncrypted: any;
bcAdminEncryptedWallet: any;
bcAdminOfficeId: string;
```

The encrypted Wallet of the administrator user is stored in bcAdminEncryptedWallet, while we have their user ID, password, and office ID in bcAdminId, bcAdminSecretEncrypted, and bcAdminOfficeId respectively. The company office wallet information is stored in the Office collections together with other attributes relevant to characterize the office in the solution:

```
bcEnrollmentID?: any;
bcEncryptedWallet?: any;
bcMSP?: string;
```

3.6 ADDITIONS OF BLOCKCHAIN QUERIES FOR VARIOUS ENTITIES

3.6.1 Requirements

Initially, a few queries were created in the chaincode to support searching of different assets (vehicle /weight request) based on their properties. In version 2 of the MVP, these queries were supposed to be extended to include more asset types (company information) and types of queries.

Due to moving the querying capabilities of the solution from blockchain to MongoDB cache for better performance, this feature requirements of additional queries on BC level are obsolete and therefore not implemented.

3.6.2 Design

Not applicable as this feature is out of scope

3.7 DATA MODEL AND CHAINCODES GENERALIZATION FOR EASIER CODE EXTENSION

3.7.1 Requirements

Redesigning of data model and chaincode generalization to allow easier addition of new data entities, and chaincode logic relating to those data entities, including entity storage, retrieval, and access management functions.

3.7.2 Design

The chaincode data model and smart contracts design was updated to create a hierarchy of inheritance of generic asset contract, with common operations, which can be extended for each additional asset type (Figure 30).

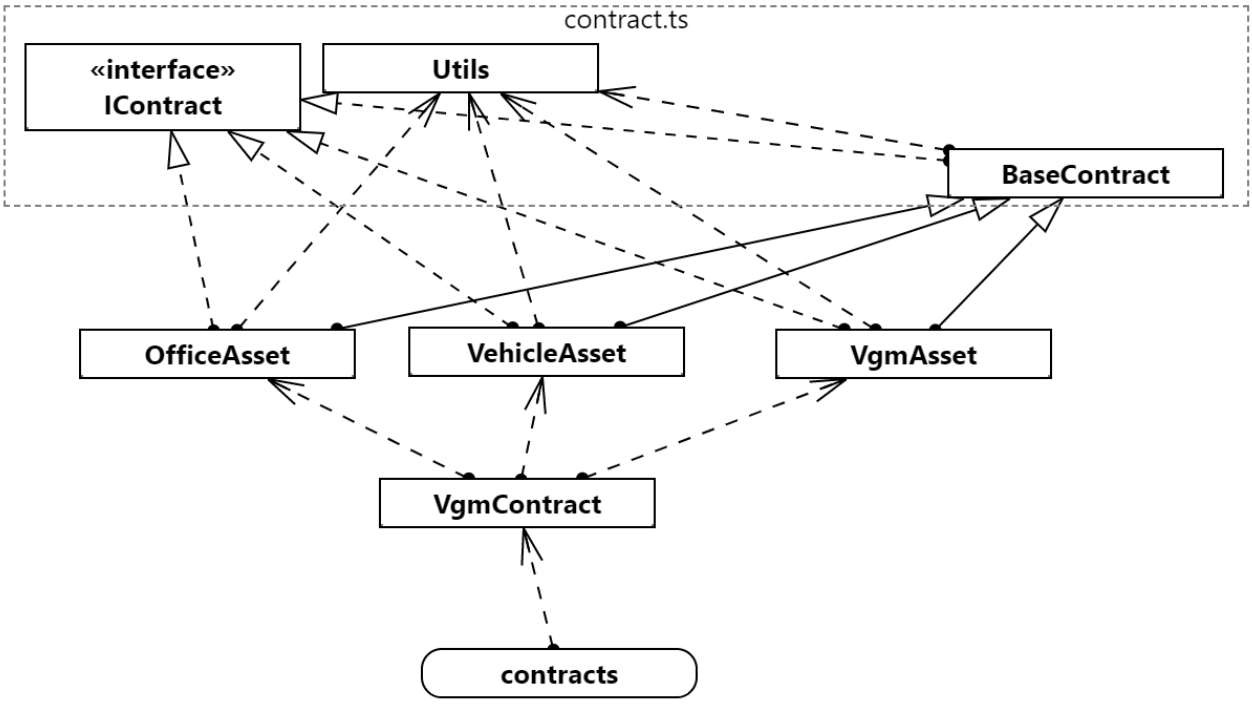


Figure 30 - Smart Contracts design

The IContract is an interface that indicates that any class that implements it must at least implement the following methods:

```
getAssetType(): string;
getKey(ID: string): string;
validateObjectBeforeCreate(ID: string, newAsset: any): any;
validateObjectAfterRead(existingAsset: any): any;
validateObjectBeforeUpdate(ID: string, existingAsset: any, newAsset: any): any;
assertBeforeDelete(ID: string, existingObjectContent: any): boolean;
parseReadObject(json: string, classType: any): any;
```

The BaseContract class is the class which includes the common logic to any chaincode for managing an asset in the DLT. The BaseContract is composed by a set of common attributes for any asset chaincode, the IContract empty methods which will be specialized by the inherited BaseContract classes to control and validate the access and registration of specific assets, according to their asset business rules and the common methods to manage any asset:

```
assetExists(ID: string): boolean;
readAsset(ID: string): any;
createAsset(ID: string,  content: any, officeIDs: string = []);
updateAsset(ID: string, content: any, officeIDs: string = []);
deleteAsset(ID: string): boolean;
```

```
getAssetHistory(ID: string): any

getQueryResults(queryString: string): any[]

getQueryResultsWithPagination(queryString: string, pageSize: number,
bookmark: string, currentPage: number): AssetList
```

From the BaseContract class, any asset to be registered in the DLT can be easily created by specializing the BaseContract with a specific Asset (e.g. OfficeAsset, VehicleAsset, VgmAsset, ChargesAsset, DocumentAsset), and implementing the business rules through the IContract methods to establish the rules to validate an object and determine the company office rights before creating, updating or deleting it, retrieving the IDs of assets according to query parameters or validating an object after reading it or obtaining the asset history and filtering the attributes the company office identity has access to.

Finally, the VgmContract class is the class which serves as an entry point to the chaincode, and that allows calling any method of assets such as OfficeAsset, VehicleAsset, VGMAsset, ChargesAsset, and DocumentAsset.

3.8 CONFIGURABLE BC CLIENT

3.8.1 Requirements

The way conPESO application (or any BC client application) interacts with the BC network is via Fabric client SDK. This can be illustrated in Figure 24, where the VGM UI backend incorporates a BC client SDK and uses it to communicate to Fabric network.

The Fabric client SDK provides simple API to connect to the network and submit transactions to the ledger or query the content of the ledger.

Fabric Client SDK provides a Gateway class, which defines the endpoint of the Fabric to connect to (URL of organizational peer), and which supports various connection options to define how to instantiate and interact with the Fabric network.

Configuring these connection options differently will result in successful/unsuccessful creation of connection to Fabric network and to different runtime behaviours of the BC client.

To support flexibility in specifying those connection properties, conPESO UI was extended to include a page where BC client connection options can be specified and modified.

3.8.2 Design

The following options are passed to BC client for fine-grained configuration of the gateway:

- `bcActive`: indication whether to use BC as backend, or to work only with MongoDB
- `peers`: definition of the peer endpoint URL to connect the gateway to. This will be the organizational peer to which the client will submit transaction invocations.
- `bcAsLocalHost`: inidication whether to connect to locally run (dockers) BC environment, or external IP.
- `bcChannelName`: the name of the channel to connect to
- `bcContractName`: the smart contract (chaincode) name which will be invoked when submitting transactions

Figure 31 shows the UI supporting configuration of these options.

Figure 31 - Connection configuration UI

There is a section for blockchain in the configuration view, in which we have options to:

- Activate or deactivate the connection.
- View and edit the JSON that determines the composition and URL of the different components of the Fabric network: peers, channels, organizations, orderers, and certification authorities.
- Read or change identifier, password, and office of the network administrator.
- Read or change affiliation identifier that will be used to register new identities.
- Read or change channel name.
- Read or change the name of the smart contract.
- Read or change whether we are going to work with blockchain locally in development, or on already deployed in production.
- View the encrypted content of the administrator user's wallet.

4 DATA SHARING BLOCKCHAIN NETWORK - CONTAINER PICK-UP USE CASE

In this section, we describe the design of the additions and extensions to the Container Pick-Up (CPU) use case and the resulting Minimal Viable Product (MVP) developed for the port of Thessaloniki. This use case has already been described in deliverables D3.4, D4.2 and D5.3[1]. In this deliverable, we give an update on the state of the functionality and features that were described as "next steps" in D3.4 and D4.2 and in the Action Plan of D5.3. These features will be part of an updated second version of the MVP (MVP v2). Table 6 below presents them along with their status ("in progress", "done", "dropped"). Features "in progress" will be finalized by the submission of D4.4, while "dropped" means that the specific feature will not be part of CPU MVP v2.

#	Feature/activity	Status	Comments
1a	Integration with Port of Thessaloniki computing infrastructure	In progress	
1b	Notification of the CPU backend by the port infrastructure about truck departure	In progress	
2	Addition of basic analytics on CPU use case data in the "Home" page of the frontend	In progress	
3	Shipping agent can only view bookings created for their COREORs	Done	Privacy issue that was addressed.
4	A trucking company only has access to the container information of the COREORs that reference them	Done	Privacy issue that was addressed.
5	Deploy the solution on a cloud infrastructure	Dropped	THPA made it clear that this is not something they want.
6	Access to Data Governance from within CPU frontend	Dropped	It was decided that this would unnecessarily complicate the frontend. It is easy for the user to simply have two browser tabs open, one for CPU and one for Data Governance.
7	Management of COREORs and bookings by the port through the CPU frontend	Dropped	THPA never intended for its own workflow to change, therefore acceptance and rejection of COREORs/bookings is still going to be handled through their systems.

Table 6 - CPU MVP v2 new features and their status

In the following sections, we address the main features, namely 1a, 1b and 2, by explaining their inclusion in version 2 of the MVP, their requirements, and design.

4.1 INTEGRATION WITH PORT OF THESSALONIKI COMPUTING SYSTEMS

4.1.1 Requirements

Figure 32 describes the infrastructure that participates in the use case, the participant users and the data that are passed amongst them. The main scenario for the CPU use case is depicted in the sequence diagram of Figure 33.

As can be seen in Figure 32, there are two infrastructure components in the system, i) the DataPorts CPU MVP, consisting of the CPU frontend and the Fabric backend, and ii) the port infrastructure. Participating in the use case and interacting with the infrastructure are users from three organization types: port, shipping agent, and trucking company. As can be seen, each infrastructure has its own separate user accounts. There is a port operator user for the CPU MVP and another port operator user in the port infrastructure, and the same goes for the trucking company. Note that all participant organizations have access to the port's systems although no shipping agent interaction between a shipping agent user and the port infrastructure is shown. This is because this interaction plays no role in our scenario as the shipping agent does not enter any information into the port's systems. In contrast, the trucking company representative still has to apply for a booking through the port's system.

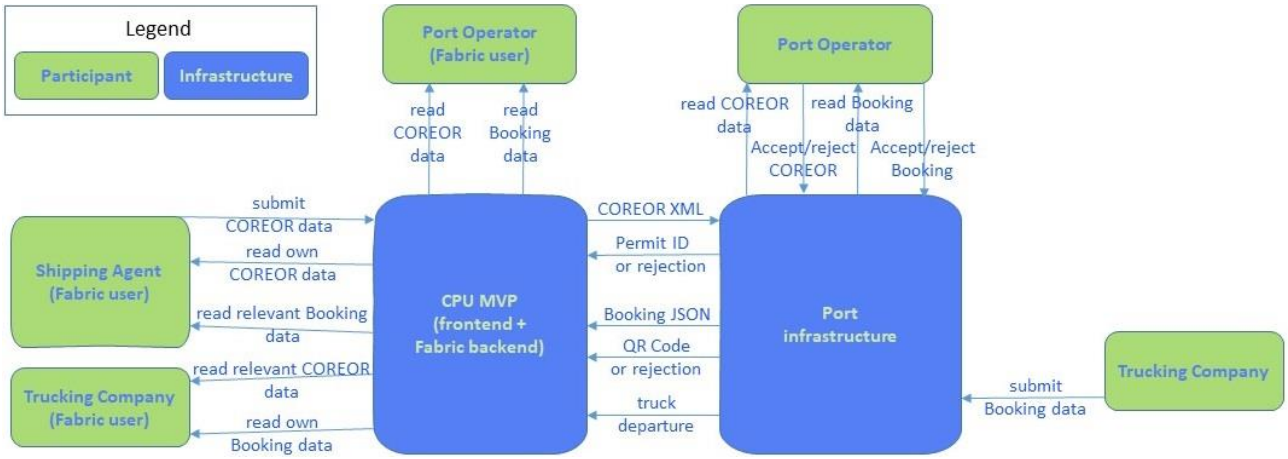


Figure 32 - Data providers and data consumers in the CPU use case

The sequence diagram of Figure 33 contains two alternative fragments: i) "decision on COntainer RElease ORder (COREOR)" (red) and ii) "decision on booking" (green) – with the former containing the latter. The red alternative fragment describes the two possible outcomes of a COREOR request, acceptance, or rejection of the request, while the green alternative fragment, contained within the sequence of an accepted COREOR, describes the two outcomes of accepting or rejecting a booking request. Table 7 contains the description of the steps taken in the main CPU scenario. The numbering corresponds to the numbers in Figure 33.

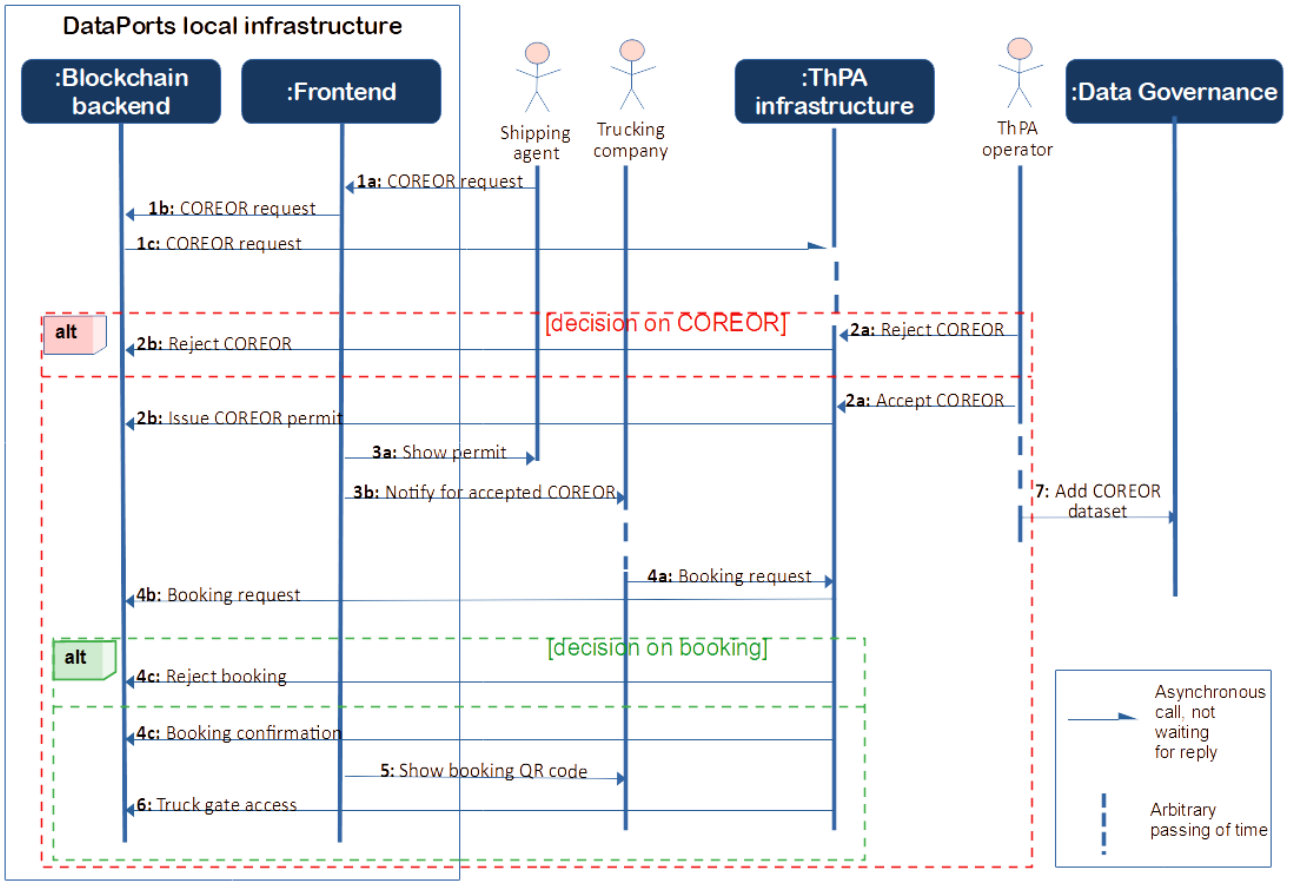


Figure 33 - Sequence diagram for the CPU use case

In Table 7, the main requirements for the integration of the CPU use case with the port's computing infrastructure are steps 1c, 2b (both), 4b, 4c (both), 6. All these requirements have to do with the need to specify an API for secure communication between the CPU MVP backend and the port infrastructure. As an additional requirement we can say that any user should be able to retrieve the COREOR and booking data that they are entitled to accessing. This can be seen in Figure 32, where the port operator can view any and all COREORs and bookings, while the shipping agent and trucking company can only view COREORs and bookings that are relevant to them.

Step	Description
1a	The Shipping Agent uses the DataPorts CPU solution frontend to register a COntainer RElease ORder (COREOR) request[5], specifying the Trucking Company to pick-up the container.
1b	A copy of this request is saved on the local blockchain infrastructure.
1c	The COREOR request is sent in XML format, to ThPA's TOS (Terminal Operating System), in a secure way.
Alt: Decision on COREOR	
Scenario: Rejection	

[5] The COREOR message is an order to release containers, which gives permission for them to be picked up by, or on behalf of, a specified party. It is used in Electronic Data Interchange (EDI) between trading partners and is specified by the United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport (UN/EDIFACT)

Step	Description
2a	A ThPA Operator checks the request through the TOS and other relevant processes (invoicing, customs clearance) and identifies a problem with the COREOR, thus rejecting it.
2b	The rejection is communicated securely to the DataPorts CPU MVP blockchain backend through an API that exposes the CPU smart contracts. This updates the request in the blockchain, marking it as rejected.
Scenario: Acceptance	
2a	A ThPA Operator checks the request through the TOS and other relevant processes (invoicing, customs clearance), sees that everything is in order and approves the request. A Permit ID is issued for that COREOR.
2b	The Permit ID is communicated securely to the DataPorts CPU MVP blockchain backend through an API that exposes the CPU smart contracts. This updates the request in the blockchain, adding the Permit ID.
3a	The Shipping Agent can view the complete COREOR request information in their DataPorts CPU solution dashboard.
3b	The Trucking Company receives a notification containing the Permit ID.
4a	Using the issued Permit ID, the Trucking Company can proceed to arrange the pick-up of the container, following the normal process of booking an available timeslot through the TAS and Freezone port systems.
4b	The ThPA systems notify the DataPorts CPU solution, through our API, of the new booking, in a secure way.
Alt: Decision on booking	
Scenario: Rejection	
4c	Through a second API call, the blockchain backend is notified in a secure way that the booking was rejected.
Scenario: Acceptance	
4c	Through a second API call, the blockchain backend is notified in a secure way that the booking was accepted.
5	When the booking is accepted, the Trucking Company can view, in their DataPorts CPU solution dashboard, the QR code issued for it.
6	When the truck leaves the port yard, the blockchain backend is notified through an API call, in a secure way.
7	At a later stage, the ThPA operator can make the COREOR dataset available on Data Governance.

Table 7 - CPU Scenario Description

4.1.2 Design

To cater for the needs posed by the above requirements, we have implemented, since M16, the REST API that we presented in section 5.4 of deliverable D4.2. For ease of reference, we include this API here as well:

- `/coreorRequest`, POST
- `/issueCoreorPermit`, POST

- `/getCoreor`, GET
- `/getCoreorsParameterised`, GET
- `/rejectCoreor`, POST
- `/bookingRequest`, POST
- `/bookingConfirmation`, POST
- `/getBooking`, GET
- `/getBookingsParameterised`, GET
- `/rejectBooking`, POST

The corresponding Fabric chaincode (smart contracts) invoked by these endpoints have been described in Annex C of deliverable D4.2.

Only two requirements are yet to be satisfied: i) step 6 of Table 7 and ii) providing a secure means of communication between the two infrastructures.

4.1.2.1 Truck departure

This requirement specifies that the CPU blockchain backend should record the time of a truck's departure from the port yard. This is a new feature that comes to complete the CPU scenario, by providing information on the successful conclusion of a booking. It was decided upon when it was made clear that the port can indeed provide such information through the computing infrastructure that is communicating with the CPU MVP. The corresponding REST endpoint is:

`/truckDeparture`, POST

Given the departure information we can now judge the final status of a booking. If the booking has a departure time, then it can be deemed completed. If not, and the date of the booking has come to pass, the booking can be deemed as failed.

4.1.2.2 Secure communications between CPU MVP and port infrastructure

For the purposes of communication between the CPU MVP and the port infrastructure, security needs to cover two aspects:

- authentication
- encryption

Encryption is covered by the HTTPS protocol used in invoking the REST API offered by the two parties. Authentication is covered by the use of bearer tokens. Both CPU MVP and the port offer a login endpoint, which returns a pair of tokens: an access token and a refresh token. The calling party (the party that has logged in) must use the access token in every subsequent REST call they make, placing it in the "`Authorization`" header in the format "`Bearer <token>`". However, both tokens have an expiration time, with the refresh token expiring slightly later than the access token. Once the access token expires, the calling side must make a REST call to a special endpoint to replace their tokens with two new tokens. On the CPU MVP side this endpoint is the following:

`/refreshTokens`, POST

Whenever both the access and refresh tokens are expired, a login must again be performed.

4.2 ADDITION OF BASIC ANALYTICS FOR THE CPU USE CASE

4.2.1 Requirements

The requirements for basic analytics being present in the CPU frontend were mentioned both in section 6.4.5 of D3.4 and in section 5.5 of D4.2, in both cases assigned to version 2 of the MVP. Examples of such analytics can be:

- For the port operator
  - which shipping agent has the most COREORs
  - which shipping agent has the highest COREOR rejection rate
  - which trucking company has the most bookings
  - which trucking company has the highest bookings failure rate
- For the shipping agent
  - which trucking company has been used the most
  - which trucking company has the highest bookings failure rate
- For the trucking company
  - which shipping agent are we working most with
  - percentage of bookings that were rejected or failed

4.2.2 Design

The result of the analytics performed on the CPU use case data will be a series of charts like the ones displayed in Figure 34, Figure 35, and Figure 36.

Figure 34 - COREORs per Shipping Agent

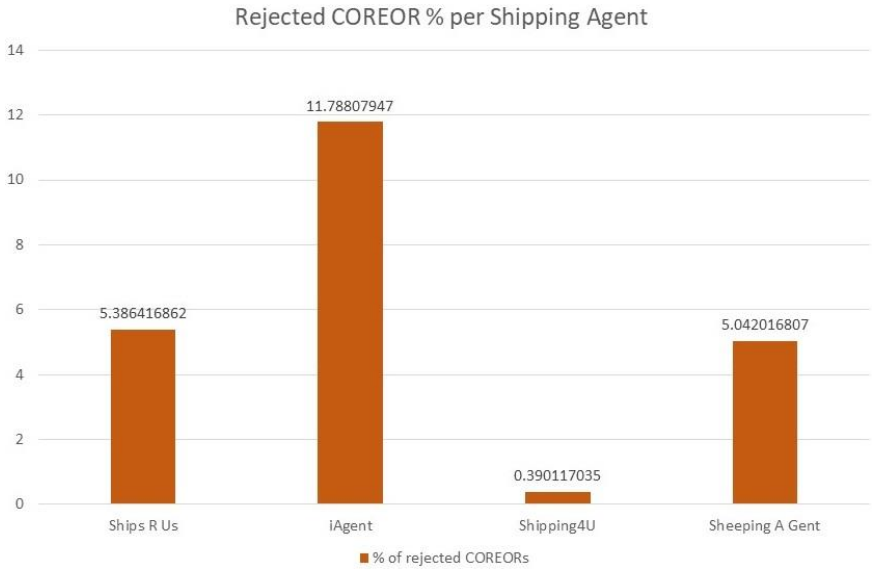


Figure 35 - Rejected COREOR percentage per Shipping Agent

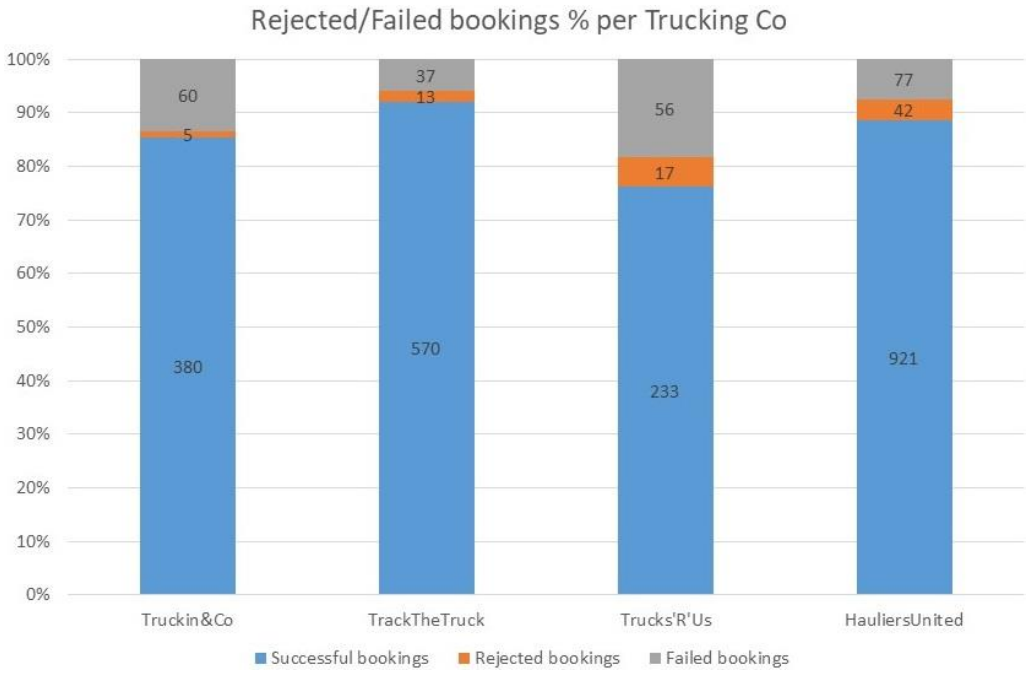


Figure 36 - Rejected/Failed bookings percentage per Trucking Company

These charts will be shown in the home page of every user, comprising a dashboard that will be tailored according to the user's organization, i.e., different charts with different data will be shown to the port, shipping agent, and trucking company users. This dashboard will be similar to the one depicted in Figure 37 below.

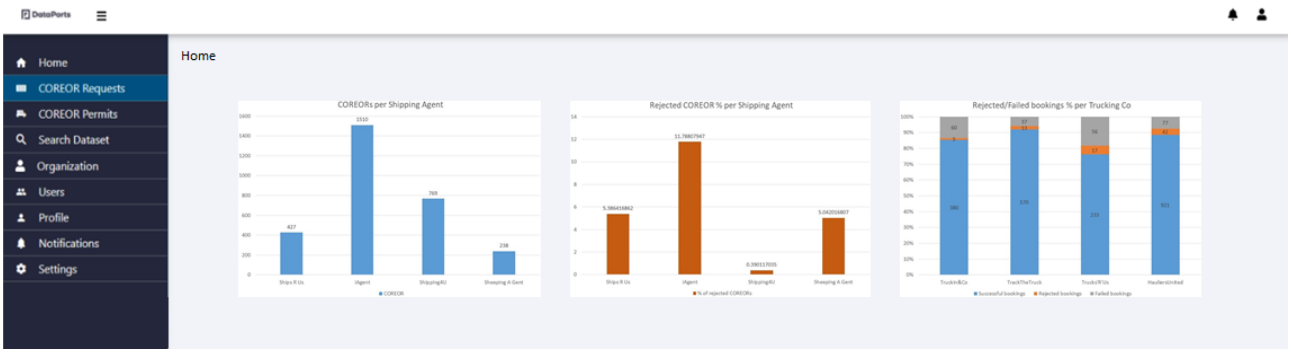


Figure 37 - CPU MVP homepage dashboard

5 CONCLUSIONS

In our previous deliverable, D4.2 - Blockchain based data governance rules M20, we articulated the implementation details of each of our three devised blockchain prototypes: DG, VGM, and CPU. Each BC network is unique and plays a different role in the overall DataPorts platform. While the Data Governance network is a core component of the platform aiming at providing governance of the data sets owned by the different stakeholders of the platform, the two other networks fulfil specific objectives in the local ports aiming at improving specific processes. This difference dictates the nature of the networks and emphasises on the specific peculiar features making each of them unique.

The deliverable in hand is the natural continuation of D4.2 as it relates to each of the enhancements that were already identified in D4.2 as next steps for each of the BC networks. Furthermore, being DataPorts a dynamic project, we also incorporated new features that resulted from new requirements received since M20 of the project. Special attention and efforts have been invested towards the integration of the networks – DG in the DataPorts platform and VGM and CPU within the operational systems in the ports.

This report relates to each enhancement or extension and provides the requirements that they satisfy along with a detailed design. The partners working on the three prototypes have already developed part of these new features, and their completion will end in M30 and will be reported in D4.4 – Blockchain based data governance rules.

We would like to conclude with a single sentence for each BC network that characterizes the highest achievement foreseen for each of the MVPs V2.

- Data Governance is integrated within the platform with full capabilities for managing the access rights to datasets in the DataPorts platform and enabling a trusted, secure, and trackable exchange of data among the stakeholders of the DataPorts ecosystem.
- VGM becomes fully operational in the Valencia port and replaces the current system they apply for weighting the containers in the port ensuring a secure, trusted, and non-repudiation process, including support for privacy of the shared data.
- CPU becomes fully operational in the Thessaloniki port supporting the sharing of COREOR requests and booking data in an efficient, verifiable, and permanent way among the three concerned organization types: the port, shipping agents, and trucking companies.

6 REFERENCES AND ACRONYMS

6.1 REFERENCES

[1] Hyperledger Fabric – Hyperledger," 2020. [Online]. Available: https://www.hyperledger.org/use/fabric. [Accessed 15-March-2022].

[2] Gaur, N., Desrosiers, L., Ramakrishna, V., Novotny, P., Baset, S.A., and O'Dowd, A. (2018). Hands-on Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer. Packt Publishing.

6.2 ACRONYMS

Acronym List	
API	Application Programming Interface
BC	Blockchain
CA	Certificate Authority
COREOR	COntainer RElease ORder
CPU	Container Pick-Up
D	Deliverable
DG	Data Governance
DLT	Distributed Ledger Technology
DoA	Description of Action
DS	Data Source
EDIFACT	Electronic Data Interchange for Administration, Commerce and Transport
IM	Identity Manager
JWT	JSON Web Token
M	Month
MVP	Minimal Viable Product
PCI-DSS	Payment Card Industry Data Security Standard
PCS	Port Community System
SDK	Software Development Kit
SI	Semantic Interoperability
T	Task
TOS	Terminal Operating System
VGM	Verified Gross Mass
WP	Work Package

Table 8 - Acronyms