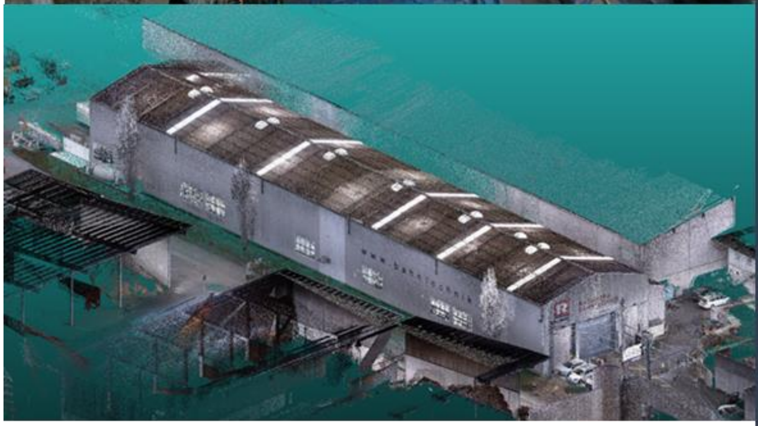


COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

D7.9 – Digital Twin Platform v1

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310

D7.9 – Digital Twin Platform v1

Dissemination Level:	Public
Deliverable Type:	Demonstrator
Lead Partner:	UCL
Contributing Partners:	Hypertech, UPM
Due date:	31-05-2022
Actual submission date:	31-05-2022

Authors

Name	Beneficiary	Email
Kyriakos Katsigarakis	UCL	k.katsigarakis@ucl.ac.uk
Georgios N. Lilis	UCL	g.lilis@ucl.ac.uk
Dimitrios Rovas	UCL	d.rovas@ucl.ac.uk
Salvador Gonzalez-Gerpe	UPM	salvador.gonzalez.gerpe@upm.es
Apostolos Papafragkakis	Hypertech	a.papafragkakis@hypertech.gr
Giorgos Giannakis	Hypertech	g.giannakis@hypertech.gr

Reviewers

Name	Beneficiary	Email
Amy Wilson	UEDIN	Amy.L.Wilson@ed.ac.uk
Damiano Falcioni	BOC	damiano.falcioni@boc-eu.com

Version History

Version	Editors	Date	Comment
0.1	UCL	01.05.2022	ToC
0.3	UCL	10.05.2022	Draft version of sections 1,2,3,4,5
0.6	UCL, UPM, Hypertech	14.05.2022	Contributions in section 4
0.8	UEDIN, BOC	27.05.2022	Internal review
0.9	UCL	30.05.2022	Internal review comments addressed
1.0	UCL, Hypertech	31.05.2022	Submission to EC

Disclaimer

Executive Summary

The COGITO Deliverable “D7.9 – Digital Twin Platform v1” documents the COGITO Digital Twin Platform and reports the outcomes of work performed thus far in “T7.5 – Digital Twin Platform Development and Testing”. In summary, the Digital Twin Platform (DTP) is a cloud-based and semantically enabled data integration middleware that includes a comprehensive suite of services to guarantee scalability, reliability, and enhanced security, and it is responsible for: a) providing authentication and authorisation to COGITO applications and users, b) handling data from various input sources such as BIM authoring tools, project management tools, cameras, LiDAR scanners and IoT devices, and c) responding to data requests performed by the other COGITO applications. The DTP follows a multi-layered architecture comprising six core layers. Each layer contains a set of software components implementing the Service-Oriented Architecture (SOA) design pattern and performing various business logic operations to support the main objectives of the COGITO solution. The primary output of the work conducted thus far in T7.5 is the implementation, deployment and testing of the DTP that comprises only a set of the core software components following the Minimum Viable Product (MVP) approach. These components have been deployed in the various layers of the DTP based on the outcomes of “T7.1 – Digital Twin Platform Design & Interface Specification”. This document mainly focuses on presenting the current version of the DTP and its software components along with the functionalities they provide, the technology stacks they build upon, the interfaces they use, the usage instructions and the assumptions and restrictions.

In the first version, the DTP provides a set of core functionalities such as i) authenticating users and applications, ii) generating the knowledge graph based on the latest version of the COGITO ontologies and, iii) responding to data requests performed by other COGITO applications. Its usage is demonstrated and evaluated with example files obtained during the development phase from various online sources.

Table of contents

List of Figures

List of Tables

List of Acronyms

Term	Description
AAI	Authentication and Authorisation Infrastructure
AMQP	Advanced Message Queueing Protocol
API	Application Programming Interface
BIM	Building Information Model
COGITO	Construction Phase diGItal Twin mOdel
DB	Database
DCC	Digital Command Centre
DI	Dependency Injection
DT	Digital Twin
DTP	Digital Twin Platform
ETL	Extract, Transform and Load
GUI	Graphical User Interface
IFC	Industry Foundation Classes
IoT	Internet of Things
JMS	Java Messaging System
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MQTT	Message Queue Telemetry Transport
MVD	Model View Definition
RDF	Resource Description Framework
REST	Representational State Transfer
SHACL	SHapes Constraint Language
SOA	Service Oriented Architecture
SSE	Server Sent Events
SSO	Single-Sign On
STEP	Standard for the Exchange of Product Data
STOMP	Streaming Text-Oriented Messaging Protocol
TD	WoT Thing Description
TDD	Things Description Directory
VM	Virtual Machine
WODM	Word Order Definition and Monitoring Tool
XML	Extensible Markup Language

1 Introduction

This deliverable reports on the implementation of the DTP reflecting the outcomes of the work conducted between M7 and M18 in “T7.5 – Digital Twin Platform Development and Testing”. It builds upon the detailed architecture of “T7.1 – Digital Twin Platform Design & Interface Specification”, the overall COGITO system architecture of “T2.4 – COGITO System Architecture Design”, and the definition of the COGITO ontology network of “T3.2 – COGITO Data Model, Ontology Definition and Interoperability Design”. This work presents the first version of the core software components located in the various layers of the DTP.

In summary, the DTP consists of six core layers [1]. The *Authentication Layer* ensures that user and application access is restricted to specific roles and groups. The *Data Ingestion Layer* is responsible for loading new datasets and orchestrating the execution of the Extract, Transform and Load (ETL) and Model Checking (MC) services for generating the knowledge graph and populating the corresponding databases. At the same time, the *Data Persistence Layer* provides a cloud-based data storage solution including graph, relational and time-series databases. The *Data Management Layer* satisfies the data needs of the various COGITO applications by providing a runtime environment, which contains a set of reusable modules performing well-defined business-logic operations. for handling data requests and delivering the responses using the *Messaging Layer*. The *Data Post-Processing Layer* provides software components responsible for performing data completeness checking and handling BIM models that conform to Industry Foundation Classes (IFC) standard.

Each of the above layers contains a set of core software components performing various business logic operations to support the main objectives of the COGITO solution.

1.1 Scope and Objectives of the Deliverable

The main scope of this deliverable is to report on the implementation of the core software components located in the various layers of the DTP. These components are based either on existing open-source projects or have been designed and developed from scratch. The first release of the DTP consists of the following software components:

- **Identity Provider:** This component is part of the *Authentication Layer* and provides a fully functional identity and access management solution based on the open-source project Keycloak. It ensures that access is restricted to specific users and applications with the appropriate permissions.
- **Input Data Management component:** This component is part of the *Data Ingestion Layer* and is responsible for registering external applications, configuring user roles, creating projects, and supervising the internal business-logic operations of the DTP.
- **BIM Management component:** This component handles BIM models that conform to the Industry Foundation Classes (IFC) standard. It performs various BIM related business-logic operations such as serialising/deserialising, querying, updating, and merging IFC models.
- **Knowledge Graph Generator:** This component is part of the *Data Ingestion Layer* and is responsible for populating COGITO’s ontologies, validating the generated knowledge graph and generating the Thing Descriptions (TD)[1]. It supports the transformation of heterogeneous data such as IFC, JSON, XML and CSV coming from various input data sources.
- **Digital Twin Runtime component:** This component is part of the *Data Management Layer* and is responsible for creating and hosting application-driven modules used for orchestrating the data processing operations. It ensures that the data coming from the *Persistence Layer* are synchronised and harmonised before being forwarded to the other COGITO applications.

[1] The TD is an entity which contains meta-data of Things, where a Thing is an abstraction of physical or virtual objects.

1.2 Relation to other Tasks and Deliverables

This deliverable is the outcome of the "T7.5 – Digital Twin Platform Development and Testing", which falls under the activities of "WP7 – COGITO Digital Twin Platform". There are several dependencies of this work to other deliverables and tasks:

- The configuration of the Keycloak Identity Provider is based on the work performed in "T2.1 – Elicitation of Stakeholder Requirements" and the corresponding deliverable "D2.1 Stakeholder requirements for the COGITO system".
- The development of the first release of the DTP and its core software components is based on the work performed in "T7.1 – Digital Twin Platform Design & Interface Specification" and the corresponding deliverable "D7.1 – Digital Twin Platform Design & Interface Specification v1".
- The deployment and testing of the various ETL and MC components is based on the work performed in "T7.2 – Extraction, Transformation and Loading tools (ETL) and Model-Checking" and the corresponding deliverable "D7.3 – Extraction Transformation & Loading Tools and Model Checking v1".
- The creation and deployment of DTP's application-driven modules used for orchestrating the various data processing operations is based on the work performed in "T2.4 – COGITO System Architecture Design" and the corresponding deliverable "D2.5 – COGITO System Architecture v2".

1.3 Structure of the Deliverable

This deliverable is organised according to the identified functional requirements of the DTP. As mentioned above, the DTP is a cloud-based data integration middleware responsible for: i) providing an authentication and authorisation mechanism to COGITO users and applications, ii) loading and validating information coming from various input data sources, and iii) supervising and orchestrating data processing operations and data delivery requests. This deliverable is structured as follows:

- Section 1 summarises the outcomes of the work conducted in "T7.1 – Digital Twin Platform Design & Interface Specification" and their relationships with the development activities of "T7.5 – Digital Twin Platform Development and Testing".
- Section 2 presents the overall architecture and the deployment of the core software components in the different layers of the DTP.
- Section 3 presents the Identity Provider, which is part of the *Authentication Layer* and is responsible for authenticating COGITO's users and applications.
- Section 4 presents the first release of the core components contained in the *Data Ingestion Layer*, which are responsible for loading data from various input data sources, managing projects, populating the ontologies, and validating the knowledge graphs.
- Section 5 presents the DT Runtime component, which is part of the *Data Management Layer* and is responsible for handling the data requests and harmonising the data before being delivered to the final destinations.
- Section 6 presents the conclusions along with a release plan for the final version of the DTP.

2 Digital Twin Platform

At this phase of the project, the development activities in "T7.5 – Digital Twin Platform Development and Testing" follow the Minimum Viable Product (MVP) approach. The first release of the DTP provides all necessary features to be usable by the developers and the early users of the COGITO system [2]. During the integration, they can provide valuable feedback for developing the final release. This section describes the overall architecture and the deployment characteristics of the core software components installed in the various layers of the DTP. Some of these components are based on open-source projects, while others are developed from scratch.

2.1 Software Components Classification

As mentioned in the previous section, the DTP is responsible for loading the as-designed and as-built data, populating the ontology network, validating the knowledge graphs, and handling the data requests performed by the other COGITO applications. DTP's architecture design is based on a multi-layered approach comprising of six core layers. Each layer has different deployment characteristics and contains a set of software components as shown in Figure 1.

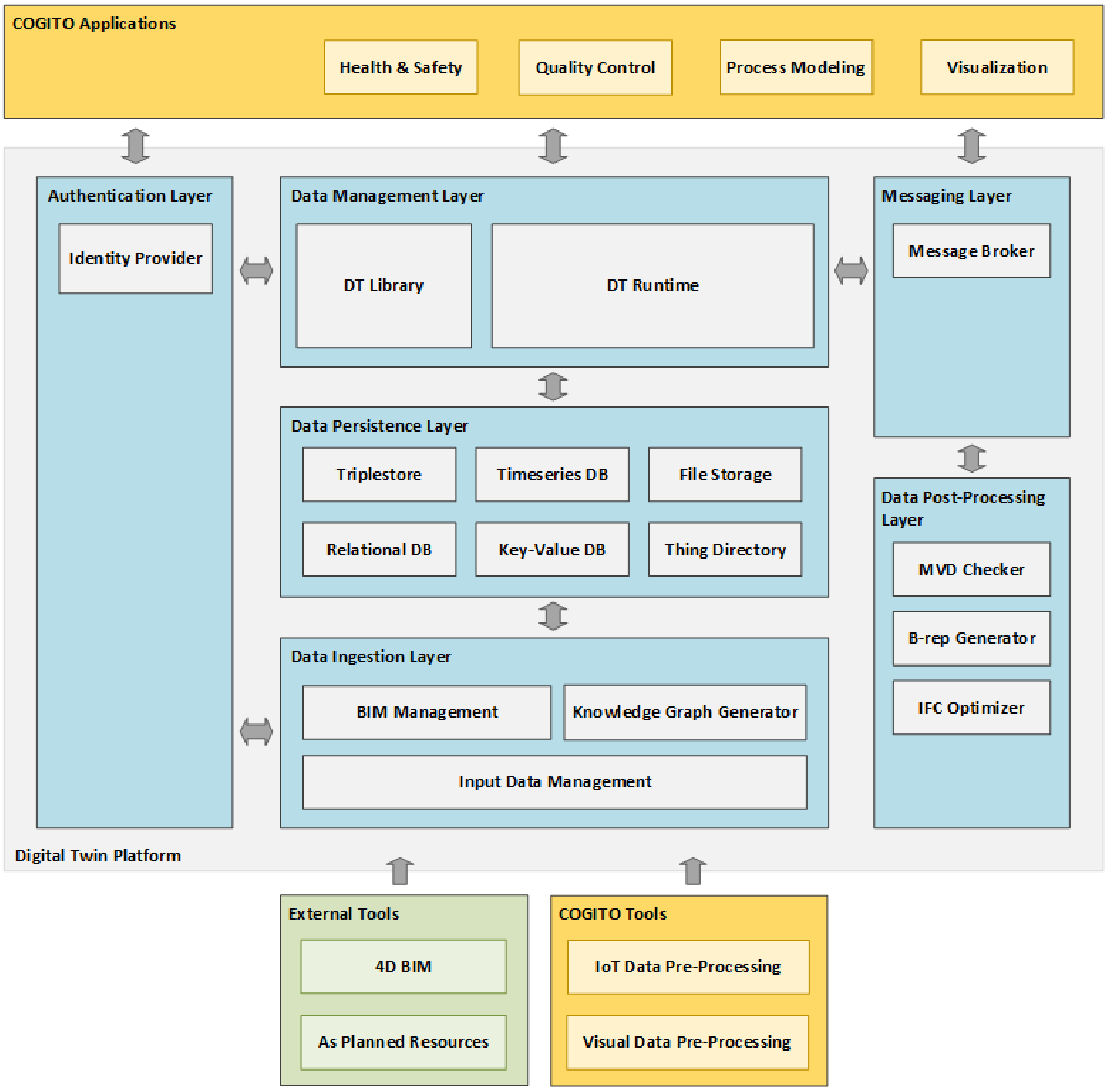


Figure 1 DTP's overall architecture along with its software components

The software components are deployed into the DTP layers implementing various business-logic operations to support the main objectives of the COGITO system. Based on their non-functional requirements they are classified into three categories:

- **Standalone Software Applications:** This category contains software components configured to run on one computational node. They provide static endpoints in various protocols (HTTP, TCP, WS) and common security standards (SSL, TLS). For instance, the Identity Provider, the Input Data Management component, the DT Runtime component, and the Message Broker are deployed in the DTP as standalone software applications.
- **Microservices:** This category contains software components configured to run on multiple computational nodes simultaneously. These components are packaged and deployed as Docker containers in a cloud-computing infrastructure, providing flexibility, high-availability, and scalability. They implement the Service-Oriented Architecture (SOA) design pattern to achieve asynchronous communication through the Messaging Layer that provides an integrated communication environment supporting various asynchronous messaging protocols. For instance, the MVD Completeness Checker, the B-rep Generator and the IFC Optimiser are deployed in the DTP as microservices.
- **Software Libraries:** This category contains low-level software packages such as parsers and application-specific algorithms used to develop other components and applications. They often provide multithread processing operations and exchange information through Programming APIs.

2.2 Deployment Environment

Currently, only the core software components have been deployed into the DTP. Figure 2 illustrates an overview regarding the actual implementation status.

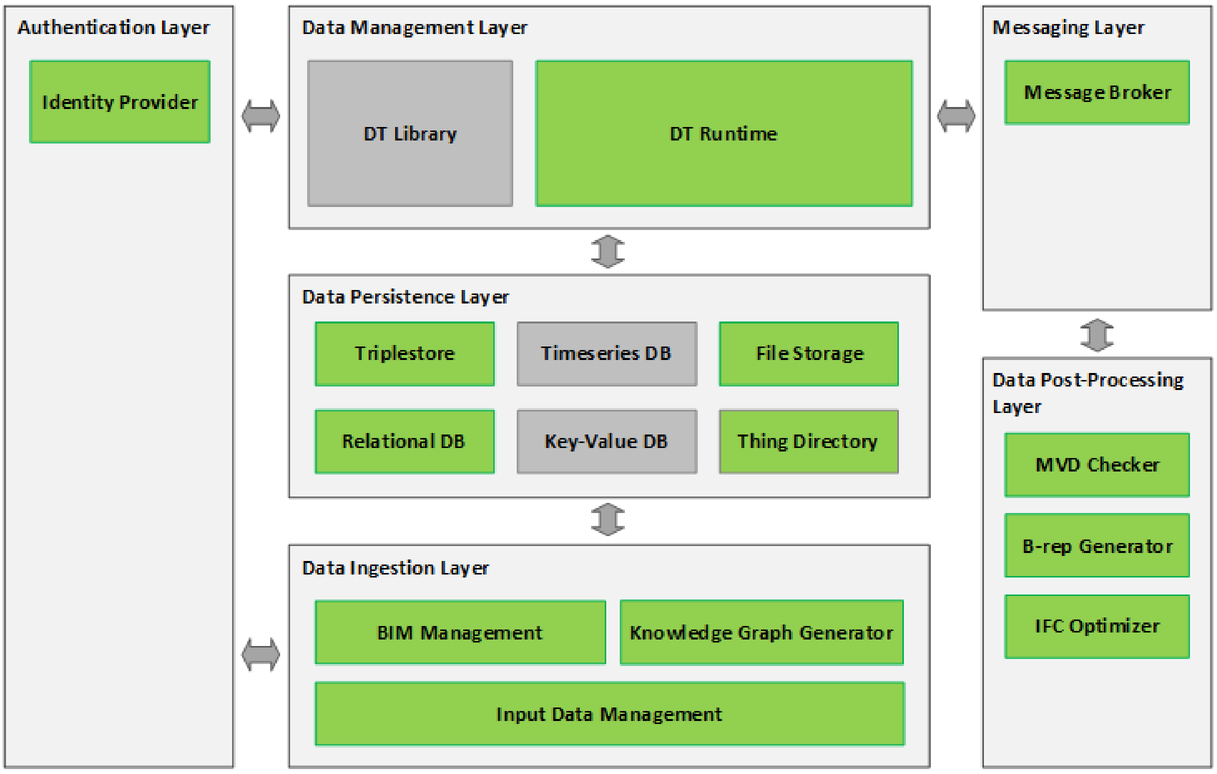


Figure 2 Deployment of the core software components following the MVP approach

As shown in Figure 2, the software components included in the first release of the DTP are highlighted in green and explained as follows:

The **Authentication Layer** contains the first release of the *Identity Provider* that offers a central identity and access management solution for COGITO users. It's based on the open-source project Keycloak, an industry-standard implementation supporting various authentication protocols such as OpenID Connect and SAML 2.0.

The **Messaging Layer** contains the first release of the *Message Broker* that offers an integrated solution enabling asynchronous bi-directional communication between DTP's software components and other COGITO applications. It is based on the open-source project Apache ActiveMQ Artemis[2] offering a high-performance message broker that supports multiple messaging protocols.

The **Data Ingestion Layer** contains the first releases of the following core software components:

- The *Input Data Management* component provides a Graphical User Interface (GUI) and a REST API which offers core functionalities such as project creation, user management, and loading of the as-planned data.
- The *BIM Management* component offers a Java-based API for serialising/deserialising, querying, updating, and merging IFC data.
- The *Knowledge Graph Generator* includes a) the first release of the various ETL tools described in "D7.3 – Extraction, Transformation & Loading Tools and Model Checking v1" and, b) the Thing Manager, responsible for supervising the data transformation processes, validating the knowledge graphs and generating the Thing Descriptions.

The **Data Persistence Layer** contains the first releases of the following core software components:

- The *File Storage System* offers a REST API for storing, retrieving, and deleting files.
- The *Relational Database* is used for storing metadata related to projects, users, roles, connected applications, and data collections.
- The *Triplestore* is used for storing the validated RDF data.
- The *Thing Description Directory* is used for storing the Thing Descriptions generated by the Thing Manager.

The **Data Management Layer** contains the first releases of the following core software components:

- The *DT Runtime component* responsible for i) hosting and supervising application-driven modules performing various data processing operations and, ii) providing configurable endpoints for the interaction with the other COGITO tools
- The *DT Library* which contains a set of ready-made coding blocks allowing external developers to create their own application-driven modules.

The **Data Post-Processing Layer** contains a set of components that use the asynchronous communication channels provided by the Messaging Layer. Depending on the complexity of the input data, these components can have higher execution times than other components. Currently, the Data Post-Processing Layer contains the following software components:

- The *MVD Checker* validates IFC data in terms of completeness and semantic consistency by applying predefined rules created by the MVD specification.
- The *B-rep Generator* uses the geometric information included in the IFC and generates triangulated B-rep solids of the structural and non-structural elements.
- The *IFC Optimiser* performs lossless compression of an IFC to speed up loading and data processing operations. It's generating a new IFC with reduced file size.

In general, the DTP consists of software components that require one running instance and other software components that can have multiple running instances simultaneously. The software components that require one running instance are deployed on Virtual Machines (VM). In this case, the open-source server Nginx is used as a reverse proxy server for forwarding the HTTP requests to the local destinations of the deployed standalone

[2] ActiveMQ Artemis https://activemq.apache.org/

applications. Currently, the Identity Provider, the Input Data Management, the File Storage System, and the DT Runtime are behind Nginx.

On the other hand, the software components that implement the SOA design pattern are deployed as containerised applications on a private cloud computing environment hosted on dedicated physical servers. In this case, the running instances exchange data with the DTP through a central messaging system using local endpoints that are not exposed to the internet. Within the first release of the DTP, the MVD Completeness Checker, the B-rep Generator and the IFC Optimiser are deployed using the Docker technology. Table 1 summarises the main characteristics and the deployment status of the software components that have been analysed previously.

Table 1 Main characteristics of DTP's software components

Core Layers	Software Components	Type	Origin	Deployment Status
Authentication Layer	Identity Provider	Standalone Application	Open source	Deployed
Data Ingestion Layer	Input Data Management	Standalone Application	Developed from scratch	Deployed
	BIM Management	Software Library	Developed from scratch	Tested
	Knowledge Graph Generator	Standalone Application	Developed from scratch	Tested
Data Persistence Layer	Relational DB (data model)	Standalone Application	Developed from scratch	Deployed
	File Storage System	Standalone Application	Developed from scratch	Deployed
	Triplestore (server)	Standalone Application	Open source	Deployed
	Thing Directory	Standalone Application	Developed from scratch	Tested
	Timeseries DB (server)	Standalone Application	Open source	Pending
	Key-Value DB (server)	Standalone Application	Open source	Pending
Data Management Layer	DT Runtime	Standalone Application	Developed from scratch	Deployed
	DT Library	Standalone Application	Developed from scratch	Tested
Data Post-Processing Layer	MVD Checker	Microservice	Partially developed	Deployed
	B-rep Generator	Microservice	Partially developed	Deployed
	IFC Optimiser	Microservice	Partially developed	Deployed
Messaging Layer	Message Broker	Standalone Application	Open source	Tested

3 Authentication and Authorisation Infrastructure

The various COGITO applications can be classified based on their functional and non-functional requirements. Some applications offer a Graphical User Interface (GUI) and require a system to authenticate and authorise the COGITO users. The Authentication Layer provides an Authentication and Authorization Infrastructure (AAI), allowing the DTP to manage the users and their roles by providing various functionalities to COGITO system, such as registration, password recovery, authentication, and authorisation endpoints.

3.1 Identity Provider

The AAI solution of the DTP relies on the Keycloak[3] open-source identity and access management solution. Keycloak is an industry-standard implementation for identity and access management supporting various protocols such as the OpenID Connect and SAML 2.0. Within COGITO, the OpenID Connect protocol is used, offering Single Sign-On (SSO) capabilities to the COGITO applications.

3.1.1 Overview

The Identity Provider has been deployed as standalone software application behind the Nginx reverse proxy server and offers an integrated solution for access management. The authentication process of a user is divided into three main parts:

1. The COGITO application redirects the user to Keycloak to perform the authentication process.
2. The user provides the credentials, and if the authentication is successful, Keycloak redirects the user back to the COGITO application.
3. The COGITO application performs a new request to the Keycloak service for retrieving the Access, ID, and Refresh tokens.

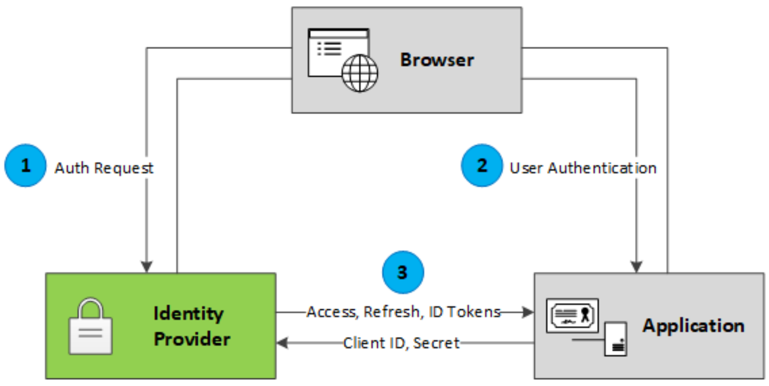


Figure 3 Identity Provider's user authentication process

3.1.2 Technology Stack and Implementation Tools

The Identity Provider is based on the open-source project Keycloak. The work conducted in T7.5 is related to the installation and configuration of the service. The service is behind the Nginx reverse proxy server, which handles the SSL encryption and forwards the HTTP requests to the local destination.

Table 2 Libraries and Technologies used in the Identity Provider

Technology Name	Version	License
Keycloak	16.1.1	Apache Licence 2.0
Nginx	1.20.2	BSD
Certbot	1.22	Apache Licence 2.0

[3] Keycloak Identity and Access Management https://www.keycloak.org

3.1.3 API Documentation

The Keycloak server is configured to provide i) an *Authentication REST API* responsible for granting access to users based on their credentials and ii) an *Admin REST API* that allows administrator users to access the management resources that are provided by Keycloak's Admin Console.

3.1.3.1 Authentication REST API

Each COGITO application that requires authentication through the Identity Provider has a unique ClientID and a Secret generated by DTP developers during the configuration of Keycloak. The endpoints and the parameters required for preparing external applications to connect to the Keycloak server are listed in Table 3.

Table 3 Identity Provider's Authentication API

Name	Method	Endpoint
OpenID Endpoint	GET	https://auth.cogito-project.com/auth/realms/cogito/.well-known/openid-configuration
Auth URL	GET	https://auth.cogito-project.com/auth/realms/cogito/protocol/openid-connect/auth
Access Token URL	GET	https://auth.cogito-project.com/auth/realms/cogito/protocol/openid-connect/token

The **OpenID Endpoint** provides the main configuration parameters of the authentication server. The response is a JSON object which includes all available endpoints, scopes, and signing algorithms. The **Auth URL** is the endpoint of the authorisation server. It is used to retrieve an authorization code which is included in the redirected URL after a successful login. On the other hand, the **Access Token URL** is the endpoint of the authentication server. It is used by COGITO application to request the Access, ID and Refresh Tokens and it uses as parameter the authorisation code.

3.1.3.2 Admin REST API

The Identity Provider offers a fully functional Admin REST API which offer access to all features provided by the Keycloak's Admin Console. This API is mainly used within the Input Data Management component by the DTP Identity Manager for retrieving the complete list of registered users and roles. This information is required for assigning or removing roles from the registered users. The endpoints and their parameters are listed in Table 4.

Table 4 Identity Provider's Admin API

Name	Method	Endpoint
Get Users	GET	https://auth.cogito-project.com/auth/admin/realms/cogito/users
Get Roles	GET	https://auth.cogito-project.com/auth/admin/realms/cogito/roles
Get User Roles	GET	https://auth.cogito-project.com/auth/admin/realms/cogito/users/{user-id}/role-mappings/realm
Assign a Role to User	POST	https://auth.cogito-project.com/auth/admin/realms/cogito/users/{user-id}/role-mappings/realm
Remove a Role from a User	DELETE	https://auth.cogito-project.com/auth/admin/realms/cogito/users/{user-id}/role-mappings/realm

The requests Assign a Role to User (POST) and Remove a Role from a User (DELETE) require the list of the selected roles as body parameter. Each user role is defined as JSON object that contains the Role Id and the Role Name. The detailed documentation[4] and the Postman collection of this API are available online.

3.1.4 Usage Walkthrough

The COGITO applications automatically redirect the unauthenticated users to the main page of the Identity Provider. If the user has valid credentials, he/she can proceed with the authentication process using the login page as shown in Figure 4.

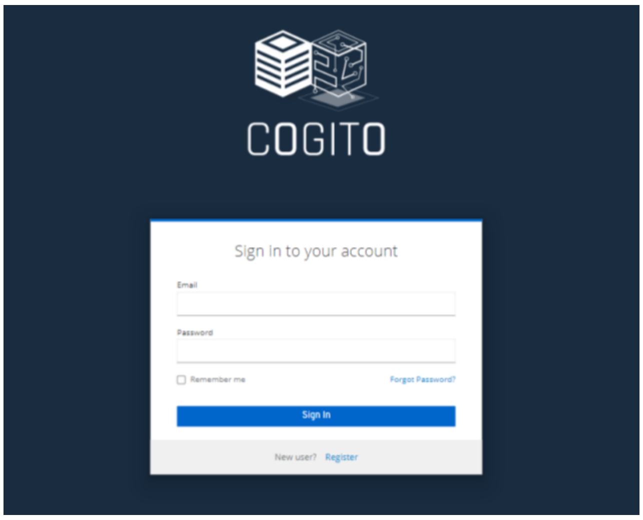


Figure 4 Identity Provider's user login page

Otherwise, a registration process is required. The Identity Provider offers a registration page for registering new users as shown in Figure 5.

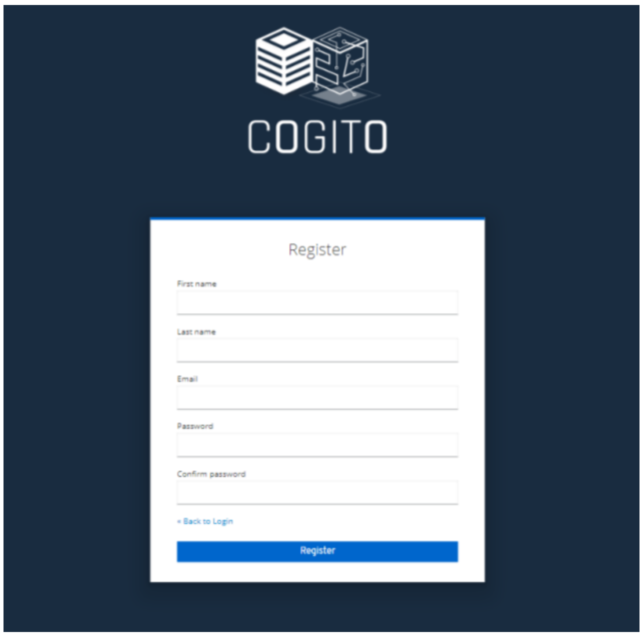


Figure 5 Identity Provider's user registration page

After the registration process, the users still have no access to the COGITO system. The DTP Identity Manager should assign the proper roles to grant users access to COGITO applications. Furthermore, each user has access

[4] Digital Twin Platform API documentation https://api.cogito-project.com

to Keycloak's Account Management Console, which provides access to a basic account management system as shown in Figure 6. The URL[5] of this console is open to the internet and offers an alternative way for users to register without accessing the COGITO applications. It also provides pages for updating the users account information and resetting their password.

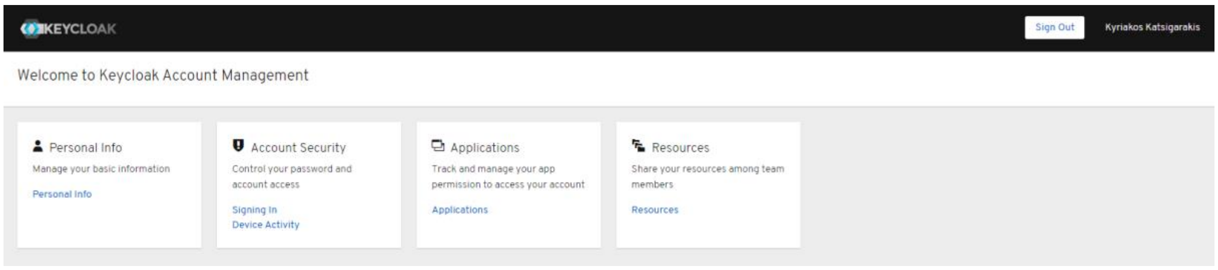
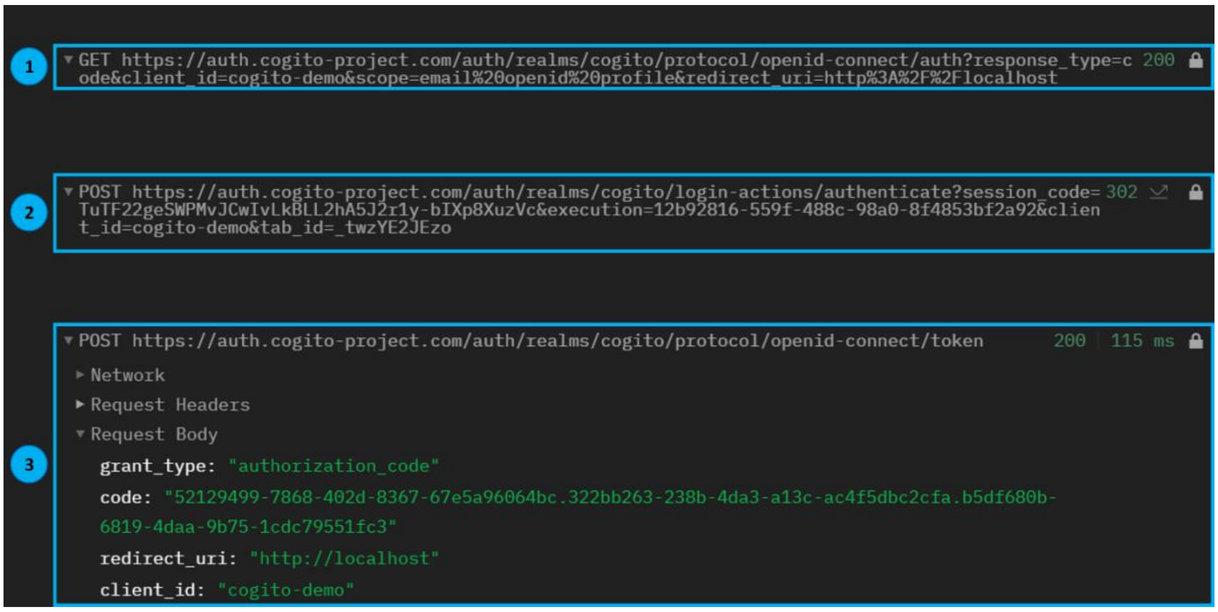

KEYCLOAK
Sign Out
Kyriakos Katsigarakis
Welcome to Keycloak Account Management
Personal Info
Manage your basic information
Personal Info
Account Security
Control your password and account access
Signing In
Device Activity
Applications
Track and manage your app permission to access your account
Applications
Resources
Share your resources among team members
Resources

Figure 6 Identity Provider's Account Management console

3.1.5 Application Example

As shown in the example of Figure 7, the user authentication process has three steps. Initially, the COGITO application redirects the user **(1)** to Keycloak to perform the authentication process. Next, the user provides the credentials, and if the authentication is successful, Keycloak redirects **(2)** the user back to the COGITO application. Finally, the COGITO application performs a new POST request to the Keycloak **(3)** for retrieving the Tokens.



```
1  ▼ GET https://auth.cogito-project.com/auth/realms/cogito/protocol/openid-connect/auth?response_type=c 200
     ode&client_id=cogito-demo&scope=email%20openid%20profile&redirect_uri=http%3A%2F%2Flocalhost

2  ▼ POST https://auth.cogito-project.com/auth/realms/cogito/login-actions/authenticate?session_code= 302
     TuTF22geSWPMvJCwIvLkBLL2hA5J2r1y-bIXp8XuzVc&execution=12b92816-559f-488c-98a0-8f4853bf2a92&clien
     t_id=cogito-demo&tab_id=_twzYE2JEzo

   ▼ POST https://auth.cogito-project.com/auth/realms/cogito/protocol/openid-connect/token      200   115 ms
     ▶ Network
     ▶ Request Headers
     ▼ Request Body
3      grant_type: "authorization_code"
       code: "52129499-7868-402d-8367-67e5a96064bc.322bb263-238b-4da3-a13c-ac4f5dbc2cfa.b5df680b-
       6819-4daa-9b75-1cdc79551fc3"
       redirect_uri: "http://localhost"
       client_id: "cogito-demo"
```

Figure 7 Identity Provider's user authentication example

3.1.6 Licensing

The Identity Provider is based on the open-source project Keycloak and provides an identity and access management solution which is released under the **Apache License 2.0**.

3.1.7 Installation Instructions

This component is deployed as standalone software application in the Authentication Layer. It provides various endpoints which are open to the internet. No file download, installation or maintenance is required by the COGITO users.

[5] Account Management Console https://auth.cogito-project.com/auth/realms/cogito/account/#/

3.1.8 Development and Integration Status

As mentioned previously, the work performed in this component was focused mainly on installation and configuration activities. Currently, the service is fully functional, and the configuration is aligned with the outcomes of the "T2.1 – Elicitation of Stakeholder Requirements" and "T2.4 – COGITO System Architecture Design". The stakeholders identified in T2.1 have been introduced in Keycloak as different roles [3]. Furthermore, the COGITO applications that include user authentication as a functional requirement are registered to Keycloak as Clients.

3.1.9 Requirements Coverage

The Identity Provider covers some of DTP's functional and non-functional requirements defined in T2.4 and the corresponding deliverable "D2.5 – COGITO System Architecture v2". The functional and non-functional requirements which are related to this component are presented in Table 5. The Req-1.1 is fully covered thanks to the central identity and access management solution. Additionally, Req-2.3 is achieved due to Nginx server which offers SSL encryption.

Table 5 Identity Provider's requirements coverage

Type	ID	Description	Status
Functional	Req-1.1	Authenticates COGITO users and applications	Achieved
Non-Functional	Req-2.3	Security	Achieved

3.1.10 Assumptions and Restrictions

The first release of the Identity Provider has been deployed under certain assumptions and restrictions, listed below:

- It has been configured to manage a single realm instance dedicated to the COGITO project. Currently, all tests regarding the authentication mechanism have been performed using the IDM component and Postman. The COGITO tool developers have access to the credentials of a demo client that they can use to test the authentication and authorisation system.
- The roles and the clients are configured based on the outcomes of the T2.1 and T2.4. New roles or clients can be added or removed during the integration phase based on the needs.
- It supports email notifications allowing users to reset their passwords. In the final release, this functionality will include additional email notifications related to security issues.

4 Project Creation and Ontology Population

One of the core functionalities of the DTP, is to load the as-planned data and to populate the corresponding knowledge graphs and databases. Before the construction works start, the as-planned data of a project are loaded into the DTP through the Input Data Management component. Within COGITO, the as-planned data come from three different sources: a) BIM authoring tools such as Autodesk Revit and Autodesk Civil 3D, providing the 3D BIM model along with the 4D semantics; b) project management tools such as Microsoft Project and Primavera P6, providing the detailed schedule of the construction works; and c) ERP solutions, providing the as-planned resources.

When the as-planned data are available, data completeness checking, and file-size optimisation operations are performed to ensure that the input files meet the requirements of the various transformation tools. Once the data are ready, the Knowledge Graph Generator (KGG) can generate the complete knowledge graph which represents a network of physical and virtual entities (i.e., workers, machinery, equipment, zones, building elements, activities) and illustrates the relationships between them as shown in the example of Figure 8 [4]. Furthermore, it generates and stores the corresponding Thing Descriptions into the Thing Description Directory. A Thing Description is an object which follows the WoT Thing Description specification and provides a set of meta-data and interfaces of Things, where a Thing is an abstraction of physical or virtual entities.

Figure 8 Example of a knowledge graph generated by the KGG component

In this section, we present the first release of the components involved in the processes of project initialisation, data loading and ontology population and validation.

4.1 Input Data Management component

The first release of the Input Data Management (IDM) component has been deployed in the Data Ingestion Layer. It provides a web-based GUI that allows users with proper permissions to create new projects, assign users to projects, and upload the as-planned data. Furthermore, it offers a REST API allowing other COGITO applications to access information related to the projects, users, and roles.

4.1.1 Prototype Overview

The IDM component is deployed as a standalone application into the Data Ingestion Layer. It uses modern web technologies to deliver a rich GUI and offers authorised access to COGITO users through the Identity Provider.

The implementation of this component is based on Spring Boot technology which is built on top of the Spring Framework. It contains an embedded version of the Apache Tomcat which hosts all required software packages. The web application follows the Model-View-Control (MVC) approach and uses the Spring Security Framework with the Spring Keycloak Adapter for managing the access policies of the COGITO users.

It also provides a REST API allowing the COGITO applications to interact with the DTP for project creation, user management and loading as-planned data. Figure 9 shows the main interactions between the IDM component and the other entities of the DTP.

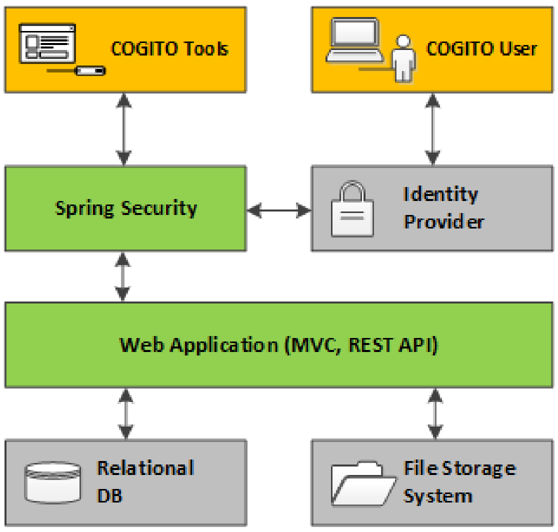


Figure 9 IDM's sub-components along with their interactions

4.1.2 Technology Stack and Implementation Tools

The IDM component is based completely on open-source technologies. As mentioned previously, it is built on top of the Spring Framework and is deployed as standalone application. It's installed behind the Nginx reverse proxy server, which handles the SSL encryption and forwards the HTTP requests to the correct destination.

Table 6 Libraries and Technologies used in the Input Data Management component

Technology Name	Version	License
Spring Framework	5.3.1	Apache Licence 2.0
Spring Boot	2.3.0	Apache Licence 2.0
Spring Security	5.5.0	Apache Licence 2.0
Thymeleaf	3.0.15	Apache Licence 2.0
Hibernate	5.6.9	LGPL 2.1
MySQL	8.0.24	GPLV2
Nginx	1.20.2	BSD
Certbot	1.22	Apache Licence 2.0

4.1.3 Input, Output and API Documentation

The IDM component provides a REST API enabling the COGITO applications to access data related to the projects. This API is used by various COGITO applications such as the Digital Command Centre (DCC), the Process Modeling and Simulation (PMS) and the Work Order Definition and Monitoring (WODM) to retrieve data related to the available projects and their users. The endpoints along with their parameters are listed in Table 7.

Table 7 Input Data Management component's REST API

Name	Method	Endpoint	PP = Path Parameter
			FP = Form Parameter

Get all Projects	GET	https://dtp.cogito-project.com/api/projects	N/A
Get a Project	GET	https://dtp.cogito-project.com/api/projects/{projectId}	(PP) projectId: GUID
Get all Users of a Project	GET	https://dtp.cogito-project.com/api/projects/{projectId}/users	(PP) projectId: GUID
Get all Properties of a Project	GET	https://dtp.cogito-project.com/api/projects/{projectId}/properties	(PP) projectId: GUID
Get all Users	GET	https://dtp.cogito-project.com/api/users	N/A
Get all Roles of a User	GET	https://dtp.cogito-project.com/api/users/{userId}/roles	(PP) userId: GUID
Get a User	GET	https://dtp.cogito-project.com/api/users/{userId}	(PP) userId: GUID
Get all Projects of a User	GET	https://dtp.cogito-project.com/api/users/{userId}/projects	(PP) userId: GUID

The following are examples of JSON responses returned by the GET requests of IDM's REST API. For some requests, the responses have no body content. In this case, the HTTP status code is used: i) the "*200 - OK*" is returned if the request is successful, the "*404 - Not Found*" is returned if the resource is unavailable and iii) the response "*400 - Bad Request*" is returned if an error has occurred.

Get all Projects

```
[
    {
        "name": "DEMO_01",
        "id": "1cf55b98-db69-46f8-a64e-a531d512d4d3"
    },
    {
        "name": "DEMO_02",
        "id": "c69407ff-2f81-4138-85d1-21a5e9e24550"
    }
]
```

Get a Project

```
{
    "name": "DEMO_01",
    "description": "4D BIM provided by UEDIN",
    "id": "1cf55b98-db69-46f8-a64e-a531d512d4d3"
}
```

Get all Users of a Project

```
[
    {
        "firstname": "Kyriakos",
        "id": "1b20a241-c5fe-422e-8043-01461da1a2c3",
        "email": "katsigarakis@gmail.com",
        "lastname": "Katsigarakis"
    }
]
```

Get all Properties of a Project

```
[
    {
        "name": "country",
        "value": "GR"
    },
    {
        "name": "datastore",
        "value": "https://his.cogito-project.com/demo_01"
    },
    {
        "name": "triplestore",
        "value": "https://triplestore.cogito-project.com/demo_01"
    }
]
```

Get all Users

```
[
    {
        "firstname": "Kyriakos",
        "id": "1b20a241-c5fe-422e-8043-01461da1a2c3",
        "email": "katsigarakis@gmail.com",
        "lastname": "Katsigarakis"
    },
    {
        "firstname": "Georgios",
        "id": "685d8e70-8f7d-4c23-97f9-69dffd52677d",
        "email": "g.lilis@ucl.ac.uk",
        "lastname": "Lilis"
    },
    {
        "firstname": "Frédéric",
        "id": "47e5af5c-832c-472c-abcb-88d3bc80efcc",
        "email": "f.bosche@ed.ac.uk",
        "lastname": "Bosché"
    },
    {
        "firstname": "Giorgos",
        "id": "4d3f182e-997a-4d63-a843-81d498d2240c",
        "email": "g.giannakis@hypertech.gr",
        "lastname": "Giannakis"
    }
]
```

Get all Roles of a User

```
[
    {
        "name": "DTP Developer",
        "id": "95f260f6-880a-4899-abfc-0777a07e2a36"
    },
    {
        "name": "DTP Project Manager",
        "id": "90080412-bd71-4397-92a0-d01415c463b7"
    }
]
```

Get a User

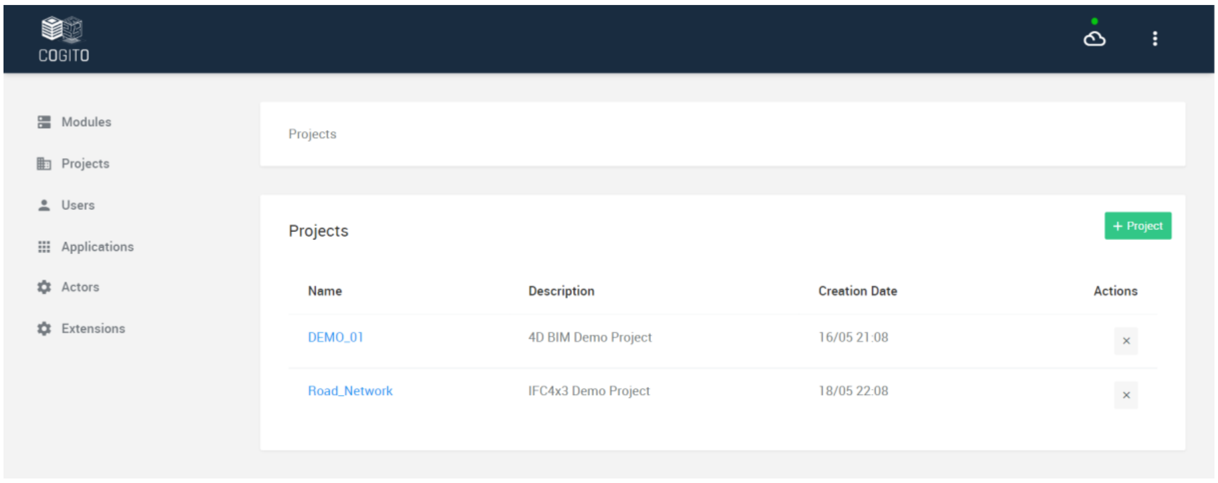
```
{
    "firstname": "Kyriakos",
    "id": "1b20a241-c5fe-422e-8043-01461da1a2c3",
    "email": "katsigarakis@gmail.com",
    "lastname": "Katsigarakis"
}
```

Get all Projects of a User

```
[
    {
        "name": "DEMO_01",
        "id": "1cf55b98-db69-46f8-a64e-a531d512d4d3"
    }
]
```

4.1.4 Usage Walkthrough

As shown in Figure 10, when the COGITO users sign into the IDM component through the DTP's Identity Provider, they see the table of projects created in the past and some of their details, such as the project description and creation date. On the right side of the table, the Actions column contains a button for deleting a project after confirmation. By clicking on any project, they can see further information and perform additional actions. Depending on their roles, users may have access to projects they created or the entire list. The role DTP Project Manager provides access to all projects, while the role Project Manager to projects created by the active user.

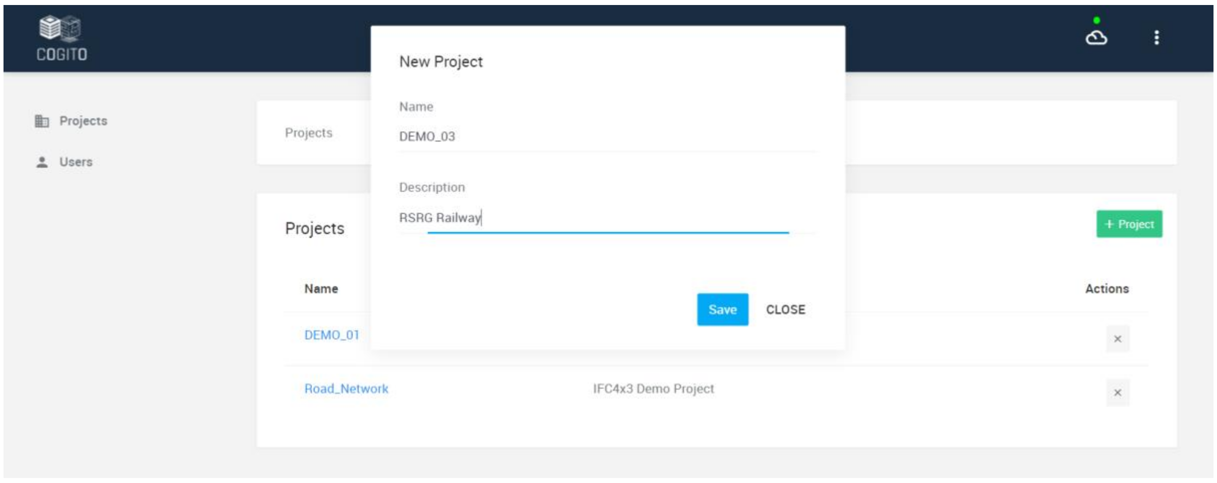


Projects

Name	Description	Creation Date	Actions
DEMO_01	4D BIM Demo Project	16/05 21:08	×
Road_Network	IFC4x3 Demo Project	18/05 22:08	×

Figure 10 View of all user's projects

On the same page, the users can create a new project by clicking the "+ Project" button. As shown in Figure 11 a modal popup window appears which allows the users to provide basic information such as the project name and the project description. At any time, users can cancel the project creation process by clicking on the Close button.



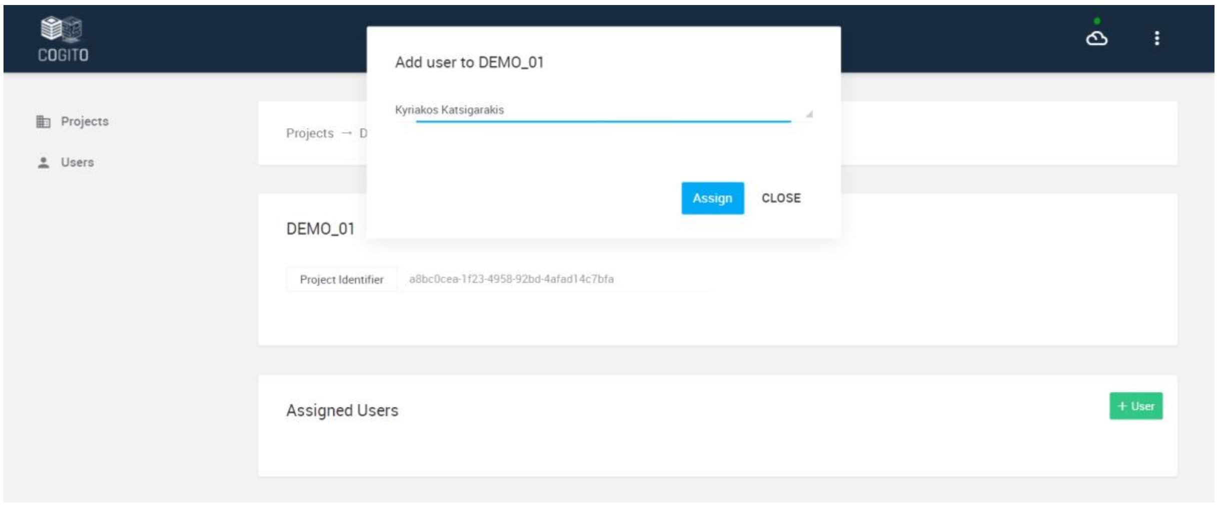
New Project

Name: DEMO_03

Description: RSRG Railway

Figure 11 Creation of a new project

Once the project is created, the users can proceed with the configuration. They need to define the project members and, if required, add global project parameters. On the main page of a project, the users with proper roles can assign project members by clicking the "+ User" button. As shown in Figure 12, a modal window containing a combo box element with all registered users appears. The assignment of the selected user with the project is done by clicking the Assign button. At any time, users can cancel the assignment process by clicking on the Close button.



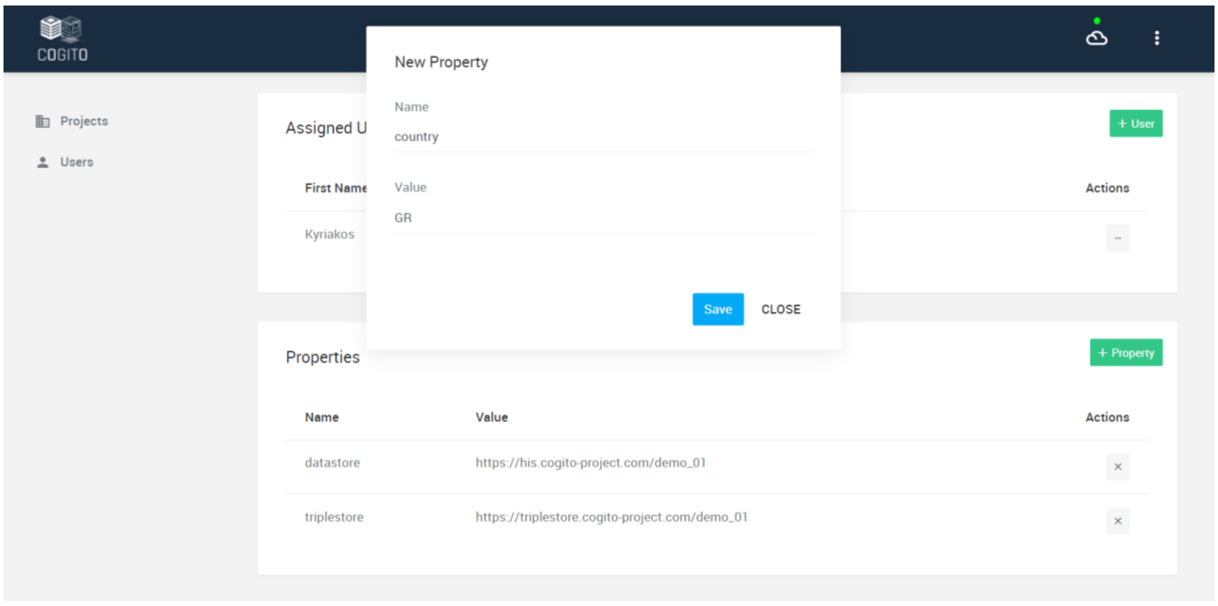
Add user to DEMO_01

Kyriakos Katsigarakis

Project Identifier: a8bc0cea-1f23-4958-92bd-4afad14c7bfa

Figure 12 Assignment of registered users to a project

Following the assignment process, users with proper roles can create global properties by clicking the "+ Property" button. As shown in Figure 13, a modal popup window appears allowing users to fill in the property name and its value. In the first release of the IDM component, there are no restrictions on the creation of project properties.



New Property

Name: country

Value: GR

Name	Value	Actions
datastore	https://his.cogito-project.com/demo_01	×
triplestore	https://triplestore.cogito-project.com/demo_01	×

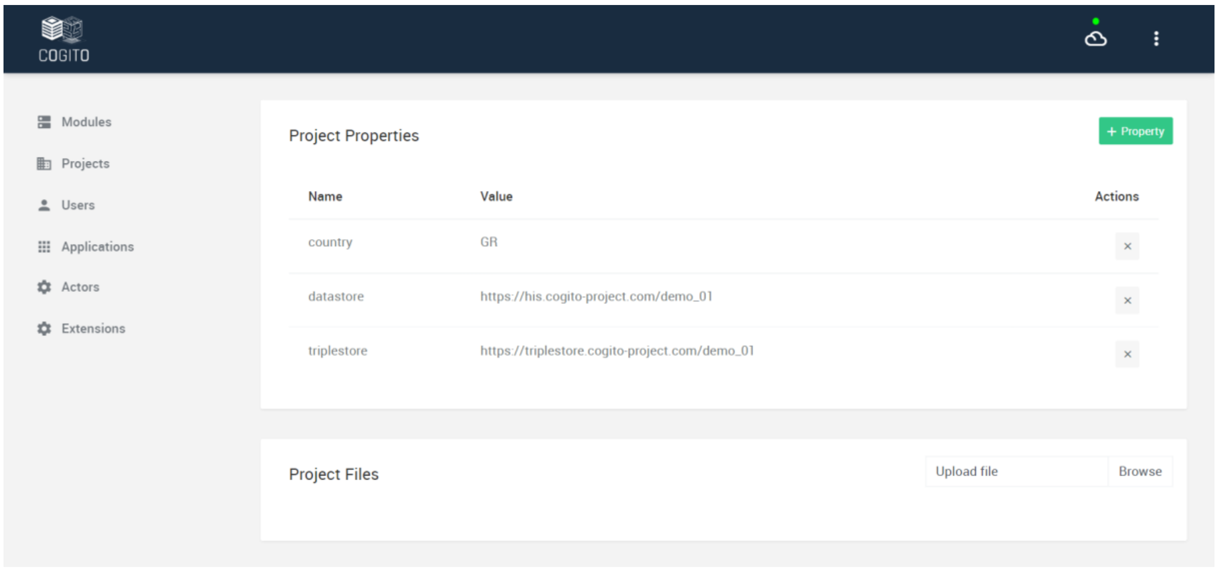
Figure 13 Creation of a new property in a project

On the same page, users can upload the as-planned data by clicking the "Browse" button as shown in Figure 14. Currently, the as planned data consist of three different files:

1. The 3D BIM is provided in IFC, particularly in versions IFC4 or IFC4x3. The DTP has specific data requirements described in "D7.3 - Extraction, Transformation & Loading Tools and Model Checking". For instance, the 4D data for each element, such as the construction zone and the task identifier.
2. The construction schedule is provided in CSV or MS Project XML file format, which contains the tasks and their properties.
3. The as-planned resources are provided in a CSV file format, containing the resource types and their properties.

The detailed data structures of the as-planned data are presented in "D7.3 – Extraction, Transformation & Loading Tools and Model Checking". Upon the upload process, the IDM will store these files into DTP's File

Storage System, and it will send an internal notification to DTP's Data Management Layer for initialising a sequence of various data processing operations.



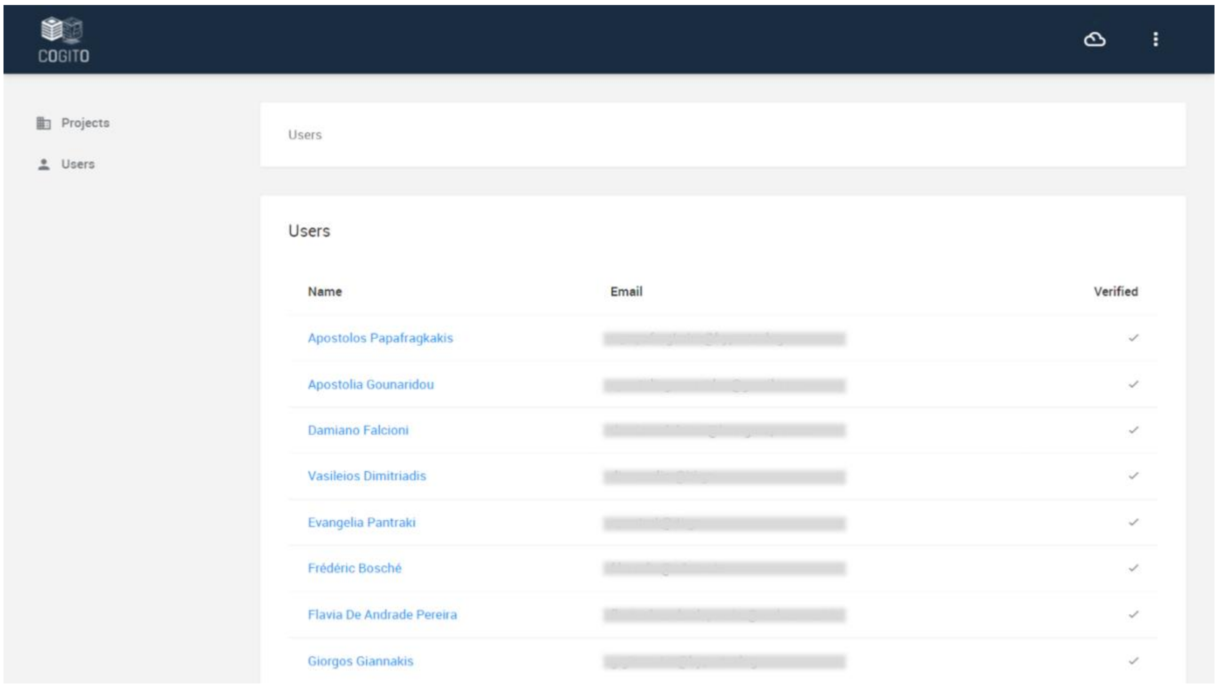
Project Properties

Name	Value	Actions
country	GR	×
datastore	https://his.cogito-project.com/demo_01	×
triplestore	https://triplestore.cogito-project.com/demo_01	×

Project Files

Figure 14 Uploading the as-planned data using the IDM component

In the left sidebar of the IDM application, the link "Users" is visible only to users who have the role of DTP Identity Management, as shown in Figure 15. This role allows users to manage the access policies of the registered users.



Users

Name	Email	Verified
Apostolos Papafragkakis	[redacted]	✓
Apostolia Gounaridou	[redacted]	✓
Damiano Falcioni	[redacted]	✓
Vasileios Dimitriadis	[redacted]	✓
Evangelia Pantraki	[redacted]	✓
Frédéric Bosché	[redacted]	✓
Flavia De Andrade Pereira	[redacted]	✓
Giorgos Giannakis	[redacted]	✓

Figure 15 View of all registered users

By clicking on any account, they can view its data and roles. The users with the role DTP Identity Management can assign roles to any account by clicking the "+ Role" button. As shown in Figure 16, a modal window that contains a combo box element with the available roles appears. It is worth mentioning that all roles are predefined and configured directly in Keycloak. The assignment of the selected role with the user account is done by clicking the Assign button. At any time, users can cancel the assignment process by clicking on the Close button.

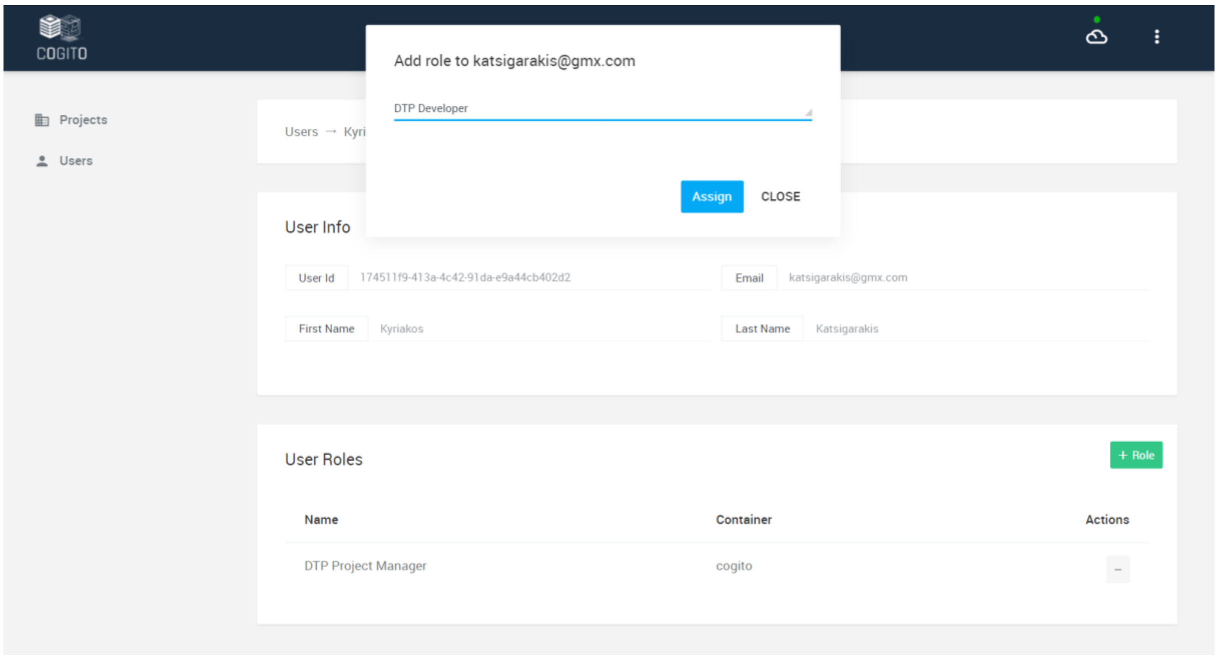


Figure 16 Assigning a role to the user account

4.1.5 Licensing

The IDM is a closed source component. The lead group in charge of development of the DTP is receiving requests for providing access to the component within the project.

4.1.6 Installation Instructions

This component is deployed as standalone software application in the Data Ingestion Layer. It provides a rich GUI and a REST API which are open to the internet. No file download, installation or maintenance is required by the COGITO users.

4.1.7 Development and Integration Status

As mentioned previously, the first release of the IDM component has been deployed in the Data Ingestion Layer. Currently, the service is fully functional, and it provides all core functionalities that have been identified in "D7.1 – Digital Twin Platform Design & Interface Specification v1". The WP7 technical partners will provide some minor improvements in the final release of the component. For instance, a notification system will notify the users via email when they join/leave a project or have a new role. Furthermore, there are some discussions within the consortium for improving the GUI and the end-user experience.

4.1.8 Requirements Coverage

This component covers some of the DTP's functional and non-functional requirements defined in T2.4 and the corresponding deliverable "D2.5 – COGITO System Architecture v2". The functional and non-functional requirements related to the IDM component are presented in Table 8. The Req-1.2 is fully covered by the embedded web-based application that enables users to create, load and manage projects, users, and files. Additionally, the Req-2.3 and Req-2.4 are achieved due to Nginx reverse proxy server which offers SSL encryption and load balancing capabilities.

Table 8 IDM component's Requirements Coverage

Type	ID	Description	Status

Functional	Req-1.2	Receives as-planned data (BIM models, construction schedule, available resources)	Achieved
Non-Functional	Req-2.3	Security	Achieved
	Req-2.4	High availability	Achieved

4.1.9 Assumptions and Restrictions

The first release of the IDM component has been deployed under certain assumptions and restrictions, listed below:

- It has been developed from scratch following the MVP approach providing the basic functionalities essential for creating and managing projects. The final release will provide more features and an improved GUI.
- It supports IFC4x3, although more tests and refinements are required. It has been tested using a sample file of a road network.
- Currently, it doesn't support email notifications. The final release, the IDM will send emails notifying users for new assignments to projects and roles.

4.2 Knowledge Graph Generator

The Knowledge Graph Generator (KGG) component is part of the Data Ingestion Layer and oversees: i) the execution of the various ETL tools, which are responsible for transforming COGITO's as-planned input files to Resource Description Framework[6] (RDF) data, ii) the validation of the semantic links included in these RDF data and, iii) the generation of the Thing Descriptions. The ETL tools included in the KGG component have been described in "D7.3 - Extraction, Transformation & Loading Tools and Model Checking". The first release of the KGG contains various sub-components and ETL tools that have been packaged as containerised services using Docker and deployed in a cloud computing environment.

In this section, we present the first release of the core KGG's components involved in the orchestration of the ETL tools and the generation of the Thing Descriptions.

4.2.1 Prototype Overview

The main functionalities of the KGG component are the population and validation of COGITO's knowledge graph and the generation of the Thing Descriptions. For this purpose, the KGG contains two core sub-components, the Thing Manager and the Wrapper module as shown in Figure 17.

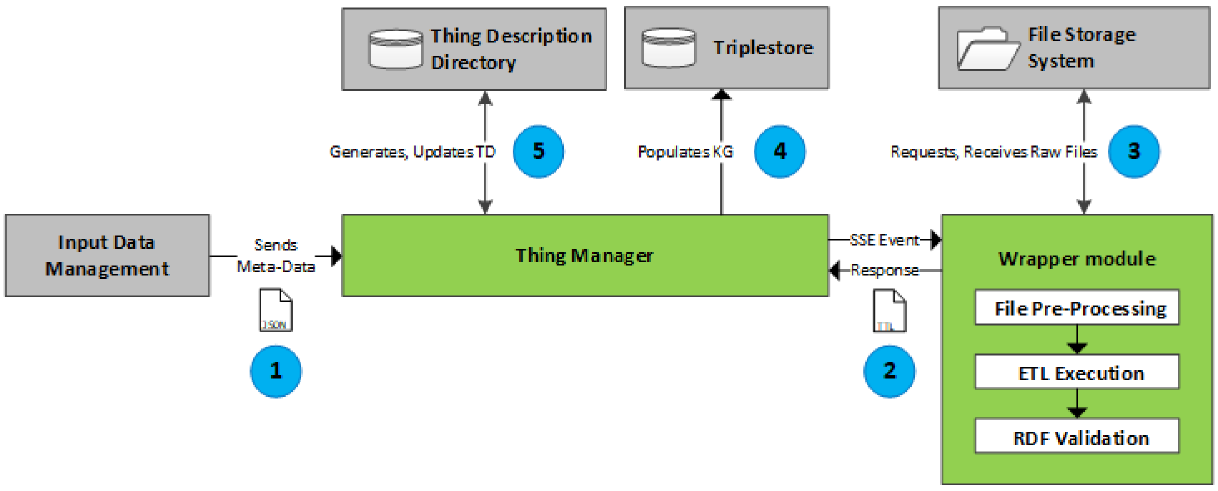


Figure 17 High-level Architecture of Knowledge Graph Generation

A quick overview of Figure 17, shows that the starting point for generating the final knowledge graph and the Thing Descriptions is the IDM component. The users with proper roles upload the as-planned data of a project via IDM's GUI. Once the files are loaded and stored in the File Storage System, the IDM component performs a request **(1)** to the Thing Manager for creating an empty container for the specific project. Next, the Thing Manager sends asynchronous messages to the Wrapper module **(2)** via the Server-Sent Event (SSE) protocol for invoking the various ETL tools. The Wrapper module is responsible for i) retrieving the as-planned input data files **(3)** from the File Storage System, ii) if required, executing pre-processing operations on the files, iii) triggering the executions of the various ETL tools and, iv) performing data validation operations on the generated RDF data by applying SHACL rules. Finally, the Thing Manager stores the returned TTL files to the Triplestore **(4)** and generates and stores the corresponding Thing Descriptions to the Thing Description Directory (TDD) **(5)**.

4.2.2 Technology Stack and Implementation Tools

The Thing Manager and the Wrapper module, which are the core sub-components of the KGG, have been developed in Python using a set of open-source technologies and libraries listed in Table 9.

[6] The RDF is a general framework for representing interconnected data on the web https://www.w3.org/RDF/

Table 9 Libraries and Technologies used in the Thing Manager and the Wrapper module

Technology Name	Version	License
Flask	2.1.1	BSD 3-Clause License
requests	2.27.1	MIT License
wheezy.template	3.1.0	MIT License
Flask-SSE	1.0.0	MIT License
APScheduler	3.9.1	MIT License
rdflib	6.1.1	BSD 3-Clause License
SPARQLWrapper	2.0.0	W3C® SOFTWARE NOTICE AND LICENSE

4.2.3 Input, Output and API Documentation

The Thing Manager component provides a REST API allowing other DTP components, such as the IDM and the DT Runtime, to trigger the various generation and validation operations and to retrieve the generated RDF data in TTL and the corresponding Thing Descriptions. The endpoints containing the project extension are relative to the use of a project in a 1-1 relation with the project defined in the IDM component, which is the one reflected in the ontology. In addition, this component will be of internal use for the DTP, so there will be no collision with other endpoints that contain the term project in their definition. The endpoints provided along with their parameters are listed in Table 10.

Table 10 Thing Manager's REST API

Description	Method	Endpoint	PP = Path Parameter
			FP = Form Parameter

Creates a new project, its respective triples and thing description	POST	/project/{projectId}	(PP) projectId: GUID (FP) name: Text (FP) description: Text
Updates an existing project, its respective triples and thing description	PUT	/project/{projectId}	(PP) projectId: GUID (FP) name: Text (FP) description: Text
Deletes an existing project, its respective triples and thing description, and the thing descriptions associated to it in cascade mode	DELETE	/project/{projectId}	(PP) projectId: GUID
Retrieves the thing description of an existing project	GET	/project/{projectId}	(PP) projectId: GUID
Adds files to an existing project, creates respective triples and thing descriptions	POST	/project/{projectId}/file	(PP) projectId: GUID (FP) format_of_file: Text (FP) type_of_file: Text (FP) uri_of_file: Text (FP) metatada: Text
Deletes file from project and its respective triples and thing descriptions	DELETE	/project/{projectId}/file	(PP) projectId: GUID (FP) format_of_file: Text (FP) type_of_file: Text (FP) uri_of_file: Text (FP) metatada: Text
Retrieves from KGG the respective TTL file generated, saves it into the triple store and generate respective thing descriptions for specific elements of the graph	GET	/project/{projectId}/file/ttl	(PP) projectId: GUID (FP) format_of_file: Text (FP) type_of_file: Text (FP) name_of_file: Text

The TDD is a persistence service that contains the Thing Descriptions created by the Thing Manager component. It uses the WoT Hive implementation, compliant with the W3C Web of Things Directory standard specification. The TDD component provides a REST API allowing other DTP components to discover, create, retrieve, update, and delete Thing Descriptions. The endpoints provided along with their parameters are listed in Table 11.

Table 11 Thing Description Directory's REST API

Description	Method	Endpoint	PP = Path Parameter
			FP = Form Parameter

Provides the Thing Description of the WoT Hive directory	GET	/.well-known/wot-thing-description	N/A
Creates an anonymous Thing Description, provided in the body as JSON-LD framed. The generated id is output in the response headers	POST	/api/things	(body) : JSON-LD
Partially updates an existing Thing Description, the updates must be provided in JSON-LD framed	PATCH	/api/things/{id}	(PP) id: GUID
Solves a SPARQL query following the standard. The response is formatted in JSON. Other formats supported by the API: XML, CSV, and TSV.	GET	/api/search/sparql?{query}	(PP) query: Text
Deletes an existing Thing Description	DELETE	/api/things/{:id}	(PP) id: GUID

4.2.4 Application Example

An example of the operations performed by the KGG is illustrated in Figure 18. Initially, the IDM component creates a project entry into the Thing Manager, which includes the project id, project name, and project description. Next, the IDM uploads the as-planned input data files, and the Thing Manager through the Wrapper module triggers the execution of the various ETL tools.

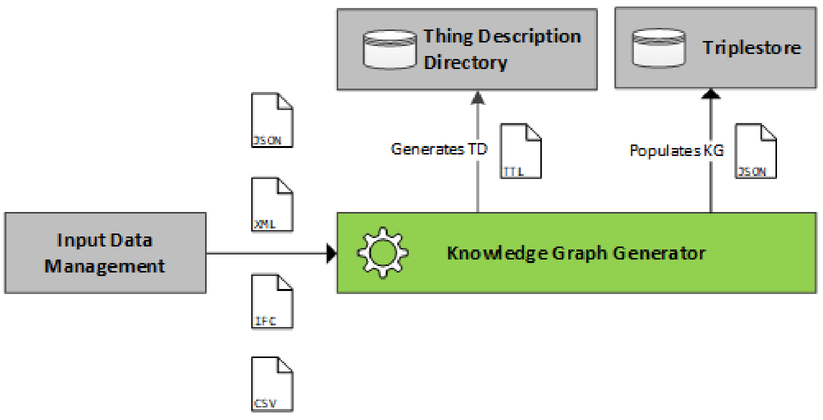


Figure 18 Generation of COGITO's TTLs and TDs

The outputs of the KGG component consist of the TTL data illustrated in Figure 19, and the corresponding JSON-LD data illustrated in Figure 20.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix data: <http://data.cogito.iot.linkeddata.es/resources/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix facility: <https://cogito.iot.linkeddata.es/def/facility#> .
@prefix process: <https://cogito.iot.linkeddata.es/def/process#> .
@prefix resource: <https://cogito.iot.linkeddata.es/def/resource#> .

data:Project_cogito:project:1
a facility:Project ;
facility:hasName "test1"^^<http://www.w3.org/2001/XMLSchema#string> ;
facility:hasDescription "test description"^^<http://www.w3.org/2001/XMLSchema#string> ;
facility:projectID "cogito:project:1"^^<http://www.w3.org/2001/XMLSchema#string> ;
```

```
facility:isRelatedToProcess process:data/1789B04B-1A16-EC11-9EF0-B00CD17291CA .

data:Process_cogito:project:1_1789B04B-1A16-EC11-9EF0-B00CD17291CA
a process:Process ;
process:processID '1789B04B-1A16-EC11-9EF0-B00CD17291CA' ;
process:hasName '20210916_Planificacion_UTE_SOT.xml' ;
process:hasCreationDate '2020-07-31T09:00:00'^^<http://www.w3.org/2001/XMLSchema#dateTime>;
process:hasCost data:cost/1789B04B-1A16-EC11-9EF0-B00CD17291CA ;
process:isPlannedIn data:interval/1789B04B-1A16-EC11-9EF0-B00CD17291CA .

data:Task_cogito:project:1_1789B04B-1A16-EC11-9EF0-B00CD17291CA_0
a process:Task ;
process:taskId '1889B04B-1A16-EC11-9EF0-B00CD17291CA' ;
process:taskUid '0' ;
process:hasName 'SOT-MURCIA'^^<http://www.w3.org/2001/XMLSchema#string> ;
process:hasCreationDate ''^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
process:isPlannedIn data:interval/1889B04B-1A16-EC11-9EF0-B00CD17291CA ;
process:hasPriority '500'^^<http://www.w3.org/2001/XMLSchema#integer> ;
process:hasProgress 'PT0H0M0S'^^<http://www.w3.org/2001/XMLSchema#string> ;
process:hasStatus 'PT0H0M0S'^^<http://www.w3.org/2001/XMLSchema#string> ;
process:hasCost data:cost/1889B04B-1A16-EC11-9EF0-B00CD17291CA .

...

data:ResourceType_cogito:project:1_1789B04B-1A16-EC11-9EF0-
B00CD17291CA_Truck_mounted_concrete_boom_pump_1
a resource:ResourceType ;
a resource:EquipmentType ;
resource:resourceTypeId 1 ;
resource:name 'Truck_mounted_concrete_boom_pump'^^<http://www.w3.org/2001/XMLSchema#string> ;
resource:initials 1 ;
resource:masUnit 2 ;
resource:costPerHour 2000 .

...
```

Figure 19 Example of RDF generated by the ETL tools contained in the KGG component

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "cogito:project:1",
    "title": "demo",
    "description":"/api/things/uuid:project:0c8c5e40-dbdf-484f-b563-dcf550f84d90"
    "securityDefinitions": {
        "nosec_sc": {"scheme":"nosec"}
    },
    "security": ["nosec_sc"],
    "properties": {
        "demo": {
            "forms": [
{
    "href": "/files/uuid:file:3741aa23-c870-4937-bfa7-2fbfec971c0a",
    "contentType": "json"
},
{
    "href": "/files/uuid:file:3741aa23-c870-4937-bfa7-2fbfec971c0b",
    "contentType": "ifc"
},
{
    "href": " /files/uuid:file:3741aa23-c870-4937-bfa7-2fbfec971c0c",
    "contentType": "xml"
},
{
    "href": "/files/uuid:file:3741aa23-c870-4937-bfa7-2fbfec971c0d",
    "contentType": "csv"
},
            ]
        },
        "http://openmetrics.eu/openmetrics#Space_1157": {
            "forms": [
                {
                   "href":"/sparql?query=http://openmetrics.eu/openmetrics#Space_1354",
                   "contentType": "ttl"
                }
            ]
        },
        "http://openmetrics.eu/openmetrics#Space_1354": {
            "forms": [
                {
                   "href":"/sparql?query= http://openmetrics.eu/openmetrics#Space_1354",
                   "contentType": "ttl"
                }
            ]
        },
    …
}
```

Figure 20 Example of Thing Description generated by the Thing Manager

4.2.5 Licensing

As mentioned previously, the KGG component consists of i) the Thing Manager, which handles the requests done by other DTP components and generates the Thing Descriptions, ii) the Wrapper module, which is called internally by the Thing Manager and performs the requests to the ETL tools for generating and validating the RDF data, and iii) the various ETL tools, which are responsible for transforming the raw input data files to RDF triples. All these sub-components are open-source and licenced under the Apache Licence 2.0.

4.2.6 Installation Instructions

The installation instructions are simply to deploy a Docker container along with a configuration file indicating the endpoints where will be allocated the WoT Thing Directory (TDD), the Triple Store and the Thing Manager. In addition, a docker-compose file containing the execution of the containers belonging to the Thing Manager, Wrappers, Triplestore and Thing Directory will be provided.

4.2.7 Development and Integration Status

Currently, the KGG component is partially developed. The first release of the Thing Manager and the ETL tools have been tested and deployed in the DTP. They support the generation of the complete knowledge graph covering the data requirements of the COGITO use cases. In the final release, the Thing Manager will support i) the automatic validation of the knowledge graph by applying checking rules using the SHACL specification and ii) the generation of the Thing Descriptions for all instances included in the graph.

4.2.8 Requirements Coverage

As shown in Table 12, the KGG component covers one of the functional requirements of the DTP included in "D2.5 – COGITO System Architecture v2". The Req-1.5 is fully covered by the KGG sub-components and the involved ETL tools. The first release of the Thing Manager can orchestrate the data transformation processes and to store the output RDF data in the Triplestore.

Table 12 KGG component's Requirements Coverage

Type	ID	Description	Status
Functional	Req-1.5	Populates COGITO's ontology	Achieved

4.2.9 Assumptions and Restrictions

The operation of the Knowledge Graph Generator tool relies on the following assumptions and restrictions.

- The communication between the Wrappers and the Thing Manager must be done by means of events, created by the Thing Manager in different channels to which the different Wrappers must be subscribed according to their function to be performed.
- The Thing Manager configuration must always contain the endpoints required for Thing Manager deployment on a specific port, the Triplestore endpoint and the Thing Directory endpoint.
- The information sent to the Thing Manager must always be in JSON format, if the sender is not a Wrapper.
- In case you want to register a file corresponding to a project, you must send as information in JSON format, the identifier of the project where you want to register the file and the URL where it is stored, to extract the information to be transformed to RDF.
- Thing Descriptions must follow the structure defined in the W3C standard.

5 Data Processing Operations and Data Delivery

Once the as-planned data are loaded to the DTP's Data Persistence Layer and the knowledge graphs are generated, the COGITO applications can interact with the DTP for a) providing the as-built data, which consists of imagery and location tracking information along with their meta-data, and b) performing data requests that require special handling due to the diversity of the various domains included in COGITO.

Within COGITO, the **Data Pre-Processing tools** are responsible for providing the as-built data to DTP. More specifically this category contains the following tools:

- The *IoT Data Pre-Processing tool* is responsible for pre-processing raw location tracking data coming from various IoT sensors.
- The *Visual Pre-Processing tool*, responsible for pre-processing raw visual data and point clouds coming from cameras and LiDAR scanners.

On the other hand, the remaining COGITO tools are interacting with DTP's Data Management Layer and are classified as follows:

- The **Health and Safety tools** are responsible for generating hazards mitigation measures and producing warnings notifications to the on-site workers for their proximity to hazardous areas.
- The **Workflow Modeling and Simulation tools** are responsible for monitoring and optimising the construction processes.
- The **Quality Control tools** are responsible for comparing the as-designed and as-built data and detecting potential defects.
- The **Visualisation tools** are responsible for retrieving and visualizing various data stored in the DTP, to support on-site and off-site activities of relevant stakeholders.

The Data Management Layer manages the data requests of these tools by offering an actor-based runtime environment and a web-based application for the configuration of the various endpoints required. This section presents the first release of the DT Runtime component, which is responsible for orchestrating the internal data processing operations, harmonizing their responses, and sending them to the COGITO tools.

5.1 Digital Twin Runtime component

The Digital Twin (DT) Runtime component is a lightweight data integration container for hosting configurable modules implemented to perform various data processing operations. It is based on the open-source framework Akka[7] that offers a toolkit for simplifying the deployment of concurrent and distributed applications. In other words, Akka is a powerful reactive high-performance framework optimised for running on the Java Virtual Machine (JVM). Within COGITO, this implementation can handle multiple requests simultaneously performed by the various COGITO tools and to respond through the provided REST API.

5.1.1 Prototype Overview

This component contains a set of ready-made software actors for facilitating tool developers to design and deploy dynamic modules that can handle complex requests. An actor is an extensible program-code template that can contain configurable parameters (user credentials, database credentials, triplestore locations) for interacting with other components and executes its business-logic operations and forwards the response to the next actors using framework's embedded lightweight messaging system. Thus, the actors are created once by the DTP developers and then reused in the various configurable modules of the DT Runtime component, as shown in Figure 21. It is worth mentioning that the first release includes a fully functional REST API for interacting with the other COGITO tools.

[7] Akka Actor Model https://www.akka.io

The final release of this component will contain various adapters allowing asynchronous communication through enterprise messaging protocols such as AMQP, KAFKA and MQTT.

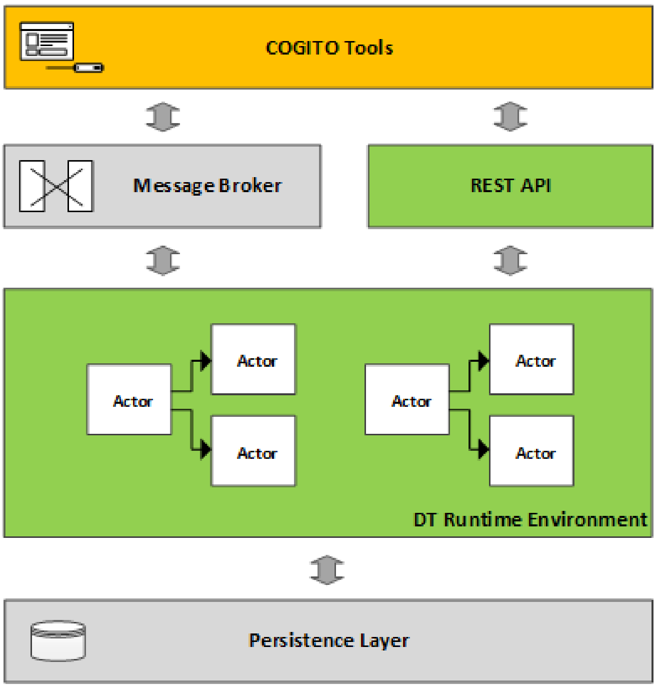


Figure 21 Digital Twin Runtime component's main interactions

For instance, the generation of an IFC file from the corresponding 4D BIM model for a given time frame, has a two-step process: first, the execution of a SPARQL query for retrieving the identifiers of the BIM elements that are involved to active tasks, and second, the filtering of the original IFC with the identifiers returned from the first step. The output of this module is an IFC file that contains only the BIM elements of the active tasks as shown in Figure 22.

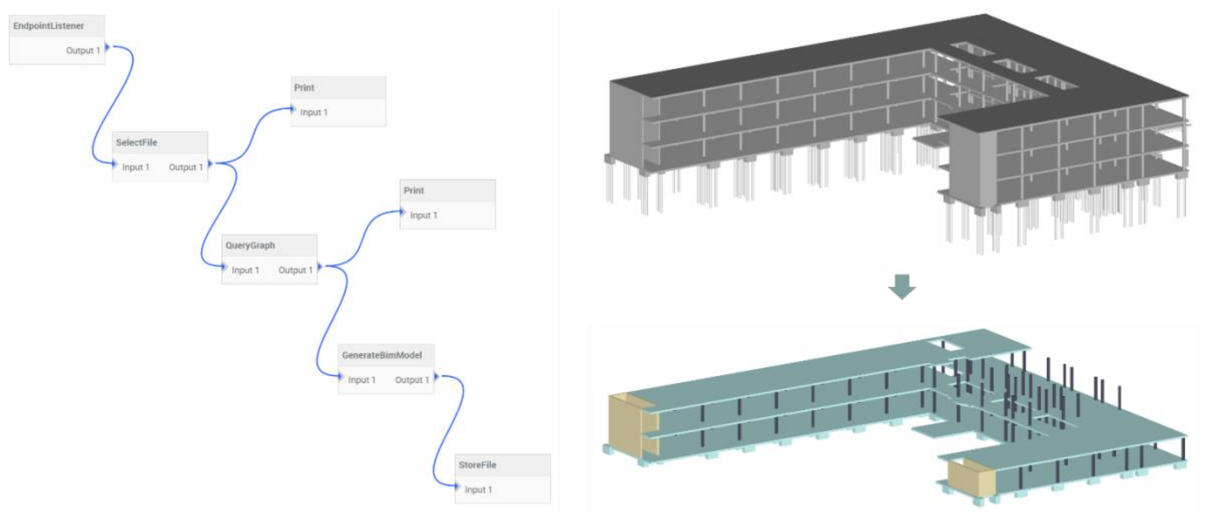


Figure 22 Example of DT Runtime component's configurable module and its IFC output

5.1.2 Technology Stack and Implementation Tools

The DT Runtime component is based on open-source technologies. It is built on top of the Spring Framework and is deployed as standalone web-based application. It has been installed and configured to run behind the Nginx reverse proxy server, which handles the SSL encryption and forwards the HTTP requests and the web-socket packets to the correct destination.

The integration between Spring and Akka frameworks is possible through the Akka Extension mechanism. This technology enables external Dependency Injection (DI) frameworks like Spring to manage the lifecycle of the actor instances.

Table 13 Libraries and Technologies used in the Digital Twin Runtime component

Technology Name	Version	License
Spring Framework	5.3.1	Apache Licence 2.0
Spring Boot	2.3.0	Apache Licence 2.0
Spring Security	5.5.0	Apache Licence 2.0
Spring Integration	5.5.5	Apache Licence 2.0
Akka	2.6.15	Apache Licence 2.0
Apache Jena	3.13.0	Apache Licence 2.0
Thymeleaf	3.0.15	Apache Licence 2.0
Hibernate	5.6.9	LGPL 2.1
MySQL	8.0.24	GPLV2
Nginx	1.20.2	BSD
Certbot	1.22	Apache Licence 2.0

5.1.3 Input, Output and API Documentation

The DT Runtime component provides a REST API enabling the various COGITO applications to request data fetched by actors interacting with the Data Persistence Layer. The application developers register their tools to the DT Runtime component and configure the endpoints of each tool as defined in the "D2.5 – COGITO System Architecture v2". On the other hand, the lead group in charge of the development of DTP is responsible for deploying and connecting these endpoints with the proper modules. The REST API allows the COGITO tools to manage their data collections for each endpoint. Thus, the COGITO tools can perform requests for handling data collections and the contained files. The endpoints provided along with their parameters are listed in Table 14. The detailed documentation and the Postman collections of DTP's REST APIs are available online[8].

Table 14 DT Runtime component's REST API Endpoints

Name	Method	REST Endpoint	PP = Path Parameter
			FP = Form Parameter

Get the Application	GET	https://dtp.cogito-project.com/api/application	N/A
Get all Endpoints of the Application	GET	https://dtp.cogito-project.com/api/application/endpoints	N/A
Get all Data Collections of an Endpoint	GET	https://dtp.cogito-project.com/api/endpoints/{endpointId}/collections	(PP) endpointId: GUID
Get a Data Collection	GET	https://dtp.cogito-project.com/api/collections/{collectionId}	(PP) collectionId: GUID
Create a new Data Collection	POST	https://dtp.cogito-project.com/api/collections/create	(FP) endpointId: GUID (FP) projectId: GUID
Update the Status of a Data Collection	POST	https://dtp.cogito-project.com/api/collections/{collectionId}/status	(PP) collectionId: GUID (FP) status: Text

[8] Digital Twin Platform API documentation https://api.cogito-project.com

Delete a Data Collection	DELETE	https://dtp.cogito-project.com/api/collections/{collectionId}/delete	(PP) collectionId: GUID
Get all Files of a Data Collection	GET	https://dtp.cogito-project.com/api/collections/{collectionId}/files	(PP) collectionId: GUID
Upload Files into a Data Collection	POST	https://dtp.cogito-project.com/api/collections/{collectionId}/upload	(PP) collectionId: GUID (FP) files: File
Update the Type of a File	POST	https://dtp.cogito-project.com/api/files/{fileId}/type	(PP) fileId: GUID (FP) type: Text
Delete a file	DELETE	https://dtp.cogito-project.com/api/files/{fileId}/delete	(PP) fileId: GUID

The following are examples of JSON responses returned by the GET requests of the REST API. For some requests, the responses have no body content. In this case, the HTTP status code is used: i) the "*200 - OK*" is returned if the request is successful, the "*404 - Not Found*" is returned if the resource is unavailable and iii) the response "*400 - Bad Request*" is returned if an error has occurred.

Get the Application

```
{
    "name": "Demo",
    "id": "36ea91a9-d2a8-4916-8de3-8a8ecda4dfeb"
}
```

Get all Endpoints of the Application

```
[
    {
        "name": "4D",
        "description": "4D",
        "id": "81cff0ea-1bf0-4a22-8903-ac5374c02436"
    }
]
```

Get all Data Collections of an Endpoint

```
[
    {
        "project": "1cf55b98-db69-46f8-a64e-a531d512d4d3",
        "id": "47368c01-1d50-459a-997d-f8635e3c0511",
        "creation_date": "2022-05-03 11:46:12.0",
        "status": "completed"
    },
    {
        "project": "c69407ff-2f81-4138-85d1-21a5e9e24550",
        "id": "0b781566-2dee-4120-bbd7-ec3895e67f67",
        "creation_date": "2022-05-03 12:54:14.0",
        "status": "incomplete"
    }
]
```

Get a Data Collection

```
{
    "project": "c69407ff-2f81-4138-85d1-21a5e9e24550",
    "id": "5425dc1d-a9e2-4d52-89fd-5732013b84ff",
    "creation_date": "2022-05-03 11:49:46.0",
    "status": "completed"
}
```

Get all Files of a Data Collection

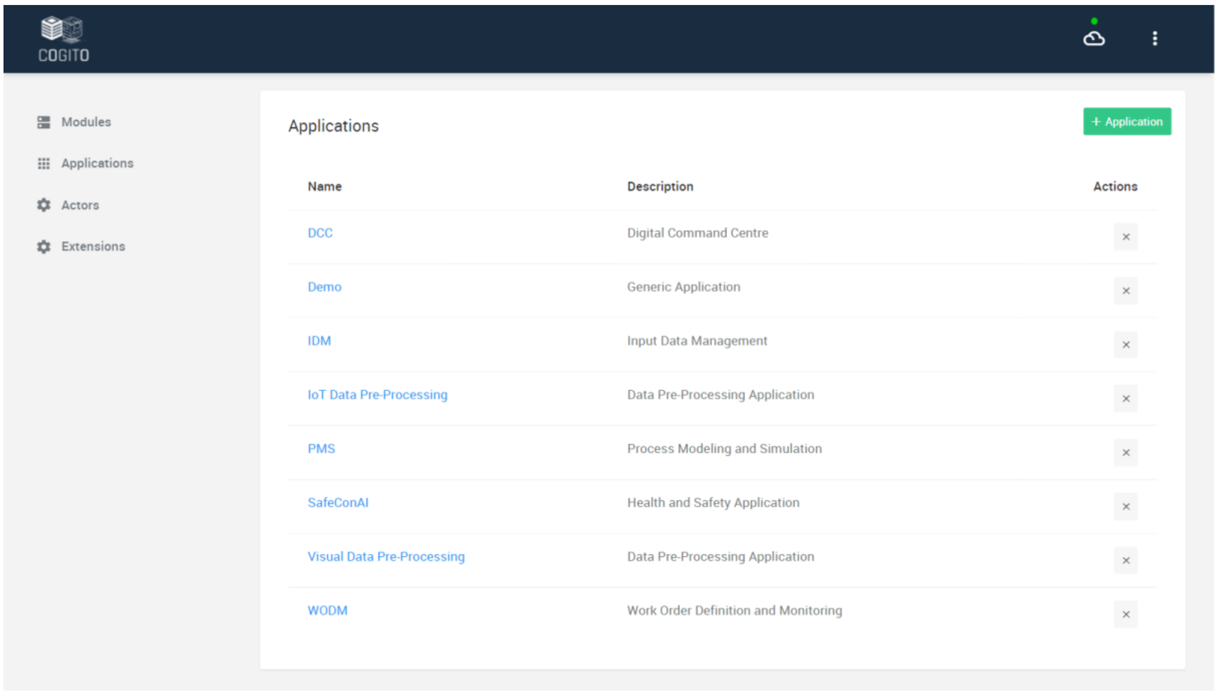
```
[
    {
        "date": "2022-05-03 11:49:58.0",
        "extension": "CSV",
        "name": "rst_advanced_sample_project_MSP2010_V2",
        "id": "b4984a23-1feb-45fc-9d79-3b03436b8f68",
        "type": "CONSTRUCTION_SCHEDULE",
        "url": "https://dtp.cogito-project.com/file/b4984a23-1feb-45fc-9d79-3b03436b8f68/download"
    },
    {
        "date": "2022-05-03 11:49:55.0",
        "extension": "CSV",
        "name": "resources",
        "id": "f2a52a51-8fc2-4f2f-a7d9-f3fe4968bed3",
        "type": "AS_PLANNED_RESOURCES",
        "url": "https://dtp.cogito-project.com/file/f2a52a51-8fc2-4f2f-a7d9-f3fe4968bed3/download"
    }
]
```

5.1.4 Usage Walkthrough

The DT Runtime component offers a GUI for configuring and managing DTP's functional modules and their interfaces with the other COGITO tools. Thus, the GUI consists of three main parts: i) The "*Registration of COGITO Applications*" used for registering the COGITO applications and configuring their access policies, endpoints, and notification channels, ii) The "*Registration of COGITO Actors*" used for configuring the meta data of the actors and their properties, and iii) The "*Deployment of COGITO Modules*" used to create the data processing workflows required by the COGITO applications. These three parts are presented in detail in the following subsections.

5.1.4.1 Registration of COGITO Applications

The users who have a developer role can sign in to the DT Runtime component through the DTP's Identity Provider. The "Applications" button shows the list of applications added in the past, as shown in Figure 23. On the right side of the screen, the "Actions" column contains buttons for deleting the applications after confirmation via a popup window. By clicking on any application name, the users can see further information and perform additional actions. Users may have access only to their applications or the entire list, depending on their roles. The role of the DTP Developer allows access to all applications. In contrast, various composite roles like DCC Developer, PMS Developer, WODM Developer and others allow access to applications created by the specific user.

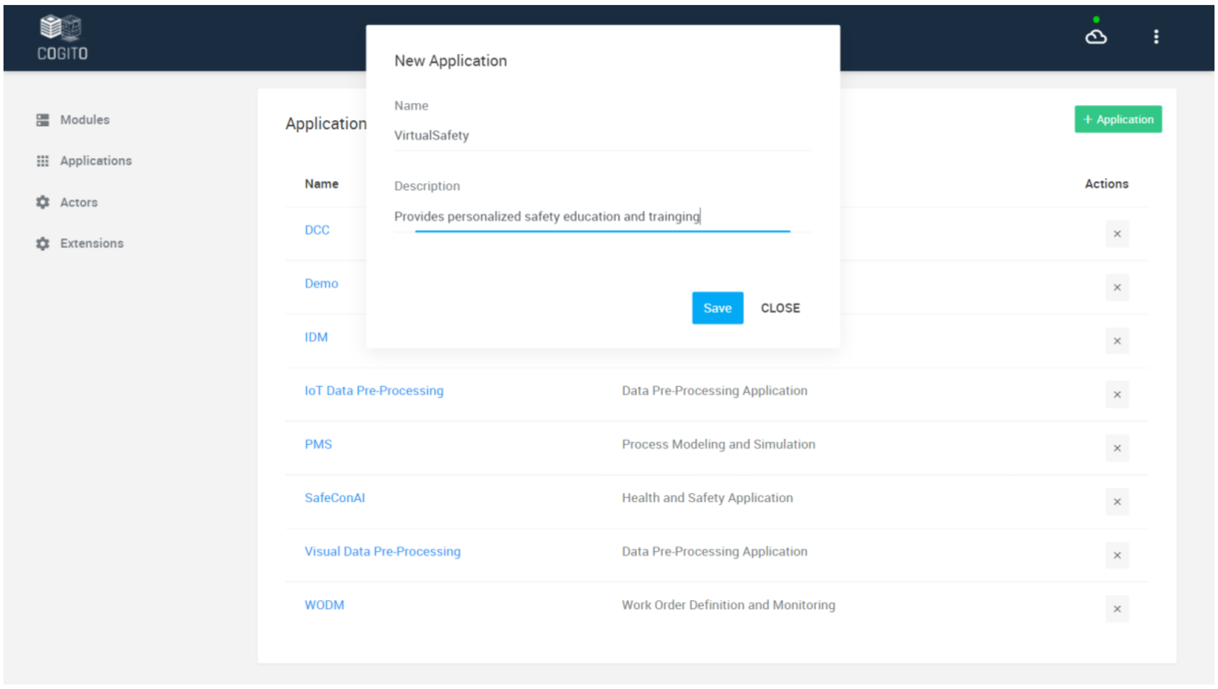


Applications

Name	Description	Actions
DCC	Digital Command Centre	×
Demo	Generic Application	×
IDM	Input Data Management	×
IoT Data Pre-Processing	Data Pre-Processing Application	×
PMS	Process Modeling and Simulation	×
SafeConAI	Health and Safety Application	×
Visual Data Pre-Processing	Data Pre-Processing Application	×
WODM	Work Order Definition and Monitoring	×

Figure 23 View of all user's applications

On the same page, users who have a developer role can add a new application by clicking the "+ Application" button. As shown in Figure 24, a modal window appears which allows users to provide the application name and a short description. At any time, users can cancel the application registration process by clicking on the Close button.

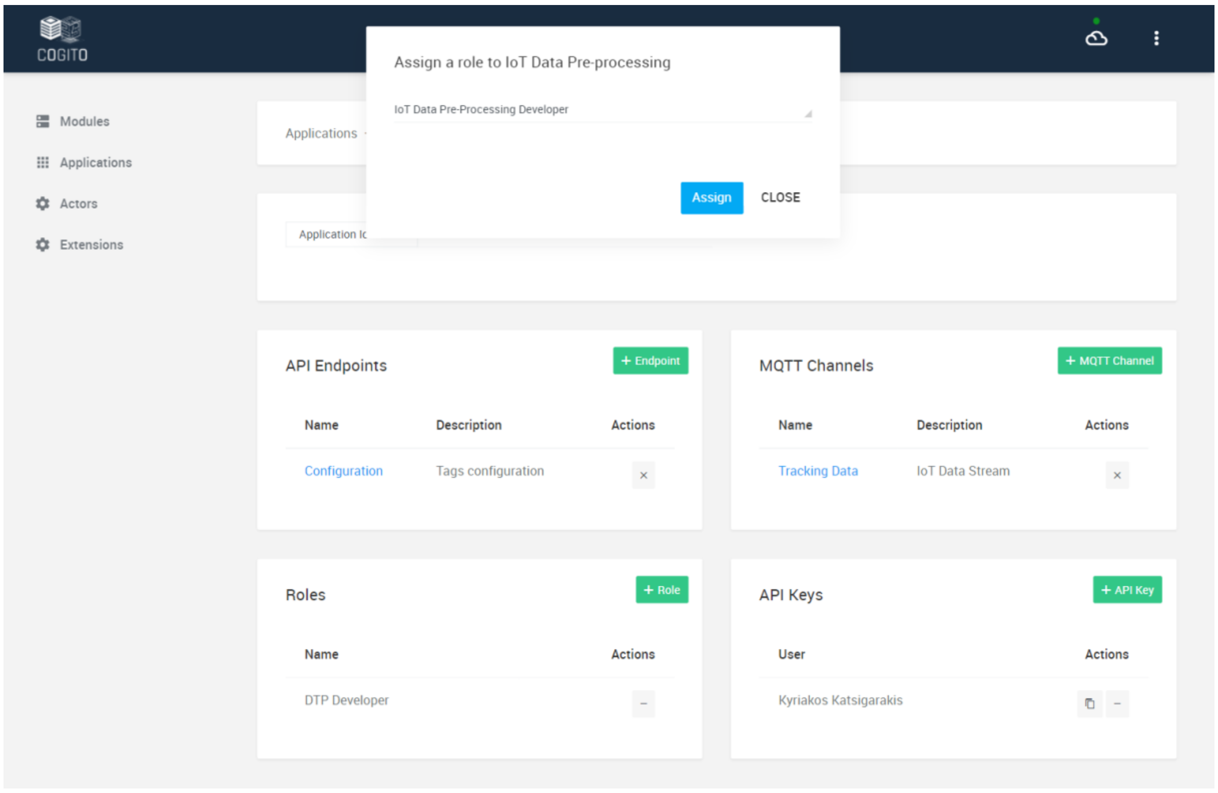


Name: VirtualSafety
Description: Provides personalized safety education and trainging

Name	Description	Actions
DCC		×
Demo		×
IDM		×
IoT Data Pre-Processing	Data Pre-Processing Application	×
PMS	Process Modeling and Simulation	×
SafeConAI	Health and Safety Application	×
Visual Data Pre-Processing	Data Pre-Processing Application	×
WODM	Work Order Definition and Monitoring	×

Figure 24 Creation of a new application

Once the application is added, the users can proceed with the configuration process. Based on COGITO's system architecture described in D2.5, they must configure the REST endpoints and, if required, the MQTT channels. Furthermore, the users can grant access to other users who have different roles by clicking the "+ Role" button. As shown in Figure 25, a modal window appears that contains a combo box element with all available roles. The assignment of the selected role to the application is done by clicking the Assign button. The users can cancel the assignment process by clicking on the Close button.



API Endpoints

Name	Description	Actions
Configuration	Tags configuration	×

MQTT Channels

Name	Description	Actions
Tracking Data	IoT Data Stream	×

Roles

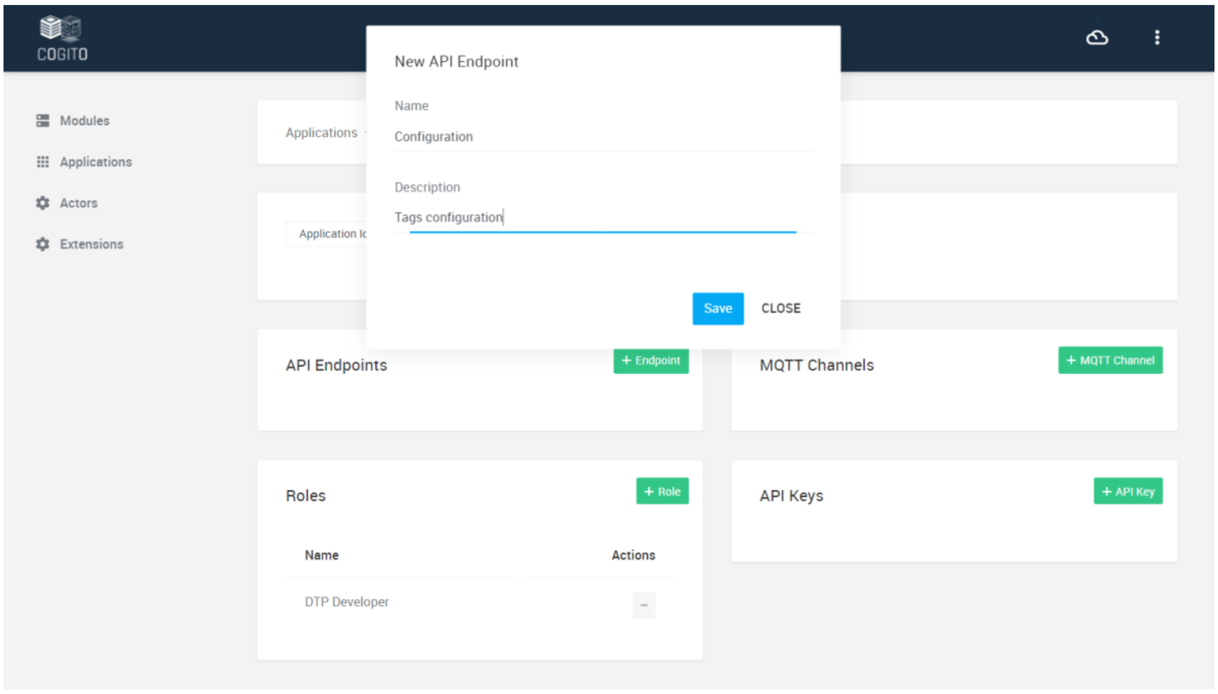
Name	Actions
DTP Developer	–

API Keys

User	Actions
Kyriakos Katsigarakis	–

Figure 25 Assignment of a role to the application

Continuing with the configuration, users can create new endpoints by clicking the "+ Endpoint" button. As shown in Figure 26, a modal window appears allowing the users to fill in a name and a short description. At any time, users can cancel the endpoint creation process by clicking on the Close button.



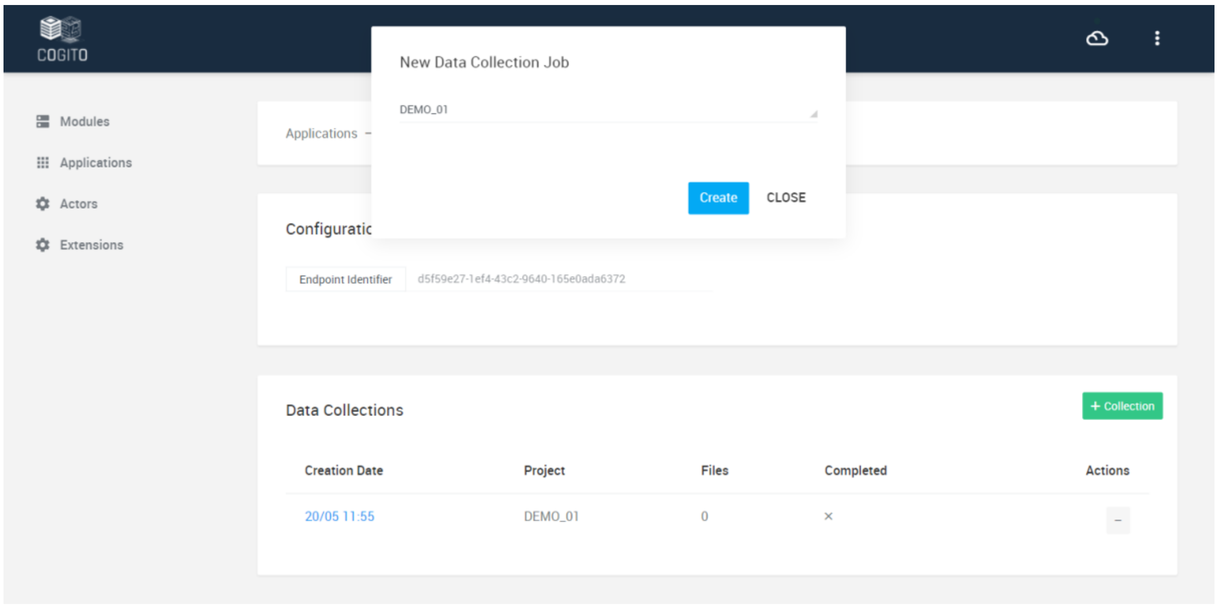
Name: Configuration

Description: Tags configuration

Name	Actions
DTP Developer	–

Figure 26 Creation of new endpoint

Once the endpoint is added, the COGITO users and tools can create data collections using this GUI or the REST API accordingly. As shown in Figure 27, a modal window appears that contains a combo box element with the available projects. This action is identical to the request "Create a new Data Collection" of the REST API described in the previous section.



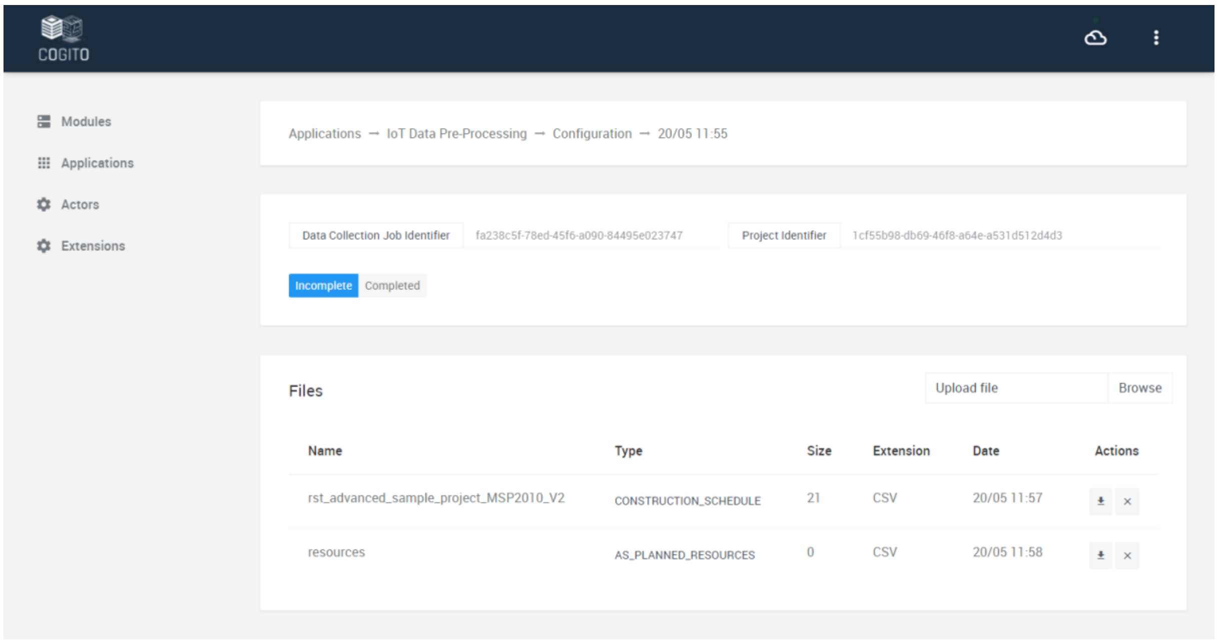
DEMO_01

Endpoint Identifier: d5f59e27-1ef4-43c2-9640-165e0ada6372

Creation Date	Project	Files	Completed	Actions
20/05 11:55	DEMO_01	0	×	–

Figure 27 Creation of a new Data Collection

Once the data collection is created, the COGITO users and tools can upload files, annotate the file types, and update the data collection status. The annotation of the files is essential in case of the data collection contains multiple files that have the same format and different data structures. For instance, as shown in Figure 28, the data collection contains multiple CSV files with different data structures. In this case, the type is used to differentiate the files. Furthermore, the status parameter is used as a triggering mechanism that produces notifications to the active data processing services of the DTP. As mentioned previously, the REST API contains all the functionalities demonstrated here using the web-based application.



Applications → IoT Data Pre-Processing → Configuration → 20/05 11:55

Data Collection Job Identifier: fa238c5f-78ed-45f6-a090-84495e023747
Project Identifier: 1cf55b98-db69-46f8-a64e-a531d512d4d3

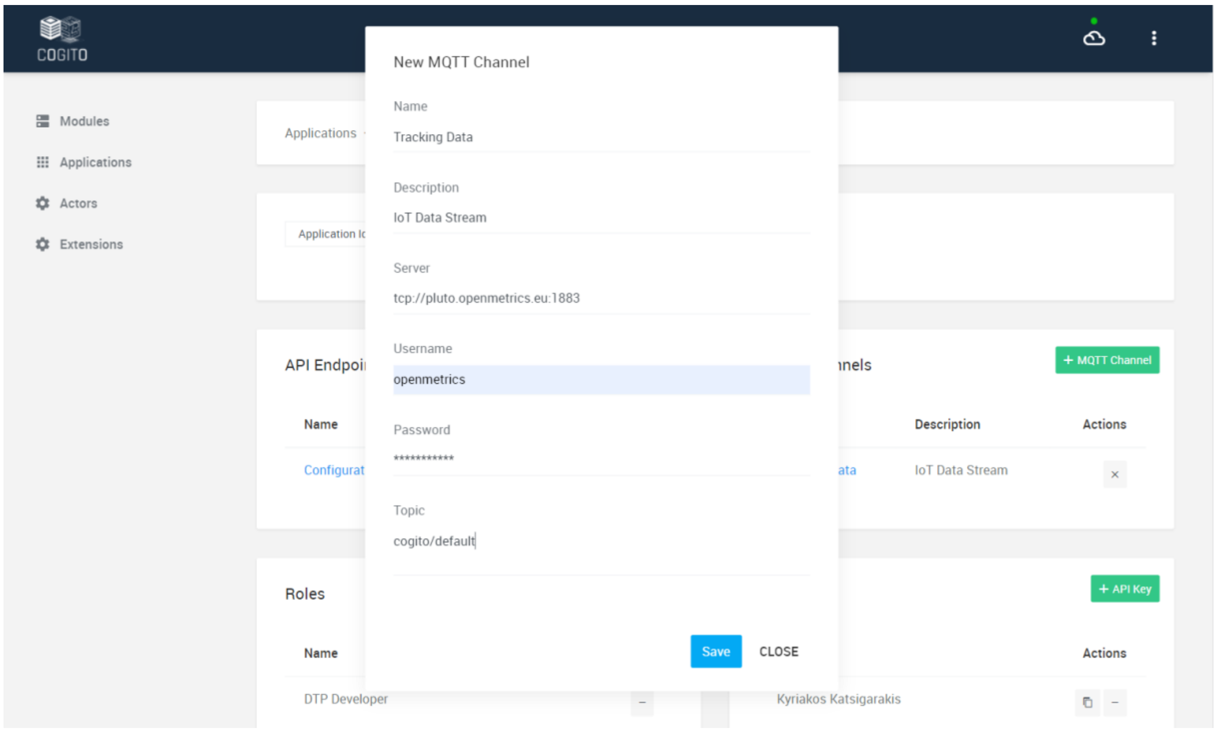
Incomplete | Completed

Files

Name	Type	Size	Extension	Date	Actions
rst_advanced_sample_project_MSP2010_V2	CONSTRUCTION_SCHEDULE	21	CSV	20/05 11:57	
resources	AS_PLANNED_RESOURCES	0	CSV	20/05 11:58	

Figure 28 Uploading and annotating files in a Data Collection

The first release of the DT Runtime component can handle data streams and supports various asynchronous messaging protocols. Currently, the web-based application allows users with proper roles to configure MQTT connections between external applications and the DT Runtime component. This functionality is essential for various COGITO tools. For instance, the IoT Data Pre-processing module uses this technology for streaming real-time location tracking data into the DTP.



New MQTT Channel

Name: Tracking Data

Description: IoT Data Stream

Server: tcp://pluto.openmetrics.eu:1883

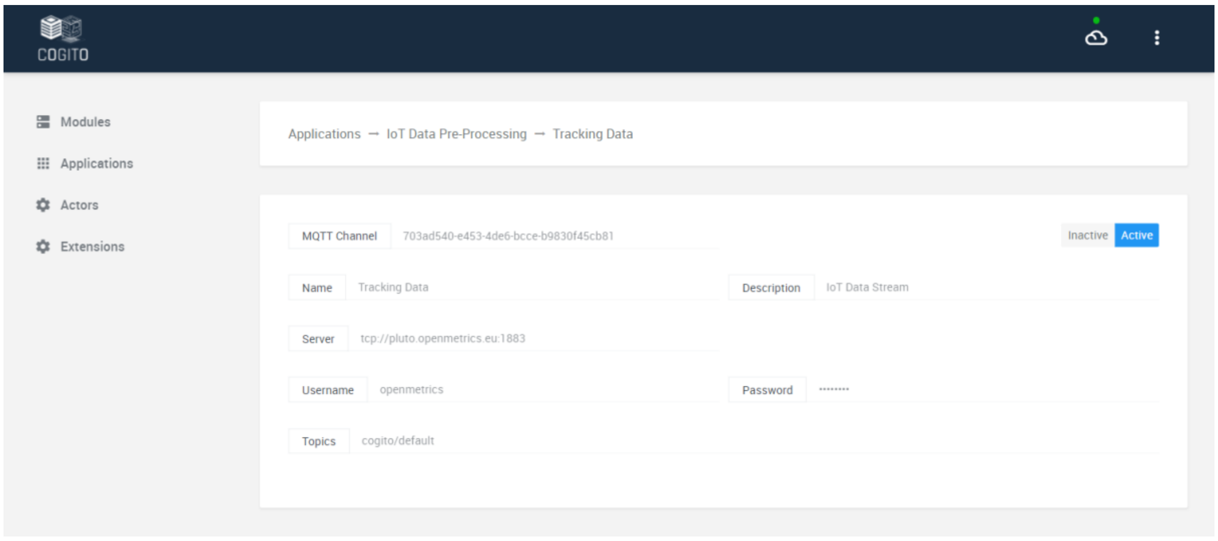
Username: openmetrics

Password: ***********

Topic: cogito/default

Figure 29 Creation of a new MQTT channel

In addition, various COGITO tools such as WOEA, VisualQC, and ProActiveSafety use the publish/subscribe messaging pattern for sending and receiving notifications through the Messaging Layer. As shown in Figure 30, the GUI provides a button for enabling and disabling the configured channels.

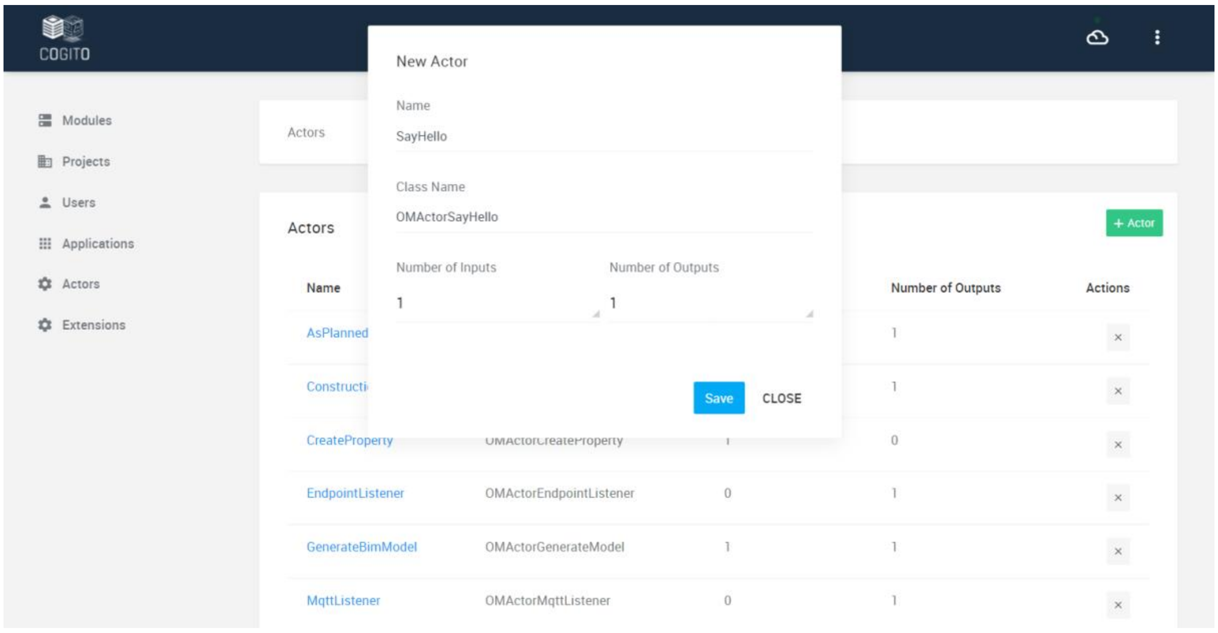


MQTT Channel	703ad540-e453-4de6-bcce-b9830f45cb81
Name	Tracking Data
Description	IoT Data Stream
Server	tcp://pluto.openmetrics.eu:1883
Username	openmetrics
Password	••••••••
Topics	cogito/default

Figure 30 Enabling or disabling an MQTT channel

5.1.4.2 *Registration of COGITO Actors*

The DT Runtime component allows users with proper roles to configure new actors through the web-based application. These actors are reusable blocks of code with specific names and behaviours that implement business logic operations and can be placed and interconnected into the modules. The users can create a new actor by clicking the "+ Actor" button. As shown in Figure 31, a modal window appears allowing the users to fill in the actor's name and some necessary meta data such as the Java class name of the actor as well as the number of inputs and outputs. The creation of the actor is done by clicking the Save button. At any time, the users can cancel the creation process by clicking on the Close button.



Name	SayHello
Class Name	OMActorSayHello
Number of Inputs	1
Number of Outputs	1

Name			Number of Outputs	Actions
AsPlanned			1	×
Constructi			1	×
CreateProperty	OMActorCreateProperty	1	0	×
EndpointListener	OMActorEndpointListener	0	1	×
GenerateBimModel	OMActorGenerateModel	1	1	×
MqttListener	OMActorMqttListener	0	1	×

Figure 31 Creation of a new Actor

Once the actor is created, the users can proceed with the definition of the properties. Each actor can have a set of configurable properties. The values of these properties are set during the configuration of the module.

Optionally, the DT Runtime component allows users to set default values to the properties in the modal window as shown in Figure 32.

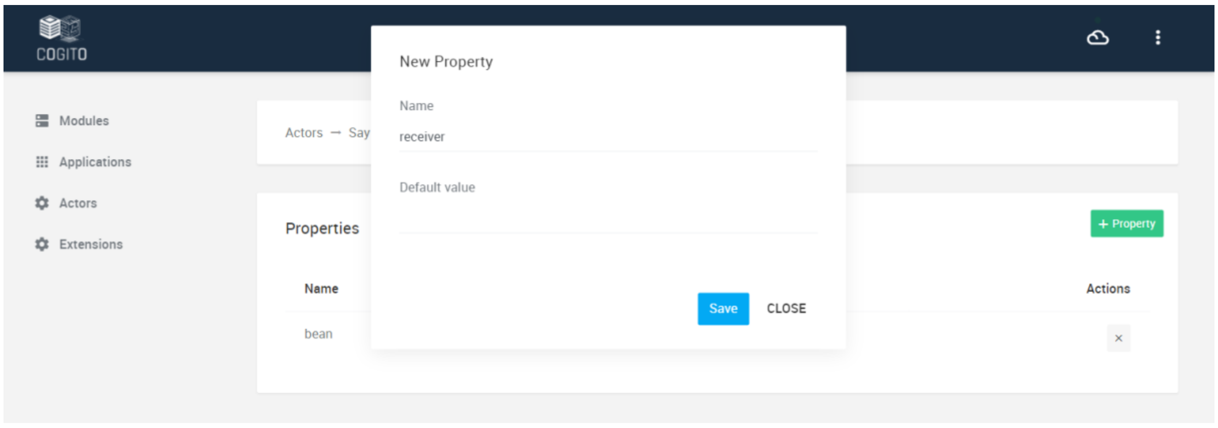


Figure 32 Creation of a new property in an Actor

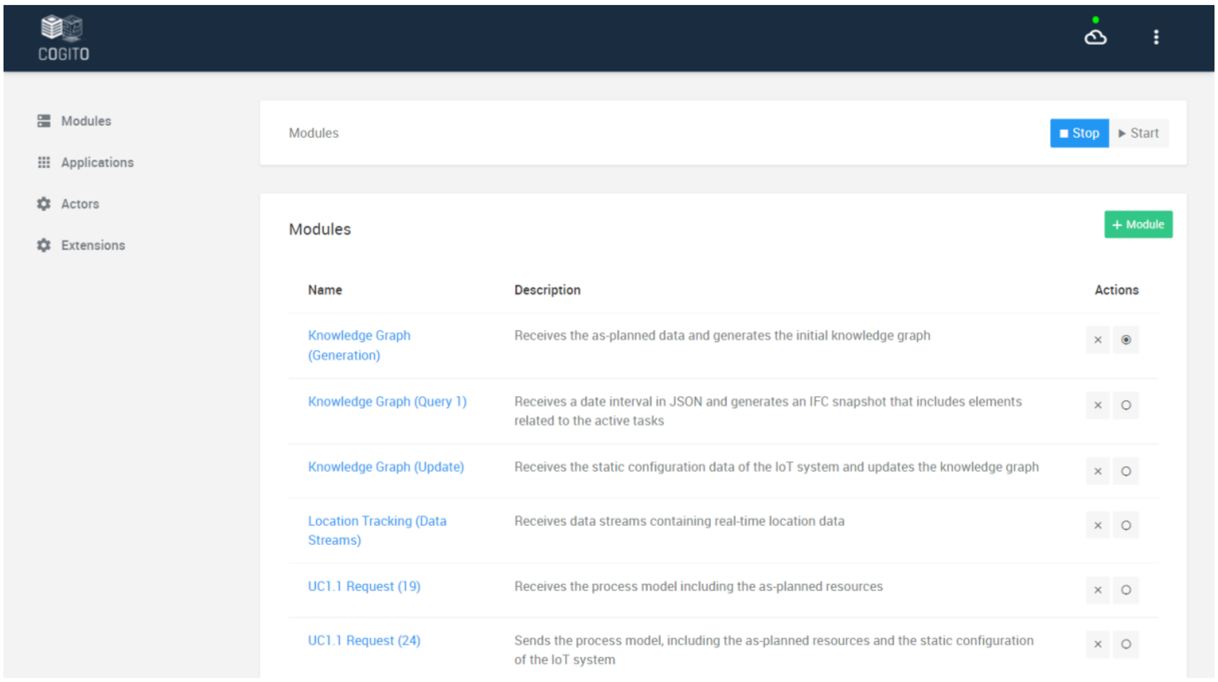
Next, the users must provide and load the Java code that implements the business logic of the actor. For instance, a simple actor in charge to say "Hello <receiver>!" has a code as follows.

```
@Component
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class OMActorSayHello extends OMActorBase {

        private static final Logger log = LoggerFactory.getLogger(OMActorSayHello.class);

        private String receiver;

        public OMActorSayHello(Entity entity) {
                super(entity);
                log.info("say hello actor, path=" + getSelf().path().toString());
                receiver = entity.getProperty("receiver").getValue();
        }

        @Override
        public void onReceive(Object message) throws Throwable {
                if(message instanceof String) {
                        String msg = (String) message;
                        if(msg.equals("launch")) {
                                OMBody omBody = new OMBody();
                                omBody.setBody("Hello " + receiver + "!");
                                tell(omBody);
                        }
                }
        }
}
```

The above actor is designed to accept the string "launch" as an input message. In this case, it will produce an output string "Hello <receiver>!". The <receiver> is an actor property that the users will provide later during the module's configuration. The loading of the above code into the DT Runtime component is done, through the Java ClassLoader mechanism. This technology allows developers to load dynamically Java classes into the Java Virtual Machine (JVM).

5.1.4.3 *Deployment of COGITO Modules*

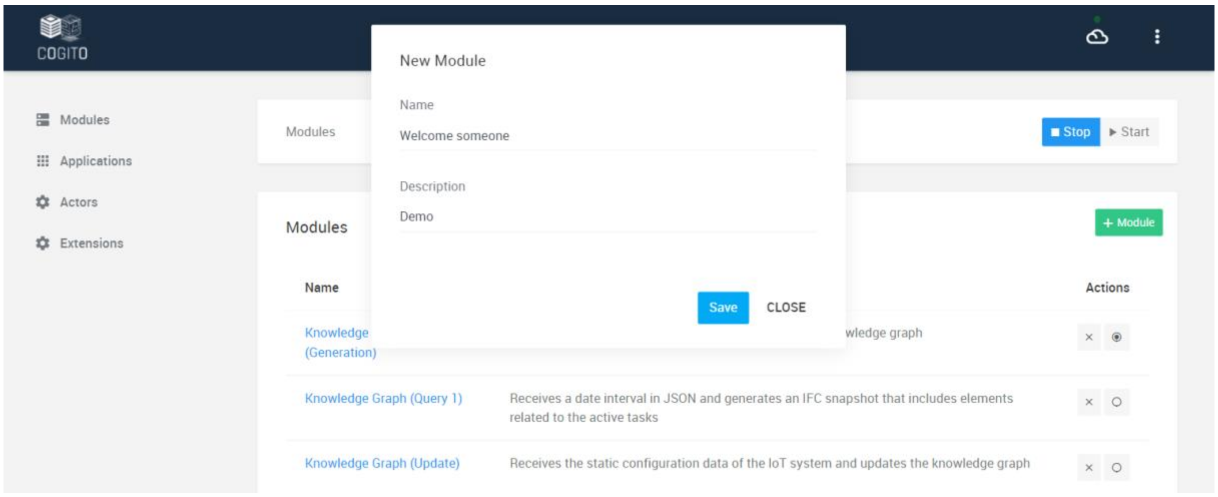
The users who have the DTP Developer role can sign in to the DT Runtime component through the DTP's Identity Provider. The "Modules" button, which is located on the left side of the screen, shows the complete list of modules created in the past, as shown in Figure 33. On the right side of the table, the "Actions" column contains buttons for deleting, activating, or deactivating the installed modules. On the top of the screen, there is the main switch for starting and stopping the entire actor system, and it can be used to perform a manual restart of the active modules. This action is needed when the users demand activation, deactivation, or module installation.



Name	Description	Actions
Knowledge Graph (Generation)	Receives the as-planned data and generates the initial knowledge graph	× ◉
Knowledge Graph (Query 1)	Receives a date interval in JSON and generates an IFC snapshot that includes elements related to the active tasks	× ○
Knowledge Graph (Update)	Receives the static configuration data of the IoT system and updates the knowledge graph	× ○
Location Tracking (Data Streams)	Receives data streams containing real-time location data	× ○
UC1.1 Request (19)	Receives the process model including the as-planned resources	× ○
UC1.1 Request (24)	Sends the process model, including the as-planned resources and the static configuration of the IoT system	× ○

Figure 33 View of all created modules

On the same page, users can create a new module by clicking the "+ Module" button. As shown in Figure 34, a modal window appears which allows users to provide a module name and a short description. At any time, users can cancel the module creation process by clicking on the Close button.



New Module

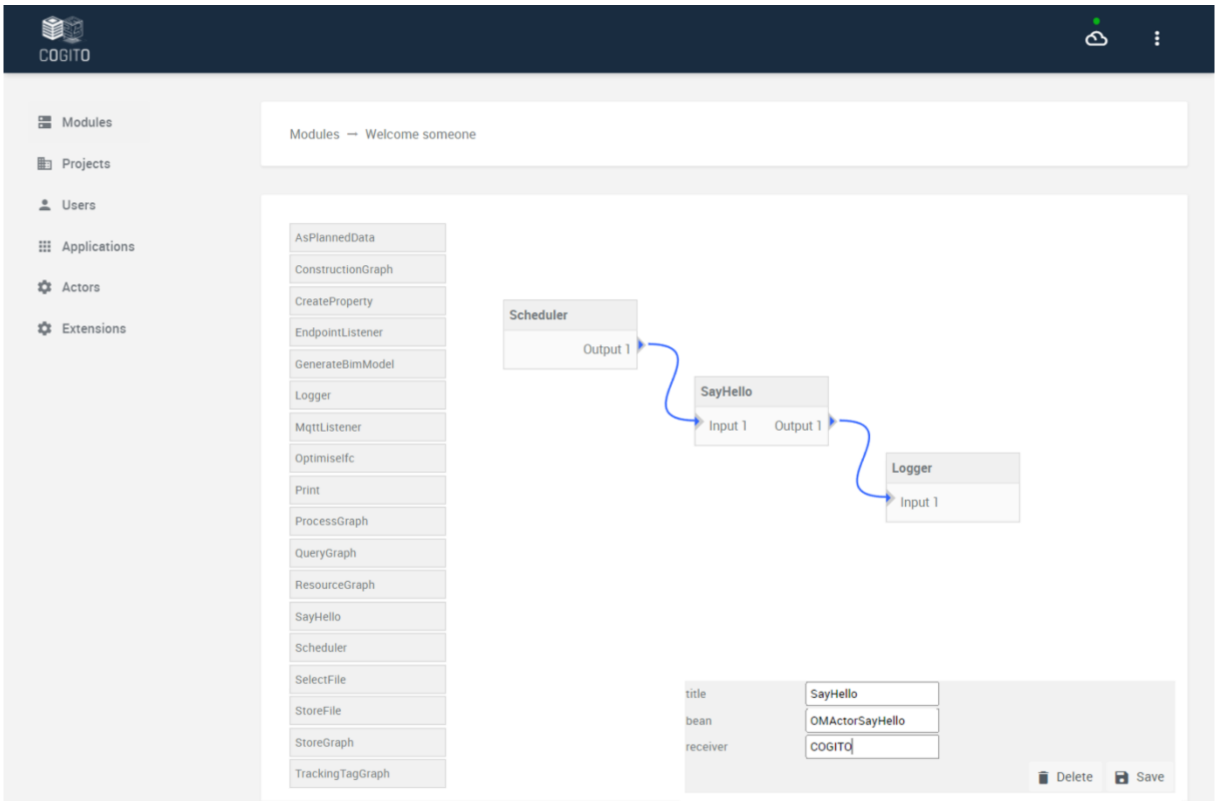
Name: Welcome someone

Description: Demo

Name	Description	Actions
Knowledge (Generation)	wledge graph	× ◉
Knowledge Graph (Query 1)	Receives a date interval in JSON and generates an IFC snapshot that includes elements related to the active tasks	× ○
Knowledge Graph (Update)	Receives the static configuration data of the IoT system and updates the knowledge graph	× ○

Figure 34 Creation of a new module

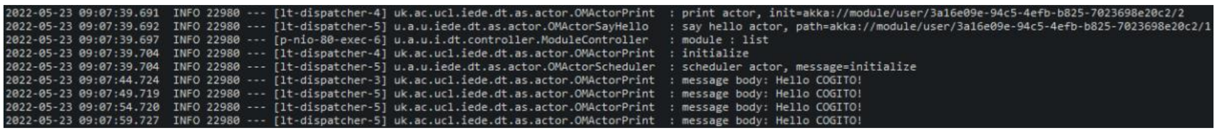
By clicking on the "Welcome someone" module we just created, the users can view the workspace for editing and configuring the module's logic. On the left side of the screen, there is a list of the available actors, as shown in Figure 35. The users can drag and drop actors from the list to the empty area and define their interlinks. For instance, the actor “SayHello” presented previously has one input connected with the actor “Scheduler”, which triggers the execution and one output connected with the actor “Logger”, which prints the content of the messages into the system’s console. As shown in Figure 35, a modal window appears if any actor is selected, allowing the users to fill in or edit the actor properties. The values of these properties are stored in the module and not in the actor type. This means that if the same actor is reused in multiple modules the values of the properties are not shared between the module instances.



title	SayHello
bean	OMActorSayHello
receiver	COGITO

Figure 35 Workspace for configuring and editing modules

Going back to the example, when the “SayHello” actor is selected using the cursor, the property “receiver” is listed in the modal window. After filling the textbox and clicking the Save button, the module is ready for deployment. Figure 36 shows that when the actor system is restarted, the message “Hello COGITO!” appears periodically on the console.



```
2022-05-23 09:07:39.691  INFO 22980 --- [lt-dispatcher-4] uk.ac.ucl.iede.dt.as.actor.OMActorPrint  : print actor, init=akka://module/user/3a16e09e-94c5-4efb-b825-7023698e20c2/2
2022-05-23 09:07:39.692  INFO 22980 --- [lt-dispatcher-5] u.a.u.iede.dt.as.actor.OMActorSayHello   : say hello actor, path=akka://module/user/3a16e09e-94c5-4efb-b825-7023698e20c2/1
2022-05-23 09:07:39.697  INFO 22980 --- [p-nio-80-exec-6] u.a.u.i.dt.controller.ModuleController   : module : list
2022-05-23 09:07:39.704  INFO 22980 --- [lt-dispatcher-4] uk.ac.ucl.iede.dt.as.actor.OMActorPrint  : initialize
2022-05-23 09:07:39.704  INFO 22980 --- [lt-dispatcher-5] u.a.u.iede.dt.as.actor.OMActorScheduler  : scheduler actor, message=initialize
2022-05-23 09:07:44.724  INFO 22980 --- [lt-dispatcher-3] uk.ac.ucl.iede.dt.as.actor.OMActorPrint  : message body: Hello COGITO!
2022-05-23 09:07:49.719  INFO 22980 --- [lt-dispatcher-5] uk.ac.ucl.iede.dt.as.actor.OMActorPrint  : message body: Hello COGITO!
2022-05-23 09:07:54.720  INFO 22980 --- [lt-dispatcher-5] uk.ac.ucl.iede.dt.as.actor.OMActorPrint  : message body: Hello COGITO!
2022-05-23 09:07:59.727  INFO 22980 --- [lt-dispatcher-5] uk.ac.ucl.iede.dt.as.actor.OMActorPrint  : message body: Hello COGITO!
```

Figure 36 DT Runtime component’s system console

5.1.5 Application Example

Within “UC1.1 – Efficient and detailed project workflow planning using the project’s construction schedule and as-planned BIM model”, the PMS tool requests **(5)** and receives **(6)** from the DTP the as-planned 4D BIM of the project, including the as-planned resources as shown in Figure 37. Although the geometric information is not required, the response includes semantically linked information of a) building elements with their construction zones, b) tasks with their properties and, c) as-planned resource types with their properties.

When a new project is configured, and the users upload the as-planned data to the DTP, the IDM component triggers the Thing Manager, which manages the execution of the ETL tools and generates the knowledge graph of the project. At this stage, PMS can request the as-planned non-geometric 4D BIM data because this information is already included in the knowledge graph.

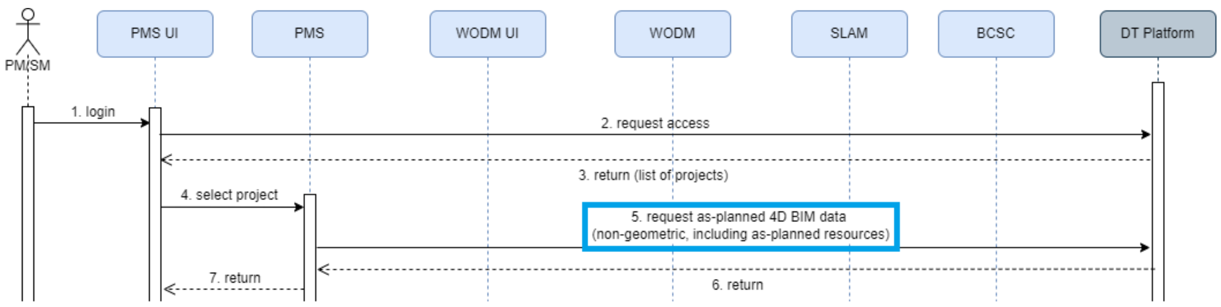

PM/SM
PMS UI
PMS
WODM UI
WODM
SLAM
BCSC
DT Platform
1. login
2. request access
3. return (list of projects)
4. select project
5. request as-planned 4D BIM data (non-geometric, including as-planned resources)
6. return
7. return

Figure 37 Part of the UC1.1 sequence diagram

As shown in Figure 38, a dedicated application-driven module is connected to the endpoint of the DTP which is responsible for handling the request and providing the response to PMS.

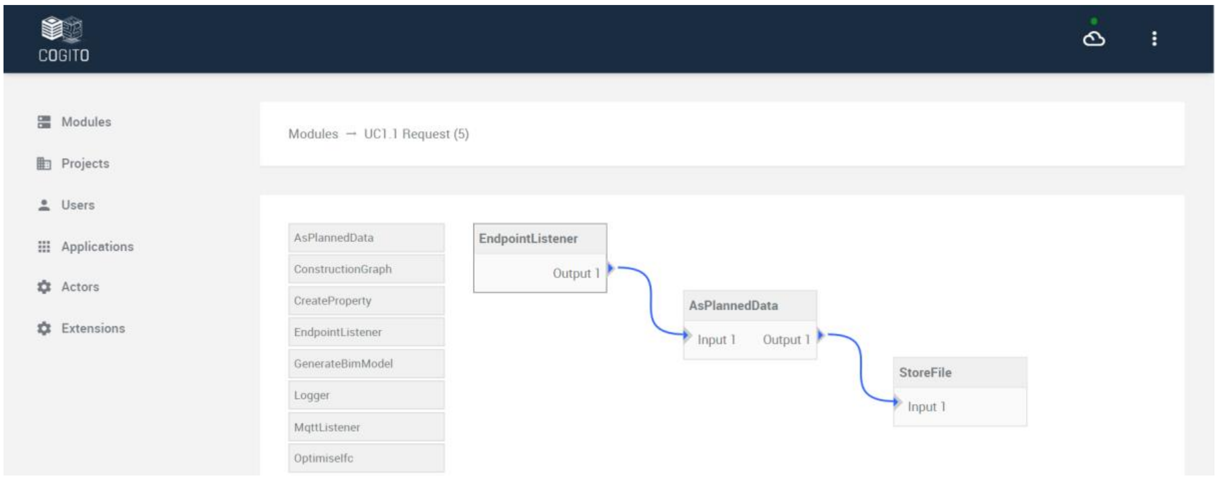


Figure 38 The module UC.1.1 – Request 5 which provides the as-planned 4D BIM data

The response is a JSON file containing data extracted from the knowledge graph using a series of SPARQL queries. The structure of the JSON is designed to meet the input requirements of the PMS tool.

```
{
        "project_id": "1cf55b98-db69-46f8-a64e-a531d512d4d3",
        "elements": {
                "2RKo5Qbz5FeBxTqQLoNXq6": {
                        "zone": "0hozoFnxj9leOc81mNbdT5",
                        "name": "Part:239050"
                },
                "2RKo5Qbz5FeBxTqQLoNXq5": {
                        "zone": "0hozoFnxj9leOc81mNbdT5",
                        "name": "Part:239049"
                }
                ...
        },
        "roles": {

                "1": {
                        "cost_per_hour": 2000,
                        "quantity": 2,
                        "name": "Truck mounted concrete boom pump",
                        "type": "Equipment"
                },
                "2": {
                        "cost_per_hour": 1500,
                        "quantity": 4,
                        "name": "Concrete mixer truck",
                        "type": "Equipment"
                }
                ...
        },
        "project_name": "Project_1",
        "zones": {
                "0hozoFnxj9leOc81mNbdSu": {
                        "name": "01 - Entry Level"
                },
                "0hozoFnxj9le0c81mNbdT5": {
                        "name": "Roof"
                },
                "0hozoFnxj9le0c81mNbdSx": {
                        "name": "02 - Floor"
                },
                "0hozoFnxj9le0c81mNbdSw": {
                        "name": "03 - Floor"
                }
        },
        "tasks": {
                "11": {
                        "end_date": "2010-01-26T17:00:00",
                        "element_list": [
                                "29FKXAz_b8mfMby7Zgrevz",
                                "29FKXAz_b8mfMby7Zgrh1x",
                                "29FKXAz_b8mfMby7Zgren8"
                        ],
                        "name": "Emergency staircase shaft",
                        "start_date": "2010-01-18T08:00:00"
                },
                "33": {
                        "end_date": "2009-12-04T17:00:00",
                        "element_list": [
                                "01U2Ox69TF78CjGAzXHCiE",
                                "01U2Ox69TF78CjGAzXHCcM",
                                "01U2Ox69TF78CjGAzXHApE",
                        ],
                        "name": "Piles and Caps",
                        "start_date": "2009-11-23T08:00:00"
                }
                ...
        }
}
```

5.1.6 Licensing

The DT Runtime is a closed-source component. On the other hand, the libraries containing the actors defined within COGITO are open source. The COGITO developers can extend the functionalities of the DTP by configuring, implementing, and deploying new actors. The lead group in charge of development provides access to additional documentation and scripts to help developers implement new actors and deploy new modules.

5.1.7 Installation Instructions

This component is deployed as a standalone web-based application in a Virtual Machine (VM) and is part of the Data Management Layer. It provides a GUI and a REST API open to the internet. No file download, installation or maintenance is required by the users.

5.1.8 Development and Integration Status

The first release of the DT Runtime component has been deployed in the Data Management Layer. This component provides all core functionalities that have been identified in “D7.1 – Digital Twin Platform Design & Interface Specification v1”. In the final release, the team in charge of the development will improve the actor system and the various messaging adapters, and it will provide a library containing the final actors implementing the data processing operations of the COGITO system.

5.1.9 Requirements Coverage

The DT Runtime component covers most of DTP’s functional and non-functional requirements listed in “D2.5 – COGITO System architecture v2”. The complete list of the functional and non-functional requirements related to the DT Runtime component are presented in Table 15. The Req-1.3 and Req-1.7 are fully covered by the DT Runtime component’s REST API, which allows the COGITO tools to upload and download files from the DTP. The Req-1.4 is fully covered by the various messaging adapters implemented in the DT Runtime component, which are configured via the GUI and are in charge of routing the streaming data to the active modules. The Req-1.6 is achieved by the actor-based system, which provides data integration and a real-time execution environment allowing developers to define complex routing scenarios using the defined actors. On the other hand, the actor-based system used within the DT runtime component, covers all non-functional requirements. The Akka Framework provides the Akka Cluster technology that allows the deployment of the actor system in multiple machines offering horizontal scalability and high availability. The Akka Framework has a good performance of ~50 million messages per second on a single machine and around ~2.5 million actors per GB of heap memory.

Table 15 DT Runtime component's Requirements Coverage

Type	ID	Description	Status
Functional	Req-1.3	Receives as-built data (video, images, and point-clouds)	Achieved
	Req-1.4	Handles real-time data of location tracking sensors	Achieved
	Req-1.6	Orchestrates the execution of the included ETL services	Achieved
	Req-1.7	Manages the data requests of the COGITO tools by providing configurable API and the execution environment	Achieved
Non-Functional	Req-2.1	Scalability	Achieved
	Req-2.2	Responsiveness	Achieved
	Req-2.3	Security	Achieved
	Req-2.4	High availability	Achieved

5.1.10 Assumptions and Restrictions

The first release of the DT Runtime component has been deployed under certain assumptions and restrictions, listed below:

- It has been developed from scratch following the MVP approach providing the core functionalities essential for responding to data requests performed by the other COGITO tools. The final release will provide additional functionalities and an improved GUI.
- It currently contains modules supporting only the requests of UC1.1. In the final release, the DT Runtime component will include modules covering the requests for all UCs of the COGITO system.
- The interconnection between the applications' configurable endpoints and the DT Runtime component is based on the Spring Events technology. In the final release, other technologies will be tested and evaluated.
- It currently handles real-time location tracking data and notifications coming from MQTT brokers. The final release will support additional protocols such as KAFKA and SSE.

6 Conclusions

This demonstrator deliverable presented in detail the first release of COGITO’s DT Platform. The DTP plays a central role in COGITO’s system as it is responsible for i) providing an authentication and authorisation mechanism to COGITO users and applications, ii) handling input data from various external sources such as BIM authoring tools, project management tools, cameras, LiDAR scanners and IoT devices, and iii) responding to data requests performed by the other COGITO tools. In the first version of this deliverable, four core components of the DTP were presented along with the functionalities they provide, the technology stacks they build upon, the interfaces they use, the usage instructions and the assumptions and restrictions.

The DTP is based on a multi-layered architecture comprising six core layers following the “D7.1 – Digital Twin Platform Design & Interface Specification v1”. The components presented in this demonstrator deliverable have been deployed in the various layers of the DTP based on their functional and non-functional requirements. More specifically, the first release of the DTP includes i) the Identity Provider contained in the Authentication Layer, responsible for providing an identity and access management solution, ii) the Input Data Management component contained in the Data Ingestion Layer, responsible for managing users, roles, projects and loading the as-planned input data, iii) the Knowledge Graph Generator contained in the Data Ingestion Layer, responsible for generating COGITO’s knowledge graphs and Thing Descriptions, and iv) The DT Runtime component contained in the Data Management Layer, responsible for creating and hosting various application-driven modules used in the various data processing operations.

The final version of the DTP is expected to be released along with the corresponding deliverable “D7.10 – Digital Twin Platform v2” on M24. Thus far, the software components have been tested only with example files obtained from various online sources. The next version of DTP’s software components will be tested on the COGITO’s pre-validation sites in “T8.2 – COGITO ICT System Pre-Validation” and the validation sites in “T8.4 – Demonstration of COGITO Tools and Construction Projects” with real data.

References

[1] COGITO, “D7.1 - COGITO Digital Twin Platform,” 2021.

[2] COGITO, “D2.5 - COGITO System Architecture v2,” 2022.

[3] COGITO, “D2.1 - Stakeholder requirements for the COGITO system,” 2021.

[4] COGITO, “D3.3 - COGITO Data Model & Ontology Definition and Interoperability Design v2,” 2022.

COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310