

COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

D7.10 –
Digital Twin
Platform

v2



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 965310

D7.10 – Digital Twin Platform v2

Dissemination Level:	Public
Deliverable Type:	Demonstrator
Lead Partner:	UCL
Contributing Partners:	Hypertech, UPM
Due date:	31-10-2022
Actual submission date:	21-12-2022

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
Raúl García-Castro	UPM	rgarcia@fi.upm.es
Giorgos Giannakis	Hypertech	g.giannakis@hypertech.gr
Apostolos Papafragkakis	Hypertech	a.papafragkakis@hypertech.gr

Reviewers

Name	Beneficiary	Email
Giorgos Giannakis	Hypertech	g.giannakis@hypertech.gr
Panos Andriopoulos	QUE	panos@que-tech.com

Version History

Version	Editors	Date	Comment
0.1	UCL	01.10.2022	ToC
0.3	UCL	10.10.2022	Draft version of sections 1,2,3,4,5
0.6	UCL, UPM	14.12.2022	Contributions in section 4
0.8	Hypertech, QUE	18.12.2022	Internal review
0.9	UCL	20.12.2022	Internal review comments addressed
1.0	UCL, Hypertech	21.12.2022	Submission to EC

Disclaimer

©COGITO Consortium Partners. All right reserved. COGITO is a HORIZON2020 Project supported by the European Commission under Grant Agreement No. 958310. The document is proprietary of the COGITO consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights. The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Communities. Neither the European Union

institutions and bodies nor any person acting on their behalf may be held responsible for the use, which may be made, of the information contained therein.



Executive Summary

The COGITO Deliverable “D7.10 - Digital Twin Platform v2” documents the final version of the COGITO Digital Twin Platform and reports the outcomes of work performed 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 offering a range of integration and connectivity options to the COGITO tools. The DTP is responsible for: a) providing authentication and authorisation to COGITO 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 tools. It offers enterprise features such as a Messaging Layer enabling asynchronous communication between the DTP and the COGITO tools, a sophisticated routing system allowing developers to configure complex routing scenarios and a runtime environment for running configurable data-driven modules serving the data requirements of the various COGITO tools. 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 in “T7.5 - Digital Twin Platform Development and Testing” is the implementation, deployment and testing of the DTP that comprises a set of core 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 “T7.2 – Digital Twin Platform Design & Interface Specification v2”. This document focuses on presenting the final version of the DTP and its 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.

Table of contents

Executive Summary	3
Table of contents	4
List of Figures	6
List of Tables	7
List of Acronyms	8
1 Introduction	9
1.1 Scope and Objectives of the Deliverable	9
1.2 Relation to other Tasks and Deliverables	10
1.3 Structure of the Deliverable.....	10
1.4 Updates to the first version of the Digital Twin Platform	11
2 Digital Twin Platform	12
2.1 Components Classification	12
2.2 Components Target Platform	13
3 Authentication and Authorisation Infrastructure	16
3.1 User Authentication and Authorisation	16
3.1.1 Overview	16
3.1.2 Technology Stack and Implementation Tools	16
3.1.3 API Documentation.....	17
3.1.4 Usage Walkthrough	18
3.1.5 Application Example	19
3.1.6 Licensing	19
3.1.7 Installation Instructions	19
3.1.8 Development and Integration Status	20
3.1.9 Requirements Coverage	20
3.1.10 Assumptions and Restrictions	20
4 Project Creation and Ontology Population	21
4.1 Input Data Management component	21
4.1.1 Prototype Overview.....	21
4.1.2 Technology Stack and Implementation Tools	22
4.1.3 Input, Output and API Documentation	23
4.1.4 Usage Walkthrough	25
4.1.5 Licensing	28
4.1.6 Installation Instructions	28
4.1.7 Development and Integration Status	28
4.1.8 Requirements Coverage	28

4.1.9	Assumptions and Restrictions	29
4.2	Knowledge Graph Generator	30
4.2.1	Prototype Overview.....	30
4.2.2	Technology Stack and Implementation Tools	30
4.2.3	Input, Output and API Documentation	31
4.2.4	Application Example	32
4.2.5	Licensing	34
4.2.6	Installation Instructions	34
4.2.7	Development and Integration Status	34
4.2.8	Requirements Coverage	34
4.2.9	Assumptions and Restrictions	35
5	Data Processing and Data Delivery.....	36
5.1	Digital Twin Runtime component	36
5.1.1	Prototype Overview.....	36
5.1.2	Technology Stack and Implementation Tools	37
5.1.3	Input, Output and API Documentation	38
5.1.4	Usage Walkthrough	40
5.1.5	Application Example	49
5.1.6	Licensing	51
5.1.7	Installation Instructions	51
5.1.8	Development and Integration Status	51
5.1.9	Requirements Coverage	51
5.1.10	Assumptions and Restrictions	52
6	Conclusions	53
	References	54

List of Figures

Figure 1 DTP's overall architecture along with its software components	12
Figure 2 Identity Provider's user authentication process	16
Figure 3 Identity Provider's user login page	18
Figure 4 Identity Provider's user registration page	18
Figure 5 Identity Provider's Account Management console	19
Figure 6 Identity Provider's user authentication example	19
Figure 7 Example of a knowledge graph generated by the KGG component	21
Figure 8 IDM's sub-components along with their interactions	22
Figure 9 View of all user's projects	25
Figure 10 Creation of a new project	25
Figure 11 Assignment of registered users to a project	26
Figure 12 Creation of a new property in a project	26
Figure 13 Uploading the as-planned data using the IDM component	27
Figure 14 View of all registered users	27
Figure 15 Assigning a role to the user account	28
Figure 16 High-level Architecture of Knowledge Graph Generation	30
Figure 17 Generation of COGITO's TTLs and TDs	32
Figure 18 Example of RDF generated by the ETL tools contained in the KGG component	33
Figure 19 Example of Thing Description generated by the Thing Manager	33
Figure 20 Digital Twin Runtime component's main interactions	37
Figure 21 Example of DT Runtime component's configurable module and its IFC output	37
Figure 22 View of all user's applications	41
Figure 23 Creation of a new application	41
Figure 24 Assignment of a role to the application	42
Figure 25 Creation of new endpoint	43
Figure 26 Creation of a new Data Collection	43
Figure 27 Uploading and annotating files in a Data Collection	44
Figure 28 View of all user's connections	44
Figure 29 Creation of a new connection	45
Figure 30 Creation of a new messaging channel	45
Figure 31 Creation of a new Actor	46
Figure 32 Creation of a new property in an Actor	46
Figure 33 View of all created modules	47
Figure 34 Creation of a new module	48
Figure 35 Workspace for creating and configuring modules	49
Figure 36 DT Runtime component's system console	49
Figure 37 Part of the UC1.1 sequence diagram	49
Figure 38 The module UC.1.1 – Return 6 which provides the as-planned 4D BIM data	50

List of Tables

Table 1 Main characteristics of DTP's software components	14
Table 2 Libraries and Technologies used in the Identity Provider	16
Table 3 Identity Provider's Authentication API	17
Table 4 Identity Provider's Admin API	17
Table 5 Identity Provider's requirements coverage	20
Table 6 Libraries and Technologies used in the Input Data Management component	22
Table 7 Input Data Management component's REST API	23
Table 8 IDM component's Requirements Coverage.....	28
Table 9 Libraries and Technologies used in the Thing Manager and the Wrapper module	31
Table 10 Thing Manager's REST API	31
Table 11 Thing Description Directory's REST API	32
Table 12 Knowledge Graph Generator URLs	34
Table 13 KGG component's Requirements Coverage	34
Table 14 Libraries and Technologies used in the Digital Twin Runtime component	38
Table 15 DT Runtime component's REST API Endpoints	38
Table 16 DT Runtime component's Requirements Coverage	51

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
GCC	Geometric Clash Checker
GUI	Graphical User Interface
IFC	Industry Foundation Classes
IoT	Internet of Things
JMS	Java Messaging System
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MC	Model Checking
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 final implementation of the DTP reflecting the outcomes of the work conducted 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 ontologies of “T3.2 - COGITO Data Model, Ontology Definition and Interoperability Design”. This work presents the final 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 access is restricted to specific roles and groups. The *Data Ingestion Layer* is responsible for loading the input data and orchestrating the execution of the Extract, Transform and Load (ETL) and Model Checking (MC) services to generate the knowledge graph and populate 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 COGITO applications by providing a runtime system, which hosts a set of configurable modules performing well-defined business logic operations. At this point, a COGITO application sends a request for data, and the *Data Management Layer* responds to the request using asynchronous communication channels provided by the *Messaging Layer*. Finally, the *Data Post-Processing Layer* provides software components responsible for performing ETL and MC operations on BIM data which conform to the Industry Foundation Classes (IFC) standard [2].

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 final 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 final 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.
- **DT Runtime:** This component is part of the *Data Management Layer* and is responsible for creating and hosting various 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.
- **DT Library:** This component is part of the *Data Management Layer* and provides a set of reusable ready-made coding blocks facilitating COGITO developers to create their own data-driven modules using a web-based graphical environment.

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

- **Message Broker:** This component is part of the *Messaging Layer* and is responsible for transmitting asynchronously messages and notifications between the COGITO applications and the DTP. It is based on the Apache ActiveMQ Artemis message broker which supports various messaging protocols such as AMQP, STOMP and MQTT.
- **MVD Checker:** This component is part of the *Data Post-Processing Layer* and is responsible for performing MVD-based data completeness checking on the BIM model.
- **B-Rep Generator:** This component is part of the *Data Post-Processing Layer* and is responsible for exporting the geometric information included in the BIM model and generating triangulated B-rep solids of the structural and non-structural elements. It exports the geometric information using open formats such as OBJ and glTF.
- **IFC Optimiser:** This component is part of the *Data Post-Processing Layer* and is responsible for performing lossless compression of a BIM model that conforms to the IFC standard.
- **Geometric Clash Checker (GCC):** This component is part of the *Data Post-Processing Layer* and is responsible for detecting clash and containment errors and creating additional semantic relationships between existing entities of the unified knowledge graph.

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 on 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 design and interface specification of the DTP and its core components based on the work performed in “T7.1 – Digital Twin Platform Design & Interface Specification” and the corresponding deliverable “D7.2 – Digital Twin Platform Design & Interface Specification v2”.
- The deployment and testing of the various ETL and MC components are based on the work performed in “T7.2 – Extraction, Transformation and Loading tools (ETL) and Model-Checking” and the corresponding deliverable “D7.4 – Extraction Transformation & Loading Tools and Model Checking v2”.
- The creation and deployment of DTP’s data-driven modules used for orchestrating the various data processing operations are 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 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 DTP’s overall architecture and the deployment of its core components in the different layers.
- 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 Input Data Management, the BIM Management, and the Knowledge Graph Generator components, which are contained in the *Data Ingestion Layer* and are responsible for loading data from various input data sources, managing projects, populating the COGITO ontologies, and validating the unified knowledge graph.

- Section 5 presents the DT Runtime and the DT Library components, which are part of the *Data Management Layer* and are 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.

1.4 Updates to the first version of the Digital Twin Platform

The first version, “D7.9 – Digital Twin Platform v1”, presented the first release of DTP’s core components in detail. As mentioned in the previous section, DTP plays a central role in COGITO’s solution as it is responsible for: i) providing an authentication and authorisation infrastructure, ii) loading input data files from various sources such as BIM authoring tools, project management tools, cameras, LiDAR scanners and IoT devices, and iii) satisfying the data needs of other COGITO applications by responding to data requests using configurable data-driven modules. Since the submission of the first version, software components have been added or updated providing additional features to meet the functional and non-functional requirements of the COGITO solution. This deliverable comprises the following changes:

- The Data Management Layer is responsible for responding to various data requests performed by the COGITO applications. To achieve this, the DT Runtime is used for the execution of the available data-driven modules and the DT Library is used for providing a set of ready-made coding blocks facilitating COGITO’s developers to create and configure the data-driven modules. In the first release, the functionality of the DT Library has been tested. In the final release, the DT Library contains the complete list of coding blocks needed to support the various UCs.
- The Data Post-Processing Layer is responsible for hosting various time-consuming MC and ETL services utilising asynchronous communication technologies. The first release of the DTP included: i) the MVD Checker component for performing MVD-based data completeness checking, ii) the B-rep Generator component for generating triangulated geometric solids of the structural and non-structural elements, and iii) the IFC Optimiser for performing lossless compression on the BIM data. In the final release, the Geometric Clash Checker (GCC) component has been added for detecting clash and containment errors and creating additional semantic relationships in the knowledge graph.
- The Messaging Layer is responsible for transmitting messages and notifications between the DTP and the other COGITO applications in an asynchronous manner. In the final release, the ActiveMQ Artemis message broker has been fully integrated and tested. The final release of the DT Runtime component offers a Graphical User Interface (GUI) for configuring DTP’s connectors and channels enabling asynchronous communication with the other COGITO tools.

2 Digital Twin Platform

The development activities performed in “T7.5 – Digital Twin Platform Development and Testing” followed the Minimum Viable Product (MVP) approach. The final release of the DTP provides the necessary features to be usable by the developers and the users of the COGITO system [3]. 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 Components Classification

As mentioned in the previous section, the DTP is responsible for loading the as-planned 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.

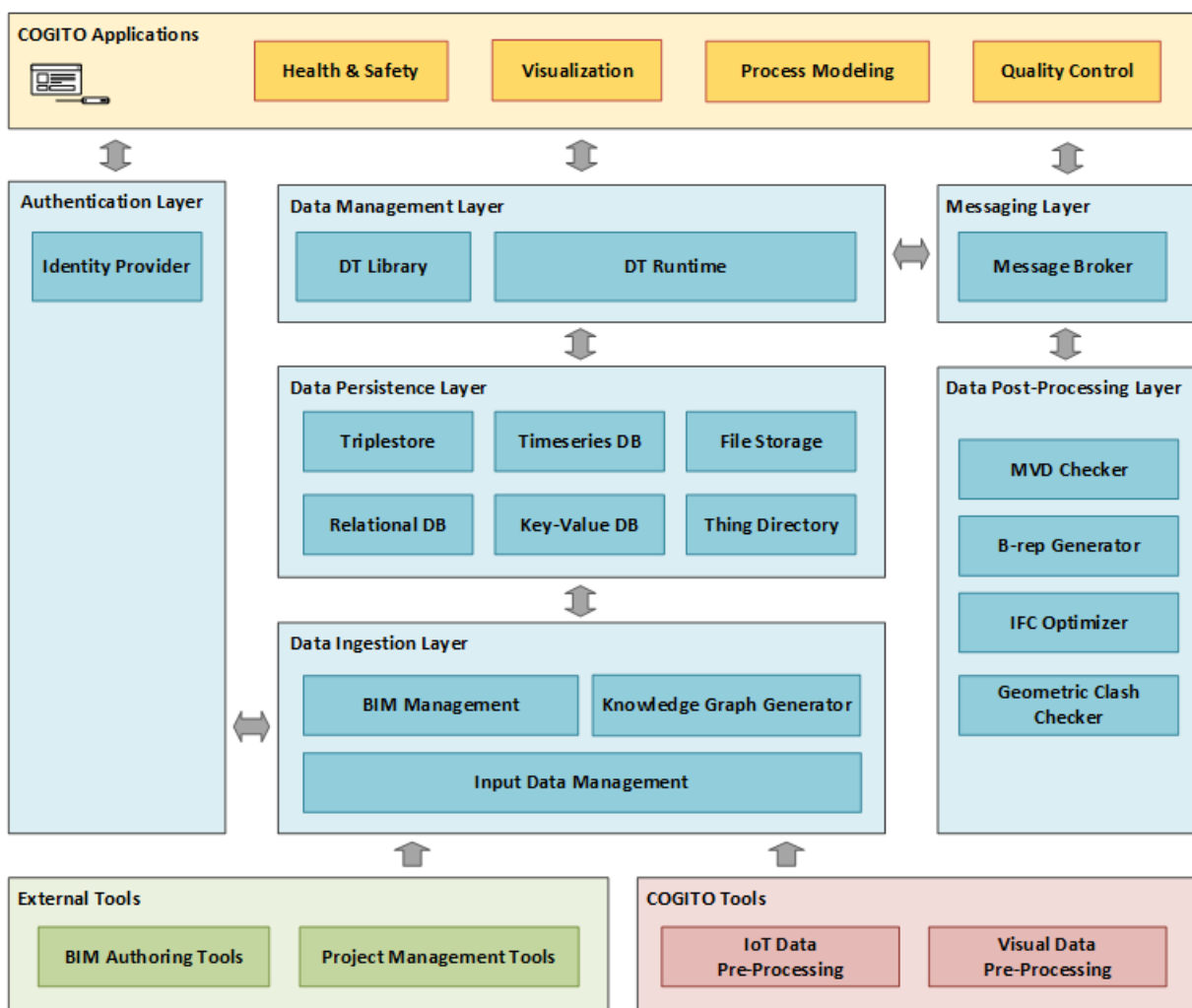


Figure 1 DTP’s overall architecture along with its software components

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 the following categories:

- **Monolithic Applications:** This category contains standalone applications in which the various components of the different layers, such as the business logic, the data access, and the graphical user

interface, are included in a single package deployed on a specific target platform. These applications can interact with others through static endpoints implementing various protocols and security standards. For instance, the Identity Provider, the Input Data Management, and the Message Broker are deployed in the various layers of the DTP as standalone applications.

- **Service-Oriented Architecture (SOA):** This category contains applications implementing the SOA design pattern. Their software components are often packaged and deployed as individual modules on specific target platforms. In this case, the internal software components exchange data through a central messaging system enabling asynchronous communication based on enterprise messaging specifications and protocols. For instance, MVD Checker, the B-rep Generator, the IFC Optimiser, and the Geometric Clash Checker (GCC) are following the SOA design pattern.
- **Microservices:** This category contains applications configured to run on multiple computational nodes. Their software components are packaged as container images and deployed in a cloud-computing infrastructure, providing flexibility, high-availability, and horizontal scalability. In contrast with the SOA, they do not use a central messaging system for exchanging messages. Each application is responsible for providing its endpoints and communication channels. For instance, the data-driven modules installed in DTP's Runtime component are based on the microservices architecture.
- **Software Libraries:** This category contains low-level software packages such as parsers and algorithms used to support other components and applications. They often provide multithread processing and exchange information through well-defined APIs. For instance, the BIM Management component is used in DTP's Data Ingestion Layer as internal dependency package.

2.2 Components Target Platform

Most DTP's software components have a single instance deployed on dedicated computational nodes of a cloud-native environment. In this case, the Nginx server is in charge of forwarding the HTTP requests to the local destinations of the deployed services. For instance, the Identity Provider, the Input Data Management, and the Message Broker use Nginx as a reverse proxy server. On the other hand, some software components can have multiple instances simultaneously. These components utilise the SOA design pattern, exchange data with the DTP through a message broker using asynchronous protocols and offer load-balancing and high-availability capabilities. For instance, the MVD Checker, the B-rep Generator, the IFC Optimiser and the Geometric Clash Checker use such technology. The software components included in the different layers of the DTP are the following:

The **Authentication Layer** contains 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 an advance *Message Broker* that offers an integrated solution enabling asynchronous bi-directional communication between DTP and other COGITO applications. It is based on the open-source project Apache ActiveMQ Artemis² offering a high-performance message broker that supports multiple messaging protocols.

The **Data Ingestion Layer** contains the final 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 final release of the various ETL tools described in "D7.4 - Extraction, Transformation & Loading Tools and Model Checking v2" and, b) the Thing Manager,

² ActiveMQ Artemis <https://activemq.apache.org/>

responsible for supervising the data transformation processes, validating the knowledge graphs and generating the Thing Descriptions.

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

- The *File Storage System* offers an API for storing, retrieving, and deleting files.
- The *Relational Database* is used for storing metadata related to projects, users, roles, registered applications, and data collections.
- The *Timeseries DB* is used for storing IoT data generated by the IoT Data Pre-Processing tool.
- The *Key-Value DB* is used for storing the IFC objects.
- The *Triplestore* is used for storing the RDF data generated by the various.
- The *Thing Description Directory* is used for storing the Thing Descriptions generated by the Thing Manager.

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

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

The **Data Post-Processing Layer** contains a set of components that utilise the SOA design pattern and use the asynchronous messaging 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 a reduced file size.
- The *Geometric Clash Checker* detects clash and containment errors to create additional semantic relationships between existing entities of the knowledge graph.

Table 1 summarises the main characteristics of the software components that have been analysed previously.

Table 1 Main characteristics of DTP's software components

Core Layers	Software Components	Type	Origin
Authentication Layer	Identity Provider	Monolithic Application	Open source
Data Ingestion Layer	Input Data Management	Monolithic Application	Developed from scratch
	BIM Management	Software Library	Developed from scratch
	Knowledge Graph Generator	Microservices	Developed from scratch
Data Persistence Layer	Relational DB (data model)	Monolithic Application	Developed from scratch
	File Storage System	Monolithic Application	Developed from scratch

	Triplestore (server)	Monolithic Application	Open source
	Thing Directory	Monolithic Application	Developed from scratch
	Timeseries DB (server)	Monolithic Application	Open source
	Key-Value DB (server)	Monolithic Application	Open source
Data Management Layer	DT Runtime	Monolithic Application & Microservices	Developed from scratch
	DT Library	Software Library	Developed from scratch
Data Post-Processing Layer	MVD Checker	SOA	Developed from scratch
	B-rep Generator	SOA	Developed from scratch
	IFC Optimiser	SOA	Developed from scratch
	Geometric Clash Checker	SOA	Developed from scratch
Messaging Layer	Message Broker	Monolithic Application	Open source

3 Authentication and Authorisation Infrastructure

The COGITO tools can be classified based on their functional requirements. Some applications require a central system to authenticate and authorise the users, while others require authentication to access DTP's REST APIs. The final version of the Authentication Layer provides an Authentication and Authorisation Infrastructure (AAI), enabling DTP to authenticate and authorise COGITO users. The User Authentication and Authorisation component, aptly named Identity Provider, manage the users and their roles by providing the necessary functionalities to the COGITO applications, such as registration, password recovery, authentication, and authorisation endpoints.

3.1 User Authentication and Authorisation

The AAI solution of the DTP relies on the Keycloak open-source identity and access management solution. Keycloak is an industry-standard identity and access management implementation supporting various protocols such as OpenID Connect and SAML 2.0. Most applications within COGITO use the OpenID Connect protocol, which offers Single Sign-On (SSO) capabilities. On the other hand, PMS uses the SAML 2.0 protocol. The Identity Provider issues an assertion to PMS when a user is authenticated. Then, PMS can grant the user access to its resources based on this assertion. Like OpenID Connect, SAML 2.0 enables SSO capabilities and facilitates secure information sharing within the COGITO system.

3.1.1 Overview

The Identity Provider has been deployed as a 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.

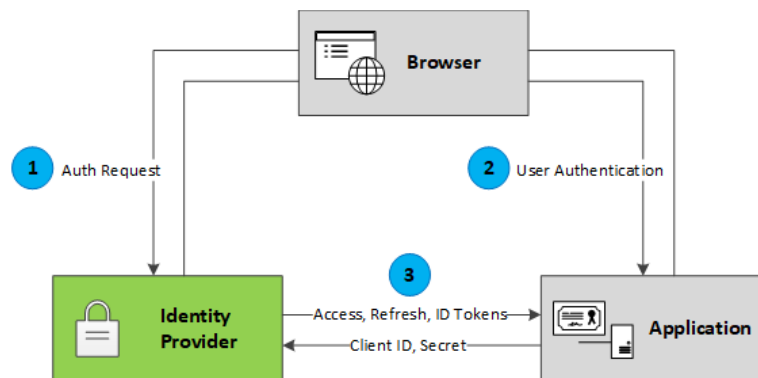


Figure 2 Identity Provider's user authentication process

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

3.1.3 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 authorisation 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 the COGITO application to request the Access, ID and Refresh Tokens.

3.1.3.2 Admin REST API

The Identity Provider offers a fully functional Admin REST API, which offers 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 parameters. Each user role is defined as a JSON object that contains the Role Id and the Role Name. The detailed documentation³ 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 3.

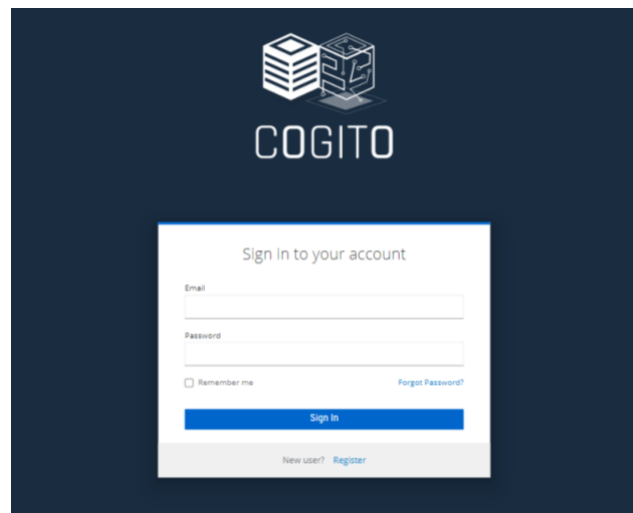


Figure 3 Identity Provider's user login page

Otherwise, a registration process is required. The Identity Provider offers a registration page for registering new users, as shown in Figure 4.

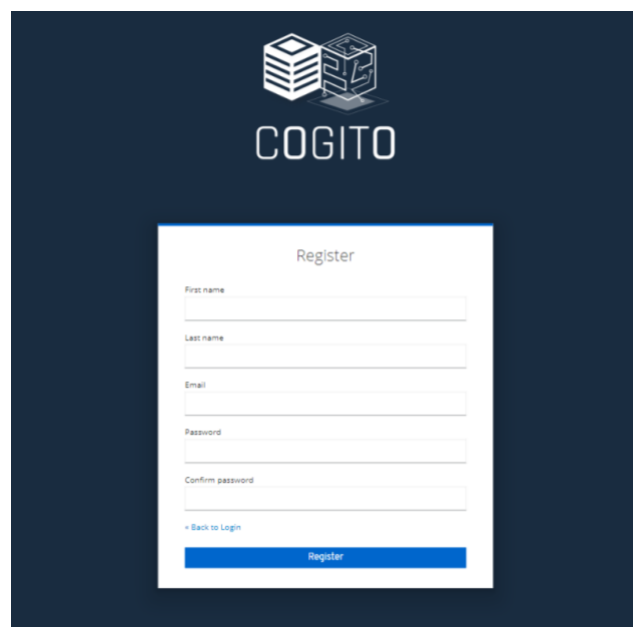


Figure 4 Identity Provider's user registration page

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

³ 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 5. The URL⁴ 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 passwords.

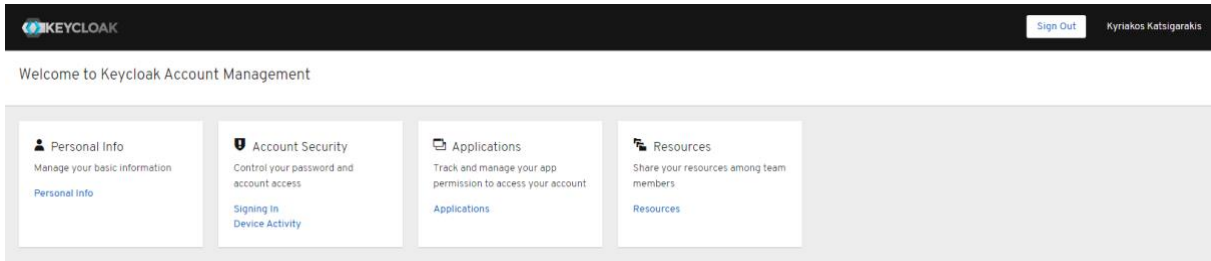


Figure 5 Identity Provider's Account Management console

3.1.5 Application Example

As shown in the example of Figure 6, 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.

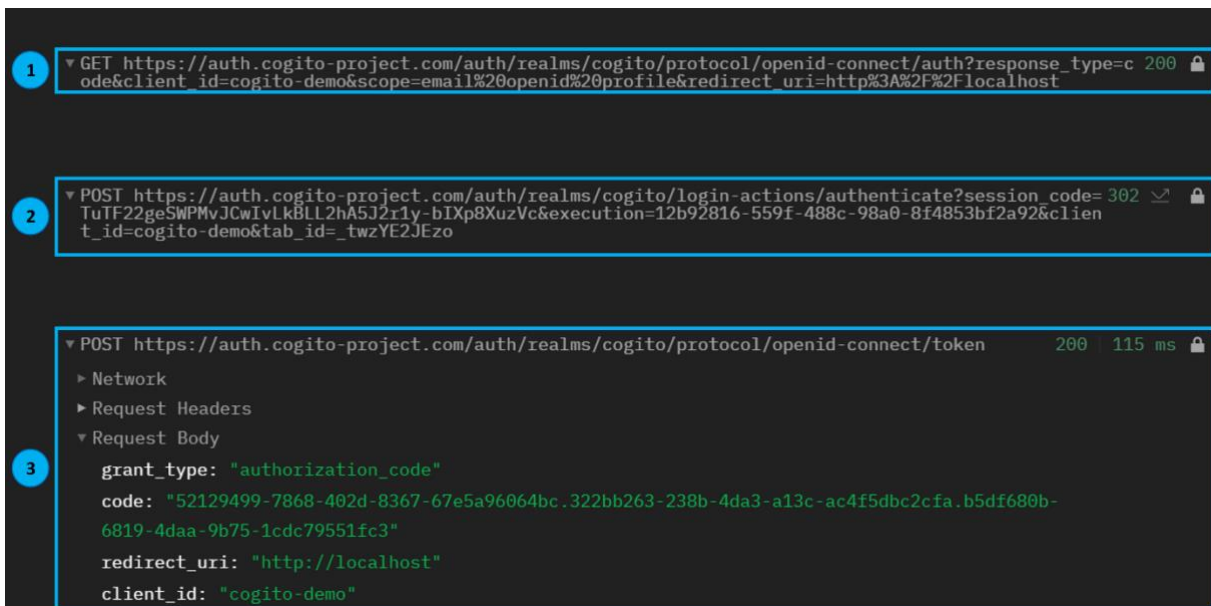


Figure 6 Identity Provider's user authentication example

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 a 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.

⁴ Account Management Console <https://auth.cogito-project.com/...>

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 [4] are configured in the Identity Provider as realm roles. Furthermore, the COGITO tools, which offer GUI and user authentication, are registered in the Identity Provider as realm 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 by using the Nginx server to offer 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 Identity Provider is a monolithic application based on the open-source project Keycloak. It has been configured to manage a single realm instance dedicated to the COGITO system. The various COGITO tools have been configured as realm clients, and their corresponding credentials (Client ID, Secret) have been generated and delivered to COGITO’s tools developers. Furthermore, the user roles have been introduced as realm roles based on the outcomes of “T2.1 - Elicitation of Stakeholder Requirements”. Currently, most of COGITO’s tools developers have tested the core functionalities provided by the Identity Provider. The full integration of the Identity Provider is expected to take place as part of “T8.1 - End-to-end ICT System Integration, Testing and Refinement”, when all involved tools should be capable of providing their full functionality.

4 Project Creation and Ontology Population

One of the core functionalities of the DTP, is to load the as-planned data and 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 quality 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 7 [5]. Furthermore, it generates and stores the corresponding Thing Descriptions in 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 7 Example of a knowledge graph generated by the KGG component

In this section, we present the final 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 final release of the Input Data Management (IDM) component has been successfully 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.

The IDM component also provides a REST API allowing the COGITO applications to interact with the DTP for project creation, user management and loading the as-planned data. In the same way as before, the Spring Security Framework is used to authenticate requests performed by the various COGITO tools using API Keys. The API Key is a unique string that identifies HTTP requests associated with a COGITO tool and ensures that only authorised clients can access the API. COGITO's tool owners and developers can use the IDM component to generate or remove API keys from their accounts. Figure 8 shows the main interactions between the IDM component and the other entities of the DTP.

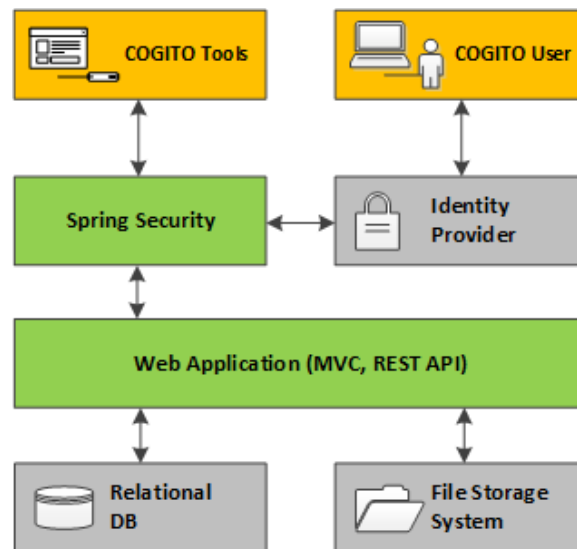


Figure 8 IDM's sub-components along with their interactions

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 a 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 the 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-4c23-97f9-69dff52677d",
    "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 9, 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.

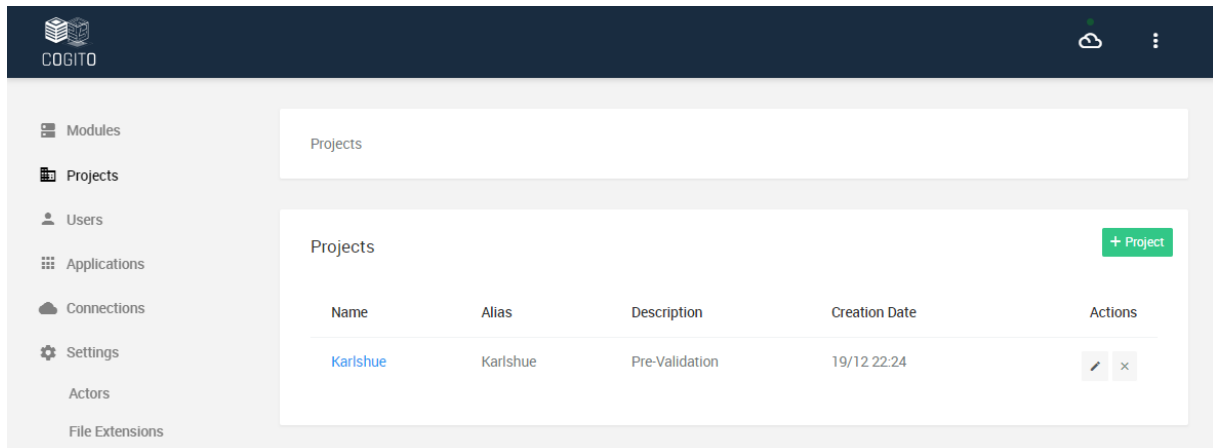


Figure 9 View of all user's projects

On the same page, the users can create a new project by clicking the “+ Project” button. As shown in Figure 10, 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.

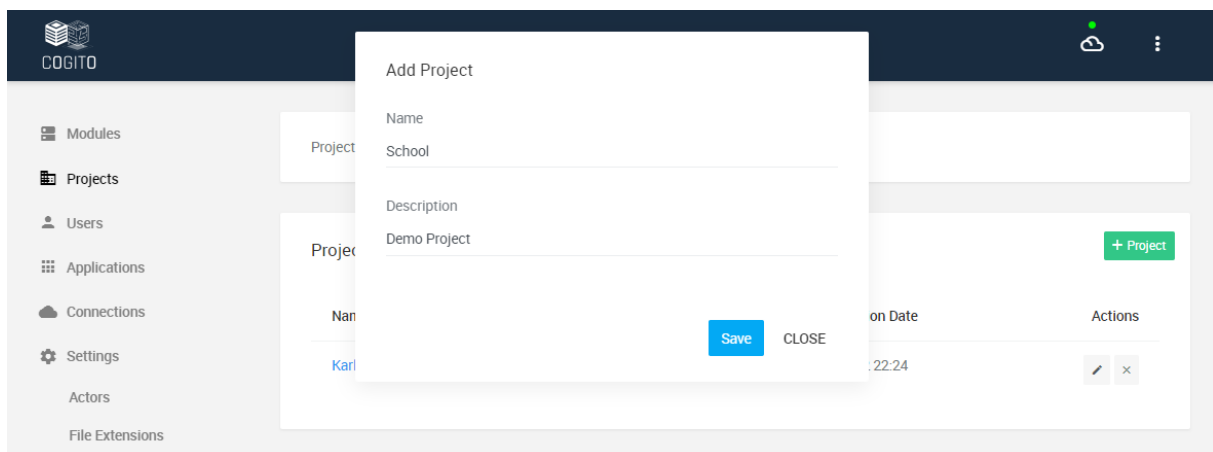


Figure 10 Creation of a new project

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 11, 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.

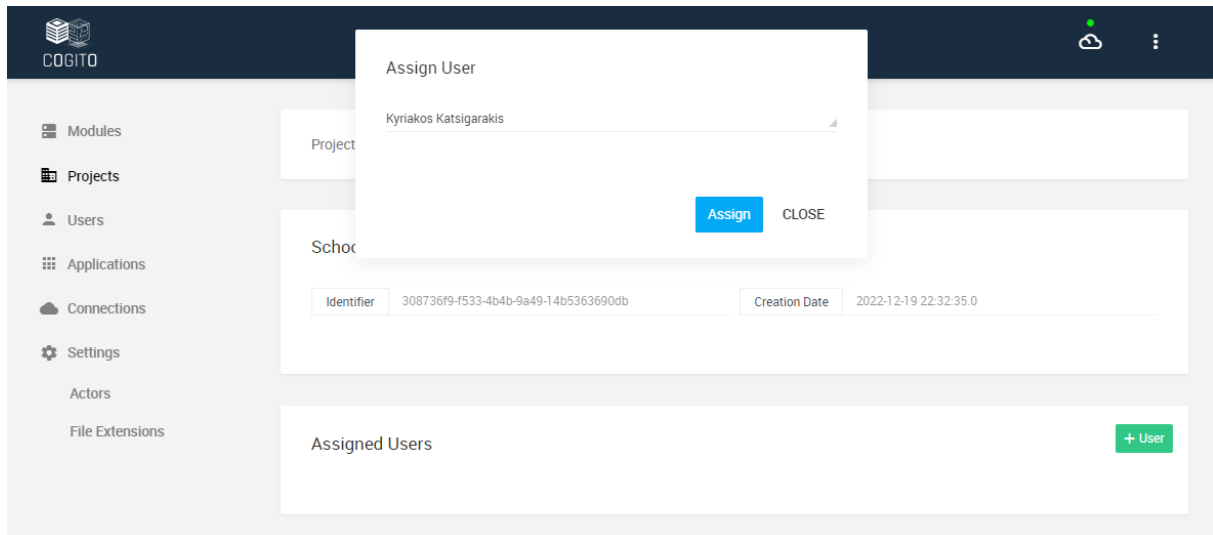


Figure 11 Assignment of registered users to a project

Following the assignment process, users with proper roles can create global properties by clicking the “+ Property” button. As shown in Figure 12, a modal popup window appears allowing users to fill in the property name and its value. There are no restrictions on the creation of project properties.

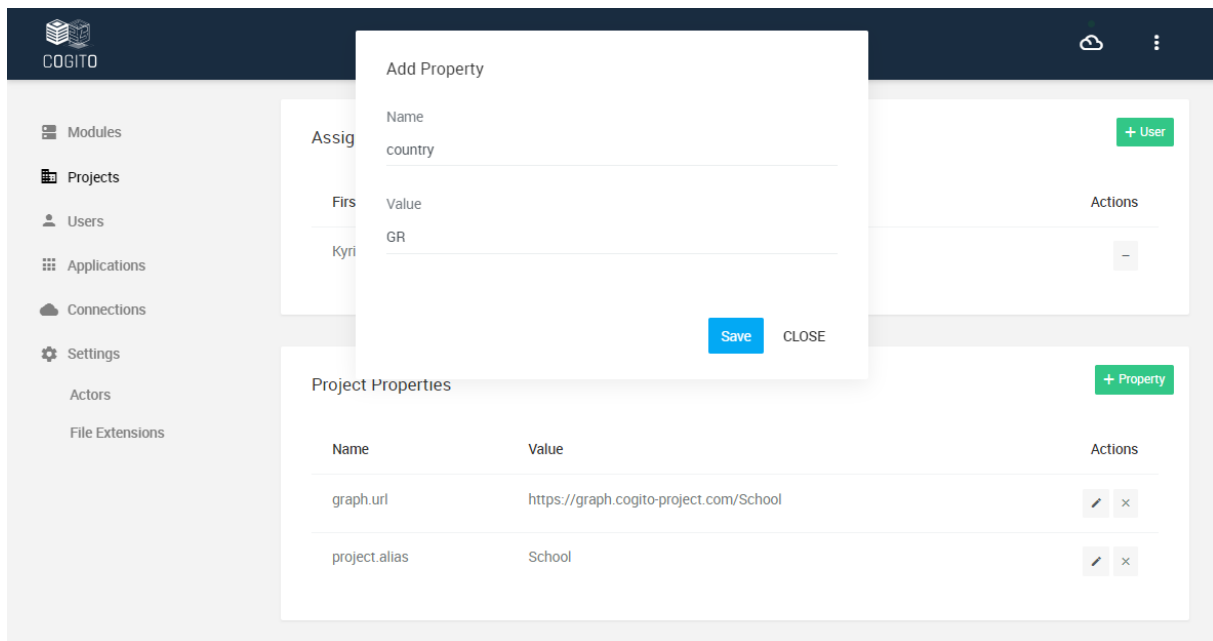


Figure 12 Creation of a new property in a project

On the same page, users can upload the as-planned data by clicking the “Browse” button as shown in Figure 13. 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 "D7.4 - Extraction, Transformation & Loading Tools and Model Checking v2". For instance, the 4D information 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 “D7.4 – Extraction, Transformation & Loading Tools and Model Checking v2”. Upon the upload process, the IDM will store these files in 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.

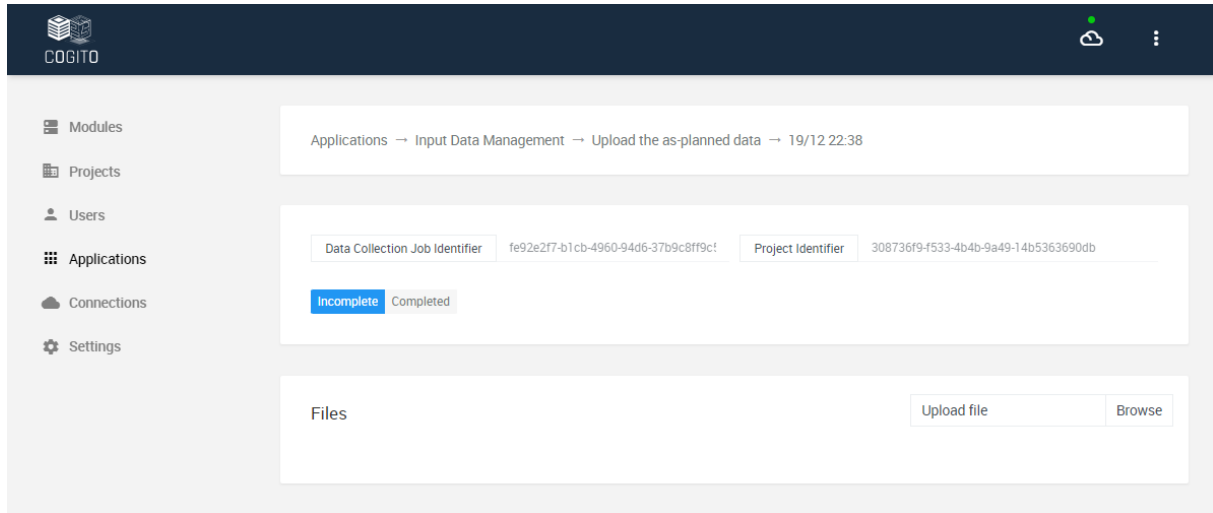


Figure 13 Uploading the as-planned data using the IDM component

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 14. This role allows users to manage the access policies of the registered users.

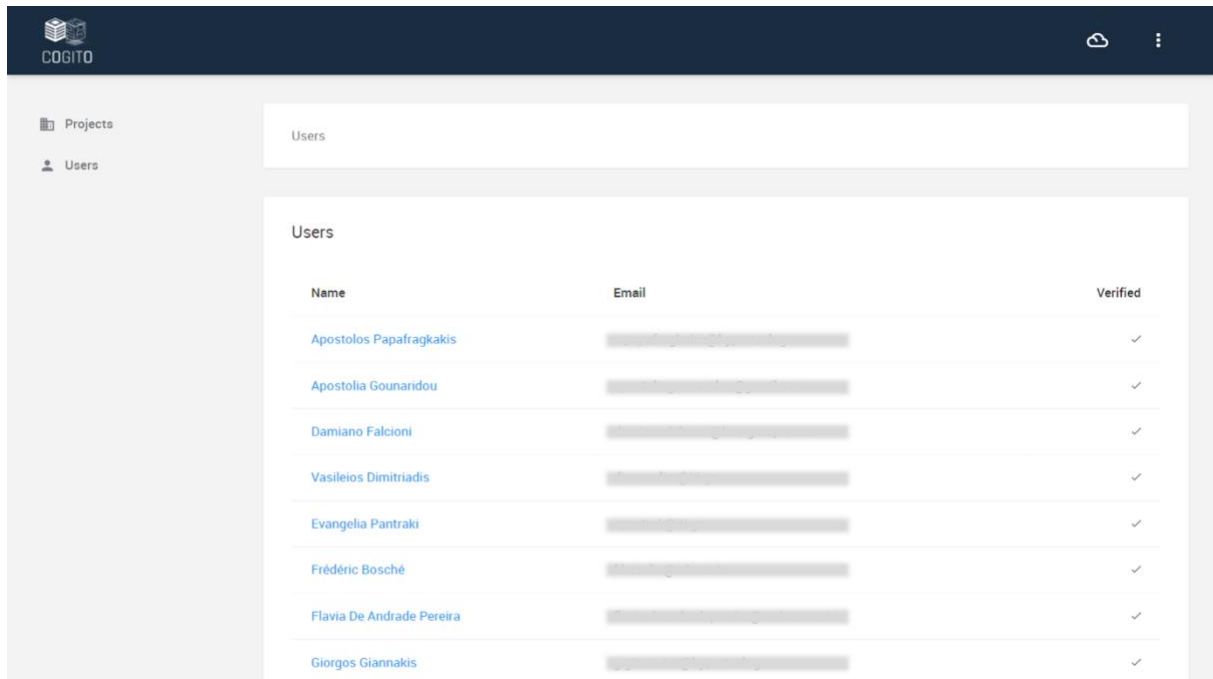


Figure 14 View of all registered users

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 15, 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.

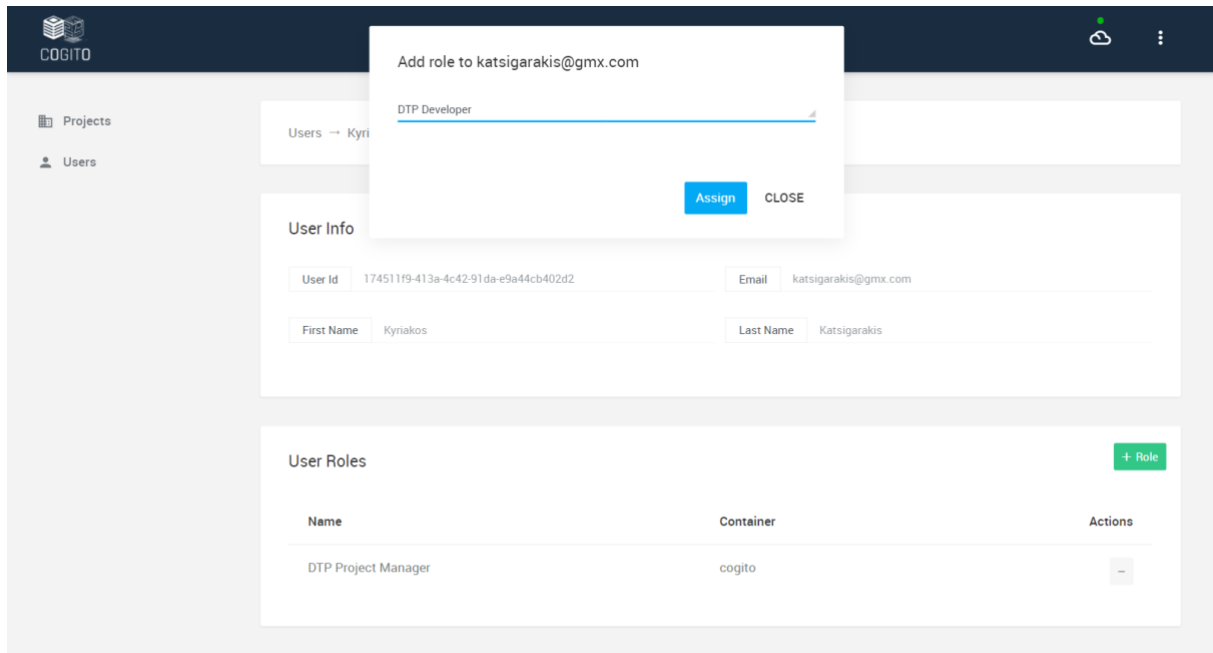


Figure 15 Assigning a role to the user account

4.1.5 Licensing

The IDM is a closed source component. The lead group in charge of the 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 a monolithic 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 final 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 “D7.2 – Digital Twin Platform Design & Interface Specification v2”. The development team will continue providing support and maintaining the IDM component as part of the “T8.1 - End-to-end ICT System Integration, Testing and Refinement” activities.

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 with the help of 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 IDM component is a monolithic application developed from scratch following the MVP approach providing core functionalities essential for creating and managing projects. Currently, it supports both IFC4 and IFC4x3 versions of the IFC specification, although more tests are required. The full integration is expected to take place as part of “T8.1 - End-to-end ICT System Integration, Testing and Refinement”, when all involved tools should be capable of providing their full functionality and data from the pilot sites will be available.

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 Framework⁵ (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 "D7.4 - Extraction, Transformation & Loading Tools and Model Checking v2". The final 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 final release of the core KGG 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 16.

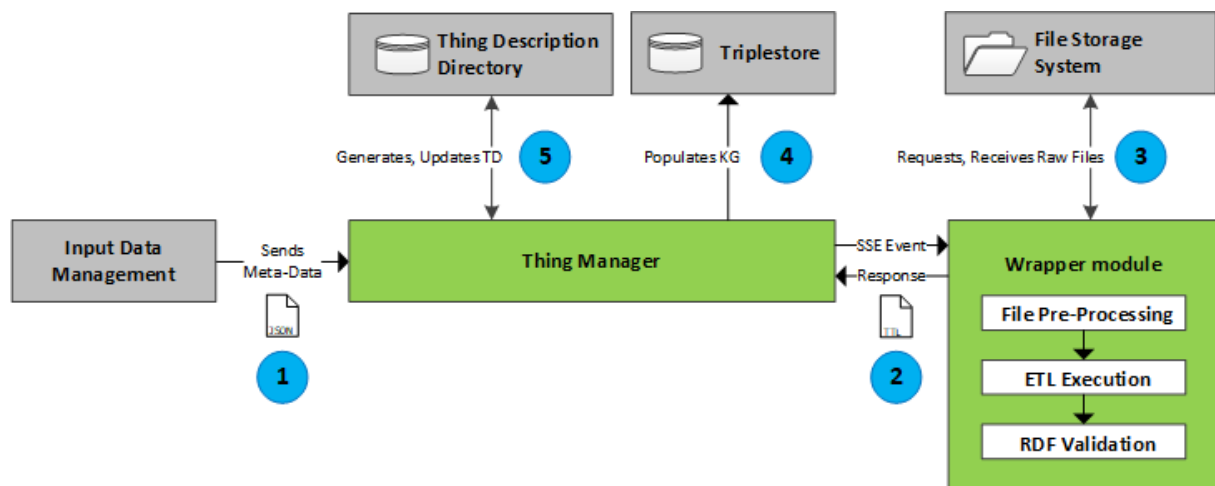


Figure 16 High-level Architecture of Knowledge Graph Generation

A quick overview of Figure 16, 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.

⁵ 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
wheelz.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) metadata: 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) metadata: 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 17. 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.

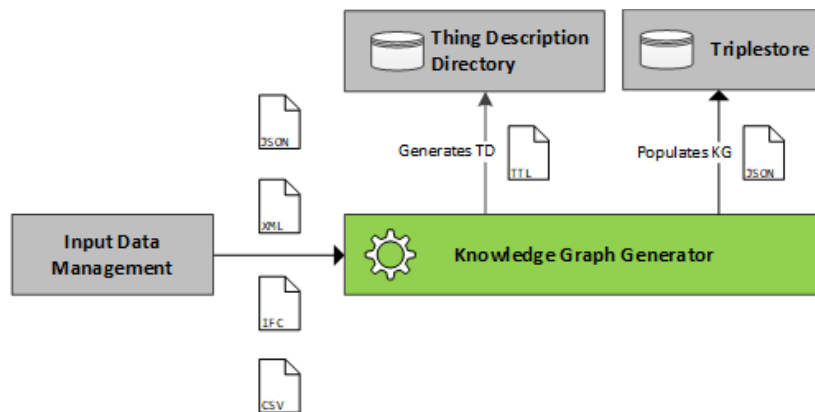


Figure 17 Generation of COGITO's TTLs and TDs

The outputs of the KGG component consist of the TTL data illustrated in Figure 18, and the corresponding JSON-LD data illustrated in Figure 19.

```

@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:taskId '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 18 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-dbd5f-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 19 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 process is deploying a Docker container and configuring a docker-compose file indicating the endpoints of the Thing Directory (TDD), the Triplestore and the Thing Manager (TM). The KGG has been deployed on a private cloud environment, and the provided services are online and open to the internet. The corresponding URLs are listed in Table 12.

Table 12 Knowledge Graph Generator URLs

KGG Component	Description	URL
Thing Manager	Handles the requests done by the DTP and generates the Thing Descriptions.	https://data.cogito.iot.linkeddata.es/tm/
Thing Directory	Stores the Thing Description generated by the Thing Manager.	https://data.cogito.iot.linkeddata.es/tdd/api/
Triplestore	Stores the RDF data.	https://triplestore.cogito.iot.linkeddata.es/
Wrapper module	Performs the requests to the ETL tools for generating and validating the RDF data.	https://data.cogito.iot.linkeddata.es/tm/stream?channel=project-id,bim-file,schedule-file,resources-file

4.2.7 Development and Integration Status

The KGG component implementation and deployment activities are completed. The final version of the Thing Manager and the Wrapper modules have been tested and deployed in DTP's Data Ingestion Layer. They support the generation of the knowledge graph covering the data requirements of the identified UCs. Furthermore, the Thing Manager supports automatic validation of the knowledge graph using SHACL shapes created based on the as-planned data requirements and the generation of the Thing Descriptions based on the RDF data produced by the various ETL tools.

4.2.8 Requirements Coverage

As shown in Table 13, 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 final release of the Thing Manager can orchestrate the data transformation processes and store the output RDF data in the Triplestore.

Table 13 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 Docker configuration file must always contain the endpoints required for the deployment on a cloud-based environment such as the Thing Manager endpoint, 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.
- Thing Descriptions follow the structure defined in the W3C standard.

5 Data Processing 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* is 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 hazard mitigation measures and producing warning 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 visualising 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 final release of the DT Runtime component, which is responsible for orchestrating the internal data processing operations, harmonising 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 Akka⁶ 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 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 containing configurable parameters for interacting with other components, executing its business-logic operations, and forwarding the response to the next actors using the framework's embedded lightweight messaging system. Thus, the actors are created once by the DTP developers, packaged in the DT Library, and then reused in the various configurable modules of the DT Runtime component, as shown in Figure 20. Within COGITO, most of the provided actors are responsible for executing sequences of SPARQL queries to extract information from the knowledge graph and merge the responses into a single JSON file. These actors implement the different data processing operations defined in COGITO's UCs to satisfy the data needs of the various COGITO tools.

⁶ Akka Actor Model <https://www.akka.io>

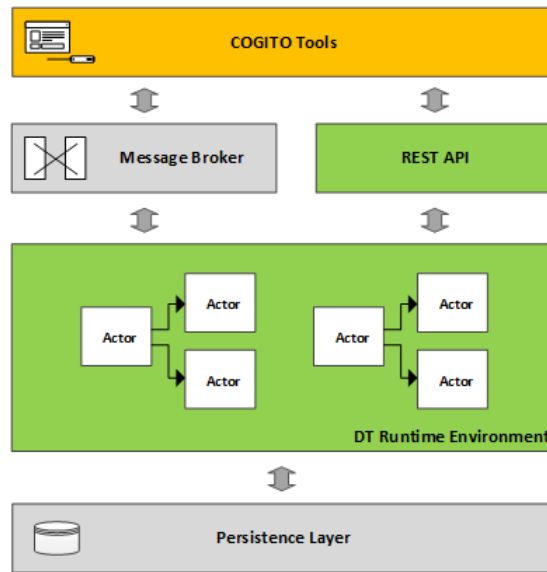


Figure 20 Digital Twin Runtime component's main interactions

For instance, the generation of an IFC file from the corresponding 4D BIM information for a given time frame has a two-step process:

- Execution of a SPARQL query for retrieving the identifiers of the BIM elements that are involved in active tasks.
- 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 21.

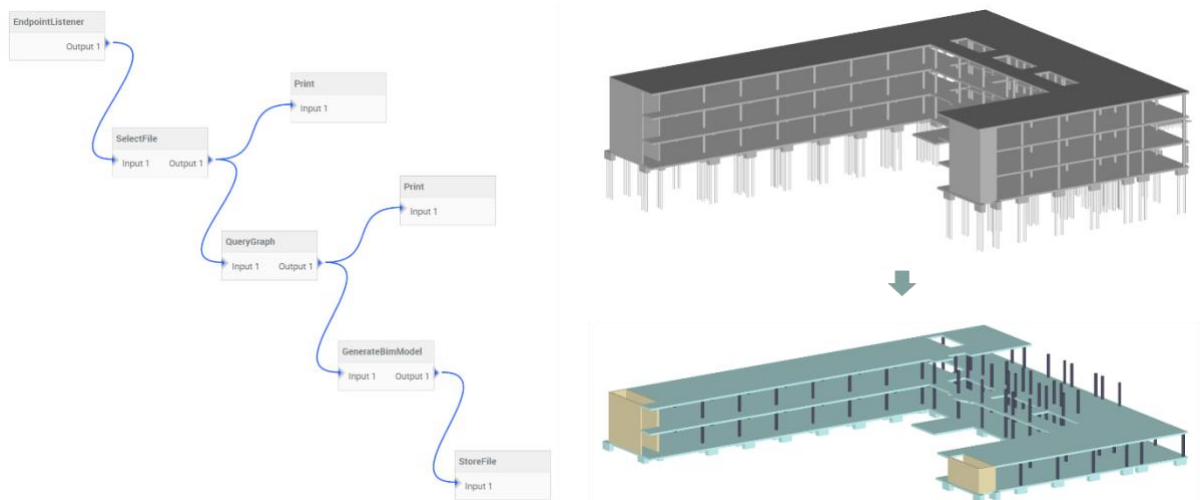


Figure 21 Example of DT Runtime component's configurable module and its IFC output

The final release of the DT Runtime component includes a fully functional REST API for interacting with the COGITO tools and various adapters allowing asynchronous communication through the Messaging Layer utilising enterprise messaging protocols such as AMQP, KAFKA, STOMP and MQTT.

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 a 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 14 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 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 15. The detailed documentation and the Postman collections of DTP’s REST APIs are available online⁷.

Table 15 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 all Data Collections of an Endpoint by Project	GET	https://dtp.cogito-project.com/api/endpoints/{endpointId}/projects/{projectId}/collections	(PP) endpointId: GUID (PP) projectId: GUID
Get a Data Collection	GET	https://dtp.cogito-project.com/api/collections/{collectionId}	(PP) collectionId: GUID

⁷ Digital Twin Platform API documentation <https://api.cogito-project.com>

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
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 all Data Collections of an Endpoint by Project

```
[
  {
    "project": "1cf55b98-db69-46f8-a64e-a531d512d4d3",
    "id": "47368c01-1d50-459a-997d-f8635e3c0511",
    "creation_date": "2022-05-03 11:46:12.0",
    "status": "completed"
  }
]
```

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 into 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 22. 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 with access to applications are created by the specific user.

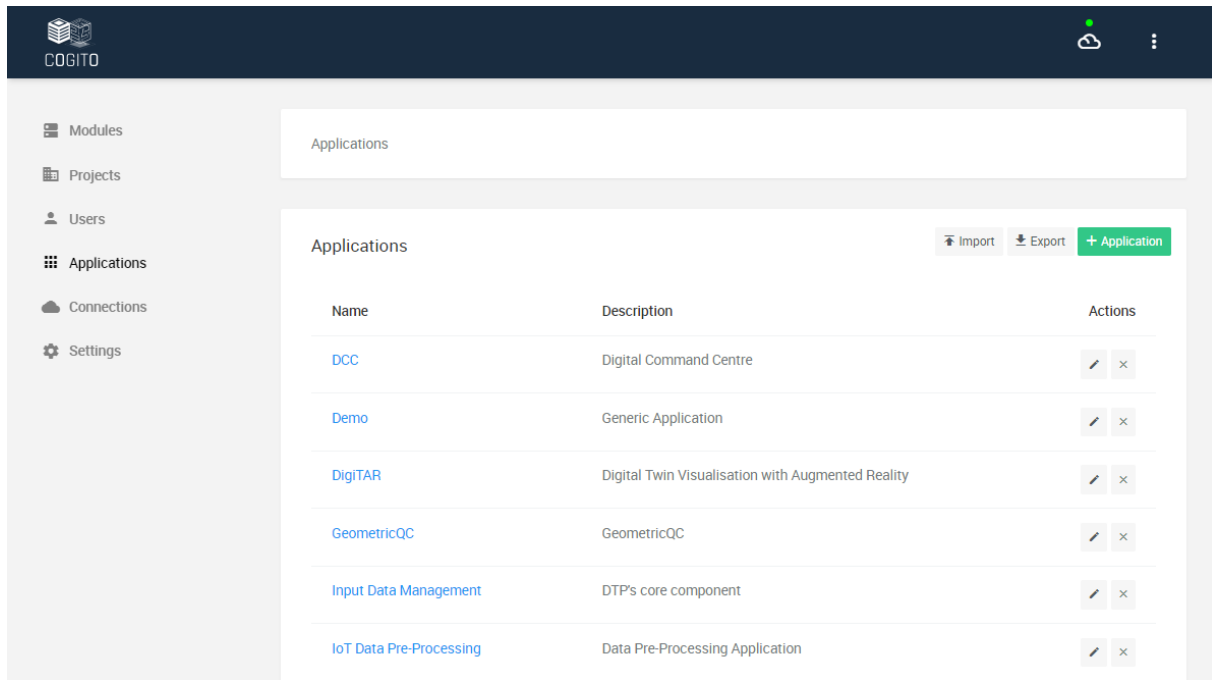


Figure 22 View of all user's applications

On the same page, users who have a developer role can add a new application by clicking the “+ Application” button. As shown in Figure 23, 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.

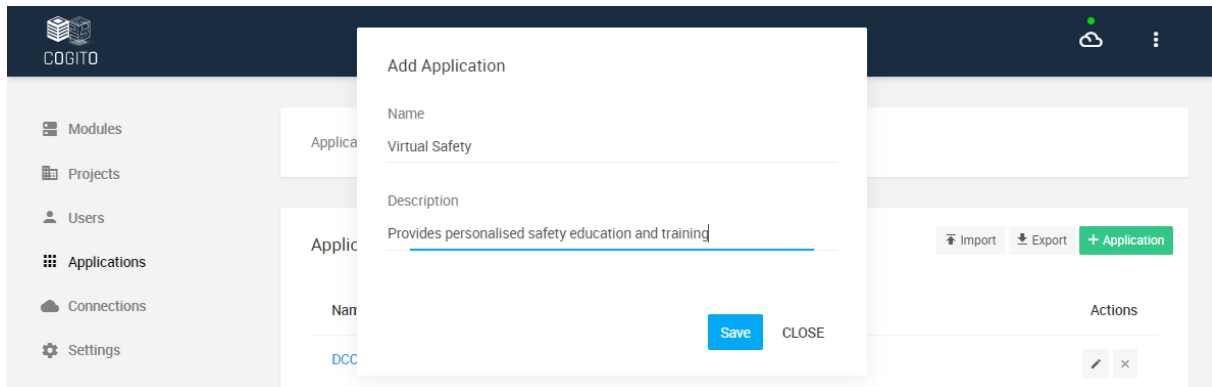


Figure 23 Creation of a new application

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 asynchronous channels. Furthermore, the users can grant access to other users who have different roles by clicking the “+ Role” button. As shown in Figure 24, a modal window appears that contains a combo box element with all available roles.

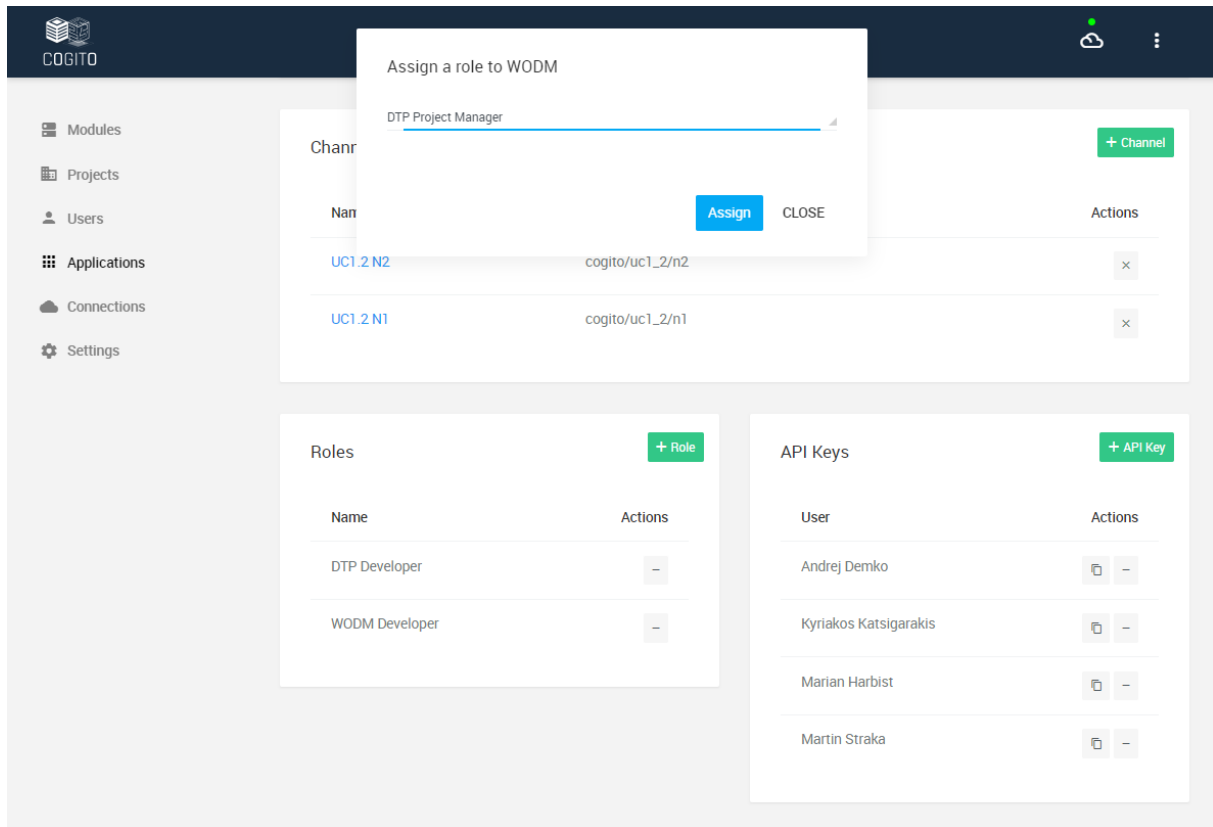


Figure 24 Assignment of a role to the application

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. Continuing with the configuration, users can create new endpoints by clicking the "+ Endpoint" button. As shown in Figure 25, 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.

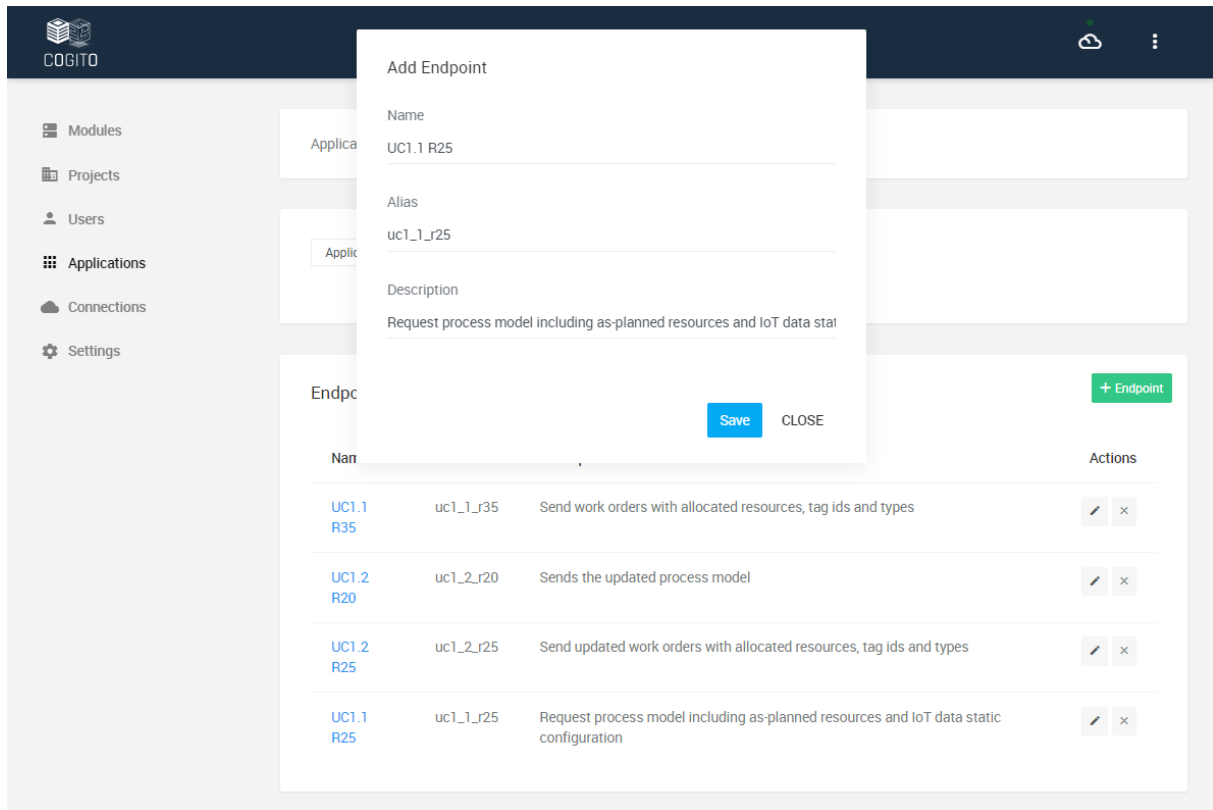


Figure 25 Creation of new endpoint

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 26, 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.

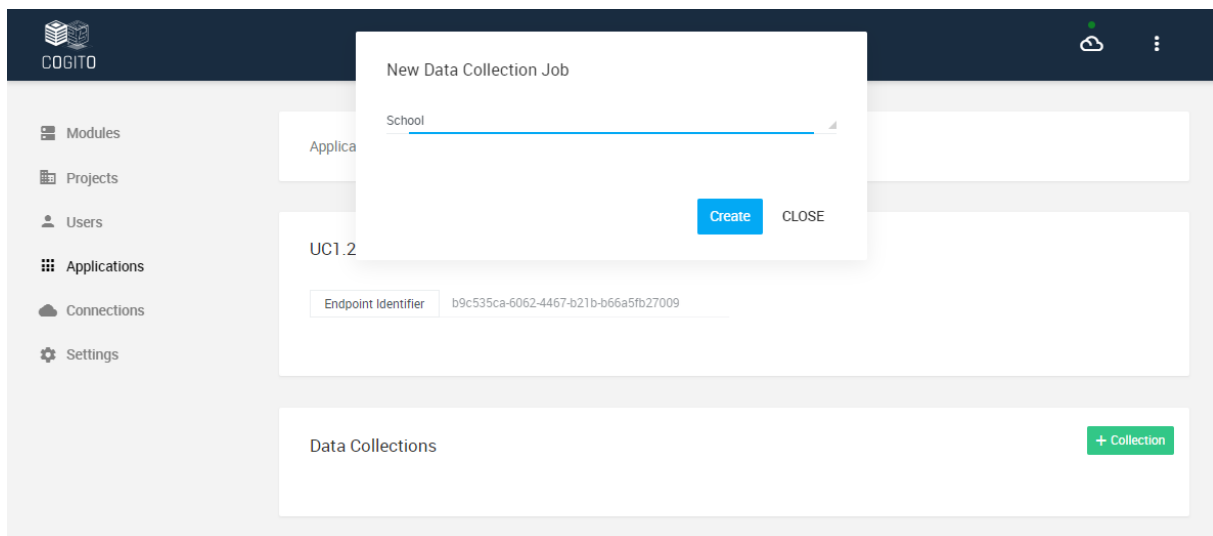


Figure 26 Creation of a new Data Collection

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 with the same format and different data structures. For instance, as shown in Figure 27, 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.

Figure 27 Uploading and annotating files in a Data Collection

The final release of the DT Runtime component can transmit messages and notifications using various asynchronous messaging protocols. The users who have the Developer role can sign-in to the DT Runtime component through the DTP's Identity Provider. The "Connections" button shows the list of the connections configured in the past, as shown in Figure 28.

Figure 28 View of all user's connections

On the right side of the screen the "Actions" column contains buttons for deleting a connection after confirmation and for changing the status of the corresponding connector. By clicking on any connection name, the users can see further information and perform additional actions. The role of the DTP Developer allows access to the complete list of configured connections. In contrast, the other developer roles allow access to connections created by the specific users. The final release of the DT Runtime component allows users with the proper roles to configure AMQP, STOMP, KAFKA and MQTT connections. 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.

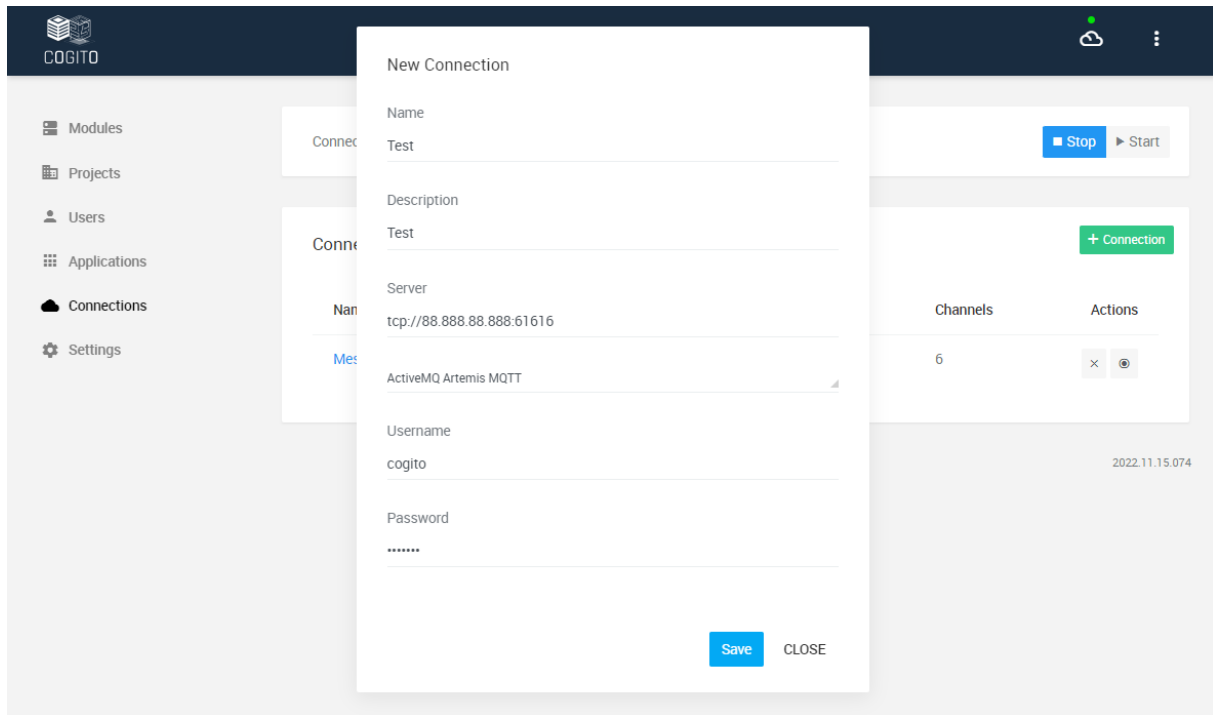


Figure 29 Creation of a new connection

On the same page, users who have a developer role can add a new connection by clicking the “+ Connection” button. As shown in Figure 29, a modal window appears, which allows users to provide the name, the description, the server, the type, and the credentials. Users can cancel the connection creation process at any time by clicking on the Close button. Once the connection is created, the users can create the messaging channels to enable asynchronous communication between the DT Runtime component and the Messaging Layer. As shown in Figure 30, the users can create messaging channels by clicking the “+ Channel” button. A modal window appears, which allows the users to provide the name, the topic, and the type of the messaging channel. At any time, users can cancel the process by clicking on the Close button.

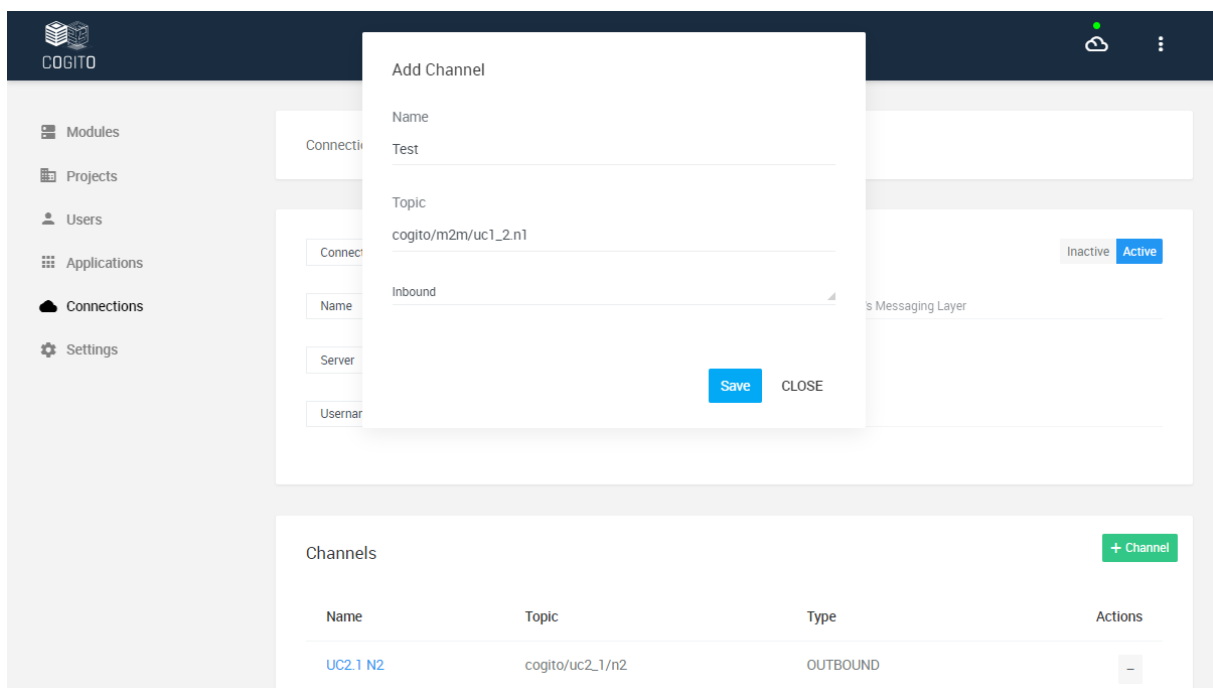


Figure 30 Creation of a new messaging channel

5.1.4.2 Registration of COGITO Actors

The DT Runtime component allows users with proper roles to configure new actors through GUI. These actors are reusable ready-made 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 new actors 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 metadata 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.

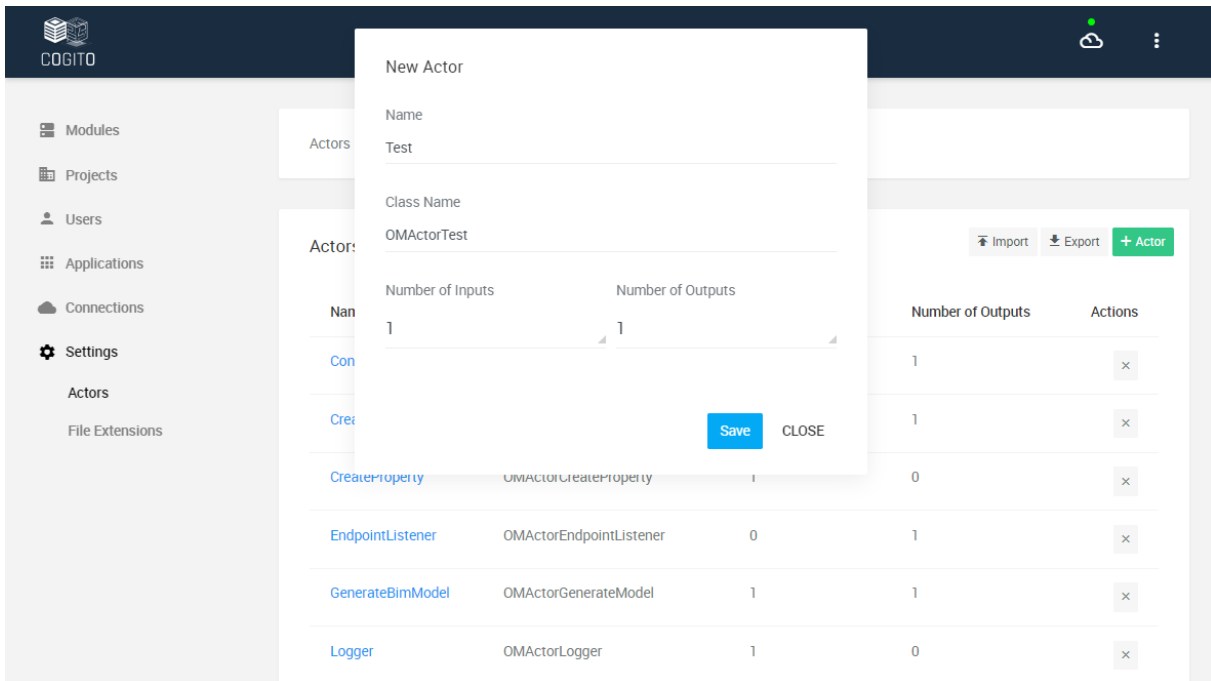


Figure 31 Creation of a new Actor

Once the actor is created, the users can proceed with the definition of its 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.

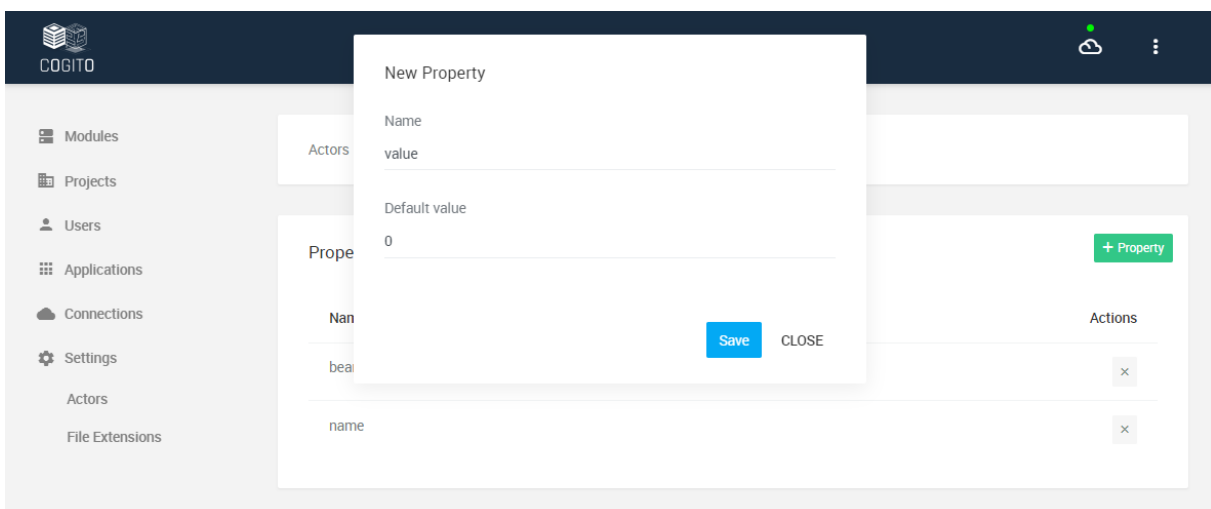


Figure 32 Creation of a new property in an Actor

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 Java classes dynamically into the Java Virtual Machine (JVM).

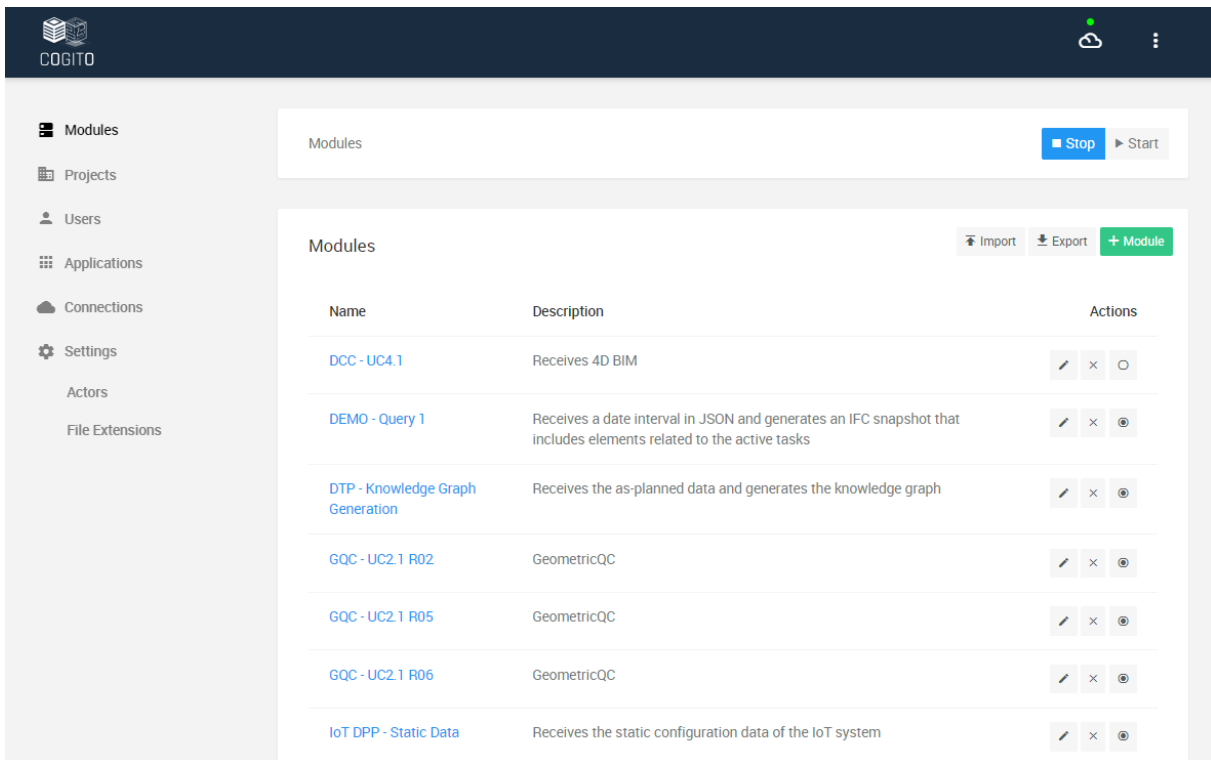
5.1.4.3 Deployment of COGITO Modules

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 and changing the status of 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
DCC - UC4.1	Receives 4D BIM	[Edit] [Delete] [Refresh]
DEMO - Query 1	Receives a date interval in JSON and generates an IFC snapshot that includes elements related to the active tasks	[Edit] [Delete] [Refresh]
DEMO - Test	Demo	[Edit] [Delete] [Refresh]
DTP - Knowledge Graph Generation	Receives the as-planned data and generates the knowledge graph	[Edit] [Delete] [Refresh]
GQC - UC2.1 R02	GeometricQC	[Edit] [Delete] [Refresh]

Figure 33 View of all created modules

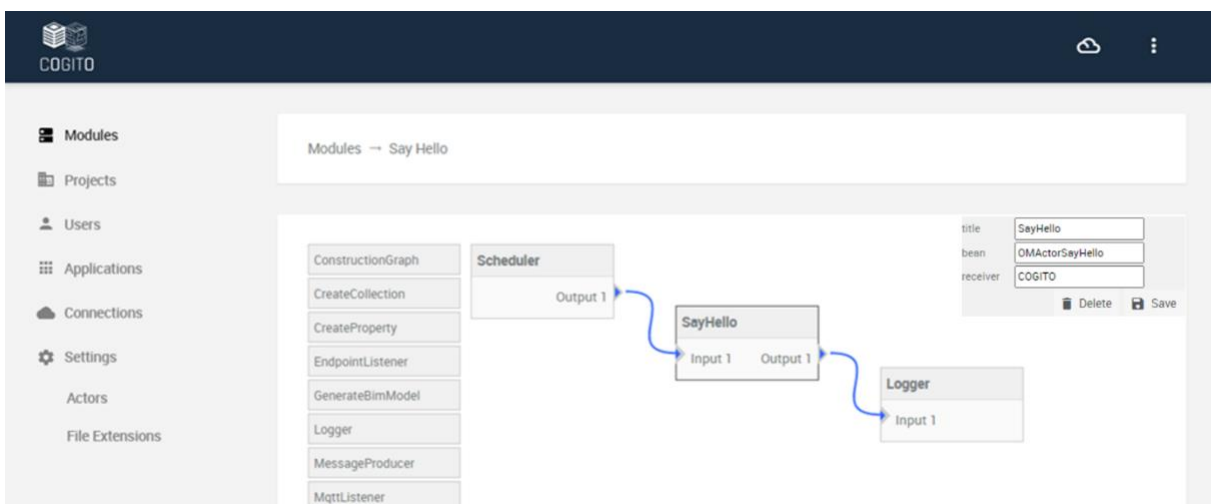
On the same page, users can create a new module by clicking the “+ Module” button. As shown in Figure 34, a modal window appears, which allows users to provide a module name and a short description. At any time, users can cancel the module creation process by clicking on the Close button.



Name	Description	Actions
DCC - UC4.1	Receives 4D BIM	[Edit] [Close] [Refresh]
DEMO - Query 1	Receives a date interval in JSON and generates an IFC snapshot that includes elements related to the active tasks	[Edit] [Close] [Refresh]
DTP - Knowledge Graph Generation	Receives the as-planned data and generates the knowledge graph	[Edit] [Close] [Refresh]
GQC - UC2.1 R02	GeometricQC	[Edit] [Close] [Refresh]
GQC - UC2.1 R05	GeometricQC	[Edit] [Close] [Refresh]
GQC - UC2.1 R06	GeometricQC	[Edit] [Close] [Refresh]
IoT DPP - Static Data	Receives the static configuration data of the IoT system	[Edit] [Close] [Refresh]

Figure 34 Creation of a new module

By clicking on the “Say Hello” 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 different module instances.



Modules → Say Hello

Available Actors:

- ConstructionGraph
- CreateCollection
- CreateProperty
- EndpointListener
- GenerateBimModel
- Logger
- MessageProducer
- MqttListener

Flow Diagram:

```

graph LR
    Scheduler[Scheduler] -- Output 1 --> SayHello[SayHello]
    SayHello -- Output 1 --> Logger[Logger]
  
```

Properties Modal for SayHello:

- title: SayHello
- bean: OMActorSayHello
- receiver: COGITO

Figure 35 Workspace for creating and configuring 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.i.de.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.de.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.i.de.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.

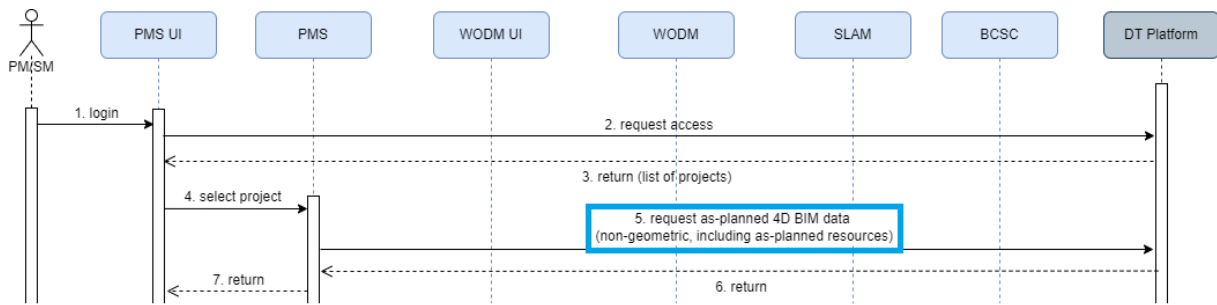


Figure 37 Part of the UC1.1 sequence diagram

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.

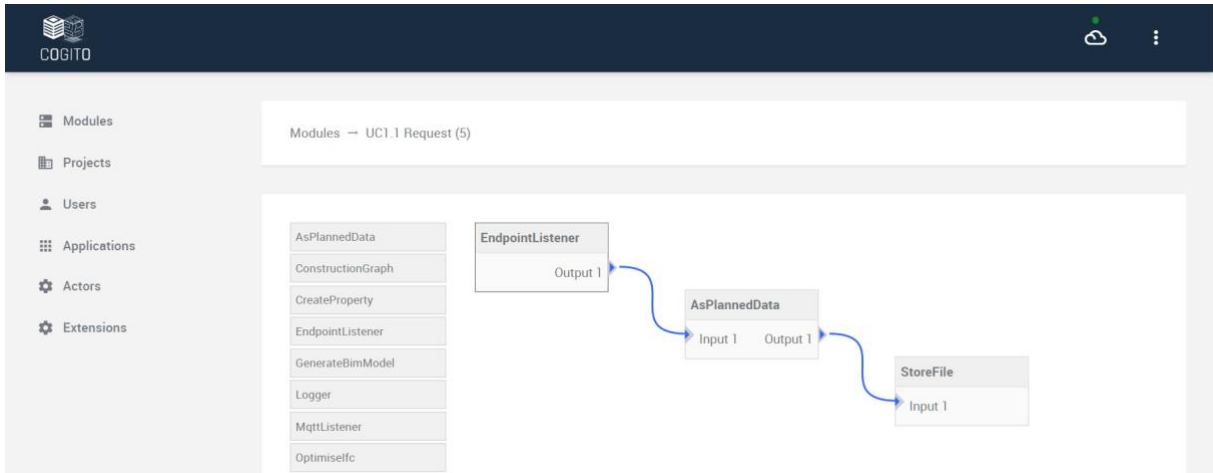


Figure 38 The module UC.1.1 – Return 6 which provides the as-planned 4D BIM data

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": "05ed3739-2c37-432c-9bfc-ffa6b8e13653",
  "elements": {
    "2$mt2h0xL62099U1LRjHNS": {
      "zone": "1H47rA2LHBvQsib8uffHd2",
      "name": "Safety guardrail",
      "type": "SAFETY_ELEMENT"
    },
    "0hrzoQIQnBZgkuIidDvGka": {
      "zone": "0ci7qHdzj6WUWEi_RJwQjv",
      "name": "Safety guardrail",
      "type": "SAFETY_ELEMENT"
    },
    ...
  },
  "resources": {
    "11": {
      "cost_per_hour": 300,
      "quantity": 1,
      "name": "Site supervisor",
      "type": "Human"
    },
    "12": {
      "cost_per_hour": 200,
      "quantity": 1,
      "name": "Foreman",
      "type": "Human"
    },
    ...
  },
  "project_name": "School_4",
  "zones": {
    "3Bw$FzFQfA2hHdyNNhc3y": {
      "linked_zones": [
        "1uCOqCAFj0zRwKxpovlHug"
      ],
      "name": "Fall_Hazard_Space",
      "type": "SAFETY_ZONE"
    },
    "1Qquu8oEfEQORgjpTHkU6e": {
      "linked_zones": [
        "1uCOqCAFj0zRwKxpovlHwK"
      ],
      "name": "Fall_Hazard_Space",
      "type": "SAFETY_ZONE"
    },
    ...
  },
  "tasks": {
    "5": {
      "end_date": "2009-11-20T17:00:00",
      "previous_task_list": [
        "4"
      ],
      "work_quantity_unit": "m2",
      "element_list": [
        {
          "action": "ADDED",
          "element_id": "1XY1A5V154-Qn$$bw1zKpQ"
        },
        {
          "action": "ADDED",
          "element_id": "0Yp1k128fErOfnriUHAM6S"
        }
      ]
    }
  }
}
```

```

        },
        ...
    ],
    "name": "Floor Slab (Floor 0)",
    "parent_task": "4",
    "type": "NORMAL_TASK",
    "start_date": "2009-11-09T08:00:00",
    "work_quantity": 200
}
}
}
}

```

5.1.6 Licensing

The DT Runtime is a closed-source component. On the other hand, the DT Library component which contains the actors defined based on the UCs of COGITO are open source. The COGITO developers can extend the functionalities of the DTP by creating, implementing, and deploying new actors and modules. 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 final 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 “D7.2 – Digital Twin Platform Design & Interface Specification v2”. During the integration activities in T8.1 the team in charge of the development will maintain the DT Library to ensure that the data processing operations implemented in the various actors meet the final requirements 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 is presented in Table 16. 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, on one hand, 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 16 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 final 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 DT Library contains actors supporting the requests of all defined UCs. These actors are used in the data-driven modules defined in the DT Runtime component. Based on the needs, additional modules can be added or updated during the integration phase.
- It handles real-time location tracking data and notifications coming from Messaging Layer in various messaging protocols. The integration of the DTP with the IoT Data Pre-Processing component will allow realistic testing of the KAFKA streaming technology, possibly requiring adaption or fine-tuning.

6 Conclusions

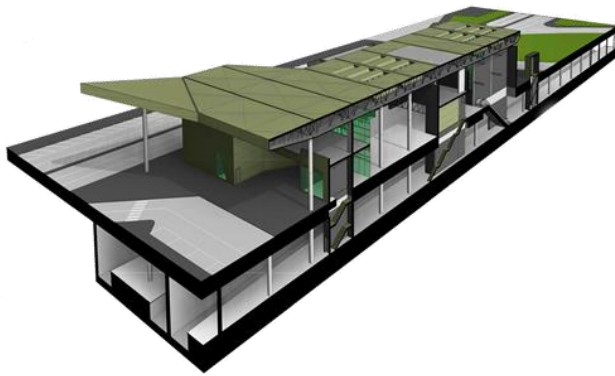
This demonstrator deliverable presented in detail the final release of COGITO's DT Platform. The DTP plays a central role in COGITO's system as it is responsible for i) providing an authentication and authorisation mechanism to COGITO users and applications, ii) handling input data from various external 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 final version of this deliverable, all 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 "D7.2 – Digital Twin Platform Design & Interface Specification v2". 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 final 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 and the DT Library contained in the Data Management Layer, responsible for creating and hosting various application-driven modules used in the various data processing operations.

The development of DTP is fully aligned with all defined end-user and functional/non-functional requirements as defined in "D2.1 - Stakeholder requirements for the COGITO system" and "D2.5 - COGITO System Architecture v2" respectively. The final version implements all the required functionality and features. The full integration between DTP and the rest of the COGITO tools is expected to take place as part of the "T8.1 - End-to-end ICT System Integration, Testing and Refinement" where any required adaption will be addressed.

References

- [1] COGITO, "D7.2 - COGITO Digital Twin Platform v2," 2022.
- [2] ISO 16739, "Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries (ISO 16739:2013)," CEN, 2016.
- [3] COGITO, "D2.5 - COGITO System Architecture v2," 2022.
- [4] COGITO, "D2.1 - Stakeholder requirements for the COGITO system," 2021.
- [5] COGITO, "D3.3 - COGITO Data Model & Ontology Definition and Interoperability Design v2," 2022.



COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu



THE UNIVERSITY
of EDINBURGH



POLITÉCNICA



ferrovial
construction



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