H2020 – ICT-13-2018-2019

MUSKETEER

Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)

Grant No 824988

D7.4 Final prototype of the MUSKETEER client connectors

September 21

Imprint

Contractual Date of Delivery to the EC:	**30 September 2021**
Author(s):	**Davide Profeta (ENG), Susanna Bonura (ENG)**
Participant(s):	**ENG, IBM, IDSA**
Reviewer(s):	**Stefano Braghin (IBM), Stephanie Rossello (KUL)**
Project:	**Machine learning to augment shared knowledge in federated privacy-preserving scenarios (MUSKETEER)**
Work package:	**WP7**
Dissemination level:	**Public**
Version:	**1.0**
Contact:	**Susanna Bonura – susanna.bonura@eng.it**
Website:	**www.MUSKETEER.eu**

Legal disclaimer

The project Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER) has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 824988. The sole responsibility for the content of this publication lies with the authors.

Copyright

Executive Summary

The MUSKETEER Client Connector is the component required by a participant to join the MUSKETEER Platform. That software application supports the MUSKETEER platform participants in exchanging the machine learning models and at the same time it prevents the sharing of private data in line with the data sovereignty principles.

This document provides a report that describes the architecture, instructions to install, configure and use the MUSKETEER Client Connector so to interact with the MUSKETEER Cloud Platform and describes the application programming interfaces developed and tests executed on such component.

The source code of the first prototype version of the MUSKETEER Client Connector is available at the following URLs released as open source under GNU AGPLv3 license:

- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-backend, for the back-end component and
- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-frontend, for the front-end component dedicated to the MUSKETEER project.

Document History

Version	Date	Status	Author	Comment
0.1	06 August 2021	Table of Contents	Susanna Bonura	First Draft
0.2	27 August 2021	First round of contribution to Sections	Davide Profeta	Update
0.3	31 August 2021	Second round of contribution to Sections	Davide Profeta	Update
0.4	10 September 2021	Third round of contribution to Sections	Davide Profeta	Update
0.5	12 September 2021	For internal review	Susanna Bonura	Draft for review
0.6	13 September 2021	Review inputs	Stefano Braghin	Update
0.7	15 September 2021	Review inputs	Stephanie Rossello	Update
0.8	20 September 2021	Updated version addressing comments received during the internal review process	Davide Profeta	Update
0.9	21 September 2021	Finalization	Susanna Bonura	Update
1.0	30 September 2021	Clean and submission	Gal Weiss	Final

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Abbreviation	Definition
API	Application Programming Interface
CA	Consortium Agreement
CC	Client Connector
DP	Differential Privacy
DC	Data Connector
DV	Data Value
FS	Feature Selection
FSM	Finite State Machine
GA	Grant Agreement
IDR	Intermediate Data Representation
IDS	Industrial Data Space
LC	Logistic Classifier
LGFS	Linear Greedy Feature Selection
MK	Master Key
ML	Machine Learning
MLP	Multi-Layer Perceptron
MN	Master Node
OS	Operating System
PERT	Program evaluation and review technique
PK	Public Key
POM	Privacy Operation Mode
PP	Privacy Preserving
PPML	Privacy Preserving Machine Learning
RAM	Reference Architecture Model
ROC	Receiver Operating Characteristics
SQL	Structured Query Language
TA	Task Alignment
UI	User Interface
WN	Worker Node

1 Introduction

1.1 Purpose

The purpose of this document is to provide a report which describes the main user interactions with the final prototype of the MUSKETEER Client Connector (D7.4. – Final prototype of the MUSKETEER client connectors).

The client connector is the component required by a participant to join the MUSKETEER Platform. It is the software application supporting MUSKETEER platform participants in the federated ML model exchange, share and process, so to guarantee the data sovereignty principles.

The client-side connectors have to support the set of privacy operation modes made available throughout the project according to the architecture defined in T3.1 and meet the requirements of the federated and privacy-preserving machine learning services designed in WP4 (for more details we refer to D4.1).

Moreover, the client component provides services for locally combining model updates into one consistent, up-to-date model instance. The client component serves as adapter for the integration and industrial validation of the MUSKETEER platform in WP7.

This version of the MUSKETEER Client Connector prototype together with this report are the final results of the task *T7.2 - Development of client connectors for industrial scenarios*, which aims to assemble and provide the privacy and security machine learning services developed in WP4 and WP5 and providing the functionalities (in their final version) to communicate with MUSKETEER Federated Machine Learning platform server designed and developed in WP3.

This report provides instructions to install, configure and use the MUSKETEER Client Connector so to interact with the MUSKETEER Cloud Platform and describes the application programming interfaces developed and tests executed on such component.

1.2 Related Documents

As already mentioned, the deliverable D7.4. – Final prototype of the MUSKETEER Client Connector, is the second and final one of the task *T7.2 - Development of client connectors for industrial scenarios*.

For the development of the MUSKETEER Client Connector presented in this document, several deliverables were considered as input (see Figure 1) both directly and indirectly linked to the WP7.

The input deliverables are:

D2.1 - Industrial and technical requirements.

D3.2 - Architecture design – Final version.

D3.4 - Final prototype of the MUSKETEER platform

D4.3 - Pre-processing, normalization, data alignment and data value estimation algorithms – Final version.

D4.5 - Machine learning algorithms over federated operation modes - final version.

D4.7 - Machine learning algorithms over semi honest operation modes - final version.

D7.2 - Client connectors architecture design – Final version.

D6.2 - Scalability of machine learning algorithms over ever POMs.

D6.4 - Data value extraction and monetization strategies.

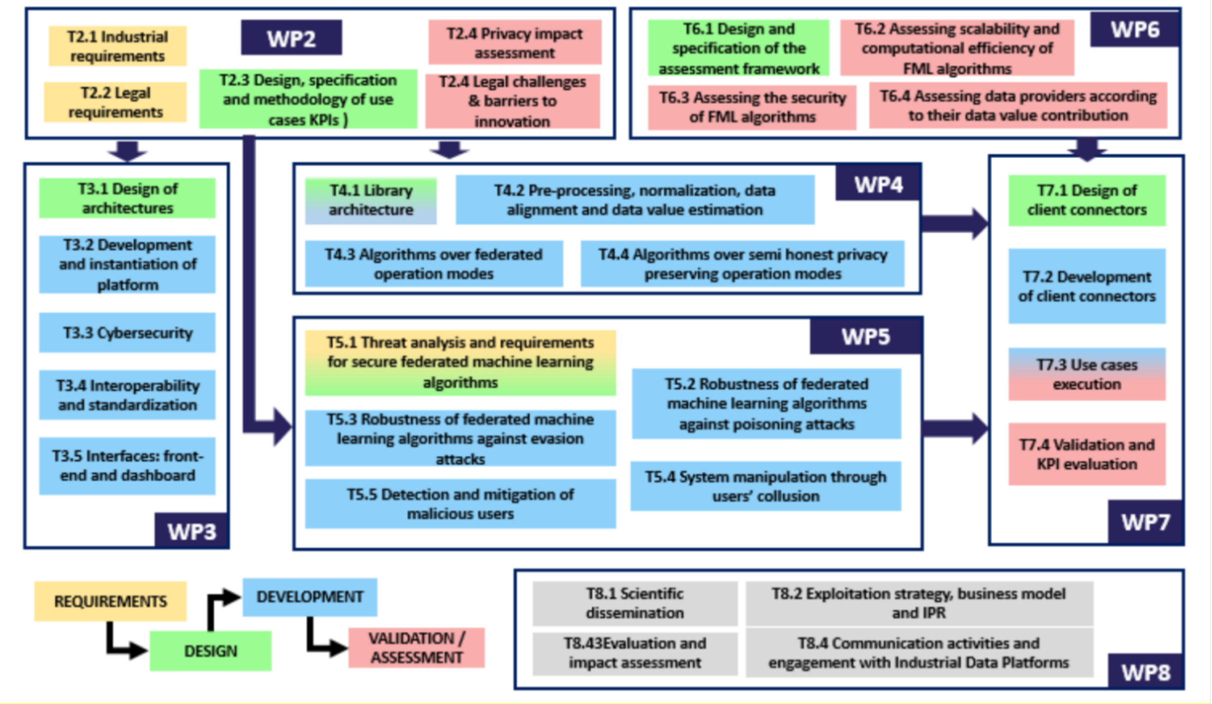

WP2
T2.1 Industrial requirements
T2.2 Legal requirements
T2.3 Design, specification and methodology of use cases KPIs)
T2.4 Privacy impact assessment
T2.4 Legal challenges & barriers to innovation
WP6
T6.1 Design and specification of the assessment framework
T6.2 Assessing scalability and computational efficiency of FML algorithms
T6.3 Assessing the security of FML algorithms
T6.4 Assessing data providers according to their data value contribution
WP3
T3.1 Design of architectures
T3.2 Development and instantiation of platform
T3.3 Cybersecurity
T3.4 Interoperability and standardization
T3.5 Interfaces: front-end and dashboard
WP4
T4.1 Library architecture
T4.2 Pre-processing, normalization, data alignment and data value estimation
T4.3 Algorithms over federated operation modes
T4.4 Algorithms over semi honest privacy preserving operation modes
WP5
T5.1 Threat analysis and requirements for secure federated machine learning algorithms
T5.2 Robustness of federated machine learning algorithms against poisoning attacks
T5.3 Robustness of federated machine learning algorithms against evasion attacks
T5.4 System manipulation through users' collusion
T5.5 Detection and mitigation of malicious users
WP7
T7.1 Design of client connectors
T7.2 Development of client connectors
T7.3 Use cases execution
T7.4 Validation and KPI evaluation
REQUIREMENTS
DESIGN
DEVELOPMENT
VALIDATION / ASSESSMENT
WP8
T8.1 Scientific dissemination
T8.2 Exploitation strategy, business model and IPR
T8.43Evaluation and impact assessment
T8.4 Communication activities and engagement with Industrial Data Platforms

Figure 1 - MUSKETEER's PERT diagram

1.3 Document Structure

In Section 2, the general description of the MUSKTEER Client Connector components is presented, together with the instructions to install and use the MUSKETEER Client Connector.

Section 3 presents the documentation of the Client Connector's APIs.

In Section 4, the results of the integration testing activities that were performed during the implementation phase are presented.

Finally, Section 5 concludes the deliverable. It outlines the main findings of the work done.

2 MUSKETEER Client Connector – Final Release

The final version of the Client Connect integrates new functionalities, provided by the updates obtained from the development of machine learning libraries and communications libraries that exchange data and information with the cloud. These features include those about the User Interface, like the new session that shows trained models and the deletion of their own tasks. The user can now also apply pre-processing algorithms when defining a task. The Data Connector component has been extended and improved in order to allow the user to link the Client Connector also with the PKL data.

A new implementation of the "*Abstract Communication Interface*" (D7.3 the first implementation) has been developed to allow the user to communicate between node and cloud, following IDSA guidelines [1]. Such new communication library is named HTTP CLOUD MESSENGER (HCM).[3]

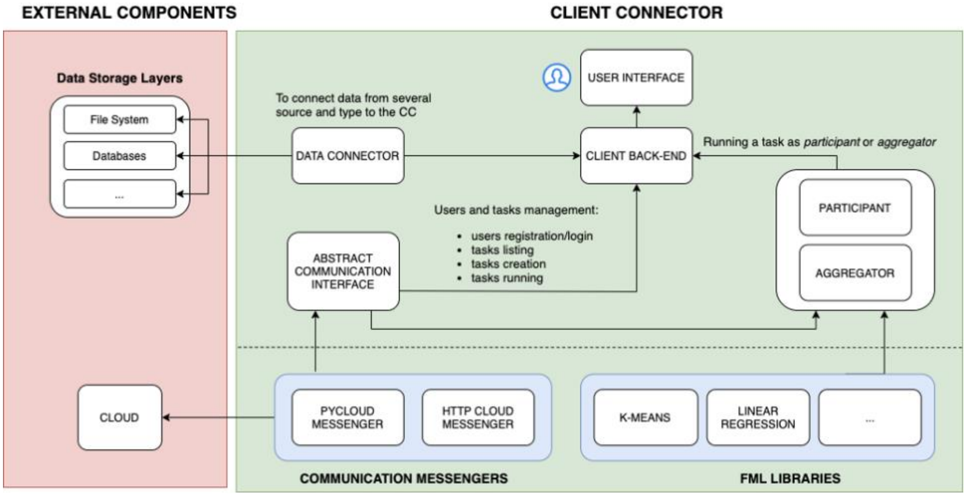


Figure 2 – MUSKETEER Client Connector Architecture

The HTTP CLOUD MESSENGER (HCM) allows a new kind of communication (HTTP protocol) by using an existing component developed by ENG, the TRUE (TRUsted Engineering) CONNECTOR (https://github.com/Engineering-Research-and-Development/true-connector). This ensures to the user a communication to cloud according to IDSA specification [3].

The HCM[3] connector aims to enable MUSKETEER (through the HCM) to be part of the IDS ecosystem providing the following main functionalities:

- Trusted data exchange through HTTP/HTTPS, web sockets and IDSCP.

- Full support of the IDS Information Model for metadata representation.
- Integration of the Access Control mechanisms supporting state-of-the-art IDS Identity Providers services.
- Integration of the Usage Control mechanisms.
- Registering transactions to the IDS Clearing House.
- Full compliance with the IDS broker for registering itself and querying.

The *Abstract Communication Interface* component allows to import and use an implementation of the communication library. In the MUSKETEER project there are two *Communication Messenger* libraries: the *pycloudmessenger* library and the *HttpCloudMessenger* library. Both available below:

pycloudmessenger: *https://github.com/IBM/pycloudmessenger*

HttpCloudMessenger: https://github.com/Engineering-Research-and-Development/musketeer-client-connector-backend/tree/master/httpcloudmessenger

After configuring and installing the communication messengers, the connector will be able to communicate to a cloud according the chosen implementation.

The HTTP CLOUD MESSENGER introduces the possibility to start a new communication according to IDSA specification [3] as shown in the flow below.

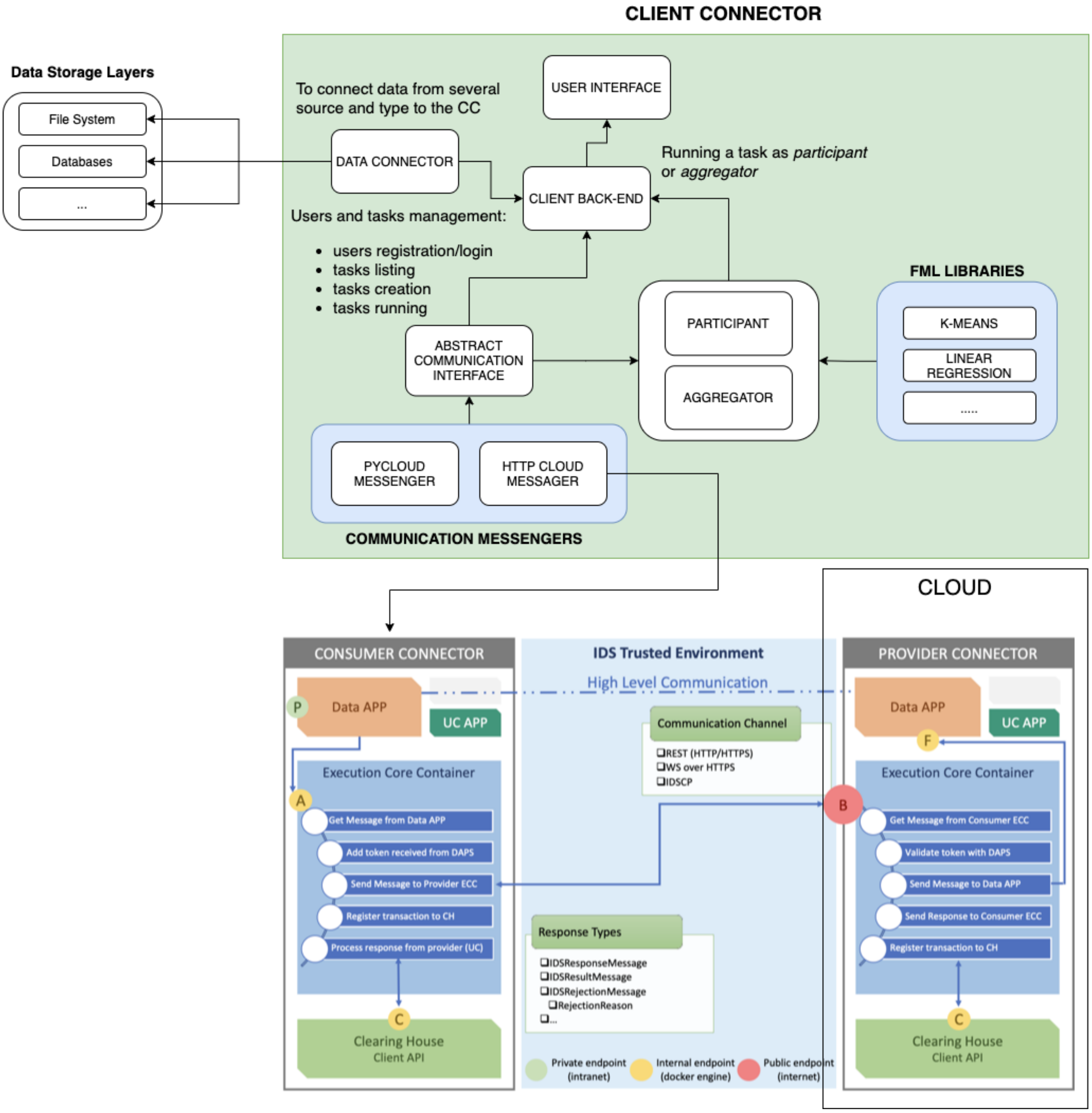


Figure 3 – MUSKETEER Client Connector: HTTP CLOUD communication

As we can see from the image above, the communication is not direct between HCM and the cloud, but it goes through the True Connector components. Moreover, the *Data App - P* into *Consumer Connector* receives the requests from HCM through REST API. Which makes the request compliant to True Connector message, sending it to the *Data App – F* on the cloud side. We can choose one of the protocols listed below to communicate to the cloud: REST API, Web Socket or IDSCP.

Currently a version is available that allows to show a complete flow of the message that start from CC and arrive to cloud responding to requests of IDSA [3]. Therefore, the *Data App – F* provides appropriate responses to the requests coming from CC, like the request for login or to get a tasks list.

The next Section will show all the final functionalities provided by Client Connector. It is useful also as User Guide for the Musketeer User who wants to interact with the platform through the user interface of the Client Connector. All the REST APIs provided by the CC back-end will follow in Section 3.

2.1 Installation guide

In the MUSKETEER project, federated ML is extended to support different Privacy Operation Modes (POMs), which control the amount and type of information that the data owners share during the model training and validation process. In POMs 1-3 (which closely follow conventional federated ML protocols), the model training is coordinated by a user initiator, called aggregator who creates and publishes a task, while the data owners act as participants by joining the task. Model training is typically performed iteratively throughout a number of rounds which is either determined a priori, or dynamically, e.g. by considering a model convergence criterion. In each round, the aggregator dispatches the current central version of the model to all the participants. Then the participants compute updates to that model based on their local data and send the updates back to the aggregator. Model updates can either be in the form of gradients, or in the form of new versions of the model. Upon having received the updates from all participants, the aggregator incorporates them (e.g. by taking an average of all the updates) into the new version of the central model. After the training rounds have completed, the aggregator holds the final version of the model, which can then be centrally stored for later use and/or deployed by the participants in their local production environments. The Client Connector supports and integrates the latest versions of the MUSKETEER communication messenger library and the MUSKETEER Machine Learning Library (MMLL); the user will then interact through a graphical interface exposed by the Client Connector application, designed to have a user experience as guided and simple as possible to interact with the internal and imported external components to create, execute and monitor tasks in a federated ecosystem according to their needs (e.g. choosing algorithms that apply a specific level of privacy) and on their own datasets, as well as obtain the trained models and get metrics to evaluate the final model through representative charts.

The following Sections describe the steps to install and run the Client Connector according the abovementioned approach.

2.1.1 Installation

As a requirement, it is necessary to have a Docker engine installed on the host machine to run the Client Connector application.

The source code of the last version of the Client Connector is available at the following URLs released as open source under GNU AGPLv3 license:

- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-backend, for the back-end component and
- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-frontend, for the front-end component dedicated to the MUSKETEER project.

Another implemented and released component as open source under the GNU AGPLv3 license, is the Back-End Data Application (BEDA). It's sub-component fork of the TRUE (TRUsted Engineering) Connector (https://github.com/Engineering-Research-and-Development/true-connector) for the IDS (International Data Space) ecosystem implemented by ENGINEERING; the BEDA is responsible for preparing and processing HTTP requests originating from the communication library configured and installed in the Client Connector, in order to:

1) Build a message that complies with the IDS standards.
2) Guarantee secure and reliable communication between a consumer connector and a provider connector through the BEDA component and the other two sub-components composing the TRUE Connector, namely: Execution Core Container (ECC), and Usage-Control (UC), already implemented by ENGINEERING and whose link is shown above.

The source code of BEDA, adapted to the MUSKETEER project, is available at the following link: https://github.com/musketeer-eng-team/true-connector-basic_data_app.

Regarding the Client Connector, enriched with the components of the TRUE Connector, and a communication library using HTTP invocations, a more detailed description is given in the final paragraph of this Section. From here on it is explained how to install and configure the Client Connector through its basic components in order to interact with the MUSKETEER platform.

As a first step create the Docker image of the backend components. From the project root folder, run the following command through the terminal:

- *docker build -f Dockerfile -t MYBUILDIMAGE*

The same *MYBUILDIMAGE* name chosen must be inserted in the *docker-compose.yml* file.

Before running the *docker-compose.yml* the user must also configure the Docker volumes for the backend component. In particular, it is necessary to specify:

- FS_PATH_DATA: a filesystem path directory where there are the datasets that you want to bind to the Client Connector.
- FS_PATH_RESULTS: a filesystem path directory where to store all the results file generated by the task you run and complete.

The *docker-compose.yml* contains both the backend image just created and the frontend component. The frontend Docker image is located on a repository accessible through authentication to our Docker registry. To log in, run the following command:

- *docker login gitlab.alidalab.it:5000/musketeer/ngx-musketeer-client*, followed by USER and PASSWORD that have been provided.

Finally, run the following command to run and up the Desktop Client Connector:

- *docker-compose pull && docker-compose up*

Another open source component that will be downloaded, is the Docker image "docker.io/bitnami/mongodb:4.4", a Mongo database already pre-configured to contain the information related to the metadata of the datasets uploaded by the user through the Client Connector. A NoSQL database was chosen in order to perform CRUD operations: create, read, update, and delete documents; moreover, thanks to the Mongo documents, in contrast to the SQL tables, we can insert new fields (new metadata information), if needed, to describe new types of connections to the data resulting from the extension of the sub-component Data Connector. The Data Connector (DC) is responsible to import and read by reference the user data to the Client Connector.

The frontend Docker image will be automatically pulled from the register, if it is not present. It may take some minutes to download all the required dependencies based on your internet connection. Once it is done, the local server will be running at '127.0.0.1:5000', whilst you can use the User Interface by opening a browser and writing the following URL: '127.0.0.1:4500' (or 'localhost:4500').

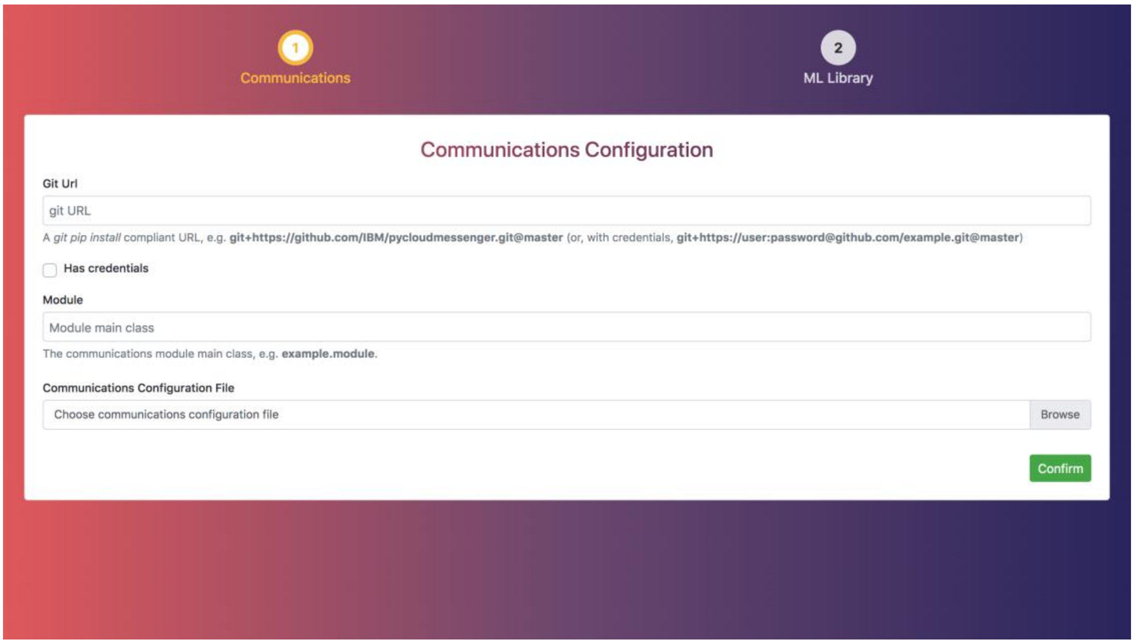
2.1.2 Configuration

This Section describes the configuration steps once the Desktop Client Connector has been started for the first time. These steps consist in the installation and configuration of the two external components presented in the Client Connector architecture: the Communication Messenger and Federated Machine Learning Python-based library.

Once you open the page on localhost:4500 from your browser for the first time, you will be redirected to localhost:4500/configure, where it is possible to configure and install the first Communication Messenger component as shown in Figure 4 below. As shown in the Figure, the required information is the following:

- Git Url: a Git URL where the communication library is hosted.
- Module: the communication module main class, in the form *package.module*.

- Communications Configuration File: a Json file containing all the information needed by the communication messenger library to connect towards the core server.



Communications Configuration

Git Url

git URL

A *git pip install* compliant URL, e.g. **git+https://github.com/IBM/pycloudmessenger.git@master** (or, with credentials, **git+https://user:password@github.com/example.git@master**)

□ Has credentials

Module

Module main class

The communications module main class, e.g. **example.module**.

Communications Configuration File

Choose communications configuration file

Figure 4 - Communication Configuration step

In the MUSKETEER project, the communication messenger used is the *pycloudmessenger* library developed by IBM and available at the following GIT repository: https://github.com/IBM/pycloudmessenger. For this instance, the settings used are the following:

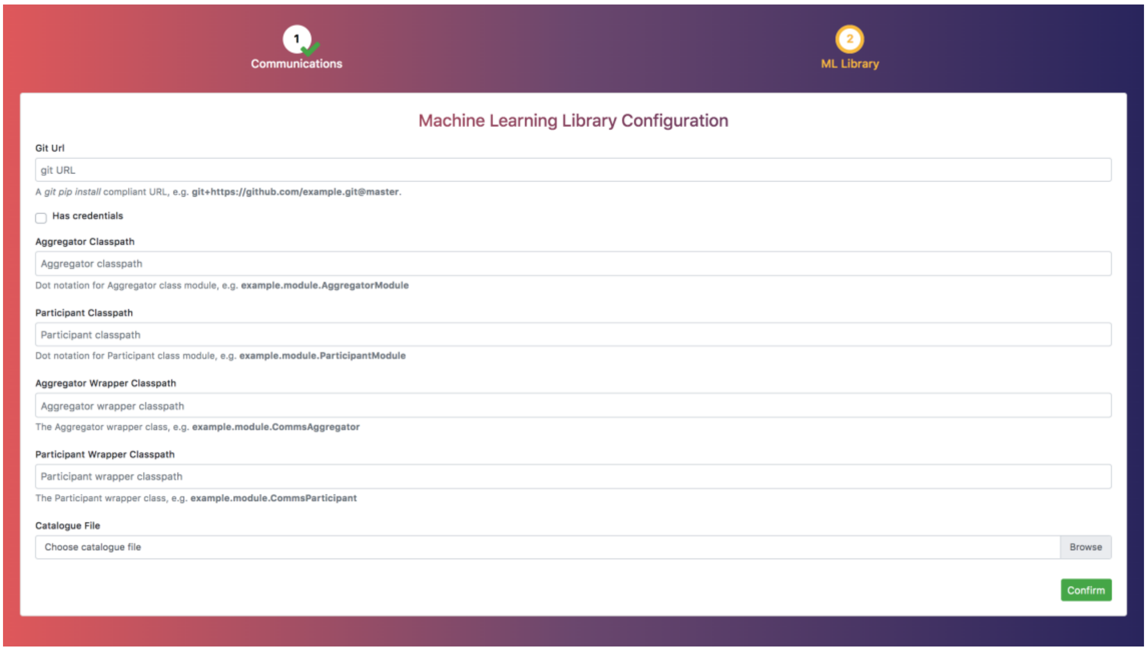
- Git Url: *git+https://github.com/IBM/pycloudmessenger.git@master*
- Module: *pycloudmessenger.ffl.fflapi*
- Communications Configuration File: the Json file provided by IBM.

Once you have entered this information you can confirm clicking on the related button and install the library. If the installation is successful you will proceed to the next step. If something has gone wrong you will be notified with an error message.

The next step, as shown in Figure 4 below, concerns the configuration and installation of the machine learning library. For the Machine Learning library configuration, the required information is:

- Git Url: a Git URL where the machine learning library is hosted.
- Aggregator Classpath: the aggregator class module where are present the main classes to instantiate the objects of the machine learning algorithms related to the role of aggregator.

- Participant Classpath: the participant class module where are present the main classes to instantiate the objects of the machine learning algorithms related to the role of participant.



Machine Learning Library Configuration

Git Url

git URL

A *git pip install* compliant URL, e.g. **git+https://github.com/example.git@master**.

□ Has credentials

Aggregator Classpath

Aggregator classpath

Dot notation for Aggregator class module, e.g. **example.module.AggregatorModule**

Participant Classpath

Participant classpath

Dot notation for Participant class module, e.g. **example.module.ParticipantModule**

Aggregator Wrapper Classpath

Aggregator wrapper classpath

The Aggregator wrapper class, e.g. **example.module.CommsAggregator**

Participant Wrapper Classpath

Participant wrapper classpath

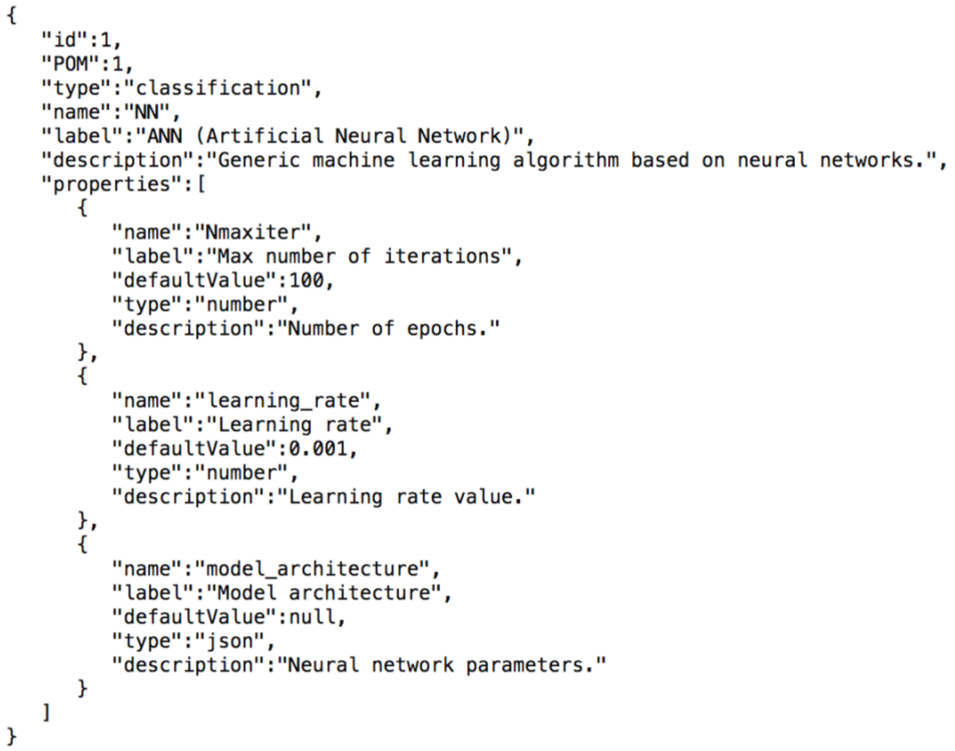
The Participant wrapper class, e.g. **example.module.CommsParticipant**

Catalogue File

Choose catalogue file

Figure 5 - Machine Learning Library Configuration step

- Aggregator Wrapper Classpath: the aggregator wrapper class module used to wrap the communication messenger library related to the role of aggregator.
- Participant Wrapper Classpath: the participant wrapper class module used to wrap the communication messenger library related to the role of participant.
- Catalogue File: it is a JSON file containing the meta-model of the algorithms that are available in the machine learning library imported. In Figure 6 is shown a meta-model example of a single algorithm, related to an Artificial Neural Network algorithm.



```
{
   "id":1,
   "POM":1,
   "type":"classification",
   "name":"NN",
   "label":"ANN (Artificial Neural Network)",
   "description":"Generic machine learning algorithm based on neural networks.",
   "properties":[
      {
         "name":"Nmaxiter",
         "label":"Max number of iterations",
         "defaultValue":100,
         "type":"number",
         "description":"Number of epochs."
      },
      {
         "name":"learning_rate",
         "label":"Learning rate",
         "defaultValue":0.001,
         "type":"number",
         "description":"Learning rate value."
      },
      {
         "name":"model_architecture",
         "label":"Model architecture",
         "defaultValue":null,
         "type":"json",
         "description":"Neural network parameters."
      }
   ]
}
```

Figure 6 – Meta-model algorithm example

This catalogue file defines the available algorithms (specifying the POMs because not all the algorithms can be implemented for all the POMs) collecting the meta-models and all the required information. This is useful in the creation task step of the User Interface, where you choose the algorithm. In fact, it allows you to select among the algorithms defined in this file.

As shown in the Json example of the meta-model, a set of algorithm information is described including: the type of algorithm (between clustering, regression, classification), for which POM it is appointed and a description of the algorithm parameters that can then be valorised by the user during the task creation.

In the MUSKETEER project, the Machine Learning Library are developed by UC3M and TREE and it is available at the following GIT private repository: https://github.com/Musketeer-H2020/MMLL. The information to set at this last configuration step are the following:

- Git URL: *git+https://github.com/Musketeer-H2020/MMLL.git*

  It's needed to set also the GitHub personal access token (PAT) to download the MML library.

- Aggregator Classpath: *MMLL.nodes.MasterNode.MasterNode*
- Participant Classpath: *MMLL.nodes.WorkerNode.WorkerNode*
- Aggregator Wrapper Classpath: *MMLL.comms.comms_pycloudmessenger.Comms_master*

- Participant Wrapper Classpath: *MMLL.comms.comms_pycloudmessenger.Comms_worker*
- Catalogue file: a JSON file containing the list of algorithms metadata as described above in Figure 6.

This information will make it possible to correctly integrate the MUSKETEER Machine Learning library and execute on-demand the tasks created by the user, i.e. execute a particular algorithm according to a specific POM chosen and contained in the installed and configured library.

As for the previous step, once you have filled all the information, confirm for the machine learning library installation. Properly installed also this component, you will be redirected to the login/registration page.

2.2 User registration and login

Once you have configured the Client Connector you will be redirected to the login page, as shown in Figure 6 below.

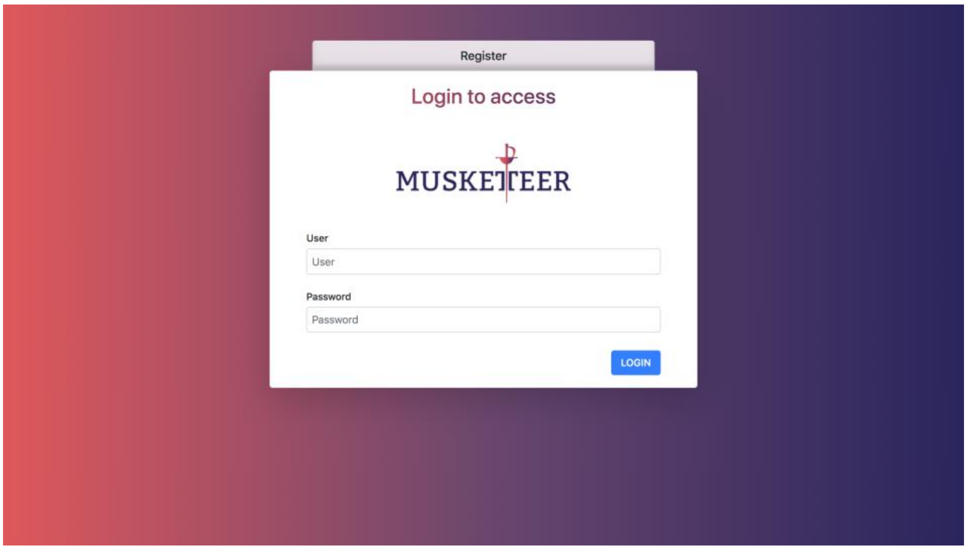
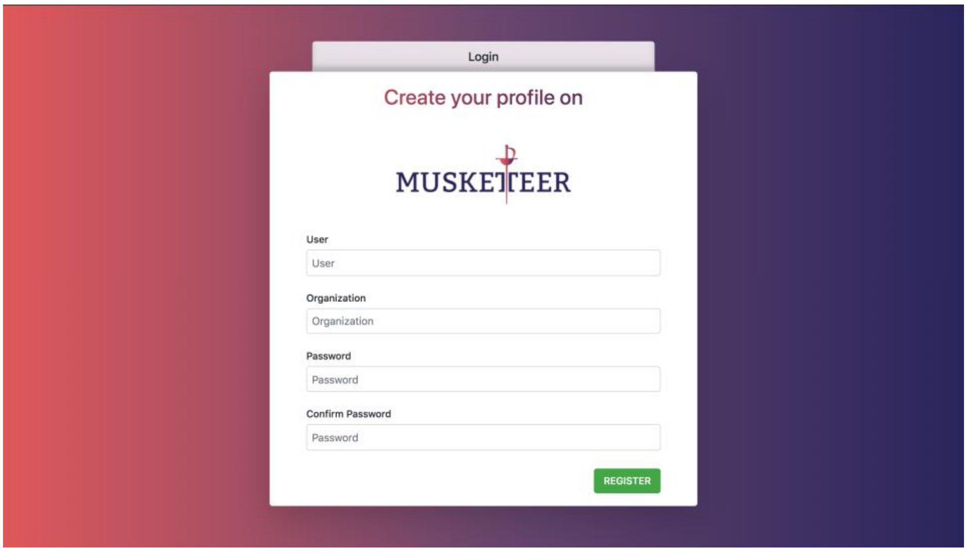


Figure 6 - User login page

If the user is already registered to the target platform, the MUSKETEER platform, it is possible to access with their own credentials. Otherwise, the user can click on the window behind the login window to register a new user. To register a new user, it is necessary to insert the following information, as shown in the Figure 7 below:

- Username.
- Organization name.
- Password.
- Confirm of the password.


Login

Create your profile on

MUSKETEER

User
User

Organization
Organization

Password
Password

Confirm Password
Password

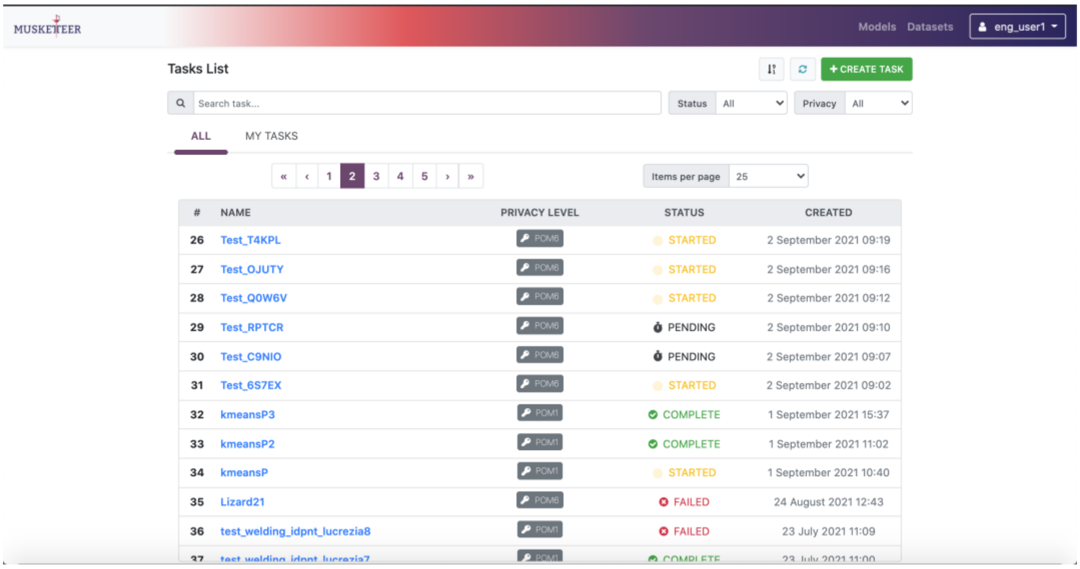
Figure 7 - User registration page

Enter the login credentials and you authenticate to the target MUSKETEER server accessing to the main page. Under the hood is used the Communication Messenger library and the configuration parameters that have been described in Section 4.

2.3 Tasks listing and browsing

The main page of the Client Connector is located on http/localhost:4500/tasks. The user, on this page, can view and browse the tasks that are stored through the Musketeer platform.

The main page lookout is shown in the following Figure 8.



Tasks List

Search task...

Status All

Privacy All

Items per page 25

#	NAME	PRIVACY LEVEL	STATUS	CREATED
26	Test_T4KPL	POM6	STARTED	2 September 2021 09:19
27	Test_OJUTY	POM6	STARTED	2 September 2021 09:16
28	Test_Q0W6V	POM6	STARTED	2 September 2021 09:12
29	Test_RPTCR	POM6	PENDING	2 September 2021 09:10
30	Test_C9NIO	POM6	PENDING	2 September 2021 09:07
31	Test_6S7EX	POM6	STARTED	2 September 2021 09:02
32	kmeansP3	POM1	COMPLETE	1 September 2021 15:37
33	kmeansP2	POM1	COMPLETE	1 September 2021 11:02
34	kmeansP	POM1	STARTED	1 September 2021 10:40
35	Lizard21	POM6	FAILED	24 August 2021 12:43
36	test_welding_idpnt_lucrezia8	POM1	FAILED	23 July 2021 11:09
37	test_welding_idpnt_lucrezia7	POM1	COMPLETE	23 July 2021 11:00

Figure 8 - Main page

As shown in the Figure, tasks can be filtered in several ways:

- by name.
- by status: created, pending, started, completed, failed.

- by privacy operation mode (POM).

In addition, a client-side task pagination has been added in order to more easily browsing the tasks and lighten the whole page.

Each task, at a high level, is represented by a name, the level of privacy adopted, its current status, and its creation date. By clicking on the name of a task, it is possible to access the details of that specific task, where it is also possible to participate or aggregate to the task through a series of actions that are described in the following paragraphs.

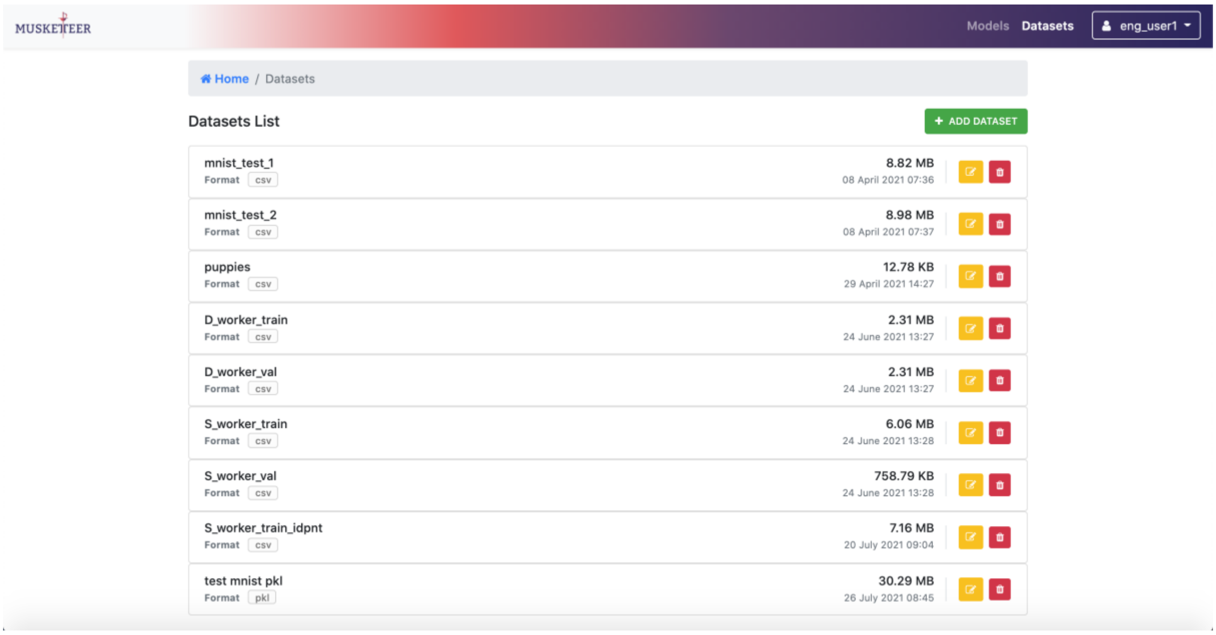
From the main page it is then possible to access the other following Sections of the Client Connector by clicking on the buttons in the top bar:

- Task creation (http://localhost:4500/tasks/create): by clicking on the green button "create task" on the top right.
- Models (http://localhost:4500/models): display the models trained and stored in the Musketeer platform as a result of completed tasks.
- Datasets (http://localhost:4500/datasets): to access the area where you can manage your metadata datasets that you want to connect to the Client Connector, and that will be processed by the machine learning algorithms during a task execution.
- The user's area from where it is possible to change or update the configuration of the installed libraries; change the user's password, remove the user's account, or log-out of the platform.

2.4 Data connection

The Datasets page is accessible at the following URL: http://localhost:4500/datasets. The user can connect dataset to the Client Connector through the provision of basic information (metadata) to access a dataset from a particular data storage and format. The Client Connector supports the binding of dataset in CSV and PKL format from the user's File System to the Client Connector, being able to specify whether or not the dataset has a header in the case of CSV format, and if it contains label information, in the case of PKL format data.

The following Figure 9 shows the data connection page. The list of all datasets connected by the user in the Client Connector is shown here. For each dataset the following information are shown: dataset label, format, size (in MB) and date of insertion. In addition, on the right of each dataset, there is a button for editing the dataset, e.g. if you want to change its label; and one for removing the metadata of the dataset. It is not the dataset that is physically removed but only the references to connect to it via the Data Connector component.



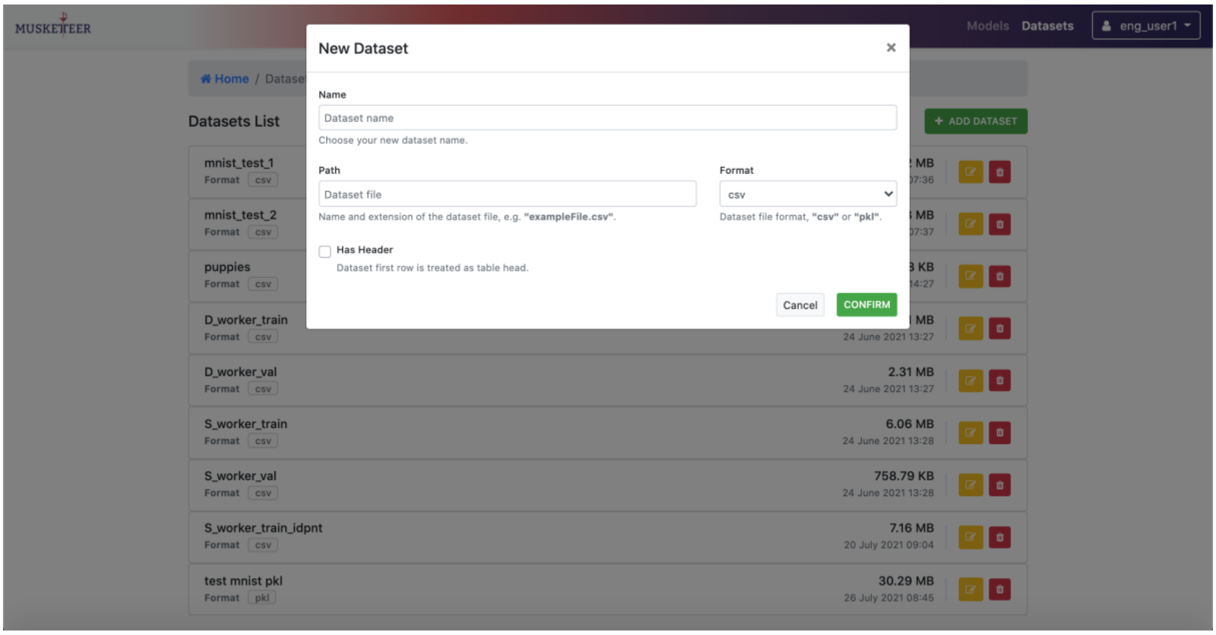
Datasets List

mnist_test_1	Format csv	8.82 MB	08 April 2021 07:36
mnist_test_2	Format csv	8.98 MB	08 April 2021 07:37
puppies	Format csv	12.78 KB	29 April 2021 14:27
D_worker_train	Format csv	2.31 MB	24 June 2021 13:27
D_worker_val	Format csv	2.31 MB	24 June 2021 13:27
S_worker_train	Format csv	6.06 MB	24 June 2021 13:28
S_worker_val	Format csv	758.79 KB	24 June 2021 13:28
S_worker_train_idpnt	Format csv	7.16 MB	20 July 2021 09:04
test mnist pkl	Format pkl	30.29 MB	26 July 2021 08:45

Figure 9 - Data connection page – datasets list

On the top side, clicking on the "Add Dataset" button opens a modal, as shown in Figure 10. This modal requests a set of basic information: name (as label) of the dataset, the file dataset name that is inside the dataset folder set by the user during the Client Connector installation and configuration (see Section 2.1), and others information that may be specific to the dataset format being imported, such as the header in the case of data in CSV format.

By clicking on the "Confirm" button the new dataset will be added in the list of datasets; if something went wrong an error message will be displayed.



New Dataset ×

Name
Dataset name
Choose your new dataset name.

Path
Dataset file
Name and extension of the dataset file, e.g. "exampleFile.csv".

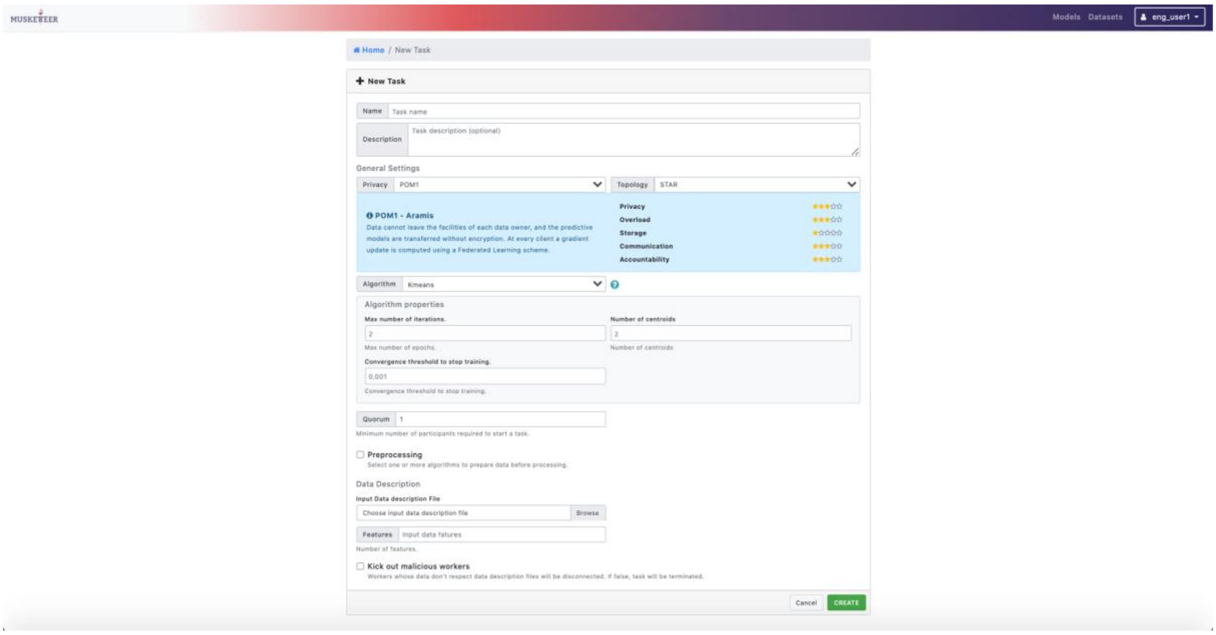
Format
csv
Dataset file format, "csv" or "pkl".

□ Has Header
Dataset first row is treated as table head.

Figure 10 - Data connection page – add new dataset

2.5 Tasks creation

From the main page you can access the tasks creation page (located on "http://localhost:4500/tasks/create") by clicking on the "Create task" button. Figure 10 below shows the details of all the information that can be set while creating a new task by the user.



Name: Task name

Description: Task description (optional)

General Settings

Privacy: POM1

Topology: STAR

POM1 - Aramis
Data cannot leave the facilities of each data owner, and the predictive models are transferred without encryption. At every client a gradient update is computed using a Federated Learning scheme.

Privacy	★★★☆☆
Overload	★★★☆☆
Storage	★☆☆☆☆
Communication	★★★☆☆
Accountability	★★★☆☆

Algorithm: Kmeans

Algorithm properties

Max number of iterations.
2
Max number of epochs.

Number of centroids
2
Number of centroids

Convergence threshold to stop training.
0.001
Convergence threshold to stop training.

Quorum: 1
Minimum number of participants required to start a task.

☐ Preprocessing
Select one or more algorithms to prepare data before processing.

Data Description

Input Data description File
Choose input data description file

Features: Input data fatures
Number of features.

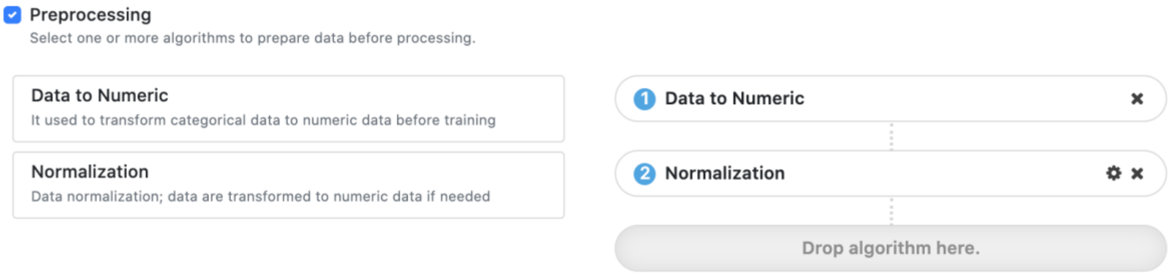
☐ Kick out malicious workers
Workers whose data don't respect data description files will be disconnected. If false, task will be terminated.

Figure 10 - Task creation page detail

As shown in the previous Figure, starting from the top, the user can set the following information:

- Name (required): a task name.
- Description (optional): a task description.
- Privacy (required): a select box to choose the level of privacy (POM) the user wants to apply. Each POM is described by a description and a set of characteristics in comparison with the other POMs.
- Topology (required): the topology of the task (RING or STAR),
- Algorithm (required): a select box to choose an algorithm, according to the POM selected, that the user wants to apply for its task. Once an algorithm has been selected, the properties of the algorithm that can be set by the user will be shown. The information on the algorithms and the parameters of each algorithm were loaded during the configuration phase of the Client Connector as explained in Section 2.1.2.
- Quorum (required): minimum number of participants required to start the task.

- Pre-processing (optional): the user can insert a set of pre-processing algorithms, included in the imported MMLL library, to be placed in the pipeline; the following Figure 11 shows the insertion, via drag-and-drop, of two pre-processing algorithms which, as explained in D4.3, transform the categorical data of the dataset into numerical data, and therefore perform a data normalisation before the data processing. By clicking on the gear in the pre-processing algorithm "normalization", it is possible to set the specific parameters of the pre-processing algorithm in question.



☑ Preprocessing
Select one or more algorithms to prepare data before processing.

Data to Numeric
It used to transform categorical data to numeric data before training

Normalization
Data normalization; data are transformed to numeric data if needed

Figure 11 - Task creation page – Pre-processing step

In order to apply the pre-processing algorithms, and the dataset verification checks, it is necessary to also insert a data description file as specified in the D4.3. document. Other information concerning the structure of the datasets are:

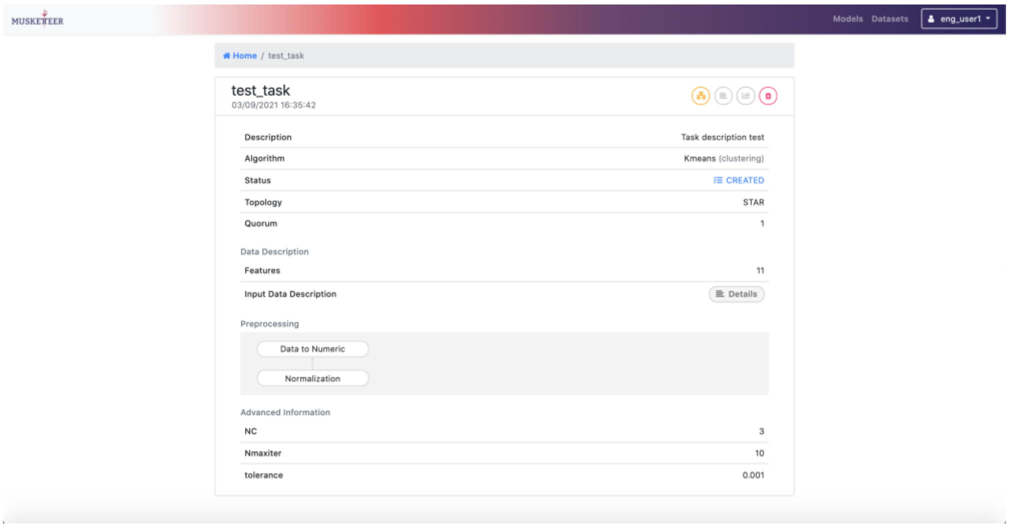
- Data description file (required): a file containing all the information describing the input dataset required for the execution of the task and therefore for the training of the resulting machine learning model.
- Features (required): number of the input dataset features.
- Labels (required): number of the input dataset labels.

Finally, the user can specify by checkbox, whether he wants to abort the entire task when a user participates with data that does not comply with those specified in the data description file, or to kick the individual user out, and continue the execution of the task with the remaining participants.

After filling in all the required information you can create the new task by clicking on the "Create" button. The new task just created will now be present in the list of tasks, where you can look again all the parameters that have been set during the task creation. As the creator of the task you can run it as an aggregator; waiting for new participants in the task until the chosen quorum is reached.

2.6 Tasks detail

From the home page of the Client Connector, as already discussed in Section 2.2, it is possible to view the complete list of tasks. Clicking on a particular task opens the task detail page; an example is shown in Figure 12 below.



test_task
03/09/2021 16:35:42

Description	Task description test
Algorithm	Kmeans (clustering)
Status	CREATED
Topology	STAR
Quorum	1

Data Description

Features	11
Input Data Description	Details

Preprocessing

Advanced Information

NC	3
Nmaxiter	10
tolerance	0.001

Figure 12 - Task detail

The task detail page, as shown in the Figure above, contains all the information that was put during the creation of the task; users can therefore consult the selected task in detail, and decide whether or not to participate on it. The detail page of a task contains basic information about the task, including the description, the type of algorithm, the status and the quorum of participants to be reached for the task to start; information about the structure of the dataset; the pre-processing algorithms, which are executed in the pipeline before the algorithm starts; and finally, information about the algorithm parameters chosen by the task creator.

In the top right-hand corner of the detail page, there are buttons to interact with the task. In the case shown above, since the user is the creator of the task, the following buttons are displayed, starting from the left:

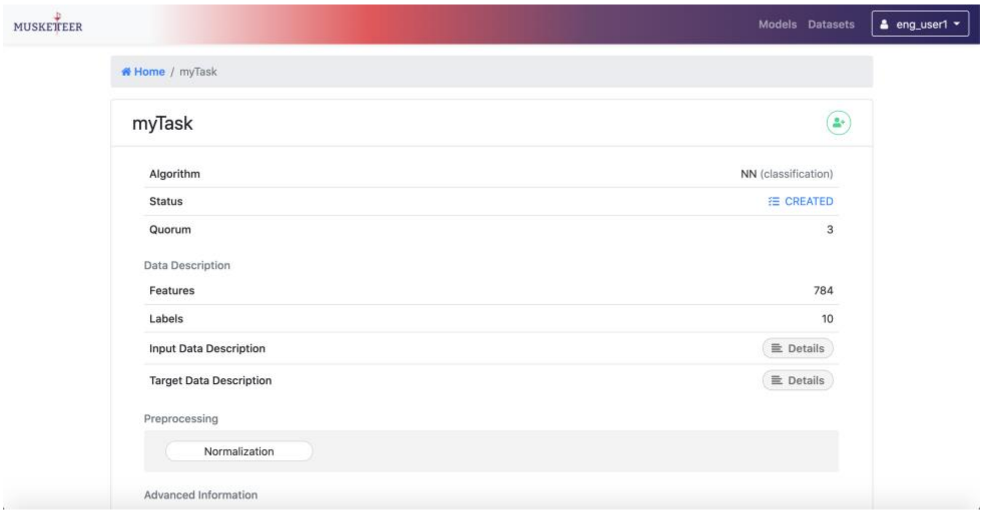
- the button to aggregate to the task, shown only if the logged-in user is the creator of the task, otherwise the button to participate a task is shown;
- the button to access your task logs; it’s not clickable if the task has not yet started;

- the button to display a result chart image, created at the end of task completion, showing metrics to evaluate the final model trained during the task execution;
- the button to delete the task, only by the task creator.

In the following paragraphs, the mentioned actions on tasks are described in more detail.

2.6.1 Execution

As discussed in the Section introduction, the task creator is the only one who can act as an aggregator. Another user, on the other hand, may participate in a task that has been created or is waiting to reach a quorum, as shown in Figure 13 below.



myTask

Algorithm	NN (classification)
Status	CREATED
Quorum	3

Data Description

Features	784
Labels	10
Input Data Description	Details
Target Data Description	Details

Preprocessing

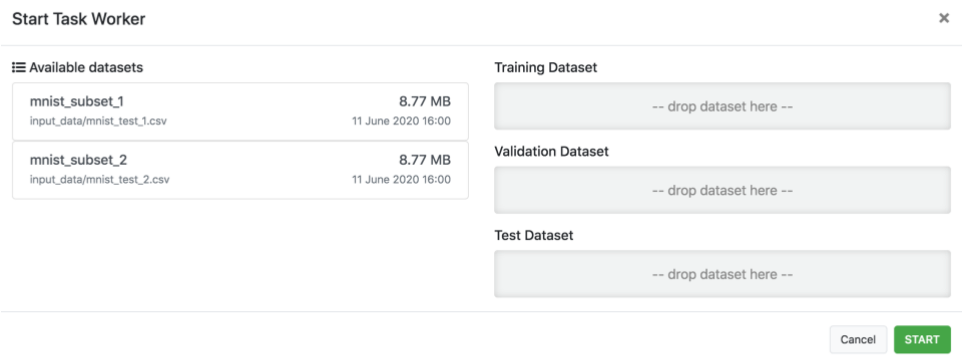
Normalization

Advanced Information

Figure 13 - Task detail – participant user

The aggregator collects the weights of the models received from each participant and aggregates them to obtain an aggregated machine learning model.

A participant can join and execute the task as participant by clicking on the green button as shown in the Figure above. As shown in the next Figure 14, it opens a modal where the user can drag-and-drop their dataset and start the task. Only the training dataset is required to execute a task as a participant.



Start Task Worker

Available datasets

mnist_subset_1 input_data/mnist_test_1.csv	8.77 MB 11 June 2020 16:00
mnist_subset_2 input_data/mnist_test_2.csv	8.77 MB 11 June 2020 16:00

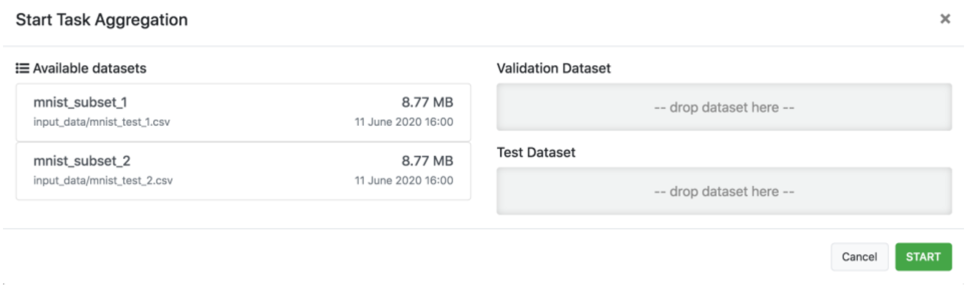
Training Dataset: -- drop dataset here --

Validation Dataset: -- drop dataset here --

Test Dataset: -- drop dataset here --

Figure 14 - Task worker modal

The following Figure 15 shows the task aggregator modal clicking on the aggregate button. For the aggregator the validation and test datasets are required. When at least one participant has joined a task, it switches to 'PENDING' status, waiting to reach the chosen quorum and then switching to 'STARTED' status. During or after starting a task as a participant or aggregator, the user can monitor its execution by reading the logs managed by the Client Connector; while, on test data, at the end of the task execution, a resulting chart will be generated depending on the algorithm type. These features will be discussed in more detail in the next paragraphs.



Start Task Aggregation

Available datasets

mnist_subset_1 input_data/mnist_test_1.csv	8.77 MB 11 June 2020 16:00
mnist_subset_2 input_data/mnist_test_2.csv	8.77 MB 11 June 2020 16:00

Validation Dataset: -- drop dataset here --

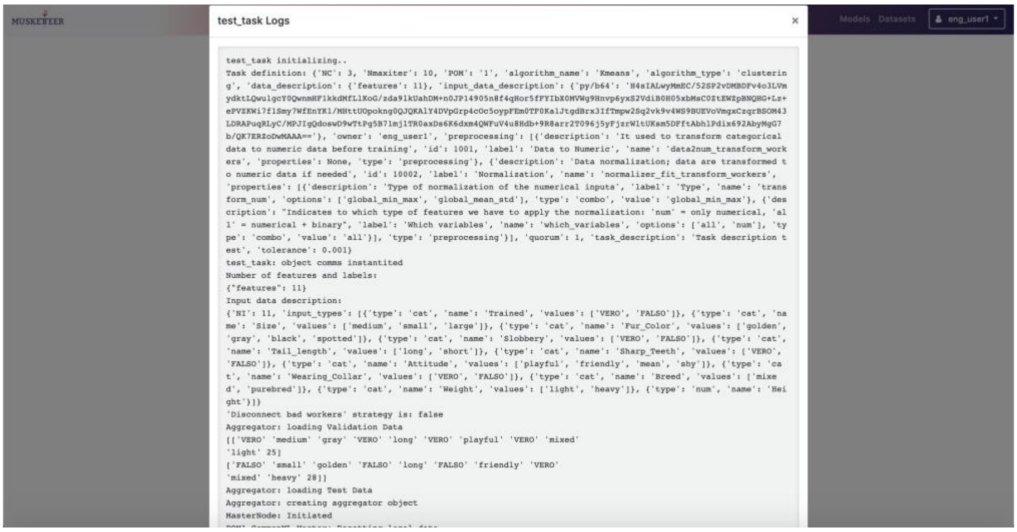
Test Dataset: -- drop dataset here --

Figure 15 - Task aggregator modal

2.6.2 Logs

When a user is participating to a task as an aggregator or participant, it is possible to monitor the progress in the task by viewing the logs produced by the script responsible for running the algorithm defined in the task.

From the detail of a task, clicking on the "logs" button displays the logs in a modal as shown in Figure 16 below. The inside of the modal containing the text with the logs is automatically updated if there are new logs; this is achieved by using a Server-Sent Events (SSE) technology enabling the client to receive automatic updates from the Client Connector back-end via an HTTP connection.



test_task Logs

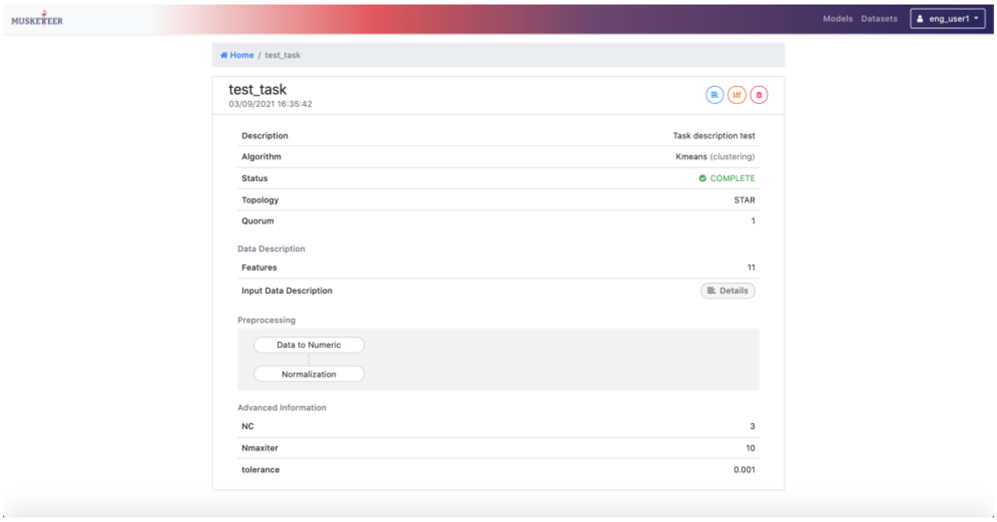
```
test_task initializing..
Task definition: {'NC': 3, 'Nmaxiter': 10, 'POM': '1', 'algorithm_name': 'Kmeans', 'algorithm_type': 'clusterin
g', 'data_description': {'features': 11}, 'input_data_description': {'py/b64': 'H4sIALwyMmEC/52SP2vDMBDFv4o3LVm
ydktLQwulgcY0QwnmHF1kkdMfLlKoG/zda9lkUahDM+n0JP14905n8f4qHor5fFYIbX0MVWg9Hnvp6yxS2VdiB0H05xbMsC0ZtEWZpBNQHG+Lz+
ePVZKWi7f1Smy7WfEnYKl/MHttUOpokng0QJQKAlY4DVpGrp4cOc5oypFEm0TF0KalJtgdBrx3IfTmpw2Sq2vk9v4WS9BUEVoVmgxCzqrBSOM43
LDRAPuqRLyC/MPJIgQdoswD9wTtPg5B7lmjlTR0axDs6K6dxm4QWFuV4u8Hdb+9R8arr2T096j5yFjzrWltUKsm5DFftAbhlPdix692AbyMgG7
b/QK7ERZoDwMAAA=='}, 'owner': 'eng_user1', 'preprocessing': [{'description': 'It used to transform categorical
data to numeric data before training', 'id': 1001, 'label': 'Data to Numeric', 'name': 'data2num_transform_work
ers', 'properties': None, 'type': 'preprocessing'}, {'description': 'Data normalization; data are transformed t
o numeric data if needed', 'id': 10002, 'label': 'Normalization', 'name': 'normalizer_fit_transform_workers',
'properties': [{'description': 'Type of normalization of the numerical inputs', 'label': 'Type', 'name': 'trans
form_num', 'options': ['global_min_max', 'global_mean_std'], 'type': 'combo', 'value': 'global_min_max'}, {'des
cription': "Indicates to which type of features we have to apply the normalization: 'num' = only numerical, 'al
l' = numerical + binary", 'label': 'Which variables', 'name': 'which_variables', 'options': ['all', 'num'], 'ty
pe': 'combo', 'value': 'all'}], 'type': 'preprocessing'}], 'quorum': 1, 'task_description': 'Task description t
est', 'tolerance': 0.001}
test_task: object comms instantited
Number of features and labels:
{"features": 11}
Input data description:
{'NI': 11, 'input_types': [{'type': 'cat', 'name': 'Trained', 'values': ['VERO', 'FALSO']}, {'type': 'cat', 'na
me': 'Size', 'values': ['medium', 'small', 'large']}, {'type': 'cat', 'name': 'Fur_Color', 'values': ['golden',
'gray', 'black', 'spotted']}, {'type': 'cat', 'name': 'Slobbery', 'values': ['VERO', 'FALSO']}, {'type': 'cat',
'name': 'Tail_length', 'values': ['long', 'short']}, {'type': 'cat', 'name': 'Sharp_Teeth', 'values': ['VERO',
'FALSO']}, {'type': 'cat', 'name': 'Attitude', 'values': ['playful', 'friendly', 'mean', 'shy']}, {'type': 'ca
t', 'name': 'Wearing_Collar', 'values': ['VERO', 'FALSO']}, {'type': 'cat', 'name': 'Breed', 'values': ['mixe
d', 'purebred']}, {'type': 'cat', 'name': 'Weight', 'values': ['light', 'heavy']}, {'type': 'num', 'name': 'Hei
ght'}]}
'Disconnect bad workers' strategy is: false
Aggregator: loading Validation Data
[['VERO' 'medium' 'gray' 'VERO' 'long' 'VERO' 'playful' 'VERO' 'mixed'
'light' 25]
['FALSO' 'small' 'golden' 'FALSO' 'long' 'FALSO' 'friendly' 'VERO'
'mixed' 'heavy' 28]]
Aggregator: loading Test Data
Aggregator: creating aggregator object
MasterNode: Initiated
```

Figure 16 - Task logs

2.6.3 Result chart

Once the task has reached at least one participant, it switches from "CREATED" status to "PENDING" status; once the chosen quorum of the task has been reached, it switches to "STARTED" status, indicating that the task is successfully started on both the aggregator and participant sides. If something goes wrong, the task will switch to "FAILED" status, and you can analyse the logs of the individual users (as shown in the previous paragraph) in order to understand the reason for the error.

The following Figure 17 shows the detail of a completed example task.



test_task
03/09/2021 16:35:42

Description	Task description test
Algorithm	Kmeans (clustering)
Status	COMPLETE
Topology	STAR
Quorum	1

Data Description

Features	11
Input Data Description	Details

Preprocessing

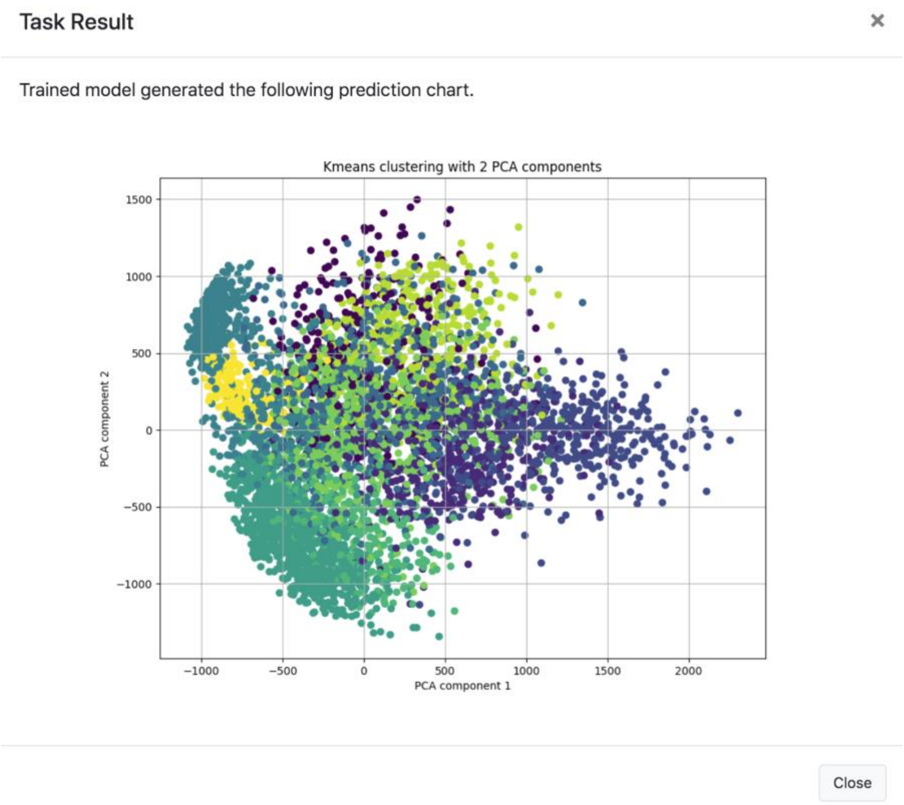
Data to Numeric
Normalization

Advanced Information

NC	3
Nmaxiter	10
tolerance	0.001

Figure 17 - A completed task card

A orange button appears for completed tasks, as shown in the previously Figure, that opens a modal showing the resulting chart as shown in Figure 18.

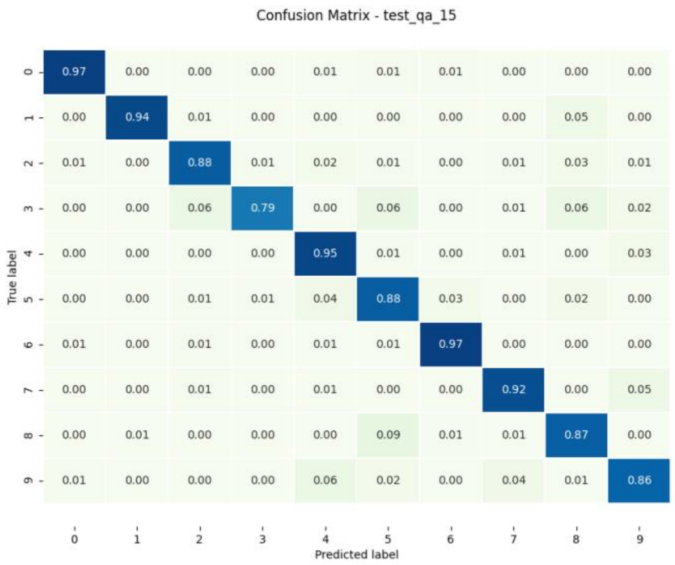

Task Result

Trained model generated the following prediction chart.

Figure 18 – Clustering Algorithm – 2D Scatterplot result

A different chart is produced depending on the type of task algorithm chosen:

- clustering: if the algorithm is clustering type, a scatterplot of the clustered data from the resulting model is produced on the two principal components of the test dataset, applying a Principal Component Analysis (PCA) algorithm, in order to visualise the clusters in a 2D space;
- classification: a confusion matrix is generated on the test data as shown in the Figure 19 below;



	0	1	2	3	4	5	6	7	8	9
0	0.97	0.00	0.00	0.00	0.01	0.01	0.01	0.00	0.00	0.00
1	0.00	0.94	0.01	0.00	0.00	0.00	0.00	0.00	0.05	0.00
2	0.01	0.00	0.88	0.01	0.02	0.01	0.00	0.01	0.03	0.01
3	0.00	0.00	0.06	0.79	0.00	0.06	0.00	0.01	0.06	0.02
4	0.00	0.00	0.00	0.00	0.95	0.01	0.00	0.01	0.00	0.03
5	0.00	0.00	0.01	0.01	0.04	0.88	0.03	0.00	0.02	0.00
6	0.01	0.00	0.01	0.00	0.01	0.01	0.97	0.00	0.00	0.00
7	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.92	0.00	0.05
8	0.00	0.01	0.00	0.00	0.00	0.09	0.01	0.01	0.87	0.00
9	0.01	0.00	0.00	0.00	0.06	0.02	0.00	0.04	0.01	0.86

Figure 19 - Classification Algorithm – Confusion matrix result

- regression: using the test data, the coefficient of determination (R2) and root mean squared error (RMSE) metrics are evaluated, as shown in the Figure 20 below.

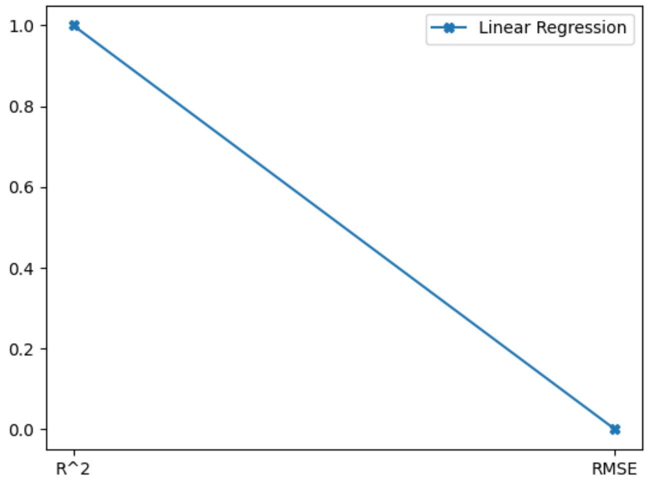
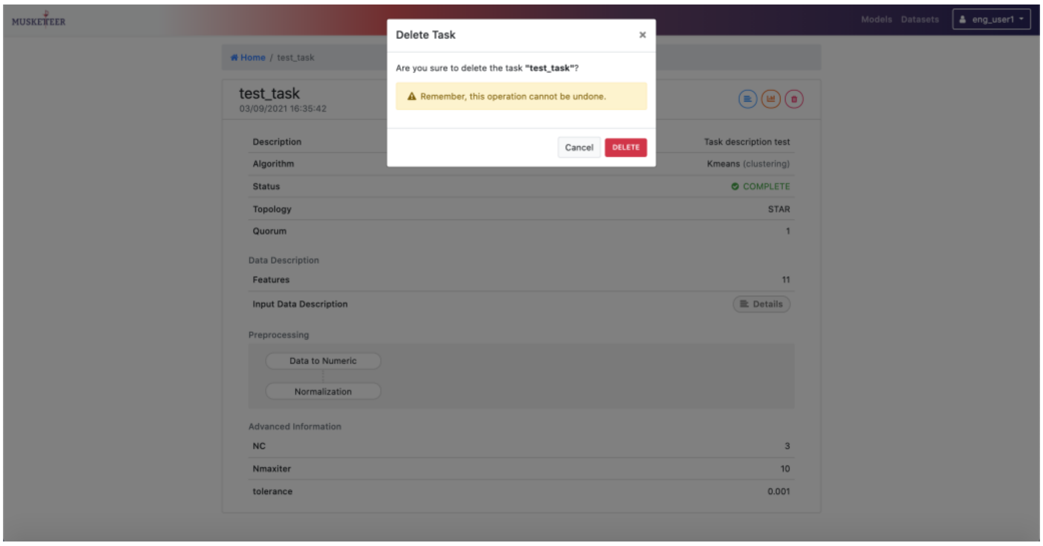


Figure 20 - Regression Algorithm – R2 and RMSE result

2.6.4 Task deletion

The button for deleting a task can only be used if you are the creator of that task. A task can be deleted from the MUSKETEER platform even if it has been completed or failed; once deleted, it will no longer be present in the list of tasks stored in the platform.

Figure 21 below shows the deletion action of a completed example task.



Delete Task

Are you sure to delete the task **"test_task"**?

Remember, this operation cannot be undone.

test_task
03/09/2021 16:35:42

Description	Task description test
Algorithm	Kmeans (clustering)
Status	COMPLETE
Topology	STAR
Quorum	1

Data Description

Features	11
Input Data Description	Details

Preprocessing

Data to Numeric
Normalization

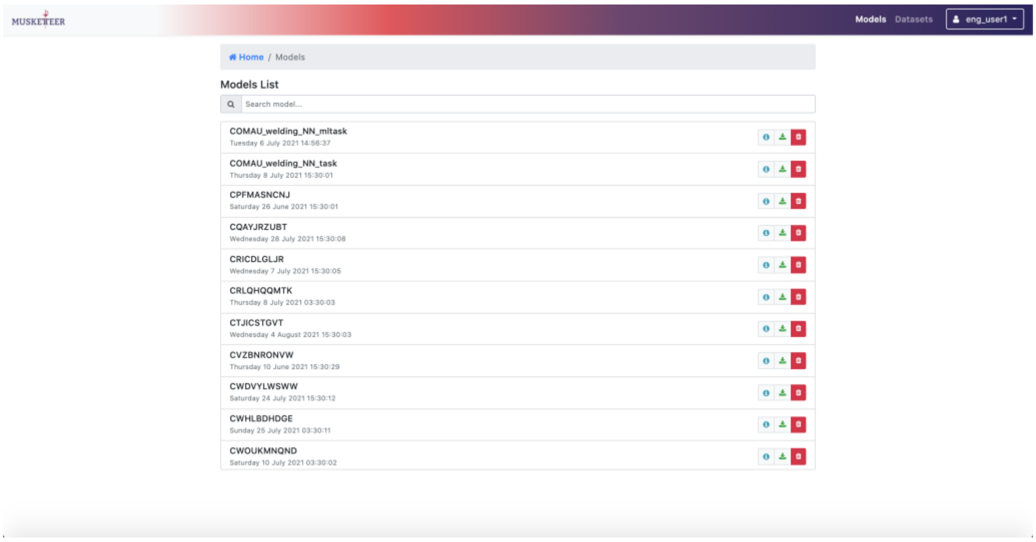
Advanced Information

NC	3
Nmaxiter	10
tolerance	0.001

Figure 21 – Task deletion

2.7 Models

The "Models" Section, at the URL "http://localhost:4500/models", contains the list of models trained and uploaded to the MUSKETEER platform. Figure 22 below shows the page in object, as well as the list of models, with the name of their completed tasks.



Models List

COMAU_welding_NN_mltask
Tuesday 6 July 2021 14:56:37

COMAU_welding_NN_task
Thursday 8 July 2021 15:30:01

CPFMASNCNJ
Saturday 26 June 2021 15:30:01

CQAYJRZUBT
Wednesday 28 July 2021 15:30:08

CRICDLGLJR
Wednesday 7 July 2021 15:30:05

CRLQHQQMTK
Thursday 8 July 2021 03:30:03

CTJICSTGVT
Wednesday 4 August 2021 15:30:03

CVZBNRONVW
Thursday 10 June 2021 15:30:29

CWDVYLWSWW
Saturday 24 July 2021 15:30:12

CWHLBDHDGE
Sunday 25 July 2021 03:30:11

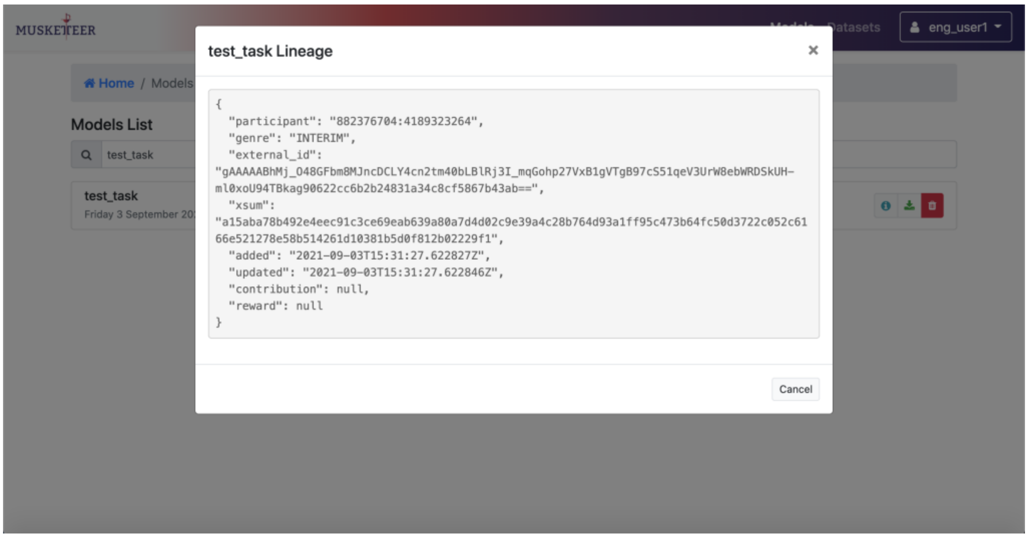
CWOUKMNQND
Saturday 10 July 2021 03:30:02

Figure 22 – List of trained models

Models can be filtered by name; it is also possible, using the buttons on the right of each model, to request the following actions through the configured communication library:

- Model lineage: request for the model lineage, an example is shown in the following Figure 23;
- Model download: request for the model download, choosing the type of format in which to save the model: PKL, ONNX and PMML.
- Model deletion: request for the model deletion.

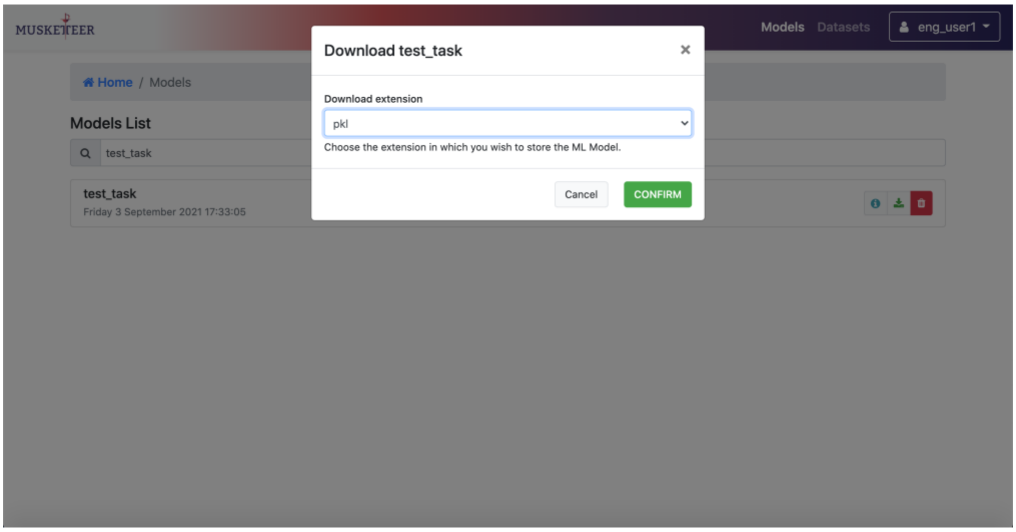
Further details are given in D3.4, as the communication library used towards the MUSKETEER platform has been implemented by IBM



```
{
  "participant": "882376704:4189323264",
  "genre": "INTERIM",
  "external_id":
"gAAAAABhMj_O48GFbm8MJncDCLY4cn2tm40bLBlRj3I_mqGohp27VxB1gVTgB97cS51qeV3UrW8ebWRDSkUH-
ml0xoU94TBkag90622cc6b2b24831a34c8cf5867b43ab==",
  "xsum":
"a15aba78b492e4eec91c3ce69eab639a80a7d4d02c9e39a4c28b764d93a1ff95c473b64fc50d3722c052c61
66e521278e58b514261d10381b5d0f812b02229f1",
  "added": "2021-09-03T15:31:27.622827Z",
  "updated": "2021-09-03T15:31:27.622846Z",
  "contribution": null,
  "reward": null
}
```

Figure 23 – Model lineage example

Figure 24 below shows the modal displayed for a model download request. Before confirming, it is necessary to choose the data format in which to save the model downloaded from the MUSKETEER platform. If you do not have the permissions to download a given model, or the specified data format is not available, an appropriate error message will be shown to the user; vice versa, a message will be shown that the model has been successfully stored in the data volume specified during the configuration of the Client Connector (as explained in Section 2.1).

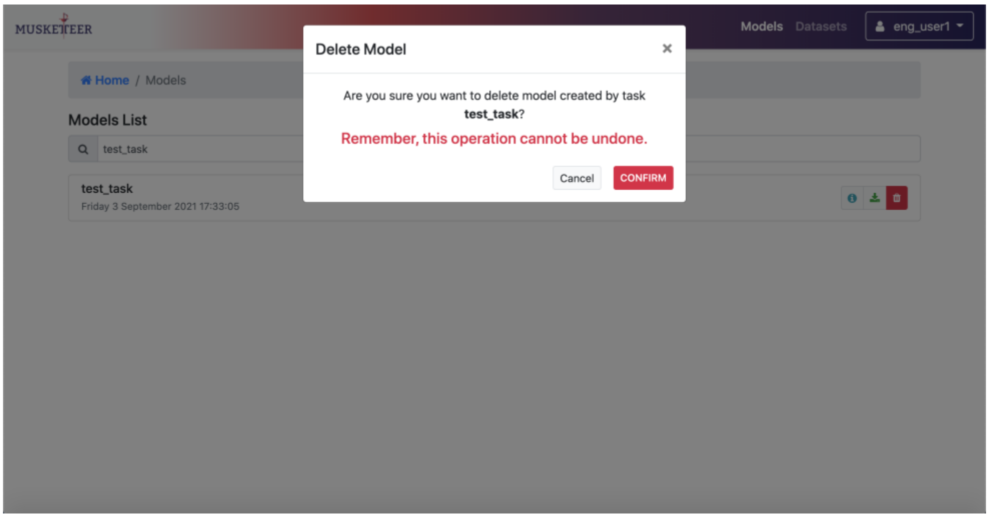


Download extension: pkl

Choose the extension in which you wish to store the ML Model.

Figure 24 – Download request of a model

In the same way, when requesting the deletion of a model, as shown in Figure 25, an error message will be displayed if you do not have deletion permissions, otherwise the model will be correctly deleted from the MUSKETEER platform.



Are you sure you want to delete model created by task **test_task**?

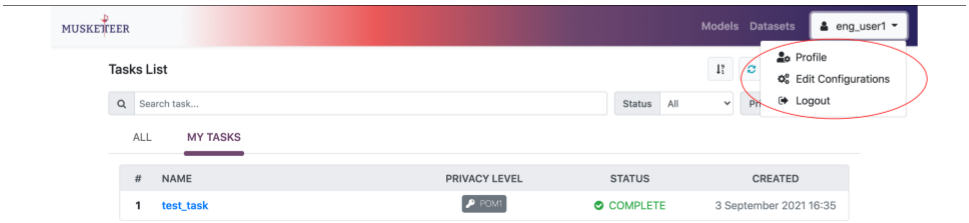
Remember, this operation cannot be undone.

test_task
Friday 3 September 2021 17:33:05

Figure 25 – Deletion request of a model

2.8 Client Connector settings

This paragraph describes the settings provided by the Client Connector, which can be accessed by moving the cursor to the top right of the user name, as shown in the following Figure 26.



#	NAME	PRIVACY LEVEL	STATUS	CREATED
1	test_task	POM1	COMPLETE	3 September 2021 16:35

Figure 26 – Client Connector settings

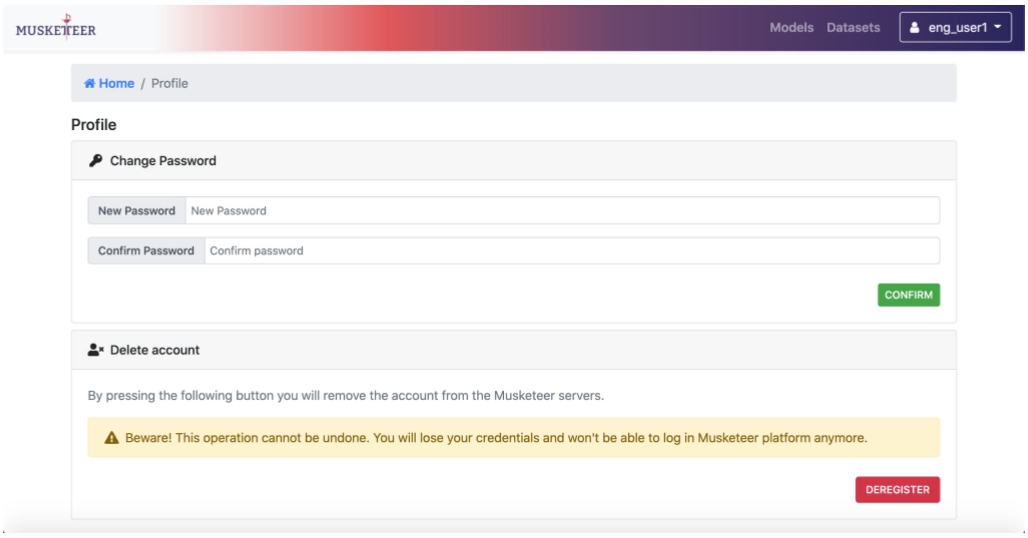
The user can then access the profile area, modify or update their imported library configurations, or log-out from the Client Connector.

2.8.1 Profile

The profile area page, shown in Figure 27, located at the URL "http://localhost:4500/settings/profile", integrates two other functionalities of the imported communication library in order to:

- change the user's password by providing a new one;

- remove the user's account from the platform, following final confirmation by the user.



Home / Profile

Profile

Change Password

New Password: New Password

Confirm Password: Confirm password

Delete account

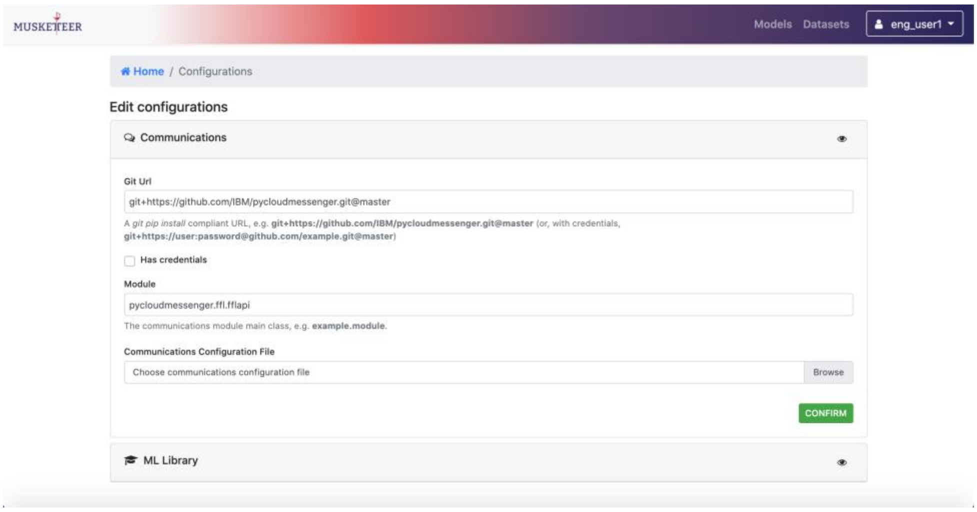
By pressing the following button you will remove the account from the Musketeer servers.

Beware! This operation cannot be undone. You will lose your credentials and won't be able to log in Musketeer platform anymore.

Figure 27 – Profile area page

2.8.2 Edit libraries configurations

The Client Connector configurations, once set as explained in Section 2.1, can be updated by going to the URL: "http://localhost:4500/settings/edit-configurations" or by clicking on the "Edit Configurations" area from the user settings.



Home / Configurations

Edit configurations

Communications

Git Url: git+https://github.com/IBM/pycloudmessenger.git@master

A *git pip install* compliant URL, e.g. **git+https://github.com/IBM/pycloudmessenger.git@master** (or, with credentials, **git+https://user:password@github.com/example.git@master**)

□ Has credentials

Module: pycloudmessenger.ffl.fflapi

The communications module main class, e.g. **example.module.**

Communications Configuration File: Choose communications configuration file

ML Library

Figure 28 – Edit libraries configurations page

The edit configurations page is presented as shown in Figure 28. From this page it is possible to modify the two libraries already imported, the one for communication to the target server and the one for federated machine learning, separately. Thus, the user can always update the libraries already set or change them to include new ones.

3 Musketeer Client Connector APIs

In this Section are documented all the exposed RESTful API by the Client Connector Back-End component. The documentation includes the category of the resource, the method (e.g. whether GET or POST), the endpoint, a description of the resource; and then more detailed information such as the required parameters, an example or schema of the request body and the response.

MUSKETEER Client Connector RESTful API	
API Reference ID	#1
Category	CATALOGUE
Method	GET
Endpoint	/cc/catalogue/algorithms

Resource description

Get the list of algorithms metadata provided during the Client Connector configuration steps.

Parameters

Request body example/schema

Response example/schema

```
{
  "algorithms": [
    {
      "id": 1,
      "POM": 1,
      "type": "clustering",
      "name": "Kmeans",
      "label": "Kmeans",
      "description": "Kmeans clustering algorithm.",
      "properties": [
        {
          "name": "Nmaxiter",
          "label": "Max number of iterations.",
          "defaultValue": 2,
          "type": "number",
          "description": "Max number of epochs."
        },
        {
          "name": "NC",
          "label": "Number of centroids",
          "defaultValue": 2,
          "type": "number",
          "description": "Number of centroids"
        },
        {
          "name": "tolerance",
          "label": "Convergence threshold to stop training.",
          "defaultValue": 0.001,
          "type": "number",
          "description": "Convergence threshold to stop training."
        }
      ]
    }
  ]
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#2
Category	CATALOGUE
Method	GET

Endpoint	/cc/catalogue/poms
Resource description	
Get the list of privacy operation modes (POMs) provided towards the MUSKETEER project.	
Parameters	
Request body example/schema	
Response example/schema	

```
{
  "poms": [
   {
    "name": "Aramis",
    "privacy": 1,
    "description": "Data cannot leave the facilities of each data owner, and the predictive models are transferred without encryption. At every client a gradient update is computed using a Federated Learning scheme.",
    "label": "POM1",
    "specs": {
      "privacy": 3,
      "overload": 3,
      "client": true,
      "server": false,
      "storage": 1,
      "communication": 3,
      "accountability": 3
    }
   },
   {
    "name": "Athos",
    "privacy": 2,
    "description": "The server can operate in the encrypted domain without having access to the unencrypted model. This schema is designed for use cases where the same data owner has data allocated in different locations, data cannot be moved for legal/architectural reasons, and the predictive model will be private.",
    "label": "POM2",
    "specs": {
      "privacy": 4,
      "overload": 3,
      "client": true,
      "server": false,
      "storage": 1,
      "communication": 4,
      "accountability": 2
    }
   }
  ]
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#3
Category	CLIENT CONNECTOR - CONFIGURATION
Method	GET
Endpoint	/cc/configurations/step
Resource description	
Get a number indicating the configuration status of the Client Connector: 1 - to install the communication and FML algorithms library; 2 - to install the FML algorithm library; -1 - Client Connector fully configured with the libraries installed.	
Parameters	
Request body example/schema	

Response example/schema

{
"step": -1
}

MUSKETEER Client Connector RESTful API	
API Reference ID	#4
Category	CLIENT CONNECTOR - CONFIGURATION
Method	POST
Endpoint	/cc/configurations/comm
Resource description	
Add the communication library configuration to integrate into the Client Connector. Required information: - comms_git_url: Git URL where to download the comms package - comms_git_token: Git Token to access private repository - comms_module: module name to import - comms_config: JSON configuration for the comms instance used.	
Parameters	
Request body example/schema	

{
"comms_git_url": "str",
"comms_git_token": "str"
"comms_module": "str",
"comms_config": {}
}	

Response example/schema	

{
"success": true
}	

MUSKETEER Client Connector RESTful API	
API Reference ID	#5
Category	CLIENT CONNECTOR - CONFIGURATION
Method	POST
Endpoint	/cc/configurations/comm

Resource description

Add the federated machine learning library configuration to integrate into the Client Connector. Required information: - mmll_git_url: Git URL where to download the Federated Machine Learning package - mmll_masternode_classpath - mmll_workernode_classpath - mmll_comms_wrapper_classpath - mmll_algorithms: JSON file containing the algorithms specifications and details.

Parameters

Request body example/schema

```
{
  "mmll_git_url": "str",
  "mmll_masternode_classpath": "str",
  "mmll_workernode_classpath": "str",
  "mmll_comms_wrapper_classpath": "str",
  "mmll_algorithms": {}
}
```

Response example/schema

```
{
  "success": true
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#6
Category	CLIENT CONNECTOR - CONFIGURATION

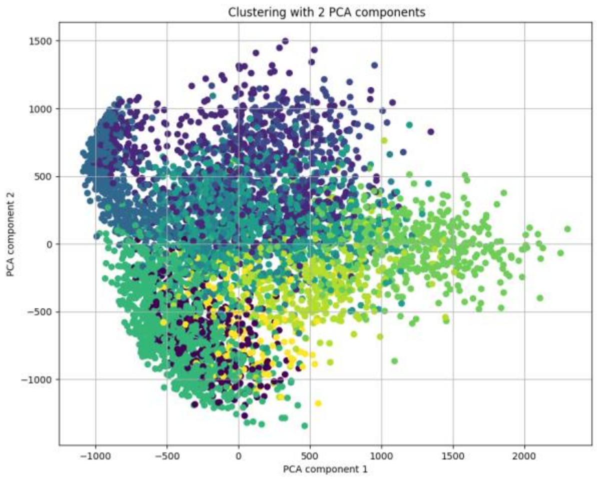
Method	GET
Endpoint	/cc/configurations/comm
Resource description	
Get the communication library configuration.	
Parameters	
Request body example/schema	
Response example/schema	

{
"comms_git_url": "str",
"comms_git_token": "str"
"comms_module": "str",
"comms_config": {}
}	

MUSKETEER Client Connector RESTful API	
API Reference ID	#7
Category	CLIENT CONNECTOR - CONFIGURATION
Method	GET
Endpoint	/cc/configurations/mmll
Resource description	
Get the Musketeer Machine Learning library configuration.	
Parameters	
Request body example/schema	
Response example/schema	

```
{
  "mmll_git_url": "str",
  "mmll_masternode_classpath": "str",
  "mmll_workernode_classpath": "str",
  "mmll_comms_wrapper_classpath": "str",
  "mmll_algorithms": {}
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#8
Category	CLIENT CONNECTOR - CONFIGURATION
Method	GET
Endpoint	/cc/results/image?task
Resource description	
Get a chart image resulting from the execution and completion of a Federated Machine Learning task.	
Parameters	
Query parameters: 1) task: the name of the completed task for which the result chart image has to be retrieved.	
Request body example/schema	
Response example/schema	



MUSKETEER Client Connector RESTful API	
API Reference ID	#9
Category	CLIENT CONNECTOR - CONFIGURATION
Method	GET
Endpoint	/cc/results/stream/logs?task&mode
Resource description	
Get the logs, as an EventStream, related to a specified task the user is participating/aggregating to.	
Parameters	
Query parameters: 1) task: name of the task. 2) mode (participant/aggregator): specifies whether the user is accessing a task logs as aggregator or participant.	
Request body example/schema	
Response example/schema	
{	

```
 "line": "Task initializing..\nData description: \n{\"features\": 11}\nAggregator: creating worker node object\nWorkerNode Anonymous: Loading comms\nWorkerNode Anonymous: Loading Data Connector\nWorkerNode Anonymous: Initiated\nParticipant: loading Training Data\nWorkerNode: Received input/target_data_description\nPOM1_KMeans_Worker Anonymous: READY and waiting instructions\nPOM1_KMeans_Worker Anonymous: Received CHECK_DATA from Master\nPOM1_KMeans_Worker Anonymous: Checking data\nPOM1_KMeans_Worker Anonymous: Sent ACK_CHECK_DATA to master\nPOM1_KMeans_Worker Anonymous: Received SEND_PREPROCESSOR from Master\nPOM1_KMeans_Worker Anonymous: Receiving preprocessor\nPOM1_KMeans_Worker Anonymous: Shape of original dataset: (200, 11)\nPOM1_KMeans_Worker Anonymous: Training set transformed using preprocessor data2num\nPOM1_KMeans_Worker Anonymous: Shape of transformed dataset: (200, 26)\nPOM1_KMeans_Worker Anonymous: Final preprocessor stored\nPOM1_KMeans_Worker Anonymous: Sent ACK_SEND_PREPROCESSOR to master\nPOM1_KMeans_Worker Anonymous: Received SEND_MEANS from Master\nPOM1_KMeans_Worker Anonymous: Obtaining means\nPOM1_KMeans_Worker Anonymous: Sent COMPUTE_MEANS to master\nPOM1_KMeans_Worker Anonymous: Received SEND_STDS from Master\nPOM1_KMeans_Worker Anonymous: Obtaining stds\nPOM1_KMeans_Worker Anonymous: Sent COMPUTE_STDS to master\nPOM1_KMeans_Worker Anonymous: Received SEND_PREPROCESSOR from Master\nPOM1_KMeans_Worker Anonymous: Receiving preprocessor\nPOM1_KMeans_Worker Anonymous: Training set transformed using preprocessor\nPOM1_KMeans_Worker Anonymous: Final preprocessor stored\nPOM1_KMeans_Worker Anonymous: Sent ACK_SEND_PREPROCESSOR to master\nPOM1_KMeans_Worker Anonymous: Received SEND_CENTROIDS from Master\nPOM1_KMeans_Worker Anonymous: Initializing centroids\nPOM1_KMeans_Worker Anonymous: Sent INIT_CENTROIDS to master\nPOM1_KMeans_Worker Anonymous: Received COMPUTE_LOCAL_CENTROIDS from Master\nPOM1_KMeans_Worker Anonymous: Updating centroids\n"

}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#10
Category	CLIENT CONNECTOR - CONFIGURATION
Method	GET
Endpoint	/cc/datasets
Resource description	
Get the list of datasets metadata the user has registered into the Client Connector.	
Parameters	
Request body example/schema	

Response example/schema

```
[
 {
  "_id": "606eb290c03c776522229783",
  "name": "mnist_test_1",
  "added": "2021-04-08T07:36:48.907000",
  "format": "csv",
  "module": "CsvConnector",
  "path": "input_data/mnist_test_1.csv",
  "dimension": 9251376,
  "header": false
 },
 {
  "_id": "606eb2a8c03c776522229784",
  "name": "mnist_test_2",
  "added": "2021-04-08T07:37:12.821000",
  "format": "csv",
  "module": "CsvConnector",
  "path": "input_data/mnist_test_2.csv",
  "dimension": 9418067,
  "header": false
 },
 {
  "_id": "60fe760f0469f12319531e9f",
  "name": "mnist pkl",
  "added": "2021-07-26T08:45:03.951000",
  "format": "pkl",
  "module": "PklConnector",
  "path": "input_data/mnist_demonstrator_data.pkl",
    "dimension": 31760256,
    "label": true
  }
]
```

MUSKETEER Client Connector RESTful API

API Reference ID	#11
Category	CLIENT CONNECTOR - CONFIGURATION
Method	POST
Endpoint	/cc/datasets

Resource description

Register a new dataset metadata into the Client Connector specifying the required information, depending on the type of storage (e.g. local), and the data format (e.g. CSV or PKL) used.

Parameters

Request body example/schema

```
{
  "type": "FileSystem",
  "spec": {
    "name": "name of the dataset",
    "path": "mnist_test_1.csv",
    "format": "csv",
    "header": false
  }
}
```

Response example/schema

```
{
  "success": true
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#12
Category	CLIENT CONNECTOR - CONFIGURATION
Method	DELETE
Endpoint	/cc/datasets/<_id>
Resource description	
Delete a specified dataset metadata through its unique identifier assigned to it.	
Parameters	
Path parameters: 1) _id: the dataset unique identifier.	
Request body example/schema	
Response example/schema	

{
"success": true
}	

MUSKETEER Client Connector RESTful API	
API Reference ID	#13
Category	CLIENT CONNECTOR - CONFIGURATION
Method	PUT
Endpoint	/cc/datasets/<_id>
Resource description	

Modify/Update a specified dataset metadata through its unique identifier assigned to it, and inserting the information to update.
Parameters
Path parameters: 1) _id: the dataset unique identifier.
Request body example/schema

{
"name": "modified dataset name"
}

Response example/schema

{
"success": true
}

MUSKETEER Client Connector RESTful API	
API Reference ID	#14
Category	CLIENT CONNECTOR - COMMUNICATION
Method	POST
Endpoint	/cc/comms/login
Resource description	
Access to the target platform using the configured communication messenger library.	
Parameters	
Request body example/schema	

{
"user": "username",
"password": "password"
}	

Response example/schema

{
"success": true
}

MUSKETEER Client Connector RESTful API	
API Reference ID	#15
Category	CLIENT CONNECTOR - COMMUNICATION
Method	POST
Endpoint	/cc/comms/logout
Resource description	
Logout from the target platform.	
Parameters	
Request body example/schema	
{}	
Response example/schema	

{
"success": true
}	

MUSKETEER Client Connector RESTful API	
API Reference ID	#16
Category	CLIENT CONNECTOR - COMMUNICATION
Method	POST
Endpoint	/cc/comms/registration
Resource description	

Register an account to the target platform using the configured communication messenger library; information to set are the following: username, password and organization name.
Parameters
Request body example/schema

{
"user": "username",
"password": "password",
"org": "organization name"
}

Response example/schema

{
"success": true
}

MUSKETEER Client Connector RESTful API	
API Reference ID	#17
Category	CLIENT CONNECTOR - COMMUNICATION
Method	PATCH
Endpoint	/cc/comms/change_password
Resource description	
Change the password access credential (of the logged-in user) to the target platform using the configured communication messenger library.	
Parameters	
Request body example/schema	

{
"new_password": "password"
}	

Response example/schema

{
"success": true
}

MUSKETEER Client Connector RESTful API	
API Reference ID	#18
Category	CLIENT CONNECTOR - COMMUNICATION
Method	DELETE
Endpoint	/cc/comms/deregister
Resource description	
Deregister the logged-in user from the target platform using the configured communication messenger library.	
Parameters	
Request body example/schema	
Response example/schema	

{
"success": true
}	

MUSKETEER Client Connector RESTful API	
API Reference ID	#19
Category	CLIENT CONNECTOR - COMMUNICATION
Method	GET
Endpoint	/cc/comms/tasks

Resource description

Get the list of the tasks stored in the target platform using the configured communication messenger library. The information retrieved includes: task name, status, task topology, date of creation and update of the task, and a "definition" field that contains the description of the task and all the information needed to execute a specific algorithm, of a given POM, using the Federated Machine Learning library configured in the Client Connector.

Parameters

Request body example/schema

Response example/schema

```
[
 {
  "task_name": "taskExample",
  "status": "COMPLETE",
  "queue": null,
  "topology": "STAR",
  "definition": "{\"NC\": 3, \"Nmaxiter\": 5, \"POM\": \"1\", \"algorithm_name\": \"Kmeans\",
\"algorithm_type\": \"clustering\", \"data_description\": {\"features\": 11}, \"input_data_description\":
{\"py/b64\":
\"H4sIAKRBL2EC/52SP2vDMBDFv4o3LVmydktLQwulgcY0QwnmHF1kkdMfLlKoG/zda9lkUahDM+n0JP14905n
8f4qHor5fFYIbX0MVWg9Hnvp6yxS2VdiB0H05xbMsC0ZtEWZpBNQHG+Lz+ePVZKWi7f1Smy7WfEnYK1/MHttU
Opokng0QJQKAlY4DVpGrp4cOc5oypFEm0TF0Ka1JtgdBrx3IfTmpw2Sq2vk9v4WS9BUEVoVmgxCzqrBSOM43L
DRAPuqRLyC/MPJIgQdoswD9wTtPg5B71mjlTR0axDs6K6dxm4QWFuV4u8Hdb+9R8arr2T096j5yFjzrWltUKsm5
DFftAbhlPdix692AbyMgG7b/QK7ERZoDwMAAA==\"}, \"owner\": \"test5\", \"preprocessing\":
[{\"description\": \"It used to transform categorical data to numeric data before training\", \"id\": 1001,
\"label\": \"Data to Numeric\", \"name\": \"data2num_transform_workers\", \"properties\": null, \"type\":
\"preprocessing\"}, {\"description\": \"Data normalization; data are transformed to numeric data if needed\",
\"id\": 10002, \"label\": \"Normalization\", \"name\": \"normalizer_fit_transform_workers\", \"properties\":
[{\"description\": \"Type of normalization of the numerical inputs\", \"label\": \"Type\", \"name\":
\"transform_num\", \"options\": [\"global_min_max\", \"global_mean_std\"], \"type\": \"combo\",
\"value\": \"global_mean_std\"}, {\"description\": \"Indicates to which type of features we have to apply the
normalization: 'num' = only numerical, 'all' = numerical + binary\", \"label\": \"Which variables\", \"name\":
\"which_variables\", \"options\": [\"all\", \"num\"], \"type\": \"combo\", \"value\": \"all\"}], \"type\":
```

```
\"preprocessing\"}], \"quorum\": 1, \"task_description\": \"A FML clustering description.\", \"tolerance\": 0.001}",
    "added": "2021-09-01T09:02:30.218370Z",
    "updated": "2021-09-01T09:09:59.472765Z"
  }
]
```

<table>
<tr><th colspan="2">MUSKETEER Client Connector RESTful API</th></tr>
<tr><td><b>API Reference ID</b></td><td>#20</td></tr>
<tr><td><b>Category</b></td><td>CLIENT CONNECTOR - COMMUNICATION</td></tr>
<tr><td><b>Method</b></td><td>POST</td></tr>
<tr><td><b>Endpoint</b></td><td>/cc/comms/tasks</td></tr>
<tr><td colspan="2"><b>Resource description</b></td></tr>
<tr><td colspan="2">Create a new task by inserting the name of the task and all required fields to describe a task towards the configured communication messenger and federated machine learning libraries.</td></tr>
<tr><td colspan="2"><b>Parameters</b></td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2"><b>Request body example/schema</b></td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2"><b>Response example/schema</b></td></tr>
<tr><td colspan="2">{
"success": true
}</td></tr>
</table>

<table>
<tr><th colspan="2">MUSKETEER Client Connector RESTful API</th></tr>
<tr><td><b>API Reference ID</b></td><td>#21</td></tr>
<tr><td><b>Category</b></td><td>CLIENT CONNECTOR - COMMUNICATION</td></tr>
<tr><td><b>Method</b></td><td>GET</td></tr>
</table>

Endpoint	/cc/comms/tasks/<task_name>

Resource description

Get the task information of a specified task using the configured communication messenger library; in addition to the task specification, another information, useful for the graphical interface, is included so that action icons are shown or not depending on the logged in user. For instance, only the task creator can aggregate or delete his own task.

Parameters

Path parameters:

1) task_name: name of the task.

Request body example/schema

Response example/schema

```
{
  "task_name": "taskExample",
  "status": "COMPLETE",
  "topology": "STAR",
  "definition": "{\"NC\": 3, \"Nmaxiter\": 5, \"POM\": \"1\", \"algorithm_name\": \"Kmeans\",
\"algorithm_type\": \"clustering\", \"data_description\": {\"features\": 11}, \"input_data_description\":
{\"py/b64\":
\"H4sIAKRBL2EC/52SP2vDMBDFv4o3LVmydktLQwulgcY0QwnmHF1kkdMfLlKoG/zda9lkUahDM+n0JP14905n
8f4qHor5fFYIbX0MVWg9Hnvp6yxS2VdiB0H05xbMsC0ZtEWZpBNQHG+Lz+ePVZKWi7f1Smy7WfEnYK1/MHttU
Opokng0QJQKAlY4DVpGrp4cOc5oypFEm0TF0Ka1JtgdBrx3IfTmpw2Sq2vk9v4WS9BUEVoVmgxCzqrBSOM43L
DRAPuqRLyC/MPJIgQdoswD9wTtPg5B71mjlTR0axDs6K6dxm4QWFuV4u8Hdb+9R8arr2T096j5yFjzrWltUKsm5
DFftAbhlPdix692AbyMgG7b/QK7ERZoDwMAAA==\"}, \"owner\": \"test5\", \"preprocessing\":
[{\"description\": \"It used to trans-form categorical data to numeric data before training\", \"id\": 1001,
\"label\": \"Data to Numer-ic\", \"name\": \"data2num_transform_workers\", \"properties\": null, \"type\":
\"preprocessing\"}, {\"description\": \"Data normalization; data are transformed to numeric data if needed\",
\"id\": 10002, \"label\": \"Normalization\", \"name\": \"normalizer_fit_transform_workers\", \"properties\":
[{\"description\": \"Type of normalization of the numerical inputs\", \"label\": \"Type\", \"name\":
\"transform_num\", \"options\": [\"global_min_max\", \"global_mean_std\"], \"type\": \"combo\",
\"value\": \"global_mean_std\"}, {\"description\": \"Indicates to which type of features we have to apply the
normalization: 'num' = only numerical, 'all' = numerical + binary\", \"label\": \"Which variables\", \"name\":
\"which_variables\", \"options\": [\"all\", \"num\"], \"type\": \"combo\", \"value\": \"all\"}], \"type\":
\"preprocessing\"}], \"quorum\": 1, \"task_description\": \"A FML clus-tering description.\", \"tolerance\":
0.001}",
  "added": "2021-09-01T09:02:30.218370Z",
  "updated": "2021-09-01T09:09:59.472765Z",
  "actions": {
    "aggregate": -1,
    "participate": -1,
    "result": 1,
    "logs": 1,
    "delete": 1
  }
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#22
Category	CLIENT CONNECTOR - COMMUNICATION
Method	DELETE
Endpoint	/cc/comms/tasks/<task_name>
Resource description	
Delete a specified task in the target platform using the configured communication messenger library.	
Parameters	
Path parameters: 1) task_name: name of the task.	
Request body example/schema	
Response example/schema	

{
"success": true
}	

MUSKETEER Client Connector RESTful API	
API Reference ID	#23
Category	CLIENT CONNECTOR - COMMUNICATION
Method	GET
Endpoint	/cc/comms/tasks/created

Resource description

Get the list of the tasks created by the user in the target platform using the configured communication messenger library.

Parameters

Request body example/schema

Response example/schema

```
[
 {
  "task_name": "myTaskExample",
  "status": "COMPLETE",
  "queue": null,
  "topology": "STAR",
  "definition": "{\"NC\": 3, \"Nmaxiter\": 5, \"POM\": \"1\", \"algorithm_name\": \"Kmeans\",
\"algorithm_type\": \"clustering\", \"data_description\": {\"features\": 11}, \"input_data_description\":
{\"py/b64\":
\"H4sIAKRBL2EC/52SP2vDMBDFv4o3LVmydktLQwulgcY0QwnmHF1kkdMfLlKoG/zda9lkUahDM+n0JP14905n
8f4qHor5fFYIbX0MVWg9Hnvp6yxS2VdiB0H05xbMsC0ZtEWZpBNQHG+Lz+ePVZKWi7f1Smy7WfEnYK1/MHttU
Opokng0QJQKAlY4DVpGrp4cOc5oypFEm0TF0Ka1JtgdBrx3IfTmpw2Sq2vk9v4WS9BUEVoVmgxCzqrBSOM43L
DRAPuqRLyC/MPJIgQdoswD9wTtPg5B71mjlTR0axDs6K6dxm4QWFuV4u8Hdb+9R8arr2T096j5yFjzrWltUKsm5
DFftAbhlPdix692AbyMgG7b/QK7ERZoDwMAAA==\"}, \"owner\": \"test5\", \"preprocessing\":
[{\"description\": \"It used to transform categorical data to numeric data before training\", \"id\": 1001,
\"label\": \"Data to Numeric\", \"name\": \"data2num_transform_workers\", \"properties\": null, \"type\":
\"preprocessing\"}, {\"description\": \"Data normalization; data are transformed to numeric data if needed\",
\"id\": 10002, \"label\": \"Normalization\", \"name\": \"normalizer_fit_transform_workers\", \"properties\":
[{\"description\": \"Type of normalization of the numerical inputs\", \"label\": \"Type\", \"name\":
\"transform_num\", \"options\": [\"global_min_max\", \"global_mean_std\"], \"type\": \"combo\",
\"value\": \"global_mean_std\"}, {\"description\": \"Indicates to which type of features we have to apply the normalization: 'num' = only numerical, 'all' = numerical + binary\", \"label\": \"Which variables\", \"name\": \"which_variables\", \"options\": [\"all\", \"num\"], \"type\": \"combo\", \"value\": \"all\"}], \"type\": \"preprocessing\"}], \"quorum\": 1, \"task_description\": \"A FML clustering description.\", \"tolerance\": 0.001}",

    "added": "2021-09-01T09:02:30.218370Z",

    "updated": "2021-09-01T09:09:59.472765Z"

  }

]
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#24
Category	CLIENT CONNECTOR - COMMUNICATION
Method	GET
Endpoint	/cc/comms/tasks/joined
Resource description	
Get the list with all the joined tasks in the target platform using the configured communication messenger library.	
Parameters	
Request body example/schema	
Response example/schema	

```
[

  {

    "task_name": "task_example",

    "tstatus": "PENDING",

    "queue": "1ff2b3704aede04eecb51e50ca698efd50a1379b/worker/task_example",

    "status": "CREATED",

    "added": "2021-09-01T13:38:34.360357Z",
   "updated": "2021-09-01T13:38:34.360377Z"
  }
]
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#25
Category	CLIENT CONNECTOR - COMMUNICATION
Method	GET
Endpoint	/cc/comms/tasks/assigned

Resource description

Get the list of all the tasks the user is participating in the target platform using the configured communication messenger library.

Parameters

Request body example/schema

Response example/schema

```
[
  {
   "task_name": "task_example",
   "tstatus": "PENDING",
   "queue": "1ff2b3704aede04eecb51e50ca698efd50a1379b/worker/task_example",
   "status": "CREATED",
   "added": "2021-09-01T13:38:34.360357Z",
   "updated": "2021-09-01T13:38:34.360377Z"
  }
]
```

MUSKETEER Client Connector RESTful API

API Reference ID	#26
Category	CLIENT CONNECTOR - COMMUNICATION
Method	GET
Endpoint	/cc/comms/models
Resource description	
Get the list with all the available trained models in the target platform using the configured communication messenger library.	
Parameters	
Request body example/schema	
Response example/schema	

```
[
 {
  "task_name": "task_1",
  "added": "2021-05-17T14:46:22.790308Z"
 },
 {
  "task_name": "task_2",
  "added": "2021-06-17T09:21:03.729240Z"
 },
 {
  "task_name": "task_3",
  "added": "2021-06-17T09:49:07.843029Z"
 }
]
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#27

Category	CLIENT CONNECTOR - COMMUNICATION
Method	GET
Endpoint	/cc/comms/<task_name>?extension
Resource description	
Requests a trained model, related to a specified task, in the target platform using the configured communication messenger library; the object obtained is then downloaded and saved locally by selecting one of the three extensions (formats) chosen: PKL, PMML, ONNX. An error message will be sent if a particular extension is not supported by the model.	
Parameters	
Query parameters: 1) extension (PKL, PMML, ONNX): the format in which to save the trained model. Path parameters: 1) task_name: name of the task.	
Request body example/schema	
Response example/schema	

{
"success": true,
"message": "The model resulting from task_name is saved in your local file system."
}	

MUSKETEER Client Connector RESTful API	
API Reference ID	#28
Category	CLIENT CONNECTOR - COMMUNICATION
Method	DELETE
Endpoint	/cc/comms/<task_name>
Resource description	

Requests a model deletion from the selected task in the target platform using the configured communication messenger library.
Parameters
Path parameters: 1) task_name: name of the task.
Request body example/schema
Response example/schema

{
"success": true
}

MUSKETEER Client Connector RESTful API	
API Reference ID	#29
Category	CLIENT CONNECTOR - COMMUNICATION
Method	GET
Endpoint	/cc/comms/<task_name>/lineage
Resource description	
Requests the model lineage, related to a specified task, in the target platform using the configured communication messenger library.	
Parameters	
Path parameters: 1) task_name: name of the task.	
Request body example/schema	
Response example/schema	
{ "participant": "879034368:4144300032",	

```
  "genre": "INTERIM",
  "external_id":          "gAAAAABhL4J4fqbU8ZwArkrDdmLxiQyrepminchsNDypeUiNRkdH0U-jmP1GH1Cre5Y--
xmxqN5JFIFEvPHDLZ4HTFmtsvLCmQdb8f9356689b484d9c4c8184c094f6c9==",
  "xsum":
"a15aba78b492e4eec91c3ce69eab639a80a7d4d02c9e39a4c28b764d93a1ff95c473b64fc50d3722c052c6166
e521278e58b514261d10381b5d0f812b02229f1",
  "added": "2021-09-01T13:39:06.013474Z",
  "updated": "2021-09-01T13:39:06.013493Z",
  "contribution": null,
  "reward": null
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#30
Category	CLIENT CONNECTOR – FEDERATED MACHINE LEARNING
Method	POST
Endpoint	/cc/fml/aggregate

Resource description

As a task owner, aggregate to a task by entering the information needed to access a validation and test dataset. Then run the script that will execute the Federated Machine Learning algorithm as aggregator defined in the task towards the configured and installed Federated Machine Learning and communication messenger libraries.

Parameters

Request body example/schema

```
{
  "task_name": "test_task",
  "datasets": {
    "validation": {
      "_id": "608ac2699bc3c510ce454131",
      "name": "my validation dataset",
      "added": "2021-04-29T14:27:53.641000",
      "format": "csv",
      "module": "CsvConnector",
      "path": "/input_data/validation.csv",
      "dimension": 13083,
      "header": true
    },
    "test": {
      "_id": "608ac2699bc3c510ce454131",
      "name": "my test dataset",
      "added": "2021-04-29T15:02:20.641000",
      "format": "csv",
      "module": "CsvConnector",
      "path": "/input_data/test.csv",
      "dimension": 7094,
      "header": true
    }
  }
}
```

Response example/schema

```
{
  "message": "Task test_task started as aggregator."
}
```

MUSKETEER Client Connector RESTful API	
API Reference ID	#31
Category	CLIENT CONNECTOR – FEDERATED MACHINE LEARNING
Method	POST
Endpoint	/cc/fml/participate
Resource description	

As a participant, join a task by entering at least the information needed to access a training dataset, and optionally a validation and test dataset. Then run the script that will execute the Federated Machine Learning algorithm as participant defined in the task towards the configured and installed Federated Machine Learning and communication messenger libraries.

Parameters

Request body example/schema

```
{
  "task_name": "test_task",
  "datasets": {
    "training": {
      "_id": "608ac2699bc3c510ce454131",
      "name": "my training dataset",
      "added": "2021-04-29T14:27:53.641000",
      "format": "csv",
      "module": "CsvConnector",
      "path": "/input_data/training.csv",
      "dimension": 13083,
      "header": true
    }
  }
}
```

Response example/schema

```
{
 "message": "Task test_task started as participant."
}
```

4 Unit Testing and Integration Testing

Software testing is the investigation conducted over a software artefact or product in order to provide useful insights concerning the quality of the software artefact or product under test. Within the context of software testing, a broad list of activities that can be performed depending on the phase of the software development lifecycle and their aim and purpose. There are two major categories of activities in software testing which are considered as the core and required activities on every software development project, namely the unit testing and the integration testing.

The main difference between those activities is the context of their execution within the software development lifecycle. In a nutshell, unit testing the software testing method that is performed on each individual unit or module that is developed in order to ensure and verify their functionality, while integration testing is the method where all components, composed by multiple units or modules, are combined and tested as a group. Hence, on each integration cycle, unit testing is performed first in order to verify that all developed units and modules are operating as expected on an individual level, while the integration testing is performed after the unit testing in order to verify the correctness of the interfaces between two or more components on a group level.

To the aim of client connector testing, an integration testing strategy was formulated and adopted from the early stages of the development. The strategy dictated for the design and execution of small and easily executable unit tests that are verifying the functionalities and the quality of each individual module of the component. The list of unit tests was expanded as the project evolved, while several existing tests were updated and enhanced in order to accommodate the new features and functionalities of each module between the platform releases. For the integration testing aspect, the strategy adopted the Umbrella approach: within this approach, a mixture of the activities performed on the top-down approach and the bottom-up approach is performed. In particular, both functional data and the flow of information are tested. To achieve this, the input for functions are integrated following the bottom-up approach and the outputs of the functions are then integrated following the top-down approach.

4.1 Unit Testing

MUSKETEER Client Connector Unit Test	
Test Case Reference ID	CM-01
Test Case Name	Check classpath configuration
Component Name	CONFIGURATION MANAGER

Description of the test case
After the library installation (communication messenger or machine learning library), the configuration manager checks that the classpath (set by the user during the configuration steps) exists and it can be correctly imported.
Input of the test case
Name of the classpath to be imported.
Output of the test case
True if the classpath can be correctly imported.
Results of the test case
SUCCESS

MUSKETEER Client Connector Unit Test	
Test Case Reference ID	CM-02
Test Case Name	Check module configuration
Component Name	CONFIGURATION MANAGER
Description of the test case	
After the library installation (communication messenger or machine learning library), the configuration manager checks that the import of the module set by the user during the configuration steps can be correctly imported.	
Input of the test case	
Name of the module library to be imported.	
Output of the test case	
True if the module can be correctly imported.	
Results of the test case	
SUCCESS	

MUSKETEER Client Connector Unit Test	
Test Case Reference ID	DC-03
Test Case Name	Check dataset existence
Component Name	DATA CONNECTOR
Description of the test case	
During dataset metadata submission, before loading this information to the database, it is first checked whether the corresponding path contains the dataset.	
Input of the test case	
Dataset metadata needed by the Data Connector (e.g. file path) to access the target dataset.	
Output of the test case	
True if the dataset exists at the specified location.	
Results of the test case	
SUCCESS	

MUSKETEER Client Connector Unit Test	
Test Case Reference ID	DC-04
Test Case Name	Check model existence
Component Name	DATA CONNECTOR
Description of the test case	
After requesting and downloading the model from the server, it is saved on the user-configured data volume; a check for the existence of the saved model on the file system is performed.	
Input of the test case	
Path where the model is to be saved.	
Output of the test case	
True if the model has been correctly saved in the defined path.	

Results of the test case
SUCCESS

Table 1 - Unit test results

Test Case Ref ID	Test Case Name	Component Name	Test Case Result
CM-01	*Check classpath configuration*	*Configuration Manager*	*SUCCESS*
CM-02	*Check module configuration*	*Configuration Manager*	*SUCCESS*
DC-03	*Check dataset existence*	*Data Connector*	*SUCCESS*
DC-04	*Check model existence*	*Data Connector*	*SUCCESS*

4.2 Integration testing

MUSKETEER Client Connector Integration Test	
Test Case Reference ID	#01
Test Case Name	Login
Components Involved	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*
Description of the test case	
The user can browse through the CC only after logging in with valid credentials.	
Input of the test case	
Valid and invalid credentials.	
Output of the test case	
- The user can manage their own resources after entering valid credentials. - The user cannot see and manage their own resources entering invalid credentials.	

Results of the test case
SUCCESS

Test Case Reference ID	#02
Test Case Name	Delete Task
Components Involved	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*
Description of the test case	
A user can delete a task only if he is the owner of it.	
Input of the test case	
- Request for deletion of an own task from UI; - Request for deletion of a non-proprietary task from UI.	
Output of the test case	
- User completes deletion of task successfully; - The user cannot complete the deletion of the task.	
Results of the test case	
SUCCESS	

MUSKETEER Client Connector Integration Test	
Test Case Reference ID	#03
Test Case Name	Libraries setup
Components Involved	*CC front-end – CC back-end – pycloudmessenger library – MML library*
Description of the test case	
The user can use the CC functionalities through the user interface after that he has configured the communication messenger and FML libraries.	
Input of the test case	

- FML library configuration; - Communication messenger library configuration.
Output of the test case
- The user can browse through the user interface only after that he has correctly configured both requested libraries.
Results of the test case
SUCCESS

MUSKETEER Client Connector Integration Test	
Test Case Reference ID	#04
Test Case Name	Result image chart
Components Involved	*CC front-end – CC back-end – MUSKETEER platform – pycloudmessenger library – MML library*
Description of the test case	
The user can see the image representing some evaluation metrics of the resulting model after the task completion.	
Input of the test case	
- Create and correctly run a task until completion.	
Output of the test case	
- Into the results location, initially configured by the user, contains result images of each completed task.	
Results of the test case	
SUCCESS	

MUSKETEER Client Connector Integration Test	
Test Case Reference ID	#05
Test Case Name	Task aggregation

Components Involved	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*
Description of the test case	
Only the task owner can join to task as aggregator.	
Input of the test case	
- Aggregate to a task from the user interface as task owner; - Join to the same task as participant with another user.	
Output of the test case	
- The user who owns the task correctly joins its task as an aggregator; a non-owner user can only join as a participant.	
Results of the test case	
SUCCESS	

MUSKETEER Client Connector Integration Test	
Test Case Reference ID	#06
Test Case Name	Task creation
Components Involved	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*
Description of the test case	
A user creates and publishes a task (aggregator), and another user (participants) sees it in the list of tasks and join it	
Input of the test case	
- A user creates a new task through the user interface.	
Output of the test case	
- A participant can correctly check and consult the created task and join it through the user interface.	
Results of the test case	
SUCCESS	

MUSKETEER Client Connector Integration Test	
Test Case Reference ID	#07
Test Case Name	Registration
Components Involved	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*
Description of the test case	
User registration to the MUSKETEER platform.	
Input of the test case	
- Username; - Organization name; - Password; - Confirm of the password; - A credential file imported during the communication messenger library configuration.	
Output of the test case	
- The user can register on the MUSKETEER platform and then using their own credentials to log in through the user interface.	
Results of the test case	
SUCCESS	

MUSKETEER Client Connector Integration Test	
Test Case Reference ID	#08
Test Case Name	Login - 2
Components Involved	*CC front-end – CC back-end – httpcloudmessenger library – TRUE Connector – Producer Data App*
Description of the test case	

A user access the target platform. The server side is simulated using a specific pre-configured producer Data App to provide the same output specifications as the MUSKETEER platform.
Input of the test case
- Username; - Organization name; - Password; - Confirm of the password; - A credential file imported during the communication messenger library configuration.
Output of the test case
A user access the target platform.
Results of the test case
SUCCESS

<table>
<tr><th colspan="2">MUSKETEER Client Connector Integration Test</th></tr>
<tr><td><b>Test Case Reference ID</b></td><td>#09</td></tr>
<tr><td><b>Test Case Name</b></td><td>Task listing</td></tr>
<tr><td><b>Components Involved</b></td><td><i>CC front-end – CC back-end – httpcloudmessenger library – TRUE Connector – Producer Data App</i></td></tr>
<tr><td colspan="2"><b>Description of the test case</b></td></tr>
<tr><td colspan="2">A logged in user can get the list of tasks store in the target platform. The server side is simulated using a specific pre-configured producer Data App to provide the same output specifications as the MUSKETEER platform.</td></tr>
<tr><td colspan="2"><b>Input of the test case</b></td></tr>
<tr><td colspan="2">The user requests the list of tasks.</td></tr>
<tr><td colspan="2"><b>Output of the test case</b></td></tr>
<tr><td colspan="2">The listing tasks successfully shown in the main page of the CC GUI.</td></tr>
<tr><td colspan="2"><b>Results of the test case</b></td></tr>
</table>

SUCCESS

Table 2 - Integration test results

Test Case Ref ID	Test Case Name	Components involved	Test Case Result
#01	*Login*	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*	*SUCCESS*
#02	*Delete task*	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*	*SUCCESS*
#03	*Libraries setup*	*CC front-end – CC back-end – pycloudmessenger library – MML library*	*SUCCESS*
#04	*Result image chart*	*CC front-end – CC back-end – MUSKETEER platform – pycloudmessenger library – MML library*	*SUCCESS*
#05	*Task aggregation*	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*	*SUCCESS*
#06	*Task creation*	*CC front-end – CC back-end – pycloudmessenger*	*SUCCESS*

		library – MUSKETEER platform	
#07	*Registration*	*CC front-end – CC back-end – pycloudmessenger library – MUSKETEER platform*	*SUCCESS*
#08	*Login - 2*	*CC front-end – CC back-end – httpcloudmessenger library – TRUE Connector – Producer Data App*	*SUCCESS*
#09	*Task listing*	*CC front-end – CC back-end – httpcloudmessenger library – TRUE Connector – Producer Data App*	*SUCCESS*

5 Conclusion

The purpose of this document *D7.4 – Final prototype of the MUSKETEER Client connectors*, is to explain the key components and the main user interactions with the final version of the Client Connector to exploit the MUSKETEER Platform functionalities.

The source code of the final prototype version of the MUSKETEER Client Connector is released as open source under GNU AGPLv3 license [2][3].

This final release is the result of several iterative executions of the demonstration scenarios, in accordance with the technical requirements specified in WP2.

As a follow-up, within D7.5 and D7.6 feedbacks on its usage in smart manufacturing and health domains, are under collection so to evaluate the overall MUSKETEER platform, by means of the KPIs Evaluation Framework defined and validated in WP2 (described in D2.7).

6 References

[1] https://internationaldataspaces.org/wp-content/uploads/IDS-RAM-3.0-2019.pdf

[2] https://github.com/Engineering-Research-and-Development/musketeer-client-connector-backend

[3] https://github.com/Engineering-Research-and-Development/musketeer-client-connector-frontend