Tailored IoT & BigData Sandboxes and Testbeds for Smart, Autonomous and Personalized Services in the European Finance and Insurance Services Ecosystem

Infinitech

D5.14 – Datasets for Algorithms Training & Evaluation - II

Revision Number	3.0
Task Reference	T5.1
Lead Beneficiary	UBI
Responsible	Konstantinos Perakis
Partners	AGRO ASSEN ATOS BOUN CP CTAG ENG FBK FTS GEN GFT INNOV ISPRINT JRC JSI PI PRIVE RB SIA UBI UPRC WEA
Deliverable Type	Report (R)
Dissemination Level	Public (PU)
Due Date	2021-12-31
Delivered Date	2021-12-22
Internal Reviewers	IBM, ASSEN
Quality Assurance	INNOV
Acceptance	WP Leader Accepted and Coordinator Accepted
EC Project Officer	Beatrice Plazzotta
Programme	HORIZON 2020 - ICT-11-2018
	This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

Contributing Partners

Partner Acronym	**Role**[1]	**Author(s)**[2]
UBI	Lead Beneficiary	Konstantinos Perakis, Dimitris Miltiadou
INNOV	Contributor	George Fatouros
PRIVE	Contributor	Anna Semeniuk
UPRC	Contributor	George Makridis
NBG	Contributor	Eleni Perdikouri
CXB	Contributor	Mario Maawad Marcos
FTS	Contributor	Juergen Neises, Jean Baptiste Rouquier
JSI	Contributor	Maja Skrjanc
BOUN	Contributor	Orkan Metin
PI	Contributor	Aschi Massimiliano
ATOS	Contributor	Jorge Mira Prats
ISPRINT	Contributor	Aristodemos Pnevmatikakis
WEA	Contributor	Carlos Albo Portero
AGRO	Contributor	Gregory Mygdakos
ABILAB	Contributor	Marco Rotoloni
IBM	Internal Reviewer	Fabiana Fournier
ASSEN	Internal Reviewer	Ilesh Dattani
INNOV	Quality Assurance	Filia Filippou

Revision History

Version	Date	Partner(s)	Description
0.1	2021-10-29	UBI	ToC Version
0.2	2021-11-10	UBI	Initial contributions on Section 2 and 3
0.3	2021-11-14	UBI	Contributions on Section 2 and 3
0.35	2021-11-16	UBI	Updated Contributions on Section 2 and 3
0.40	2021-11-17	NNOV, PRIVE, UPRC, NBG, CXB, FTS, JSI, BOUN, PI, ATOS, ISPRINT, WEA, AGRO, ABILAB	Contributions on Section 4
0.45	2021-11-23	UBI	Updated Contributions on Section 2 and 3
0.50	2021-11-30	INNOV, PRIVE, UPRC, NBG, CXB, FTS, JSI, BOUN, PI, ATOS, ISPRINT, WEA, AGRO, ABILAB	Updated contributions on Section 4

[1] Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

[2] Can be left void

0.60	2021-12-09	NNOV, PRIVE, UPRC, NBG, CXB, FTS, JSI, BOUN, PI, ATOS, ISPRINT, WEA, AGRO, ABILAB	Updated contributions on Section 4
0.65	2021-12-13	NNOV, PRIVE, UPRC, NBG, CXB, FTS, JSI, BOUN, PI, ATOS, ISPRINT, WEA, AGRO, ABILAB	Updated contributions on Section 4
0.70	2021-12-14	UBI	Finalisation of sections 2, 3, 4 and 5
1.0	2021-12-15	UBI	Initial Version for Internal Review
2.0	2021-12-20	UBI	Version for Quality Assurance
3.0	2021-12-22	UBI	Version for Submission

Executive Summary

The document at hand, entitled "D5.14 – Datasets for Algorithms Training & Evaluation - II" constitutes the final report of the efforts and the produced outcomes of Task T5.1 "Data Collection for Algorithms Training & Evaluation" of WP5. The purpose of this deliverable is threefold: a) to deliver the final and fully functional version of the **INFINITECH Data Collection component**, b) to present the **core characteristics** and the **role of synthetic datasets within INFINITECH** and c) to provide the updated documentation of the **list of real and synthetic datasets** which will be exploited within INFINITECH.

Hence, the scope of the deliverable at hand can be described in the following axes:

- To deliver the fully functional version of the **INFINITECH Data Collection component.** The specific component has been designed and implemented within the context of INFINITECH in order to address the difficulties and peculiarities of the data collection process. To this end, the final **high-level architecture of the component** is documented by presenting the component's three main modules, namely the **Data Retrieval**, the **Data Mapper** and the **Data Cleaner**. For each module, the deliverable presents the **final detailed design specifications** and the **final list of use cases** that are addressed in conjunction with the respective **sequence diagrams**. Finally, the deliverable provides the **final documentation of the implementation details of all three modules** with the help of UML diagrams as well as a **detailed presentation of the delivered solution** with a walkthrough from the user's perspective.
- To present the results of the comprehensive **analysis of the key characteristics of the synthetic datasets** as well as the definition of their role within INFINITECH. To this end, the deliverable presents the **main categories of synthetic datasets** and the methods utilised for the **synthetic data generation process** supplement with the toolset available for this process. Moreover, the deliverable presents the **use cases and motivation for the usage of synthetic datasets** and the **role within the INFINITECH project** as leveraged by the project's pilot. The analysis concludes with the list of synthetic datasets which are exploited by the INFINITECH pilots. It should be noted that the results remained unchanged from the previous iteration of the deliverable and they were reported for coherency reasons.
- To provide the updated **documentation of the datasets that are collected and utilised by the INFINITECH pilots**. The deliverable presents the supplementary **documentation of the list of the datasets**, real and synthetic, that are leveraged, updating the existing details of the datasets as presented in the previous iterations. For each dataset, the **details of their scope and content**, their **format** and the **pseudonymization or anonymisation requirements**, is documented.

The outcomes of this deliverable will drive the implementation activities of the rest of the tasks of WP5. The deliverable delivers the final implementation of the INFINITECH Data Collection component and documents the required updates and enhancements as introduced from M18 till M27. The current deliverable constitutes the final report of T5.1 and concludes the activities of this specific task.

Table of Contents

List of Figures

List of Tables

Abbreviations/Acronyms

Abbreviation	Definition
AI	Artificial Intelligence
AML	Anti-Money Laundering
API	Application Programmable Interface
BDA	Big Data Application
CSV	Comma Separated Values
CTF	Combating Terrorist Financing
EO	Earth Observation
ES	Expected Shortfall
FTP	File Transfer Protocol
HDFS	Hadoop Distributed File System
HPC	High-performance computing
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
LOCF	Last Observation Carried Forward
ML	Machine Learning
NOCB	Next Observation Carried Backward
ORC	Optimized Row Columnar
PDF	Portable Document Format
PNG	Portable Network Graphics
PoC	Proof of Concept
PSD2	Payment Service Directive 2
RA	Reference Architecture
RESTful	Representational State Transfer compliant
RWD	Real-World Data
SME	Small and Medium-Sized Enterprises
UI	User Interface
UML	Unified Modelling Language
VaR	Value-at-Risk
XML	Extensible Mark-up Language

1 Introduction

The scope of deliverable D5.14 “Datasets for Algorithms Training & Evaluation – II” is to document the efforts undertaken within the context of T5.1 “Data Collection for Algorithms Training & Evaluation” of WP5 from M18 till M27. In this context, the deliverable D5.14 constitutes the final iteration of the work performed under T5.1 and is prepared in accordance with the INFINITECH Description of Action on M27 of the project.

Datasets are key enablers for all Big Data based platforms in order to exploit its capabilities and offerings. Nevertheless, several steps are required before they can provide valuable results, insights and hidden knowledge as data need to be properly harmonised, annotated, cleaned, and stored. These tasks are considered as one of the most demanding and challenging areas of the entire data lifecycle which is referenced as the data collection aspects of the lifecycle [1]. Datasets are broadly categorised into two major categories, the real data which are produced by data sources and data providers and the synthetic data which are generated utilising multiple techniques and methodologies either to overcome the unavailability of real data or to overcome several data privacy restrictions and regulations imposed.

Datasets constitute one of the main ingredients of the INFINITECH platform and within the context of the INFINITECH project, both real and synthetic datasets are utilised by the project’s pilots. In order to be effectively collected and stored into the INFINITECH platform and the INFINITECH stakeholders to be able to exploit its offerings and added value, it is imperative that a flexible and efficient mechanism is provided. To this end, within the context of Task 5.1, the consortium has analysed the key requirements related to data collection in big data enabled platforms in order to formulate the design specifications of a data collection component which will be integrated and implemented as part of the INFINITECH reference architecture (RA) that is tailored to the needs of the finance and insurance sectors.

The presented solution enables the design and implementation of data collection pipelines which effectively address the needs for the data collection aspects of both the data providers of the INFINITECH project and the stakeholders of the financial and insurance sectors. In addition to this, the consortium exploits state-of-the-art approaches and methods in order to generate synthetic data in various scenarios of the financial and insurance sectors towards the implementation of innovative financial and insurance services.

Depending on the scope and target goals of the finance and insurance scenarios in the INFINITECH pilots, different approaches are followed for the data collection aspects. The core aspect of all these scenarios are the datasets which are collected and utilised within the scope of each pilot. It should be noted that the details of the data collection processes of the pilots are documented in deliverable D7.2.

1.1 Objective of the Deliverable

The purpose of the deliverable is to report the outcomes of the work performed within the context of Task 5.1 from M18 till M27 of the project. The deliverable reports the updates of the work performed during the second period of the task execution and it is mainly focused on the implementation and delivery of the final version of the INFINITECH Data Collection component, as well as of the data collection process designed and implemented within the context of each INFINITECH pilot and the final list of required datasets.

The first objective of the deliverable is to present the fully functional version of the INFINITECH Data Collection component. The design specifications of INFINITECH Data Collection component, as documented in the first iteration of the deliverable, is designed and implemented aiming to address the need for a holistic mechanism that will empower the data providers to configure and execute data collection pipelines tailored to their needs. The component is designed following a modular architecture composed by three distinct modules, namely the Data Retrieval, the Data Mapper and the Data Cleaner. The design specifications of each module are defined and described in detail along with the list of functionalities offered by each module. The design specifications are supplemented with a concrete list of use cases that each component addresses along with the respective sequence diagrams which depict the interactions of the stakeholders and the components. During the second iteration, these design specifications have driven the implementation activities of the component which was the main focus of this period. To this end, the deliverable documents the final implementation details of the INFINITECH Data Collection component by presenting the complete list of implemented functions, as well as the code structure of each module in the form of UML diagrams. Finally, the deliverable presents the implemented and delivered solution by providing a walkthrough of the delivered functionalities and by describing how they can be leveraged by the INFINITECH stakeholders.

The second objective of the deliverable is to present the role of the Synthetic Datasets and their usage within the context of INFINITECH. To this end, the results of the executed thorough analysis are presented. Within this analysis, the scope, categories as well as advantages and limitations are presented. The analysis presents the most common approaches followed for the generation of synthetic data along with a complete toolset composed of libraries, tools and framework that are utilised in the process. Finally, the analysis presents the relevance and added value of synthetic datasets for INFINITECH project presenting the use cases in the finance and insurance sectors for which synthetic data are utilised.

The third and final objective of the deliverable is to document the updated list of datasets that are exploited within the context of each pilot of INFINITECH. Towards this direction, the deliverable presents the list of datasets, real and synthetic ones, that each pilot utilises for its scenarios by documenting the details of their content, their format and the pseudo-anonymisation and anonymisation activities applied.

Deliverable D5.14 constitutes the second and final iteration of the deliverable, and as per the INFINITECH Description of Action, Task T5.1 lasts until M27. Therefore, this version of the deliverable constitutes the final report of the work performed within the scope of Task 5.1 and provides all the updates and enhancements that were introduced from the first iteration.

1.2 Insights from other Tasks and Deliverables

Deliverable D5.14 is released in the scope of WP5 “Data Analytics Enablers for Financial and Insurance Services” and documents the outcomes of the work performed within the context of T5.1 “Data Collection for Algorithms Training & Evaluation”. The task is directly related to the outcomes of WP2 “Vision and Specifications for Autonomous, Intelligent and Personalized Services” in which the overall requirements of the INFINITECH platform are defined. Task 5.1 received as input the outcomes of Task 2.1, in which the collected user stories of pilots of the project and the extracted user requirements were formulated and reported within deliverable D2.1 and D2.2. In addition to this, the outcomes of T2.3, in which one can find the fundamental building blocks of the INFINITECH platform and their specifications in terms of technologies, as well as the elicited technical requirements that are linked to these building blocks, as reported in deliverable D2.5 and D2.6, were provided as input to T5.1. Furthermore, the outcomes of T2.7, in which the INFINITECH Reference Architecture (INFINITECH RA) was formulated, were also provided as input to T5.1 and had driven the design and implementation aspects of the Data Collection component that constitutes a core part of the INFINITECH platform. Finally, the work reported in this deliverable is tightly interconnected with the work performed in the rest of the tasks of WP5, namely T5.2, T5.3, T5.4 and T5.5, as it covers the required data collection aspects for the datasets that will be used on all the aforementioned tasks.

1.3 Structure

This document is structured as follows:

- Section **Error! Reference source not found.** introduces the document, describing the context of the outcomes of the work performed within the task and highlights its relation to the rest of tasks of the project and deliverables of the project.
- Section 2 documents the design specifications, the use case addressed along with relevant sequence diagrams, the implementation details of the INFINITECH Data Collection component as well as a presentation of the implemented solution from the user’s perspective.
- Section 3 presents the key characteristics, advantages and limitations of the synthetic datasets, the details of the synthetic data generation process and how they will be leveraged in INFINITECH.

- Section 4 presents the details of the datasets that are collected from the INFINITECH pilots in order to execute their designed use cases. Finally,
- Section 5 concludes the document.

2 INFINITECH Data Collection

Updates from D5.13:
The particular section documents the necessary updates related to the advancements on the implementation of the INFINITECH Data Collection component. In detail, the following updates are introduced in the section (while the rest of sub-sections remain unchanged from the previous version):

- The updated implementation details of the INFINITECH Data Collection where all the implemented functions and services are documented (sub-section 2.3)
- The presentation of the delivered fully functional INFINITECH Data Collection with a walkthrough to its functionalities from the user's perspective (sub-section 2.4)

2.1 Design Specifications

Updates from D5.13:
The specific sub-section remained unchanged from previous version.

Data Ingestion typically involves all the processes and operations that are performed for the gathering or retrieval of data from different data sources or data providers, the correlation and annotation of the included data entities with several ontologies and semantics and finally the cleaning of the data with multiple data cleaning operations before they are stored in the underlying storage solution of the system or the data warehouse. In the big data era, where a tremendous amount of diverse information is generated by a plethora of data sources, systems, devices, and platforms, data ingestion becomes a crucial part of any designed solution. However, at the same time data ingestion becomes rather challenging not only because of the volume of data that grows exponentially, but also due to the nature of data sources that generate data in a large variety of formats and at different velocities.

Towards this end, the INFINITECH Data Collection component was designed aiming to provide an abstract and holistic mechanism for the data providers that will address the various connectivity and communication challenges with the variety of data sources that are exploited in the finance and insurance sector, as well as the peculiarities/specificities of the various data providers of the specific sectors. Hence, the scope of the Data Ingestion mechanism is threefold:

a) to enable the acquisition and retrieval of heterogeneous data from diverse data sources and data providers,
b) to facilitate the mapping of the entities included in the data to the corresponding entities of an underlying data model towards the data annotation and
c) to enable the data cleaning operations that will address the data quality issues of the acquired data.

The INFINITECH Data Collection component is composed of three main modules:

a) the Data Retrieval that undertakes the responsibility to retrieve or receive the new datasets from a data source or data provider,
b) the Data Mapper that is responsible for the generation of the mapping between the entities of retrieved or received dataset and the ones of an underlying data model entities based on the data provider's input and
c) the Data Cleaner that undertakes the responsibility to perform the data cleaning operations on the retrieved or received dataset, again based on the data provider's input.

Figure 1 depicts the high-level architecture of the INFINITECH Data Collection. As illustrated, the INFINITECH Data Collection component is capable of retrieving or fetching new datasets from a variety of data sources and to receive new datasets via its exposed RESTful APIs. The newly acquired dataset is fed in the process by the Data Retrieval in order to be provided as input to the Data Mapper. The Data Mapper performs the data mapping operations and generates the mapping between the data entities of the newly acquired dataset and the entities of an underlying data model, that is provided by the data provider, based on the input of the data provider. In the final step, the newly acquired dataset is provided as input to the Data Cleaner in which the data cleaning operations are performed based on the input of the data provider in order to provide the "cleaned data".

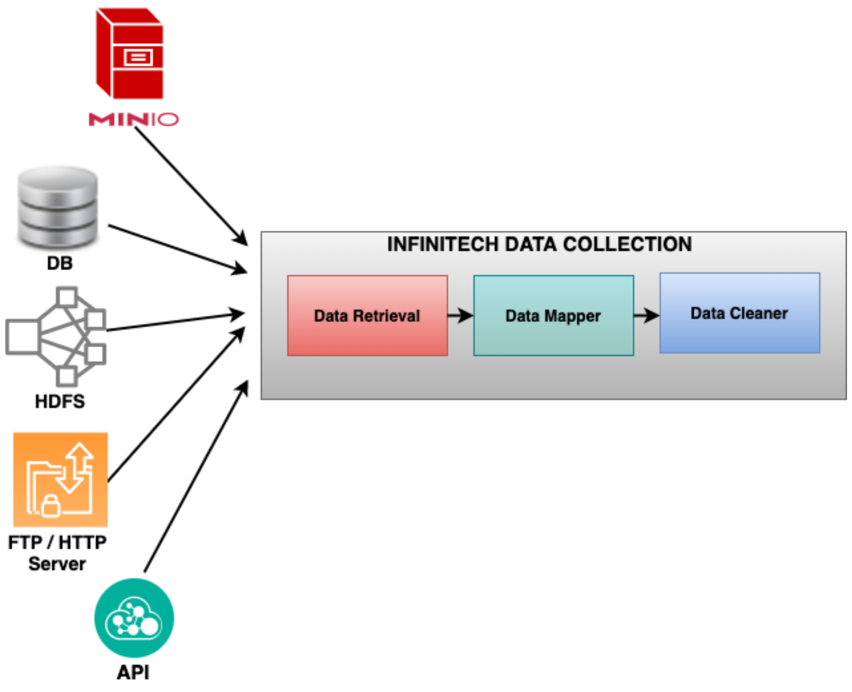


Figure 1: High-level architecture of the INFINITECH Data Collection

In the following subsections, the functionalities of the three main modules are presented in detail.

2.1.1 Data Retrieval

The scope of the Data Retrieval module is to facilitate the data acquisition from any relational database, HDFS deployments, FTP or HTTP servers, MinIO storage servers as well as from any API of the data source. The prerequisite is that the appropriate information is provided by the data provider to the process prior to its invocation. Additionally, the Data Retrieval module enables the reception of new datasets that are pushed from the data provider to its exposed RESTful APIs. Hence, the Data Retrieval module supports all the aforementioned data source types which are considered the most commonly used data source types in every Big Data ecosystem. Nevertheless, the modular architecture of the described solution facilitates the expansion of the list of supported data source types in an effortless and effective manner upon the needs of the data providers.

The process is able to retrieve new information either periodically or on-demand and can be automated as a background process using the concept of data source profiles that are configured by the data provider based on their needs.

Hence, the main functionalities of the Data Retrieval module are as follows:

a) The retrieval of new dataset from a data source by establishing the required connection to the data source's API and pulling the provided information based on the provided configuration
b) The receival (pushing) of new dataset from a data source and local storage of the received data through well-defined RESTful APIs
c) The retrieval and fetching of files from an FTP or HTTP server by establishing the required connection and downloading the files locally
d) The data retrieval of new data from Relational Databases by establishing the appropriate connection and executing the appropriate query based on the provided configuration

e) The retrieval and fetching of files from an HDFS deployment or MinIO storage server by establishing the required connection and fetching the files locally
f) The creation of a data source profile that contains all the relevant information of properly connecting and retrieving new datasets from a specific data source which enables the automation of the data retrieval process
g) The periodic (in preconfigured time intervals) or immediate and on-demand retrieval of new datasets based on the provided configuration in the data source profile.

2.1.2 Data Mapper

The scope of the Data Mapper module is to enable the mapping of the data entities included in a new dataset and the data model that is provided by the data provider. In this sense, the data provider is able to create the mappings for each entity of the new dataset to a specific data entity of the data model. To achieve this, at first the Data Mapper module offers the means to integrate a data model during its initial configuration. Then, during processing, the data entities of the provided dataset are extracted and displayed to the data provider via its user friendly and easy-to-use user interface. Through this user interface the data provider is able to select the corresponding entities of the integrated data model that will be mapped to the entities of the dataset. The generated mappings are stored for later reuse in a JavaScript Object Notation (JSON) format.

The whole process can be performed as a background process also, if the data provider creates the corresponding data mapping profile for a specific dataset beforehand, which will be used in an automated way for the execution of the process, when a new dataset for the specific profile is received.

Hence, the main functionalities of the Data Mapper module, are as follows:

a) The integration of data model that is provided by the data provider during the module configuration
b) The extraction of the data entities of the input dataset which are represented to the data provider
c) The display of the entities of the integrated data model
d) The creation of the mapping between the data entities of the input dataset and the entities of the integrated data model based on the data provider selection
e) The generation and storage of the produced mapping in JSON format
f) The creation of a data mapping profile that contains all the relevant information of generating the mapping in the form of mapping rules between the entities of the input dataset and the entities of the integrated data model
g) The automated or on-demand execution via API of the mapping process in the case of existence of data mapping profile.

It should be noted at this point that the development of a data model is out of scope of T5.1 and WP5 and that it is the responsibility of the data provider that exploits the Data Mapper module to provide it. However, the Data Mapper provides the means to fetch, interpret, and integrate the provided data model in the mapping process via the module's internal configuration. In particular, the stakeholder of the Data Mapper module is able to define the path of the provided data model and the Data Mapper module will interpret the provided data model during the module's start-up operation.

2.1.3 Data Cleaner

The scope of the Data Cleaner module is to provide the data cleaning operations that will ensure that the provided input datasets, that are originating from a variety of heterogeneous data sources, are clean and complete to the extent possible. The specific functionalities enable the detection and correction (or removal) of inaccurate, corrupted or incomplete values in the data entities of the datasets towards the increase of the data quality of the specific data, as well as its value during data processing. To this end, the Data Cleaning module's operations are performed on top of the input dataset in order to produce the "cleaned" results.

Under the hood, the data cleaning process is a four-step process that includes:

a) the validation of the values of the data entities against a set of constraints,
b) the correction of the errors identified based on a set of data correction operations,
c) the data completion of the values for the required/mandatory data entities with missing values with a set of data completion operations and
d) the maintenance of complete history records containing the history of errors identified and the data cleaning operations that were performed to address them.

The cleaning process is based on the set of data cleansing rules that are defined by the data provider on a data entity level via the Data Cleaner's user friendly and easy-to-use user interface and include the data validation rules, as well as the data correction and data completion actions performed for these rules.

In the same manner as with the Data Mapper, the Data Cleaner has been designed in a way that enables the execution of the process as a background process with the only prerequisite being the creation of a data cleaning profile for a specific dataset beforehand that will be leveraged in order to automated the process execution upon the receival of a new dataset for which the corresponding data cleaning profile can be applied.

To this end, the main functionalities of the Data Cleaner module are as follows:

a) The extraction of the data entities of the input dataset which are represented to the data provider
b) The definition of data cleaning rules for each extracted data entity that include the validation of the values of the specific data entity against a set of constraints, the desired data correction operation or the missing value handling operation
c) The execution of the data cleaning operations on the input dataset based on the set of data cleaning rules defined by the data provider towards the generation of the "cleaned" dataset
d) The creation of a data cleaning profile that contains all the data cleaning rules of the specific dataset
e) The automated or on-demand execution via API of the cleaning process in the case of existence of data cleaning profile.

2.2 Use Cases and Sequence Diagrams

Updates from D5.13:
The specific sub-section remained unchanged from previous version.

As explained in Section 2.1, the INFINITECH Data Collection component is composed of three main modules, namely the Data Retrieval, the Data Mapper and the Data Cleaner. In this section, the detailed documentation of all the use cases encapsulated on each module are presented describing in detail all the information of each use case. Furthermore, for each use case the corresponding sequence diagram that depicts the interactions of the involved stakeholders and the involved components are also presented.

2.2.1 Data Retrieval

2.2.1.1 API to pull information

The specific functionality undertakes the responsibility of retrieving (pulling) information from a specific data source provider either upon the triggering of the process (on-demand pull) or in a predefined time interval (scheduled pull). To initiate the process, the mechanism of the Data Retriever requires the definition of the configuration (profile) of the specific data source. The configuration contains all the connections details required in order to properly access and retrieve the required information in the form of JSON file from a data source's APIs. The acquired information is locally stored in order to be used in the next steps of the process.

Table 1: Definition of the Data Source Profile (API)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of an accessible API capable of providing the requested information upon the successful connection and request 2. The existence of the configuration details of the respective API endpoint
Post-conditions:	1. The data source profile is available
Data Attributes	1. Data Source Name 2. Data Source Owner 3. Data Source Type = API 4. API URL path 5. Authentication Details (Username/Password, JWT, etc) 6. Pull time interval (None or number of seconds, minutes, days)
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to register the new data source profile 1. An acknowledgement is returned and the data source profile is stored in the list of known data sources
Pass Metrics	1. The new data source profile is available in the list of known sources for the data retrieval process
Fail Metrics	1. The new data source profile is not available in the list of known data sources

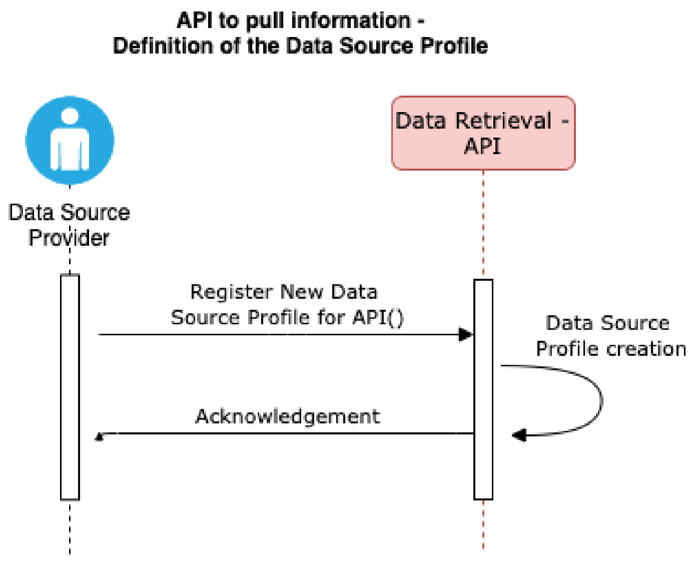

API to pull information -
Definition of the Data Source Profile
Data Source Provider
Data Retrieval - API
Register New Data Source Profile for API()
Data Source Profile creation
Acknowledgement

Figure 2: API to pull information – Definition of the data source profile (API)

Upon the successful creation of the data source profile in the previous step, the data retrieval via an API of the data provider can be triggered on-demand. In this case, the Data Retrieval component will initiate a new pull request to the data provider's API, utilising the information included in the data source profile and the new dataset will be retrieved and stored locally for further processing.

Table 2: API Data Retrieval (on-demand)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of the data source profile
Post-conditions:	1. New information has been retrieved and stored locally for further processing
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to retrieve new information based on a specific data source profile 2. The Data Retrieval initiates the request to the respective API and retrieves the new information 3. The new information is stored locally in a JSON format
Pass Metrics	1. The new information is available locally as a JSON format file
Fail Metrics	1. The request is rejected and no information is retrieved from the API of the data source profile

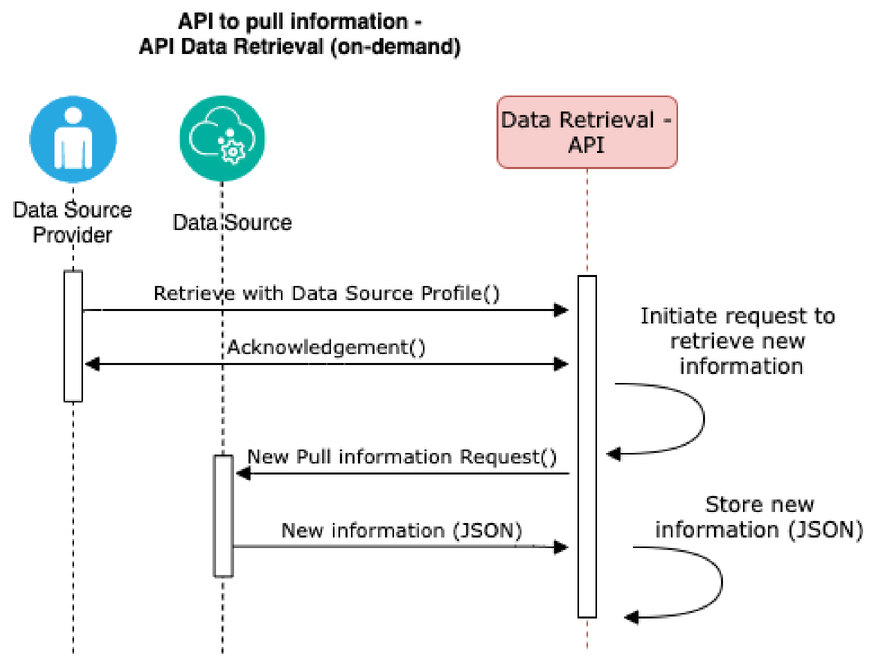

API to pull information -
API Data Retrieval (on-demand)
Data Source Provider
Data Source
Data Retrieval - API
Retrieve with Data Source Profile()
Acknowledgement()
Initiate request to retrieve new information
New Pull information Request()
New information (JSON)
Store new information (JSON)

Figure 3: API Data Retrieval (on-demand)

An alternative way is to preconfigure the process via the respective information in the data source profile to be executed in predefined time intervals (scheduled pull). In this case, the Data Retrieval component will initiate a new pull request to the data provider's API based on the schedule defined in the data source profile and the new dataset will be retrieved and stored locally for further processing.

Table 3: API Data Retrieval (scheduled)

Stakeholders involved:	N/A
Pre-conditions:	1. The existence of the data source profile 2. The Pull time interval is set in the respective data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing based on the configured pull time interval
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The Data Retrieval initiates the request to the respective API and retrieves the new information based on a specific data source profile and the configured pull time interval 2. The new information is stored locally in a JSON format
Pass Metrics	1. The new information is available locally as a JSON format file
Fail Metrics	1. The request is rejected and no information is retrieved from the API of the data source profile

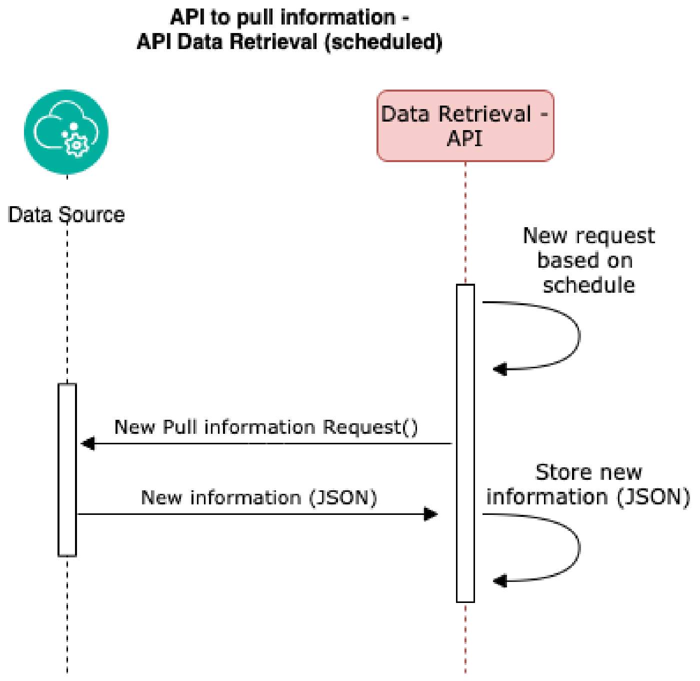

API to pull information -
API Data Retrieval (scheduled)
Data Source
Data Retrieval - API
New request based on schedule
New Pull information Request()
New information (JSON)
Store new information (JSON)

Figure 4: API Data Retrieval (scheduled)

2.2.1.2 API to push information

An alternative option for the data provider to ingest new datasets is the utilisation of the RESTful API that is provided by the Data Retrieval module. In detail, the data provider initiates a request to the Data Retrieval APIs to push new information providing the required information and the new dataset in the form of an attachment file or in the request body in the JSON format. The Data Retrieval module processes the request and stores the received file locally for further processing.

Table 4: API to push information

Stakeholders involved:	Data Source Provider
Pre-conditions:	N/A
Post-conditions:	1. New information is pushed to the Data Retrieval and stored locally for further processing
Data Attributes	1. Data Type (CSV, JSON, XML) 2. Data Source Name 3. Data Source Owner 4. Authorisation Token (JWT) 5. Data Source File (included as an attachment or in the request body)
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to push new information 2. The Data Retrieval's retrieves the new information stores it locally in the provided file format for further processing
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	1. The request is rejected and no information is stored locally

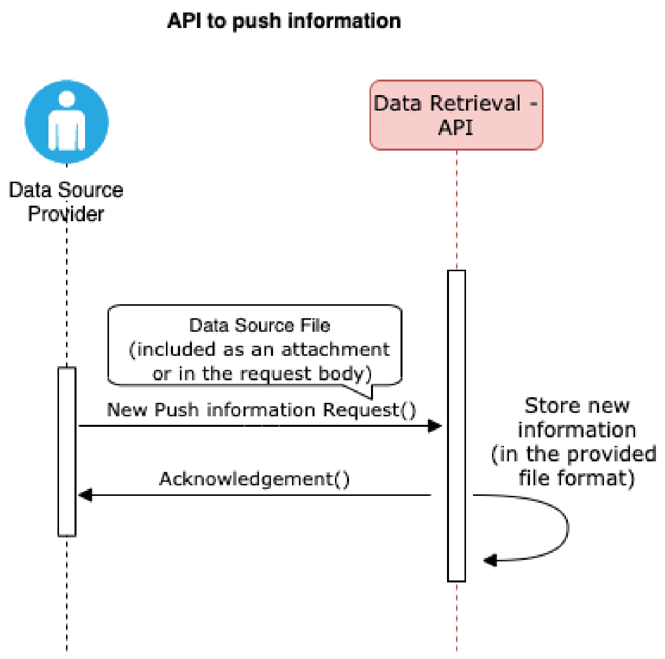

API to push information
Data Source Provider
Data Retrieval - API
Data Source File (included as an attachment or in the request body)
New Push information Request()
Acknowledgement()
Store new information (in the provided file format)

Figure 5: API to push information

2.2.1.3 Fetch files from an FTP / HTTP server

The specific functionality undertakes the responsibility of retrieving (pulling) files from a specific data source provider's FTP or HTTP server, either upon the triggering of the process (on-demand pull) or in a predefined time interval (scheduled pull). To initiate the process, the mechanism of the Data Retriever requires the definition of the configuration (profile) of the FTP or HTTP server. The configuration contains all the connection details required in order to properly access and retrieve the required file from the data source provider's FTP or HTTP server. The acquired information is locally stored in order to be used in the next steps of the process.

Table 5: Definition of the Data Source Profile (FTP or HTTP)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of an accessible FTP or HTTP server capable of providing the requested information upon the successful connection and request 2. The existence of the configuration details of the respective FTP/HTTP server
Post-conditions:	1. The data source profile is available
Data Attributes	1. Data Source Name 2. Data Source Owner 3. Data Source Type = FTP or HTTP 4. File path 5. Connection URL 6. Connection Port 7. Connection Username 8. Connection Password 9. Pull time interval (None or number of seconds, minutes, days)
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to register the new data source profile 2. An acknowledgement is returned and the data source profile is stored in the list of known data sources
Pass Metrics	1. The new data source profile is available in the list of known sources for the data retrieval process
Fail Metrics	1. The new data source profile is not available in the list of known data sources

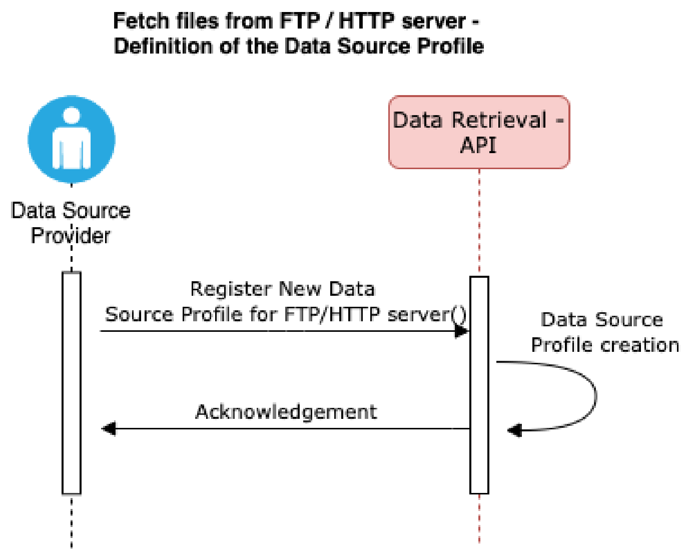

Fetch files from FTP / HTTP server -
Definition of the Data Source Profile
Data Source Provider
Data Retrieval - API
Register New Data Source Profile for FTP/HTTP server()
Data Source Profile creation
Acknowledgement

Figure 6: Definition of the Data Source Profile (FTP or HTTP)

Upon the successful creation of the data source profile in the previous step, the data retrieval process can be triggered via the API of Data Retrieval from the data provider on-demand. In this case, the Data Retrieval module will initiate a new request to the respective FTP or HTTP, utilising the information included in the data source profile and the new file will be retrieved and stored locally for further processing.

Table 6: FTP or HTTP Data Retrieval (on-demand)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of the data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to retrieve new information from the FTP or HTTP server based on a specific data source profile 2. The Data Retrieval initiates the request to the respective server and retrieves the new information 3. The new information is stored locally in the provided file format
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	1. The request is rejected and no information is retrieved from the server defined in the data source profile

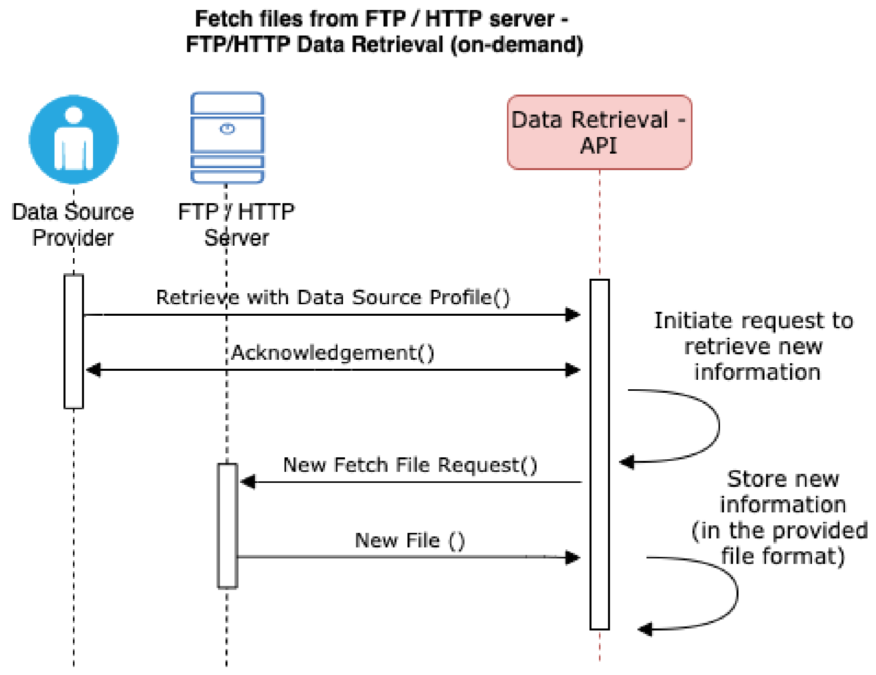

Fetch files from FTP / HTTP server -
FTP/HTTP Data Retrieval (on-demand)
Data Source Provider
FTP / HTTP Server
Data Retrieval - API
Retrieve with Data Source Profile()
Acknowledgement()
Initiate request to retrieve new information
New Fetch File Request()
New File ()
Store new information (in the provided file format)

Figure 7: FTP or HTTP Data Retrieval (on-demand)

An alternative option provided is to preschedule the execution of the process via the respective information in the data source profile in order to be executed in predefined time intervals (scheduled pull). In this case, the Data Retrieval module will initiate a new request to the respective FTP or HTTP based on the schedule defined in the data source profile and the new file will be retrieved and stored locally for further processing.

Table 7: FTP or HTTP Data Retrieval (scheduled)

Stakeholders involved:	N/A
Pre-conditions:	1. The existence of the data source profile 2. The Pull time interval is set in the respective data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing based on the configured pull time interval
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The Data Retrieval initiates the request to the respective server and retrieves the new information based on a specific data source profile and the configured pull time interval 2. The new information is stored locally in the provided file format
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	1. The request is rejected and no information is retrieved from the server defined in the data source profile

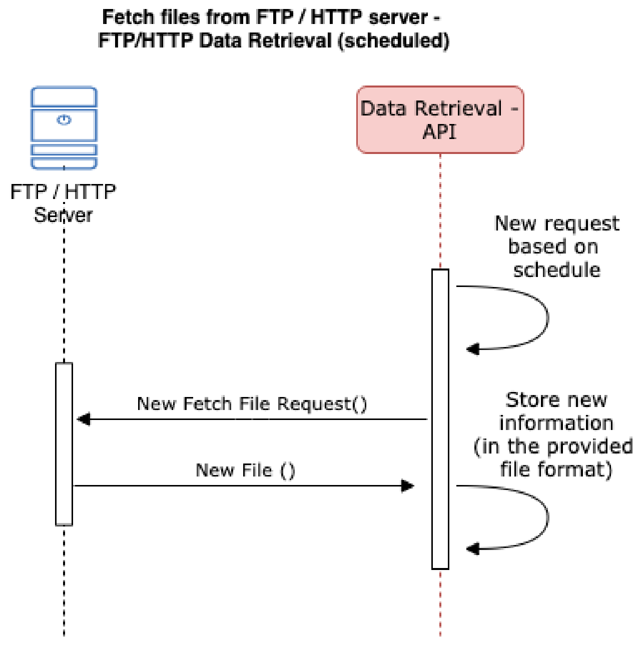

Fetch files from FTP / HTTP server -
FTP/HTTP Data Retrieval (scheduled)
Data Retrieval - API
FTP / HTTP Server
New request based on schedule
New Fetch File Request()
New File ()
Store new information (in the provided file format)

Figure 8: FTP or HTTP Data Retrieval (scheduled)

2.2.1.4 Relational Databases

This functionality undertakes the responsibility of retrieving new data from relational databases by establishing the appropriate connection and executing the appropriate query based on the provided configuration. The operation can be executed either upon the triggering of the process (on-demand) or in a predefined time interval (scheduled). To initiate the process, the mechanism of the Data Retriever requires the definition of the configuration (profile) of the specific relational database, containing all the connection details required in order to properly access and retrieve the required information from the respective database. The acquired information is locally stored in order to be used in the next steps of the process.

Table 8: Definition of the Data Source Profile (DB)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of an accessible SQL Database capable of providing the requested information upon the successful connection and request 2. The existence of the configuration details of the respective DB deployment
Post-conditions:	1. The data source profile is available
Data Attributes	1. Data Source Name 2. Data Source Owner 3. Data Source Type = DB 4. DB name 5. Executed DB Query 6. Connection URL 7. Connection Port 8. Connection Username 9. Connection Password 10. Pull time interval (None or number of seconds, minutes, days)

Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to register the new data source profile 2. An acknowledgement is returned and the data source profile is stored in the list of known data sources
Pass Metrics	1. The new data source profile is available in the list of known sources for the data retrieval process
Fail Metrics	1. The new data source profile is not available in the list of known data sources

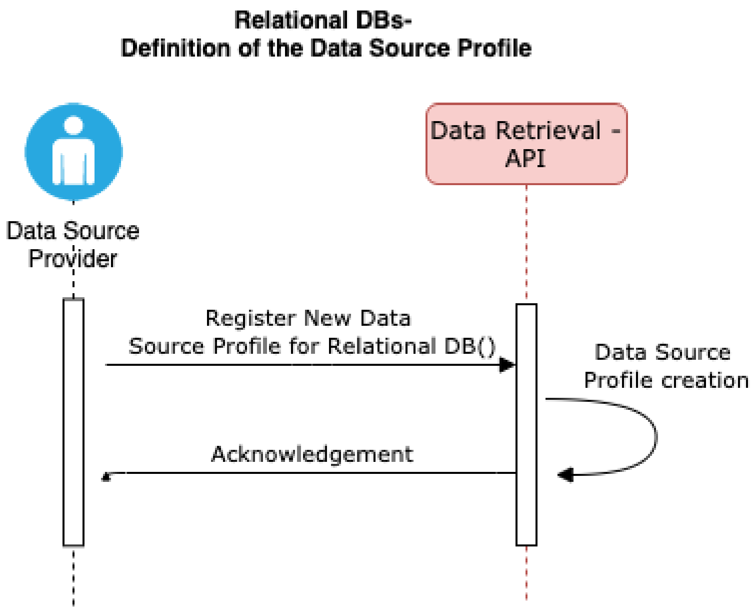

Relational DBs-
Definition of the Data Source Profile
Data Source Provider
Data Retrieval - API
Register New Data Source Profile for Relational DB()
Data Source Profile creation
Acknowledgement

Figure 9: Definition of the Data Source Profile (DB)

Upon the successful creation of the data source profile in the previous step, the data retrieval process can be triggered via the API of Data Retrieval from the data provider on-demand. In this case, the Data Retrieval module will initiate a new connection to the respective relational database utilising the information included in the data source profile, execute the defined query and retrieve the results and store them locally for further processing.

Table 9: DB Data Retrieval (on-demand)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of the data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to retrieve new information from the defined DB based on a specific data source profile

	2. The Data Retrieval initiates the request to the respective DB and retrieves the new information 3. The new information is stored locally in the provided file format
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	2. The request is rejected and no information is retrieved from the server defined in the data source profile

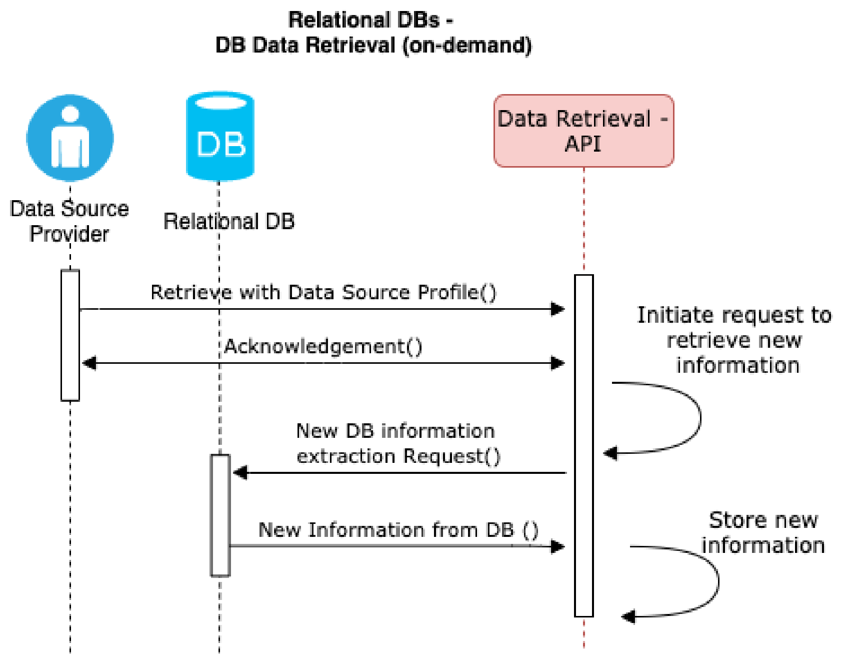

Relational DBs -
DB Data Retrieval (on-demand)
DB
Data Source Provider
Relational DB
Data Retrieval - API
Retrieve with Data Source Profile()
Acknowledgement()
Initiate request to retrieve new information
New DB information extraction Request()
New Information from DB ()
Store new information

Figure 10: DB Data Retrieval (on-demand)

In the same manner as with the rest of the options, the execution of the process can be prescheduled by setting the respective information in the data source profile, so as to be executed in predefined time intervals (scheduled pull). In this case, the Data Retrieval module will initiate a new connection to the respective relational database based on the schedule defined in the data source profile, execute the predefined query, retrieve the results and store them locally for further processing.

Table 10: DB Data Retrieval (scheduled)

Stakeholders involved:	N/A
Pre-conditions:	1. The existence of the data source profile 2. The Pull time interval is set in the respective data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing based on the configured pull time interval
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The Data Retrieval initiates the request to the respective DB and retrieves the new information based on a specific data source profile and the configured pull time interval

	2. The new information is stored locally in the provided file format
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	1. The request is rejected and no information is retrieved from the server defined in the data source profile

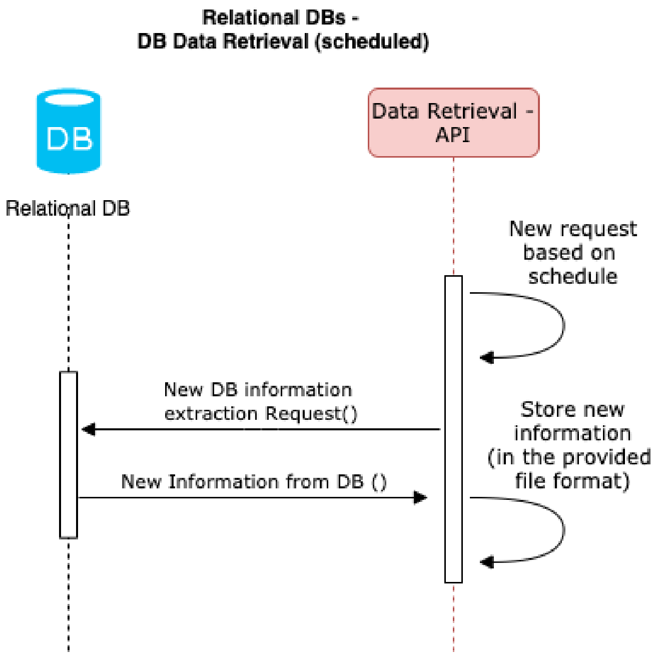

Relational DBs -
DB Data Retrieval (scheduled)
DB
Relational DB
Data Retrieval - API
New request based on schedule
New DB information extraction Request()
New Information from DB ()
Store new information (in the provided file format)

Figure 11: DB Data Retrieval (scheduled)

2.2.1.5 Retrieve files from HDFS

The specific functionality undertakes the responsibility of retrieving (pulling) files from a specific data source provider's HDFS deployment, either upon the triggering of the process (on-demand pull) or in a predefined time interval (scheduled pull). To initiate the process, the mechanism of the Data Retriever requires the definition of the configuration (profile) of the HDFS. The configuration contains all the connection details required in order to properly access and retrieve the required file from the data source provider's HDFS deployment. The acquired information is locally stored in order to be used in the next steps of the process.

Table 11: Definition of the Data Source Profile (HDFS):

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of an accessible HDFS deployment capable of providing the requested information upon the successful connection and request 2. The existence of the configuration details of the respective HDFS deployment
Post-conditions:	1. The data source profile is available
Data Attributes	1. Data Source Name 2. Data Source Owner 3. Data Source Type = HDFS

	4. HDFS File Path 5. Connection URL 6. Connection Port 7. Authentication details 8. Pull time interval (None or number of seconds, minutes, days)
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to register the new data source profile 2. An acknowledgement is returned and the data source profile is stored in the list of known data sources
Pass Metrics	1. The new data source profile is available in the list of known sources for the data retrieval process
Fail Metrics	1. The new data source profile is not available in the list of known data sources

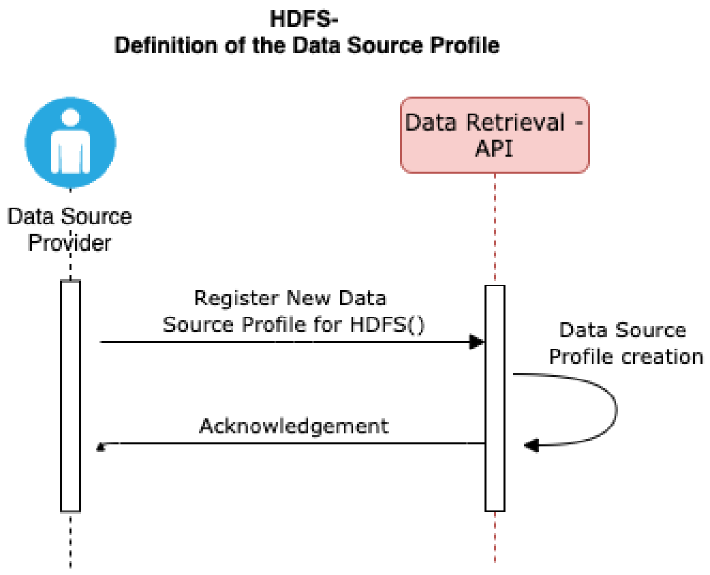

HDFS-
Definition of the Data Source Profile
Data Source Provider
Data Retrieval - API
Register New Data Source Profile for HDFS()
Data Source Profile creation
Acknowledgement

Figure 12: Definition of the Data Source Profile (HDFS)

Upon the successful creation of the data source profile in the previous step, the data retrieval process can be triggered via the API of Data Retrieval from the data provider on-demand. In this case, the Data Retrieval component will initiate a new request to the respective HDFS deployment, utilising the information included in the data source profile and the new file will be retrieved and stored locally for further processing.

Table 12: HDFS Data Retrieval (on-demand)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of the data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing
Data Attributes	1. Data Source Profile ID

Normal Flow	4. The data source provider initiates a request to the Data Retrieval's API endpoint to retrieve new information from the defined HDFS deployment based on a specific data source profile 5. The Data Retrieval initiates the request to the respective HDFS deployment and retrieves the new information 6. The new information is stored locally in the provided file format
Pass Metrics	2. The new information is available locally in the provided file format
Fail Metrics	3. The request is rejected and no information is retrieved from the server defined in the data source profile

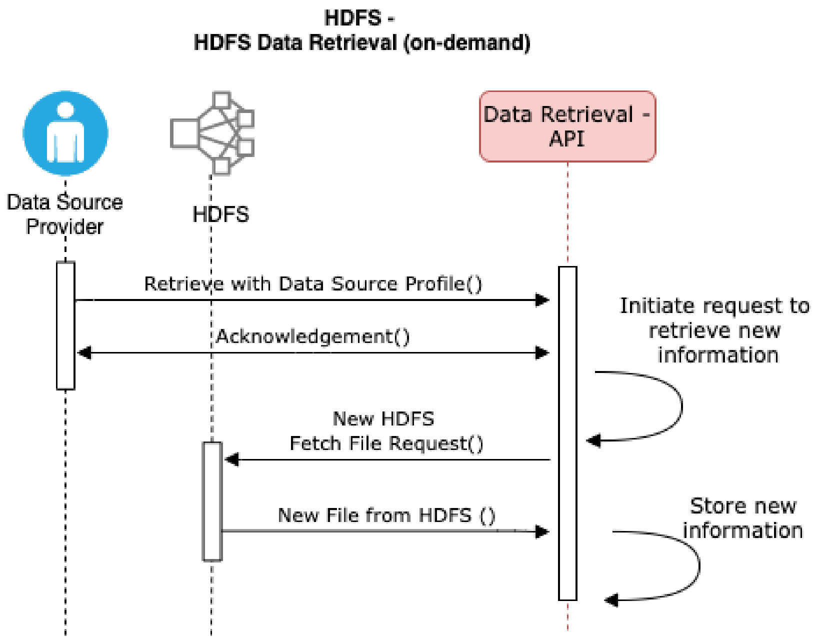

HDFS -
HDFS Data Retrieval (on-demand)
Data Source Provider
HDFS
Data Retrieval - API
Retrieve with Data Source Profile()
Acknowledgement()
Initiate request to retrieve new information
New HDFS Fetch File Request()
New File from HDFS ()
Store new information

Figure 13: HDFS Data Retrieval (on-demand)

On the other hand, the execution of the process can be executed in predefined time intervals (scheduled pull). In this case, the Data Retrieval module will initiate a new connection to the HDFS deployment based on the schedule defined in the data source profile, retrieve the new file and store it locally for further processing.

Table 13: HDFS Data Retrieval (scheduled)

Stakeholders involved:	N/A
Pre-conditions:	1. The existence of the data source profile 2. The Pull time interval is set in the respective data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing based on the configured pull time interval
Data Attributes	1. Data Source Profile ID

Normal Flow	1. The Data Retrieval initiates the request to the respective HFDS deployment and retrieves the new information based on a specific data source profile and the configured pull time interval 2. The new information is stored locally in the provided file format
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	1. The request is rejected and no information is retrieved from the server defined in the data source profile

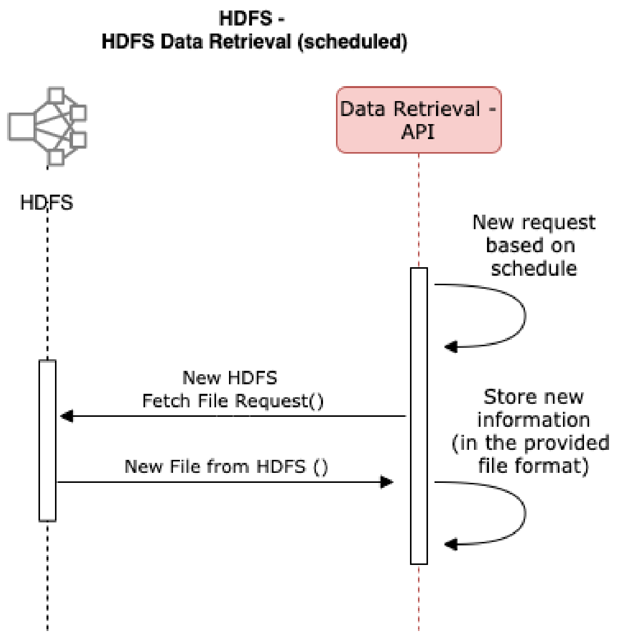

HDFS -
HDFS Data Retrieval (scheduled)
HDFS
Data Retrieval - API
New request based on schedule
New HDFS Fetch File Request()
New File from HDFS ()
Store new information (in the provided file format)

Figure 14: HDFS Data Retrieval (scheduled)

2.2.1.6 Retrieve files from MinIO

The specific functionality undertakes the responsibility of retrieving (pulling) files from a specific data source provider's MinIO storage server. The process can be triggered at any time (on-demand pull) or automatically in a predefined time interval (scheduled pull). As with the rest of the cases, Data Retriever requires the definition of the configuration (profile) of the MinIO storage server that should contain all the required information (connection details), in order to be able to establish a connection to the storage server and retrieve the required file. The retrieved file is locally stored and is fed to the Data Mapper or Data Cleaner for further processing.

Table 14: Definition of the Data Source Profile (MinIO):

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of an accessible MinIO storage server capable of providing the requested information upon the successful connection and request 2. The existence of the configuration details of the respective MinIO storage server
Post-conditions:	1. The data source profile is available

Data Attributes	1. Data Source Name 2. Data Source Owner 3. Data Source Type = MinIO 4. Bucket Name 5. File Path 6. Endpoint 7. Access Key 8. Access Secret 9. Pull time interval (None or number of seconds, minutes, days)
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to register the new data source profile 1. An acknowledgement is returned and the data source profile is stored in the list of known data sources
Pass Metrics	1. The new data source profile is available in the list of known sources for the data retrieval process
Fail Metrics	1. The new data source profile is not available in the list of known data sources

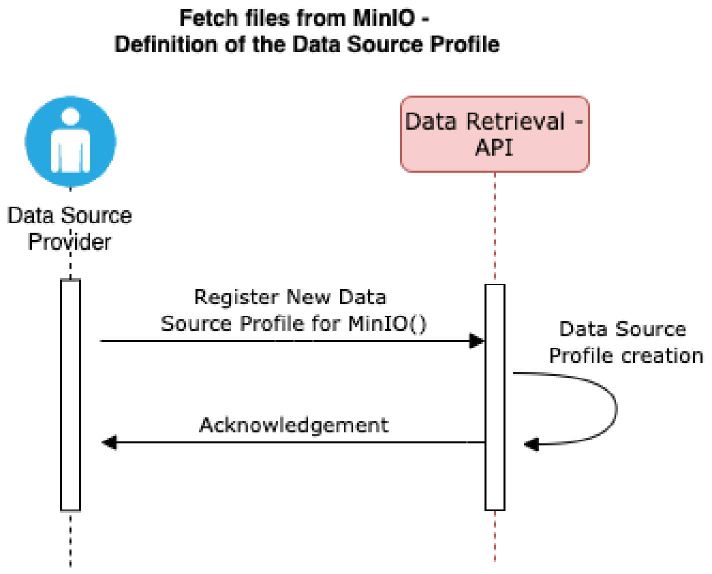

Fetch files from MinIO -
Definition of the Data Source Profile
Data Source Provider
Data Retrieval - API
Register New Data Source Profile for MinIO()
Data Source Profile creation
Acknowledgement

Figure 15: Definition of the Data Source Profile (MinIO)

Once the data source profile has been successfully registered in accordance with the previous step, the data retrieval process can be initiated by utilising the API of Data Retrieval and the on-demand retrieval of the requested file is triggered. Under the hood, the Data Retrieval module will initiate and establish a connection to the specified MinIO storage server, based on the information included in the data source profile and pull locally the requested file.

Table 15: MinIO Data Retrieval (on-demand)

Stakeholders involved:	Data Source Provider

Pre-conditions:	1. The existence of the data source profile
Post-conditions:	1. New information is retrieved and stored locally for further processing
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The data source provider initiates a request to the Data Retrieval's API endpoint to retrieve new information from the specific MinIO storage server based on a specific data source profile 2. The Data Retrieval initiates the request to the respective MinIO storage server and retrieves the new information 3. The new information is stored locally in the provided file format
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	1. The request is rejected and no information is retrieved from the server defined in the data source profile

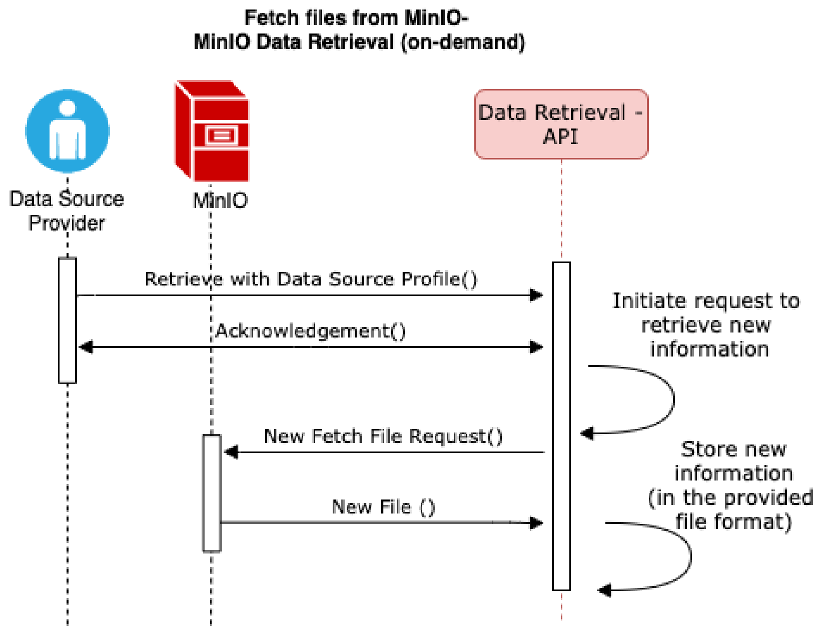

Fetch files from MinIO-
MinIO Data Retrieval (on-demand)
Data Source Provider
MinIO
Data Retrieval - API
Retrieve with Data Source Profile()
Acknowledgement()
Initiate request to retrieve new information
New Fetch File Request()
New File ()
Store new information (in the provided file format)

Figure 16: MinIO Data Retrieval (on-demand)

While the process can be triggered and executed at any time, there is also the option to execute the process automatically in predefined time intervals (scheduled pull) provided that the respective parameter is set in the data source profile. In this case, the Data Retrieval module initiates a new connection to the MinIO storage based on the Pull time interval that is set in the data source profile, retrieves the new file and stores it locally for further processing.

Table 16: MinIO Data Retrieval (scheduled)

Stakeholders involved:	N/A

Pre-conditions:	1. The existence of the data source profile 2. The Pull time interval is set in the respective data source profile
Post-conditions:	1. New information isretrieved and stored locally for further processing based on the configured pull time interval
Data Attributes	1. Data Source Profile ID
Normal Flow	1. The Data Retrieval initiates the request to the respective MinIO storage server and retrieves the new information based on a specific data source profile and the configured pull time interval 2. The new information is stored locally in the provided file format
Pass Metrics	1. The new information is available locally in the provided file format
Fail Metrics	1. The request is rejected and no information is retrieved from the server defined in the data source profile

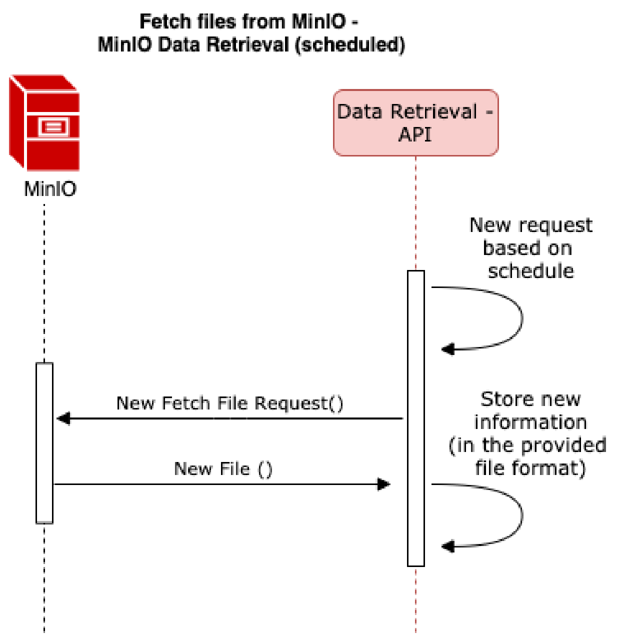

Fetch files from MinIO -
MinIO Data Retrieval (scheduled)
MinIO
Data Retrieval - API
New request based on schedule
New Fetch File Request()
New File ()
Store new information (in the provided file format)

Figure 17: MinIO Data Retrieval (scheduled)

2.2.2 Data Mapper

2.2.2.1 Data Mapping (on demand without profile)

For the Data Mapping process, the data provider is able to perform an on-demand mapping process for a specific dataset without the use of data mapping profile. This process can be executed only through the user interface of the Data Mapper, as the data provider will be instructed to provide their input by selecting the proper entities of the underlying data model that will be mapped to the entities that are extracted by their dataset. During this process, the data provider is given the option to save the provided information as a data mapping profile that can be used to automate the process in the future executions.

Table 17: Data Mapping (on demand without profile)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. A new dataset is available for the data mapping process 2. The existence of a data model that is integrated during the process start-up
Post-conditions:	1. The data mapping of the data entities of the datasets with the underlying integrated data model is available
Data Attributes	N/A
Normal Flow	1. The data source provider selects the dataset that will be utilised as input in the data mapping process. 2. The data entities of the dataset are extracted and presented to the data source provider 3. For each data entity, the data source provider selects the corresponding entity from the underlying integrated data model that the data entity will be mapped 4. The data source provider is presented with the option to save the provided information as a data mapping profile for the specific dataset 5. The data mapping between the dataset and the underlying integrated data model l is produced and stored locally in JSON format
Pass Metrics	1. The data mapping is available in JSON format 2. In case the data mapping profile option is selected, the existence of the data mapping profile for later reuse
Fail Metrics	1. The data mapping is not available. 2. In case the data mapping profile option is selected, the data mapping profile is missing

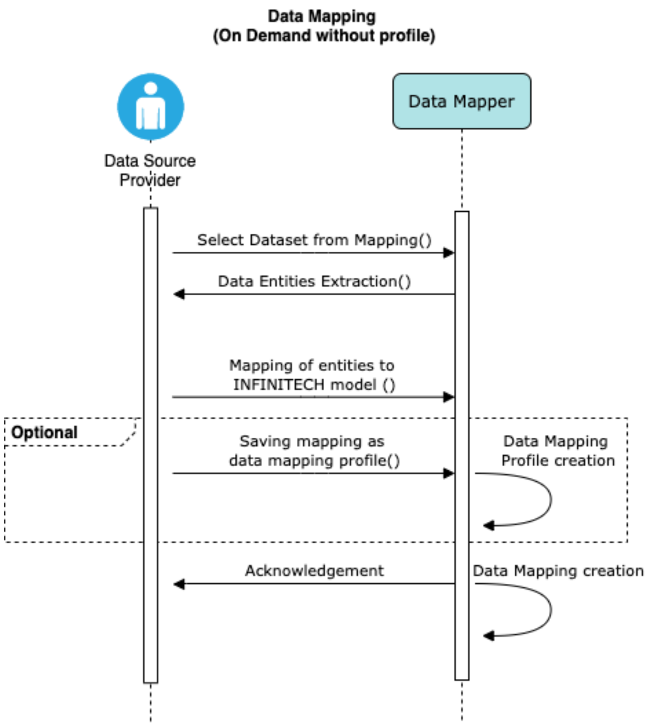

Data Mapping
(On Demand without profile)
Data Source Provider
Data Mapper
Select Dataset from Mapping()
Data Entities Extraction()
Mapping of entities to INFINITECH model ()
Optional
Saving mapping as data mapping profile()
Data Mapping Profile creation
Acknowledgement
Data Mapping creation

Figure 18: Data Mapping (on demand without profile)

2.2.2.2 Data Mapping (on demand with profile)

For the Data Mapping process, the data provider is able to perform an on-demand mapping process for a specific dataset, utilising an existing data mapping profile. This process can be executed through the user interface of the Data Mapper and upon selection of the data mapping profile. The profile is then applied to the specific dataset and the results are available for verification by the data source provider. Upon verification, the data mapping is produced and stored for later use.

Table 18: Data Mapping (on demand with profile)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. A new dataset is available for the data mapping process 2. The existence of the underlying integrated data model 3. The existence of the data mapping profile for the specific dataset
Post-conditions:	1. The data mapping of the data entities of the datasets with the underlying integrated data model is available
Data Attributes	1. Data Mapping Profile ID
Normal Flow	1. The data source provider selects the dataset that will be utilised as input in the data mapping process. 2. The data entities of the dataset are extracted and presented to the data source provider 3. The data source provider selects the data mapping profile for the specific dataset in order to be applied 4. The data mapping profile is applied to the specific dataset is available for verification by the data source provider

	5. Upon verification, the data mapping between the dataset and the underlying integrated data model is produced and stored locally in JSON format
Pass Metrics	1. The data mapping is available in JSON format
Fail Metrics	1. The data mapping is not available.

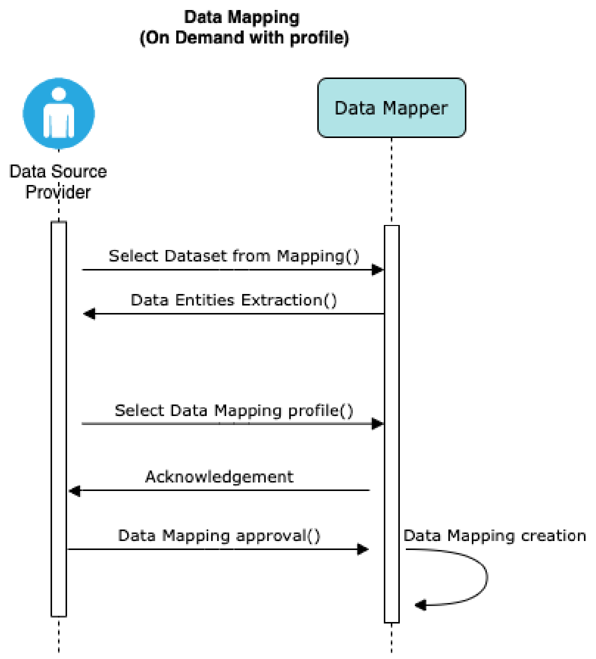


Figure 19: Data Mapping (on demand with profile)

2.2.2.3 Data Mapping Profile Registration via API

An alternative way of utilising the Data Mapper is with the use of the provided RESTful APIs. However, this option requires the definition of the data mapping profile of the specific dataset. The data mapping profile contains all the required information that will enable the automation of the data mapping process.

Table 19: Data Mapping Profile Registration via API

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of the configuration details for the data mapping of the datasets
Post-conditions:	1. The data mapping profile is available
Data Attributes	1. Data Source Name 2. Data Source Owner 3. Data Mapping rules
Normal Flow	1. The data source provider initiates a request to the Data Mapper's API endpoint to register the new data mapping profile

	2. An acknowledgement is returned and the data mapping profile is stored in the list of data mapping profiles
Pass Metrics	1. The new data mapping profile is available in the list of data mapping profiles for the data mapping process
Fail Metrics	1. The new data mapping profile is not available in the list of known data mapping profiles

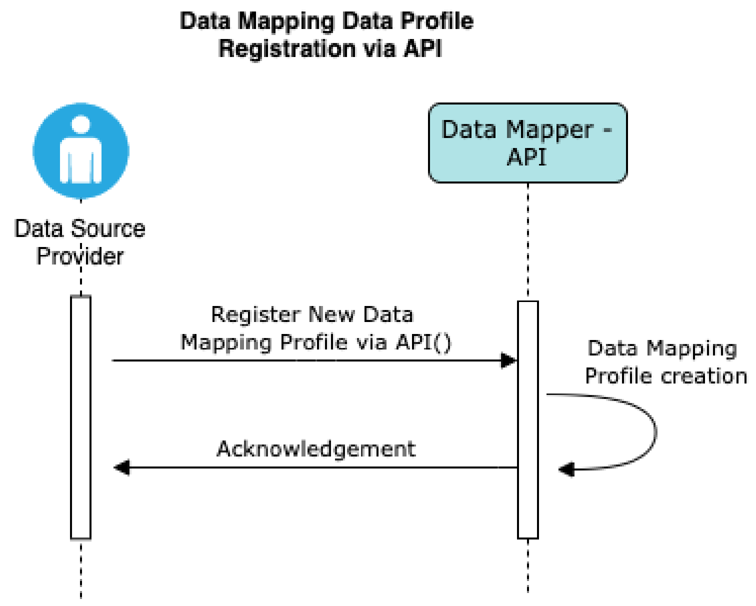

Data Mapping Data Profile Registration via API
Data Source Provider
Data Mapper - API
Register New Data Mapping Profile via API()
Data Mapping Profile creation
Acknowledgement

Figure 20: Data Mapping Profile Registration via API

2.2.2.4 Data Mapping (via API with profile)

The data mapping process can be triggered via the respective RESTful APIs, with the only precondition being the existence of a data mapping profile. The data source provider can initiate a request to the Data Mapper's API endpoint to perform the data mapping on the selected dataset based on a specific data mapping profile. Alternatively, the Data Retrieval could be the one to trigger the Data Mapper as part of the automated end-to-end data collection process.

Table 20: Data Mapping (via API with profile)

Stakeholders involved:	Data Source Provider or Data Retrieval
Pre-conditions:	1. A new dataset is available for the data mapping process 2. The existence of the common information mode of the INFINITECH 3. The existence of the data mapping profile for the specific dataset
Post-conditions:	1. The data mapping of the data entities of the datasets with the common information mode of the INFINITECH is available
Data Attributes	1. Dataset Name 2. Local file path of the dataset 3. Data Mapping Profile ID

Normal Flow	1. The data source provider or Data Retrieval initiates a request to the Data Mapper's API endpoint to perform the data mapping on the selected dataset based on a specific data mapping profile 2. The Data Mapper performs the data mapping operations on the selected dataset and produces the data mapping. 3. The new data mapping information is stored locally in a JSON format
Pass Metrics	1. The data mapping is available in JSON format
Fail Metrics	1. The data mapping is not available.

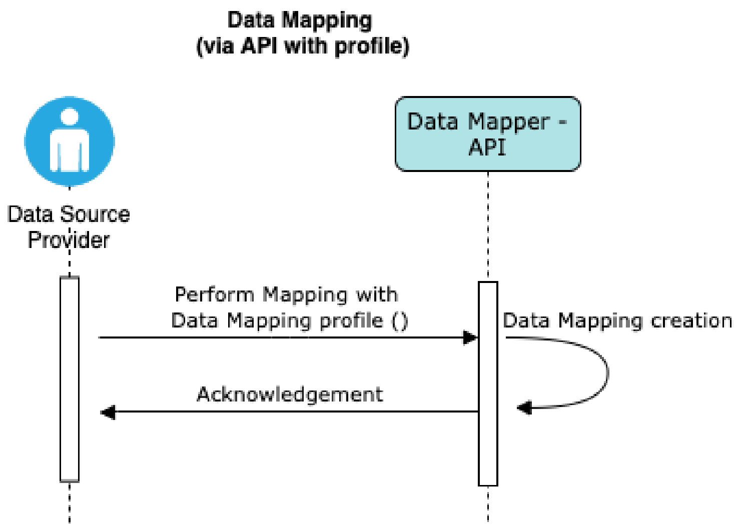


Figure 21: Data Mapping (via API with profile)

2.2.3 Data Cleaner

2.2.3.1 Data Cleaning (on demand without profile)

In the same manner as for the data mapping, for the Data Cleaning process, the data provider is able to perform an on-demand cleaning process for a specific dataset without the use of data cleaning profile. This process can be executed only through the user interface of the Data Cleaner, since the input from the data provider is required in order to define the cleaning rules that include the validation rules, as well as the data cleaning and data completion actions that will be performed in case the validation identifies an error. During this process, the data provider is given the option to save the provided information as a data cleaning profile that can be used to automated the process in future executions.

Table 21: Data Cleaning (on demand without profile)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. A new dataset is available for the data cleaning process
Post-conditions:	1. The produced “cleaned” dataset is available

Data Attributes	N/A
Normal Flow	1. The data source provider selects the dataset that will be utilised as input in the data cleaning process. 2. The data entities of the dataset are extracted and presented to the data source provider 3. For each data entity, the data source provider selects the data validation rule, as well as the data cleaning and data completion action that will be performed in case of a data validation error. 4. The data source provider is presented with the option to save the provided information as a data cleaning profile for the specific dataset 5. The data cleaning operation is performed and the "cleaned" dataset is available.
Pass Metrics	1. The "cleaned" dataset is available 2. In case the data cleaning profile option is selected, the existence of the data cleaning profile for later reuse
Fail Metrics	1. The data cleaning is not performed. 2. In case the data cleaning profile option is selected, the data cleaning profile is missing

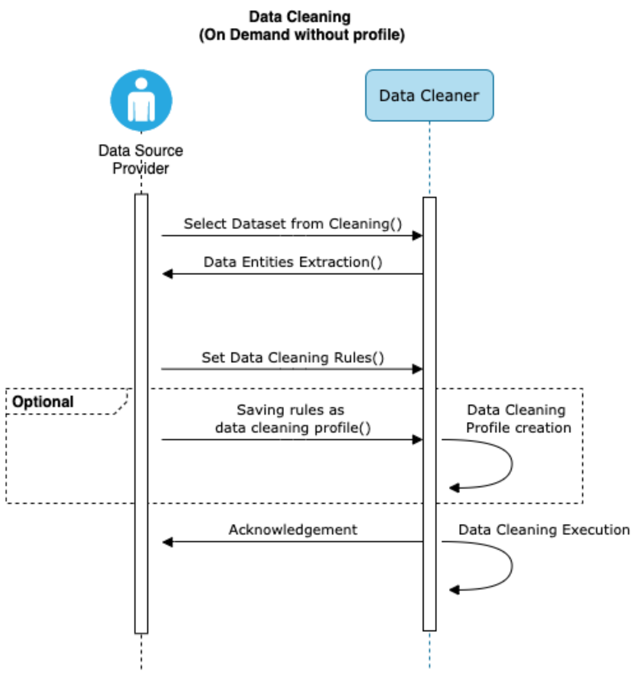

Data Cleaning
(On Demand without profile)
Data Source Provider
Data Cleaner
Select Dataset from Cleaning()
Data Entities Extraction()
Set Data Cleaning Rules()
Optional
Saving rules as data cleaning profile()
Data Cleaning Profile creation
Acknowledgement
Data Cleaning Execution

Figure 22: Data Cleaning (on demand without profile)

2.2.3.2 Data Cleaning (on demand with profile)

In the case where a data cleaning profile exists, the Data Cleaning process can be executed by the data provider on demand through the user interface of the Data Cleaner. In detail, upon the selection of the input dataset, the entities of the dataset are extracted and the data provider can select a data cleaning profile that can be used in the process. The Data Cleaning process is executed by applying the cleaning rules set in this profile and the "cleaned" dataset is available for further processing.

Table 22: Data Cleaning (on demand with profile)

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. A new dataset is available for the data cleaning process 2. The existence of the data cleaning profile for the specific dataset
Post-conditions:	1. The produced "cleaned" dataset is available
Data Attributes	1. Data Cleaning Profile ID
Normal Flow	1. The data source provider selects the dataset that will be utilised as input in the data cleaning process. 2. The data source provider selects the data cleaning profile for the specific dataset in order to be applied 3. The data cleaning profile is applied to the specific dataset is available for verification by the data source provider 4. Upon verification, the data cleaning operation is performed and the "cleaned" dataset is available.
Pass Metrics	1. The "cleaned" dataset is available
Fail Metrics	1. The data cleaning is not performed.

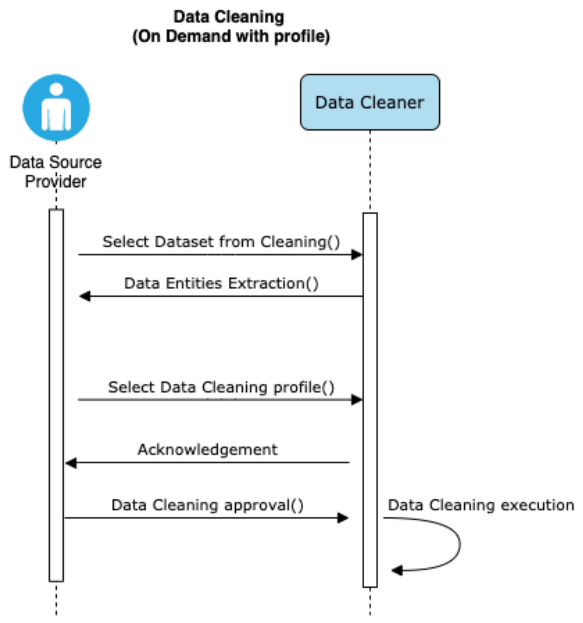


Figure 23: Data Cleaning (on demand with profile)

2.2.3.3 Data Cleaning Profile Registration via API

The execution of the Data Cleaning process can be performed also by utilising the RESTful APIs of the Data Cleaner. In this case, the first step is the registration of a data cleaning profile for a specific dataset via the respective API. The data cleaning profile contains all the required information that will enable the automation of the data cleaning process.

Table 23: Data Cleaning Profile Registration via API

Stakeholders involved:	Data Source Provider
Pre-conditions:	1. The existence of the configuration details for the data cleaning of the datasets
Post-conditions:	1. The data cleaning profile is available
Data Attributes	1. Data Source Name 2. Data Source Owner 3. Data Cleaning rules
Normal Flow	1. The data source provider initiates a request to the Data Cleaner's API endpoint to register the new data cleaning profile 2. An acknowledgement is returned and the data cleaning profile is stored in the list of data cleaning profiles
Pass Metrics	1. The new data cleaning profile is available in the list of data cleaning profiles for the data cleaning process
Fail Metrics	1. The new data cleaning profile is not available in the list of known data cleaning profiles

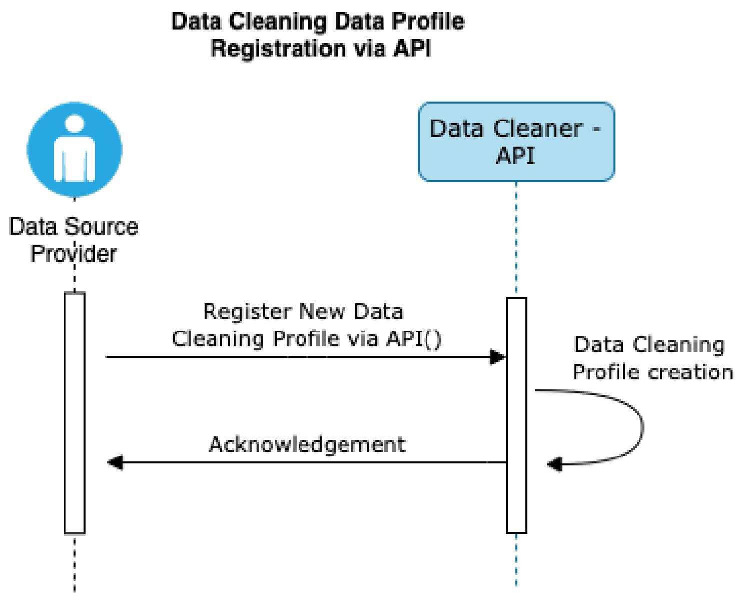

Data Cleaning Data Profile Registration via API
Data Source Provider
Data Cleaner - API
Register New Data Cleaning Profile via API()
Data Cleaning Profile creation
Acknowledgement

Figure 24: Data Cleaning Profile Registration via API

2.2.3.4 Data Cleaning (via API with profile)

The data cleaning process can be triggered via the respective RESTful APIs of the Data Cleaner, provided that a data cleaning profile has been registered beforehand. In this case, the data source provider can initiate a request to the Data Cleaner's API endpoint to perform the data cleaning on the selected dataset based on a specific data cleaning profile. Furthermore, in terms of automation of the end-to-end data collection process, the Data Mapper could trigger the Data Cleaner in order to perform the next step in the process.

Table 24: Data Cleaning (via API with profile)

Stakeholders involved:	Data Source Provider or Data Mapper
Pre-conditions:	1. A new dataset is available for the data cleaning process 2. The existence of the data cleaning profile for the specific dataset
Post-conditions:	1. The produced "cleaned" dataset is available
Data Attributes	1. Dataset Name 2. Local file path of the dataset 3. Data Cleaning Profile ID
Normal Flow	1. The data source provider or Data Mapper initiates a request to the Data Cleaner API endpoint to perform the data cleaning on the selected dataset based on a specific data cleaning profile 2. The Data Cleaner performs the data cleaning operations on the selected dataset and produces the "cleaned" dataset.
Pass Metrics	1. The "cleaned" dataset is available
Fail Metrics	1. The data cleaning is not performed.

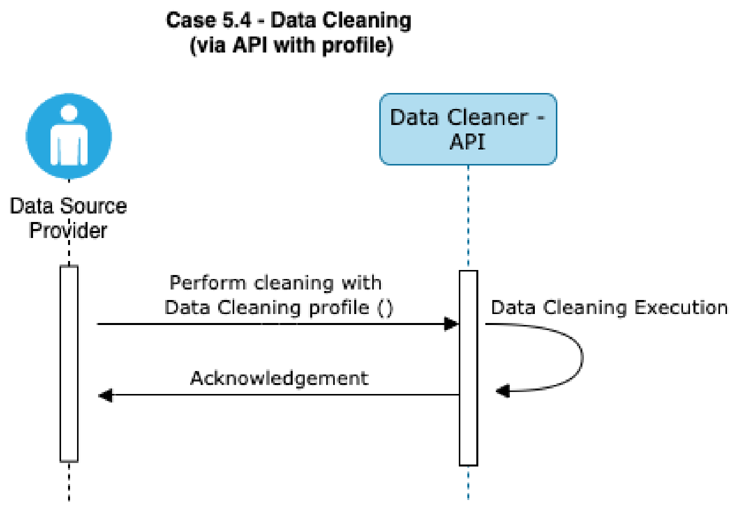


Figure 25: Data Cleaning (via API with profile)

2.2.4 Data Collection Usage Examples

As documented in Section 2.1, the INFINITECH Data Collection component has a modular architecture composed of three distinct modules, namely the Data Retrieval, the Data Mapper and the Data Cleaner, which are designed with a variety of functionalities. One of the core aspects of the design specifications of all three modules, is their high level of modularity and that they are highly configurable in terms of both operation and integration. As a result, they enable the design and execution of many different data collection pipelines that are tailored to the needs of each data provider. In this sense, a data collection pipeline can be designed by the data provider in a way that the data provider can utilise the user interfaces of the modules (or any number of them) to execute a data collection process, where in this case manual steps, executed by the data provider, will be included in the process. On the other hand, a data collection pipeline can be designed by the data provider in order to operate in a fully automated manner, in which case the whole process is executed via the use of APIs without any manual intervention.

In the following paragraphs, two main examples of the utilisation of the INFINITECH Data Collection are presented. In the first example, the data provider utilises the user interfaces of the modules in order to retrieve a new dataset from the API of a specific data source.

Hence, in the first step, the data provider registers the new data source profile for the respective API in the Data Retrieval via its user interface. Upon the creation of the new data source profile, the data source provider invokes via the user interface the retrieval of the new dataset based on this specific data source profile. Under the hood, the Data Retrieval initiates a new request to the respective API and fetches the new dataset in the form of a JSON file.

Once the dataset retrieval is completed, the data provider is moved to the next step that includes the data mapping process. The data provider exploits the user interface of the Data Mapper in order to select the newly fetched dataset. Upon the selection of the dataset, the data entities of the dataset are extracted and presented to the data source provider. The data source provider is able to select the entity from the underlying integrated data model that corresponds to each specific dataset entity and create the respective data mapping.

Upon the successfully data mapping completion, the data source provider is moved to the third and final step that includes the data cleaning process. In the same manner, the data entities of the dataset are extracted and presented to the data source provider and the data source provider is able to set the data cleaning rules for each entity of the dataset. Upon the completion of the data cleaning process, the produced “cleaned” dataset is available for later usage. The described example execution is depicted in Figure 26.

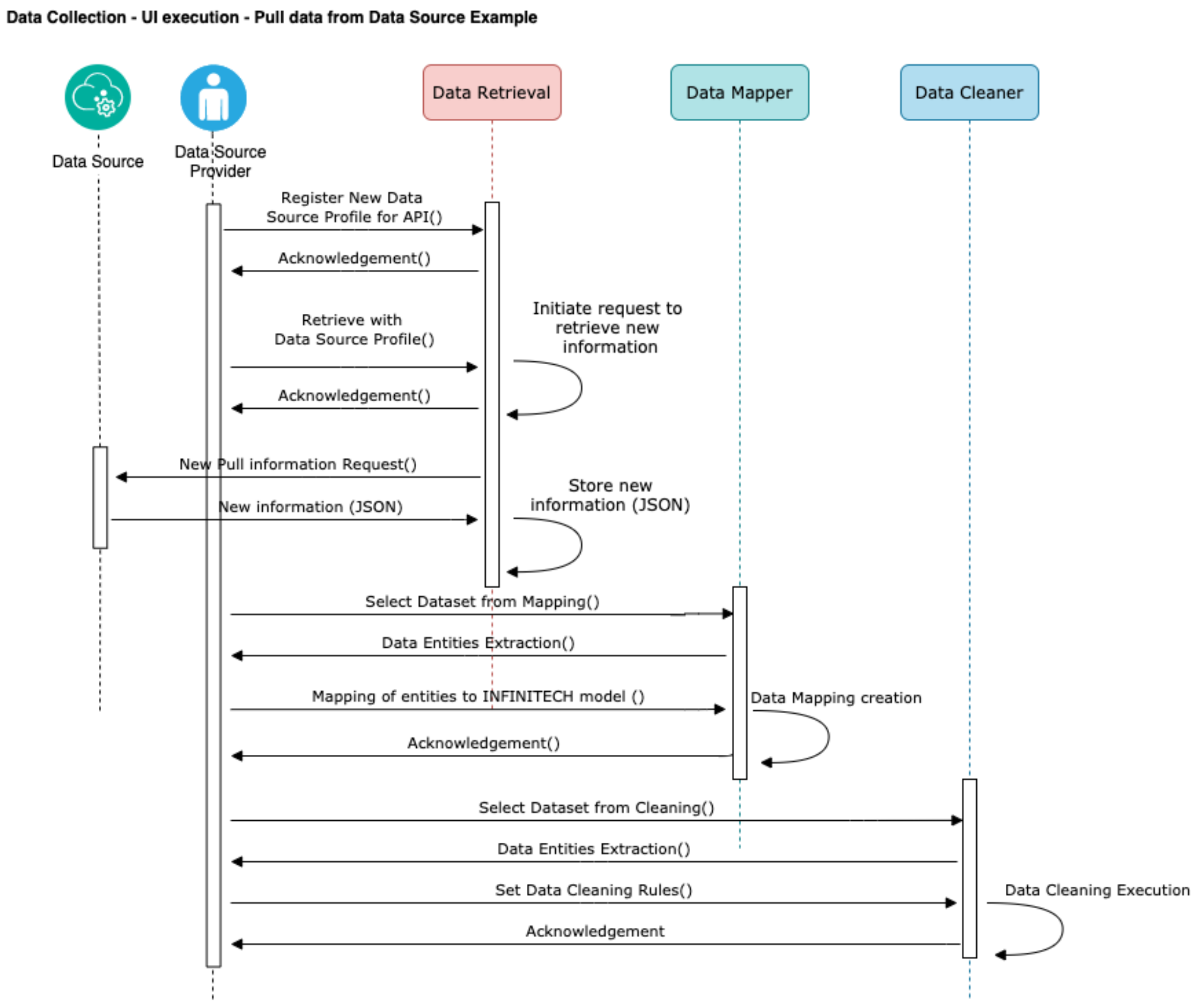

Data Collection - UI execution - Pull data from Data Source Example
Data Source
Data Source Provider
Data Retrieval
Data Mapper
Data Cleaner
Register New Data Source Profile for API()
Acknowledgement()
Retrieve with Data Source Profile()
Acknowledgement()
Initiate request to retrieve new information
New Pull information Request()
New information (JSON)
Store new information (JSON)
Select Dataset from Mapping()
Data Entities Extraction()
Mapping of entities to INFINITECH model ()
Data Mapping creation
Acknowledgement()
Select Dataset from Cleaning()
Data Entities Extraction()
Set Data Cleaning Rules()
Data Cleaning Execution
Acknowledgement

Figure 26: Data Collection Usage Example – Manual

On the other hand, the same process can be fully automated by exploiting the provided RESTful APIs of the INFINITECH Data Collection.

During the first step, the data source provider registers the corresponding data source profile, data mapping profile and data cleaning profile for the specific dataset that will be retrieved from the API of the data source. Once all profiles are successfully registered, the data source provider initiates the retrieval of the new dataset based on the defined data source profile via the corresponding Data Retrieval's API. The Data Retrieval initiates the request to the corresponding data source API and retrieves the new dataset.

Upon the successful retrieval of the new dataset, the data source provider is informed and they initiate the data mapping process with the respective data mapping profile via the Data Mapper's API. Upon the successful data mapping generation, the data provider is informed and they initiate the final step of the data cleaning process by invoking the respective API of the Data Cleaner with the corresponding data cleaning profile.

Once the data cleaning process is complete, the data provider is informed and the "cleaned" data are ready for use. The described example execution is depicted in Figure 27.

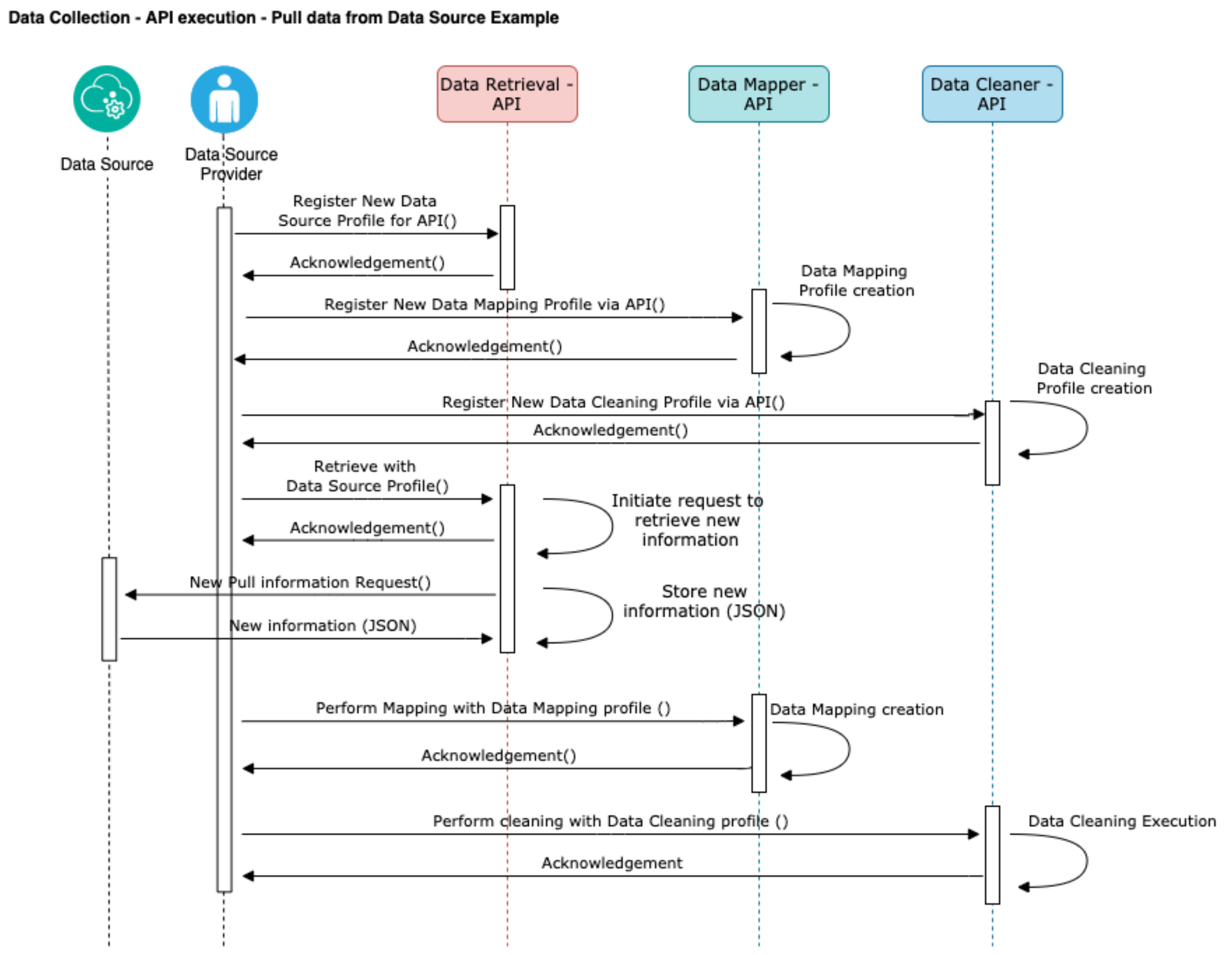

Data Collection - API execution - Pull data from Data Source Example
Data Source
Data Source Provider
Data Retrieval - API
Data Mapper - API
Data Cleaner - API
Register New Data Source Profile for API()
Acknowledgement()
Data Mapping Profile creation
Register New Data Mapping Profile via API()
Acknowledgement()
Data Cleaning Profile creation
Register New Data Cleaning Profile via API()
Acknowledgement()
Retrieve with Data Source Profile()
Initiate request to retrieve new information
Acknowledgement()
New Pull information Request()
Store new information (JSON)
New information (JSON)
Perform Mapping with Data Mapping profile ()
Data Mapping creation
Acknowledgement()
Perform cleaning with Data Cleaning profile ()
Data Cleaning Execution
Acknowledgement

Figure 27: Data Collection Usage Example – Automated via APIs

It should be noted that the presented end-to-end executions are only indicative use cases of the modularity and configurability of the INFINITECH Data Collection. As highlighted in the previous sections, the INFINITECH Data Collection can be configured in multiple ways depending on the needs of the data source provider, addressing multiple use cases where the three modules are properly configured and combined.

2.3 Implementation of the INFINITECH Data Collection

Updates from D5.13:
The specific sub-section has been updated in order to include the latest implementation details of the INFINITECH Data Collection component.

As described in section 2.1, the INFINITECH Data Collection component is composed by three distinct modules namely the Data Retrieval, the Data Mapper and the Data Cleaner, which are integrated in order to formulate the configurable solution that enables the data source providers to perform data ingestion processes tailored to their needs. The implementation of each of the three modules is driven by the design specifications that are also documented in section 2.1 of the current deliverable. In the following subsections, the final and complete implementation details of each module are presented, providing a high-level overview of how the source code is organised with the help of UML diagrams along with the implementation details of the respective functions of each module.

2.3.1 Data Retrieval

The updated class diagram of the final fully functional release of the Data Retrieval is depicted in Figure 28. As displayed in the UML diagram, the Data Retrieval is composed by the *DataRetrievalApplication that* constitutes the main service of the module and that is responsible for handling all the interactions and the communication of the Data Retrieval module with the other two modules of the INFINITECH Data Collection, as well as for the orchestration of the internal services of the module.

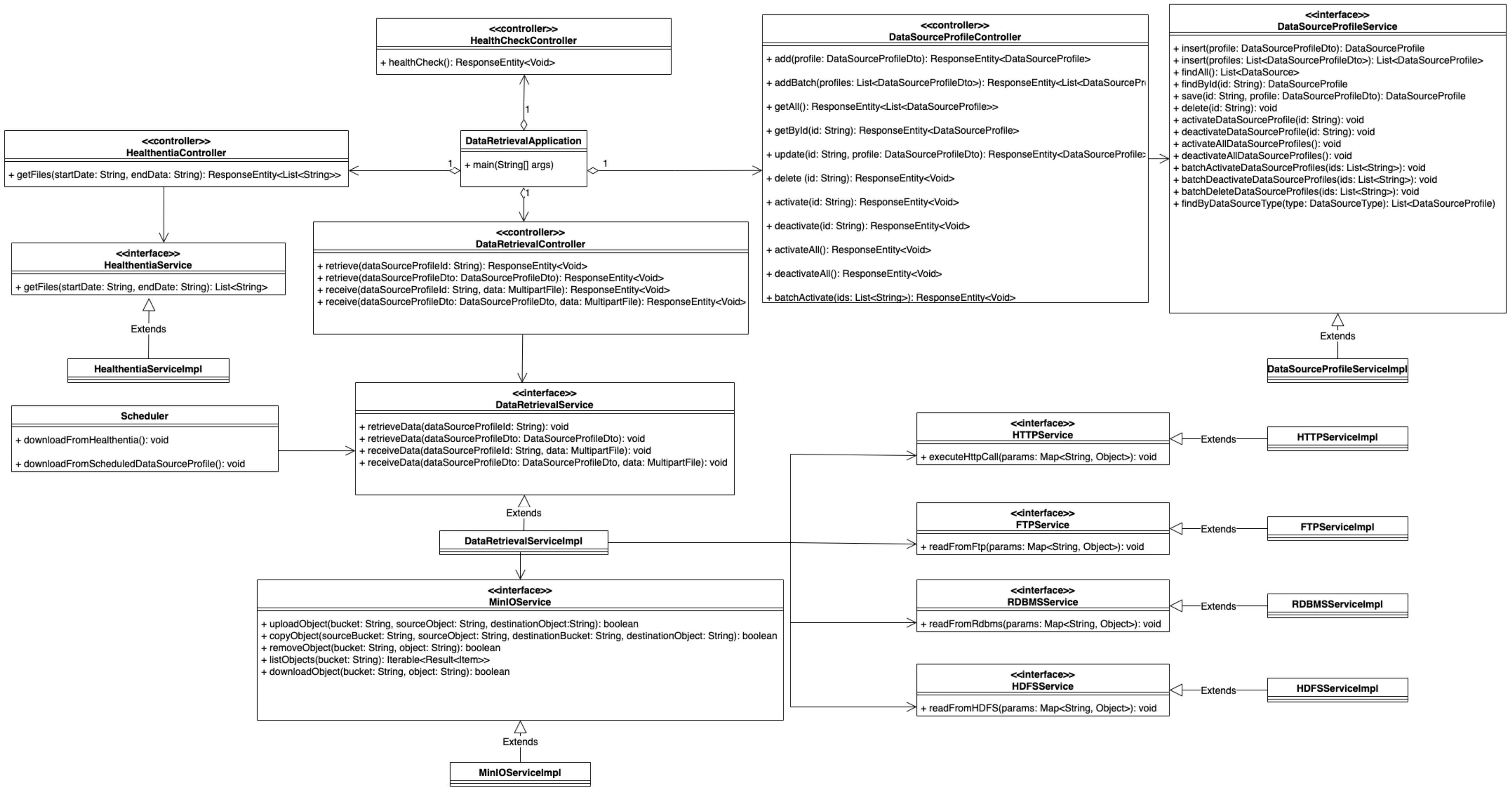
In detail, the *DataRetrievalApplication* mainly interacts with the main controllers of the Data Retrieval module, namely the *HealthCheckController,* the *DataSourceProfileController*, the *DataRetrievalController,* and the *HealthentiaController,* which undertake the interaction with the underlying internal services. These internal services are not exposed to the rest of the modules and the interaction with these services is explicitly performed through their internal interfaces that are exposed only to the respective controllers.

DataRetrievalApplication: This is the entry point class of the Data Retrieval microservice and it contains the proper functionality to initialize the application.

- ***main(String[] args)****:* This is the main function of the microservice, which is responsible for initiating the application.

HealthCheckController: This class is a REST controller responsible for exposing the proper health-check functionality of the application.

- ***healthCheck():*** This function is responsible for returning a status related to the health of the application. If the service is running properly, a status of 200 will be returned, while If the service is running with any problem a status of 500 will be returned. On the contrary, if the service is not running at all, a response will never be returned.



<<controller>>
HealthCheckController
+ healthCheck(): ResponseEntity<Void>
1
DataRetrievalApplication
+ main(String[] args)
1
1
1
<<controller>>
HealthentiaController
+ getFiles(startDate: String, endData: String): ResponseEntity<List<String>>
<<interface>>
HealthentiaService
+ getFiles(startDate: String, endDate: String): List<String>
Extends
HealthentiaServiceImpl
<<controller>>
DataRetrievalController
+ retrieve(dataSourceProfileId: String): ResponseEntity<Void>
+ retrieve(dataSourceProfileDto: DataSourceProfileDto): ResponseEntity<Void>
+ receive(dataSourceProfileId: String, data: MultipartFile): ResponseEntity<Void>
+ receive(dataSourceProfileDto: DataSourceProfileDto, data: MultipartFile): ResponseEntity<Void>
<<controller>>
DataSourceProfileController
+ add(profile: DataSourceProfileDto): ResponseEntity<DataSourceProfile>
+ addBatch(profiles: List<DataSourceProfileDto>): ResponseEntity<List<DataSourcePr
+ getAll(): ResponseEntity<List<DataSourceProfile>>
+ getById(id: String): ResponseEntity<DataSourceProfile>
+ update(id: String, profile: DataSourceProfileDto): ResponseEntity<DataSourceProfile>
+ delete (id: String): ResponseEntity<Void>
+ activate(id: String): ResponseEntity<Void>
+ deactivate(id: String): ResponseEntity<Void>
+ activateAll(): ResponseEntity<Void>
+ deactivateAll(): ResponseEntity<Void>
+ batchActivate(ids: List<String>): ResponseEntity<Void>
<<interface>>
DataSourceProfileService
+ insert(profile: DataSourceProfileDto): DataSourceProfile
+ insert(profiles: List<DataSourceProfileDto>): List<DataSourceProfile>
+ findAll(): List<DataSource>
+ findById(id: String): DataSourceProfile
+ save(id: String, profile: DataSourceProfileDto): DataSourceProfile
+ delete(id: String): void
+ activateDataSourceProfile(id: String): void
+ deactivateDataSourceProfile(id: String): void
+ activateAllDataSourceProfiles(): void
+ deactivateAllDataSourceProfiles(): void
+ batchActivateDataSourceProfiles(ids: List<String>): void
+ batchDeactivateDataSourceProfiles(ids: List<String>): void
+ batchDeleteDataSourceProfiles(ids: List<String>): void
+ findByDataSourceType(type: DataSourceType): List<DataSourceProfile)
Extends
DataSourceProfileServiceImpl
Scheduler
+ downloadFromHealthentia(): void
+ downloadFromScheduledDataSourceProfile(): void
<<interface>>
DataRetrievalService
+ retrieveData(dataSourceProfileId: String): void
+ retrieveData(dataSourceProfileDto: DataSourceProfileDto): void
+ receiveData(dataSourceProfileId: String, data: MultipartFile): void
+ receiveData(dataSourceProfileDto: DataSourceProfileDto, data: MultipartFile): void
Extends
DataRetrievalServiceImpl
<<interface>>
HTTPService
+ executeHttpCall(params: Map<String, Object>): void
Extends
HTTPServiceImpl
<<interface>>
FTPService
+ readFromFtp(params: Map<String, Object>): void
Extends
FTPServiceImpl
<<interface>>
RDBMSService
+ readFromRdbms(params: Map<String, Object>): void
Extends
RDBMSServiceImpl
<<interface>>
HDFSService
+ readFromHDFS(params: Map<String, Object>): void
Extends
HDFSServiceImpl
<<interface>>
MinIOService
+ uploadObject(bucket: String, sourceObject: String, destinationObject:String): boolean
+ copyObject(sourceBucket: String, sourceObject: String, destinationBucket: String, destinationObject: String): boolean
+ removeObject(bucket: String, object: String): boolean
+ listObjects(bucket: String): Iterable<Result<Item>>
+ downloadObject(bucket: String, object: String): boolean
Extends
MinIOServiceImpl

Figure 28: Data Retrieval UML diagram

DataSourceProfileService: This class is a Java Interface, responsible for exposing the proper methods that have to do with the management of data source profiles that are used for the data retrieval process.

- ***insert(profile: DataSourceProfileDto):*** This function accepts as parameter a new data source profile and stores it in the underlying database. The newly created profile is then returned to the user.
- ***insert(profiles: List<DataSourceProfileDto>)****:* This function accepts as parameter a list of new data source profiles and stores them in the underlying database. The newly created profiles are then returned to the user.
- ***findAll()****:* This function returns a list of all the available data source profiles.
- ***findById(id: String):*** This function accepts as parameter an id of a data source profile and returns that specific profile, if it exists.
- ***save(id: String, profile: DataSourceProfileDto):*** This function accepts as parameter an id of a data source profile, as well as a new profile configuration, and updates the profile that corresponds to the given id, based on the values of the newly given profile.
- ***delete(id: String)****:* This function accepts as parameter an id of a data source profile and deletes that specific profile if it exists.
- ***activateDataSourceProfile(id: String)****:* This function accepts as parameter an id of a data source profile and activates that profile, if it exists.
- ***deactivateDataSourceProfile(id: String)****:* This function accepts as parameter an id of a data source profile and deactivates that profile, if it exists.
- ***activateAllDataSourceProfiles()****:* This function is responsible for activating all available data source profiles.
- ***deactivateAllDataSourceProfiles()****:* This function is responsible for deactivating all available data source profiles.

- ***batchActivateDataSourceProfiles(ids: List<String>)****:* This function accepts as parameter a list of ids of data source profiles and activates those profiles.
- ***batchDeactivateDataSourceProfiles(ids: List<String>)****:* This function accepts as parameter a list of ids of data source profiles and deactivates those profiles.
- ***batchDeleteDataSourceProfiles(ids: List<String>)****:* This function accepts as parameter a list of ids of data source profiles and deletes those profiles.
- ***findByDataSourceType(type: DataSourceType)****:* This function accepts as parameter a data source profile type (e.g. MinIO, HDFS, HTTP, etc), and returns a list of profiles that are configured for the specific type.

DataSourceProfileServiceImpl: This class implements the interface *DataSourceProfileService* and all its corresponding functions. No further functionality is provided.

DataSourceProfileController: This class is a REST-based controller that exposes the data source profiles management functionalities, by taking advantage of the functions available from the interface *DataSourceProfileService*.

- ***add(profile: DataSourceProfileDto)****:* This function accepts as parameter a new data source profile and stores it in the underlying storage, by calling the *insert* function of the *DataSourceProfileService*.
- ***addBatch(profiles: List<DataSourceProfileDto>)****:* This function accepts as parameter a list of new data source profiles and stores them in the underlying storage, by calling the *insert* function of the *DataSourceProfileService*.

- ***getAll()****:* This function returns a list of all the available data source profiles, by calling the *findAll* function of the *DataSourceProfileService*.
- ***getById(id: String)****:* This function accepts an id as a path variable and returns the data source profile that corresponds to that id, by calling the *findById* function of the *DataSourceProfileService*.
- ***update(id: String, profile: DataSourceProfileDto)****:* This function accepts an id as path variable and a new datasource profile in the request body and updates the data source profile that corresponds to that given id with the value of the new profile, by calling the *save* function of the *DataSourceProfileService*.
- ***delete (id: String)****:* This function accepts an id as a path variable and deletes the data source profiles that correspond to that id, by calling the *delete* function of the *DataSourceProfileService*.
- ***activate(id: String)****:* This function accepts an id as a path variable and activates the data source profiles that correspond to that id, by calling the *activateDataSourceProfile* function of the *DataSourceProfileService*.
- ***deactivate(id: String)****:* This function accepts an id as a path variable and deactivates the data source profiles that correspond to that id, by calling the de*activateDataSourceProfile* function of the *DataSourceProfileService*.
- ***activateAll()****:* This function is responsible for activating all the available data source profiles, by calling the *activateAllDataSourceProfiles* function of the *DataSourceProfileService*.
- ***deactivateAll()****:* This function is responsible for deactivating all the available data source profiles, by calling the de*activateAllDataSourceProfiles* function of the *DataSourceProfileService*.
- ***batchActivate(ids: List<String>)****:* This function accepts a list of data source profile ids in the request body and activates the profiles that correspond to those ids, by calling the *batchActivateDataSourceProfiles* function of the *DataSourceProfileService.*
- ***batchDeactivate(ids: List<String>)****:* This function accepts a list of data source profile ids in the request body and deactivates the profiles that correspond to those ids, by calling the *batchDeactivateDataSourceProfiles* function of the *DataSourceProfileService.*

- ***batchDelete(ids: List<String>)****:* This function accepts a list of data source profile ids in the request body and deletes the profiles that correspond to those ids, by calling the *batchDeleteDataSourceProfiles* function of the *DataSourceProfileService.*

DataRetrievalService: This class is a Java Interface, responsible for exposing the proper methods that have to do with the proper collection of data from various sources.

- ***retrieveData(dataSourceProfileId: String)****:* This function accepts as parameter the id of an existing data source profile and triggers the data retrieval process, based on the configuration available in this profile. This function is preferred for automated data retrieval.
- ***retrieveData(dataSourceProfileDto: DataSourceProfileDto)****:* This function accepts as parameter a new data source profile and triggers the retrieval process based on that profile. This function is used for the manual data retrieval where the data source profile is explicitly defined each time and we are not interested in storing it.
- ***receiveData(dataSourceProfileId: String, data: MultipartFile)****:* This function accepts as parameters a data source profile id as well as a Multipart file and is responsible for storing that file in the intermediate MinIO storage and further process it based on the data source profile that corresponds to the given id.
- ***receiveData(dataSourceProfileDto: DataSourceProfileDto, data: MultipartFile)****:* This function accepts as parameters a new data source profile as well as a Multipart file and is responsible for

storing that file in the intermediate MinIO storage and further process it based on the given data source profile.

DataRetrievalServiceImpl: This class implements the interface *DataRetrievalService* and all its corresponding functions. No further functionality is provided.

DataRetrievalController: This class is a REST-based controller that exposes the data retrieval functionalities, by taking advantage of the functions available from the interface DataRetrievalService.

- ***retrieve(dataSourceProfileId: String)****:* This function accepts as path variable the id of an existing data source profile and triggers the retrieval process by making use of the *retrieveData* function of the *DataRetrievalService*.
- ***retrieve(dataSourceProfileDto: DataSourceProfileDto)****:* This function accepts as a request body a new data source profile and triggers the retrieval process by making use of the *retrieveData* function of the *DataRetrievalService*.
- ***receive(dataSourceProfileId: String, data: MultipartFile)****:* This function accepts both the id of an existing data source profile, as well as a Multipart file in form-data format and processes the incoming file by making use of the *receiveData* function of the *DataRetrievalService*.
- ***receive(dataSourceProfileDto: DataSourceProfileDto, data: MultipartFile)****:* This function accepts both a new data source profile, as well as a Multipart file in form-data format and processes the incoming file by making use of the *receiveData* function of the *DataRetrievalService*.

MinIOService: This class is a Java Interface, responsible for exposing the proper methods that have to do with data collection and other functionalities on-top of a MinIO installation.

- ***uploadObject(bucket: String, sourceObject: String, destinationObject:String)****:* This function accepts as parameters the name of a MinIO bucket, the name of a local file and the name that the file should have in MinIO and uploads that file to the corresponding bucket with the given name.
- ***copyObject(sourceBucket: String, sourceObject: String, destinationBucket: String, destinationObject: String)****:* This function accepts as parameters the names of two MinIO buckets as well two names of objects and copies the object from the first bucket to the second one.

- ***removeObject(bucket: String, object: String)****:* This function accepts as parameters the name of a MinIO bucket and an object and is responsible for deleting that object from the specific bucket.
- ***listObjects(bucket: String)****:* This function is responsible for listing all files inside a MinIO bucket.
- ***downloadObject(bucket: String, object: String)****:* This function accepts as parameters the name of a MinIO bucket and object and downloads that object as a temporary file in the local file system.

MinIOServiceImpl: This class implements the interface *MinIOService* and all its corresponding functions. No further functionality is provided.

HTTPService: This class is a Java Interface, responsible for exposing the proper methods that have to do with data acquisition using the HTTP protocol, such as REST APIs and HTTP servers.

- ***executeHttpCall(params: Map<String, Object>)****:* This function is responsible for making an HTTP call based on the given parameters and stores that data that comes as a result from that HTTP call.

HTTPServiceImpl: This class implements the interface *HTTPService* and all its corresponding functions. No further functionality is provided.

FTPService: This class is a Java Interface, responsible for exposing the proper methods that have to do with the collection of data from FTP servers.

- ***readFromFtp(params: Map<String, Object>)****:* This function is responsible for retrieving data from an FTP server based on the parameters given as argument.

FTPServiceImpl: This class implements the interface *FTPService* and all its corresponding functions. No further functionality is provided.

RDBMSService: This class is a Java Interface, responsible for exposing the proper methods that have to do with the collection of data from relational databases such as MySQL and PostgreSQL.

- ***readFromRdbms(params: Map<String, Object>)****:* This function is responsible for retrieving data from a relational database management system based on the parameters given as argument.

RDBMSServiceImpl: This class implements the interface *RDBMSService* and all its corresponding functions. No further functionality is provided.

HDFSService: This class is a Java Interface, responsible for exposing the proper methods that have to do with the collection of data from Hadoop HDFS installations.

- ***readFromHDFS(params: Map<String, Object>)****:* This function is responsible for retrieving data from a Hadoop HDFS based on the parameters given as argument.

HDFSServiceImpl: This class implements the interface *HDFSService* and all its corresponding functions. No further functionality is provided.

HealthentiaService: This class is a Java Interface, responsible for exposing the proper functionality needed specifically for pilot 12 and the files that are being collected from the Healthentia API.

- ***getFiles(startDate: String, endDate: String)****:* This function accepts as parameter a start and end date and returns a list of all the already downloaded files from the retrieval functionality that are available in the intermediate MinIO storage.

HealthentiaServiceImpl: This class implements the interface *HealthentiaService* and all its corresponding functions. No further functionality is provided.

HealthentiaController: This class is a REST-based controller that is meant to be consumed explicitly by pilot 12 and exposes the proper functionality required by that pilot, by taking advantage of the functions that are available in the *HealthentiaService* interface.

- ***getFiles(startDate: String, endData: String)****:* This function accepts as query parameters a start and end date in format YYYY-MM-DD and returns a list of the available Healthentia files in MinIO, by calling the *getFiles* function of the class *HealthentiaService*.

Scheduler: This class encapsulates the proper functionality to provide scheduled runs for the necessary data source profiles, as well specific implementations to cover special needs.

- ***downloadFromHealthentia()****:* This function is a scheduled task developed specifically for the needs of the pilot 12 in order to fetch data from the Healthentia API by taking into consideration several factors such as the current date and the various data types available for retrieval.
- ***downloadFromScheduledDataSourceProfile()****:* This function is a generic scheduled task responsible for running the necessary data source profiles based on the pre-configured interval by always using the same configuration.

2.3.2 Data Mapper

Figure 29 depicts the class diagram of the Data Mapper and as illustrated in the UML diagram, the *DataMapperApplication* is the main service of the module, acting as the interface of the module with the rest of the modules of the INFINITECH Data Collection and providing the level of abstraction on top of the internal services of the module that are not exposed outside the module. In this sense, the *DataMapperApplication* interacts with the main services of the Data Mapper, namely the *MappingModelService, MappingConfigurationService, MinioService* and the *MappingService via the corresponding internal controllers* which undertake the interaction with the underlying internal services. These internal services are not exposed to the rest of the modules and the interaction with these services is explicitly performed through their internal interfaces that are exposed only to the respective controllers.

DataMapperApplication: This is the entry point class of the Data Mapper microservice and it contains the proper functionality to initialize the application.

- ***main(String[] args)****:* This is the main function of the microservice, which is responsible for initiating the application.

MappingModelService: This class provides the proper functionalities which are responsible for the CRUD operations on a Mapping model.

- ***getMappingModelFields(String modelName):*** The specific function takes as input a model name and returns all the fields that belong to it as a list.
- ***getMappingModelField(FieldID id):*** The specific function, takes as input a FieldID object (that contains both model and field name) and fetches the respective field.
- ***deleteMappingModelFields(String modelName):*** The specific function takes as input a mapping mode's name and deletes all its fields.
- ***updateMappingModelField(UpdateFieldDTO update):*** The specific takes as input an UpdateFieldDTO and performs the requested update to the respective field of the mapping model.
- ***updateMappingModelFields(MappingModelDTO mappingModel):*** The specific function performs a batched update on the fields of the mapping model that is given in the arguments.
- ***insertNewMappingModelField(String modelName, FieldDTO field):*** The specific function takes as input a mapping model name and a field and adds the field in the mapping model.

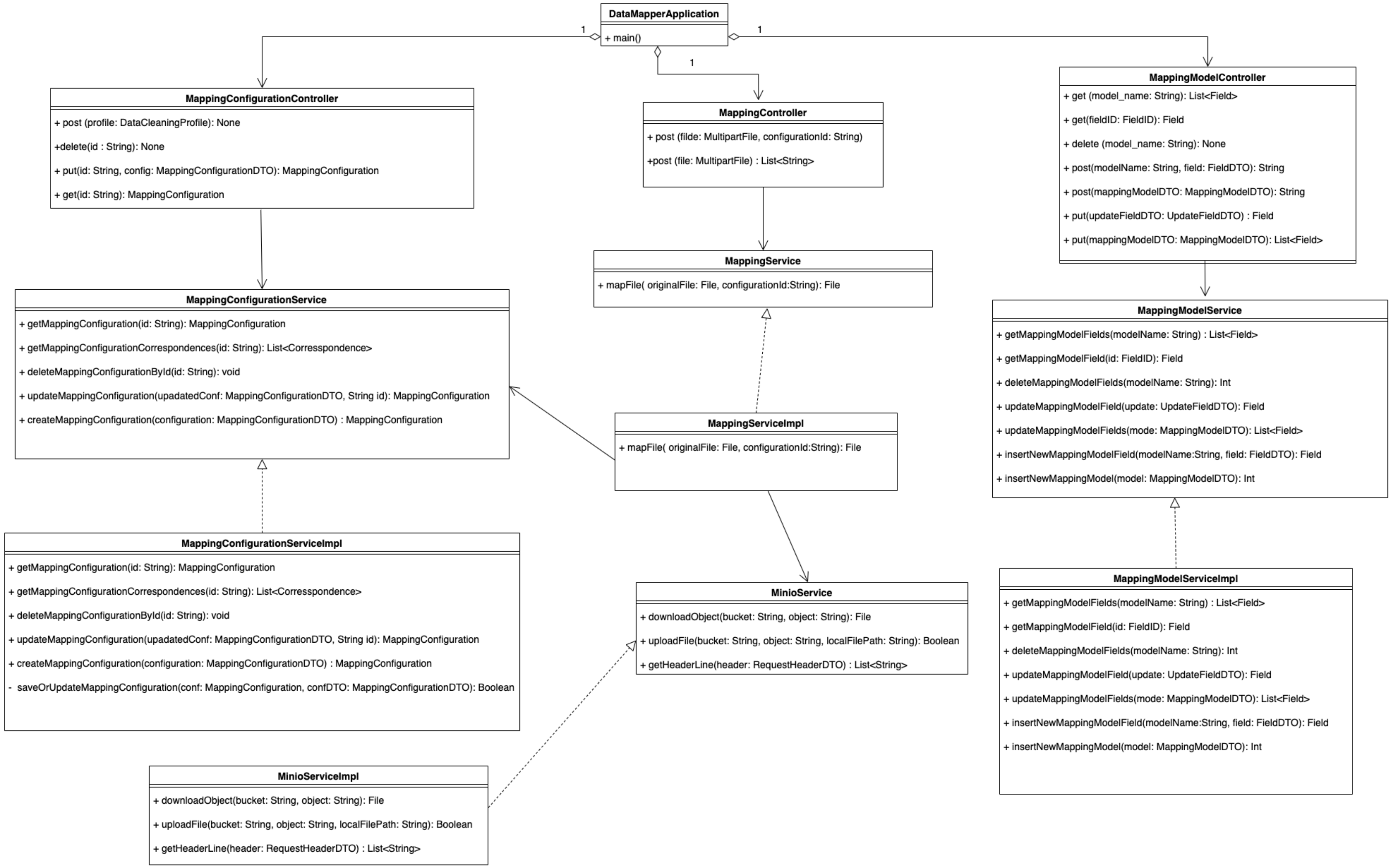

DataMapperApplication
+ main()
1
1
1
MappingConfigurationController
+ post (profile: DataCleaningProfile): None
+delete(id : String): None
+ put(id: String, config: MappingConfigurationDTO): MappingConfiguration
+ get(id: String): MappingConfiguration
MappingController
+ post (filde: MultipartFile, configurationId: String)
+post (file: MultipartFile) : List<String>
MappingModelController
+ get (model_name: String): List<Field>
+ get(fieldID: FieldID): Field
+ delete (model_name: String): None
+ post(modelName: String, field: FieldDTO): String
+ post(mappingModelDTO: MappingModelDTO): String
+ put(updateFieldDTO: UpdateFieldDTO) : Field
+ put(mappingModelDTO: MappingModelDTO): List<Field>
MappingService
+ mapFile(originalFile: File, configurationId:String): File
MappingConfigurationService
+ getMappingConfiguration(id: String): MappingConfiguration
+ getMappingConfigurationCorrespondences(id: String): List<Corresspondence>
+ deleteMappingConfigurationById(id: String): void
+ updateMappingConfiguration(upadatedConf: MappingConfigurationDTO, String id): MappingConfiguration
+ createMappingConfiguration(configuration: MappingConfigurationDTO) : MappingConfiguration
MappingModelService
+ getMappingModelFields(modelName: String) : List<Field>
+ getMappingModelField(id: FieldID): Field
+ deleteMappingModelFields(modelName: String): Int
+ updateMappingModelField(update: UpdateFieldDTO): Field
+ updateMappingModelFields(mode: MappingModelDTO): List<Field>
+ insertNewMappingModelField(modelName:String, field: FieldDTO): Field
+ insertNewMappingModel(model: MappingModelDTO): Int
MappingServiceImpl
+ mapFile(originalFile: File, configurationId:String): File
MappingConfigurationServiceImpl
+ getMappingConfiguration(id: String): MappingConfiguration
+ getMappingConfigurationCorrespondences(id: String): List<Corresspondence>
+ deleteMappingConfigurationById(id: String): void
+ updateMappingConfiguration(upadatedConf: MappingConfigurationDTO, String id): MappingConfiguration
+ createMappingConfiguration(configuration: MappingConfigurationDTO) : MappingConfiguration
- saveOrUpdateMappingConfiguration(conf: MappingConfiguration, confDTO: MappingConfigurationDTO): Boolean
MinioService
+ downloadObject(bucket: String, object: String): File
+ uploadFile(bucket: String, object: String, localFilePath: String): Boolean
+ getHeaderLine(header: RequestHeaderDTO) : List<String>
MappingModelServiceImpl
+ getMappingModelFields(modelName: String) : List<Field>
+ getMappingModelField(id: FieldID): Field
+ deleteMappingModelFields(modelName: String): Int
+ updateMappingModelField(update: UpdateFieldDTO): Field
+ updateMappingModelFields(mode: MappingModelDTO): List<Field>
+ insertNewMappingModelField(modelName:String, field: FieldDTO): Field
+ insertNewMappingModel(model: MappingModelDTO): Int
MinioServiceImpl
+ downloadObject(bucket: String, object: String): File
+ uploadFile(bucket: String, object: String, localFilePath: String): Boolean
+ getHeaderLine(header: RequestHeaderDTO) : List<String>

Figure 29: Data Mapper UML diagram

- ***insertNewMappingMode(MappingModelDTO mappingModel):*** The specific function performs the insertion of a new mapping model to the Mapper's database.

MappingConfigurationService: This class provides the proper functionalities which are responsible for the CRUD operations on a Mapping Configuration.

- ***getMappingConfiguration(String id):*** The specific function fetches the mapping configuration identified by the provided id.
- ***getMappingConfigurationCorrespondences(String id):*** The specific function fetches all field correspondences of the mapping configuration identified by the provided id.
- ***deleteMappingConfigurationId(String id):*** The specific function deletes the mapping configuration identified by the provided id.
- ***updateMappingConfiguration(String id, MappingConfigurationDTO update):*** The specific function takes as input a MappingConfiguration DTO and a mapping configuration id. It updates appropriately the mapping configuration which is identified by the given id.
- ***createMappingConfiguration(MappingConfigurationDTO update):*** The specific function performs the insertion of a new mapping configuration to the Mapper's database.

MinioService: This class provides the proper functionalities that are responsible for the communication with Minio object Storage.

- ***downloadObject(String bucket, String object):*** The specific function takes as input a bucket's name in Minio as well as the path of the object inside this bucket, and downloads and stores the particular object in the temporary storage.
- ***uploadFile(String bucket, String object, String localFilePath):*** This function uploads a file whose location is specified in *localFilePath* variable, in the specified bucket of Minio in the given object path.
- ***getHeaderLine(RequestHeaderDTO requestHeaderDTO):*** This function takes as input the path of a csv file which is stored in Minio and invokes minioClient program to return its header line as a list of strings.

MappingService: This class generates the mapping between the input data elements and the mapping instructions included in the provided mapping configuration.

- ***mapFile(File originalFile, String configurationID):*** The specific function takes as input a file and a configurationID ,and produces a new file that has been mapped based on the correspondences of the mapping configuration that the configurationID refers to.

2.3.3 Data Cleaner

The Data Cleaner implements the data cleaning process, that is documented in Section 2.1.3, and is composed of four (4) steps: a) the data validation, b) the data correction, c) the missing values imputation and d) the complete logging of the performed data cleaning operations. The class diagram of the Data Cleaner is depicted in Figure 30.

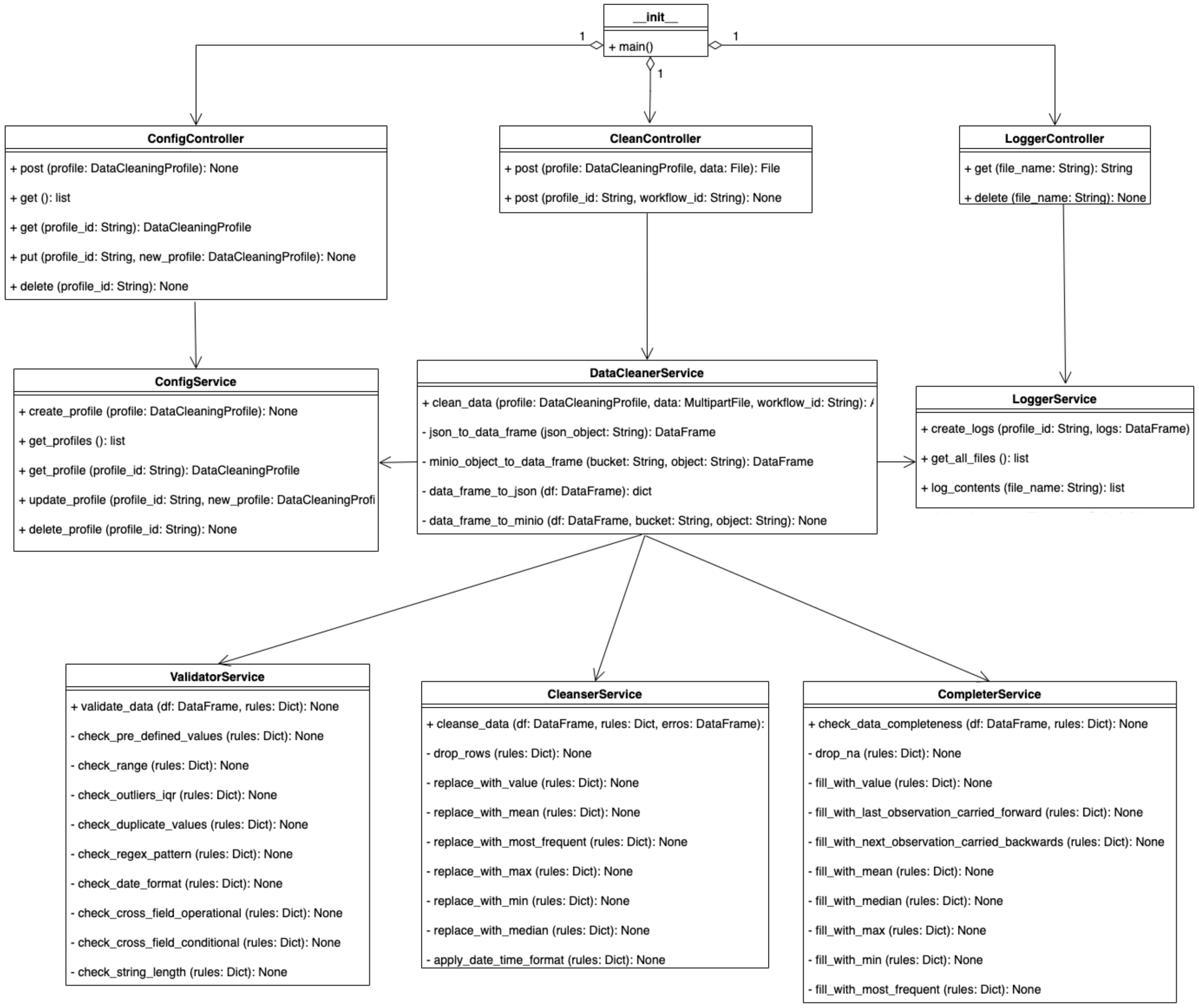

__init__
+ main()
1
1
1
ConfigController
+ post (profile: DataCleaningProfile): None
+ get (): list
+ get (profile_id: String): DataCleaningProfile
+ put (profile_id: String, new_profile: DataCleaningProfile): None
+ delete (profile_id: String): None
CleanController
+ post (profile: DataCleaningProfile, data: File): File
+ post (profile_id: String, workflow_id: String): None
LoggerController
+ get (file_name: String): String
+ delete (file_name: String): None
ConfigService
+ create_profile (profile: DataCleaningProfile): None
+ get_profiles (): list
+ get_profile (profile_id: String): DataCleaningProfile
+ update_profile (profile_id: String, new_profile: DataCleaningProfi
+ delete_profile (profile_id: String): None
DataCleanerService
+ clean_data (profile: DataCleaningProfile, data: MultipartFile, workflow_id: String): /
- json_to_data_frame (json_object: String): DataFrame
- minio_object_to_data_frame (bucket: String, object: String): DataFrame
- data_frame_to_json (df: DataFrame): dict
- data_frame_to_minio (df: DataFrame, bucket: String, object: String): None
LoggerService
+ create_logs (profile_id: String, logs: DataFrame)
+ get_all_files (): list
+ log_contents (file_name: String): list
ValidatorService
+ validate_data (df: DataFrame, rules: Dict): None
- check_pre_defined_values (rules: Dict): None
- check_range (rules: Dict): None
- check_outliers_iqr (rules: Dict): None
- check_duplicate_values (rules: Dict): None
- check_regex_pattern (rules: Dict): None
- check_date_format (rules: Dict): None
- check_cross_field_operational (rules: Dict): None
- check_cross_field_conditional (rules: Dict): None
- check_string_length (rules: Dict): None
CleanserService
+ cleanse_data (df: DataFrame, rules: Dict, erros: DataFrame):
- drop_rows (rules: Dict): None
- replace_with_value (rules: Dict): None
- replace_with_mean (rules: Dict): None
- replace_with_most_frequent (rules: Dict): None
- replace_with_max (rules: Dict): None
- replace_with_min (rules: Dict): None
- replace_with_median (rules: Dict): None
- apply_date_time_format (rules: Dict): None
CompleterService
+ check_data_completeness (df: DataFrame, rules: Dict): None
- drop_na (rules: Dict): None
- fill_with_value (rules: Dict): None
- fill_with_last_observation_carried_forward (rules: Dict): None
- fill_with_next_observation_carried_backwards (rules: Dict): None
- fill_with_mean (rules: Dict): None
- fill_with_median (rules: Dict): None
- fill_with_max (rules: Dict): None
- fill_with_min (rules: Dict): None
- fill_with_most_frequent (rules: Dict): None

Figure 30: Data Cleaner UML diagram

As displayed in Figure 30, the DataCleaner module is composed by the main service, namely the *DataCleanerService*, and a set of the internal services, each one undertaking a specific step of the data cleaning process, namely the *ConfigService, the ValidatorService*, the *CleanserService*, the *CompleterService* and the *LoggerService.*

The *DataCleanerService* orchestrates the execution of the data cleaning process by exploiting the internal services via the internally exposed interfaces that each service offers explicitly to the main service. The *DataCleanerService* executes the data cleaning process based on the set of data cleansing rules that are defined by the data source provider. In these rules, the configuration that will customise the operations of the internal services of *the ValidatorService*, the *CleanserService* and the *CompleterService* is defined. The *ConfigService* and the *LoggerService* are supplementary services that facilitate the execution of the data cleaning process. Hence, the *DataCleanerService* implements the following main functions:

- ***clean_data (profile: DataCleaningProfile, data: MultipartFile, workflow_id: String):*** The specific function is orchestrating the execution of the data cleaning process utilising an existing or a new data cleaning profile. The function receives as input the identifier of the data cleaning profile along with the path of the local file or the file itself, and based on the rules defined in the profile it invokes ValidatorService, the CleanserService and the CompleterService services.
- ***json_to_data_frame (json_object: String):*** This is an internal function that accepts a JSON formatted string and converts it into a pandas DataFrame.
- ***minio_object_to_data_frame (bucket: String, object: String):*** This is an internal function that accepts a bucket and object name from MinIO, reads that object and loads it into a pandas DataFrame.
- ***data_frame_to_json (df: DataFrame):*** This internal function takes as input a pandas DataFrame and converts it into a python dict formatted JSON.
- ***data_frame_to_minio (df: DataFrame, bucket: String, object: String):*** This is an internal function that accepts a pandas DataFrame as well as the names of a MinIO bucket and object, and stores the content of that DataFrame to specified MinIO bucket.
- ***pre_compute_column_statistics (df: DataFrame):*** This is very useful internal function that pre-computes statistics for all the columns (mean, max, min, etc), so that they can be used during the cleaning and missing value handling steps.

The *DataCleanerService* provides the *CleanController* interface that constitutes the single external exposed interface that the INFINITECH Data Collection offers from the specific service. Through this interface, the data cleaning process is initiated and executed. The interface has the following main functions:

- ***post (profile: DataCleaningProfile, data: MultipartFile):*** This controller-level function takes as input a new data cleaning profile as well as a file, triggers the **clean_data** method of the **DataCleanserService** and returns the cleaned dataset back to the user.
- ***post (profile: DataCleaningProfile, workflow_id: String):*** This controller-level function is used in the case of the "automated" cleaning process and accepts as input the id of the data cleaning profile to apply, as well as an id of the workflow process which contains the information of the path of the file to clean.

The *ConfigService* is the internal service that undertakes the profile management operations that involve the registration, storage and retrieval of the data cleaning profiles. To this end, the *ConfigService,* in accordance with the rest of profile management services of the INFINITECH Data Collection, implements proper functionality for CRUD operations on data cleaning profiles:

- ***create_profile (profile: DataCleaningProfile):*** This function accepts as input a data cleaning profile object and stores it.
- ***get_profiles ():*** This function returns a list of all the currently available cleaning profiles.

- ***get_profile (profile_id: String)**:* This function takes as input an id of a specific cleaning profile, and returns that profile, if exists.
- ***update_profile (profile_id: String, new_profile: DataCleaningProfile)**:* This function takes as input the id of an existing cleaning profile as well as new profile, and updates the old profile based on the values of the new one.
- ***delete_profile (profile_id: String)**:* This function takes as input the id of an existing cleaning profile and deletes that profile.

The *ConfigController* class is a REST controller that exposes the profile configuration functionalities to the outer world by using the proper functions of the ConfigService:

- ***post (profile: DataCleaningProfile)**:* This controller-level function is used to store a new data cleaning profile, by invoking the **create_profile** function of the **ConfigService** class.
- ***get ()**:* This controller-level function is used to fetch all the available data cleaning profiles, by invoking the **get_profiles** function of the **ConfigService** class.
- ***get (profile_id: String)**:* This controller-level function is used to fetch a specific data cleaning profile, by invoking the **get_profile** function of the **ConfigService** class.
- ***put (profile_id: String, new_profile: DataCleaningProfile)**:* This controller-level function is used to update a specific data cleaning profile, by invoking the **update_profile** function of the **ConfigService** class.
- ***delete (profile_id: String)**:* This controller-level function is used to delete a specific data cleaning profile, by invoking the **delete_profile** function of the **ConfigService** class.

The *ValidatorService* is the internal service that is responsible for the data validation checks that are performed on the selected dataset. In this sense, the service offers an extended list of data validation checks which are performed against the data entities of the selected dataset based on the preferences of the data source provider. In particular, within the data cleaning rules, the data validation rules are defined and are translated to a set of constraints that the specific data entity should conform with. The list of data validation rules includes the following:

- Conformance to a data type (i.e., Boolean, Integer, String, etc.)
- Conformance to a list of acceptable values (i.e., “Yes” or “No”)
- Conformance to a value range (i.e., the minimum and maximum acceptable values)
- Conformance to a value representation format (i.e., all dates are following the YYYY-MM-DD format)
- Conformance to a value uniformity (i.e., all time-stamps are in UTC format)
- Conformance to uniqueness (i.e., duplicate values are not acceptable)
- Conformance to non-empty value (i.e., all mandatory fields should have values)
- Conformance to cross-field validity (i.e., the sum of fields with percentage values must be equal to 100)
- Conformance to cross-field dependency (i.e., in case the field is set to a value then the other field should be set to a value)

In this sense, the *ValidatorService* implements the class that encapsulates the functionality of checking the input dataset against a set of rules and collect all identified errors.

- ***validate_data (df: DataFrame, rules: Dict)**:* This is the main function of the class. It accepts as input a pandas DataFrame representing the dataset to be validated, as well as a set of rules to verify and by consulting the internal functions of the same class, it gives as output another DataFrame which contains the violations that were detected.
- ***check_pre_defined_values (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for any violations regarding pre-defined values (closed list of values).

- ***check_range (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for any violations regarding numeric ranges.
- ***check_outliers_iqr (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for any numeric outliers, by using the Interquartile Range (IQR) metric.
- ***check_duplicate_values (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for any duplicate values in a given column.
- ***check_regex_pattern (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset against a given regular expression.
- ***check_date_format (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for compliance against a specific date format, for columns that represent date values.
- ***check_cross_field_operational (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for cross-field rules among different columns that need to comply to a specific numerical expression (e.g., column A + column B > 10).
- ***check_cross_field_conditional (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for cross-field rules among different columns that need to comply to a specific conditional expression (e.g., if column A is greater than 8, then column B should be equal to 0).
- ***check_string_length (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for compliance against a specific date format, for columns that represent date values.
- ***check_data_types (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for compliance to the given data types.
- ***check_missing_values (rules: Dict)**:* This is an internal function that accepts as parameter the validation rules and checks the input dataset for any missing values.

The *CleanserService* is the internal service that is responsible for the data correction operations that are performed on the selected dataset. The service provides a list of corrective operations which are performed against the conformance errors identified by the *ValidatorService*, based preferences of the data source provider, as defined in the data correction rules. The *CleanserService* offers, among others, the following data correction operations:

- Inconsistent value rejection and removal of a value from a record of a dataset.
- Inconsistent value rejection and removal of a complete record of a dataset.
- Inconsistent value replacement with a statistical value as the minimum or maximum value observed, the mean or median value or the most frequent value.
- Inconsistent value replacement with a specific value

In this sense, the *CleanserService* implements the service that provides the necessary functionality to apply the proper cleaning actions to the identified errors that came up during the validation stage.

- ***cleanse_data (df: DataFrame, rules: Dict, errors: DataFrame)**:* This is the main function of the class. It accepts as input a pandas DataFrame representing the dataset to be cleaned, as well as a set of rules and the validation errors identified. Then, by using the available internal functions, it uses the available cleaning rules to properly address the violated values.
- ***drop_rows (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and drops the rows that violate the given constraints.

- ***replace_with_value (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and replaces any violated values with a predefined value.
- ***replace_with_mean (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and replaces any violated values with the mean value of the column.
- ***replace_with_most_frequent (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and replaces any violated values with the most frequent value (mode) of the column.
- ***replace_with_max (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and replaces any violated values with the max value of the column.
- ***replace_with_min (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and replaces any violated values with the min value of the column.
- ***replace_with_median (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and replaces any violated values with the median value of the column.
- ***apply_date_time_format (rules: Dict)**:* This is an internal function that accepts as parameter the cleaning rules and applies the proper date format to any violated values.

The *CompleterService* is the internal service that undertakes the missing value handling operations that are performed against the selected dataset. The service provides a set of data imputation operations that are performed against the errors identified by the *ValidatorService,* which are related to non-empty value conformance as defined in the data completion rules by the data source provider. To this end, the *CompleterService* offers the filling of the missing values based on the following data imputation operations:

- Using statistical methods such as the minimum or maximum value observed, the mean or median value or the most frequent value
- Using the Linear Regression algorithm
- Using the k-Nearest Neighbours algorithm
- Using the Moving Average method
- Using the Last Observation Carried Forward (LOCF) and Next Observation Carried Backward (NOCB) methods
- Direct data imputation with a predefined value

Hence, the *CompleterService* implements the proper functionality that is needed in order to impute missing values that were identified during the validation stage.

- ***check_data_completeness (df: DataFrame, rules: Dict)**:* This is the main function of the class. It accepts as input a pandas DataFrame representing the dataset, as well as a set of rules and its goal is to properly impute any missing values across the different columns, based on the given rules and by making use of the various internal functions.
- ***drop_na (rules: Dict)**:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for dropping rows where a missing value was identified.
- ***fill_with_value (rules: Dict)**:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, using a constant value.
- ***fill_with_last_observation_carried_forward (rules: Dict)**:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, by using the Last Observation Carried Forward (LOCF) method.
- ***fill_with_next_observation_carried_backwards (rules: Dict)**:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, by using the Next Observation Carried Backwards (NOCB) method.
- ***fill_with_mean (rules: Dict)**:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, by using that column's mean value.

- ***fill_with_median (rules: Dict)****:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, by using that column's median value.
- ***fill_with_max (rules: Dict)****:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, by using that column's max value.
- ***fill_with_min (rules: Dict)****:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, by using that column's min value.
- ***fill_with_most_frequent (rules: Dict)****:* This is an internal function that accepts as parameter the missing value handling rules and is responsible for filling missing values in a specific column, by using that column's most frequent (mode) value.

The *LoggerService* is the complementary internal service that is utilised during the whole process execution in order to create and maintain the complete history records that contain the errors that were identified during the whole data cleaning operation process, as well as the corrective and missing value handling operations that were performed against these errors. Thus, the *LoggerService* offers the following internal interfaces:

- ***create_logs (profile_id: String, logs: DataFrame)****:* This function accepts as input the id of an existing cleaning profile as well as a DataFrame containing the logs gathered during the cleaning process, and creates a log file with the contents of that DataFrame.
- ***get_all_files ()****:* This function returns a list of all the available log files.
- ***log_contents (file_name: String)****:* This function accepts as input the name of a specific log file and returns a JSON object containing the contents of that file.
- ***delete_log_entry (file_name: String)****:* This function accepts as input the name of a specific log file and deletes that file.

The *LoggerController* class is a REST controller that exposes some functionalities regarding the logging to the outer world by using the proper functions of the LoggerService:

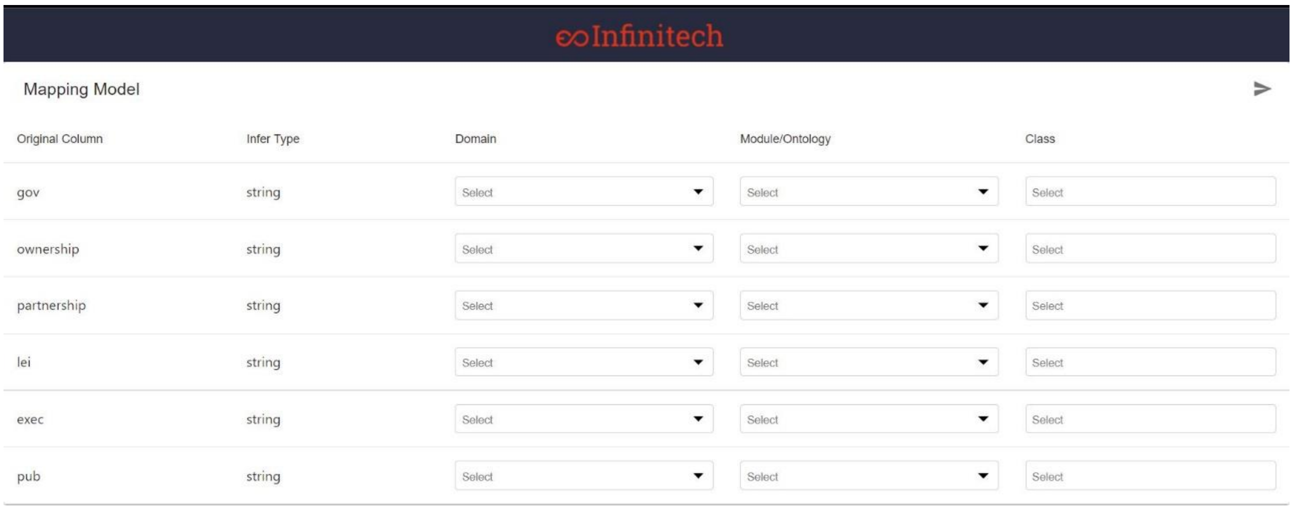
- ***get (file_name: String)****:* This controller-level function is used to fetch the contents of a specific log file, by invoking the **log_contents** function of the **LoggerService** class.
- ***delete (file_name: String)****:* This controller-level function is used to delete a specific log file, by invoking the **delete_log_entry** function of the **LoggerService** class.

2.4 The INFINITECH Data Collection Solution

As explained in section 2.1, the INFINITECH Data Collection provides via its rich set of offered functionalities the design, creation and execution of highly configurable data collection pipelines that can be either automated with the use of well-defined APIs thus leveraging the offerings only programmatically or incorporate manual steps that require human intervention and interaction with its easy-to-use user interface, as documented on section 2.2.4.

The first step of the data collection pipeline constitutes the registration of a new data source profile that contains all the information required in order for the INFINITECH Data Collection either to establish a connection to the data source (such as an API, a relational DB, an FTP or HTTP server or a HDFS deployment or MinIO storage server) to retrieve a new dataset or to receive a new dataset from the data source. In this step the Data Retrieval module of the INFINITECH Data Collection is leveraged. As explained in section 2.2.1 this is mainly a backend operation where a new data source profile is registered via the corresponding APIs of the INFINITECH Data Collection and the retrieval or reception of a new dataset is either executed on demand or in a periodical manner.

Once the new dataset is locally available by the Data Retrieval, the next step of the process is the mapping step which is executed by utilising the Data Mapper and the data model provided by the data provider. At first, the data provider is presented with a screen where the column name and infer type of each column included in the new dataset are extracted from the newly acquired dataset and presented to the data provider (Figure 31).

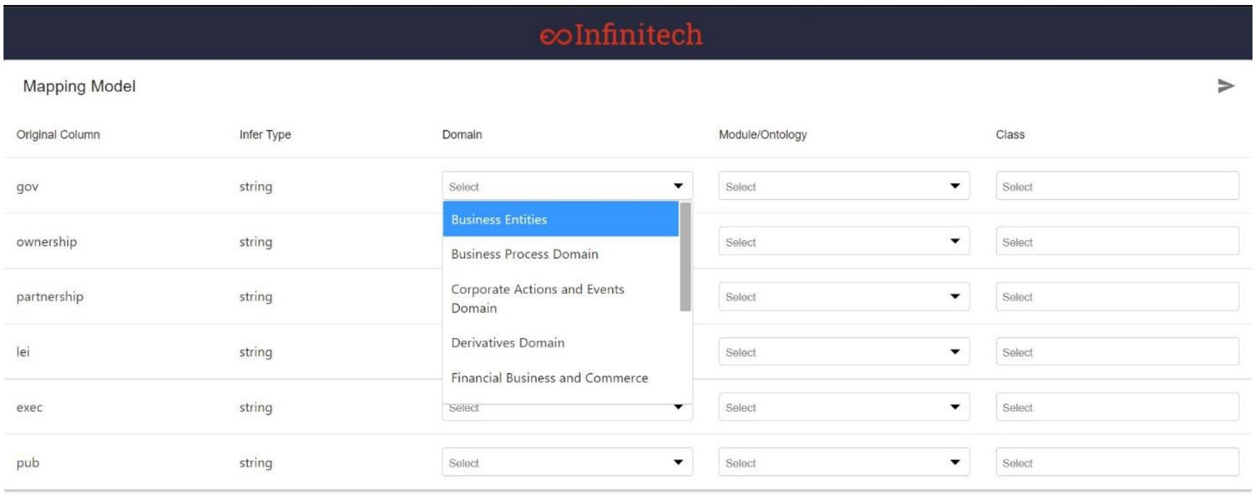


Mapping Model

Original Column	Infer Type	Domain	Module/Ontology	Class
gov	string	Select	Select	Select
ownership	string	Select	Select	Select
partnership	string	Select	Select	Select
lei	string	Select	Select	Select
exec	string	Select	Select	Select
pub	string	Select	Select	Select

Figure 31: INFINITECH Data Collection – Mapping process

During this step, the data provider is able select for each column the appropriate domain, module/ontology and class of each extracted column. In particular, the Data Mapper extracts the information included in the provided by the data provider data model and lists the available domains, modules/ontologies and classes to the data provider in order to select the appropriate option. At first, the data provider selects the corresponding domain from the list of available domains (Figure 32) and once the domain has been selected the respective modules/ontologies of the specific domain are presented to the data provider (Figure 33). Once both the domain and the corresponding module/ontology of the specific domain have been selected by the data provider, the classes included in the selected module/ontology are presented to the data provider (Figure 33).

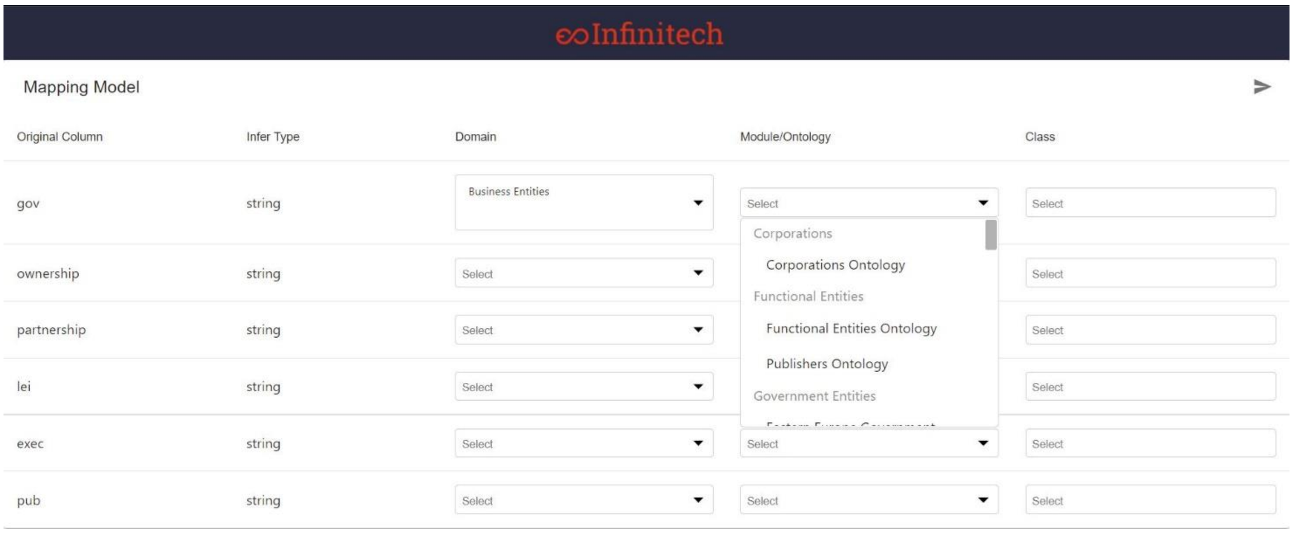


Mapping Model

Original Column	Infer Type	Domain	Module/Ontology	Class
gov	string	Select	Select	Select
ownership	string		Select	Select
partnership	string		Select	Select
lei	string		Select	Select
exec	string	Select	Select	Select
pub	string	Select	Select	Select

Business Entities
Business Process Domain
Corporate Actions and Events Domain
Derivatives Domain
Financial Business and Commerce

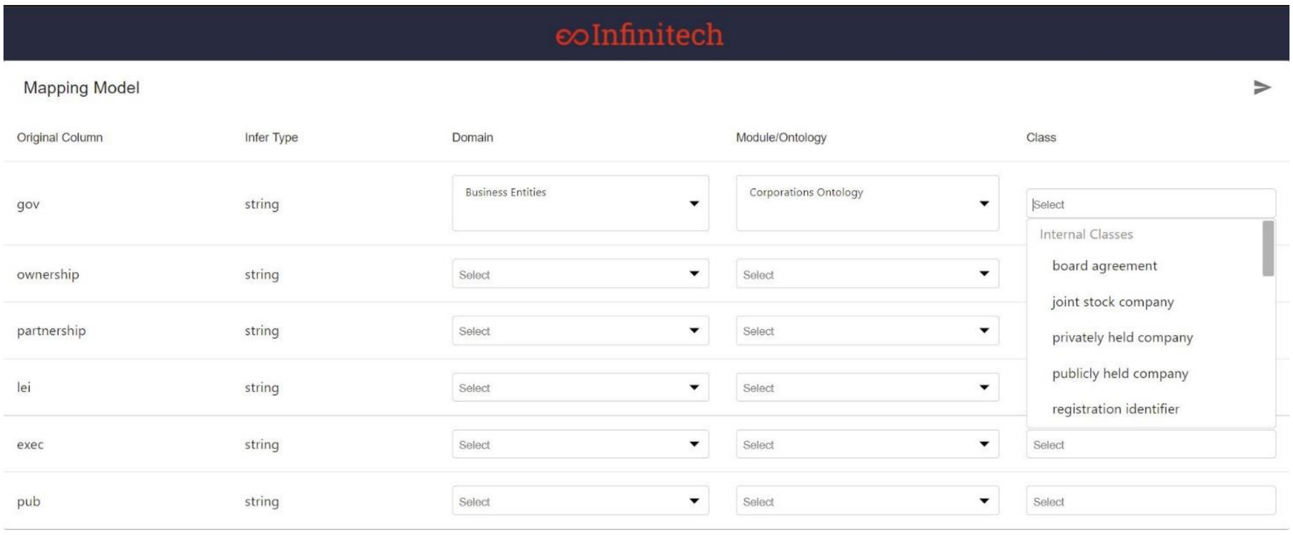
Figure 32: INFINITECH Data Collection – Mapping process (domain selection)



Mapping Model

Original Column	Infer Type	Domain	Module/Ontology	Class
gov	string	Business Entities	Select	Select
ownership	string	Select		Select
partnership	string	Select		Select
lei	string	Select		Select
exec	string	Select	Select	Select
pub	string	Select	Select	Select

Figure 33: INFINITECH Data Collection – Mapping process (module/ontology selection)



Mapping Model

Original Column	Infer Type	Domain	Module/Ontology	Class
gov	string	Business Entities	Corporations Ontology	Select
ownership	string	Select	Select	
partnership	string	Select	Select	
lei	string	Select	Select	
exec	string	Select	Select	Select
pub	string	Select	Select	Select

Figure 34: INFINITECH Data Collection – Mapping process (class selection)

The data provider is able to the select the appropriate information for each column of the dataset and thus create the mapping between the entities of the dataset and the underlying data model. The generated mapping can be saved and exported in JSON format. An additional option is offered by the Data Mapper in the case where the Data Retrieval is not utilised in the initial step and the dataset is already available locally on a MinIO storage server or as a file on the local filesystem. In that case, the data provider is instructed to provide either file containing the dataset (by a drag and drop option) or to provide the MinIO URL from which the dataset can be directly retrieved (Figure 35, Figure 36).

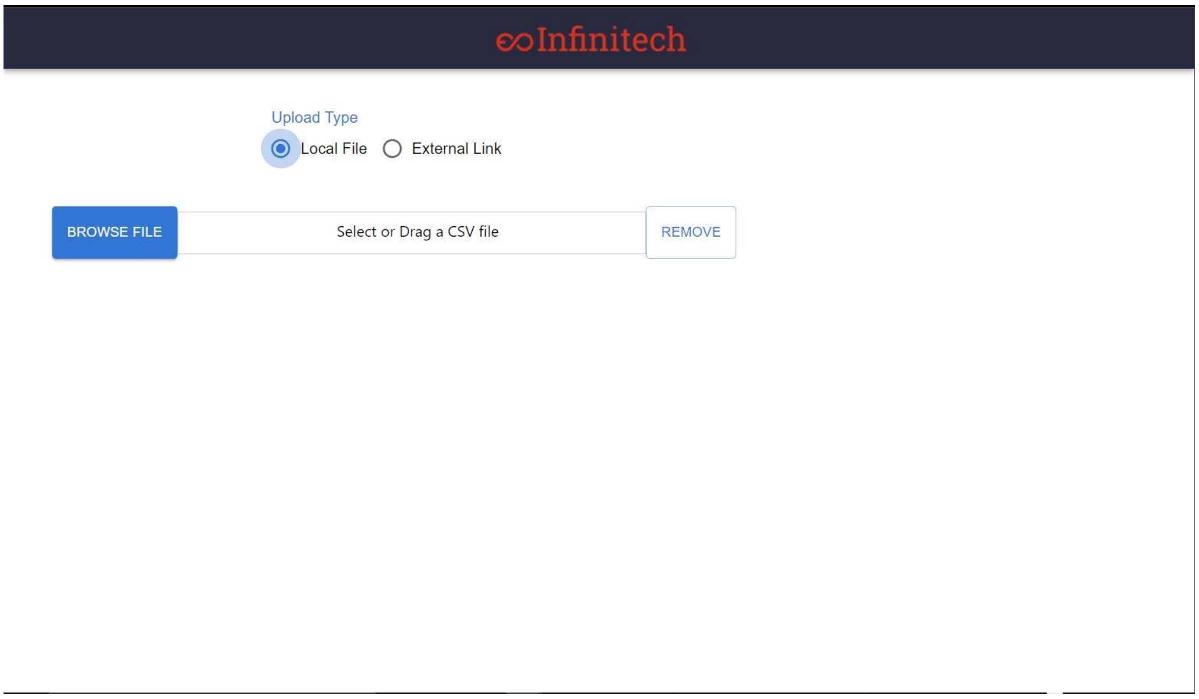


Figure 35: INFINITECH Data Collection – Mapping process (local file selection)

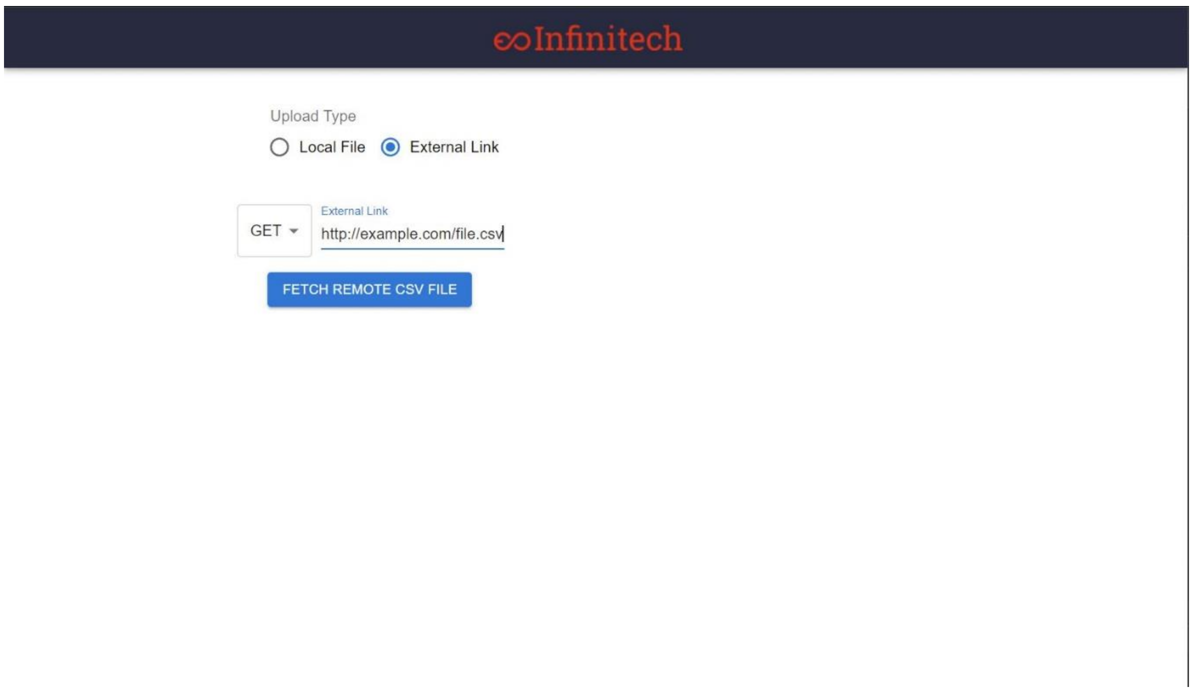
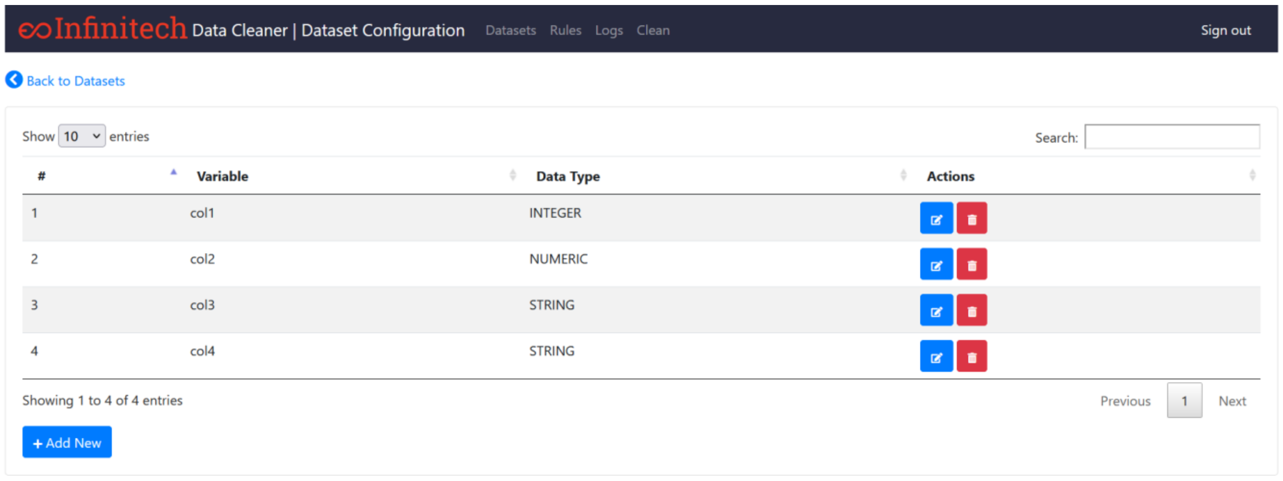


Figure 36: INFINITECH Data Collection – Mapping process (MinIO URL)

In the third and final step, the data provider is able to explore the capabilities of the Data Cleaner of the INFINITECH Data Collection in order to perform cleaning operations on new dataset. To achieve this, as explained in section 2.2.3, the data provider should set the validation, the cleaning and missing value handling rules that will be applied on a column basis. At first, in a similar manner as in the Data Mapper, the data provider is presented with a screen where the column name of each column included in the new dataset is extracted from the newly acquired dataset. At this step, the data provider should set the corresponding data type of each column as well as to select for which columns cleaning operations will be performed (Figure 37).

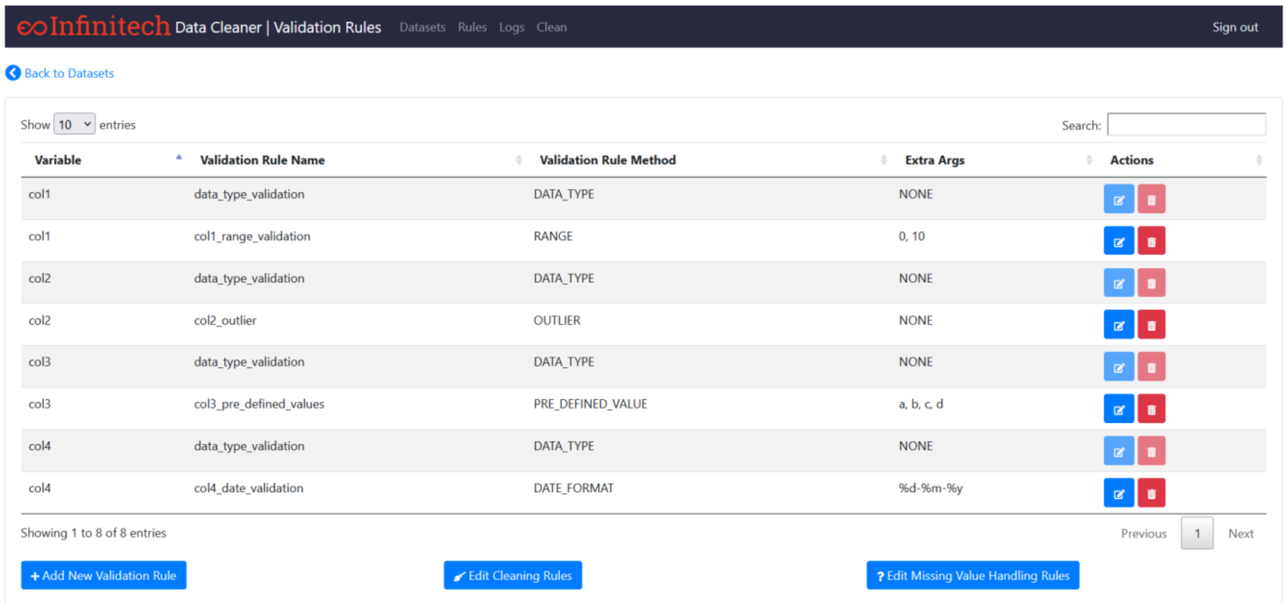


#	Variable	Data Type	Actions
1	col1	INTEGER	
2	col2	NUMERIC	
3	col3	STRING	
4	col4	STRING	

Showing 1 to 4 of 4 entries

Figure 37: INFINITECH Data Collection – Cleaning process (column type definition and selection)

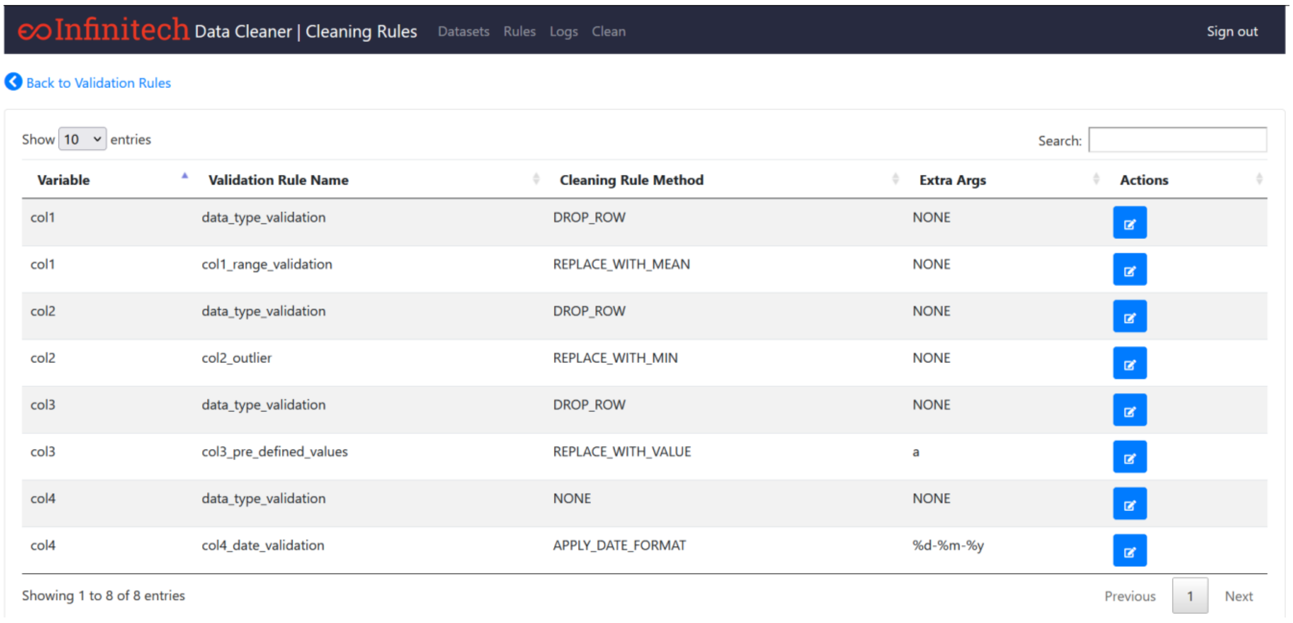
In the next step, the data provider should define the validation rules for the selected columns by selecting the validation rule method from the list of available methods and setting the corresponding parameters depending on the selected validation rule method (Figure 38). In a similar manner, the data provider selects the cleaning rules for each validation rules that was set in the previous step by selecting the cleaning rule method and by setting the required arguments (Figure 39). In the third and final step, the data provider configures the missing value handling rules for the selected columns by selecting the corresponding method and by setting the required arguments (Figure 40).


Infinitech Data Cleaner | Validation Rules
Datasets
Rules
Logs
Clean
Sign out
Back to Datasets
Show 10 entries
Search:
Previous
1
Next
+ Add New Validation Rule
Edit Cleaning Rules
? Edit Missing Value Handling Rules

Variable	Validation Rule Name	Validation Rule Method	Extra Args	Actions
col1	data_type_validation	DATA_TYPE	NONE	
col1	col1_range_validation	RANGE	0, 10	
col2	data_type_validation	DATA_TYPE	NONE	
col2	col2_outlier	OUTLIER	NONE	
col3	data_type_validation	DATA_TYPE	NONE	
col3	col3_pre_defined_values	PRE_DEFINED_VALUE	a, b, c, d	
col4	data_type_validation	DATA_TYPE	NONE	
col4	col4_date_validation	DATE_FORMAT	%d-%m-%y	

Showing 1 to 8 of 8 entries

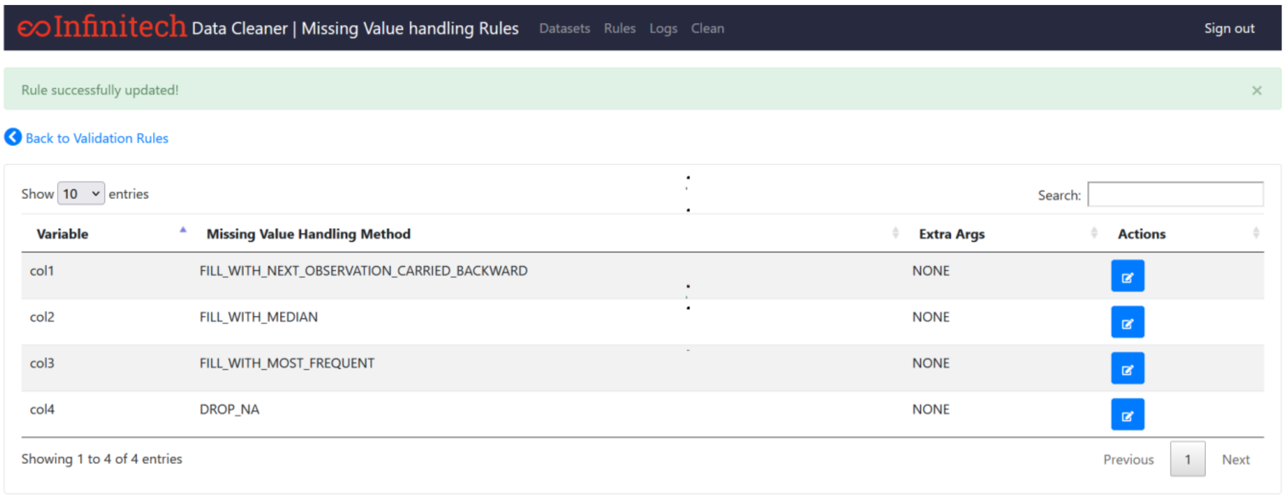
Figure 38: INFINITECH Data Collection – Cleaning process (validation rules)



Variable	Validation Rule Name	Cleaning Rule Method	Extra Args	Actions
col1	data_type_validation	DROP_ROW	NONE	
col1	col1_range_validation	REPLACE_WITH_MEAN	NONE	
col2	data_type_validation	DROP_ROW	NONE	
col2	col2_outlier	REPLACE_WITH_MIN	NONE	
col3	data_type_validation	DROP_ROW	NONE	
col3	col3_pre_defined_values	REPLACE_WITH_VALUE	a	
col4	data_type_validation	NONE	NONE	
col4	col4_date_validation	APPLY_DATE_FORMAT	%d-%m-%y	

Showing 1 to 8 of 8 entries

Figure 39: INFINITECH Data Collection – Cleaning process (cleaning rules)



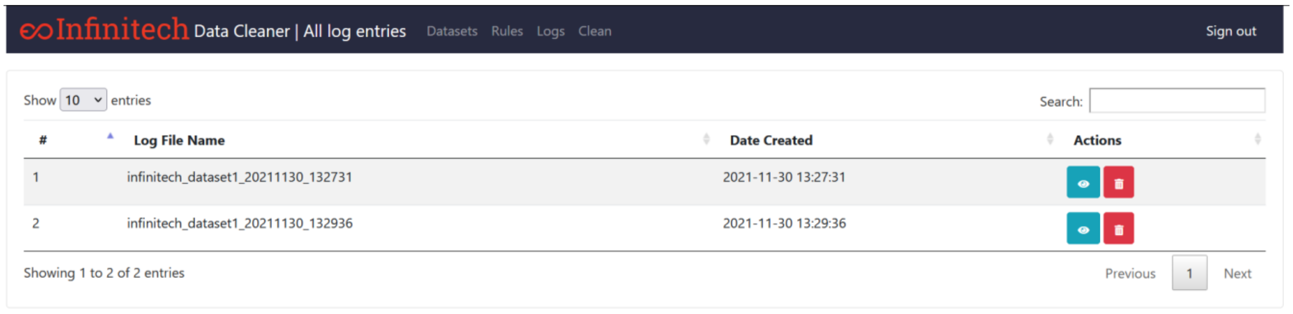
Rule successfully updated!

Variable	Missing Value Handling Method	Extra Args	Actions
col1	FILL_WITH_NEXT_OBSERVATION_CARRIED_BACKWARD	NONE	
col2	FILL_WITH_MEDIAN	NONE	
col3	FILL_WITH_MOST_FREQUENT	NONE	
col4	DROP_NA	NONE	

Showing 1 to 4 of 4 entries

Figure 40: INFINITECH Data Collection – Cleaning process (missing value handling rules)

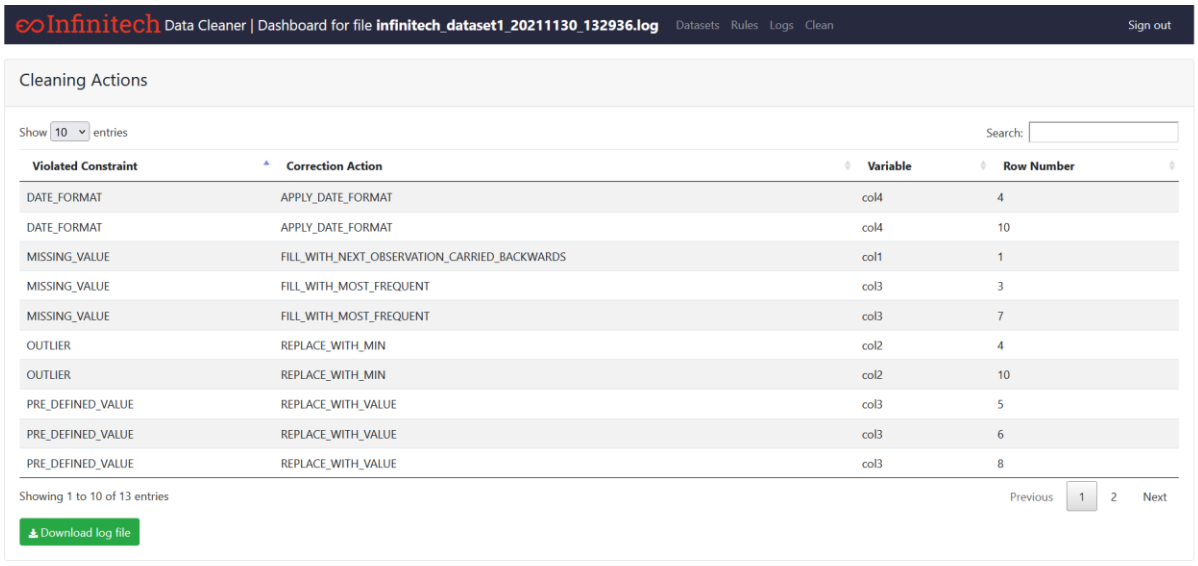
Upon the successful configuration of the required rules on the Data Cleaner, the data provider can execute the designed cleaning operations and check the execution results (Figure 41). In particular, for each execution result the Data Cleaner preserves and presents the complete history of the executed operations (constraints violated, corrective actions taken on a column and row basis) as well as some summary statistics of the executed operations (Figure 42, Figure 43).



#	Log File Name	Date Created	Actions
1	infinitech_dataset1_20211130_132731	2021-11-30 13:27:31	
2	infinitech_dataset1_20211130_132936	2021-11-30 13:29:36	

Showing 1 to 2 of 2 entries

Figure 41: INFINITECH Data Collection – Cleaning process (Execution Logs)


Infinitech Data Cleaner | Dashboard for file infinitech_dataset1_20211130_132936.log
Datasets
Rules
Logs
Clean
Sign out
Cleaning Actions
Show 10 entries
Search:
Showing 1 to 10 of 13 entries
Previous
1
2
Next
Download log file

Violated Constraint	Correction Action	Variable	Row Number
DATE_FORMAT	APPLY_DATE_FORMAT	col4	4
DATE_FORMAT	APPLY_DATE_FORMAT	col4	10
MISSING_VALUE	FILL_WITH_NEXT_OBSERVATION_CARRIED_BACKWARDS	col1	1
MISSING_VALUE	FILL_WITH_MOST_FREQUENT	col3	3
MISSING_VALUE	FILL_WITH_MOST_FREQUENT	col3	7
OUTLIER	REPLACE_WITH_MIN	col2	4
OUTLIER	REPLACE_WITH_MIN	col2	10
PRE_DEFINED_VALUE	REPLACE_WITH_VALUE	col3	5
PRE_DEFINED_VALUE	REPLACE_WITH_VALUE	col3	6
PRE_DEFINED_VALUE	REPLACE_WITH_VALUE	col3	8

Figure 42: INFINITECH Data Collection – Cleaning process (Detailed logs)

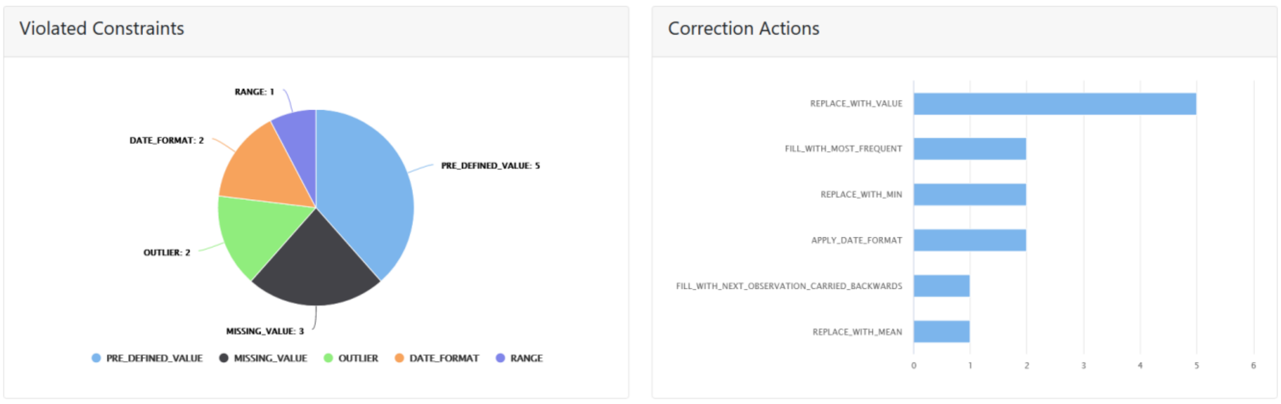


Figure 43: INFINITECH Data Collection – Cleaning process (Summary statistics)

The specific section presented the implemented solution from the user's perspective when the provided user interfaces are leveraged. However, as explained in the previous sections the complete solution can be leveraged to design and execute a data collection pipeline which can be operated in a fully automated manner, in which case the whole process is executed via the use of APIs without any manual intervention.

3 INFINITECH Synthetic Datasets

The particular section remained unchanged from the previous version. It presents the key characteristics of the synthetic datasets while also defining their role within the INFINITECH project.

In the age of Big Data, a large variety of data sources produce data at an exponential level. Public and private organisations are struggling to effectively collect all these enormous amounts of data that contain valuable detailed information that researchers and decision makers need in order to perform analyses, formulate predictions, evaluate their strategies and ultimately solve both simple or complex problems. Despite the increased availability of datasets and the acknowledged request for unrestricted availability of datasets imposed by the researchers and decision makers, the preservation of the privacy aspects of the individuals whose sensitive and private information is often included in the datasets, is equally important [1].

Thus, several data privacy-preserving techniques, such as pseudonymization, anonymisation, pseudo anonymisation, masking techniques, are employed on datasets in order to cope with the privacy concerns. However, besides the pure need for availability of datasets there is also the need for reliable, relevant and adequate data that meet certain characteristics, which are critical for the aspired analysis. The data privacy preserving techniques are posing several restrictions on these required characteristics, hence the quality of the utilised datasets is in most cases highly compromised. Nevertheless, the generation of Synthetic Datasets, in which private and sensitive data in the original dataset are replaced with synthetic data, is a viable alternative to the data privacy preserving techniques.

3.1 The characteristics of Synthetic Datasets

Synthetic datasets are datasets that contain artificially generated data instead of real data which are usually generated with the help of algorithms and a variety of data modelling techniques. In recent years, synthetic data generation has gained focus and significant effort has been invested in research for this topic, not only for its effective usage in a privacy preserving manner, but also for its effectiveness to support validation of new algorithms and applications which require data that are either not available or not accessible due to privacy concerns [2].

One of the major advantages of synthetic data is that they can be made publicly available with minimum risk of data disclosure and maximum utility [1], since the data included is randomly generated with constraints to hide sensitive private information and retain certain statistical information or relationships between attributes in the original data [2]. Hence, synthetic datasets can eliminate the barriers in numerous cases when privacy requirements limit data availability, the way data can be used and shared between multiple third parties in a collaborative nature thus effectively support research and innovation, as well as decision making.

Another major advantage of synthetic datasets is that they are generated in a such way that they address specific needs or conditions and with specific characteristics that are not available in existing (real) data. Synthetic datasets are often generated by exploiting several algorithms in a manner that enables more flexibility on the data manipulation aspect towards the effective testing of a broader range of conditions and use cases. In detail, synthetic datasets generation enables [3]:

a) the control over the data distributions used for the testing and validation of an algorithm's performance,
b) the fair performance comparison between different algorithms and
c) the creation of data records with the finest level of granularity in each attribute.

Hence, synthetic data can be valuable to a wide range of activities, including effective testing of new software systems and applications, training and validation of machine learning models, as well as all the cases where data are needed but they are not available, are too expensive to be generated as real data, or do not exist at all.

In general, synthetic datasets can be classified into the following three main categories:

a) **Fully Synthetic Data:** The fully synthetic data does not contain any real data. For this reason, re-identification of any individual is almost impossible, while at the same time it is ensured that all variables of the datasets are fully available. In this case, during the fully synthetic data generation, the procedure that is followed utilizes multiple imputation in order to replace the values of certain attributes for all the data points in the dataset [1]. The process includes multiple steps in which the density function of attributes in real data is identified and the parameters of these functions are estimated, before generating a privacy protected series of randomly selected values from these estimated density functions [2].
b) **Partially Synthetic Data:** The partially synthetic data contain real data and only the sensitive data is replaced with synthetic data. Hence, only data that raise risk of personal or sensitive information disclosure is replaced. Their generation is largely dependent on the utilised imputation model and the risk of disclosure is higher than in fully synthetic data, as real data is still included in the dataset. To generate the synthetic values for the selected attributes, multiple imputation and model-based techniques are also used [2].
c) **Hybrid Synthetic Data:** The hybrid synthetic data are generated utilising a limited volume of real data or synthetic data that were generated by domain experts. In the course of hybrid synthetic data generation, the distribution of the real data is analysed and the nearest record in the synthetic data is chosen, while ensuring both the relationship and the integrity between other attributes of the dataset. Hybrid synthetic data holds the advantages of both fully and partially synthetic data, providing adequate privacy preservation with high utility compared to fully synthetic and partially synthetic data, but at the cost of more memory and processing time [2].

In order to benefit from the usage of the synthetic datasets, it is important to possess a thorough knowledge of the domain for which data will be generated, as well as to follow a set of principles during the generation process, and finally to choose the correct methodology for the generation process, depending on the needs that these datasets will cover. During the synthetic data generation, a set of principles and restrictions are implied. Synthetic datasets are domain-dependent and for this reason it is crucial to utilise a real dataset during the synthetic data generation process, in order to ensure the properties of the dataset are satisfied. For this reason, a good understanding of the domain for which data are generated is also required [2]. Furthermore, both the domain and the data type that are generated are significantly affecting the complexity of the required synthetic data generation process.

Synthetic datasets have also restrictions as it is the case of real data. It should be acknowledged that during the synthetic data generation process, specific attributes of the data are only replicated, hence in principal general trends are simulated. While fully synthetic data has strong resistance to disclosure risk, these data lack truthfulness [2]. In addition to this, most of the times fully synthetic data are extremely useful for research purposes, however their usage in commercial products is usually limited. On the other hand, the support of partially synthetic data generation is very limited from the available techniques as most of the techniques that are available are mostly suitable for fully synthetic data generation. Finally, the complexity of the process is strongly dependent on the type of data that should be replicated, as for example static data is usually less challenging than streaming data for which the distribution of data is not usually known beforehand.

The complexity of the synthetic data generation process is highly dependent on the domain for which data should be synthesized, as well as the usage purpose of the generated synthetic datasets. As such, usually it goes beyond the simplistic solution of drawing numbers from a distribution that is created either based on the observation of the existing distribution from real data or from a comprehensive understanding of how the distribution would be like in the dataset in the case where real data do not exist. Usually, data modelling techniques are applied with the aim of replicating the required data. The existence or not of real data practically drives the decisions related to the data modelling technique. In the first case, the real data are exploited in order to generate fully synthetic or partially synthetic data by extracting the characteristics of the data and modelling them using usually probability density functions in order to perform multiple imputation. In the latter case where real do not exist or are not available due to privacy restrictions, the domain expert should define rules, constraints and relationships to effectively model the data, thus it is imperative to have extensive knowledge of the domain for which data will be generated [2]. In both cases, machine learning and deep learning models are utilised in the process in order to perform the multiple imputation, treating the values of the selected attributes as missing values which are generated using models such as Decision Trees, Random Forest, Support Vector Machine and Generative Adversarial Networks, and more [1].

The synthetic data generation process is facilitated by a large variety of libraries, frameworks and tools that the domain experts can utilise based on their needs and preferences. The most dominant libraries and frameworks are based on the Python and R programming languages, while there are also libraries and frameworks offered in other programming languages such as Java.

Scikit-learn[3] is the python well-established ML library that offers an extensive list of ML algorithms which are exploited in a large variety of synthetic data generation processes, such as the regression problem generation, classification problem generation, clustering problem generation, anisotropic cluster generation, concentric ring cluster data generation and moon-shaped cluster data generation. On the other hand, when it comes for categorical data generation, pydbgen[4] is a lightweight python library that is capable of generating a large database with multiple tables, filled with meaningful yet random data which can be ingested into a database, loaded as a dataframe in Pandas[5] or saved as an Excel file for further processing. Synthpop[6] is also a popular R library utilised for producing synthetic versions of microdata containing confidential information enabling their sharing and exploratory analysis. DataGenerator[7] is a Java based library that is capable of producing large volumes of data, framing data production as a modelling problem, with a user providing a model of dependencies among variables and the library traversing the model to produce relevant datasets. Another example is several online open-source or commercial platforms that enable the generation of realistic synthetic (usually structured) datasets which are mostly utilised for testing purposes such as Mockaroo[8], GenerateData[9] and OnlineDataGenerator[10]. In addition to these libraries and online tools, a plethora of candidate libraries and frameworks are available for usage depending on the scope of the synthetic data generation process and the preferences of the domain experts.

3.2 The role of Synthetic Datasets in INFINITECH

The financial and insurance sectors generate data in massive and increasing volume with high velocity and in a large variety of data types and formats. Despite the availability of these data, multiple restrictions and barriers are imposed to their usage and sharing due to the nature of the personal and sensitive information that is included on these datasets. As both sectors are highly regulated with strict legislations and processes, these collected data are mostly stored in silos within the organisations and severe limitations are applied when it comes to processing and sharing them even within the same organisation's different departments and especially outside the organisation's boundaries. As a consequence, the various research and business development activities are facing the lack of data that will enable them to perform the required and effective analysis, prediction generation and strategy evaluation.

[3] Scikit-learn, https://scikit-learn.org/stable/

[4] pydbgen, https://pydbgen.readthedocs.io/en/latest/

[5] Pandas Python, https://pandas.pydata.org/

[6] Synthpop, https://www.synthpop.org.uk/

[7] DataGenerator, https://finraos.github.io/DataGenerator/

[8] Mockaroo, https://www.mockaroo.com/

[9] GenerateData, http://generatedata.com/

[10] Online Data Generator, https://www.onlinedatagenerator.com/

Towards this end, the synthetic data generation approach is providing an appealing solution to overcome these restrictions and the problem of data unavailability in the finance and insurance sectors, as it ensures the required privacy preservation and at the same provides the means to the researchers and decision makers to perform the research and business development activities. In this sense, within the context of INFINITECH the synthetic data generation will be leveraged in order to overcome the aforementioned difficulties where needed and to facilitate the design and implementation of innovative financial and insurance services. In particular, the use cases and motivation for synthetic data generation, which are applicable in the finance and insurance sectors and are taken under consideration in INFINITECH, can be described in the following axes [4]:

- **Internal data use restrictions**: The imposed privacy requirements that limit data availability and prevent data sharing internally within a finance or insurance organization can be mitigated with the use of synthetic datasets.
- **Lack of historical data**: In some cases, the lack or limited availability of historical data which are needed in order to effectively perform the required analysis is posing limitations. Hence, with the use of synthetic data, these barriers can be removed.
- **Class imbalance**: The datasets which are utilised in several use cases, such as fraud detection, are imbalanced hence traditional machine learning techniques and anomaly detection techniques will often fail. Thus, synthetic data with more realistic attribute values generated with the appropriate data imputation techniques are characterized as more suitable for the aspired analysis.
- **Training advanced ML models**: Training of advanced machine learning models require vast amount of data, which should also probably be transferred into the appropriate infrastructure that is capable of performing such computationally intensive operations and might be outside of the boundaries of the organisation. Hence, the usage of synthetic datasets resolves both the need for large volume of data and the imposed privacy requirements.
- **Data Sharing**: The main restriction of the data originating from the finance and insurance sectors is that data sharing is severely restricted. Thus, data sharing between the organisations or within the research community which will enable the design, experimentation and implementation of innovative financial and insurance services, is very limited. However, the nature of synthetic data offers the potential to meet the imposed data sharing restrictions and enable the research and innovation.

Within the context of INFINITECH, several pilots exploit the benefits offered by the usage of synthetic datasets towards the validation of the INFINITECH offerings from both a technical/technological and a business/economic perspective. In this context, synthetic datasets are prepared by the pilot's domain experts in order to be exploited during the execution phase of each pilot. The format of the synthetic datasets will vary from semi-structured (*CSV, JSON, GeoJSON*) to unstructured formats (*txt, NetCDF*) depending on the needs of each pilot. In the same manner, the volume of each synthetic data that will be exploited will vary from 10 MB per executed use case to 20 GB, depending on the executed scenario.

The main categories of the synthetic data that will be utilized within the context of the INFINITECH Pilots are the following:

a) Customer Profiles
b) Customer Accounts
c) Customer Data
d) Financial Transaction Data
e) Connected Car Data
f) Traffic Data
g) Activity Tracking Data
h) Crop Biophysical Parameters and Loss Data
i) Seasonal Crop Yield Prediction Data
j) SME Company Profile Data

These synthetic data are available in the respective testbed of each pilot that exploit them. The data collection process that will be utilised in order to ingest the respective synthetic data along with any real data that will exploited into the testbed of each pilot is presented in Section 4. The list of synthetic datasets of each pilot along with in-depth details is described in Appendix B**.**

4 INFINITECH Pilot Use Cases

Updates from D5.13:
The particular section received several updates in order to document the advancements on the data collection process of the pilot from the previous version. Besides the several optimisations and updates that were introduced across the whole section, the following updates should be highlighted:

- *The description of the data collection process of two additional pilots (Pilot 7 & Pilot 15) was introduced*
- *The description of one pilot (Pilot 1) has been removed as the specific pilot has been deprecated from the project as part of the latest amendment process*

The scope of the current section is to provide the updated documentation of the details of the datasets, both real and synthetic, that are collected by the INFINITECH pilots in order to be harmonized, anonymised and ingested into the underlying storage. The list of collected dataset vary based on the extended list of diverse scenarios from the finance and insurance sectors that the INFINITECH pilots are implementing. Hence, different approaches are followed within the context of each depending on their scope and the peculiarities of each scenario. It should be noted that the complete documentation with regards to the data collection processes that are followed on each pilot and their implementation are thoroughly documented within the deliverable D7.2.

In the following paragraphs, the focus is on presenting the updated and final list of the datasets that are leveraged by the pilots, accompanied by the details for the information included, their data format and the anonymisation needs, where applicable. These datasets are collected by the data collection process of each pilot as documented in D7.2.

It should be noted that the description of the data collection processes of Pilot#1 and Pilot #3 is not included in the current version of the deliverable for various reasons. On the one hand, Pilot #1 has been deprecated from the project as part of the latest amendment process at the moment of writing the deliverable, while on the other hand Pilot #3 is currently in redesign state.

4.1 Pilot #2 - Real-time risk assessment in Investment Banking

The scope of Pilot #2 is to design and implement a real-time risk assessment and monitoring procedure that aims to facilitate the generation of risk information for asset management with two standard metrics, namely the VaR (Value-at-Risk) and the ES (Expected Shortfall). To this end, the pilot performs the measurement of market risks of assets portfolios, while also evaluating what-if scenarios for pre-trade analysis.

With regards to the datasets that are exploited, the pilot mainly leverages Foreign Exchange (FOREX) market data in order to calculate the Value-at-Risk (VaR) for various portfolio compositions. In detail, two main types of input datasets are used, the first one includes trades associated with a portfolio composition to be analysed ("TradeData"), while the second one includes price data ("TickData") regarding the FOREX market. The pilot also leverages alternative data, such as derived analysis data and news data, that are used in the performed analysis. All data types have quite simple structures and are presented in the table below:

Table 25: Pilot #2 List of datasets

Dataset Name	Dataset description	Data format	Anonymization
TradeData	Trades associated with a portfolio composition. The TradeData will comprise the following three columns:	CSV / Real-time Data stream	No anonymisation

	▪ SymbolID, i.e. the name of the instrument in FOREX trading e.g., GBPUSD for the exchange of GBP to USD ($) ▪ Timestamp (in UNIX format) denotes when the trading took place. ▪ Quantity, i.e. the amount traded which will be a negative number in case of selling and a positive number in case of buying.		
TickData	High frequency (~1 sec) ticker data. Tick data is associated with any change in the security price, whether that movement is up or down. The given dataset comprises tick data of major Forex instruments such as EUR/USD, GBP/USD, EUR/CHF and EUR/CAD for the period 01/04/2020 - 30/09/2021. The dataset has the following columns: ▪ SymbolID, i.e., the name of the instrument in FOREX trading, e.g., GBPUSD for the exchange of GBP to USD ($), ▪ Timestamp (in UNIX format), ▪ Open, ▪ High, ▪ Low, ▪ Close. Columns 3-6 denote the price (e.g., open price) of the instrument for the given timestamp.	CSV / Real-time Data stream	No anonymisation
Derived analysis data	Daily Value at Risk and Expected Shortfall estimations based on the input Trades and Tick data	CSV	No anonymisation
News articles and Twitter data	Open source sample data for Market Sentiment Analysis	Text	No anonymisation

To facilitate the process, both TradeData and TickData datasets are provided in two flavours, the Real-time market feeds associated with the real-time operation of the market and the historical datasets that are used for various calculations. The market data are real data, retrieved from a financial data service provider while the trade data are the trading signals generated by JRC's algorithmic trading strategies, and no real trades are executed. The data are accessible to their end-users (i.e., traders) and no anonymization processes will take place.

For testing and experimentation purposes existing datasets in CSV formats are used. Hence, the format of the data is CSV and real-time data stream containing text and numeric data related to public market data and synthetic portfolio composition data.

To support experimentation, validation and deployment in production, the data collection processes leverage data from the following data sources:

- Forex APIs, which are mainly used for testing and experimentation purposes. They provide access to both real-time and historical values of FX data.
- Data from the JRC Trading Platforms, which are used to provide real-time trading values with high ingestion rates.

- Data from the JRC Trading Datawarehouse, which provide access to historical information (e.g., close values) for the forex assets that are entailed in the pilot.

4.2 Pilot #4 Personalized Portfolio Management ("Why Private Banking cannot be for everyone?")

The scope of Pilot #4 is to develop and integrate within the SaaS based Privé Managers Wealth Management Platform an Optimization algorithm (further on called Privé Optimizer "AIGO"), as well as to improve and expand its capabilities as an artificial intelligence engine to aid investment propositions for retail clients. Hence, pilot #4 aims to exploit the capabilities of AI-Based Portfolio construction for Wealth Management processes in order facilitate the advisors and/or end-customers utilising the Privé Managers Wealth Management Platform to effectively consume the offered risk-profiling and investment proposal capabilities, starting from their personal risk-awareness.

The innovative AIGO genetic algorithm is capable of proposing investments which are in turn evaluated based on a set of fitness factors which are composed by easy-to-use, personalized set of criteria. Based on the fitness factors, "health" score will be generated for each portfolio that will drive the definition of the "fittest" investments. To achieve this, a series of datasets are leveraged which are presented in the following table:

Table 26: Pilot #4 List of datasets

Dataset Name	Dataset description	Data format	Anonymization
Customer Transactions Data	Customer Transactions Data fetched directly from the Bank or an Asset Manager. It consists of customer securities and cash transactions through their deposit accounts.	CSV	Confidential data
Financial Market Price Data	Financial Market Price Data fetched from several Market Data Providers. It consists of price data for Stocks, Bonds, Mutual Funds and or other assets like certificates/warrants.	Text	Open, partially license agreements with data providers needed
Financial Market Asset Master Data	Financial Market Asset Master Data fetched from several Market Data Providers. It consists of asset related characteristics (e.g. expiration date, minimum investment amount, asset class breakdowns).	Text	Open, partially license agreements with data providers needed
Customer Risk Profile Data	Customer Risk Profile Data fetched directly from the Bank or an Asset Manager. It consists of customer Risk Profile Data through their account data and profiling, based on B2B customers parameters.	CSV	Confidential data
Mutual Fund, ETF and Structured Products Breakdown	Mutual Fund, ETF and Structured Products Breakdown fetched from several Market Data Providers. It consists of asset breakdowns based on bank data or market data providers breakdown.	CSV	Open/Confidential data, partially license agreements with data providers needed
Customer Economic Outlook	Customer Economic Outlook fetched directly from the Bank or an Asset Manager based on questionnaires and Customer Profiles.	CSV	Confidential data

Risk metrics figures	Portfolio data is used to calculate portfolio metrics like portfolio performance, volatility, sharpe ratio and other risk figures.	CSV or customized	No need as they are synthetic.

With regards to the anonymisation of the aforementioned confidential datasets, no anonymisation is foreseen as the data are privately kept within the premises of Privé. Additionally, the input that is taken from these datasets in order to be fed to the optimisation process does not include any sensitive or private data.

4.3 Pilot #5b - Business Financial Management (BFM) tools delivering a Smart Business Advise

The scope of Pilot #5b is to provide the means to the SMEs which are clients of Bank of Cyprus (BoC) to effectively and efficiently manage their financial health, in multiple areas such as cash flow management, continuous spending/cost analysis, budgeting, revenue review and VAT provisioning by employing a set of AI-powered Business Financial Management tools and harnessing available data to generate personalized business insights and recommendations.

Hence, the multiple diverse datasets are collected and processed in the course of development of the specific pilot. In detail, most of the datasets that are utilized, are extracted from BoC databases and are related to financial transactions, account profiles and relevant data of BoC's SME customers. Furthermore, BoC's SME customers transaction data with other banks which are retrievable from Open Banking under PSD2 are also collected, as well as open-source macroeconomic data from sources like Eurostat or data.gov.cy.

The use of surrogate/analogous data is leveraged in the context of the CashFlow prediction service to increase the added-value and impact of the utilized deep learning probabilistic models, since they benefit from the increased diversity and volume of the training data. It is expected that the usage of analogous data will help the generalization of some, if not all deep learning models in the case of time series forecasting. Within the context of the pilot, the Iterated Amplitude Adjusted Fourier-Transformed (IAAFT) is used in order to produce the surrogate data that was then merged with the real ones in order to enhance the time-series historical transactions dataset.

The following table depicts the list of datasets that are used, providing for each dataset a short description, the respective data format and the anonymization requirements.

Table 27: Pilot #5b list of datasets

Dataset Name	Dataset description	Data format	Anonymization
Transaction Data from the Bank	SME Customers Transactions Dataset from BoC	CSV	Already anonymized
Transaction Data from Open Banking (PSD2)	SME Customers Transactions Dataset from financial institutions other than BoC. BATCH input (e.g. every 6 hours or every night)	CSV	Already anonymized
Accounts Data from Bank	Account data regarding SME customers of BoC. E.g. Balances, Available amount, account type	CSV	Already anonymized
Customer Data from Bank	Customer Demographics	CSV	Already anonymized
Other Data (Market)	Macroeconomic SME related data from public/private resources (e.g.	CSV / JSON	Open source data

	https://www.data.gov.cy/ & https://ec.europa.eu/eurostat/data/database)		
Other Data from SME	Other Data that is provided by the SME ERP/Accounting system (e.g. number of customers, suppliers, stock). Non- transactions related data	CSV/ JSON	Via the GRAD anonymization solution
Transaction Data from SME	Data that is provided by the SME ERP/Accounting system and relates directly or indirectly to account debit/ credit transactions. For instance, invoice data; non PSD2 data (i.e. payment account related) e.g. saving accounts in financial institutions other than BOC	CSV/ JSON	Via the GRAD anonymization solution
Surrogate data	Surrogate data created based on Iterated Amplitude Adjusted Fourier-Transformed IAAFT in order to enhance the time-series historical transactions dataset used for the Cash-Flow prediction service.	CSV	No need as they are synthetic.

As described in the table above, most of the datasets will already be anonymised before being imported to the INFINITECH platform. Data originating from BoC activities and databases are pseudonymized before being extracted to the platform using the tokenization approach, where sensitive data are being replaced with non-sensitive equivalents, which can only be reversed in the tokenization vault that resides on BoC premises.

4.4 Pilot #6 - Personalized Closed-Loop Investment Portfolio Management for Retail Customers

The scope of Pilot #6 is to design and deliver a personalized investment recommendations system tailored to the needs of the retail customers of NBG. In this process, a variety of diverse datasets from different data sources in large volumes is leveraged towards the aim of providing investment recommendations to retail customer more targeted, automated, effective, as well as context-aware (i.e., tailored to state of the market).

The datasets are generated from NBG Operational Databases in CSV/TXT format and are in a way anonymized, as all critical information is converted to specific IDs that the reference to the actual information is privately kept within the bank premises. The list of the datasets that are utilised is as follows:

Table 28: Pilot #6 List of datasets

Dataset Name	Dataset description	Data format	Anonymization
Deposit Account Transactions	Customers' transactions performed through their deposit accounts for the last two (2) years, for each deposit product account that the customer possesses. The Deposits Account transactions data include category amount, date, time and channel that were performed.	CSV / Text	Already anonymized
Cards Transactions	Customers' transactions performed through their cards for the last two (2) years, for each card product that the customer is a card holder. The Cards Transactions data include date, time, card type, merchant category, amount in euros and foreign currency, instalment or cash purchase, transaction type and channel (POS, e-commerce, ATM).	CSV / Text	Already anonymized

Instruments Historical Prices	Historical prices of investment instruments that NBG offers to the customers for the last two (2) years, including for each instrument the details of date, closing price, closing price currency, total transactions volume, market code, effective and ending date.	CSV / Text	Already anonymized
Investment Related Transactions	Customers' transactions related to investment products for the last two (2) years, containing date, time, account used, instrument, settled amount, settled currency, maturity date, quantity, instrument price, investment amount gross and net, financial market, investment transaction type.	CSV / Text	Already anonymized
Instruments Characteristics	Detailed characteristics for all instruments that will be considered in the pilot, including instrument type, instrument asset class, currency, ISIN, issue date, maturity date, instrument pieces (e.g. shares).	CSV / Text	Already anonymized
CRM Data	Customer related data like demographics, product ownership and responses to MIFID questionnaires. The data that will be available for the pilot include customer type, birth date, gender, marital status, child number, profession category, origin country, type of employment, customer risk category (bank's calculation engine).	CSV / Text	Already anonymized

4.5 Pilot #7 - Avoiding Financial Crime

The scope of Pilot #7 is to design and deliver a Data Model to identify fraudulent transactions. In essence, the pilot has chosen a banking service that lately has been used by fraudsters to commit fraud. This service is the immediate loan, which is a loan that doesn't need to be approved by the bank, but it is pre-approved based on the economic profile of the client. Fraudsters are now aware that although the client has no money in the account, they can ask for this immediate loan and commit fraud for a larger amount making the fraud even worse than before because the client suffer from the fraud of money they don't have and the fraudster leaves him with no money and a debt.

The following table depicts the list of datasets that are used, providing for each dataset a short description, the respective data format and the anonymization requirements.

Table 29: Pilot #7 List of datasets

Dataset Name	Dataset description	Data format	Anonymization
List of Immediate loans	Real list of all the immediate loans processed by internet from October 2020 to March 2021. The Dataset has been labelled with the fraudulent transactions therefore we can apply supervised algorithms to the model. In the Dataset there are all the fields describing an operation of this kind, and for this reason was necessary to anonymize the personal and sensitive data as this Dataset contained details of clients. Concretely, the following fields have been anonymized: - FK_NUMPERSO : hash - IP_TERMINAL : hash	CSV / Text	Anonymized

	- FK_NUMPERSO_TIT_LOE: hash - SALDO_ANTES_PRESTAMO: rounded to hundreds - FECHA_ALTA_CLIENTE: modified data to -100 days - FK_NUMPERSO: is a user identifier - IP_TERMINAL is the IP connection - FK_NUMPERSO_TIT_LOE is a identifier - SALDO_ANTES_PRESTAMO: Amount in the account - FECHA_ALTA_CLIENTE: Date of client's acquisition		

4.6 Pilot #8 - Platform for Anti Money Laundering Supervision (PAMLS)

The scope of Pilot #8 is to design and implement a platform that performs anti-money laundering Supervision (PAMLS) with the aim of enhancing the efficiency and effectiveness of the existing supervisory activities in the area of anti- money laundering and combating terrorist financing (AML/ CTF). Pilot #8 is leveraging the characteristics of Big Data which are owned by the Bank of Slovenia (BOS) and other competent authorities (FIU). To this end, the PAMLS platform offers, among others, a risk assessment tool, a screening tool, a search engine and a distributed channel.

Within the context of the specific pilot, multiple real datasets originating from various heterogeneous data sources are exploited. The relevant information of the aforementioned datasets is summarized in the following table:

Table 30: Pilot #8 list of datasets

Dataset Name	Dataset description	Data format	Anonymization
TARGET2 transactions	Transactions executed by the Slovenian payment institutions within TARGET2 (Trans-European Automated Real-time Gross Settlement Express Transfer System): ▪ high value (above 50.000 EUR), urgent transactions in EUR; ▪ transactions processed through BOS payment systems (responsible BOS Payment Settlement and Systems department - PPS). TARGET2 transactions include following data relevant for PAMLS: *transaction date, transaction value, payer data (transaction account no., name and address), payer bank data and BIC code, payee data (transaction account no., name and address), payee bank data and BIC code, intermediary bank data.*	XLSX (txt or numeric)	Anonymised
SEPA transactions	Transactions executed by the Slovenian payment institutions within SEPA (Single Euro Payments Area): ▪ domestic and international transactions within SEPA area in EUR under 50.000EUR value;	XLSX (txt or numeric	Anonymised

	▪ transactions processed through payment systems by third party provider. SEPA transactions include following data relevant for PAMLS: *transaction date, transaction value, payer data (transaction account no., name and address), payer bank data and BIC code, payee data (transaction account no., name and address), payee bank data and BIC code, intermediary bank data.*		
Slovene Financial Intelligence Unit (FIU) transactions (public data)	Transactions above 15.000 EUR, related to high risk countries and reported to the FIU.	XLSX (txt or numeric	Open Data
FI identification data	Identification information about Financial Institution (FI) that they send by report to the BOS (reports are confidential). The data are statistical data on the FI inherent risk and control environment (number of clients, number of Suspicious transactions reports (STR) etc.)	XLSX (txt or numeric	Not anonymised
ePRS data	Slovenian Business Register (public data on legal entities).	XLSX (txt or numeric	Open Data
eRTR data	Slovenian Transactions Accounts Register (public data on legal entities).	XLSX (txt or numeric	Open Data

It should be noted that due to the high confidentiality requirements, the transactions analysed within PAMLS datasets will not be transferred outside BOS premises and will not be shared on the INFINITECH platform.

Since the data included in both the TARGET2 and the SEPA transactions contain both personal and confidential data, it will be ensured that the data privacy and data confidentiality aspects are properly addressed. In this context, within the TARGET2 and SEPA transactions datasets, the payer data (transaction account number, name and address) and payee data (transaction account number, name and address) are anonymised prior to being delivered to PAMLS in order to eliminate the disclosure of personal or confidential information. With regards to the FIU transactions, the ePRS and eRTR datasets, there is no need for anonymisation as they are publicly available datasets provided by the FIU. In the same manner, the FI identification data owned by BOS do not require any anonymisation to be applied as they are compiled by statistical data with no personal or confidential data.

4.7 Pilot #9 - Analyzing Blockchain Transaction Graphs for Fraudulent Activities

The scope of Pilot #9 is to design and develop a parallel and scalable system that will facilitate the formulation of a massive Bitcoin and Ethereum blockchain transaction graph with distributed dynamic data structures, exploiting the capabilities of an HPC cluster. In this transaction graph, an analysis is performed utilising a variety of graph and machine learning algorithms in order to investigate and identify blockchain account transactions that can be traced to fraudulent activities or accounts.

Within the context of Pilot#9, massive actual public anonymous chain data which are available from Ethereum and Bitcoin chains are collected and utilised. These raw data are obtained from actual blockchain nodes or gateways or a service like Google BigQuery. Due to the nature of the blockchain technology, as well as the peculiarities of the data stored within a blockchain network, the collected blockchain datasets differ from the usual datasets. In particular, the Bitcoin raw blocks are parsed in order to be stored as files that contain the respective transactions, while the Ethereum raw blocks are firstly joined and then parsed in order to be stored as files also. The collected transactions are stored in bzip2 compressed files. The following table presents the details of the collected datasets.

Table 31: Pilot #9 list of datasets

Dataset Name	Dataset description	Data format	Anonymization
Ethereum Mainnet Blockchain Dataset	Blockchain data collected from Ethereum Mainnet containing: • Blocks from 0 to 10.199.999 • Time coverage from 30.07.2015 to 04.06.2020 • 766.899.042 transactions • 78.945.214 addresses • 43.371.941 of 40 Major ERC20 Token Transfer Transactions • Symbols of 40 Major ERC20 Tokens: USDT TRYb XAUt BNB LEO LINK HT HEDG MKR CRO VEN INO PAX INB SNX REP MOF ZRX SXP OKB XIN OMG SAI HOT DAI EURS HPT BUSD USDC SUSD HDG QCAD PLUS BTCB WBTC cWBTC renBTC sBTC imBTC pBTC • 21 GB zipped data or 81GB unzipped data	bzip2	No anonymisation
Bitcoin Blockchain Dataset	Bitcoin blockchain data containing: • Blocks from 0 to 674999 • Time coverage from 03.01.2009 to 17.03.2021 • 625.570.924 transactions • 800.017.678 addresses • 112 GB zipped data or 382 GB unzipped data	bzip2	No anonymisation

4.8 Pilot #10 - Real-time cybersecurity analytics on Financial Transactions’ BigData

The main objective of Pilot #10 is to significantly improve the detection rate of malicious events (i.e., frauds attempts) and enable the identification of security-related anomalies while they are occurring with the real-time analysis of the financial transactions of a home and mobile banking system.

Within the context of the pilot, the datasets that are used are related to SEPA bank transfer transactions. In detail, synthetic datasets that are consistent with the real data present are created in the data operations environment. During the synthetic data creation, models are built from real world data and domain knowledge. The data are synthetized using a set of generators which are based on a lightweight agent-based modelling framework while using inferred distributions. Dimensional data are generated either from classical parametric random generators, or from inferred non-parametric distributions. Every synthetic dataset is represented as a scenario, in which dimensional data are generated either from classical parametric random generators, or from inferred non-parametric distributions. The datasets are then produced as interactions between agents, which use the dimensional data as input to determine their behaviour. The starting point for the synthetic dataset creation is randomly generated personal data with no correlation with real people. The datasets are in CSV or ORC format with temporal coverage for the whole year 2019 and an estimated size of 3,5 GB and are treated as a confidential dataset. The following table documents the details of the collected datasets.

Table 32: Pilot #10 list of datasets

Dataset Name	Dataset description	Data format	Anonymization
Bank Transfer SEPA	Single Euro Payments Area (SEPA) transactions that cover predominantly normal bank transfers.	CSV or ORC	No anonymisation (fully synthetic datasets)

4.9 Pilot #11 - Personalized insurance products based on IoT connected vehicles

The main objective of Pilot #11 is to improve the risk profiles in car insurance by using the information collected from connected vehicles and applying Artificial Intelligence technologies. Within the context of this specific pilot, a "Pay as you drive" service capable of adapting the insurance costs to the actual driver's way of driving is developed. A "Fraud detection" service aiming to help insurance companies is also developed.

In order to prevent data protection issues, an Anonymization Tool that is offered by GRAD is used. Within Pilot #11, the Anonymization Tool is exploited in order to effectively address all the data privacy concerns related to the location privacy. Therefore, the specialised anonymisation algorithm offered by the tool is applied to the geolocated data with the aim of ensuring that user's exact spatial coordinates are not revealed as this could lead to a possible re-identification of the data subjects.

The open standard FIWARE NGSI serves as the basis for the implementation of the data gathering process within Pilot #11 and specifically an instance of its NGSIv2 interface. In this context, all data stored in the Connected Car Platform, that manages the data collection process of Pilot #11, are generated according to the **FIWARE Vehicle Data model** [5], the **FIWARE Alert Data model** [6] and the **FIWARE Weather Observed Data model** [7] and in NGSI format, so that they can be retrieved using the same standard and the whole process is aligned with the published ETSI NGSI-LD specification [8].

Table 33: Pilot #11 List of datasets

Dataset Name	Dataset description	Data format	Anonymization
Connected Vehicles	Data collected from connected vehicles (provided Automotive Technology Centre of Galicia – CTAG). CANBus + NMEA (location) from real connected vehicles.	CSV	GRAD Anonymisation Tool
Simulated Vehicles	Simulated Urban mobility data (mainly vehicles CAN Signals) generated through SUMO tool. One scenario reporting 30K vehicles (around 8 Gb of data)	JSON	No anonymisation (synthetic datasets)
Roads datasets	Roads data extracted from OpenStreetMap (2 Gb for considered scenarios).	JSON	N/A
Weather information	Collected from AEMET weather stations. 2k registries (around 1 Mb of data).	JSON	N/A

Traffic Events	Traffic events published by DGT. 1k registries (around 0,5 Mb of data).	JSON	N/A
Historical datasets (CANBus data + NMEA)	Historical datasets (CANBus data + NMEA) from real connected vehicles	CSV	Anonymised
Historical datasets (traffic events)	Historical datasets from traffic events	JSON	N/A
Motor Insurance Data	Data concerning motor insurance including data from the policies (duration, covers), data from vehicles (licence No, VIN etc.) and data from drivers (age, experience etc.)	CSV	Anonymised

4.10 Pilot #12 - Real World Data for Novel Health-Insurance products

The scope of Pilot #12 is to assist health insurance companies in improving the risk insurance profiles using the information collected by activity trackers and questionnaires, and by applying IoT & ML technologies on the collected information. In this context, the pilot aims to deliver two distinct services, one performing risk assessment and another one for fraudulent behaviour detection.

To this extend, Real-World Data (RWD) spanning from physiological, psychological to social and environmental aspects, as well as synthetic data (simulated lifestyle) are collected in the context of the pilot. These data are either measured using sensors on devices, or are reported by the participants via questionnaires. Both the measurements and the questionnaire posting and answering are managed by the Healthentia platform[11] of Innovation Sprint. Besides the RWD, synthetic data are provided, utilising a simulation tool that Innovation Sprint is developing. The data types are similar to the ones collected by actual Pilot #12 participants. The produced synthetic data are stored in Healthentia platform, so they will be accessed by the Pilot #12 INFINITECH infrastructure in the same as manner as with the actual data.

Currently the pilot has finished with the initial, proof of concept (PoC) and the data collection phases and is recruiting participants from the general population. The data to be collected in this third phase is a more mature version of that of the previous phases documented in D5.13. The new version is documented in the following table.

Table 34: Pilot #12 List of datasets

Dataset Name	Dataset description	Data format	Anonymization
Physiological measurements	The data physiological data collected are related to steps, distance travelled, time spent in different activity intensity categories, energy burned, floors climbed, exercise sessions. In the case where the participant has an activity tracker (Fitbit, Garmin, or anything feeding Apple Health with data), then resting heart rate, time spent in different heart rate zones, time to bed and wake up, time spent in different sleep zones is also measured. In the case where no activity tracker is available, widgets are employed to collect time of sleep and wake up. In	JSON	GRAD Anonymisation Tool

[11] Healthentia Platform, https://innovationsprint.eu/healthentia/

	both cases, liquid and food intake is also collected via widgets, while body weight and various symptoms (blood pressure, body temperature, cough, diarrhoea, fatigue, headache and pain) can be entered at arbitrary times by the participants.		
Health self-assessment	Daily health perception is collected via a questionnaire.	JSON	GRAD Anonymisation Tool
Quality of life self-assessment	A weekly questionnaire is employed to collect the perception on different aspects of life quality.	JSON	GRAD Anonymisation Tool

The data collected by Healthentia is ingested into the NOVA testbed for model training. Since the data contain personal and / or sensitive data, the Anonymisation Tool offered by GRAD is used in order to anonymise the data in a way the participants' privacy is preserved, hence removing any data privacy concerns. In particular, different anonymization algorithms (such as generalization, randomization, etc.) are applied to avoid the existence of data combinations that could lead to a possible re-identification of the data subjects. Additionally, a set of privacy and utility metrics that allow to measure the risk that remains after anonymizing the data and the impact of the anonymization process on the quality of the data, are calculated by the Anonymisation Tool and the results of the applied anonymisation algorithms will be assessed before the final decision is reached. The synthetic data will be bypassing the Anonymization Tool, as they do not need anonymisation.

4.11 Pilot #13 - Alternative/automated insurance risk selection - product recommendation for SME

The main objective of Pilot #13 is to implement a data analysis platform applying machine learning and artificial intelligence technologies to better predict the insurance needs of SMEs. In this context, the platform will generate a risk map of the SMEs based on their daily activities and will predict how the risk will vary on time. Therefore, the pilot will design and implement a service that effectively monitors the current risks of SMEs, as well as their risk variance in the future, in order to improve the control of the accident rate, the renewal of insurance policies and offer personalised insurance cover.

To this end, real data that are obtained from a variety of open data sources such as the web, social media profiles, official registers, opinion platforms, business directories, e-commerce platforms and more, are leveraged. The dataset is composed of data originating from 50000 EU SMEs and has both structured and unstructured data in image (PNG) format and text format, while the estimated data volume is expected to be around 1 TB of data. The following table summarize the main data sources that will be exploited in the context of Pilot #13:

Table 35: Pilot #13 List of datasets

Dataset Name	Dataset description	Data Format	Anonymisation
SMEWIF	SMEs website information and functionalities. Description of the text contained in the website of the companies, services and structure of the company: • Name, Brand Name • Business Activity • Phone Number, Brand Address, Website Phone Number	AWS S3 / Dynamo DB (text format)	Not required (Open data)

	• Description from company • Website • Security Protocol • Play Store, APP Store • Location from website • Nº Opinions, Average Rating • E-Mail • Business Hours • Categories • Claimed		
ROPS	Review and opinions platforms. Reputation information and opinions of clients about products and services.	AWS S3 / Dynamo DB (text format)	Not required (Open data)
EUBD	European SMEs Business Directories. Official and legal information about the companies, social object, activities, other companies where they have equity: • Legal Name • Registered Address • Address • Court • Last Announcements in Commercial Register • Incorporated • Company Type • Age of company • Number of employees • SHARE CAPITAL • SIC/NACE CODE • VAT NUMBER	AWS S3 / Dynamo DB (text format)	Not required (Open data)
GIO	SMEs geolocation information and characteristics images and geographical information.	AWS S3 / Dynamo DB (text format	Not required (Open data)
SMSIP	Social media SMEs information and presence. Evaluation and track of the channels where the SMEs have online presence.	AWS S3 / Dynamo DB (text format	Not required (Open data)

In addition to the real data, synthetic data are leveraged towards the aim of increasing the prediction accuracy and the usability of the designed machine learning models. The synthetic data creation aims to produce hybrid synthetic data by building models from real world data and domain knowledge and leveraging a simulator that is driven by these models to generate synthetic data. Within the context of the pilot, either open data or synthetic data are utilised. Hence, the need for any anonymisation process is not foreseen.

4.12 Pilot #14 - Big Data and IoT for the Agricultural Insurance Industry

The objective of Pilot #14 is to deliver a commercial service module which will enable insurance companies to exploit the untapped market potential of Agricultural Insurance (AgI), taking advantage of innovations in Earth Observation (EO), weather intelligence & ICT technology. Towards this end, EO is leveraged to implement a crucial new supplementary information source which will be utilised by insurance companies in their products design and risk assessment processes related to nature disasters. On the other hand, weather intelligence related to data assimilation, numerical weather prediction and ensemble seasonal forecasting will be leveraged with the aim of effectively verifying the occurrence of catastrophic weather events, as well as predicting predict future perils, providing crucial information to the agricultural insurance companies for possible threats of their portfolios.

In this context, Pilot #14 mainly exploits the data produced by the satellite and the weather intelligence engine which are complemented by anonymised in-situ data of the insured parcels that are currently used both as input and calibration data for the existing insurance services. In detail, the following data are utilised in Pilot #14:

Table 36: Pilot #14 List of datasets

Dataset Category	Dataset Name	Dataset description	File Format
Earth Observation data	Sentinel-2	Sentinel-2 products follow a specific naming code as described below: *S2A_MSIL2A_YYYYMMDDTHHMMSS_Nxxyy_ROOO_Txxxxx_<Product Discriminator>* where: • S2A or S2B = Spacecraft • MSI = Instrument • L2A = Processing level • YYYYMMDDTHHMMSS = Sensing start time and sensing stop time of the first line of granule in date UTC time format • Nxxyy = Processing Baseline number (e.g. N0204) • ROOO = Relative Orbit number (R001 - R143) • Txxxxx = Tile Number field	netcdf4, tif
	Sentinel-1	The top-level Sentinel-1 product folder naming convention is composed of upper-case alphanumeric characters separated by an underscore: *MMM_BB_TTTR_LFPP_YYYYMMDDTHHMMSS_YYYYMMDDTHHMMSS_OOOOOO_DDDDDD_CCCC.EEEE* where: • The Mission Identifier (MMM) denotes the satellite and will be either S1A for the Sentinel-1A instrument or S1B for the Sentinel-1B instrument. • The Mode/Beam (BB) identifies the S1-S6 beams for SM products and IW, EW and WV for products from the respective modes. • Product Type (TTT) can be RAW, SLC, GRD or OCN. • Resolution Class (R) can be F (Full resolution), H (High resolution) or M (Medium resolution) • The Processing Level (L) can be 0, 1 or 2. • The Product Class can be Standard (S) or Annotation (A). Annotation products are only used internally by the PDGS and are not distributed. • Polarisation mode (PP) can be of dual, single or partial-dual type. • The product start and stop date and times are shown as fourteen digits representing the date and time, separated by the character 'T'. • The absolute orbit number at product start time (OOOOOO) will be in the range 000001-999999. • The mission data-take identifier (DDDDDD) will be in the range 000001-FFFFFF. The product unique identifier (CCCC) is a hexadecimal string generated by computing CRC-16 on the manifest file. The CRC-16 algorithm used to compute the unique identifier is CRC-CCITT (0xFFFF).	netcdf4, tif
	Landsat-8	Following is a typical naming of a Landsat-8 product: *LXSS_LLLL_PPPRRR_YYYYMMDD_yyyymmdd_CX_TX_prod_band* where:	tif, tfw, xml, hdf,

		• L: Landsat • X:Sensor (“O” = OLI; “T” = TIRS; “C” = OLI/TIRS) • SS: Satellite (“08” = Landsat 8) • LLLL: Processing correction level (“L1TP” = Precision Terrain; “L1GT” = Systematic Terrain; “L1GS”= Systematic) • PPP: Path • RRR: Row • YYYY: Year of acquisition, MM Month of acquisition, DD Day of acquisition • yyyy: Year of processing, mm Month of processing, dd Day of processing • CX: Collection number (“01”, “02”, etc.) • TX: Collection category (“RT” = Real-Time; “T1” = Tier 1; “T2” = Tier 2) • prod: Product, such as “toa” or “sr” • band: such as “band<1-11>,” “qa,” or spectral index.	hdr, nc, or img
	PROBA-V Dry Matter Productivity (DMP)	The DMP collection 300 m v1 product follows the naming standard as follows: *c_gls_<product>[-<RTx>]_<YYYYMMDDhhmm>_<AREA>_<SENSOR>_V<Major.Minor.Run>* where: • <product> is the DMP300. • <RTx> is an optional parameter. It is used whenever a real-time product is provided: • RT0: Near Real Time product; • RT1 or RT2: Consolidated Real Time product (in convergence period), where the number equals the number of times the RT0 product was successively updated; • RT5: Final consolidated Real Time product. • <YYYYMMDDhhmm> gives the temporal location of the file. YYYY, MM, DD, hh, and mm denote the year, the month, the day, the hour, and the minutes, respectively. • <AREA> gives the spatial coverage of the file. For example, if the <AREA> is GLOBE, then the full globe product is available. • <SENSOR> gives the name of the sensor family used to retrieve the product, so VGT referencing SPOT-VEGETATION, and PROBAV for PROBA-V. • <Major.Minor.Run> gives the version number of the product. “Major” increases when the algorithm is updated. “Minor” increases when bugs are fixed or when processing lines are updated (metadata, colour quicklook, etc.). “Run” increases whenever a new processing run (with the same major and minor version) is performed without a change in the algorithm (e.g. due to a change in input data). This version refers to Major = 1.	tif, tfw, xml, hdf, hdr, nc, or img
Weather and Climate Data	Numerical weather predictions	A typical weather product follows the naming standard as follows: *<Weather_Parameter>_<AOI>_<Year>-<Month>-<Day>-<Hour>:00:00* where: • <Weather_Parameter> is the corresponding weather parameter (Temperature, RH etc.) • <AOI> is the Area of Interest • <Year> is the corresponding year • <Month> is the corresponding month • <Day> is the corresponding day of the month • <Hour> is the corresponding hour	tif, tfw, xml, hdf, hdr, nc, or img

Insured Parcels	Parcel Data	In order to be ingested by the system, all the required information related to the insured parcels are collected in a Shapefile format file reporting the following information for each parcel: • Parcel ID number • Boundaries (i.e. lat/lon coordinates of the corners of the field) • Soil texture (i.e. % sand, % clay, % silt) • Crop type • Size (in ha) • Country/county/area • Insured from (the type of the damage, the parcel is insured of) • Insured Value • Contract Duration • Total insured area (in case not the entire parcel is insured) • Total Insured value • Expected yield (per parcel or per ha)	Shapefile format file

4.13 Pilot #15 – Open Inter-Banking pilot

The objective of Pilot #15 is to develop a collaborative pilot gathering requirements from several Italian banks through its Centre of Competence on Artificial Intelligence (AI Hub). The pilot plans to deliver a Machine Learning model that is capable of reading the internal documents of a bank, highlighting the main concepts and allow these concepts to be traced back to reference taxonomies. Moreover, Pilot #15 aims to leverage Machine Learning and Natural Language Understanding paradigms to meet its goals.

An operational environment has been made available as part of pilot, which contains all the data made available by Banks.

The resources that have been collected include the following:

- Documentary collections
- Domain resources (such as dictionaries and taxonomies conceptual)

The data is stored in virtual machines running on servers managed and hosted on premises. The collected resources will not be manipulated, but analysed in order to develop a Metadata Creation service, to unify the conceptual description with respect to a common reference model (ABI Lab Taxonomy). The access to the operational environment is protected by firewalls and all the documents and related metadata is physically stored on the servers. The banks involved in the pilot have shared their documents through the protocol Secure File Transfer Protocol (SFTP), with dedicated accounts for each of the different banks. Regarding, the document loading procedure, each bank was provided with an account and an IP address associated with the SFTP server hosted in the data centre and made available as part of the project. Through a file transfer software, each bank was able to connect to the SFTP server and, through the account provided, was able to access a dedicated folder where to upload the documents, maintaining a defined naming convention. The use of accounts delivered individually to each bank guarantees that it is not possible to read or write access to the folders made available to other banks, ensuring privacy and access control. In addition to this, a data retention process was identified focusing on project needs and deadlines. In terms of volumes, more than 600 banking documents have been collected and used for the project scope.

The following table summarizes the main data sources that will be exploited in the context of Pilot #15:

Table 37: Pilot #15 List of datasets

Dataset Name	Dataset description	Data Format	Anonymisation
Banking Documents	Dataset contains more than 600 internal banking documents, from 5 banks, in different formats (pdf, docx, etc.).	Text	No anonymisation

5 Conclusions

The purpose of this deliverable entitled "D5.14 – Datasets for Algorithms Training & Evaluation - II" was to provide the final report of the outcomes of the work performed within the context of T5.1 "Data Collection for Algorithms Training & Evaluation" of WP5. To this end, with this deliverable, the fully functional version of the INFINITECH Data Collection component is delivered, providing as supplementary documentation its final design specification, the addressed use cases accompanied by the respective sequence diagrams, the complete and final implementation details and a presentation of implemented functionalities from the delivered component. In addition to this, the deliverable reported the results of the analysis on the characteristics and role of the synthetic datasets, as well as their role in INFINITECH. Finally, it presented the updated details of the datasets that are collected within the context of the pilots of INFINITECH.

At first, the deliverable provided the supplementary documentation of the INFINITECH Data Collection building on top of the initial documentation of the previous iteration by describing the details of its scope, the motivation for the development of the component and the challenges that it addresses. Furthermore, the deliverable presented the high-level architecture of the component, that is composed of three main modules, namely the Data Retrieval, the Data Mapper and the Data Cleaner. For each module, the complete design specifications were presented focusing on their main functionalities. The deliverable presented the use cases that each module addresses along with the relevant sequence diagrams that depict the interactions between the module and the stakeholders of INFINITECH. The deliverable presented also the detailed documentation of the implementation aspects of the component by presenting the complete list of functions and services per module which were implemented following the aforementioned design specifications along with the source code structure via UML diagrams. Finally, the delivered fully functional version of the component was presented by providing a walkthrough of the delivered functionalities from the user's perspective.

Following the INFINITECH Data Collection documentation, the results of the in-depth analysis of the synthetic datasets were presented. To this end, the analysis provided the advantages and limitations of the synthetic datasets were presented along with their categorisation into three main categories. In addition to this, through the analysis the overview of most common approaches that are followed during the synthetic data generation process were elaborated together the list of state-of-the-art tools and frameworks that are utilised within this process. The analysis providing useful insights on the motivation and use cases for which the synthetic datasets are leveraged within the context of INFINITECH project, as well as the list of synthetic datasets which are used by the INFINITECH pilots. It should be noted that the results remained unchanged from the previous iteration of the deliverable and they were reported for coherency reasons.

Finally, the deliverable presented the final list of datasets, real and synthetic, that are exploited by the INFINITECH pilots. For each pilot, an overview of the implemented data collection process was documented along with the final list of datasets that each pilot utilises to realise the aspired scenarios. In detail, for each dataset the description of the information included was elaborated along with the format of each dataset and the needs for anonymisation in order to cope with the data privacy requirements.

The deliverable constitutes the final report of the outcomes and the work performed within the context of T5.1. It documented all the activities performed from M18 till M27 by extending and updating the previously elaborated outcomes documented in deliverable D5.13 on M17. The updates introduced were the results of the analysis performed on the new requirements that have been identified during this second period as well as on the feedback that was collected by the pilots of the project and the stakeholders of the platform. The deliverable at hand concludes the activities of Task 5.1.

Table 38 - Conclusions (TASK Objectives with Deliverable achievements)

Objectives	**Comment**
Enable the exploitation of the variety of finance and insurance sector data sources.	The proposed solution successfully enables the collection of information from the variety of data sources that are exploited in the finance and insurance sector. It facilitates the effortless and efficient

	data collection, data mapping and data cleaning towards the ingestion of the information in the underlying storage in order to be further processed and exploited by the INFINITECH Machine Learning services.
Provide the mechanism that addresses the various connectivity and communication challenges of the complete data collection process	The INFINITECH Data Collection delivers the required abstract and holistic mechanism for the data providers of INFINITECH to effectively and efficiently collect, map into a data model and clean the required information from a large variety of data sources. Furthermore, through its microservice-based architecture it provides the means for further expansion of the supported data sources upon needs.
Design and deliver the required solution that enables the connection and retrieval of the information for different data sources	The detailed design specifications, as well as the final and fully functional release, of the INFINITECH Data Collection, are delivered with the current deliverable. The designed functionalities successfully the needs of the INFINITECH data provider to connect and retrieve information from a data source. The implementation of the INFINITECH Data Collection based on the designed specifications has been completed.

Table 39: Conclusions – (map TASK KPI with Deliverable achievements)

KPI	**Comment**
Data Collection mechanism available for enabling the collection and ingestion of data in an INFINITECH testbed	*Target Value = 1* The specific KPI is successfully achieved with the presented solution, namely the INFINITECH Data Collection, whose final version was released in the current deliverable, accompanied by the detailed design specifications which were also documented.
Coverage of different data source types	*Target Value = 6* Per the design specifications documented in the current deliverable, the INFINITECH Data Collection is capable of: • Retrieving new information from an API • Receiving new information from its exposed API • Fetching files from an FTP or HTTP server • Retrieve new information from a Relational Database • Retrieve new information from a HDFS deployment • Retrieve new information from a MinIO storage server
Number of services exposed to the clients of the INFINITECH Data Collection.	*Target Value >= 3* The INFINITECH Data Collection exposes three main services: a) the Data Retrieval service, b) the Data Mapper service, c) the Data Cleaner service

Appendix A: Literature

[1] M. J. T. D.-M. J. B. S. A. M. N. T. M. I. .. &. D.-L.-H.-V. E. Naeem, “Trends and Future Perspective Challenges in Big Data. In Advances in Intelligent Data Analysis and Applications,” in *Springer*, Singapore, 2021.

[2] H. Surendra and H. Mohan, “A review of synthetic data generation methods for privacy preserving data publishing,” *International Journal of Scientific & Technology Research,* vol. 6, no. 3, pp. 95-101, 2017.

[3] A. Z. R. a. B. S. Dandekar, “Comparative evaluation of synthetic data generation methods,” in *ACM Conference (Deep Learning Security Workshop)*, 2017.

[4] V. Ayala-Rivera, P. McDonagh, T. Cerqueus and L. Murphy, “Synthetic Data Generation using Benerator Tool,” University College Dublin, 2013.

[5] S. Assefa, D. Dervovic, M. Mahfouz, T. Balch, P. Reddy and M. Veloso, “Generating synthetic data in finance: opportunities, challenges and pitfalls,” JPMorgan Chase & Co, 2020.

[6] “FIWARE Vehicle Data model,” [Online]. Available: https://fiware-datamodels.readthedocs.io/en/latest/Transportation/Vehicle/Vehicle/doc/spec/index.html.

[7] “FIWARE Alert Data Model,” [Online]. Available: https://fiware-datamodels.readthedocs.io/en/latest/Alert/doc/spec/index.html.

[8] “FIWARE Weather Observed Data model,” [Online]. Available: https://fiware-datamodels.readthedocs.io/en/latest/Weather/WeatherObserved/doc/spec/index.html.

[9] “ETSI NGSI-LD specification,” [Online]. Available: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.01.01_60/gs_CIM009v010101p.pdf.

[10] “FIWARE,” [Online]. Available: https://www.fiware.org/developers/.

[11] “NGSI specifications,” [Online]. Available: http://fiware.github.io/specifications/ngsiv2/stable/.

Appendix B: INFINITECH List of synthetic datasets

Pilot #	Provider	Dataset Name	Short Description	Dataset Type	Dataset Format	Dataset Volume	Synthetic Data Category	Synthetic Data Methodology / Strategy	Library, Framework, Tool utilised
P4	PRIVE	Risk metrics figures	Portfolio data is used to calcultae portfolio metrics like portfolio performance, volatility, sharpe ratio and other risk figures	Numeric	CSV or customised	Depends on customer use cases	Hybrid synthetic	The historical price data is taken to calculate the performance. Historical price is used to calculate historical portfolio backtesting and portfolio risk-metrics such as volatility/standard deviation, sharpe ratio, return, max drawdown.	Prive Managers Platform
P5b	UPRC-BOC	Surrogate Data	Surrrogate data created based on Iterated Amplitude Adjusted Fourier-Transformed IAAFT in order to enhance the time-series historical transactions dataset used for the Cash-Flow prediction service.	Numeric/ time-series	CSV	Depends on the scenario execution	Partially synthetic	The data has been created based on Iterated Amplitude Adjusted Fourier-Transformed IAAFT and historical time-series transactions	IAAFT
P10	PI	Financial transactions data	Synthetic real time dataset related to financial transactions of the following: STFTF, PCTU, Bank Transfer SEPA , Foreign Bank Transfers, Internal Transfer of funds, SMWCA	Text / Numeric	CSV	3.5GB per Year	Fully Synthetic	Build models from real world data and domain knowledge. The data will be synthetized using a set of generators which will be based on a lightweight agent-based modelling framework while using inferred distribution.	Python Libraries
P11	ATOS	Connected Car dataset (FIWARE NGSI Vehicle data models)	Real time dataset related to connected vehicles (location, status, speed, acceleration, fuel consumption, etc.) driving within selected scenario	FIWARE NGSI Vehicle extended data model	JSON	~10MB per simulation	Fully Synthetic	Datasets extracted from Real time simulation. Historical datasets would be also generated	Atos SUMO 2 NGSI

P11	ATOS	Traffic Dataset (FIWARE NGSI Road & RoadSegment data models)	Real time datasets related to traffic (road occupation, traffic status, etc.) referred to the cities included in out SUMO-Based simulation tool (Cologne, Luxemburg, Monaco)	FIWARE NGSI Road and RoadSegment extended data models	JSON	~10MB per simulation	Partially Synthetic	Datasets extracted from Real time simulation. Historical datasets would be also generated	Atos SUMO 2 NGSI
P12	iSprint	Healthentia Simulated	Activity tracking data (steps, sleep etc.) and reported data regarding nutrition, health and quality of life self-assessments	Numerical Data	JSON	Depends on the scenario execution	Fully Synthetic	Build models from real world data and domain knowledge. Use a simulator driven by the models to generate synthetic data	Python Libraries
P13	WEA	SMEs synthetic raw data	SMEs raw data to complete the model and algorithm	Text	JSON	120 Kb per target/company	Hybrid Synthetic	Build models from real world data and domain knowledge. Use a simulator driven by the models to generate synthetic data	Python Libraries
P14	AGRO	Crop Biophysical Parameters	Crop Biophysical Parameters that will be used from crop health and crop productivity assessment. Typical examples of these parameters are Biomass, Chlorophyll content, Leaf Area Index and Yield.	Raster	GeoJSON / NETCDF	Depends on the size of the Pilot area	Partially Synthetic	Datasets are produced using ESA SNAP ANN	AgroApps OCTAPUSH/ Weather Intelligence Engine
P14	AGRO	Crop Loss	Crop loss assessment after an insured peril	Raster	GeoJSON / NETCDF	Depends on the size of the Pilot area	Partially Synthetic	Datasets are produced using AGROAPPS Ensemble EO change detection methodology and machine learning methodology for translating damages to actual crop losses	AgroApps OCTAPUSH/ Weather Intelligence Engine
P14	AGRO	Seasonal Crop Yield Prediction	Crop Yield Climatology and Seasonal Crop Yield Prediction	Raster	GeoJSON / NETCDF	Depends on the size of the Pilot area	Partially Synthetic	Datasets will be produced by using the GECROS crop simulation model	AgroApps OCTAPUSH/ Weather Intelligence Engine