Collective Wisdom Driving Public Health Policies

D3.21 Reliable Information Provision in Healthcare: Software Prototype v1

Project Deliverable

This project has received funding from the European Union's Horizon 2020 Programme (H2020-SC1-2016-CNECT) under Grant Agreement No. 727560

D3.21 Reliable Information Provision in Healthcare: Software Prototype v1

Work Package:	WP3	
Due Date:	31/12/2017	
Submission Date:	02/03/2018	
Start Date of Project:	01/03/2017	
Duration of Project:	36 Months	
Partner Responsible of Deliverable:	SILO	
Version:	1.1	
Status:	☒ Final ☐ Draft ☐ Ready for internal Review ☐ Task Leader Accepted ☐ WP leader accepted ☒ Project Coordinator accepted	
Author name(s):	Dimitris Miltiadou (SiLo), Konstantinos Perakis (SiLo), Thanos Kiourtis, Dimitris Poulopoulos (UPRC)	
Reviewer(s):	M. Patiño (UPM)	U. Wajid (ICE)
Nature:	☐ R – Report ☒ D – Demonstrator	
Dissemination level:	☒ PU – Public ☐ CO – Confidential ☐ RE – Restricted	

REVISION HISTORY			
Version	Date	Author(s)	Changes made
0.1	08/01/2018	Dimitris Miltiadou (SiLo)	Draft – Index
0.2	15/01/2018	Konstantinos Perakis & Dimitris Miltiadou (SiLo)	Contributions to section 2
0.3	19/01/2018	Konstantinos Perakis & Dimitris Miltiadou (SiLo)	Contributions to section 2
0.4	24/01/2018	Konstantinos Perakis & Dimitris Miltiadou (SiLo)	Contributions to section 3
0.4	29/01/2018	Thanos Kiourtis, Dimitris Poulopoulos (UPRC)	Contributions to section 2
0.5	02/02/2018	Thanos Kiourtis, Dimitris Poulopoulos (UPRC)	Contributions to section 2
0.6	09/02/2018	Thanos Kiourtis, Dimitris Poulopoulos (UPRC)	Contributions to section 3
0.7	12/02/2018	Dimitris Miltiadou (SiLo)	Minor modifications in sections 2 and 3
0.8	15/02/2018	Konstantinos Perakis (SiLo)	Review ready Version
0.8_ICE	23/02/2018	Usman Wajid (ICE)	ICE internal review
0.8_UPM	26/02/2018	Marta Patiño (UPM)	UPM internal review
0.9	27/02/2018	Dimitris Miltiadou (SiLo)	Addressed review comments
0.95	27/02/2018	Thanos Kiourtis, Dimitris Poulopoulos (UPRC)	Addressed review comments
1.0	28/02/2018	Dimitris Miltiadou (SiLo)	Final Version
1.1	02/03/2018	ATOS	Quality review , submission to EC.

List of acronyms

IDE	Integrated Development Environment
HHR	Holistic Health Records
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
POM	Project Object Model
RFC	Request For Comments
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

Contents

List of figures

List of tables

Executive Summary

The purpose of deliverable D3.21 - Reliable Information Provision in Healthcare: Software Prototype v1 is to document the preliminary software development efforts undertaken within the context of Task 3.5 – Data Cleaning Including Sources Reliability Assessment. D3.21 is a demonstrator deliverable and the scope of the document at hand is to document the implementation details of the Data Cleaner and Source Verifier component delivered within the context of Task 3.5, as an accompanying report. The main component is composed by two sub-components, namely the Data Cleaner, and the Source Verifier. For each sub-component, the service prototype overview is provided by describing the components of the prototype. In detail, this report documents the implemented processes and internal interfaces of each of the services and the external interfaces exposed by each sub-component. Additionally, the technologies and tools used in the implementation of each prototype are documented, along with the necessary information concerning the source code availability and exploitation.

It should be noted that the development of the Data Cleaner and Source Verifier component is a living process and this report defines the initial version of the software implementation. As the project evolves, the updates and refinements on the design and the specifications of the component that will be based on the initial evaluation of the services, and on any additional requirements that may arise as the project evolves, will be documented in the deliverable "Reliable Information Provision in Healthcare: Design and Open Specification v2" (D3.20). This will drive the implementation of the second version of the Data Cleaner and Source Verifier component which will be documented in the deliverable "Reliable Information Provision in Healthcare: Software Prototype v2" (corresponding to D3.22).

1. Introduction

Deliverable D3.21 undertakes the documentation of the preliminary efforts carried out within the context of Task 3.5 - Data Cleaning Including Sources Reliability Assessment. Task 3.5 aims at the development of the processes and mechanisms that will address the volatility of the information provision towards the aim of providing the necessary accuracy, consistency and usefulness of the incoming information and that will also ensure the reliability of the data sources within the context of CrowdHEALTH. The implementation of the Data Cleaner and Source Verifier component was driven by the architecture and the design presented in D3.19 of the project (Perakis K., Miltiadou D., Mavrogiorgou A., 2017) [1]. Since D3.21 is a demonstrator deliverable, the current document is the accompanying report documenting the software implementation information of the Data Cleaner and Source Verifier component delivered within the context of Task 3.5. The Data Cleaner and Source Verifier component is composed by two sub-components, namely the Data Cleaner sub-component that will address the volatility of the information provision and the Source Verifier sub-component that will address the reliability of the data sources. For each sub-component, the corresponding prototype is documented by providing an overview of the components of the prototype and by presenting the services of the prototype along with the list of processes and internal interfaces for each of the services implemented. In addition to this, for each prototype the external exposed interface is documented, along with the technologies and tools used for the implementation of the services.

The current deliverable is organized in the following sections:

1. The first chapter introduces the deliverable, documents its scope and presents the document structure. In addition to this, it documents the positioning of the deliverable within the project and the relation of the current deliverable to the other project tasks and deliverables.
2. In the second chapter, the Data Cleaner and Source Verifier prototype is presented in two subsections. The subsections are focusing on describing the prototype of each of the sub-components (i.e. Data Cleaner, and Sources Verifier) of the main component. For each prototype all the services are presented along with the list of functions and internal interfaces. In addition to the implemented services, the exposed external interfaces are documented as well as the technologies and tools used in the implementation.
3. In the third chapter, the necessary information concerning the source code availability and exploitation is presented.
4. The final chapter concludes the current deliverable and provides references on the future developments of the main component.

2. Prototype overview

2.1. Main components of the prototype

2.1.1. Data Cleaner

The scope of the Data Cleaner component is to undertake the processes of data cleaning and data completion, to the extent possible, of the datasets provided from various heterogeneous data information sources offering an important contribution in the course of accurate HHRs (Holistic Health Records). The Data Cleaner component provides the interface that implements the data cleaning workflow as documented in deliverable D3.19 of the project (Perakis K., Miltiadou D., Mavrogiorgou A., 2017) to be utilized by the rest of the CrowdHEALTH components ensuring data accuracy and consistency of the incoming datasets.

The architecture and design of the Data Cleaner component was documented in D3.19 with the purpose of addressing the volatility of the incoming data information towards the aim of providing data accuracy, consistency and completeness to the CrowdHEALTH platform. The Data Cleaner component implements the processes that identify inaccurate or corrupted datasets containing inaccurate, incorrect, incomplete or irrelevant data elements and consequently replace, modify or delete these data elements safeguarding the reliability and appropriateness of the incoming data information. The design of Data Cleaner component as documented in D3.19 is illustrated in Figure 2-1.

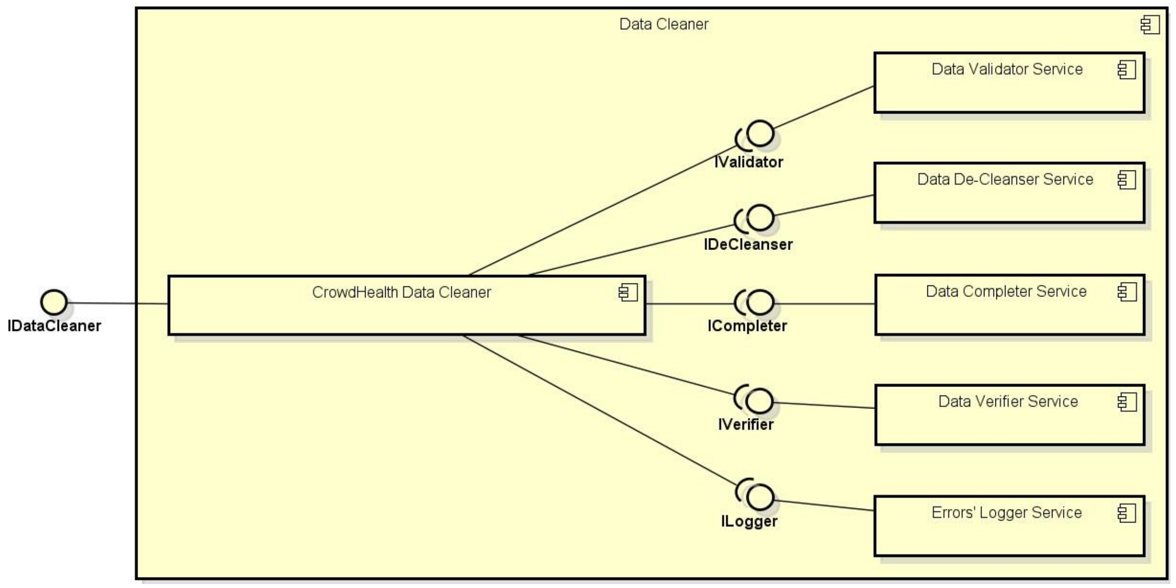

Figure 2-1: Data Cleaner component design

The software implementation of the Data Cleaner component was driven by this specification. Figure 2-2 illustrates the component UML diagram.

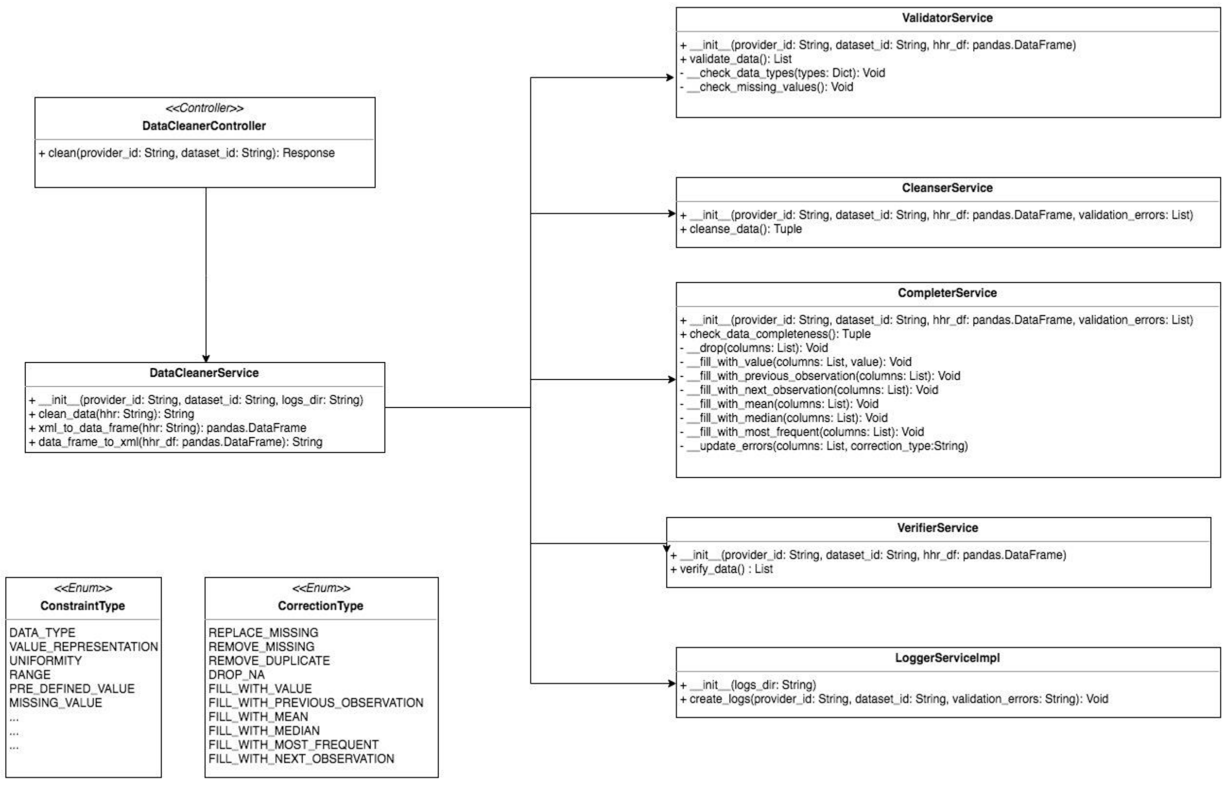
<<Controller>>
DataCleanerController
+ clean(provider_id: String, dataset_id: String): Response
DataCleanerService
+ __init__(provider_id: String, dataset_id: String, logs_dir: String)
+ clean_data(hhr: String): String
+ xml_to_data_frame(hhr: String): pandas.DataFrame
+ data_frame_to_xml(hhr_df: pandas.DataFrame): String
<<Enum>>
ConstraintType
DATA_TYPE
VALUE_REPRESENTATION
UNIFORMITY
RANGE
PRE_DEFINED_VALUE
MISSING_VALUE
...
...
...
<<Enum>>
CorrectionType
REPLACE_MISSING
REMOVE_MISSING
REMOVE_DUPLICATE
DROP_NA
FILL_WITH_VALUE
FILL_WITH_PREVIOUS_OBSERVATION
FILL_WITH_MEAN
FILL_WITH_MEDIAN
FILL_WITH_MOST_FREQUENT
FILL_WITH_NEXT_OBSERVATION
ValidatorService
+ __init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame)
+ validate_data(): List
- __check_data_types(types: Dict): Void
- __check_missing_values(): Void
CleanserService
+ __init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame, validation_errors: List)
+ cleanse_data(): Tuple
CompleterService
+ __init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame, validation_errors: List)
+ check_data_completeness(): Tuple
- __drop(columns: List): Void
- __fill_with_value(columns: List, value): Void
- __fill_with_previous_observation(columns: List): Void
- __fill_with_next_observation(columns: List): Void
- __fill_with_mean(columns: List): Void
- __fill_with_median(columns: List): Void
- __fill_with_most_frequent(columns: List): Void
- __update_errors(columns: List, correction_type:String)
VerifierService
+ __init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame)
+ verify_data() : List
LoggerServiceImpl
+ __init__(logs_dir: String)
+ create_logs(provider_id: String, dataset_id: String, validation_errors: String): Void

Figure 2-2: Data Cleaner model

As displayed in the UML diagram the Data Cleaner component is composed by one main service, namely the DataCleanerService, which in turn comprises of five internal services: the ValidatorService, the CleanserService, the CompleterService, the VerifierService and the LoggerService. The main service, DataCleanerService, handles all incoming and outgoing traffic of the DataCleaner component and is the only service exposed to the rest of the platform components. Additionally, the DataCleanerService implements the single exposed interface for data cleaning and data completion requests which is exposed by the DataCleanerController, the IDataCleanerService interface. Contrary to the main service, the five internal services are not exposed to the rest of the platform components and the DataCleanerService is interacting with these services via their internally exposed interfaces to realize the data cleaning workflow.

2.1.1.1. DataCleanerService

The DataCleanerService is the main service of the Data Cleaner component in charge of executing the data cleaning workflow of CrowdHEALTH platform. Since the data cleaning workflow comprises of several sequential steps, each one implemented by one of the internal services of the component, the DataCleanerService is responsible for the orchestration of these internal services as well as for monitoring the execution and providing the execution results to the requestor. In addition to the data cleaning workflow execution, the DataCleanerService is responsible for the implementation of the single interface for data cleaning and data completion requests, namely the IDataCleanerService.

The DataCleanerService implements the following three main functions:

- *__init__(provider_id: String, dataset_id: String, logs_dir: String)* : This is the main function initiating the DataCleanerService. It receives as input the provider identity, the dataset identity and the path to the logs directory.
- *xml_to_data_frame(hhr: String): pandas.DataFrame* : This is an internal function of the service facilitating the conversion from XML to data frame.
- *data_frame_to_xml(hhr_df: pandas.DataFrame): String* : This is an internal function of the service facilitating the conversion from data frame to XML.

In addition to the main functions, the DataCleanerService implements the single external interface of the Data Cleaner component, namely the IDataCleanerService. The interface handles the incoming requests for data cleaning and data completion and implements the following single function:

- *clean_data(hhr: String): String*: This function orchestrates the rest of the internal services towards the execution of the data cleaning workflow.

Moreover, the IDataCleanerService interface is exposed by the DataCleanerController which is responsible for handling the incoming HTTP requests of the external exposed interface and implements the following main function:

- *clean (provider_id: String, dataset_id: String): Response*: This function is responsible for handling incoming HTTP requests to execute the data cleaning workflow on a dataset and provide the results back to the requestor.

For more details on the exposed external interface please also refer to Section 2.2.

2.1.1.2. ValidatorService

The ValidatorService is the internal service responsible for the data validation processing of the incoming information data. The ValidatorService performs a variety of validation checks in order to evaluate the conformance to a set of constraints currently integrated in the business logic of the service. The current list of validation rules includes conformance to specific data types (integer, string, etc.) and identification of missing values for the data elements and will be further enriched as the project evolves. The ValidatorService implements the following functions:

- *__init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame):* This is the main function initiating the ValidatorService. It receives as input the provider identity, the dataset identity and the data frame representation of the HHR.
- *__check_data_types(types: Dict): Void:* This is an internal function validating the conformance to the data types.
- *__check_missing_values(): Void:* This is an internal function for the identification of missing values.

Additionally, the ValidatorService implements the internal interface namely IValidatorService with the following main function:

- *validate_data(): List*: This function is responsible for initiating the data validation execution. Once the data validity has been performed, the list of errors based on the evaluation of the set of constraints is returned.

2.1.1.3. CleanserService

The CleanserService is the internal service responsible for the de-cleansing processing of the incoming information data. The CleanserService eliminates the list of errors identified by the ValidationService by applying all the necessary corrective actions on the data elements marked with errors. De-cleansing is performed in an automated way based on a set of rules currently integrated in the business logic of the component. The CleanserService implements the following main function:

- *__init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame, validation_errors: List):* This is the main function initiating the CleanserService. It receives as input the provider identity, the dataset identity, the data frame representation of the HHR and the list of identified errors as provided by ValidatorService.

The CleanserService implements the internal interface namely ICleanserService with the following main function:

- *cleanse_data(): Tuple*: This function is undertaking the necessary actions to perform the de-cleansing processes. Upon receiving the incoming information data and the list of errors as input, it returns the updated incoming information data along with the list of actions performed during the de-cleansing process.

2.1.1.4. CompleterService

The CompleterService is the internal service undertaking the necessary actions to ensure the data completeness of the incoming information data. To validate the data completeness, the CompleterService is utilizing a predefined set of conformance rules indicating the mandatory fields and required non-empty attributes. These set of conformance rules is currently integrated in the business logic of the component. Following the data completeness evaluation, automated data filling is performed based on interpolation or extrapolation techniques. The CompleterService implements the following main functions:

- *__init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame, validation_errors: List):* This is the main function initiating the CompleterService. It receives as input the provider identity, the dataset identity, the data frame representation of the HHR and the list of identified errors as provided by ValidatorService.
- *__drop(columns: List): Void:* This is an internal function performing deletion of rows from the dataset based on the columns received as input. More specifically, for each column if there is an empty entry the respective row is deleted.
- *__fill_with_value(columns: List, value): Void:* This is an internal function for data value completion based on the value received as an input.
- *__fill_with_previous_observation(columns: List): Void:* This is an internal function for data value completion based on the previous observation (Last Observation Carried Forward).
- *__fill_with_next_observation(columns: List): Void:* This is an internal function for data value completion based on the next observation (Next Observation Carried Backward).
- *__fill_with_mean(columns: List): Void:* This is an internal function for data value completion based on the mean value of column.
- *__fill_with_median(columns: List): Void:* This is an internal function for data value completion based on the median value of column.
- *__fill_with_most_frequent(columns: List): Void:* This is an internal function for data value completion based on the most frequent value of column.
- *__update_errors(columns: List, correction_type:String):* This is an internal function responsible for updating the list of identified errors according to the actions undertaken.

The CompleterService implements the internal interface namely ICompleterService with the following main function:

- *check_data_completeness(): Tuple*: This function is responsible for the data completeness evaluation and data completion process of the incoming information data. Once the evaluation is complete, the data completion is performed and the updated incoming data is returned along with the updated list of identified errors and actions performed during the data completion process.

2.1.1.5. VerifierService

The VerifierService is the service responsible for the verification and evaluation of the corrective actions undertaken by CleanserService and CompleterService with the aim of ensuring the accuracy and consistency of the updated incoming information data according the CrowdHEALTH platform requirements. The VerifierServices implements the following main function:

- *__init__(provider_id: String, dataset_id: String, hhr_df: pandas.DataFrame):* This is the main function initiating the VerifierService. It receives as input the provider identity, the dataset identity and the data frame representation of the HHR.

The VerifierService implements the internal interface namely IVerifierService with the following main function:

- *verify_data() : List* : The function is responsible for initiating the verification and evaluation of the corrective actions and providing the results of the evaluation back to the requestor.

2.1.1.6. LoggerService

The LoggerService is undertaking the responsibility of keeping records containing all the errors identified and the corrective actions undertaken to address these errors during the data cleaning workflow execution by the rest of the internal services of the Data Cleaner component. For each execution of the data cleaning workflow a unique record will be created and stored in the list of the records kept by the LoggerService. In the current implementation the list of records is kept in a log file in the local file system where the LoggerService is running. LoggerService implements the following main function:

- *__init__(logs_dir: String)* : This is the main function initiating the LoggerService. It receives as input the path to the logs directory.

The LoggerService implements the internal interface namely ILoggerService with the following main function:

- *create_logs(provider_id: String, dataset_id: String, validation_errors: String): Void* : This function is responsible for creating a new record based on the information provided as input. This new record is appended at the end of the log file.

2.1.2. Sources Verifier

The scope of the Sources Verifier component is to dynamically categorize both known and unknown data sources to specific “levels of trustfulness” (i.e. data reliability, provided information type, data availability), according to a given threshold, resulting into the adaptive selection of all the available data sources in order to be connected into the CrowdHEALTH platform. The Sources Verifier component provides the interface that implements the sources verification workflow as documented in deliverable D3.19 of the project (Perakis K., Miltiadou D., Mavrogiorgou A., 2017) to be utilized by the rest of the CrowdHEALTH components providing all the necessary actions for choosing and finally keeping connected to the CrowdHEALTH platform only the trustful and reliable data sources.

The architecture and design of the Sources Verifier component was documented in D3.19 aiming to provide a predictive selection mechanism for achieving data reliability and availability during runtime, concerning both known and unknown data sources that have been identified to be connected into the CrowdHEALTH platform. The component design of Sources Verifier as documented in D3.19 is illustrated in Figure 2-3.

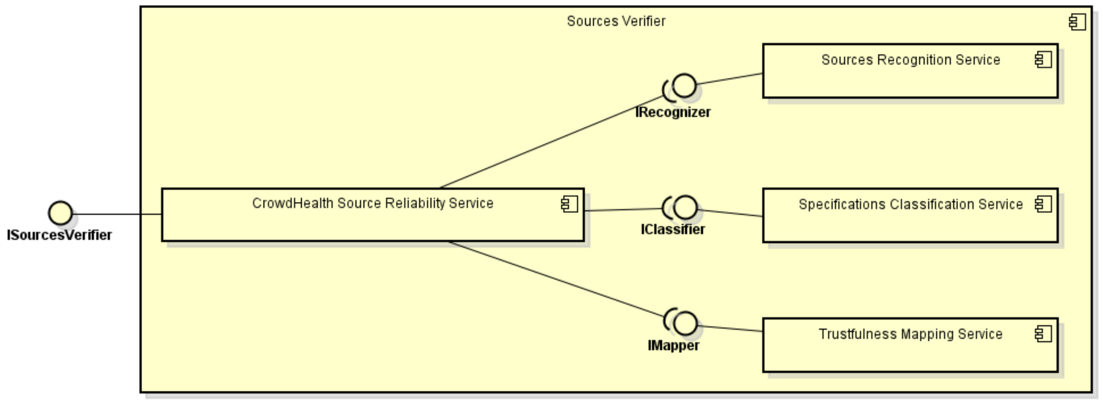
Sources Verifier
Sources Recognition Service
IRecognizer
ISourcesVerifier
CrowdHealth Source Reliability Service
IClassifier
Specifications Classification Service
IMapper
Trustfulness Mapping Service

Figure 2-3: Sources Verifier component design

In more detail, the software implementation of the Sources Verifier component has been captured based upon this component design, whose UML diagram is depicted in Figure 2-4.

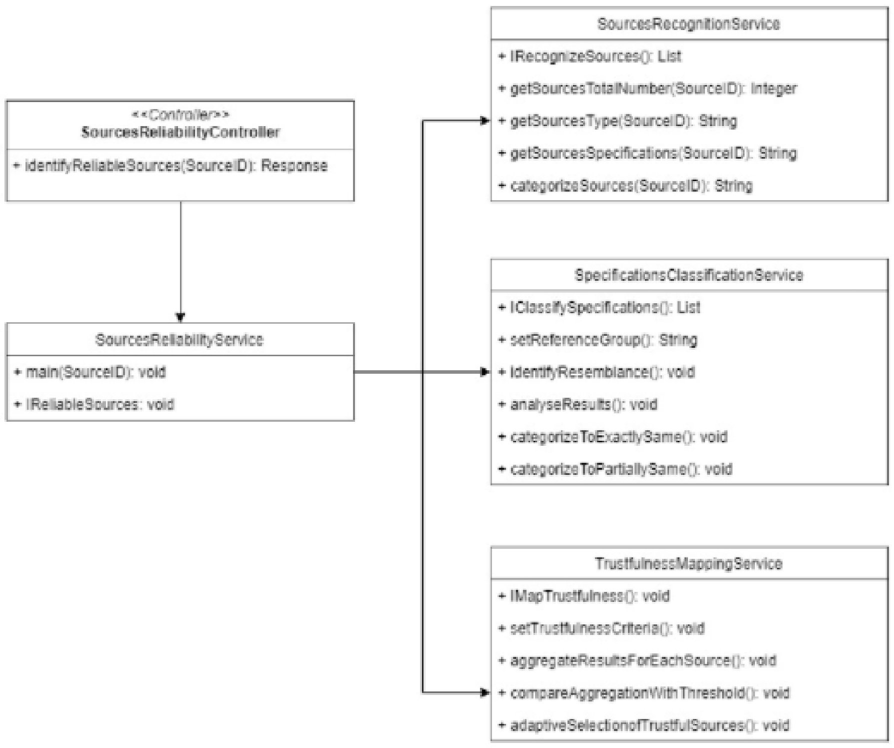
SourcesRecognitionService
+ IRecognizeSources(): List
+ getSourcesTotalNumber(SourceID): Integer
+ getSourcesType(SourceID): String
+ getSourcesSpecifications(SourceID): String
+ categorizeSources(SourceID): String
<<Controller>>
SourcesReliabilityController
+ identifyReliableSources(SourceID): Response
SpecificationsClassificationService
+ IClassifySpecifications(): List
+ setReferenceGroup(): String
+ identifyResemblance(): void
+ analyseResults(): void
+ categorizeToExactlySame(): void
+ categorizeToPartiallySame(): void
SourcesReliabilityService
+ main(SourceID): void
+ IReliableSources: void
TrustfulnessMappingService
+ IMapTrustfulness(): void
+ setTrustfulnessCriteria(): void
+ aggregateResultsForEachSource(): void
+ compareAggregationWithThreshold(): void
+ adaptiveSelectionofTrustfulSources(): void

Figure 2-4 Sources Verifier model

As displayed in the UML class diagram the Sources Verifier component is composed by one main service, namely the SourcesReliabilityService, which in turn comprises of three internal services, the SourcesRecognitionService, the SpecificationClassificationService, and the TrustfulnessMappingService. The main service, SourcesReliabilityService, handles all incoming and outgoing traffic of the Sources Verifier component and is the only service exposed to the rest of the platform components. Furthermore, the SourcesReliabilityService implements the single exposed interface for sources reliability requests that is exposed by the SourcesReliabilityController, the ISourcesReliabilityService interface. To this end, it should be mentioned that the three internal services of the SourcesReliabilityService are not exposed to the rest of the platform components, whilst the SourcesReliabilityService is interacting with each one of these internal services via their internally exposed interfaces, as described below.

2.1.2.1. SourcesReliabilityService

The SourcesReliabilityService is the main service of the Sources Verifier component, being responsible for providing all the necessary actions for choosing and finally keeping connected to the CrowdHEALTH platform only the reliable both known and unknown data sources. In more detail, the sources verification workflow consists of three sequential steps, each one being represented by the corresponding existing internal service of the SourcesReliabilityService. Therefore, the SourcesReliabilityService's responsibility is to orchestrate and manage all these internal services, while successfully completing via the ISourcesReliabilityService all the requests that are been made for using this service.

The SourcesReliabilityService implements the following one function:

- *main(SourceID): void*: This function is responsible for triggering the whole process of identifying the sources' reliability.

Apart from these functions, the ISourcesReliabilityService is implementing the single external interface of the Sources Verifier component, namely the ISourcesReliabilityService, containing one main function:

- *IReliableSources(): void:* This function is responsible for enabling the connection among the sources reliability services.

Moreover, the ISourcesReliabilityService interface is being exposed by the SourcesReliabilityController, containing one main function:

- *identifyReliableSources(SourceID): Response*: This function is responsible for handling incoming HTTP requests to execute the sources reliability workflow and provide the results back to the requestor.

More details of the exposed external interface can be found in Section 2.2.2.

2.1.2.2. SourcesRecognitionService

The SourcesRecognitionService is one of the internal services of SourcesReliabilityService, being responsible for the identification of the various available data sources that need to be connected to the CrowdHEALTH platform. The SourcesRecognitionService performs several sequential steps for identifying both known and unknown data sources' specifications (i.e. software and hardware specifications). More specifically, the SourcesRecognitionService implements the following four main functions:

- *getSourcesTotalNumber(SourceID): Integer:* This function is responsible for getting the total number of available sources that have requested to connect to the platform.
- *getSourcesType(SourceID): String:* This function is responsible for getting the type of the available sources that have requested to connect to the platform.
- *getSourcesSpecifications(SourceID): String:* This function is responsible for getting the specifications of the available sources that have requested to connect to the platform.
- *categorizeSources(SourceID): String:* This function is responsible for categorizing the sources into either known or unknown, based on the results of the previous functions.

Apart from these functions, the SourcesRecognitionService implements the single external interface of the Sources Verifier component, namely the ISourcesRecognitionService, containing one main function:

- *IRecognizeSources(): List*: This function is responsible for connecting the internal SourcesRecognitionService and the main SourcesReliabilityService.

2.1.2.3. SpecificationClassificationService

The SpecificationClassificationService is one of the internal services of SourcesReliabilityService, being responsible for the classification of the data sources' specifications based on the known data sources' specifications. Hence, the SpecificationClassificationService performs several sequential steps in order to classify the data sources based upon the similar specifications they have. To this end, two different scenarios arise, being represented from the corresponding functions of the service. More specifically, the SpecificationClassificationService implements the following five functions:

- *setReferenceGroup(): String:* This function is responsible for setting different groups of sources as reference values, in order for the classification/categorization to take place.
- *identifyResemblance(): void:* This function is responsible for identifying the resemblance of each source to the different reference values that were previously set.
- *analyzeResults(): void:* This function is responsible for analysing the results of the resemblance identification.
- *categorizeToExactlySame(): void:* This function is responsible for categorizing the sources that have exactly the same specifications with the sources that are used as reference values.
- *categorizeToPartiallySame(): void:* This function is responsible for categorizing the sources that have partially the same specifications with the sources that are used as reference values.

Apart from these functions, the SpecificationClassificationService implements the single external interface of the Sources Verifier component, namely the ISpecificationClassificationService, containing one main function:

- *IClassifySpecifications(): List* : This function is responsible for connecting the internal SpecificationsClassificationService, and the main SourcesReliabilityService.

2.1.2.4. TrustfulnessMappingService

The TrustfulnessMappingService is one of the internal services of SourcesReliabilityService, being responsible for the mapping of the data sources (that are from-now-on known) to different levels of trustfulness. Therefore, the TrustfulnessMappingService is performing several sequential steps in order decide whether each data source is considered as reliable or not, based upon a threshold level that is being set for each different “trustfulness” criterion (i.e.

data reliability, provided information type, data availability). More specifically, the TrustfulnessMappingService implements the following four main functions:

- *setTrustfulnessCriteria(): void*: This function is responsible for setting the trustfulness criteria, according to which a source will be considered as trustful or not.
- *aggregateResultsForEachSource(): void*: This function is responsible for aggregating the results that have derived for each different trustfulness criterion, into a single entity.
- *compareAggregationWithThreshold(): void*: This function is responsible for comparing the aggregated value of the previous results, with the threshold that has been set.
- *adaptiveSelectionOfTrustfulSources(): void*: This function is responsible for the adaptive selection of the sources that meet the trustfulness criteria, and are thus considered as trustful and reliable.

Apart from these functions, the TrustfulnessMappingService implements the single external interface of the Sources Verifier component, namely the ITrustfulnessMappingService, containing one main function:

- *IMapTrustfulness: void* : This function is responsible for connecting the internal TrustfulnessMappingService, and the main SourcesReliabilityService.

2.2. Interfaces

2.2.1. Data Cleaner

In the context of Data Cleaner component, as described also in Section 2.1.1.1, the external interface namely IDataCleanerService is implemented and is responsible for the execution of the data cleaning workflow. The incoming HTTP requests are handled by the DataCleanerController while the execution of the workflow is performed by DataCleanerService.

Through the IDataCleanerService interface the following two endpoints are offered:

1. Authentication/Login endpoint
2. Data Cleaning endpoint

2.2.1.1. Authentication/Login endpoint

To access the interface endpoints, authentication is mandatory. The Authentication/Login endpoint is implementing the authentication process by utilizing JSON Web Token (JWT)[1] towards a token-based authentication. JWT is an open standard (RFC 7519[2]) that defines a compact and self-contained way for securely transmitting information between parties as a

[1] JSON Web Tokens, https://jwt.io/
[2] https://tools.ietf.org/html/rfc7519

JSON object. Once the user is authenticated, the user receives the generated JWT which will be included in the subsequent requests to the interface endpoints in order to verify the identity of the user and also perform the appropriate access control check over the rest of the resources or endpoints of the interface. At this point it should be noted however that for the first version of the Data Cleaner component only a predefined user is configured in order to be used by the CrowdHEALTH components ensuring secure communication.

The Authentication/Login endpoint is documented in Table 1:

Authentication/Login Endpoint	
Description	Creates a JWT for the user which is included in the response headers. For all subsequent requests the JWT should be included in the Authorization header in order to successfully access the rest of the endpoints.
Endpoint URL	http://hostname[:port]/login
HTTP method	GET
Parameters	N/A

Request Body	Expects a request body in the following format:
	{
	“username”: “user”,
	“password”: “pass”
	}

Table 1- Authentication/Login Endpoint

2.2.1.2. Data Cleaning endpoint

This is the endpoint is responsible for handling the data cleaning workflow execution requests and for providing the updated data as a result of the execution. To access the Data Cleaning endpoint a valid JWT, as obtained by the Authentication/Login Endpoint, is mandatory. The endpoint expects the dataset for which the data cleaning workflow will be executed in HHR-compliant XML format.

The Data Cleaning endpoint is documented in Table 2:

Data Cleaning Endpoint			
Description	Initiates the data cleaning workflow and provides the results		
Endpoint URL	http://hostname[:port]/cleaner/clean/{providerId}/{datasetId}		
HTTP method	POST		
Parameters	Authorization: Valid JWT as received by Authentication/Login Endpoint. The combination of the providerId and datasetId parameters is predefined according to the project use cases identified:		
	▪ bio ◦ allergies	▪ dfki ◦ activity	▪ hulafe ◦ emergency

	◦ biosignals ◦ medication ◦ phr ▪ cra ◦ patient ◦ diagnosis ◦ treatment ◦ comorbidity ◦ behaviour ◦ coaching ◦ sideeffect	◦ allergen ◦ allergy ◦ annotation ◦ biodata ◦ datasource ◦ diet ◦ diettype ◦ dish ◦ ingredient ◦ patient ◦ recipe ◦ recipestep	◦ hah ◦ hospitalization ◦ labtest ◦ morbidity ◦ outpatient ◦ patient
Request Body	HHR compliant dataset in XML format		

Table 2 – Data Cleaning Endpoint

2.2.2. Sources Verifier

In the context of Data Cleaner component, apart from the internal interfaces offered by each different internal service as described also in Section 2.1.2.1, the external interface namely ISourcesReliabilityService is implemented being responsible for managing all the requests that are been made for using this service.

Through the ISourcesReliabilityService interface, one single endpoint is being offered:

1. Sources Reliability endpoint

2.2.2.1. Sources Reliability endpoint

The Sources Reliability endpoint is responsible for handling the sources reliability workflow execution requests and for providing the list of the reliable sources as a result of the execution.

The Sources Reliability endpoint is documented in Table 3:

Sources Reliability Endpoint	
Description	Initiates the sources reliability workflow and provides the results
Endpoint URL	http://hostname[:port]/verifier/identifyReliableSources/{SourceID}
HTTP method	POST
Parameters	Sources' provided data
Request Body	Sources that have requested to connect into a List format

Table 3: Sources Reliability Endpoint

2.3. Baseline technologies and tools

2.3.1. Data Cleaner

Data Cleaner component is developed with Python 3.5 using the Flask python micro web framework[3]. Flask is a powerful framework written in Python and based on the Werkzeug[4] toolkit and Jinja2[5] template engine, that is independent from particular libraries or tools and that supports a large list of extensions for application features. Besides the Flask framework, a list of libraries and tools has been used in the context of Data Cleaner to support several functionalities of the component. For the data structure handling Pandas[6] has been selected while NumPy[7] is used for numerical computations. For transformations from XML to Python dictionaries XmlToDict[8] was selected and in order to enable JWT in Flask framework Flask-JWT[9] was used.

2.3.2. Sources Verifier

Sources Verifier component is developed with JAVA 8 as programming language, NETBEANS[10] v8.0.2 as IDE and TOMCAT[11] 9 as application server. To this end, for the whole component an Apache Maven[12] project was constructed. Maven is a software project management and comprehension tool that is based on the concept of a project object model (POM).

[3] Flask, http://flask.pocoo.org/
[4] Werkzeug, http://werkzeug.pocoo.org/
[5] Jinja2, http://jinja.pocoo.org/
[6] Pandas, https://pandas.pydata.org/
[7] NumPy, http://www.numpy.org/
[8] XmlToDict, https://pypi.python.org/pypi/xmltodict
[9] Flask-JWT, https://pythonhosted.org/Flask-JWT/
[10] Netbeans, https://netbeans.org/
[11] Tomcat, http://tomcat.apache.org/
[12] Apache Maven, https://maven.apache.org/

3. Source code

3.1. Data Cleaner

Data Cleaner component is a Python project following the standard python module methodology. The source code resides in the cleaner directory. More specific the source code is divided into the services implemented in the course of Data Cleaner that reside under the services directory and the utility functions used by the services that reside under the util directory. The init.py file is located in the cleaner directory. It initiates the Data Cleaner component. However, to make it more simple and easy to use the Data Cleaner component is also offered as a Docker image. For more information on how to use the available Docker image please refer to Section 3.1.2. Additionally, at the top level directory the appropriate README.md file exists containing a short description of the project, the necessary information on how to build and run the project either using Docker or manually.

At this point, it should be noted that the current version of the source code available in the repository, as documented in Section **¡Error! No se encuentra el origen de la referencia.**, orresponds to the first, fully functional version of the Data Cleaner prototype driven by the architecture and the design as documented in D3.19. The list of provided functionalities will be further enriched with more techniques and processes as the project evolves and as new or revised requirements may arise from the data providers, so this source code is subject to changes.

3.1.1. Availability

Data Cleaner is provided in CrowdHEALTH's GitLab repository and can be found under the following URL:

https://crowdhealthtasks.ds.unipi.gr/CrowdHEALTH/DataCleaner

3.1.2. Exploitation

The Data Cleaner prototype is a Python project. As a result, in order to be able to run the prototype manually, Python should be properly preinstalled and preconfigured on the system and PYTHONPATH should include all the modules of Data Cleaner prototype. In order to facilitate the exploitation process, the prototype is provided as a Docker image with option to build the project and run the code. More specific, in order to build the project, run the following command while being on the top-level directory of the cloned repository:

```
docker build -t crowdhealthtasks.ds.unipi.gr:4567/crowdhealth/datacleaner .
```

To run the prototype run the following command:

```
docker run -d -p 5000:80 crowdhealthtasks.ds.unipi.gr:4567/crowdhealth/datacleaner
```

This command will start the Data Cleaner prototype as a web application and the exposed external interface will be available at *localhost:8080*.

The information described above is also available in the README.md file at the top level of the repository.

3.2. Sources Verifier

Sources Verifier component is a Maven project following the standard maven structure. More specifically, at the top level files there is the pom.xml file, as well as textual documents meant for the user to be able to read immediately on receiving the source (README.txt, LICENSE.txt). The pom.xml file contains information about the project and configuration details used by Maven to build the project, containing default values such as the build directory. Moreover, there are two subdirectories of this structure: src and target. The target directory is used to house all output of the build, while the src directory contains all of the source code for building the project. In that case, the main SourcesReliabilityService resides, along with the three (3) additional internal sub-services are placed.

3.2.1. Availability

Sources Verifier is provided in CrowdHEALTH's GitLab repository and can be found under the same URL as the Data Cleaner:

https://crowdhealthtasks.ds.unipi.gr/CrowdHEALTH/DataCleaner

3.2.2. Exploitation

The Sources Verifier prototype is a Maven project, and in order to avoid running the prototype manually, as well as preinstalling and preconfiguring Maven on the system, the prototype is provided as a Docker image with option to build the project and run the code. More specifically, as in the case of the Data Cleaner, in order to build the project, run the following command while being on the top-level directory of the cloned repository:

docker build -t crowdhealthtasks.ds.unipi.gr:4567/crowdhealth/sourcesverifier

To run the prototype run the following command:

docker run -d -p 4000:80 crowdhealthtasks.ds.unipi.gr:4567/crowdhealth/ sourcesverifier

This command will start the Sources Verifier prototype as a web application and the exposed external interface will be available at *localhost:8080*.

The information described above is also available in the README.md file at the top level of the repository.

4. Conclusions

The scope of D3.21 was to document the preliminary efforts carried out within the context of Task 3.5 - Data Cleaning Including Sources Reliability Assessment. The first version of the software implementation of the Data Cleaner and Source Verifier component was based on the architecture and design specifications documented in D3.19.

As already described in the previous sections, the Data Cleaner and Sources Verifier component is composed by two sub-components. For the first sub-component, namely the Data Cleaner, the first version includes the main service, namely the DataCleanerService, which is responsible for the orchestration of the internal services towards the execution of the data cleaning workflow, and the five internal services, namely the ValidatorService, the CleanserService, the CompleterService, the VerifierService and the LoggerService. These internal services are responsible for the data validation processing, the de-cleansing processing, the data completeness assurance, the verification and evaluation of the corrective actions and the recording of these actions as well as the errors identified. The DataCleanerService is implementing the exposed external interface of the Data Cleaner component, namely the IDataCleanerService.

For the second sub-component, namely the Source Verifier, the first version includes the main service, namely the SourceReliabilityService, which undertakes the necessary actions for identifying and keeping connected to the CrowdHEALTH platform only the reliable the data sources and the three internal services, namely the SourceRecognitionService, the SpecificationClassificationService and the TrustfulnessMappingService. These internal services are responsible for the identification of the various available data sources to be connected to the CrowdHEALTH platform, the classification of the data sources' specifications based on the known data sources' specifications and the mapping of the known data sources to different levels of trustfulness. Moreover, the SourceReliabilityService is implementing the exposed external interface of Source Verifier component, namely the ISourceReliabilityService.

In addition to the services implemented, the exposed external interfaces are documented along with the technologies and tools used for the implementation.

D3.21 is a demonstrator deliverable and the current document is the accompanying report documenting the software implementation information of the Data Cleaner and Source Verifier component containing the description of the services and the interfaces implemented, the source code availability and exploitation. It should be noted that the development of the Data Cleaner and Source Verifier component is a living process and the forthcoming second and final version, which will be documented in D3.22 - Reliable Information Provision in Healthcare: Software Prototype v2, will contain refinements and enhancements towards the aim of providing additional functionalities but also addressing additional end user requirements.

5. References

[1] Perakis K., Miltiadou D., Mavrogiorgou A. (2017). D3.19 Reliable Information Provision in Healthcare: Design & Open Specification v.1. EC H2020 CrowdHEALTH Project