CrowdHEALTH

Collective Wisdom Driving Public Health Policies

D6.4 Integration of Results 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

D6.4 Integration of Results v1

Work Package:		WP6
Due Date:		28/02/2018
Submission Date:		03/04/2018
Start Date of Project:		01/03/2017
Duration of Project:		36 Months
Partner Responsible of Deliverable:		ENG
Version:		1.1
Status:	☒ Final ☐ Draft ☐ Ready for internal Review ☒ Task Leader Accepted ☒ WP leader accepted ☒ Project Coordinator accepted	
Author name(s):	Antonio De Nigro, Francesco Torelli, Domenico Martino (ENG); Thanos Kiourtis, George Peppas, Argyro Mavrogiorgou, Ilias Maglogiannis (UPRC); Andreas Menychtas, Christos Panagopoulos (BIO); Maroje Sorić (ULJ); Santiago Aso (ATOS); Konstantinos Perakis (SILO); Salvador Tortajada (HULAFE); Sokratis Nifakos (KI); Jan Janssen, Serge Autexier (DFKI); Usman Wajid (ICE); Thanos Kosmidis (CRA); Ricardo Jimenez-Peris, Pavlos Kranas (LXS); Marta Patiño (UPM); Vegard Engen (IT-INN); Mitja Lustrek (JSI)	
Reviewer(s):	Pavlos Kranas (LXS)	Stefanos Malliaros (UPRC)
Nature:	☒ R – Report ☐ D – Demonstrator	
Dissemination level:	☒ PU – Public ☐ CO – Confidential ☐ RE – Restricted	

REVISION HISTORY			
Version	Date	Author(s)	Changes made
0.1	17/01/2018	ENG	ToC shared with the participants.
0.2	25/01/2018	ENG, SILO, IT-INN	Updated ToC shared with participants.
0.3	16/02/2018	ENG, UPRC, SILO, ICE, ATOS, LXS	First draft of executive summary, introduction, first integration cycle architecture, HHR Manager, Data sources and Gateways, Data Converter, Data Cleaner, Data Store, Aggregator. Added sections for risk stratification tool and user interface.
0.4	07/03/2018	ENG, UPRC, SILO, ICE, ATOS, LXS, KI, JSI, IT-INN, LXS, UPM, CRA, ULJ, BIO, DFKI, HULAFE	Added sections Multimodal Forecasting and Causal analysis module, Visualization, Risk Stratification tool and Data anonymization. Updated sections first integration cycle architecture, HHR Manager, Data sources and Gateways, Data Converter, Data Cleaner, Data Store and Aggregator. Added

			conclusions.
0.5	23/03/2018	LXS, UPRC	Peer review.
1.0	29/03/2018	ENG, JSI, SILO, UPRC	Peer review remarks fixed. Final release.
1.1	03/04/2018	ATOS	Quality Review. Submission to EC.

List of acronyms

ACID	Atomicity Consistency Isolation Durability
API	Application programming interface
CEP	Complex Event Processing
CSV	Comma Separated Variable
HHR	Holistic Health Record
HL7	Health Level Seven International
FHIR	Fast Healthcare Interoperability Resources Specification
JDBC	Java Data Base Connectivity
JWT	JSON Web Token
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PDT	Portable Data Terminal (stock keeping device)
POJO	Plain Old Java Object
REST	Representational State Transfer (alternative to SOAP)
SQL	Structured Query Language
XML	Extensible Markup Language

Contents

List of figures

1. Executive Summary

The present document describes the results of the first development and integration cycle of CrowdHEALTH, as well as the work performed to achieve such results. Achievements are compared to the integration plan reported in D6.1, describing the level of completeness of the developed functionalities and integration of each component, highlighting possible delays or differences with respect to the plan, and reporting the most important issues already solved or still to be solved during this activities, when occurred. When applicable, possible evolution of the components are introduced, also if not yet confirmed, in order to provide insights for the second version of the CrowdHEALTH system.

The set of produced components are firstly represented by a simplified architectural diagram, then the provided interfaces of each component are described in details, explaining which interactions shown in the architecture diagram are implemented by each interface or class.

2. Introduction

2.1. Objectives

The integration plan for the development and testing of the CrowdHEALTH system, described in deliverable D6.1 [1], defines an iterative approach composed of three cycles, each one lasting 12 months, during which the system is implemented and incrementally extended to satisfy the identified user requirements. The present document aims to describe the results of the first development and integration cycle performed during the first year of the project, and to report the work that has been performed to achieve such results with respect to the integration plan.

The document is structured as follows:

- Section 1 reports the executive summary of this document;
- Section 2 provides an introduction to the present document, describing its objectives (section 2.1), and an overview of the architecture of the CrowdHEALTH system implemented during the first year of the project;
- Section 3 describes the integration of the results. For each component that has been implemented during the reporting period, a sub-section has been introduced reporting the list of implemented functionalities, the list of integrated components, the detailed description of the implemented technical interfaces and operations, the performed work and the most relevant encountered issues during the development activities.
- Section 4 reports the conclusions resulting from the integration of the results.

2.2. First cycle architecture

Based on the identified project goals and the collected technical and use case requirements (as reported in D2.1 [2]), and on the first version of the overall architecture of CrowdHEALTH (as reported in D2.4 [3]), the following architecture has been implemented (Figure 1) during the *first cycle* development. Shortly, Figure 1 represents all the components that have been implemented – in the form of prototype, during the 1st year of the project, accompanied with the interactions among them.

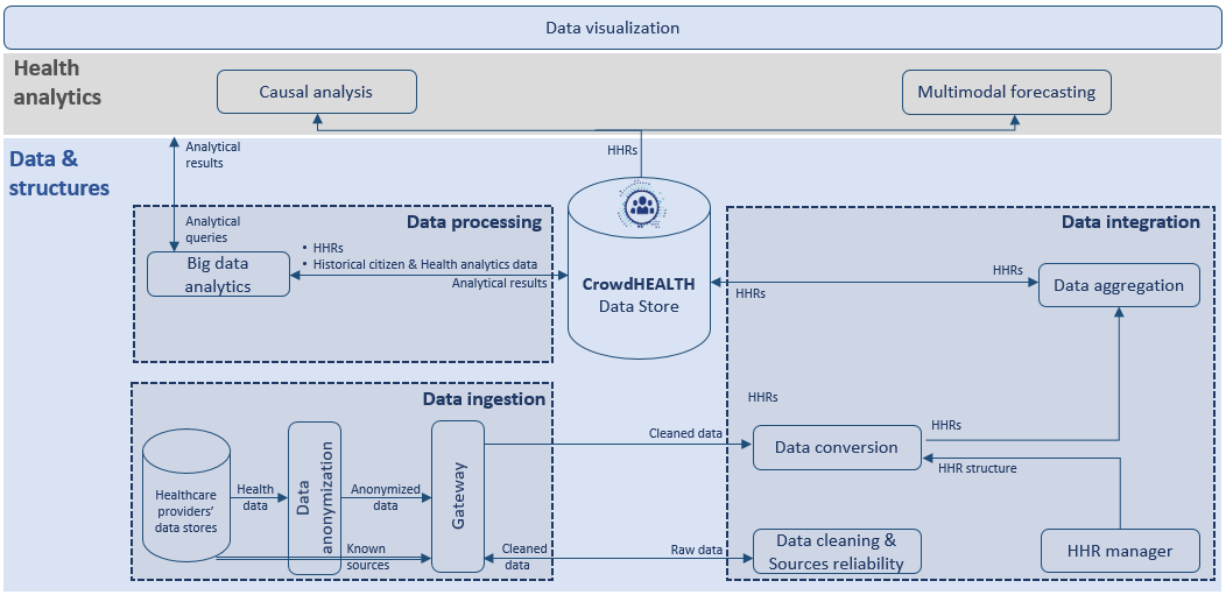

Figure 1 First cycle of CrowdHEALTH architecture

The first cycle of the architecture of the CrowdHEALTH platform consists of two (2) main pillars, the ***Data & structures***, and the ***Health analytics***.

In the context of ***Data & structures***, as it is outlined in Figure 1, the whole pillar is divided into three (3) different sub-pillars: (i) *Data ingestion*, (ii) *Data integration*, and (iii) *Data processing*, while the CrowdHEALTH Data Store belongs to the whole context of ***Data & Structures***, as it is considered to be vital for the three (3) sub-pillars.

To begin with, in the *Data ingestion* sub-pillar, the CrowdHEALTH platform takes as an input data from the healthcare providers' data stores. The *Data anonymization* component takes as an input this data to perform anonymization techniques, achieving the required data disclosure and privacy requirements. Sequentially, the anonymized data is being sent to the *Gateway* component, for solving at the same time connectivity and communication issues.

Sequentially, in the *Data integration* sub-pillar, the *Gateway* sends to the *Data cleaning & Sources reliability* component all the incoming anonymized data. In that case, the adaptive selection of the incoming sources occurs, whereas all the data derived from the chosen sources, is totally cleaned, by identifying and removing potential errors and inconsistencies. Therefore, the *Data cleaning & Sources reliability* component sends back to the *Gateway* all the cleaned data, which is then being sent to the *Data conversion* component. This component uses the *HHR manager* component, which is responsible for instantiating HHR objects that are returned to the *Data conversion* component. In sequel, the *Data conversion* component acquires as an input all the cleaned data, the new HHRs' objects, and convert them into HL7 FHIR format, making them structurally and terminologically interoperable.

Consequently, this data is sent to the *Data aggregation* component, which is responsible for aggregating them into the HHRs that will be finally stored into the CrowdHEALTH Data Store. Moreover, as for the *Data processing* sub-pillar, the *Big data analytics* is implemented, performing real-time big data analytics on the stored data (i.e. HHRs, historical citizen data), enabling correlations and extraction of situational factors among bio-signals, physical activities, medical data patterns, clinical assessment, and lab exams.

In the context of ***Health analytics***, analytical techniques are being utilized for carrying out *Causal analysis* for identifying the properties that affect the performance of policies and care plans, as well as *Multimodal forecasting* for estimating the applicability and effectiveness of health policies, and their variations and combinations to particular population segments, upon all the gathered data. Each one of these components works independently, acquiring as an input from the CrowdHEALTH Data Store all the constructed HHRs, that result from the relational tables that are stored in it. Additionally, it uses the *Big data analytics* component so as to perform its queries upon the stored data. The latter information will be provided to different entities in the ecosystem through the *Data visualization* component that is integrated with the Big Data Platform, and uses its exposed functionalities to perform both simply information retrieval and Big Data Analytics operations.

3. Integration of results

3.1. Data anonymization, user authentication and authorization

Implemented functionalities

The following list depicts the functionalities implemented by the Data anonymization component, which is responsible for performing anonymization techniques to achieve the privacy protection requirements. The exploitation and the usage of the Data anonymization component can be found in D4.20.

- Import of de-anonymized data via either CSV files or connection to external databases.
- Configuration of anonymization parameters by setting direct and indirect identifiers, configuring the parameters of the privacy model, and setting the generalization and suppression policies.
- Investigation of the available solutions that achieve k-anonymity.
- Measuring data quality of the anonymized data.
- Export of the anonymized data info CSV format.

The following list shows the working functionalities of the user authentication and authorization component during the first reporting period. The above-mentioned component is responsible for providing a single endpoint for user authentication for all the endpoints of the CrowdHEALTH system. The exploitation of the user authentication and authorization component can be found in D4.20.

- Registration of CrowdHEALTH applications and users.
- Management of CrowdHEALTH applications and users.
- Allow CrowdHEALTH applications to authenticate users via the internal OpenID Connect user base.

Integrated components

The Data anonymization component does not use any other CrowdHEALTH component.

The user authentication module uses the PDT and Visualization components, but the integration will be performed in the 2nd reporting period.

Technical interfaces

The data anonymization component does not offer any public interfaces, due to security considerations regarding data protection. The Data anonymization component requires local installation where the de-anonymized e-health data are located, to avoid transmission over the internet and potential security issues. Subsequently, if the healthcare providers do not have

their own anonymization tools and procedure, they can use the data anonymization virtual machine which is uploaded in the GITLAB server.

The user authentication and authorization module is a web based and can be triggered by the registered application and web users. The technical interfaces available from the above-mentioned module are described below.

Operation ID	3.1.2: User Registration.
URI pattern	https://crowdhealthidp.ds.unipi.gr/identity/register.
Input	Username: Contains the desired user's username First Name: It is the user's first name. Display Name: It is the desired user's display name. Last Name: It is the user's last name. Password: It is the user's desired Password. Email: It is the user's email.
Output	After the user registration, an administration is responsible for enabling the user account. With this process, we can eliminate malicious user registrations.

Operation ID	3.1.3: User authentication.
URI pattern	**Authorization Endpoint:** https://crowdhealthidp.ds.unipi.gr/oxauth/restv1/authorize **Token Entpoint:** https://crowdhealthidp.ds.unipi.gr/oxauth/restv1/token **User Information Entpoint:** https://crowdhealthidp.ds.unipi.gr/oxauth/restv1/userinfo
Methods	**Authorization Endpoint:** The GET method is used as the first step, where the application prepares an authentication request containing the required

	parameters. The Response from the Authorization Endpoint uses the GET method, and includes an authorization code. **Token Endpoint:** The authorization code is received by the application and is sent via the POST method to the Token Endpoint. The Token Endpoint replies with an access token and an ID token using the POST method. **User Information Endpoint:** The application uses the DI token to authorize the user. Next, the application can use the access token to access the User Information Endpoint for requesting user claims.
Input	**Authorization Endpoint:** scope=openid , response_type=code , client_id=<unique client id> , redirect_uri=<client's redirect uri> **Token Endpoint:** grant_type=authorization_code , code=<authorization_code> , redirect_uri=<client's redirect uri>, scope=openid , client_id=<unique client id>, client_secret=<unique client secret> **User Information Endpoint:** acess_token=<access_token value>

Performed work

During the first reporting period, all the functionalities that are described in D4.17 regarding the data anonymization, user authentication and authorization have been implemented. The Data anonymization component has been released to the use case partners to anonymize the data that will be used during the use cases, while the user authentication and authorization server has been setup in the UPRC premises. Although in D6.1 the user authentication and authorization module was not planned for the first year of the project, it has been considered essential for the first cycle demonstrations, so that the release of this tool has been anticipated.

Identified issues and possible evolutions

The main identified issue regarding the Data anonymization component is the lack of a network API to anonymize data. Although the software of the data anonymization component does not offer network connectivity, during the next reporting period the feasibility of integrating a network service over the data anonymization tool will be investigated. This process will be documented in the upcoming reporting period.

The main issues identified during the implementation of the user authentication and authorization module is the selection of the most suitable software for the requirements of CrowdHEALTH. Due to the nature of the project, it was essential to develop a solution that is compliant with the certification of the OpenID foundation, so as to ensure that the protocol stack is implemented correctly to avoid any potential security issues.

3.2. HHR Manager

Implemented functionalities

The following list reports the functionalities, at conceptual level, that have been implemented by the HHR Manager component. Details about their implementation and usage are reported in the deliverable D3.3.

- Instantiating Java POJO HHRs for physiological measurements.
- Instantiating Java POJO HHRs for fitness measurements.
- Instantiating Java POJO HHRs for symptoms.
- Instantiating Java POJO HHRs for diagnoses.
- Instantiating Java POJO HHRs for medications.
- Instantiating Java POJO HHRs for medical procedures.
- Instantiating Java POJO HHRs for nutrition.
- XML serialization and deserialization of HHR Java objects.

Integrated components

The HHR Manager do not use, and thus do not integrate, any other CrowdHEALTH component.

Technical interfaces

The HHR Manager is a library that implements the public classes HHRFactory and Serializer, which offer the following public operations to the Data Converter component.

HHRFactory

The following operations may be used by the Data Converter to instantiate Java POJO HHR objects.

Operation ID	3.2.1
Signature	*HHR create(String hhrTypeName)* Create an empty HHR object of the type specified in *hhrTypeName* parameter. A runtime exception is thrown if *hhrTypeName* is an invalid HHR type.
Input	*hhrTypeName*: The name of the type of HHR to be created (e.g. “Patient” or “ClinicalFinding.ANEMIA”). It must be a valid HHR type name, among the types defined in the HHR model.
Output	An empty object of the specified type *hhrTypeName*, implementing the HHR interface.

Operation ID	3.2.2
Signature	*<T extends HHR> T create(HHRType<T> type)* Create an empty HHR object of the specified *type*. A runtime exception is thrown if *type* is an invalid HHR type.
Input	*type:* The HHRType to be instantiated (e.g. ClinicalFinding.ANEMIA). Admitted values are the interfaces contained in the package eu.crowdhealth.hhr.model.
Output	An empty HHR object of type specified in the argument.

Operation ID	3.2.3
Signature	*<T extends HHR> T create(Class<T> hhrType)* Create an empty HHR object of the specified *hhrType*. A runtime

	exception is thrown if *hhrType* is an invalid HHR type descriptor.
Input	*hhrType:* The descriptor of the HHR model interface to be instantiated (e.g. Patient.class). Admitted values are the descriptors of the classes contained in the package eu.crowdhealth.hhr.model, which inherits from HHR.
Output	An empty HHR object of type specified in the argument.

Serializer

The following operations may be used by the Data Converter to serialize HHR Java object to XML format and vice versa.

Operation ID	3.2.4
Signature	*<T extends HHR> String toXML(T obj)* Serialize the HHR Java object *obj* into XML format.
Input	*obj:* The HHR object to be serialized in XML format.
Output	The XML representation of the *obj* HHR Java object

Operation ID	3.2.5
Signature	*HHR toObject(String hhrName, String xml)* Deserialize the XML representation of the HHR object of type *hhrName* to its Java representation.
Input	*hhrName:* The type name of HHR object to be deserialized. *xml:* The XML representation of the HHR object to be deserialized.
Output	The Java representation of the deserialized HHR object.

Operation ID	3.2.6
Signature	*<T extends HHR> String hhrCollectionToXML(Collection<T> objects)* Serialize in XML format and concatenate a set of HHR Java

	objects of the same type T.
Input	*objects:* A collection of HHR Java objects of the same type T.
Output	The concatenation of the XML representations of the Java objects provided as input.

Operation ID	3.2.7
Signature	*Collection<HHR> hhrCollectionToObject(String xml)* Deserialize a set of HHR objects represented in XML to its Java representation.
Input	*xml:* The concatenated XML representation of a set of HHR objects to be deserialized.
Output	The collection of Java objects deserialized from the *xml* input argument.

Performed work

During the reporting period, the HHR model specified in the deliverable D3.1 has been implemented. All planned functionalities have been released according to the integration plan. Additionally, a first version of the functionalities to serialize and de-serialize in XML the HHR Java objects have been released.

Identified issues and possible evolutions

The integration of the HHR Manager with the Data Converter highlighted the need to serialize and de-serialize HHR objects into XML format, although such functionalities was not planned for the first year of the project. This issue has been solved by implementing and including into the HHR Manager a first version of the Serializer class. The implemented functionalities will be improved during the second implementation cycle, if needed.

3.3. Data Sources and Gateways

Implemented functionalities

In the context of Data Sources and Gateways component and towards providing the necessary processes for solving connectivity and communication issues while also enabling multimodal data acquirement from various sources and various providers, the following functionalities have been implemented as reported in the deliverable D3.7:

- Connect to and retrieve information from database.
- Pull information via APIs.
- Receive and extract of information from files.
- Receive and extract of information via API.

Integrated components

The list of components with which the Data Sources and Gateways is being integrated, includes the following:

- Data Converter (to be completed).
- Data Cleaner (completed).

The integration of the Data Sources and Gateways has been accomplished via the exposed interfaces of both the Data Converter and the Data Cleaner components. By the time of writing of this deliverable, the integration with the Data Converter is an ongoing activity, while the integration with the Data Cleaner component has been successfully accomplished via the exposed interface IDataCleanerService of the Data Cleaner component.

Technical interfaces

The Data Sources and Gateways component provides one single interface, namely the IDataCollectorService, enabling data acquisition from a variety of external data sources for the CrowdHEALTH platform. This interface is responsible for retrieving information from the various data source providers, offering information via databases, exposed APIs or in specified file formats. Additionally, this interface is also responsible for receiving incoming information pushed to the CrowdHEALTH platform by a data source provider.

The IDataCollectorService interface implements the following three endpoints:

Authentication/Login endpoint

This endpoint is implementing the token-based authentication mechanism for the interface and is implemented using JSON Web Token (JWT) [4].

Operation ID	3.3.1
URI pattern	http://hostname[:port]/login
Methods	GET: Performs the authentication of the user. In the response headers the generated JWT is provided which is mandatory for accessing the rest of the endpoints of the interface. The authentication mechanism is provided out of the box by Spring security framework [5].

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

Output	None

Push data endpoint

This endpoint is implementing the “Receive and extract of information from files” and “Receive and extract of information via API” functionalities.

Operation ID	3.3.2
URI pattern	http://hostname[:port]/gateway/receive/{providerId}/{datasetId}
Methods	POST: Responsible for handling incoming HTTP requests in the case where information is being pushed to the CrowdHEALTH platform.
Input	*Authorization*: A valid JWT as received by Authentication/Login endpoint. *providerId*: The unique identifier of the provider. The acceptable values are: • BIO. • CRA. • DFKI. • HULAFE. *datasetId*: The unique identifier of the dataset for the specified provider.

<table>
<tr><td></td><td colspan="3">The combination of parameters providerId and datasetId is predefined (according to the identified use cases). For each providerId the list of acceptable values for the datasetId are displayed below:</td></tr>
<tr><td></td><td>▪ BIO
◦ Allergies.
◦ Biosignals.
◦ Medication.
◦ Phr.
▪ CRA
◦ Patient.
◦ Diagnosis.
◦ Treatment.
◦ Comorbidity.
◦ Behaviour.
◦ Coaching.
◦ Sideeffect.</td><td>▪ DFKI
◦ Activity.
◦ Allergen.
◦ Allergy.
◦ Annotation.
◦ Biodata.
◦ Datasource.
◦ Diet.
◦ Diettype.
◦ Dish.
◦ Ingredient.
◦ Patient.
◦ Recipe.
◦ Recipestep.</td><td>▪ HULAFE
◦ Emergency.
◦ Hah.
◦ Hospitalization.
◦ Labtest.
◦ Morbidity.
◦ Outpatient.
◦ Patient.</td></tr>
<tr><td>Output</td><td colspan="3">None</td></tr>
</table>

Pull data endpoint

This endpoint is implementing the “Connect to and retrieve information from database” and the “Pull information via APIs” functionalities.

Operation ID	*3.3.3*
URI pattern	http://hostname[:port]/gateway/{providerId}/{datasetId}
Methods	POST: Responsible for handling HTTP requests in the case of pulling information for a data source provider to the CrowdHEALTH platform
Input	*Authorization*: A valid JWT as received by Authentication/Login endpoint. *providerId*: the provider identifier. Supported values are: • ULJ • KI *datasetId*: the dataset identifier. This is parameter is optional.
Output	None

Performed work

All planned functionalities of the Data Sources and Gateways component that are included in the integration plan have been successfully delivered on time. The Data Sources and Gateways component supports all required functionalities and provides the necessary interface for the integration into the CrowdHEALTH platform. A set of the integration activities foreseen towards the full system integration, are still in progress status by the time of writing of this deliverable.

More particularly, the table below provides the status of the performed work:

Task	**Status**
Connect to and retrieve information from database	Completed successfully
Pull information via APIs	Completed successfully
Receive and extract of information from files	Completed successfully
Receive and extract of information via API	Completed successfully
Integration with Data Converter component	Ongoing activity
Integration with Data Cleaner component	Completed successfully

Identified issues and possible evolutions

During the development activities of the Data Source and Gateways component and by the time of writing of this deliverable, the most important issues identified are the following:

- The unavailability of local deployment that would facilitate the data acquisition for information retrieval (from the project data providers).
- The unavailability, at integration time, of the Data Converter component prototype in order to perform the integration activities towards the system integration as documented in the integration plan. However, this does not pose a significant issue since the integration with the exposed interface of the Data Converter (delivered as a stub) has already been implemented and tested.

3.4. Data converter

Implemented functionalities

- Dockerized maps for conversion from Raw format to HHR.
- Dynamic proxy/registry for Raw to HHR dockerized maps.
- Conversion from HHR to FHIR.

Integrated components

- HHR Manager.
- Aggregator.

Technical interfaces

Raw to HHR maps

Operation ID	3.4.1
URI pattern	http://icemain.hopto.org:7025/rawtohhr/cra/behaviour
Methods	POST: Converts received Raw document of CRA behaviour data into HHR format.
Input	*Body*: ID, user_ID, Variable, Value Raw Document
Output	HHR measure document out of the provided Raw document or body.

Operation ID	3.4.2
URI pattern	http://icemain.hopto.org:7025/rawtohhr/cra/comorbidities
Methods	POST: Converts received document with the comorbidities CRA schema to the HHR format.
Input	*Body*: ID, user_ID, Value Raw Document
Output	HHR comorbidities condition document.

Operation ID	3.4.3
URI pattern	http://icemain.hopto.org:7025/rawtohhr/cra/diagnosis
Methods	POST: Converts received document with the diagnosis CRA schema to the HHR format.
Input	*Body*: ID, user_ID, Value Raw Document
Output	HHR diagnosis condition document.

Operation ID	3.4.4
URI pattern	http://icemain.hopto.org:7025/rawtohhr/cra/patient
Methods	POST: Converts received document with the patient CRA schema to the HHR format.
Input	*Body*: ID, email, created_at Raw Document
Output	HHR patient document.

Operation ID	3.4.5
URI pattern	http://icemain.hopto.org:7025/rawtohhr/cra/sideeffect
Methods	POST: Converts received document with the sideeffect CRA schema to the HHR format.
Input	*Body*: ID, user_ID, Side_effect Raw Document
Output	HHR sideeffect condition document.

Operation ID	3.4.6
URI pattern	http://icemain.hopto.org:7025/rawtohhr/cra/treatment

Methods	POST: Converts received document with the CRA treatment schema to the HHR format.
Input	*Body*: ID, user_ID, Value Raw Document
Output	HHR treatment condition document.

Operation ID	3.4.7
URI pattern	http://icemain.hopto.org:7025/rawtohhr/hulafe/labtests
Methods	POST: Converts received document with the hulafe labtests schema to the HHR format.
Input	*Body*: PatientID, TestRequestDate, TestId, TestMagnitude, TestResult, TestUnits, TestPathology, LastPatientTest Raw Document
Output	HHR labtests document.

Operation ID	3.4.8
URI pattern	http://icemain.hopto.org:7025/rawtohhr/hulafe/morbidity
Methods	POST: Converts received document with the hulafe morbidity schema to the HHR format.
Input	*Body*: PatientID, ICD9, MainDiagnostic, DiagnosisDate, GroupAge, DiagnosisOrigin, HaHEpisode, Episode Raw Document
Output	HHR morbidity document.

Operation ID	3.4.9
URI pattern	http://icemain.hopto.org:7025/rawtohhr/hulafe/hospitalization

Methods	POST: Converts received document with the hulafe hospitalization schema to the HHR format.
Input	*Body*: PatientID, EpisodeCode, GroupAge, AdmissionServiceCode, RealServiceCode, AdmissionDate, AdmissionReasonCode, DischargeDate, DischargeReasonCode, DischargeDestinationCode, Exitus, LengthOfStay, UrgentAdmission, Surgery, SurgeryDate, PreSurgeryStay, ICD9Diagnostic, ICD9Procedure Raw Document
Output	HHR hospitalization encounter document.

Operation ID	3.4.10
URI pattern	http://icemain.hopto.org:7025/rawtohhr/hulafe/patient
Methods	POST: Converts received document with the hulafe patient schema to the HHR format.
Input	*Body*: PatientID,Gender,BirthYear,ExitusYear Raw Document
Output	HHR patient document.

Operation ID	3.4.11
URI pattern	http://icemain.hopto.org:7025/rawtohhr/hulafe/hah
Methods	POST: Converts received document with the hulafe Hospital at Home schema to the HHR format.
Input	*Body*: PatientID, EpisodeCode, InitDate, AdmissionDate, EndDate, RequestDate, AssessmentDate, Admission, LengthOfStay, SchemaCode, CircumstanceCode, FunctionCode, PatientTypeCode, OriginCode, StatusCode, SectionCode, LineCode, ServiceOriginCode, HaHDischarge Raw Document
Output	HHR HaH encounter document.

Operation ID	3.4.12
URI pattern	http://icemain.hopto.org:7025/rawtohhr/hulafe/emergency
Methods	POST: Converts received document with the hulafe emergency schema to the HHR format.
Input	*Body*: PatientID, EpisodeCode, Severity, AdmissionShiftCode, DischargeShiftCode, AdmissionServiceCode, TriageService, DestinationService, DischargeServiceCode, CircumstancesCode, ReasonsCode, RegistrationDate, FirstAttDate, AdmissionObservationDate, DischargeDate, HospitalizationDate, Registered, Classified, AdmittedHospital, Exitus, Excluded, Runaway, Attended, WaitingTimeTriage, WaitingTimeAtt, TotalLengthOfStay, ICD9 Raw Document
Output	HHR emergency encounter document.

Operation ID	3.4.13
URI pattern	http://icemain.hopto.org:7025/rawtohhr/hulafe/outpatient
Methods	POST: Converts received document with the hulafe outpatient schema to the HHR format.
Input	*Body*: PatientID, EpisodeCode, LocationCode, ServiceCode, ConsultationDate, VisitDone, BeginTime, EndTime, AgeGroup, TypeOfProvision Raw Document
Output	HHR outpatient encounter document.

Operation ID	3.4.14
URI pattern	http://icemain.hopto.org:7025/servicediscovery
Methods	GET: List of available maps.
Input	None
Output	XML describing all available maps with entry point, input format, and example output.

Convert HHR to FHIR format

Operation ID	3.4.15
URI pattern	http://{domain}:{port}/api/convert ex: http://icemain.hopto.org:7030/api/convert
Methods	POST: Converts received HHR document into FHIR format
Input	*Body*: HHR document in XML format
Output	201 CREATED + FHIR document, if the provided HHR document can be converted. 200 OK + Same document, if the provided document cannot be converted.

Performed work

Prototype implementation has been developed regarding the conversion of HHR format to FHIR. Additional efforts have been done to provide a dynamic mechanism capable of running new Raw to HHR maps.

Identified issues and possible evolutions

The integration of the Data Converter component is realised by providing REST-based interfaces to and from the component. Since many components of the CrowdHEALTH platform were being developed in parallel and a complete end-to-end integration and testing was not feasible, the REST-based approach has provided more flexibility during the development phase of the Data Converter component. This has also allowed the data converter component to be independently tested with use-case datasets. Future integration of Data Converter will also benefit from this modular approach.

3.5. Data cleaner

Implemented functionalities

The Data Cleaner component is addressing the volatility of the incoming information in the course of providing the desired data accuracy, consistency and completeness across the available information on the CrowdHEALTH platform. To achieve this, the following functionalities have been implemented as reported in the deliverable D3.21:

- Perform validation on information data.
- Perform cleaning on information data.
- Perform data completion on information data.
- Perform data verification on information data.
- Perform logging of the identified errors and corrective actions.

Integrated components

The Data Cleaner do not use, and thus do not integrate, any other CrowdHEALTH components.

Technical interfaces

The Data Cleaner component is providing the interface responsible for the data cleaning and data completion processing, namely the IDataCleanerService. This interface handles all incoming requests from the Data Sources and Gateways component in order to perform the data cleaning processing which includes ensuring the validity of the data, performing the data cleansing and data completion processing and finally the data verification.

The IDataCleanerService implements the following two endpoints:

Authentication/Login endpoint

This is endpoint is implementing the token-based authentication mechanism for the interface and is implemented using JSON Web Token (JWT) [4].

Operation ID	3.5.1
URI pattern	http://hostname[:port]/login
Methods	GET: Performs the authentication of the user. In the response headers the generated JWT is provided which is mandatory for accessing the rest of the endpoints of the interface. The

	authentication mechanism is provided out of the box by Spring security framework [5].
Input	*Expects a request body in the following format:* *{* *"username": "user",* *"password": "pass"* *}*
Output	*None.*

Data Cleaning endpoint

This endpoint is implementing all functionalities as listed in section Implemented functionalities.

Operation ID	*3.5.2*
URI pattern	http://hostname[:port]/cleaner/clean/{providerId}/{datasetId}
Methods	POST: Responsible for handling incoming HTTP requests to perform the data cleaning processing on a dataset and provided the results back to the requestor.
Input	*Authorization*: A valid JWT as received by Authentication/Login endpoint. *RequestBody*: The HHR compliant dataset for which the data cleaning will be executed in XML format *providerId*: The unique identifier of the provider. The acceptable values are: • BIO. • CRA. • DFKI. • HULAFE. *datasetId*: The unique identifier of the dataset for the specified provider. The combination of parameters providerId and datasetId is predefined (according to the identified use cases). For each providerId the list of acceptable values for the datasetId are displayed below:

▪ **BIO** ◦ Allergies. ◦ Biosignals. ◦ Medication.	▪ **DFKI** ◦ Activity. ◦ Allergen. ◦ Allergy.	▪ **HULAFE** ◦ Emergency. ◦ Hah. ◦ Hospitalization.

	◦ Phr. ▪ **CRA** ◦ Patient. ◦ Diagnosis. ◦ Treatment. ◦ Comorbidity. ◦ Behaviour. ◦ Coaching. ◦ Sideeffect.	◦ Annotation. ◦ Biodata. ◦ Datasource. ◦ Diet. ◦ Diettype. ◦ Dish. ◦ Ingredient. ◦ Patient. ◦ Recipe. ◦ Recipestep.	◦ Labtest. ◦ Morbidity. ◦ Outpatient. ◦ Patient.
Output	HHR compliant dataset in XML format.		

Performed work

The development activities for all planned functionalities of the Data Cleaner component that are included in the integration plan are still in progress, as most of these activities depend upon the availability of the actual data from the data providers, and the availability of historical information, both in one common schema, more specifically the HHR format. The delivered prototype of Data Cleaner includes several aspects of the functionalities, however there are several ongoing activities by the time of writing of this deliverable towards the completion of these functionalities. For the integration activities towards the system integration, the Data Cleaner component has been successfully integrated with Data Sources and Gateways component by providing the necessary interface.

More specifically, the table below provides the status of the performed work:

Task	Status
Perform validation on information data	Ongoing activity. The prototype currently supports the following methods: • Conformance to specific data types (integer, string, etc.). • Identify missing values. Constraints and rules definition is an ongoing activity in collaboration with the demonstrator partners.
Perform cleaning on information data	Ongoing activity. The prototype currently supports the preparation of the cleansing methods. Constraints and rules definition is an ongoing activity in collaboration with the demonstrator partners.

Perform data completion on information data	Ongoing activity. The prototype currently supports the following methods: • Drop value. • Fill with specific value. • Fill with previous observation. • Fill with next observation. • Fill with mean value. • Fill with median value • Fill with most frequent value
Perform data verification on information data	Completed successfully. Note: data verification is performing the necessary checks that the modifications/transformations introduced on the dataset by all previous steps (data validation, data cleaning and data completion) were correctly applied towards the expected data quality on the datasets. It is performed based on the existing methods defined in the previous rows. As the components evolves and more methods are added the data verification will be updated accordingly.
Perform logging of the identified errors and corrective actions	Completed successfully.
Integration with Data Sources and Gateways component	Completed successfully.

Identified issues and possible evolutions

During the development activities of the Data Cleaner components and by the time of writing of this deliverable, the most important issues identified are the following:

- There is an ongoing effort to define the constraints and rules in collaboration with demonstrator partners. These rules and constraints will specify the requirements for the development activities of the Data Cleaner component towards the completion of the functionalities of the component.
- The Data cleaner component requires historical data for the data cleaning processing. However, at the current time that this deliverable is being written, the focus is in the end-to-end integration of the various components of the platform. Due to this, the historical data required for the processing could not be available at that time.

3.6. Data store and big data analytics

Implemented functionalities

The Big Data Platform provides all functionalities that can be found in a traditional relational database management system through a standard JDBC interface. Moreover, it provides Big Data Analytics as an integral part of the platform, which internally uses both inter- and intra-query parallelism to be able to provide performance-wised analytics for Big Data in real-time, exploiting the ability of the platform to scale linearly to 100s of nodes without compromising neither the coherence of the data and the ACID properties, neither its overall performance under intensive operational workloads. The platform provides these analytics over the live data, letting the application developer to exploit the Big Data Analytics using standard SQL statements and let the platform itself to take care of returning the results. The Big Data Analytics exploit the parallel OLAP engine capabilities which can be enabled by setting the *parallel* parameter to a value greater than 0, indicating the level of parallelism needed. Additional parameters can be set to the distributed parallel query processing engine to configure its behaviour:

- *leanxcale.pq.impl*: Parallel query processing implementation to use. The default is org.apache.derby.impl.services.pq.SocketsParallelQueryService
- *leanxcale.pq.sockets.device*: Network device used for inter-workers communication. The default is loopback
- *leanxcale.pq.sockets.address*: Network address to bind for inter-workers communication.
- *leanxcale.pq.sockets.ports*: Port range used for the local worker. It should be specified as from-to, e.g., 20000-21000 to use ports 20000 to 20999
- *leanxcale.pq.sockets.buffer*: Size of each buffer used for inter-worker communication. Amount of memory used will this setting x (2 x nRemoteWorker) x nLocalWorkers at each host.The default is "64K". Suffixes M and K can be used for MByte and Kbyte
- *leanxcale.pq.scheduler*: Scheduling policy: affinity, prefer regions from a co-located region server; rr round-robin distribution of regions. The default is rr.
- *leanxcale.pq.shards*: Desired number of shards for each worker. A larger number improves load-balancing, but increases scan overhead. The default is 10

Apart from the standard functionalities that are expected from a RDBMS, the big data platform additionally provides functionalities for direct access to its internal key-value storage and to process and correlate streaming events with data at rest in real time by means of the Complex Event Processing (CEP) engine.

The functionalities to access the key-value storage can be summarized as follows:

- Initialize a connection with the key-value data store.
- Begin a transaction.
- Commit/Abort a transaction.

- Get an array of bytes representing a tuple given the value of its primary key or one of its secondary indexes.
- Scan a data table using a predicate and return a list of arrays of bytes, representing tuples that satisfy the input predicate.
- Insert an array of bytes which represents a tuple in a given data table.
- Remove tuples from a data table, based on a predicate input.
- Update the values of tuples that satisfy a predicate input.

The CEP can be accessed using a streaming API that provides functionalities to:

- Register and deploy continuous queries.
- Register to input streams of a continuous query.
- Subscribe to output streams of a continuous query.
- Create and send events to continuous queries.

Continuous queries are acyclic graph of streaming operators. Several streaming operators have already been developed:

- Stateless operators: Map, Filter, Union, Multiplexer.
- Stateful operators: Aggregate, Join, SelfJoin.
- Datastore operators.

Finally, the data store provides some business level functionalities that encapsulate the whole process of deserializing, transforming to the internal model and storing the FHIR/HHR objects to the data store. These can be summarized as follows:

- Insert FHIR/HHR objects related with *patients.*
- Insert FHIR/HHR objects related with *comorbidities.*
- Insert FHIR/HHR objects related with *side effects.*

Integrated components

The CrowdHEALTH data store integrates the Complex Event Processing engine developed by UPM and does not use any other external libraries, components or functionalities developed by other partners. It makes use of the HAPI FHIR open source specification implemented in Java [6] to internally deserialize the received input to the common data model that the components of the platform are aware of, but this is an external dependency rather than a direct one with a partner of the project.

Technical interfaces

For the LeanXcale Direct API for the internal key-value data store, the interfaces are the following:

Operation ID	3.6.1
Signature	public static synchronized void Conn.dial(String[] metaaddrs, LTM ltm) throws kv.Error Connects to the store given the addresses of the metadata server (host!port). This operation must be performed once, before any other call. And it must be performed just once. If LTM is not null, it is used as the LTM for the client library
Input	@Param*:* String[] metaaddrs: a list of addresses of the metadata servers. @Param: LTM ltm: an implementation of the LTM interface for managing transactions, or null if the internal implementation should be used.

Operation ID	3.6.2
Signature	public static byte [] Conn.get(long tid, String tname, byte [] key, short [] filds) throws kv.Error Get a tuple for a given key on the the given table, in the scope of the given transaction and returns its representation in an array of bytes, retrieving only the specified given fields.
Input	@Param*:* long tid: the id of the current transaction, whose visibility will decide which version of the tuple will be returned. @Param: String tname: the name of the table where the user wants to retrieve the tuple from. @Param: [] byte key: The primary key to get the tuple, represented by an array of bytes. @Param: [] short flids: a list of indexes, each one representing the index of the column of the tuple, that the user wants to retrieve data. If null, all fields will be returned.
Output	Byte []: the retrieved tuple in an array of bytes.

Operation ID	3.6.3
Signature	public static int Conn.scan(long tid, String tname, byte [] minkey, byte [] maxkey, byte [] prg, short [] filds, long nvals, int flags) throws kv.Error Scans the given table, in the scope of the given transaction and returns the (unsafe) integer pointing to the memory address where the result set is located. The result set contains all tuples between the rage of [minkey, maxkey) or the tuples that satisfy the given predicate. It returns only the given fields of the tuples, and the nvals records.
Input	@Param*:* long tid: the id of the current transaction, whose visibility will decide which versions of the tuples will be returned. @Param: String tname: the name of the table where the user wants to retrieve the tuples from. @Param: [] byte minKey: The minimum primary key to start scanning for tuples, represented by an array of bytes. If null it will start scanning from the beginning of the table. @Param: [] byte maxKey: The maximum primary key to stop scanning for tuples, represented by an array of bytes. If null, It won't stop scanning until it reaches the end of the table. @Param: [] byte prg: The predicate to evaluate tuples while scanning the given table. If nuil, tuples will be returned only according to the given primary key range. @Param: [] short: a list of indexes, each one representing the index of the column of the tuple, that the user wants to retrieve data. If null, all fields will be returned. @Param*:* long nvals: maximum number of tuples to be returned. If null, all tuples that satisfy the query criteria will be returned. @Param: int flags: various flags that alter the default behaviour of the data store while scanning. If null, the default behaviour will be expected.
Output	Int: the (unsafe) memory address where the result set is located. This address must be used only within the provided methods of this class. Direct memory access using Java Unsafe operators using this address are unpredictable.

Operation ID	3.6.4
Signature	public static void Conn.ups(long tid, String tname, byte [] value) inserts a tuple in the given table
Input	@Param: long tid: the id of the current transaction, which will be used for checking potential write-write conflicts and will make the given added tuple visible only to this transaction. @Param: String tname: the name of the table where the user wants to insert the tuples to. @Param: [] byte value: the tuple to be inserted, represented by an array of bytes.

Operation ID	3.6.5
Signature	public static void Conn.upd(long tid, String tname, byte [] value) throws kv.Error Updates a tuple in the given table
Input	@Param: long tid: the id of the current transaction, which will be used for checking potential write-write conflicts and will make the given modified tuple visible only to this transaction. @Param: String tname: the name of the table where the user wants to update the tuples from. @Param: [] byte value: the tuple to be updated, represented by an array of bytes.

Operation ID	3.6.6
Signature	public static void Conn.del(long tid, String tname, byte [] value) throws kv.Error Deletes a tuple in the given table
Input	@Param: long tid: the id of the current transaction, which will be used for checking potential write-write conflicts and will make the absence of the given deleted tuple visible only to this transaction. @Param: String tname: the name of the table where the user wants to update the tuples from. @Param: [] byte value: the tuple to be deleted represented by an

	array of bytes. The tuple can contain only the primary key of the tuple to be deleted. All other fields will be ignored.

Operation ID	3.6.7
Signature	public static long Conn.begin() Starts a transaction
Output	long: the unique identifier (tid) of the transaction that is started.

Operation ID	3.6.8
Signature	public static long Conn.commit(long tid) Commits a transaction
Input	long: the unique identifier (tid) of the transaction that the user wants to commit.
Output	The tid of the transaction.

Operation ID	3.6.9
Signature	public static long Conn.abort(long tid) Aborts (rollback) a transaction
Input	long: the unique identifier (tid) of the transaction that the user wants to rollback.
Output	The tid of the transaction

Operation ID	3.6.10
Signature	public Tuple(byte array) Public constructor that creates a Tuple, representing a record of the data store, given the representation of the latter in an array of bytes

Operation ID	3.6.11
Signature	public byte [] toBytes() Transforms a Tuple object to an array of bytes
Output	The representation of the tuple in an array of bytes.

For the streaming API, the interfaces are the following

Operation ID	3.6.12
Signature	public ResponseStatusJSON registerQuery(QueryDefinition queryDefinition) registers a new query in the CEP
Input	QueryDefinition: object representing the query using the JSON notation.
Output	A ResponseStatusJSON object. This object contains three fields: a String field with the query name, an integer with the response status code, and another String with the response text description.

Operation ID	3.6.13
Signature	public ResponseStatusJSON deployQuery(String queryDeployment) deploy a registered query in the CEP
Input	queryDeployment: String with the name of the query to deploy. This parameter will be different in the CEP distributed prototype.
Output	ResponseStatusJSON object. This object contains three fields: a String field with the query name, an integer with the response status code, and another String with the response text description.

Operation ID	3.6.14
Signature	public TupleSender registerToStream(RegisterToStream registerToStream)

	register with an input stream of a certain query.
Input	registerToStream: object with the query name and the stream name to which the client will connect.
Output	TupleSender object. It represents the stub that the client application uses to send tuples.

Operation ID	3.6.15
Signature	public SubscribeToStreamResponse subscribeToStream(SubscribeToStream subscribeToStream) subscribe with an output stream of a query in order to receive the result of the continuous computation..
Input	subscribeToStream: object with the query name and the stream name to which the client will subscribe.
Output	SubscribeToStreamResponseJSON object. This object contains information about the stream deployment.

More information regarding the API can be found at the D4.1 deliverable [7].

For the data imported component that inserts data elements represented in the FHIR/HHR format, the interfaces are the following:

Operation ID	3.6.16
Signature	public int upsertPatients(String patientXml) throws DataImporterException Insert a list of patients, in an HAPI-FHIR format, in the data store. If the patient already exists, then will update its values according to this input.
Input	@Param*:* String patientXml: The serialized xml which contains the list of patients, in the HAPI-FHIR format, to be inserted/updated in the data store.
Output	int: the number of rows affected (inserted or updated).

Operation ID	3.6.17
Signature	public int upsertSideEffects(String sideEffectsXml) throws DataImporterException Insert a list of side effects, in an HAPI-FHIR format, in the data store. If the side effect already exists, then will update its values according to this input.
Input	@Param*:* String sideEffectsXml: The serialized xml which contains the list of side effects, in the HAPI-FHIR format, to be inserted/updated in the data store.
Output	int: the number of rows affected (inserted or updated).

Operation ID	3.6.18
Signature	public int upsertComorbidities(String comorbiditiesXml) throws DataImporterException Insert a list of comorbidities, in an HAPI-FHIR format, in the data store. If the comorbidity already exists, then will update its values according to this input.
Input	@Param*:* String comorbiditiesXml: The serialized xml which contains the list of comorbidities, in the HAPI-FHIR format, to be inserted/updated in the data store.
Output	int: the number of rows affected (inserted or updated) .

Operation ID	3.6.19
Signature	public int upsertDisgs(String disgsXml) throws DataImporterException Insert a list of disg objects, in an HAPI-FHIR format, in the data store. If the object already exists, then will update its values according to this input.
Input	@Param*:* String disgsXml: The serialized xml which contains the list of objects, in the HAPI-FHIR format, to be inserted/updated in the data store.
Output	int: the number of rows affected (inserted or updated).

More information regarding this API can be found at the corresponding D4.10 deliverable [8].

Performed work

The data store API (LeanXcale JDBC, Direct Access API and Streaming API) are delivered on schedule, according to the integration plan as described in D6.1 [1]. No changes are foreseen for the forthcoming periods regarding the interface. Additional improvements and extensions will only affect the SQL dialect currently supported by the data store and the streaming API to support the deployment of parallel queries, in order to include the advanced functionalities required for the data lake support and for the online aggregations. The required development that is needed to support these features, as internally planned by the task T4.1 for the forthcoming reporting periods and described in D4.1, corresponds to the internal engine of the Big Data Platform, and will not affect the exposed interface to the other components.

Regarding the *Data Importer*, the delivered work currently covers basic scenarios using small amount of data, and it is planned to be extended according to the use cases' needs. Although the existing interface is not foreseen to be altered, additional functionalities will be provided in order to support the data importing of additional types of objects.

Identified issues and possible evolutions

The delivered Big Data Platform currently addresses all functional requirements described in D2.1 [2] and the delivered functionality is in compliance with the integration plan described in D6.1. However, in the context of CrowdHEALTH, the platform has not been evaluated yet in the project's identified real life scenarios, involving numerous end-users and various stakeholders, providing and requesting simultaneously big amount of data, and this might raise some issues regarding the satisfaction of some non-functional requirements. In an attempt to be pro-active and confront with these scenarios, the platform is already tested in order to evaluate its functionality and performance level under highly intensive workloads, using both and mixed OLTP and OLAP workloads, so the risk of not satisfying the non-functional requirements is considered extremely low. In any case, potential evolutions of the component with respect to its performance under highly intensive workloads will affect the internal engine of the platform and will not alter the provided interface.

3.7. Aggregator

Implemented functionalities

- Collection of FHIR data and forwarding it to the Big Data Platform.
- Aggregation of FHIR data in the Big Data Platform.

Integrated components

The Aggregator is composed of two components:

- Aggregator service that received FHIR data from the Data Converter component and forwards it to Big Data Platform.
- CrowdHEALTH data store that stores data into the Big Data Platform.

Technical interfaces

Aggregate Data in FHIR format

Operation ID	3.7.1
URI pattern	http://{domain}:{port}/api/aggregate ex: *http://icemain.hopto.org:7031/api/aggregate*
Methods	POST: Receives a FHIR document and aggregates its contents with Data store
Input	*Body*: FHIR document in XML format
Output	None

Performed work

First draft of the service with prototype functionality has been provided.

Identified issues and possible evolutions

Although the Big Data store has been deployed and tested successfully, at the integration time there are some authentication issues that restrict external access from the Aggregator service that feeds data to the Big Data Store. However, the queries to aggregate data in the Big Data Platform have been designed and upon the resolution of authentication issues, the connectivity between Big Data Store and Aggregator services can be easily established.

3.8. Multimodal Forecasting and Causal analysis module

Implemented functionalities

One of the main forecasting functionalities in the current version is the forecast of weight and the overall fitness status at the age of 18. In version one of causal analysis and forecasting module we based our development on the available datasets from SLOFIT project provided from University of Ljubljana use case and we developed three forecasting functionalities.

- Weighting forecast.
- Individual health parameter.
- Overall fitness the age of 18.

Integrated components

The main integrated components of the prototype v.1 are as follows:

- Data cleaning app: Responsible for tidying the data (fixing typos in names, merging cases where names and surnames are mixed, fixing date of birth). This prototype is used in pre-processing.
- Missing value fixer: Populates the missing values with extrapolated values in order for the models to work (as the models often rely on continuous data and cannot handle missing data well). This prototype is used in pre-processing.

Technical interfaces

Operation ID	3.8.1: Asses intervention
Signature	public float[] assesIntervention(float[] forcastedValues, Model interventionModel, String[] aggregationVariables)
Input	forcastedValues: Values that we try to change with intervention. interventionModel: what pre-built intervention model to use – has to be compatible with the forecast values. aggregationVariables: By what variables to group results – class, school, municipality
Output	Table of forecast values changed by the intervention

Operation ID	3.8.2: Prepare/modify data set
Signature	public DataSet createDataSet(Dataset ds, String[] independentVariables, String[] dependantVariables, StringtimeVariable, Time startTime, Time endTime, Time forecastStartTime, Time forecastEndTime, Rules transformationRules) Modify the existing data set, based on the selected input parameters.
Input	Ds: Original data set independentVariables: Independent variables used for training the model. dependentVariables: Dependent variables used for training the model. timeVariable: The name of the variable that represents the time value used for predictions. startTime: Defines which point in time is the starting point for the new data set. endTime: Defines which point in time is the ending point for the new data set. transformationRules: Set of rules for transforming the independent variables. Rules can include feature generation and feature selection.
Output	Output: New modified data set.

Operation ID	3.8.3: Build forecasting model
Signature	public Model buildForecastngModel(Dataset ds, String[] independentVariables, String[] dependantVariables, Algorithm forecastingAlgorithm) Builds a forecasting model based on the data.
Input	Ds: Dataset used for building a model independentVariables: Independent variables used for training the model. dependentVariables: Dependent variables used for training the model. forecastingAlgorithm: Type of machine learning algorithm used for

	building a forecasting model (e.g. SVM)
Output	Output: Forecasting model, that can be used for forecasts.

Operation ID	3.8.4: Make forecasts
Signature	public Dataset buildForecastngModel(Dataset ds, Model forecastingModel, String[] aggregationVariables, Time predictTo) Builds a forecasting model based on the data.
Input	Ds: Dataset of independent values to forecast. forecastingModel: Model used for making forecasts. aggregationVariables: By what variables to group results – class, school, municipality predictTo: Define to which point in time to predict.
Output	Output: Dataset of forecasts.

Performed work

The performed work in version 1 was based on the collaboration with the SLOFIT project from University of Ljubljana. Within version 1 we described the module architecture and the functionalities based on the provided datasets and the inputs from policy makers and researchers from the University of Ljubljana partner.

Identified issues and possible evolutions

Version one of multimodal forecasting and causal analysis module is limited only to SLOFIT use case because of the bounded data availability from use case partners during the first year of the project. In the following (in the second version), data from other use cases will be added to the module, as well as additional use case-specific functionalities.

Moreover, the interpretation with all the use case partners and the policy makers lead to the conclusion that the forecasting and causality module should be merged in one with different functionalities, instead of two different components as initially planned. This process will be described in the v2 prototype.

3.9. Risk Stratification tool

In D6.1, we planned to integrate prototypes of risk stratification tools in the first version of the CrowdHEALTH platform. However, these tools need to be developed by working directly with data from use case partners, involving a period of data analysis and then empirical procedures for training and evaluating machine learning classifiers to perform the risk stratification task. It has not been possible to access data from use case partners in time to do this work for this milestone. The first demonstration deliverable for the risk stratification tools (D5.5) has been postponed until M22.

3.10. Visualization component

Implemented functionalities

The first prototype of the visual workbench is foreseen to be released at M20 of the project. During the first year several functionalities have already been developed:

- Web application mock-up.
  - Graphical creation of SQL queries.
  - Graphical representation of SQL query results.
- Service Layer to connect with the data store.
  - Get list of tables.
  - Get schema of a table.
- Restful backend:
  - Create a table to store component-specific data.
  - Drop a table.
  - Truncate a table.
  - Get all available tables, so that the end-user of the visualization dashboard to be able to design analytical queries.
  - Get table info (i.e columns, foreign keys etc).
  - Get all available data types.
  - Insert a row to a table.
  - Delete a row from a table based on its primary key.
  - Update a row from a table, based on its primary key.
  - Get all rows from a table.
  - Get a row from a table based on its primary key.
  - Perform a get operation on a table, using a predicate on a field.

Integrated components

The Visualization component is integrated with the Big Data Platform, and it uses its exposed functionalities to perform both simply information retrieval using its Elastic JDBC driver, and Big Data Analytics operations.

User interfaces

Query Composer

The query composer can be used to create SQL query as well as Cep streaming queries.

Users can create SQL queries by adding and connecting operators into a graph. As an example the next figure where the user load the table named “Table2” as data source and then first apply a filter condition and finally she group the results.

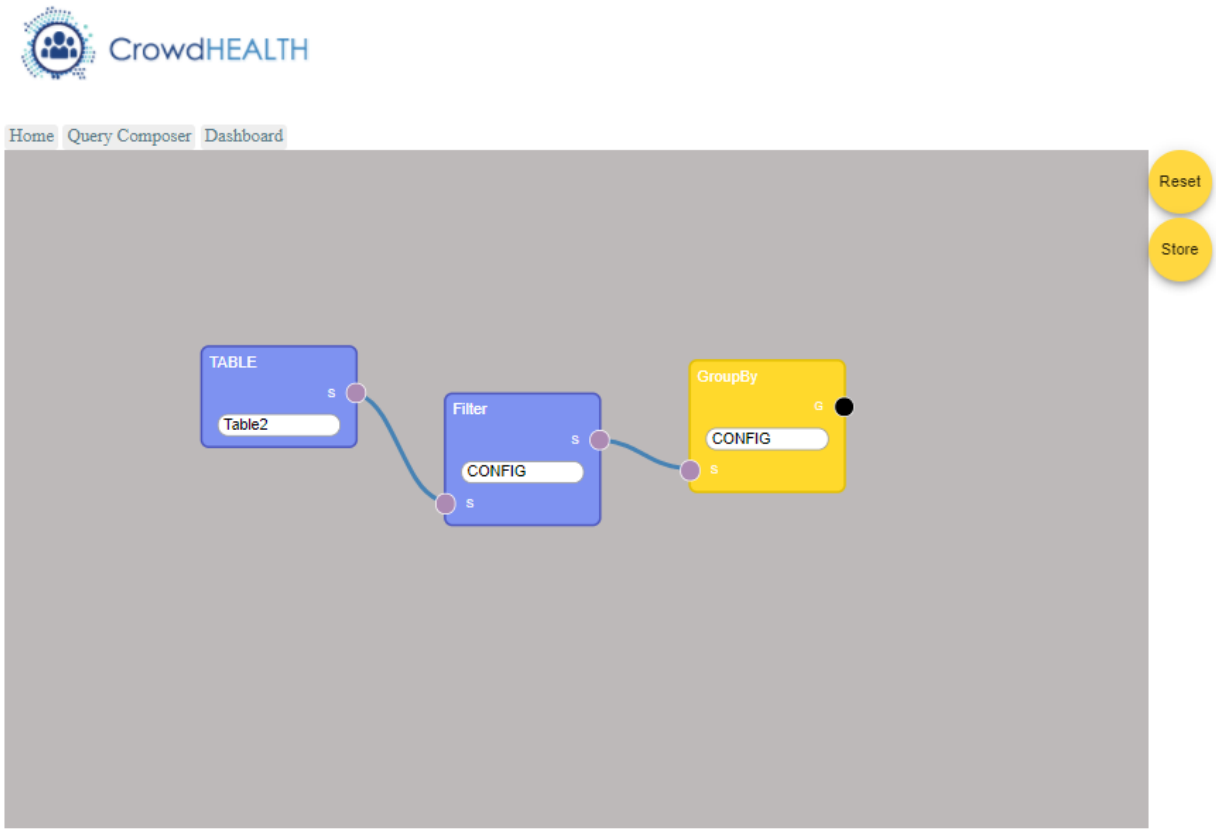

Figure 2 Query composition

The user can then visualize the result of the current query by attaching chart operators to the graph. For example, in the next picture the user decides to show the results of the query using a bar chart. In this particular case the user is visualizing the number of Patients from 4 different hospitals in the last 3 years.

Figure 3 Query results visualization

Dashboard

Using the dashboard users can see the results of stored queries by using different types of chart. In the following picture there is an example of a dashboard made by four different chart: line, tree-map, gauge and pie grid.

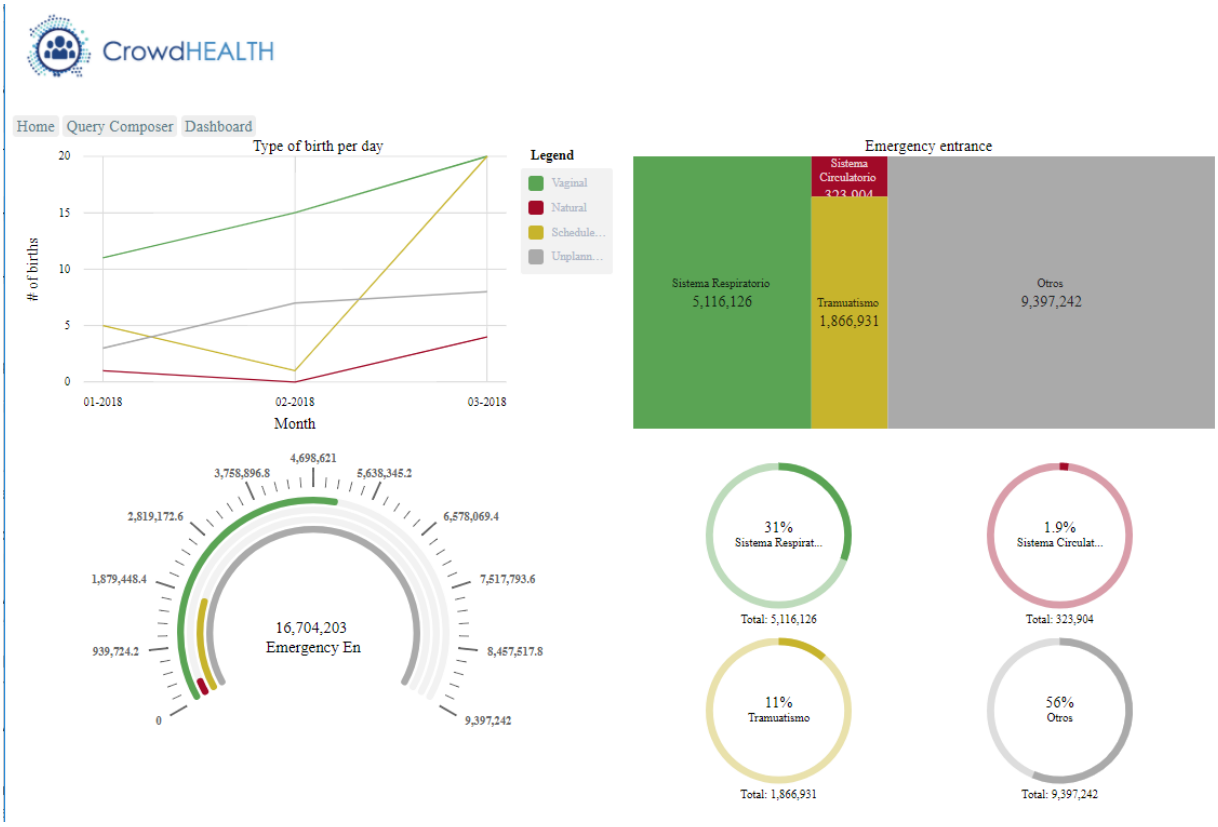

Figure 4 Dashboard

Performed work

The functionalities of the Visualization component are currently under development, as according to the DoA, its first prototype is planned to be delivered on M20. As a result, this component was not included into the integration plan of D6.1. However, some functionalities of the web application as well as the integration with the Big Data Platform has already been developed, and the performed work is considered aligned with the internal planning of T4.4.

Identified issues and possible evolutions

A critical aspect for the development of the Visualization component is its integration with the Big Data Platform, which is a crucial component for retrieving data in order to visualize the results of the data analytics to the dashboard. However, this work has already started and after preliminary integration tests no important issues are foreseen for the forthcoming period.

4. Conclusions

This report documents the achievements of the CrowdHEALTH system integration effort for the first integration cycle. An overview of the current version of the CrowdHEALTH system is reported in the section “First cycle architecture”, which shows the software components integrated during the first year of the project. The implemented and integrated functionalities have been described in detail for each component.

All software components planned in the integration plan [1] have been integrated, with respect to the realized functionalities, with the exception of the Risk Stratification Tool that has been postponed to the second integration cycle. Moreover, some planned functionality is still under development. In particular, it is not clear yet if cleaning of HHR data is needed or just cleaning of raw data as currently implemented is sufficient.

On the other hand, issues identified during the development have required the anticipation of functionalities originally planned for the second cycle of development. That is the case of the Visualization component to visualize the results of the data analytics to the dashboard, the XML serialization of the HHR objects, and the user authentication and authorization module, which are already started.

After preliminary integration tests of these components, some issue has been identified and already solved. Moreover, some improvement to be implemented in the second cycle of development have been identified. These include the possible addition of a network service over the data anonymization, improvements to SQL dialect supported by the data store, improvement of authentication functionalities.

5. References

[1] De Nigro A. et al., D6.1 Integration plan v1, Feb. 25, 2018, EC H2020 CrowdHEALTH project.

[2] Kyriazis D. et al., D2.1 State of the Art and Requirements Analysis v1, 2017, EC H2020 CrowdHEALTH Project.

[3] Kyriazis D. et al., D2.4 - Conceptual Model and Reference Architecture v1, 2017, EC H2020 CrowdHEALTH project.

[4] Auth0. JSON Web Token. [Online]. https://jwt.io

[5] Pivotal Software. Spring. [Online]. https://projects.spring.io/spring-security/

[6] University Health Network. HAPI FHIR. [Online]. http://hapifhir.io/

[7] Ricardo Jimenez-Peris et al., D4.1 Scalable Big Data Management: Design and Specification v1, 2017, EC H2020 CrowdHEALTH project.

[8] Wajid U. Kranas P., D4.10 Generating and Analysing Knowledge Framework: Software Prototype I, Jan. 10, 2018, EC H2020 CrowdHEALTH project.