CrowdHEALTH

Collective Wisdom Driving Public Health Policies

D4.10 Generating and Analysing Knowledge Framework: Software Prototype I

Project Deliverable

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

D4.10 Generating and Analysing Knowledge Framework: Software Prototype I

Work Package:		WP4
Due Date:		31/12/2017
Submission Date:		10/01/2018
Start Date of Project:		01/03/2017
Duration of Project:		36 Months
Partner Responsible of Deliverable:		ICE
Version:		1.1
Status:	☒ Final ☐ Draft ☐ Ready for internal Review ☐ Task Leader Accepted ☐ WP leader accepted ☐ Project Coordinator accepted	
Author name(s):	Usman Wajid	Pavlos Kranas
Reviewer(s):	Dimosthenis Kyriazis	Salvador Tortajada Velert
Nature:	☐ R – Report ☒ D – Demonstrator	
Dissemination level:	☒ PU – Public ☐ CO – Confidential ☐ RE – Restricted	

REVISION HISTORY			
Version	Date	Author(s)	Changes made
0.1	21/12/2017	Usman Wajid	Document structure and initial content
0.2	28/12/2017	Pavlos Kranas	Input regarding the interface and implementation between the Aggregator and the Data Repository
1.0	28/12/2017	Usman Wajid	Executive summary and overall alignment of sections
1.1	10/01/2018	ATOS	Quality Check. Submission to EC.

Acronyms

API	Application Programming Interface.
CI	Continuous Integration.
DB	Database.
FHIR	Fast Healthcare Interoperability Resources.
JAR	Java Archive.
MVC	Model-view-controller.
REST	Representational State Transfer.
WAR	Web Archive.
XML	Extensible Markup Language.

Contents

1. Executive Summary

This document describes the first prototype of the Data Aggregation component. This component is the outcome of Task 4.2 of CrowdHEALTH project. The first prototype of the Data Aggregation component is composed of two sub-components (a) Data Aggregation, and (b) Data Importer. The Data Aggregation sub-component is responsible for receiving the data from other components in the CrowdHEALTH platform, whereas the Data Importer component is responsible for importing the data within the big data platform.

This document provides an overview of the implemented functionality and vital information about the interfaces of Data Aggregation with other components. The document also describes how to access the source code and exploit the prototype. Finally, it highlights the baseline technologies underpinning the prototype.

The first prototype of the Data Aggregation component completes the flow of health data from its source to the big-data store within CrowdHEALTH platform, from where it will be used for analytics and decision support. Two more iterations of the delivered prototype are planned in the lifetime of the CrowdHEALTH project. The future prototypes will refine and enhance existing functionality with the view to provide a scalable and reliable aggregation solution for health data from heterogeneous sources.

2. Prototype overview

2.1. Main components of the prototype

2.1.1. Data Aggregation

The first prototype of the Data Aggregation component is delivered as a Maven Project. The project called 'Aggregator' implements a Data Aggregation service capable of receiving FHIR data and passing it through to the Big Data Platform for insertion and aggregation.

2.1.2. Data Importer

This component receives data coming from the *data aggregator*, in a HAPI-FHIR compatible format, and *upserts* (inserts or updates) them in the CrowdHEALTH data repository. It firstly verifies if the corresponding database schema is already defined in the data store, and if not, it creates it. Then, it is used by the *data aggregator* to push data to the repository. It translates the input data to the underlying schema, by creating the corresponding java objects, and makes use of its internal database accessibility methods to store and update data. It encapsulates all the process underneath, with the respect to the software principle of *separation of concerns.* By doing so, it allows the *data aggregator* to simply implement the business logic of collecting/aggregating/transforming data as needed, while the *data importer* implements the necessary mechanisms to store the data in a format that will further allow their ease exploitation by the CrowdHEALTH tools for analytical processing.

The first prototype of the *Data Importer* is implemented in Java as a Maven project, thus providing its binaries as a Maven Artifact. The *data aggregation* can make use of it, by simply adding to its classpath.

2.2. Interfaces

2.2.1. Data Aggregation

An instance of the Aggregator is currently hosted as a REST service with the following interface: icemain.hopto.org:7031/api/aggregate. Other components can use the specified interface/endpoint to send data for aggregation.

2.2.2. Data Importer

As already mentioned, the *Data Importer* is implemented using Maven, thus is packaged into a Java Archive (jar) that can be used or integrated with the Data Aggregation subcomponent by adding the jar file into the project's classpath, or adding the corresponding maven dependency

to Data Aggregation's pom.xml configuration file. For the first prototype, the maven dependency that needs to be added is the following:

```
<dependency>
    <groupId>eu.crowdhealth</groupId>
    <artifactId>data.importer</artifactId>
    <version>0.1-SNAPSHOT</version>
</dependency>
```

The *Data Importer* provides a simple way to store a list of CrowdHEALTH data items, in a HAPI-FHIR compatible format, to the internal data store. It provides a singleton instantiation of the eu.crowdhealth.importer.utils.DataImporter class that exposes corresponding methods that allows the *upsertion* of data. This API signature of this class is the following:

- public static DataImporter getInstance(): A static method that returns the singleton instantiation of the class. It checks if the class has already been created, and if not, it initializes the object, while taking care of all the actions that need to have been completed beforehand (i.e loads the driver for the data store, ensures the DB schema is defined in the data store, etc). If an instance of this class had already been initiated before the invocation of this method, it will always return a reference to that object. It ensures concurrency semantics and thus, it is thread-safe.
- public int upsertPatients(String patientXml) throws ImporterException: It receives a serialized list of objects that holds patient information and store them to the data repository. It returns the number of records that were affected (either newly inserted or updated). If a specific item in the list of object fails to be added, it will continue to the next items, and will log the error, so it will not break the insertion flow. However, It can throw an ImporterException in case of a major failure (i.e input String was not in a valid format, could not establish connection with the data store, etc). A sample input String can be found in the appendix of this document.
- public int upsertSideEffects(String sideEffectsXml) throws ImporterException. As above, with the difference that the list of objects holds information regarding side effects of disease. It additionally verifies if the corresponding disease is already imported in the data repository, and if not, it firstly adds it. It takes care establishing the necessary dependencies regarding foreign keys of the involved entities. If a patient that refers to a disease has not been already added to the repository, it will ignore the specific item of the list, will log an error, and will continue to the next item. A sample input String can be found in the appendix of this document.
- public int upsertComorbidities(String comorbiditiesXml) throws ImporterException: As above, regarding comorbidity information. A sample input String can be found in the appendix of this document.
- public int upsertDisgs(String disgsXml) throws ImporterException. As above, with a sample input string that can be found in the appendix of this document.

2.3. Baseline technologies and tools

2.3.1. Baseline technologies and tools for Data Converter

The following baseline technologies and tools are used by the first prototype of the Data Aggregation sub-component.

- **FHIR HAPI Server** is an open-source implementation of the FHIR specification in Java. HAPI provides a built-in mechanism for adding FHIR's RESTful Server capabilities to the Aggregator. The HAPI RESTful Server is Servlet based, so it makes it easy to deploy to any of the many compliant containers that exist e.g. Docker.
- **Spring Framework** is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. The Aggregator service uses the Spring Web model-view-controller (MVC) framework to dispatch requests to handlers that use configurable handler mappings to forward the data to the Big Data Platform for aggregation.
- **Docker** is an open source project that allows the deployment of applications or services inside containers, adding a layer of abstraction. CI, Version Control, Portability, Isolation and Security. Containers are virtual machines running an operating system and environment of the developers choice, working this way allows for continuous integration and version control with the ability to quickly deploy and roll back versions, portability as docker is widely available on a number of host operating systems, the ability to isolate sub-components within their own containers leading to well defined interfaces and also enhancing system security. These feature allow a stable deployment environment for the Aggregator service.

2.3.2. Baseline technologies and tools for Data Importer

The following baseline technologies used by the first prototype of the *Data Importer sub-*component.

- **FHIR HAPI Structures** contains the definition of the data model used by FHIR HAPI compliant components. This will allow the interaction with the data aggregator which sends objects transformed in this format.
- **LeanXcale Elastic Driver** is the software tool that allows data access with the data repository.

3. Source code

3.1. Availability

The source code of the Data Aggregation component delivered through the Aggregator project, available under the following CrowdHEALTH GIT repository:

- **Aggregator:** http://crowdhealthtasks.ddns.net/CrowdHEALTH/Aggregator/
- **Importer:** https://crowdhealthtasks.ddns.net/CrowdHEALTH/DataImporter

3.2. Exploitation

The Aggregator is currently deployed on partner resources and can be access through the specified address.

- **Aggregation Service:** The source code of the Aggregator is available at the master branch of the CrowdHEALTH GIT, under the Aggregation project repository. It is structure like follows:
  - **pom.xml file:** descriptor of the Maven project.
  - **Readme.md file:** Instructions on how to clone the project from GIT, run it on local and on Docker.
  - **src/ folder:** split in 2 subfolders
    - **main/ folder:** contains all the source code of the Aggregator, including Java classes, properties files and service descriptors used by the Spring Web MVC.
    - **test/ folder:** contains all the Unit Testing classes, properties files, example files of FHIR inputs to be aggregated.
  - ***.sh scripts:** Set of scripts to run, build and publish the service
    - **common.sh script:** stores basic data like the name of the Docker repository, image name, container name and version.
    - **docker-build.sh script:** performs a 'Docker build' operation to compile the source code, package it as a WAR and construct the Docker image.
    - **docker-run.sh script:** performs a 'Docker run' that creates a Docker container that harbours the Aggregator service and exposes the 8080 port of an internal Tomcat to 7031 (defined in commons.sh).

The Data Importer is not intended to be used independently, as a separate service endpoint, instead, it is packaged as a Java Archive to be used by the *Data Aggregator.* The source code is available at the master branch of the CrowdHEALTH GIT, under the Data Importer repository, and is structured as a maven project.

4. Appendix

The patientXML sample input (used for test insertion in the datastore):

```
<Bundle xmlns="http://hl7.org/fhir">
  <entry>
    <resource>
      <Patient xmlns="http://hl7.org/fhir">
        <extension url="http://hl7.org/fhir/StructureDefinition/identifier-validDate">
          <valueTime value="Sat Apr 01 12:55:00 UTC 2017"/>
        </extension>
        <identifier>
          <system value="https://www.careacross.com/"/>
          <value value="1"/>
        </identifier>
        <telecom>
          <system value="email"/>
          <value value="testemail@gmail.com"/>
        </telecom>
      </Patient>
    </resource>
  </entry>
  </Bundle>
```

The side effects, comorbidities and disg sample input:

```
<Bundle xmlns="http://hl7.org/fhir">
  <entry>
    <resource>
      <Condition xmlns="http://hl7.org/fhir">
        <code>
          <coding>
            <system value="http://hl7.org/fhir/sid/icd-9-cm"/>
            <code value="780.71"/>
            <display value="Chronic fatigue syndrome"/>
          </coding>
        </code>
        <subject>
          <identifier>
            <system value="https://www.careacross.com/"/>
            <value value="1"/>
          </identifier>
        </subject>
      </Condition>
    </resource>
  </entry>
 </Bundle>
```