Master Person Index (dohMPI) Which Aids with De-duplication and Maintaining Consistent ID for its Data Sources; Such as the Death Information System, EDEN

CS6440 Fall 2017 Course Project 11/29/17

Team: SPYNdoctors (Nargis Bisset, Arjun Puri, Hernando Salas, and Alex Yanovsky)

Mentors: Dr. Kailah Davis, Dr. Jeffrey Duncan, Utah Department of Health

Project TA: Paul Miller

Github link: https://github.com/CDCgov/GaTech-Fall2017-Davis-dohMPI-SPYNdoctors

Table of Contents

1. General Background and Main Goal of the Project

The project is concerned with three major parts of Utah Department of Health informatics infrastructure – EpiTrax, EDEN and DohMPI.

EpiTrax is a Java web application which serves the need of the specialists in epidemiology and provides them with access to information on patients' conditions as well as treatments accumulated in several databases on particular infectious diseases. EDEN is the database underlying the Utah's state-wide death registration system, which among other things, records cause of death, any antecedent causes, and co-morbid conditions. These pieces of information are in many cases critical for epidemiological research, however, they are not currently accessible to EpiTrax users, because of the difficulties in direct mapping of personal information in EpiTrax to that in EDEN. Fortunately, Utah informatics infrastructure includes a special tool, called DohMPI (Department of Health Master Person Index), which can be used to identify a patient within or across participating systems . Therefore, the goal of this project was to provide the web service, which would allow EpiTrax, to resolve any possible ambiguities in personal patients' data through DohMPI and thus link the body of its patient information to the detailed death information available from EDEN. The achievement of this goal allows to incorporate death information into EpiTrax, which would is instrumental for many important use cases such as improving quality of case investigations and case management among epidemiologists by not taking time and resources to contact deceased individuals.

2. Project requirements, input/output specifications

According to the requirements, the input fields may include:

- **First name**
- **Last Name**
- **Date of birth**
- **Gender**
- **Address (Street number)**
- **Address (Street name)**
- **Address (Apartment number)**
- **City**
- **Zip code**
- **Phone number**

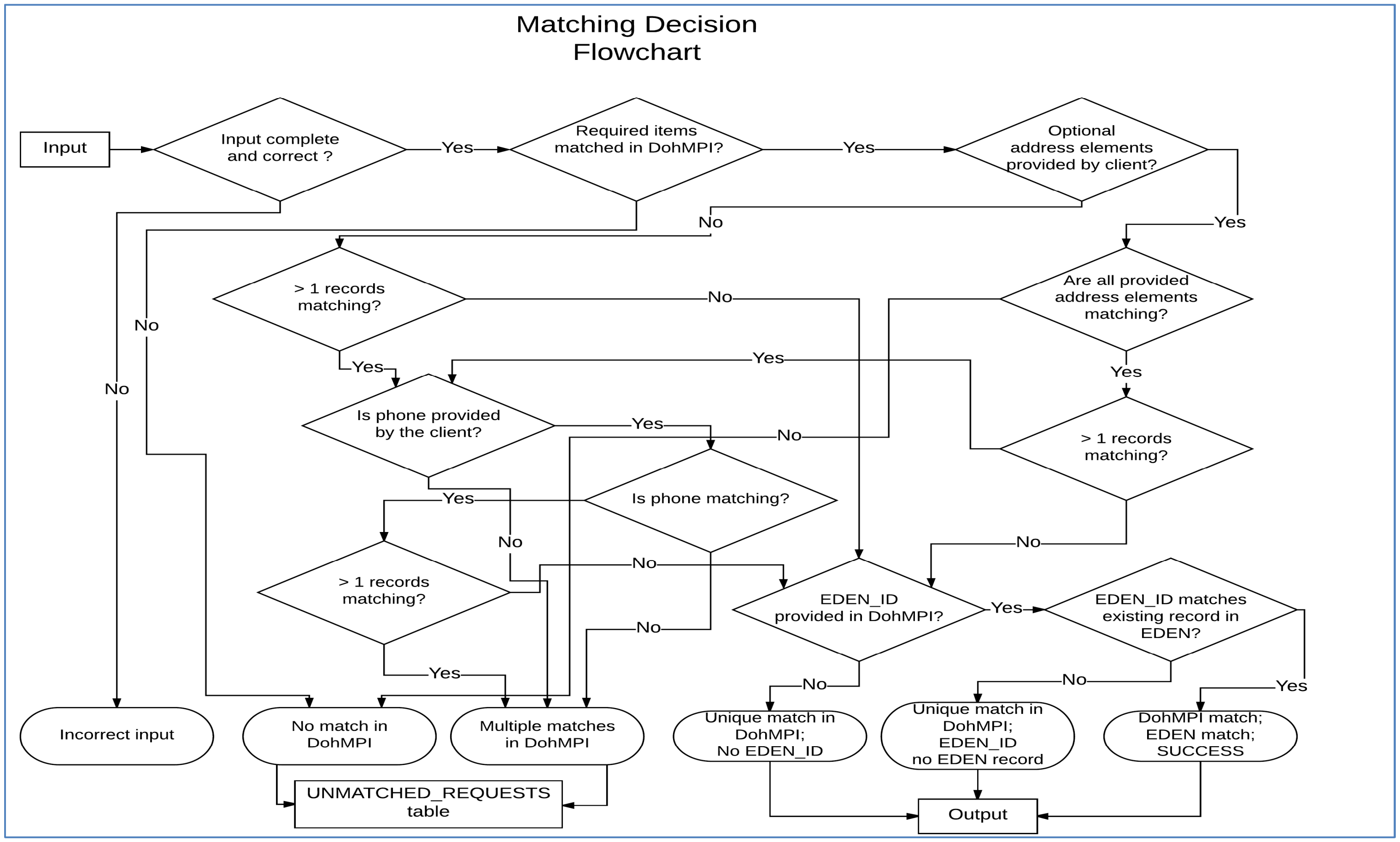
First name, **Last name** and **Date of birth** are always required. In addition, according to mentors' requirements, either **Gender** or both **Address (Street number)** and **Address (Street name)** are required. The rest of the fields are optional. All, but the **Phone number**, need to match if supplied. **Phone number** does not need to match, even if supplied, but can be used to narrow down in case of multiple matches.

The output should provide XML string, the exact contents of which will depend on the outcome of the personal data matching process (see details in the next section). At the minimum, it should have one populated field specifying the outcome of the matching; in case of successful match to the person in DohMPI, MPI_ID and personal data as they appear in DohMPI ("golden data") should be populated; if EDEN record for the matched individual is located as well, then immediate cause of death, along with additional causes are populated (see example of the output below).

In addition, in case a valid request was not matched in DohMPI, the row in the UNMATCHED_REQUESTS table should be generated recording the specified fields for future analysis of matching problems.

3. Matching Decision Logic

Based on the above requirements we suggested the following matching decision logic flowchart:

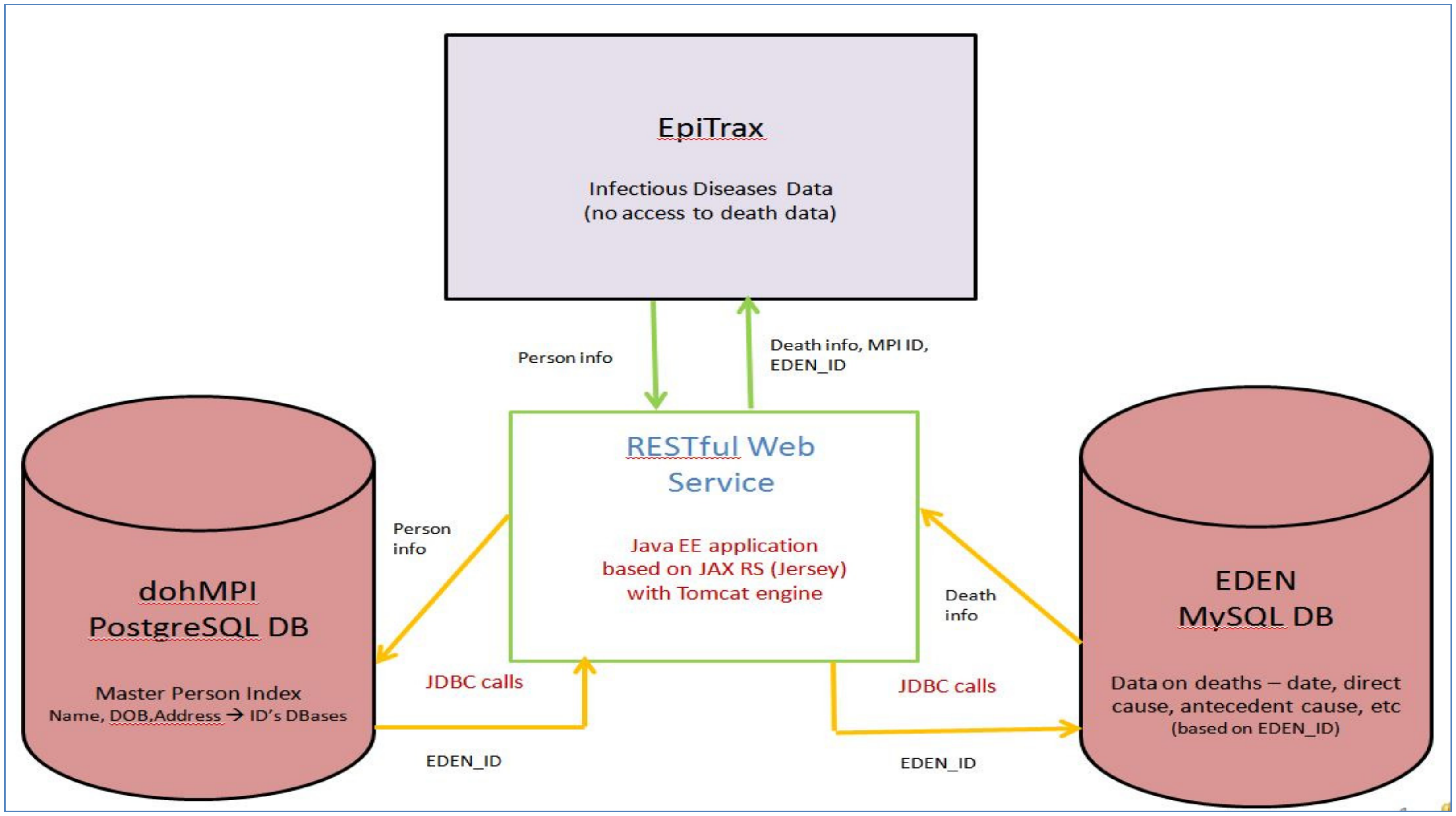


Matching Decision Flowchart
Input
Input complete and correct ?
Yes
Required items matched in DohMPI?
Yes
Optional address elements provided by client?
No
Yes
No
No
> 1 records matching?
No
Are all provided address elements matching?
Yes
Yes
Yes
Is phone provided by the client?
Yes
No
> 1 records matching?
Is phone matching?
Yes
No
No
> 1 records matching?
No
EDEN_ID provided in DohMPI?
Yes
EDEN_ID matches existing record in EDEN?
No
Yes
No
No
Yes
Incorrect input
No match in DohMPI
Multiple matches in DohMPI
Unique match in DohMPI; No EDEN_ID
Unique match in DohMPI; EDEN_ID no EDEN record
DohMPI match; EDEN match; SUCCESS
UNMATCHED_REQUESTS table
Output

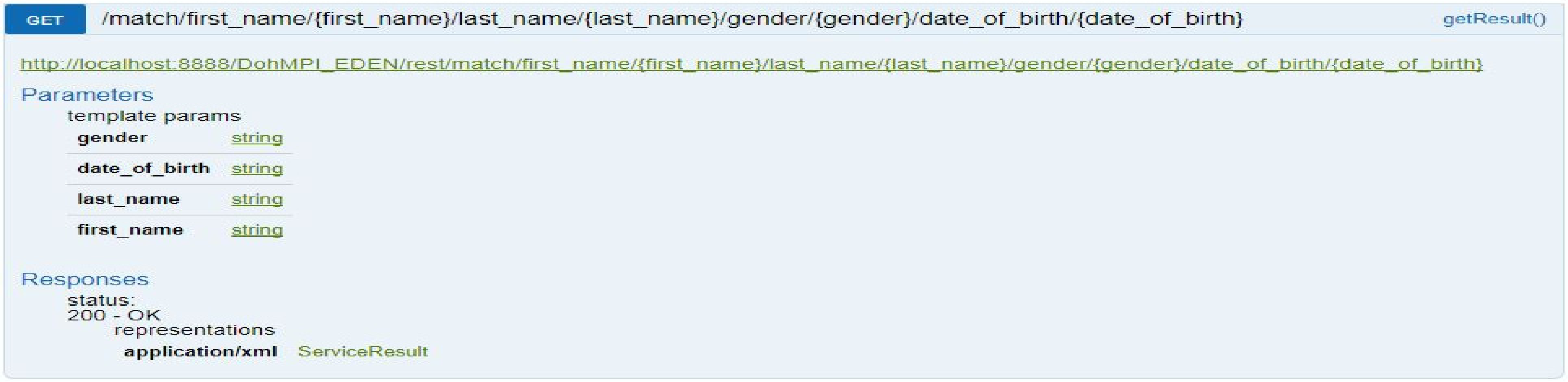
This diagram shows six distinct potential outcomes, i.e. incorrect input (missing or incorrectly formatted fields), request not matched in DohMPI (no matches or multiple matches), request matched in DohMPI, but no corresponding record in EDEN, and finally full match in both DohMPI and EDEN. Incorrect input results in XML string just informing about the detected problems in the fields, not matched requests end up adding a row in UNMATCHED_REQUESTS table, in the case of requests matched in DohMPI, XML string with suitably populated fields is returned.

4. Technical Implementation

The project deliverable represents RESTful service implemented as Java web application taking advantage of Jersey technology following the JAX-RS API specification. The service connects to DohMPI (Postgres), EDEN (MySQL) and (potentially separate) UNMATCHED_REQUESTS (Postgres) databases using JDBC technology and achieves performance improvement in case of heavy use of the service by employing connection pooling as implemented in Apache Tomcat DBCP library.


EpiTrax
Infectious Diseases Data
(no access to death data)
Person info
Death info, MPI ID, EDEN_ID
RESTful Web Service
Java EE application based on JAX RS (Jersey) with Tomcat engine
Person info
Death info
dohMPI PostgreSQL DB
Master Person Index
Name, DOB,Address → ID's DBases
JDBC calls
EDEN_ID
JDBC calls
EDEN_ID
EDEN MySQL DB
Data on deaths – date, direct cause, antecedent cause, etc
(based on EDEN_ID)

It provides two endpoints, one of which is available for POST method and can handle all input fields and another is available for GET method and can handle the restricted input of only required fields (First name, Last name, Date of birth, and Gender). This is the summary of the RESTful API generated using SWADL API documentation generator [1]:


GET /match/first_name/{first_name}/last_name/{last_name}/gender/{gender}/date_of_birth/{date_of_birth} getResult()

http://localhost:8888/DohMPI_EDEN/rest/match/first_name/{first_name}/last_name/{last_name}/gender/{gender}/date_of_birth/{date_of_birth}

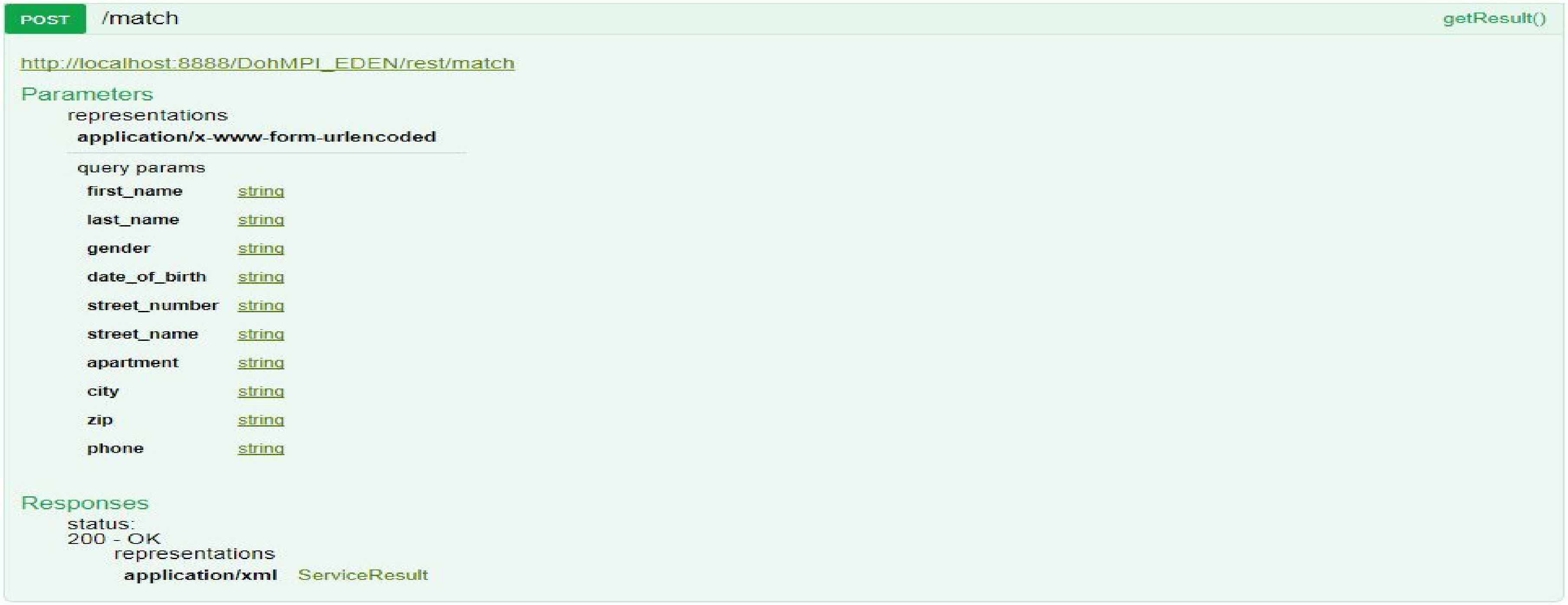
Parameters

template params

gender	string
date_of_birth	string
last_name	string
first_name	string

Responses

status:
200 - OK
representations
application/xml ServiceResult


POST /match getResult()

http://localhost:8888/DohMPI_EDEN/rest/match

Parameters

representations

application/x-www-form-urlencoded

query params

first_name	string
last_name	string
gender	string
date_of_birth	string
street_number	string
street_name	string
apartment	string
city	string
zip	string
phone	string

Responses

status:
200 - OK
representations
application/xml ServiceResult

5. Clients for Service Testing

In order to illustrate the use of the provided service, along with the working server side application we provided the HTML form which would allow to test the operation of the web service using the browser as a client, along with the stand-alone Java client application.

The HTML form will allow to test the endpoint using the POST method; the required input is obvious from the form itself. All fields, both required and optional, may be provided in the form. The endpoint using GET method accepts only basic required fields; when using GET method, the date of birth within the URL should be provided without slashes, e.g. 06221969, instead of 06/22/1969, as when using POST. The source code of the Java client, together with the whole set of project files including required libraries, will be part of the deliverables, which is intended to facilitate the use of the web service by its customers.

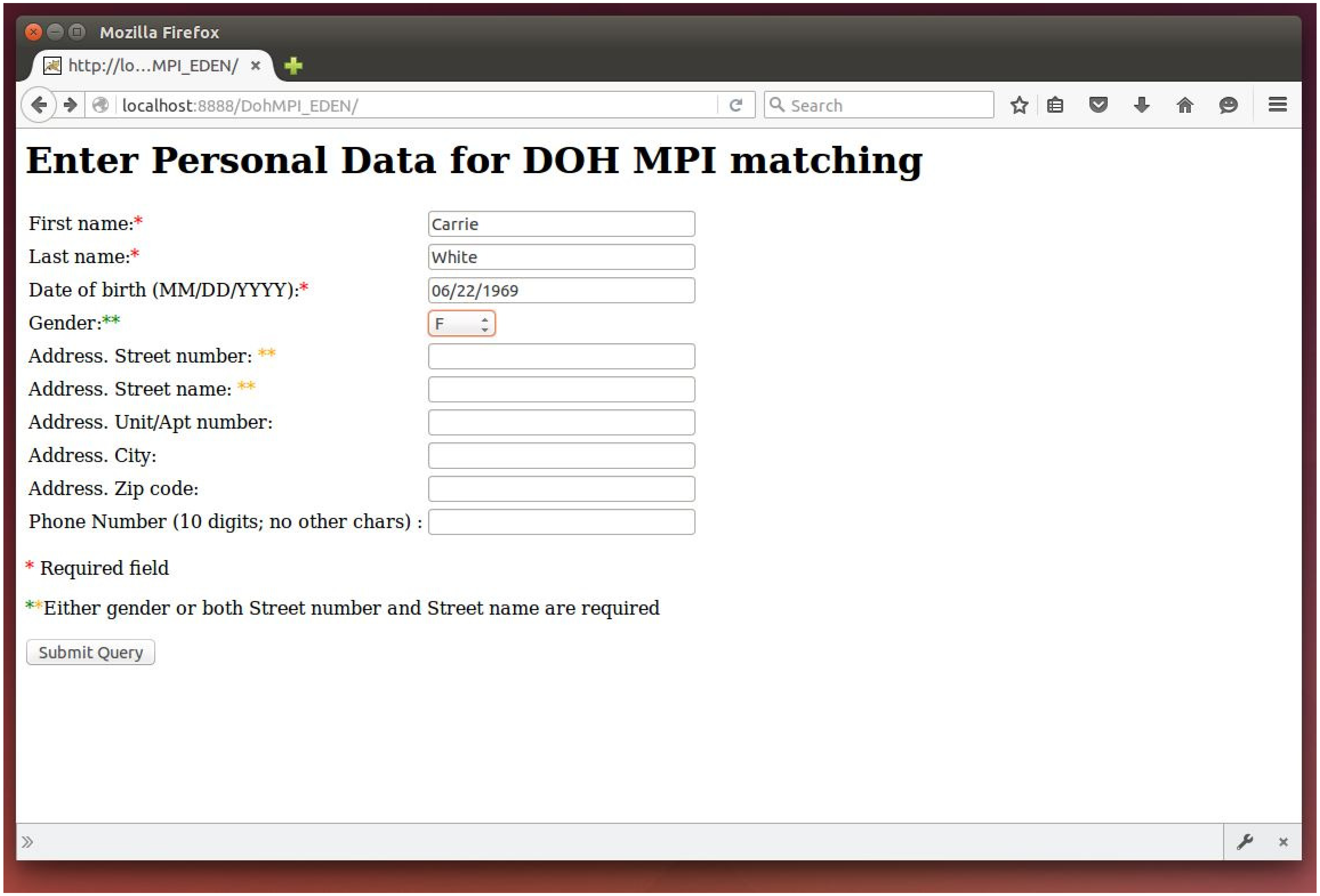
Stand-alone client provides two separate tabs for testing POST and GET endpoint. In this case slashes should be provided within the date of birth in case of GET as well as in case of POST (the client code will take care of removing them before submitting GET). Rendering of XML file in Java client application is using the XMLEditorKit library[2].

Please, see some additional examples on the testing of service and the use of the clients in Special Instructions – SPYNdoctors.pdf.

Below are the snapshots demonstrating the input and output screens for both web, as well as stand-alone Java, service clients.

In Mozilla Firefox browser:

Input form:



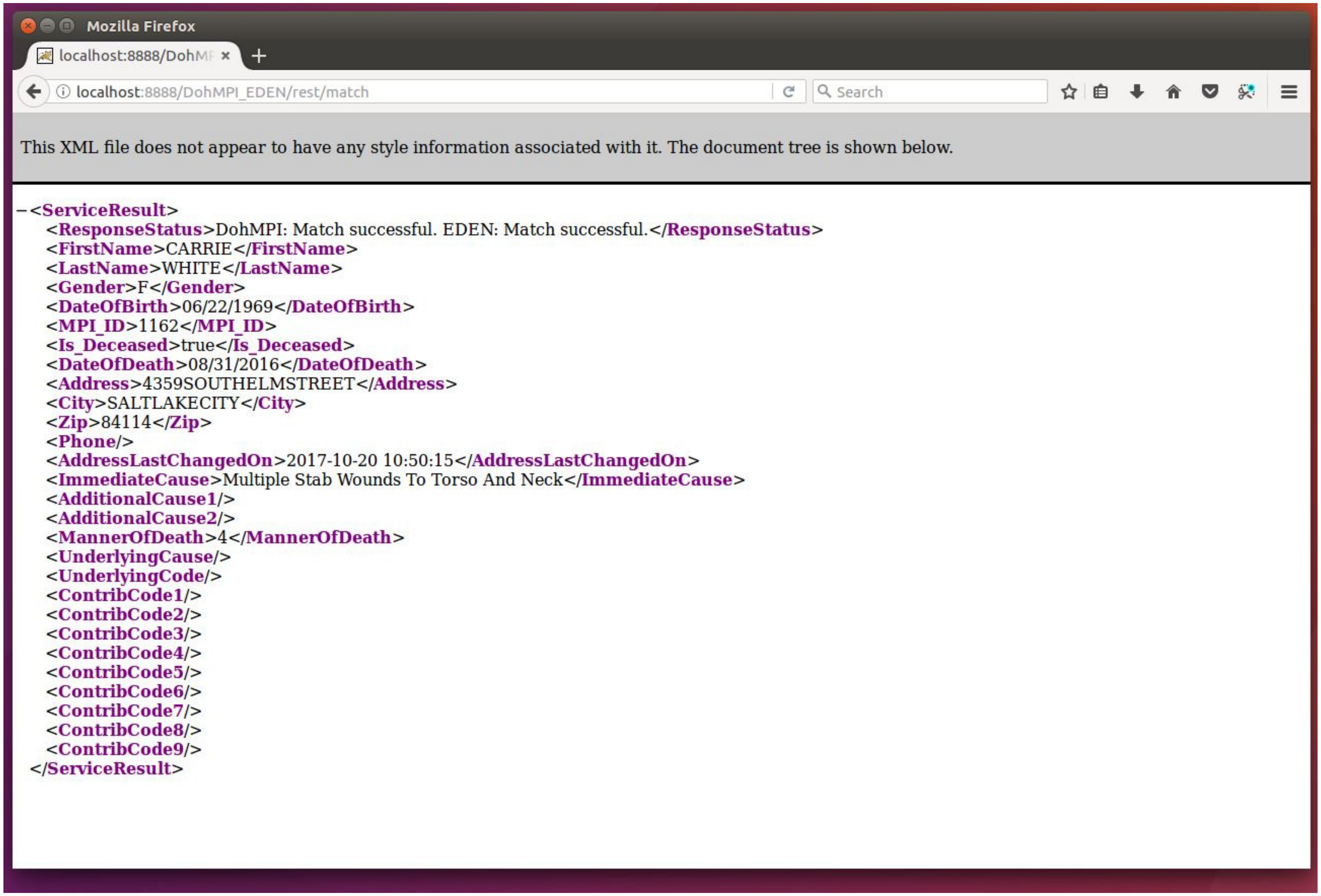
Enter Personal Data for DOH MPI matching

First name:*	Carrie
Last name:*	White
Date of birth (MM/DD/YYYY):*	06/22/1969
Gender:**	F
Address. Street number: **	
Address. Street name: **	
Address. Unit/Apt number:	
Address. City:	
Address. Zip code:	
Phone Number (10 digits; no other chars) :	

* Required field

**Either gender or both Street number and Street name are required

Service result:

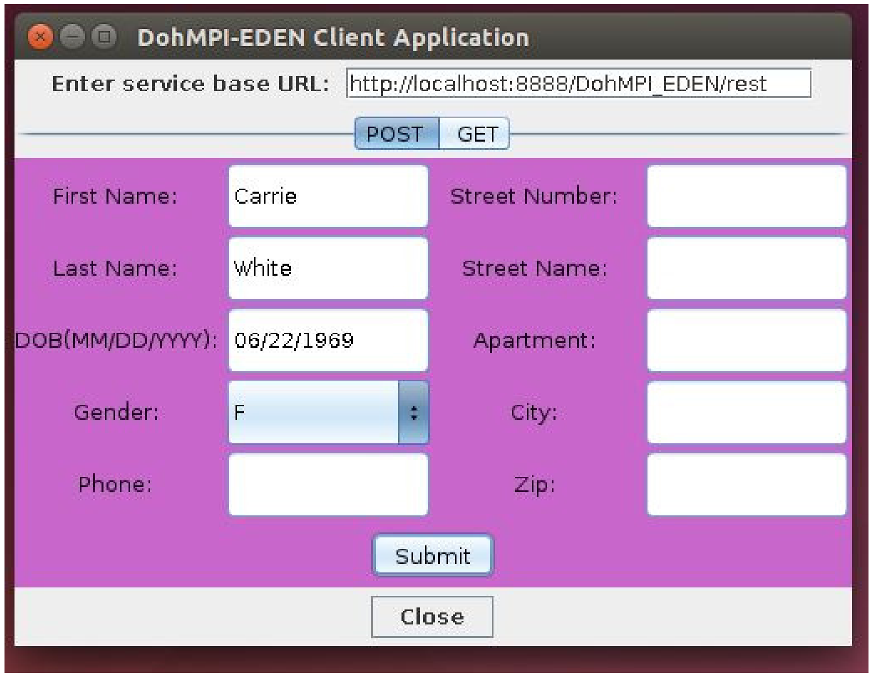


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<ServiceResult>
   <ResponseStatus>DohMPI: Match successful. EDEN: Match successful.</ResponseStatus>
   <FirstName>CARRIE</FirstName>
   <LastName>WHITE</LastName>
   <Gender>F</Gender>
   <DateOfBirth>06/22/1969</DateOfBirth>
   <MPI_ID>1162</MPI_ID>
   <Is_Deceased>true</Is_Deceased>
   <DateOfDeath>08/31/2016</DateOfDeath>
   <Address>4359SOUTHELMSTREET</Address>
   <City>SALTLAKECITY</City>
   <Zip>84114</Zip>
   <Phone/>
   <AddressLastChangedOn>2017-10-20 10:50:15</AddressLastChangedOn>
   <ImmediateCause>Multiple Stab Wounds To Torso And Neck</ImmediateCause>
   <AdditionalCause1/>
   <AdditionalCause2/>
   <MannerOfDeath>4</MannerOfDeath>
   <UnderlyingCause/>
   <UnderlyingCode/>
   <ContribCode1/>
   <ContribCode2/>
   <ContribCode3/>
   <ContribCode4/>
   <ContribCode5/>
   <ContribCode6/>
   <ContribCode7/>
   <ContribCode8/>
   <ContribCode9/>
 </ServiceResult>
```

Stand-alone Java client application:

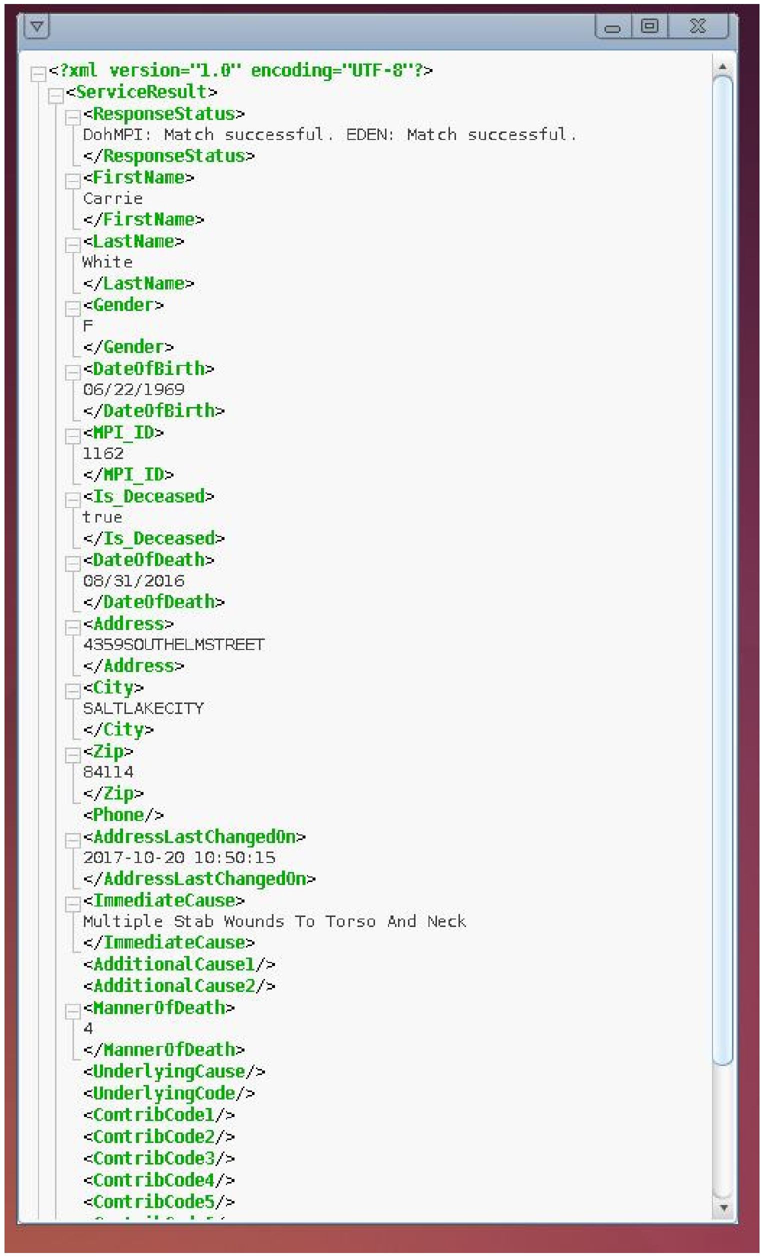
Input and output windows



Enter service base URL: http://localhost:8888/DohMPI_EDEN/rest

POST GET

First Name:	Carrie	Street Number:	
Last Name:	White	Street Name:	
DOB(MM/DD/YYYY):	06/22/1969	Apartment:	
Gender:	F	City:	
Phone:		Zip:	



```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceResult>
  <ResponseStatus>
  DohMPI: Match successful. EDEN: Match successful.
  </ResponseStatus>
  <FirstName>
  Carrie
  </FirstName>
  <LastName>
  White
  </LastName>
  <Gender>
  F
  </Gender>
  <DateOfBirth>
  06/22/1969
  </DateOfBirth>
  <MPI_ID>
  1162
  </MPI_ID>
  <Is_Deceased>
  true
  </Is_Deceased>
  <DateOfDeath>
  08/31/2016
  </DateOfDeath>
  <Address>
  4359SOUTHELMSTREET
  </Address>
  <City>
  SALTLAKECITY
  </City>
  <Zip>
  84114
  </Zip>
  <Phone/>
  <AddressLastChangedOn>
  2017-10-20 10:50:15
  </AddressLastChangedOn>
  <ImmediateCause>
  Multiple Stab Wounds To Torso And Neck
  </ImmediateCause>
  <AdditionalCause1/>
  <AdditionalCause2/>
  <MannerOfDeath>
  4
  </MannerOfDeath>
  <UnderlyingCause/>
  <UnderlyingCode/>
  <ContribCode1/>
  <ContribCode2/>
  <ContribCode3/>
  <ContribCode4/>
  <ContribCode5/>
```

6. Deployment Details

The web service is packaged as a Java web application and, as such, is delivered in the form of the .war file. The DohMPI_EDEN.war should be placed in the webapps folder of Tomcat installation and will be exploded as soon as Tomcat is restarted. The administrator of the service will also need to supply the file with details of database connections (DohMPI, EDEN, and UNMATCHED). The location of the file will be communicated to Tomcat using CATALINA_OPTS environment variable. This is an example of setting CATALINA_OPTS variable before launch of Tomcat and the contents of corresponding matcher.properties file:

```
export CATALINA_OPTS=-DMatcherPropertiesFilePath=/usr/local/tomcat/matcher.properties
```

The contents of the matcher.properties file :

```
* * *___***___
dohMPIhost=postgresdb
dohMPIport=5432
dohMPIuser=postgres
dohMPIpassword=cs6440
dohMPIdatabase=postgres
EDENhost=mysqldb
EDENport=3306
EDENuser=edenuser
EDENpassword=cs6440
EDENdatabase=eden
dohUnmatchedHost=postgresdb
dohUnmatchedPort=5432
dohUnmatchedUser=postgres
dohUnmatchedPassword=cs6440
dohUnmatchedDatabase=postgres
***____****____
```

The DohMPI_EDEN.war was built according the J2EE standards, so it should operate under any J2EE compliant Java web container, such as JBoss/WildFly, Glassfish, etc. However DohMPI_EDEN.war was tested only with Tomcat.

7. Submission for Grading (Docker)

The project is submitted for grading as the docker-compose.yml file and edenpopulate.sql file.

The docker-compose up command should spin up four containers based on the images pull from hub.docker.com.

1. The my_postgres_full image is built on the basis of official docker postgres image and will run as a container with postgres database pre-populated with schema and test data representing DohMPI database. The UNMATCHED_REQUESTS table in this grading situation is being written to the same postgres database as DohMPI, though separate set of connection details in matcher.properties file will allow the administrator of the production application to use for that purpose separate database on separate host completely independent of the location of DohMPI.

2. The mysql_perm image will spin up mysql database container and populate it during the docker-compose up execution with the schema and data representing EDEN database.
3. The my_webserver_notest image will spin up the container running Tomcat with DohMPI_EDEN web service application expanding and starting execution at the start of the server. In order to accommodate the docker-compose up situation, the application will be waiting and checking every second in the loop until all involved databases are launched and connections to them can be established.
4. The my_java_client image will execute the stand-alone Java client application. This application will connect to host X-Windows server, therefore before running docker-compose up on Unix machine xhost + command should be executed. If the docker-compose is run in some other environments, then the stand-alone client can be executed independently from the command-line using the DohMPI_EDEN_Client.jar file provided in the repository. Additional instructions and details on testing the web service using both web and stand-alone application client are provided in Special Instructions – SPYNdoctors.pdf file.

8. Information for Developers

We provide the source code for the service in the form of Java web application IntelliJ IDEA[3] project called DohMPI_EDEN. Project contains source code and all required libraries. It uses maven for resolving some of the dependencies. As the project was developed in IntelliJ IDEA and compliant with its format, the simplest way to set up development environment is to open provided project folder with IntelliJ IDEA. As this is a web application, producing the .war file upon its build, the Ultimate IntelliJ IDEA version will be required. We used versions 2016.3.4 on Windows and 2017.2.4 in Ubuntu Linux environment.

We also provide the source code for the client application. It is also provided as IntelliJ IDEA project folder called DohMPI_EDEN_Client. As this is not a web application, Community Edition of IntelliJ IDEA will be adequate for its further development.

9. References

1. SWADL RESTful API documentation generator, https://github.com/ehearty/Swadl
2. XML EditorKit library, http://java-sl.com/xml_editor_kit.html
3. IntelliJ IDEA IDE for JVM, https://www.jetbrains.com/idea/