FHIR Interface to Directory of Clinicians

Project Manual

Team DNA information

Name	GT account	Email
Aaron Higdon	ahigdon3	ahigdon3@gatech.edu
Dan Abel	eabel3	dan.abel@ymail.com
Dan Frakes	dfrakes3	dpfrakes@gmail.com
David Vinegar	dvinegar3	dbvinegar@email.wm.edu
Nate Smith	nsmith82	nsmith82@gatech.edu

Source Code Repository:
http://github.gatech.edu/gt-hit-fall2016/fhir-interface-to-directory-of-clinicians

Table of contents

Application Overview

The purpose of this project is to map the records stored in the Utah Department of Health's (UDOH) Directory of Clinicians database to the Fast Health Interoperability Resource (FHIR) specifications. This will allow developers to access this practitioner licensure data in a standardized format in accordance with the FHIR specifications, currently published in its second version of a Draft Standard for Trial Use (DSTU 2).

Team DNA developed access to the Utah Department of Health's Directory of Clinicians by implementing a FHIR RESTful web service. The web service exposes DoC contents as a FHIR Practitioner resource searchable by medical license number, clinician last name and/or clinician first name.

The underlying architecture is based on Java Enterprise Edition technologies. The application data access is accomplished through JDBC and relies on an external datasource be available to the DoC database. The application uses a JNDI lookup of this datasource by a JNDI name of 'jdbc/DoC'. Once a data connection, via the configured datasource, is available to the application SELECT queries can then be run to retrieve data. The application is built assuming read-only access to the DoC database. Data retrieved from the DoC database is then transformed utilizing Java Architecture for XML Binding (JAXB). JAXB dynamically generates Java code from the HL7 organization's DSTU2 schemas for the FHIR specification. The produced Java code is capable of transforming the information read in the application's data access layer into JSON and XML conforming to the FHIR specification.

FHIR DoC application leverages Jersey to establish the RESTful interface. The interface provides the consumer the ability to search a clinician's information based on column names from the DoC table. Results are returned as a FHIR Practitioner resource.

DoC mapping

The Directory of Clinician information is maintained in the Microsoft SQL Server RDBMS. One table contains the relevant clinician and licence information required for the application.

DOC_PD_MASTER	
ID	INTEGER
DOC_PD_ID	VARCHAR(50)
LICENSE_NO	VARCHAR(50)
PROFESSION_NAME	VARCHAR(50)
LICENSE_NAME	VARCHAR(50)
FIRST_NAME	VARCHAR(50)
LAST_NAME	VARCHAR(50)
SORT_NAME	VARCHAR(150)
FULL_NAME	VARCHAR(150)
AKA	VARCHAR(100)
NAME_SUFFIX	VARCHAR(50)
DATE_OF_BIRTH	VARCHAR(50)
SSN	VARCHAR(50)
ISSUE_DATE	VARCHAR(50)
EXPIRATION_DATE	VARCHAR(50)
LIC_STATUS	VARCHAR(50)
DATE_THIS_STATUS	VARCHAR(50)
ADDR_LINE_1	VARCHAR(100)
ADDR_LINE_2	VARCHAR(100)
ADDR_CITY	VARCHAR(50)
ADDR_STATE	VARCHAR(50)
ADDR_ZIPCODE	VARCHAR(50)
ADDR_COUNTY	VARCHAR(50)
ADDR_PHONE	VARCHAR(50)
ADDR_EMAIL	VARCHAR(50)
DATE_DECEASED	VARCHAR(50)
LICENSE_NAME2	VARCHAR(50)
DISCIPLINE	VARCHAR(50)
FEDERAL_ID	VARCHAR(50)

Figure 1
Director of Clinician Database
Table (DOC_PD_MASTER)

The objective of the FHIR DoC is to map the fields from the DoC database into the Practitioner FHIR resource.

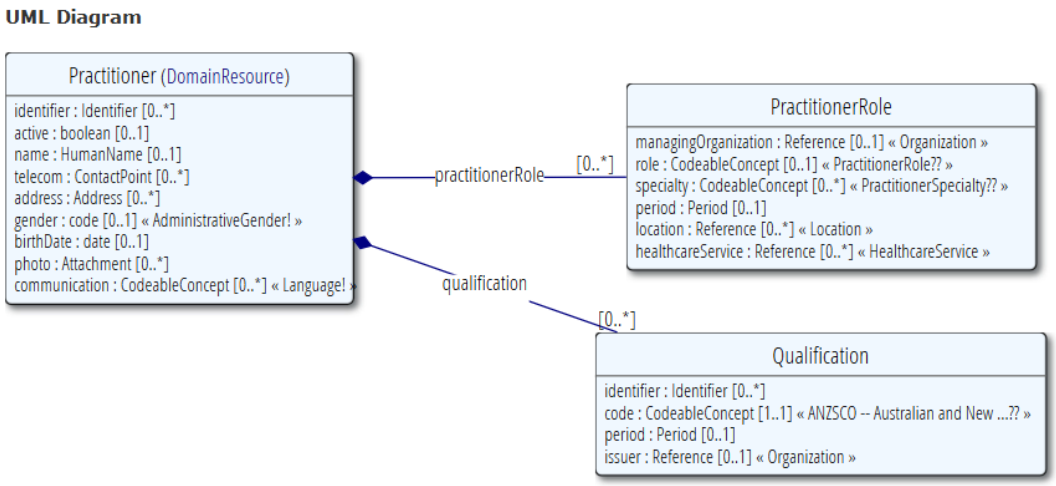


Figure 2
UML Practitioner Resource (DSTU2)

Build Procedures

Prerequisites

The following tools must be installed on your build machine:

- Maven 2
- Java 8

Download the source code ZIP file and extract the resources on to your machine. Source code is also available from the GeorgiaTech GitHub repository (http://github.gatech.edu/gt-hit-fall2016/fhir-interface-to-directory-of-clinicians). The resulting directory structure will resemble the following (see *Appendix I* for full details)

```
root
  |
  |--- dao <directory>
  |
  |--- database <directory>
  |
  |--- fhir <directory>
  |
  |--- service <directory>
  |
```

|--- pom.xml

On the command line, navigate to the project root directory and execute the following commands:

Navigate to project root directory

```
mvn clean install
```

A successful build should end with something resembling the following:

```
[INFO] Reactor Summary:
[INFO]
[INFO] Parent module ..................................... SUCCESS [  0.216 s]
[INFO] FHIR Resource module .............................. SUCCESS [  8.938 s]
[INFO] Embedded DB module ................................ SUCCESS [  1.467 s]
[INFO] Data Access module ................................ SUCCESS [  2.213 s]
[INFO] Webservice module ................................. SUCCESS [  1.067 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.012 s
[INFO] Finished at: 2016-12-03T22:50:34-05:00
[INFO] Final Memory: 38M/646M
[INFO] ------------------------------------------------------------------------
```

Deployment

The FHIR DoC Access application was written with both embedded database and external database capabilities:

Prerequisites

The following tools must be installed on your server:

- Java 8

Local Server with Embedded database (Development)

This following documentation covers the use of Jetty, an open-source Java server container used in development.

The application utilizes a system property to allow use of an embedded database.

database.useembedded=true

This can be set by two different methods:

1. Including *database.useembedded* as a System property to the JVM
2. Including a *database.properties* file on the classpath that contains the *database.useembedded* entry.

   ** If both methods are used to configure the database access type, the *database.properties* takes precedence.

For local testing:

Navigate to service webapp module, from the `root` directory (see Appendix I)

```
cd service
```

Start server

```
mvn jetty:run
```

After execution, a local Jetty server will be running locally with the application fully deployed.

Open web browser, (see Usage for available operations)

```
http://localhost:8090/
```

To change the port Jetty binds to, modify the Jetty plugin configuration inside *service/pom.xml* file and restart the Jetty server.

```
<connectors>
        <connector implementation="org.mortbay.jetty.nio.SelectChannelConnector">
                <port>8090</port>
        </connector>
</connectors>
```

When using the Jetty plugin, `database.useembedded` property is set up inside the *service/pom.xml* inside the plugin configuration:

```
<systemProperties>
      <systemProperty>
            <name>database.useembedded</name>
            <value>true</value>
      </systemProperty>
</systemProperties>
```

The Maven build is configured to utilize the Derby embedded database when running unit tests by reading the `database.useembedded` property from the configuration *database/src/test/resources/database.properties*

JEE Deployment with External Database Access

For Enterprise deployment see the following:

The Maven build produces a JEE WAR file that can be deployed to a Java Enterprise Application Server such as JBoss EAS, Weblogic Server, or others. Configuration of specific application servers is not detailed here, however the following configuration steps must be satisfied in order to deploy the application.

1. Configure a Datasource in the application server with the JDBC connection information to the DoC database. The Datasource must be discoverable with the JNDI name of 'jdbc/DoC'.
   As an example JBoss configuration see
   https://docs.jboss.org/author/display/AS71/DataSource+configuration
   Information for configuring the JDBC Connection can be found at:
   https://msdn.microsoft.com/en-us/library/ms378428(v=sql.110).aspx
2. Create a web application context for the application to use as the base of the RESTful services. An example JBoss configuration is included in
   `service/src/main/webapp/WEB-INF/jboss-web.xml`
3. Deploy the WAR from `service/target/service-1.00.000.war` to the application server.

Usage

For convenience, a sample reference web application is included to assist in testing. Once deployed, this page can be accessed at the following location:

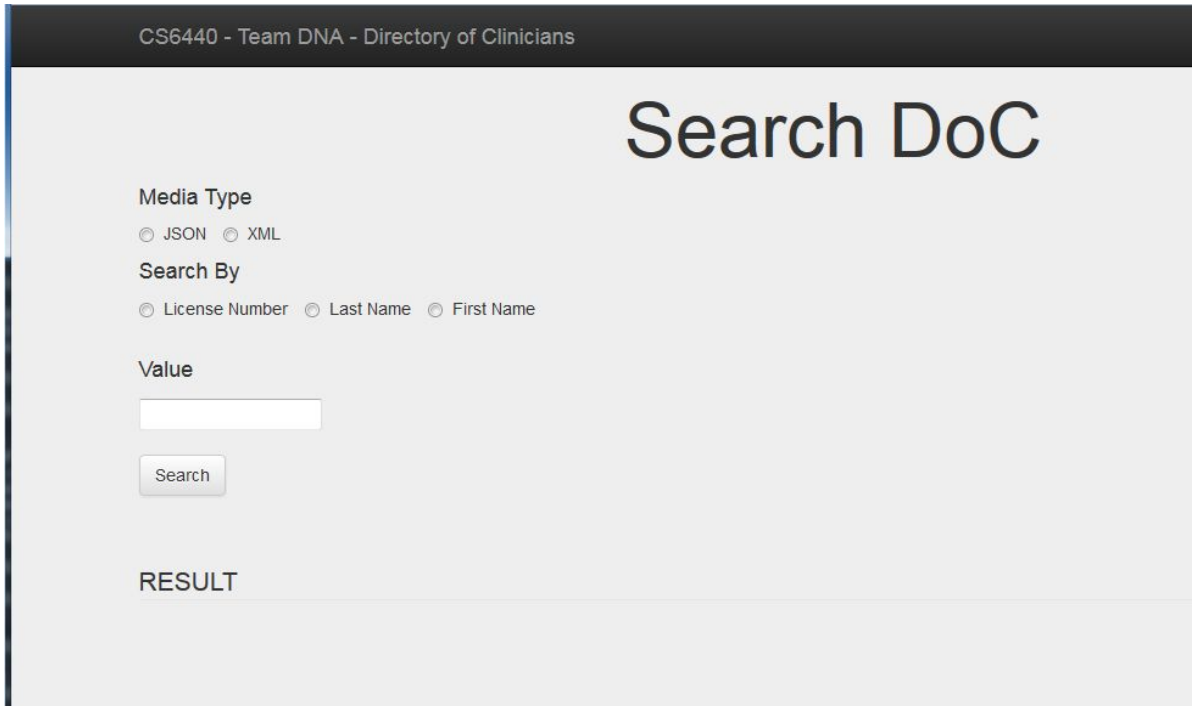
http://localhost:8090/index.html

On this page, links are provided to four sample queries.

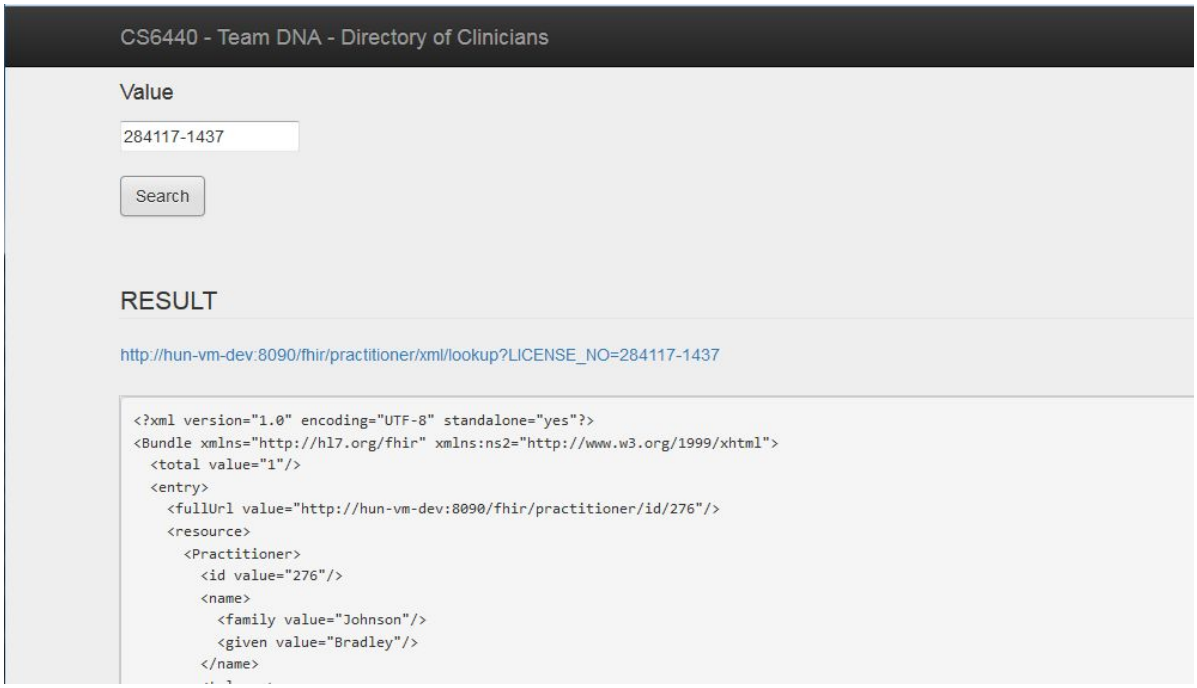
Also, a search page is provided at the following location:

http://localhost:8090/search.html

Here, the user can choose the media type and the parameter to use for the search.



When a search is performed, the results will be displayed below the search area.



Web Service Primary Operations

In a production environment, it is anticipated that all searches would be done via lookup requests in the following format:

http[s]://<hostname>[:<port>][/webcontext]/fhir/practitioner/(json|xml)/lookup ?<search parameter>=<search value>[&<search parameter>=<search value>]
(*items in [] are optional elements dependent on deployment)*

Practitioner Search

This operation returns a Practitioner FHIR resource formatted in either XML or JSON.
URL format: (*items in [] are optional elements dependent on deployment)*
Searchable terms are all of the DoC PD Master table's column names (See Figure 1).
http[s]://<hostname>[:<port>][/webcontext]/fhir/practitioner/(json|xml)/lookup?<search parameter>=<search value>[&<search parameter>=<search value>]
HTTP Query parameters can be compounded to better target result. By passing multiple search parameters the interface will behave as follows:
If different search parameters are passed then the search is an 'AND' search.

<u>Example 1:</u>
?LAST_NAME=Johnson&FIRST_NAME=Bradley
will return clinicians with:

Last Name of 'Johnson' **AND** First Name of 'Bradley'

If the same search parameters are passed then the search is an 'OR' search

Example 2:

?LAST_NAME=Johnson&LASST_NAME=Clark

will return clinicians with:

Last Name of 'Johnson' **OR** Last Name of 'Clark'

Combining both may yield unexpected results, however simple scenarios will be AND at a high level with OR at the lower:

Example 3:

?LAST_NAME=Johnson&LAST_NAME=Clark&FIRST_NAME=Bradley

will return clinicians with:

First Name of 'Bradley' **AND** (Last Name of 'Johnson' **OR** Last Name of 'Clark')

Example Usage:

Request type	Response type	Endpoint Path	Parameter
GET	JSON	/fhir/practitioner/json/lookup?LICENSE_NO={LICENSE_NO}	Practitioner license number
GET	XML	/fhir/practitioner/xml/lookup?LICENSE_NO={LICENSE_NO}	Practitioner license number
GET	JSON	/fhir/practitioner/json/lookup?LAST_NAME={LAST_NAME}	Practitioner's last name
GET	XML	/fhir/practitioner/xml/lookup?LAST_NAME={LAST_NAME}	Practitioner's last name
GET	XML	/fhir/practitioner/xml/lookup?LAST_NAME={LAST_NAME}&FIRST_NAME={FIRST_NAME}	Practitioner's last name AND practitioner's first name.
GET	XML	/fhir/practitioner/xml/lookup?LAST_NAME={LAST_NAME1}&LAST_NAME={LAST_NAME2}	Last name OR another last name

Web Service Auxiliary Operations

Additional Operations are exposed in the Web Service interface to allow for developer testing.

DoC License Number Lookup (Development Only)

This operation returns a raw view of the DoC table representation in JSON format.
URL format:
http[s]://<hostname>[:<port>][/webcontext]/fhir/licensure/byLicenseNo?licenseNo=<license number>

Request type	Response type	Endpoint Path	Parameter
GET	JSON	/fhir/licensure/byLicenseNo?licenseNo={LICENSE_NO}	Practitioner license number

Add Practitioner to Database(Development Only)

This operation allows the addition of test data into the embedded database.

Request type	Response type	Endpoint	Parameter
POST	N/A	/fhir/licensure/add/	DoCPDao object

Bulk Add Practitioners to Database from CSV file (Development Only)

This operation allows the addition of test data into the embedded database from an uploaded CSV file:.

Request type	Response type	Endpoint	Parameter

POST	N/A	/fhir/bulk/addCsvFile	multipart/form-data CSV file

Uploading Test Data

See Special Instructions

Setting up a development environment

Source code is maintained in the Georgia Tech GtHub repository (https://github.gatech.edu/gt-hit-fall2016/FHIR-Interface-to-Directory-of-Clinicians.git). The project is setup to build with Maven (https://maven.apache.org/). Project source code is organized into 4 sub projects:

database - Responsible for establishing connection to RDBMS. if the JVM property *database.useembedded* is set to *true*, Apache Derby (https://db.apache.org/derby/) is used as the RDBS.

dao - Responsible for interaction with the RDBMS via SQL queries. Returns information from the Directory of Clinician database table (DOC_PD_Master) as Java POJOs.

fhir - Responsible for compiling the FHIR schemas (https://www.hl7.org/fhir/fhir-codegen-xsd.zip) into Java POJOs via JAXB (https://jaxb.java.net). The generated code is capable of transformation of Java objects to FHIR XML/JSON.

service - Web Application that provides the RESTful interfaces. The application is built into a WAR (Web application ARchive) capable of being deployed to a JEE container.

Switch to feature branch

```
cd FHIR-Interface-to-Directory-of-Clinicians
git checkout -b "master"
```

Create IDE Project (NetBeans supports Maven projects natively)

Eclipse:

```
mvn eclipse:eclipse
```

IntelliJ:

```
mvn idea:idea
```

Then, open Eclipse workspace in the "FHIR-Interface-to-Directory-of-Clinicians" directory and *`File / Import… -General > Existing Projects into Workspace`*.

This discussion may also be helpful:
http://stackoverflow.com/questions/2061094/importing-maven-project-into-eclipse

Build Project

```
mvn clean install
```

For local testing the application can be launched using a Jetty (http://www.eclipse.org/jetty/) web server and the Maven Jetty plugin (configured in *service/pom.xml*):

Run Jetty

```
cd service
mvn jetty:run
```

The project should now be deployed to the localhost.

Technologies

- Java 8
- JDBC
- Jersey
- JAXB
- JavaDB Embedded Database
- FHIR Schema DTSU2
- Spring

Appendix I - Source Directory Structure

The following is a diagram of the source code directory structure:

```
(root)FHIR-Interface-to-Directory-of-Clinicians/
├── dao
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   └── edu
│       │   │           └── gatech
│       │   │           └── cs6440
│       │   │                   └── teamdna
│       │   │                   ├── dao
│       │   │                   │   ├── DoCPDDaoConstants.java
│       │   │                   │   ├── DoCPDDaoImpl.java
│       │   │                   │   └── DoCPDDao.java
│       │   │                   └── model
│       │   │                           └── DoCPDDo.java
│       │   └── resources
│       │       └── spring-beans.xml
│       └── test
│               ├── java
│               │   └── edu
│               │       └── gatech
│               │               └── cs6440
│               │               └── teamdna
│               │                       └── dao
│               │                       └── DoCPDDaoImplTest.java
│               └── resources
│               └── database.properties
├── database
│   ├── Doc_PD_Master.sql
│   ├── pom.xml
│   └── src
│       ├── main
│       │   └── java
│       │       └── edu
│       │               └── gatech
│       │               └── cs6440
│       │                       └── teamdna
│       │                       ├── Database.java
│       │                       └── impl
│       │                               ├── BaseDAO.java
│       │                               ├── ConfigUtils.java
│       │                               ├── CreateSQL.java
│       │                               ├── DatabaseFactory.java
│       │                               ├── EmbeddedDatabase.java
│       │                               └── JNDIDatabase.java
│       └── test
│               ├── java
│               │   └── edu
│               │       └── gatech
```

```
│              │              └── cs6440
│              │              └── teamdna
│              │                      └── impl
│              │                      ├── ConfigUtilsTest.java
│              │                      └── DatabaseFactoryTest.java
│              └── resources
│              └── database.properties
├── fhir
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   └── edu
│       │   │         └── gatech
│       │   │         └── cs6440
│       │   │                 └── teamdna
│       │   │                 └── fhir
│       │   │                         └── builder
│       │   │                         ├── AddressBuilder.java
│       │   │                         ├── BackboneElementBuilder.java
│       │   │                         ├── BundleBuilder.java
│       │   │                         ├── BundleEntryBuilder.java
│       │   │                         ├── CodeableConceptBuilder.java
│       │   │                         ├── CodingBuilder.java
│       │   │                         ├── ContactPointBuilder.java
│       │   │                         ├── DomainResourceBuilder.java
│       │   │                         ├── ElementBuilder.java
│       │   │                         ├── FhirUtils.java
│       │   │                         ├── HumanNameBuilder.java
│       │   │                         ├── IdentifierBuilder.java
│       │   │                         ├── PractitionerBuilder.java
│       │   │                         ├── PractitionerQualificationBuilder.java
│       │   │                         ├── PractitionerRoleBuilder.java
│       │   │                         ├── ResourceBuilder.java
│       │   │                         └── UriBuilder.java
│       │   ├── resource
│       │   │   └── edu
│       │   │         └── gatech
│       │   │         └── cs6440
│       │   │                 └── teamdna
│       │   │                 └── fhir
│       │   │                         └── jaxb.properties.ignore
│       │   ├── xjb
│       │   │   └── naming.xjb
│       │   └── xsd
│       │       ├── fhir-single.xsd
│       │       ├── fhir-xhtml.xsd
│       │       └── xml.xsd
│       └── test
│               └── java
│               └── edu
│                     └── gatech
│                     └── cs6440
│                             └── teamdna
│                             └── fhir
│                                     ├── builder
│                                     │   └── FhirUtilsTest.java
```

```
│                                   ├── JaxbTest.java
│                                   └── PractitionerTest.java
├── pom.xml
├── README.md
├── service
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   └── edu
│       │   │           └── gatech
│       │   │           └── cs6440
│       │   │                   └── teamdna
│       │   │                   ├── api
│       │   │                   │   ├── DoCBulkUploadService.java
│       │   │                   │   ├── DoCService.java
│       │   │                   │   └── PractitionerService.java
│       │   │                   └── business
│       │   │                           ├── BundleDirector.java
│       │   │                           ├── ConversionUtils.java
│       │   │                           ├── DoCManager.java
│       │   │                           ├── DoCPD.java
│       │   │                           └── PractitionerDirector.java
│       │   ├── resources
│       │   │   └── spring-beans.xml
│       │   └── webapp
│       │       ├── bulkUpload.html
│       │       ├── index.html
│       │       ├── search.html
│       │       └── WEB-INF
│       │               ├── jboss-web.xml
│       │               ├── jetty-env.xml
│       │               └── web.xml
│       └── test
│               └── java
│               └── edu
│                       └── gatech
│                       └── cs6440
│                               └── teamdna
│                               └── business
│                                       └── ConversionUtilsTest.java
├── bulkUpload_SampleData.csv
├── bulkUpload_Template.csv
├── CS 6440 Student Project Proposal - FHIR Interface to Directory of Clinicians.docx
├── pom.xml
└── README.md
```

Appendix II - Developer Notes

Debug Mode

When developing is is often helpful to setup debugging from the IDE and the running Jetty server. This is caable using Java's Debug Wire Protocol (JDWP) - https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/introclientissues005.html

In order to start the Jetty server with JDWP enabled, create.modify an OS environment variable *MAVEN_OPTS:*

For Windows:

```
set MAVEN_OPTS=-Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000
```

For Linux/Mac:

```
Export MAVEN_OPTS=-Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000
```

Restart the Jetty server (see Deployment). After restart, JDWP should be available on port 8000. See your IDE specifics for attaching to the Jetty JVM:

Eclipse:
https://www.eclipse.org/jetty/documentation/9.3.x/debugging-with-eclipse.html
IntelliJ:
https://www.eclipse.org/jetty/documentation/9.3.x/debugging-with-intellij.html

Refreshing Embedded Database

The Derby database is entirely file based. To refresh the database you can simply delete the Derby directory containing the DB files; for this application, this directory is named *doc-db.*
To delete the data associated with the Jetty server:

```
rm -r service/doc-db
```

For the database unit tests:

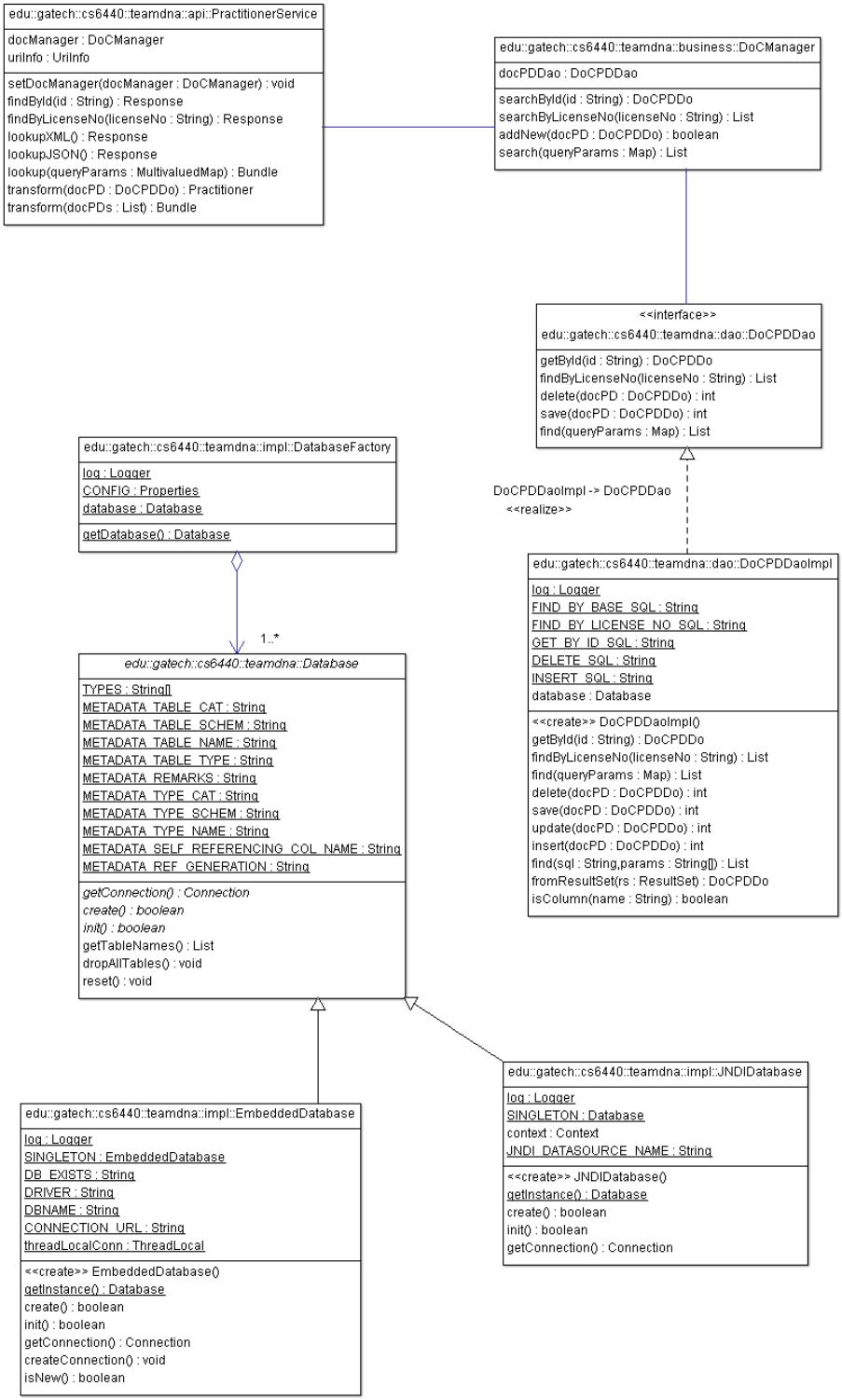
```
rm -r database/doc-db
```

For the dao unit tests:

```
rm -r dao/doc-db
```

Once the directory is removed it will be re-created as an empty database the next time it is accessed.

Appendix III - Design Class Diagram



Appendix IV - Quickstart

Run Locally

1. Download and install Java 8 JDK - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. Download and Install Maven 3 - https://maven.apache.org/download.cgi
3. Download or extract project source code

   `unzip "sourcecode - FHIR-Interface-to-Directory-of-Clinicians.zip"`
4. Open command line prompt and navigate to project root directory (see Appendix I)

   `cd FHIR-Interface-to-Directory-of-Clinicians`
5. Build the project with the following:

   `mvn clean install`
6. Navigate to the *service* webapp sub-project:

   `cd service`
7. Launch the web service

   `mvn jetty:run`
8. Open web browser to the reference web application

   http://localhost:8090/
9. Load Sample Data:
   a. Select 'Bulk Upload' or navigate

      http://localhost:8090/bulkUpload.html

   b. Click *'Browse'* and choose the *bulkUpload_SampleData.csv* CSV file

   c. Click *'Upload It'*

Deployed Solution

A sample web application has been deployed and ready with sample data at:
https://udoh-doc.herokuapp.com/

Example searches:

- https://udoh-doc.herokuapp.com/fhir/practitioner/json/lookup?LICENSE_NO=419857-3239

- https://udoh-doc.herokuapp.com/fhir/practitioner/xml/lookup?LICENSE_NO=419857-3239
- https://udoh-doc.herokuapp.com/fhir/practitioner/json/lookup?LAST_NAME=Clark
- https://udoh-doc.herokuapp.com/fhir/practitioner/xml/lookup?LAST_NAME=Clark
- https://udoh-doc.herokuapp.com/fhir/practitioner/xml/lookup?LAST_NAME=Clark&FIRST_NAME=Robert
- https://udoh-doc.herokuapp.com/fhir/practitioner/xml/lookup?LAST_NAME=Clark&LAST_NAME=Allan
- https://udoh-doc.herokuapp.com/fhir/practitioner/xml/lookup?LAST_NAME=Katz&LAST_NAME=Johnson&FIRST_NAME=Bradley