HL7 FHIR HL7 International

FHIR Intermediate Course

MODULE 3: FHIR SERVERS

Assignments

Course Overview

Module I: Implementation Guides

Most relevant FHIR Implementation Guides: Argonaut & IPS
Argonaut Development and Roadmap
Argonaut Data Query IG: Scope, Use Cases
Argonaut Provider Directory IG: Scope, Use Cases
IPS FHIR IG: Scope, Use Cases

Module II: FHIR Clients

General Guidelines for FHIR Clients
FHIR Clients in JavaScript / C# / Java [1 - Elective]

Module III: FHIR Facades

Why use FHIR Server Facade: your system on FHIR
Specific FHIR Servers (FHIR Facade)
Facade Use Case / Scenarios
Facade Architecture / Patterns
Where to put the FHIR Facade
System Integration / Integration Engine / Bus / Messaging
Facade in Java / Node.JS / C# [1 - Elective]

Module IV: FHIR Applications

Smart-On-FHIR
CDS-Hooks
Integration with Smart-On-FHIR / CDS-Hooks

Assignments For Module 3

Goals and Options

In this week we will have formal “programming” assignments. The assignments in this unit are mostly “Build your Own” . If you want to review code, just review the code described below, in the “Where to Look” section.

We will ask you to change or enhance some apps built using the Java or JS FHIR server facades.

Platform

Platform	Description
P01	Java
P02	node.js
P03	dotnet (C#)

Assignment Levels

L01	Add search parameter	40 points
L02	Add support for a new resource -	40 points
L03	Fix the implementation of an exist-ing resource	20 points

Total grade for each track and platform is 100 points
You need to obtain at least 60 points in order to complete this unit.
You can try more than one platform and/or track if you are able to, we will grade you with the maximum grade you obtained (up to 100).

General Introduction

Disclaimer: The code included for these assignments:
- It is **not supposed** to be production-ready.
- **does not support** all the requirements of the FHIR standards (specially around search, paging, etc.)
- **It is not ready** for storing or retrieving information about actual patients.

It is only used to illustrate FHIR concepts and techniques and make you think on possible strategies and implementation of the methods described through this unit.

For this assignment we are providing you with similar projects that are already running, developed using the 3 technologies we use in this course. All of them are **server applications exposing an API (they don´t have a front end)**

- **Java (Spring Boot project)**
- **Node.JS**
- **C#**

These projects are in the FHIR Server course block, in the **Projects** folder

You can edit them using your favorite editor/IDE (we used **Visual Studio Code 3** for all the platforms)

The applications share the same data model and legacy API

The data model is implemented in a Postgresql database in an Amazon instance.
The local database stores persons basic demographic data and identifiers, medications, and medication requests for some persons, prescribed by other persons.

The projects implement a read-only façade (read, search) for the legacy database, going through an automatically generated API (using an open-source tool called postgrest)
If you are interested in Postgrest, this is the link: https://postgrest.org/en/stable/

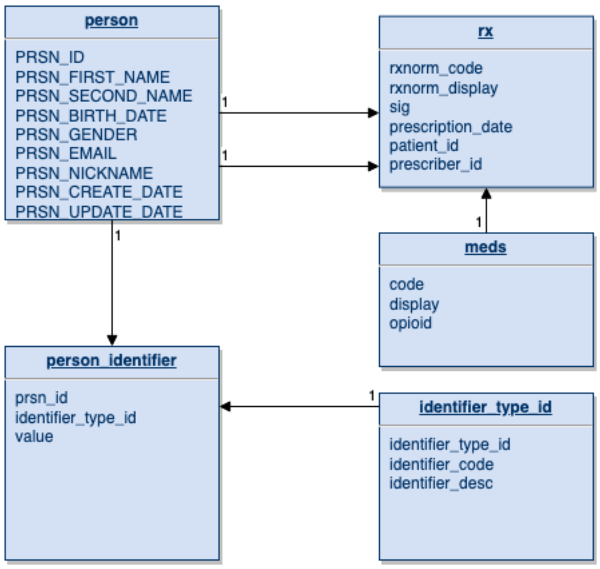
We tried to use the bare minimum dependencies to make a small functional server, like a minimum viable product embryo, just to facilitate the deployment and test of your solutions for the assignment.

Legacy Database Structure

The legacy database has the same entities for all the platforms.

The database is pre-loaded with data so you can search by parameters and id without adding any record.
Disclaimer: The data inside of the database is completely fictional, if there is any name of actual persons, is just a coincidence.

Database Diagram



Since the API is automatically generated, you will have access exactly to the same entities with the same name, through a REST url
Important: This is NOT a FHIR endpoint, is an automatically generated legacy endpoint.

Legacy API Access

You can access the API by adding the entity name to the 'database' endpoint.
So, if our Postgres database is in the IP address 10.3.4.21, and you want to access 'person' then the address for accesing 'person' is http://10.3.4.21/person
Our postgrest server is in http://3.221.164.25, port 9080 (we use port 9080 because there is an API Gateway installed in the same AWS instance called Apache APISix, making sure the IP is not attacked using DDOS)
So… the endpoint for person is http://3.221.164.25:9080/person
Try it.

Database/API Dictionary

Tables / Entities

person: Persons records, one for each person (patient or practitioner, is the same)

COLUMN	DESCRIPTION
PRSN_ID	Person unique identifier for the server, autonumeric
PRSN_FIRST_NAME	First official name
PRSN_SECOND_NAME	Second official name
PRSN_LAST_NAME	Family Name
PRSN_BIRTH_DATE	Date of Birth
PRSN_GENDER	Person Gender “male”,”female”
PRSN_EMAIL	Email address for the person
PRSN_NICK_NAME	Nick name for the person
PRSN_CREATE_DATE	time stamp for creation
PRSN_UPDATE_DATE	time stamp for update

identifier_type: Identifier types: passport, SSN, etc. Special type: NPI – only for practitioners

COLUMN	DESCRIPTION
identifier_type_id	Id for the Identifier Type
identifier_type_code	Code for the Identifier Type
identifier_type_desc	Description for the Identifier Type

person_identifier: Identifiers for each person

COLUMN	DESCRIPTION
prsn_id	Unique id for the person (from person)
identifier_type_id	Unique id for the identifier type (from identifier_type)
value	Value for the identifier for this person

meds: medications coded in rxnorm

COLUMN	DESCRIPTION
code	Code for the medication (rxnorm)
display	Description of the medication (rxnorm)
opioid	‘yes’ if the medication is an opioid ‘no’ if it is not

rx: active prescriptions by patient

COLUMN	DESCRIPTION
rxnorm_code	Code for the medication (rxnorm), from meds
rxnorm_display	Description of the medication (rxnorm)
sig	Instructions for the patient
patient_id	Internal id for the patient (from person)
prescriber_id	Internal id for the prescriber (from person)
prescription_date	Date for the prescription

Some comments on the tables

Each **person** has 1 or more records in **person_identifier,** one for each **identifier_type**

Each **person** has 0 or more records in **rx** (meaning that one prescription order was created **for** the person), linked by the **patient_id** field

Each **person** has 0 or more records in **rx** (meaning that one prescription order was created **by** the person), linked by the **prescriber_id** field

Each **meds** has 0 or more records in **rx (**meaning that there is an active prescription order for the medication)

We are not allowed to change the database structure, so if anything is missing from this picture, we need to inject it through the adapters and mappers.

Example: The **rx** table/entity has no primary id, so we created one in for the FHIR MedicationRequest, by combining some of the attributes fully identifying the row (patient, date, med, prescriber).

Mapping from Legacy to Patient

This is the result of mapping the legacy database content for a patient into a FHIR Patient resource.

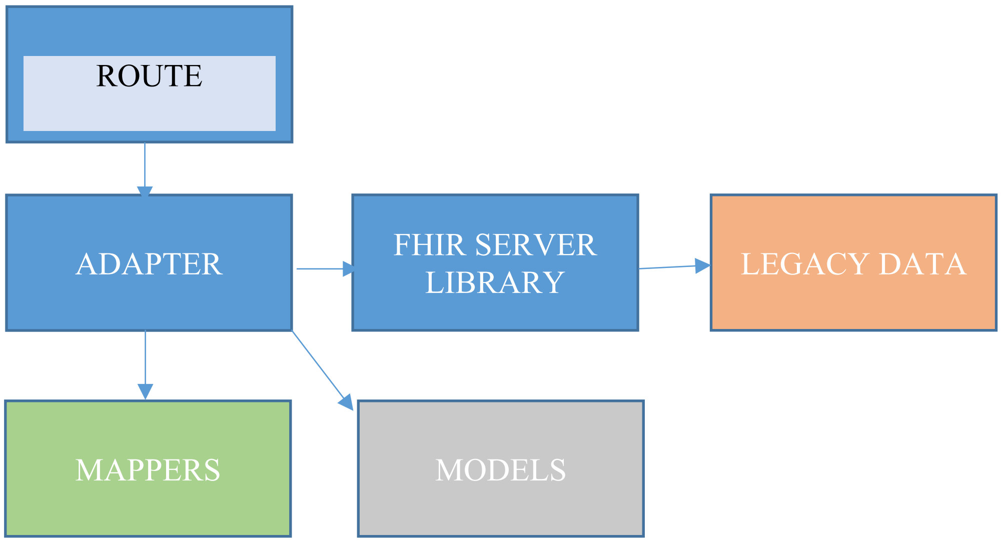
```
{
    "resourceType": "Patient",
    "id": "{{person.PRSN_ID}}",
    "text": {
        "status": "generated",
        "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">{{per-
son.PRSN_ID}}</div>"
    },
    "identifier":

         {
            "use": "official",
            "system": "https://fhir-intermediate.org/{{identifier_type_code}}",
            "value": "{{value}}",
        }
    ],
    "name": [
        {
            "use": "official",
            "text": "{{person.PRSN_FIRST_NAME}} {{person.PRSN_SECOND_NAME}}
{{person.PRSN_LAST_NAME}} ",
            "family": "{{person.PRSN_LAST_NAME}}",
            "given": [ "{{person.PRSN_FIRST_NAME}}",
                        "{{person.PRSN_SECOND_NAME}}"
            ]
        },
        {
            "use": "nickname",
            "given": ["{{person.PRSN_NICK_NAME}}"]
        }
    ],
    "telecom": [
        {
            "system": "email",
            "value": "{{person.EMAIL}}"
        }
    ],
    "gender": "{{person.PRSN_GENDER}}",
    "birthDate": "{{person.PRSN_BIRTH_DATE}}"
}
```

Where to Look

Where is the FHIR? (a FHIR pun is mandatory in every FHIR related material, we are very sorry about this):

Remember that these are mainly API facades, so usually you will have a model for the legacy database, and some combinations of services implementing adapters and mappers from/to the FHIR format from/to the legacy format.



We strongly recommend out 'Tips' codelab for each platform to find out where to change or add code to fulfill the requirements of our assignments

We provide a Postman collection to test the limited functionality of the servers.
You can find the collection here
https://tinyurl.com/fictest
Remember to change the environment according to your platform.

Setting up your projects

1) Make sure you have the pre-requisites
2) Download the code for your platform
3) The code will include TWO projects (the server project, and the tester project)
4) Depending on the platform, you need to retrieve the required components for each project

   NODE.JS: **npm install** before **npm start**

   Java: The components will be loaded when you execute **mvn run:jetty** or run the first test

   DotNet C#: The project will build, and components will be retrieved when you run it. You can also use **dotnet build**

Testing and Grading

We included **a separate project for each platform, called 'tester', to test the facades.** These projects allow you to:

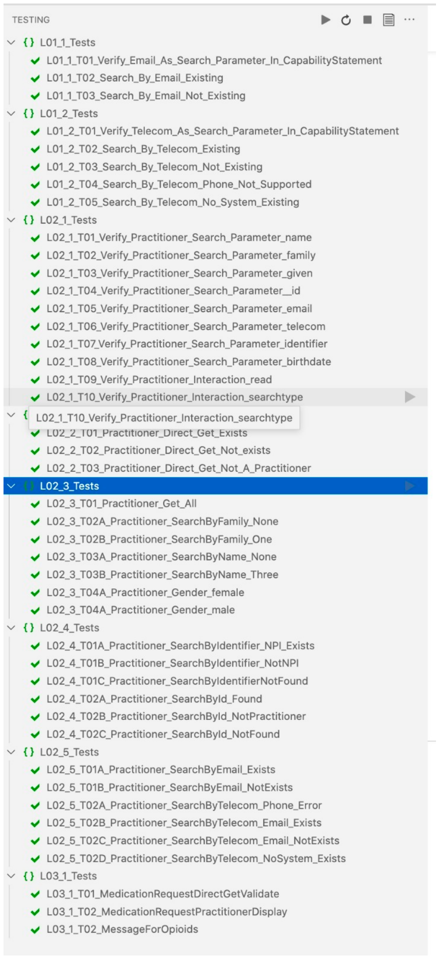
a) Test your server
b) Create your submission file to send to our server, so we can grade your work.

Remember our honor code (this is an individual effort, not a group effort), and that you will upload your submission AND your code.
Grading is based on your project's capability to fulfill the tests.
Each test has a specific partial score towards the total for the level (L1 to L3) , and you will find out the score when submitting your assignment in the course site.

Assignment Levels

L01	Add search parameter	40 points
L02	Add support for a new resource -	40 points
L03	Fix the implementation of an existing resource	20 points

This is an example of a 100/100 project in dotnet.



```
TESTING
{ } L01_1_Tests
    ✓ L01_1_T01_Verify_Email_As_Search_Parameter_In_CapabilityStatement
    ✓ L01_1_T02_Search_By_Email_Existing
    ✓ L01_1_T03_Search_By_Email_Not_Existing
{ } L01_2_Tests
    ✓ L01_2_T01_Verify_Telecom_As_Search_Parameter_In_CapabilityStatement
    ✓ L01_2_T02_Search_By_Telecom_Existing
    ✓ L01_2_T03_Search_By_Telecom_Not_Existing
    ✓ L01_2_T04_Search_By_Telecom_Phone_Not_Supported
    ✓ L01_2_T05_Search_By_Telecom_No_System_Existing
{ } L02_1_Tests
    ✓ L02_1_T01_Verify_Practitioner_Search_Parameter_name
    ✓ L02_1_T02_Verify_Practitioner_Search_Parameter_family
    ✓ L02_1_T03_Verify_Practitioner_Search_Parameter_given
    ✓ L02_1_T04_Verify_Practitioner_Search_Parameter__id
    ✓ L02_1_T05_Verify_Practitioner_Search_Parameter_email
    ✓ L02_1_T06_Verify_Practitioner_Search_Parameter_telecom
    ✓ L02_1_T07_Verify_Practitioner_Search_Parameter_identifier
    ✓ L02_1_T08_Verify_Practitioner_Search_Parameter_birthdate
    ✓ L02_1_T09_Verify_Practitioner_Interaction_read
    ✓ L02_1_T10_Verify_Practitioner_Interaction_searchtype
{   L02_1_T10_Verify_Practitioner_Interaction_searchtype
    ✓ L02_2_T01_Practitioner_Direct_Get_Exists
    ✓ L02_2_T02_Practitioner_Direct_Get_Not_exists
    ✓ L02_2_T03_Practitioner_Direct_Get_Not_A_Practitioner
{ } L02_3_Tests
    ✓ L02_3_T01_Practitioner_Get_All
    ✓ L02_3_T02A_Practitioner_SearchByFamily_None
    ✓ L02_3_T02B_Practitioner_SearchByFamily_One
    ✓ L02_3_T03A_Practitioner_SearchByName_None
    ✓ L02_3_T03B_Practitioner_SearchByName_Three
    ✓ L02_3_T04A_Practitioner_Gender_female
    ✓ L02_3_T04A_Practitioner_Gender_male
{ } L02_4_Tests
    ✓ L02_4_T01A_Practitioner_SearchByIdentifier_NPI_Exists
    ✓ L02_4_T01B_Practitioner_SearchByIdentifier_NotNPI
    ✓ L02_4_T01C_Practitioner_SearchByIdentifierNotFound
    ✓ L02_4_T02A_Practitioner_SearchById_Found
    ✓ L02_4_T02B_Practitioner_SearchById_NotPractitioner
    ✓ L02_4_T02C_Practitioner_SearchById_NotFound
{ } L02_5_Tests
    ✓ L02_5_T01A_Practitioner_SearchByEmail_Exists
    ✓ L02_5_T01B_Practitioner_SearchByEmail_NotExists
    ✓ L02_5_T02A_Practitioner_SearchByTelecom_Phone_Error
    ✓ L02_5_T02B_Practitioner_SearchByTelecom_Email_Exists
    ✓ L02_5_T02C_Practitioner_SearchByTelecom_Email_NotExists
    ✓ L02_5_T02D_Practitioner_SearchByTelecom_NoSystem_Exists
{ } L03_1_Tests
    ✓ L03_1_T01_MedicationRequestDirectGetValidate
    ✓ L03_1_T02_MedicationRequestPractitionerDisplay
    ✓ L03_1_T02_MessageForOpioids
```

1. Assignment L01: Add search parameters for Patient

Valid for all platforms: P01, P02, P03

	BUILD YOUR OWN

The current project does not support the searching by email address or telecom.
The original API allows a very limited parameterized search for persons.
There is a Search Parameter for email in the legacy API, but it's not in the FHIR facade
a. You need to support FHIR search by **email and telecom.**
b. When searching by telecom, only email is accepted as system. If you receive any other system, you need to raise an exception with this status and message:
HTTP 501 Not Implemented: The underlying server only handles email addresses for the patients, thus search by system=phone is not implemented
Your task: fix the app to include this search

2. Assignment L02: Add support for Practitioner

Valid for all platforms: P01, P02, P03

BUILD YOUR OWN

We’ve found out that the organization uses the same db structure for Practitioners.
Your job: create the same façade but for the Practitioner resource
We need you to add Practitioner support to the same project, without ruining the Patient support and without touching the legacy classes (because you cannot, they are in AWS)
Hints:
1. Practitioners are persons who have an NPI identifier (system=...url.../NPI). All the others are Patients, are (legacy) persons, but they are not Practitioners
2. birthdate is not a standard search parameter for Practitioner
3. If you search all the Practitioners by ANY search parameter, all of them will have an NPI identifier. Practitioners with no NPI identifier are not valid.
4. If you do a direct get to a Practitioner by ID, which is the same as the Patient / Person id, and the person is NOT a Practitioner, you need to return an exception:
HTTP 400 Bad Request: The person you requested is not a practitioner - Lacks a NPI identifier
5. If you do a search by identifier for a Practitioner, the only valid system=...url.../NPI
In this case you need to raise an error with a specific message
HTTP 400 Bad Request: Practitioners can only be found knowing the NPI identifier - You are specifying : PP

6. When implementing practitioner search by telecom, remember that our database has no phone element for the persons, so we apply the same rule that we applied for Patients, if the system is not email, raise an error with a message:
HTTP 501 Not Implemented: The underlying server only handles email addresses for the practitioners, thus search by system=phone is not implemented

3. Assignment L03: Fix MedicationRequest

Valid for all platforms: P01, P02, P03

	BUILD YOUR OWN

Our project already has support for MedicationRequest.
But it's not conformant with US CORE. In fact, it is not FHIR compliant.
So, you will have three goals in this assignment:

1. Make the MedicationRequest FHIR / US CORE compliant. There are some elements missing.
2. Add the element Requester.display with the full name of the referenced requesting Practitioner
3. Add a new dosageInstruction if the medication being ordered is an opioid (We provide a function called IsOpioid(rxNormCode) answering this question.