Exercise RESTful Webservices

using the example of Bahmni OpenMRS

In the last exercise, Bahmni was introduced as an OpenMRS distribution. In this exercise, we will look at the interfaces that enable communication between the Bahmni web frontend and the OpenMRS backend. In the practical example, data will be exchanged via the interfaces. We use an API client and a 3LGMZ model (MI-Lab, incl. Bahmni submodel).

An **Application Programming Interface** (API) is a set of rules that enable applications to communicate with each other. Applications can communicate with each other in a network via web APIs. The main task of a web API I is to exchange representations of data with a corresponding client application[1] . On the server side, Web APIs consist of one or more publicly accessible endpoints for a defined request-response protocol [2].

Representational State Transfer (REST) APIs have all the features of a client-server architecture and also use standardized interfaces [3].
The following properties are relevant for the exercise:

- Resources - Client requests refer to resources. These stand for an information unit, e.g. person, patient, encounter.
- Addressing - resources are uniquely addressable with a Uniform Resource Identifier (URI)
- Universal representation - A client can request a resource in XML, HTML or JSON format, for example.
- Stateless protocol - requests are handled independently of each other.
- HTTP methods - REST uses a standardized set of HTTP methods, GET, POST, DELETE, etc.

OpenMRS provides backend operations via its platform, including the APIs and the required storage/databases, to support the OpenMRS applications. Various distributions n use the OpenMRS platform as a backend (e.g. Bahmni, KenyaEMR, UgandaEMR, NigeriaMRS)[4] with a customized frontend. OpenMRS stores data objects (persons, patients, encounters, observations, ...) in its database at [5].
The REST Web API exposes data objects from OpenMRS as resources. These can be created, edited or deleted via HTTP methods [6,7]. Bahmni Web acts as the front end and communicates with the OpenMRS platform as the back end via the REST Web API.
For the exercise, we a 3LGMZ model with a detailed representation of Bahmni. Here you will also find information on the endpoints (see Interfaces in the model) and resources (see Services in the model).

1 https://education.launchcode.org/csharp-web-dev-curriculum/web-api-rest/reading/web-api/index.html
2 https://en.wikipedia.org/wiki/Web_API#Server_side
3 https://de.wikipedia.org/wiki/Representational_State_Transfer
4 https://openmrs.atlassian.net/wiki/spaces/RES/pages/26273961/Platform+Team
5 OpenMRS data model: https://docs.openmrs.org/datamodel/
6 Documentation for the OpenMRS REST API: https://rest.openmrs.org/#openmrs-rest-api
7 https://demo.mybahmni.org/openmrs/module/webservices/rest/apiDocs.htm

1. Preparation

1.1. Task - API client Bruno

Start the API client Bruno and familiarize yourself with the environment.
Download: https://github.com/usebruno/bruno/releases
Documentation: https://docs.usebruno.com/

Create a new collection "MI-Lab". Move the mouse pointer over the collection, click on the three-dot menu (Meatball Menu) and on Settings in the menu.

The authentication data must sent with every request.
Enter the authentication data of the OpenMRS Admin in the "Auth" tab and confirm with "Save": drop-down list
(Type): Basic Auth
User: superman
Password: Admin123

Enter the following Base URL in the "Presets" tab and confirm with "Save". Base
URL https://192.168.48.77/openmrs/ws/

Click on Preferences" in the top menu bar (top left) under "Collection". In the "General" tab, remove the checkmark next to "SSL/TLS Certificate Verification". Confirm with "Save".

1.2. Task - Session

Create a new request via the three-point menu of the "MI-Lab" collection→ "New Request". Give it the name "1-2 Session". In the Auth tab, select "Inherit" to accept the access data of the collection.
A resource is requested via the base URL of the web service and the path of the resource.
Go to the "Vars" tab. Add two variables to "Pre Request". Give one variable the name "endpoint-prefix" with the value "rest/v1/". Give the other variable the name "resource" with the value "session". Use the variables in the address line in double curly brackets, here with
{{endpoint-prefix}} and {{resource}}.

The address line should then look like this:
GET https://192.168.48.77/openmrs/{{endpoint-prefix}}{{resource}}
representing https://192.168.48.77/openmrs/ws/rest/v1/session

This client request outputs the session token via which the user "superman" is logged in. Send the request by clicking on the "→" arrow to the right of the address bar.
If the request was successful, the status "200 OK" is given as the response and the payload is displayed in JSON format. You can view the current session of the logged-in user here.
Leave the API client open and take a look at the 3LGMZ model!

1.3. Task - 3LGM² Model MI-Lab

Open the 3LGM² model "MI-Lab_v6.z3lgm". In the model browser, go to "02a - BAHMNI (detailed)" or alternatively go to the logical level, hold down alt and double-click on
"EMR & Hospital System (Bahmni)". At the logical level, you can now see the application systems that are integrated in Bahmni Standa rd[8]and the associated interfaces.

Go to the functional level. Here you can see a selection of the object types used in OpenMRS Bahmni[(9)]
a) Open the "Person" object type and go to the "Representation forms" tab. How is the object type "Person" represented? What represents the "Patient" object type?

8 https://demo.mybahmni.org/
9 OpenMRS data model: https://docs.openmrs.org/datamodel/

Let's take a look at the resources that are communicated via the REST API.
b) Double-click on "REST person resource" and then go to the "Services" tab. Which REST services have the resource "REST person resource" as a result?

c) Open one of these services, e.g. "OpenMRS Create a person". Look at the corresponding properties of the request in the "REST Properties" tab.
All services can also be opened in the logical tool level under Services.

2. REST web service

2.1. Task - Interfaces and services

Go back to the functional level. Start an analysis for the "Patient" object type (right-click on the object type) and look at the logical level to answer the questions.

a) Where is it communicated? Between which application systems is it communicated?

b) Which module interfaces can communicate with it? What interfaces are provided by the OpenMRS web services? Which application systems call the REST interface (JSON)?

2.2. Task - REST requests, search for people

The Bahmni Web Frontend uses the REST API of OpenMRS to search for or new patients. In this task, a person is to be searched for in OpenMRS. Using the 3LGM² model, you should send REST requests with the API client Bruno.

a) for the REST service "OpenMRS Search person by name" under Services. the necessary attributes for the request from the model. Open the API client again. Clone your last request. To do this, click on Clone in the three-dot menu of "1-2 Session". Give the new request the name "2-2 Search person". In the new request, change the requested resource to "person" in the "Vars" tab. Specify the request in the "Params" tab. Click on
"Add Param", give the parameter the name "q" with any value for the name of the person you are looking for.
The address line should look like this, for example:
GET https://192.168.48.77/openmrs/{{endpoint-prefix}}{{resource}}?q=Martha
Send the request. If the request is successful, the status "200 OK" is displayed and the persons found are listed in the payload.

b) The URI used by openMRS is the Universally Unique Identifier (UUID). The UUID makes the resource of type "person" uniquely addressable. Which UUID does the person you are looking for have? Copy the UUID to the clipboard (example "uuid": "b877bfd4-9fc0-4b7c-8b25-1b64ef099041").

c) Search for the "OpenMRS Retrieve person by uuid" service in the model. Clone the last query and name the new one "2-2 Retrieve person by uuid". Deactivate the query "q" in the tab "Params". In the "Vars" tab, a slash "/" and the copied UUID after "person".

2.3. Task - Create person and patient

In this task, a person and patient are to created. The person created is to become a sub-resource of Patient. Use the 3LGMZ model and the API client to process the task.

a) In the API client, clone the last request and name the new one "2-3 Create a person". Find the REST service "OpenMRS Create a person" in the model under Services. Take the necessary attributes for the request from the model (Note: Body(JSON) is not displayed in full). Enter the attributes of the person in JSON format under the "Body" tab. Change the values for "names", "gender", "birthdate" and "addresses" as required. Under "Vars", change the path to "person". The HTTP method "POST" is used to create the resource. Select this from the drop-down list to the left of the address line. Send the request.

If the status "201 Created" is reported back, the creation was successful. The newly created resource is in the payload together with the newly created UUID. Make a note of the UUID.

b) In the API client, clone the last request and name the new one "2-3 Create a patient".
c) To create a patient, the UUID of a person (see a)) and an unused patient ID (patient.identifiers.identifier) are required. This information is transmitted via POST in a message body. You can find a template for the message body in the model under Services the REST service "OpenMRS Create a patient". Go to the "REST Properties" tab in the "Body (JSON)" field. Copy the JSON template in Bruno to the "Body" tab (if necessary, change the body type from "No Body" to "JSON". Now insert the UUID of the "person" noted in a).
a random identifier to the patient, starting with ABC21xxxx. Remember to select the HTTP method "POST". Send the request.
If the status 400 is "Bad Request", look under error.globalErrors for the error message. If the identifier you created is already in use, think of a new one and repeat the request.
If status 201 Created is reported, the creation of the resource was successful.

If everything has worked, you can call up the newly created patient via the Bahmni web interface: https://192.168.48.77/bahmni/home/ → Log in → Clinical → All → Enter name or identifier → Search.

2.4. Task - Display, change and delete personal data

, change and delete personal data. Use the 3LGMZ model to process the task.

a) Personal data can requested as sub-resources. To do this, send the UUID of the resource in the request and the sub-resource you are looking for. Search for the REST service in the model under Services "OpenMRS Retrieve person name subresource by uuid.
Clone the request "2-2 Retrieve person by uuid" and name it "2-4 Retrieve person name subresource". Go to the "Vars" tab. Change the variable "resource" by replacing the existing UUID with the UUID of the person you are looking for. Take the UUID of the new person you have created from 2.3 a). Add "/name" after it to request the sub-resource "name". Send the request.

b) Person data can be changed directly in the resource. Clone the last request and name it "2-4 Update person Attributes". Go to the "Vars" tab and remove "/name" of the variable "resource".
Search for the corresponding service in the model. Take the necessary attributes from the model (note: Body(JSON) is not displayed in full). Enter the attributes of the person in JSON format under the "Body" tab. Enter any new values for name, gender and date of birth.
Use the HTTP method "POST" and send the request.

c) Have the sub-resource "/name" displayed again. Under the "Params" tab, use the query "v" with the value "full".
- What has changed?
- How many names does the patient have? Take a look the "names" attribute.

Look at the patient in BAHMNI check whether the new values available.

d) If the patient from c) has more than one name, delete the last name. Clone the request "2- 4 Update person Attributes" and name it "2-4 Delete person name subresource". Search for the corresponding service in the model. Notes: For the request, the UUID of the resource "person" and the UUID of the sub-resource "name" are required. The HTTP method "DELETE" is used to delete an entry. To permanently delete the sub-resource, use the following under "Params" the query "purge" with the value "true".
If you receive the HTTP status code 204, the request was successful.

3. Atom Feed

In order to integrate other application systems, they must be informed of changes in OpenMRS. Atom Feed makes it possible to publish events, such as the creation and editing of web resources. When an event occurs, an event feed (patient, encounter, lab and medication feed) is via a publisher. Other application systems subscribe to the feed to be notified of events. The feed contains the address (URL) of the corresponding resource. The systems that have subscribed to the feed can request relevant data via the REST interface and add it to their database. [10]

3.1. Task - Patient list via Atom Feed

Use the model to the client requests. the latest patients in the openMRS patient feed. Clone the first request "... Session" and name it "3-1 Atom Feed Patient". Under "Vars", give the "endpoint-prefix" the new value "atomfeed/" and "resource" the value
"/patient/recent". Send a "GET" request.

- What is in the entries?
- Where is the URL to the corresponding REST resource?
- the REST resource via the URL. To do this, create a new request copy the URL into the address bar.

3.2. Task - Insert OPD Visit (consultation)

View your patient on the Bahmni web interface in the browser and add any notes to the consultation.
Go to the Bahmni main page: https://192.168.48.77, click on "Clinical Service" log in if necessary.
User: superman
Password: Admin123

Go to Registration and search for your patient. Open the patient and click on "Start OPD Visit". The patient is now active for the medical consultation. Go back to the Bahmni Dashboard via the Home icon. Then go to "Clinical", your patient, go to "Consultation" and "Consultation Notes". Enter a doctor's note and finish by clicking Save.

3.3. Task - Retrieve OPD Visit,

Use the model to determine the client requests. the latest encounters of the openMRS Encounter Feed. Clone the last request and name it "3-3 Atom Feed Encounter".

a) the last contact (Encounter) via Atom Feed. It should contain an entry for your patient's outpatient consultation (OPD Visit). Can your Encounter be found using the feed? If so, how?
b) Query resources via the URLs provided.

10 https://bahmni.atlassian.net/wiki/spaces/BAH/pages/3506200/ Atom+Feed+Based+Synchronization+in+Bahmni

4. FHIR web service

FHIR (Fast Healthcare Interoperability Resources) is an HL7 standard for representing healthcare information electronically. OpenMRS has been implementing FHIR to ensure better interoperability between healthcare information systems.[11]

4.1. Task - FHIR requests

The FHIR web service from OpenMRS supports the "GET" method.(12)13This can be used to search for, query and list resources from the RESTful server. Clone the last request and name it "4-1 FHIR Search Patient". Under "Vars", give the "endpoint-prefix" the new value "fhir2/R4/".
Use the model again to the tasks. Search for FHIR Services.

a) Ask for a list of patients.
b) Search for all patients with the name "Bilbo". Use the parameter "?name=Bilbo".
   - How many do you find?
c) Search for all resources of the type "Observation" with the patient "Bilbo".
   - What is the query? How many hits are there?

11 https://fhir.openmrs.org/
12 Supported search queries: https://openmrs.atlassian.net/wiki/spaces/projects/pages/26938492/ARCHIVED+FHIR+Support+in+FHIR2+Module+version+1.3.0
13 Supported search parameters: https://fhir.openmrs.org/artifacts.html

5. Solutions

5.1. Solution 2.2 Task - REST requests, search for people

GET https://192.168.48.77/openmrs/ws/rest/v1/person?q=Martha
GET https://192.168.48.77/openmrs/ws/rest/v1/person/{person_uuid}
POST https://192.168.48.77/openmrs/ws/rest/v1/person
Body: JSON

```
{
   "names": [
      {
      "givenName": "Test",
      "familyName": "Person"
      }
   ],
   "gender": "M",
   "birthdate": "1990-09-02",
   "addresses": [
      {
      "address1": "Musterstrasse 1",
      "cityVillage": "Halle",
      "country": "Germany",
      "postalCode": "06112"
      }
   ]
}
```

5.2. Solution 2.3 Task - Create person and patient

GET https://192.168.48.77/openmrs/ws/rest/v1/patient?q=Martha
GET https://192.168.48.77/openmrs/ws/rest/v1/patientidentifiertype
POST https://192.168.48.77/openmrs/ws/rest/v1/patient
Body: JSON

```
{
   "person": "{person_uuid}",
   "identifiers": [
      {
         "identifier": "ABC200001",
         "identifierType": "b9a9e100-f496-11ed-b02c-0242ac150003",
         "preferred": false
      }
   ]
}
```

5.3. Solution 2.4 Task - Display, change and delete personal data

GET https://192.168.48.77/openmrs/ws/rest/v1/person/{person_uuid}/name
DELETE https://192.168.48.77/openmrs/ws/rest/v1/person/{person_uuid}/name/{name_uuid}?purge=true

5.4. Solution 3.1 Task - Patient list via Atom Feed

The entries in the Open MRS Feed Publisher include the name of the resource, the Atom Feed ID, the publication date and the content:

```
<content type="application/vnd.atomfeed+xml">
<![CDATA[/openmrs/ws/rest/v1/patient/80bb5371-1173-486d-b969-30c7815bcc74?v=full]]>
</content
```

GET https://192.168.48.77/openmrs/ws/rest/v1/patient/{patient_uuid}?v=full

The name or UUID of a patient is not stored in the feed. You can only find the contacts using the timestamp.