EHRServer v2.2 guide

Clinical Data Management and Sharing Platform

Compliant with:

Author: Pablo Pazos Gutiérrez – www.CaboLabs.com – pablo.pazos@cabolabs.com

Index

1. Introduction

EHRServer is an Open Clinical Data Management and Sharing Platform. It is free open source software, composed of an openEHR-compliant Clinical Data Repository, a simple yet powerful REST API, and an Administrative Web Console for EHR management and audit.

EHRServer was designed and developed by Pablo Pazos Gutiérrez[1] at CaboLabs Health Informatics[2].

If you want to know more, this article tells the EHRServer history.

Are you planning to use the EHRServer? let us know!

- pablo.pazos@cabolabs.com
- info@cloudehrserver.com
- https://twitter.com/CloudEHRServer

I'm interested… How can I try EHRServer?

Since EHRServer is open source, you can download it and Install it in your machine. You just need MySQL, Grails Framework, and setting up some environment variables. The whole process should take you about 10 minutes.

Another option is to subscribe to CloudEHRServer, this way you will also contribute with the project maintenance and support: https://cloudehrserver.com/pricing

[1] https://www.linkedin.com/in/pablopazosgutierrez
[2] https://cabolabs.com/

2. Installing EHRServer in your machine

2.1. Prerequisites

2.1.1 Download and Install MySQL Server

Check: https://dev.mysql.com/downloads/mysql/

2.1.2. JDK8 (not JRE, not JDK 11, this is important!)

EHRServer was tested under Java 8. To run locally from the source code, it needs JDK, not just JRE. To check if you have JDK and the version, you can test from the terminal:

> javac -version

An expected result would be similar to “javac 1.8.0_242” for JDK8.

2.1.3. Install Grails 3.3.10 (the version this is important!)

You can download it from http://www.grails.org/download.html or you can use SDKMAN! to install it.

Using SDKMAN (Linux/MacOS)

> curl -s get.sdkman.io | bash
> source "$HOME/.sdkman/bin/sdkman-init.sh" // $HOME is the user home folder
> sdk help // to check it was installed
> sdk install grails 3.3.10
> set version by default: Y
> grails -version

2.2. Installing

2.2.1. Download EHRServer

You can download latest development version of EHRServer from here:
https://github.com/ppazos/cabolabs-ehrserver/archive/master.zip

You can download the latest release from here:
https://github.com/ppazos/cabolabs-ehrserver/releases

2.2.2. Configure the database

Edit the dataSource in application.yml for the "development" environment, see:

https://github.com/ppazos/cabolabs-ehrserver/blob/master/grails-app/conf/application.yml#L209-L211

If the database you configured doesn't exist, you need to create it in your DBMS (e.g. MySQL) before running EHRServer.

2.2.3. Create working folders and configure paths

The working folders are configured per environment. That means those folders will be different if EHRServer is running on development, production or test environments[3].

opts & opts/base_opts

The project includes a folder called "opts". Inside there is a "base_opts" folder, where the default Operational Templates (definitions of openEHR clinical documents) are located. When the EHRServer is started, the OPTs from "base_opts" are copied (and renamed) to "opts", only those definitions will be used by the EHRServer. You can move the "opts" folder to any location, but you need to update the entry "app.opt_repo" in application.yml to reflect the new location of the folder.

versions

You need to create a working folder to store the committed versions. That folder should have permissions to read and write. After you create that folder, you need to update the entry "app.version_repo" on application.yml.

By default, that folder is ehrserver/versions, where "ehrserver" is the folder in which the EHRServer code is.

commits

[3] http://docs.grails.org/2.5.6/guide/conf.html#environments

This working folder will contain full logs of the commit contents for audit purposes.

2.2.4. Run the EHRServer

Run:

Execute this command line from the project folder:

```
> ehrserver/ grails -Dserver.port=8090 -Duser.timezone=UTC run-app
```

This will run the server locally, on the port 8090, using the Grails "development" environment and default UTC time zone for consistent timing. You will be able to access it in your browser from: http://localhost:8090/

Check section 3 "Quick guide on using the Administrative Web Console" for instructions on how to login.

2.2.5. Create environment variables (for "create account" and "forgot password" features)

When an account is created, EHRServer needs to send an email with some basic account information, and a link to reset the password. The email service needs to be configured to do that. These environment variables are used to configure an SMTP server to send emails:

- EHRSERVER_EMAIL_HOST: URL / IP of your SMTP server
- EHRSERVER_EMAIL_PORT: port number of your SMTP server
- EHRSERVER_EMAIL_USER: valid user on your SMTP server (probably an email address)
- EHRSERVER_EMAIL_PASS: password corresponding to the user
- EHRSERVER_EMAIL_FROM: the email address that will appear to the receiver as "from"

You can see where this configuration is used at:

https://github.com/ppazos/cabolabs-ehrserver/blob/master/grails-app/conf/application.yml#L93-L99

2.2.6. Environment variables needed for production deployment

In order to deploy EHRServer in production, a set of environment variables should be set. Here we describe each variable.

EHRSERVER _MYSQL_DB_HOST

Is the IP of the MySQL database that will be used in production.

Default value: depends on your environment.
Used from: application.yml

EHRSERVER _MYSQL_DB_PORT

Is the port of the MySQL database that will be used in production.

Default value: 3306
Used from: application.yml

EHRSERVER _MYSQL_DB_USERNAME

Is the username used to connect to the MySQL database that will be used in production.

Default value: depends on your environment.
Used from: application.yml

EHRSERVER _MYSQL_DB_PASSWORD

Is the password used to connect to the MySQL database that will be used in production.

Default value: depends on your environment.
Used from: application.yml

EHRSERVER_DB_NAME

It's the application name. It is used as the database name. Just change this for the database name.

Default value: ehrserver
Used from: application.yml

2.2.6. Other environment variables for dev and prod

EHRSERVER_ALLOW_WEB_USER_REGISTER

Set this variable to true to allow users to register themselves in the EHRServer from the Web. If it's set to false, users will be created only from the Web Console or from the API.

Used from: application.yml

EHRSERVER_REST_SECRET

This variable is used to generate valid API tokens when a user is authorized from the REST API. It is important that when you deploy a new instance of the EHRServer, this variable is modified.

Used from: application.yml

EHRSERVER_SNQUERY_KEY

This variable is used to connect to the SNQUERY service that provides resolution for SNOMED CT expressions inside openEHR data queries, enabling advanced semantic search. For testing purposes you can use the limited key "942645e3-a305-4d0d-9de1-145d65d8b2c4". To obtain an unlimited key please contact info@cabolabs.com

Used from: QuerySnomedService.groovy

2.2.7. Support for AWS S3 file storage

By default EHRServer stores files in the commit, versions and opt local folders, but it can be configured to use AWS S3 service to store those files remotely. First these environment variables should be set:

EHRSERVER_S3_ACCESS

This is the AWS S3 access key.

EHRSERVER_S3_SECRET

This is the AWS S3 secret key

EHRSERVER_S3_BUCKET

This is the AWS S3 bucket where the files will be stored.

EHRSERVER_S3_REGION

This is the AWS S3 server region

To activate the use of S3 instead of local folders, the correspondent services should be activated in resources.groovy, commenting out the file system services (FS) and uncommenting the S3 ones:

https://github.com/ppazos/cabolabs-ehrserver/blob/master/grails-app/conf/spring/resources.groovy#L10-L13

3. Quick guide on using the Administrative Web Console

You are busy, we know it. This is the shortest explanation of what you can do by using the EHRServer Web Console.

3.1. Sample user accounts

After you installed the EHRServer locally, three users are created, the credentials of those users are:

- admin@cabolabs.com / admin
- orgman@cabolabs.com / orgman

This is: email / password

The first user is a global administrator, the second is an organization administrator. Roles are no so important right now, so if you want to see everything, login as an admin!

3.2. Login and create some EHRs

With your account an organization is also created. That can be a clinic, hospital, or even a software provider.

Login into the EHRServer, using your email and password.

Create some EHRs. Those will be associated with your organization. Go to EHRs > New EHR, fill the form and click on “Create”.

Note: *the EHR’s patient is referenced through the Subject UID. No patient demographic data is stored in the EHRServer. This is a requirement of modern EHRs and the openEHR standard. So all the information in the EHRServer is anonymous, and the Subject UID allows you to reference a patient record that is stored in an external system, like a Master Patient Index or a Demographic Server.*

Go back to EHRs, and you will have a new and empty EHR. Keep the EHR UID on sight, we will use it to commit some data to it.

So far so good, now you need to add some data an EHR!

3.3. Commit data to an EHR

The easiest way of committing data to an EHR in EHRServer, is to use Insomnia REST Client[4] and our test scripts[5]. Since committed documents should comply with a template, you should use one of the existing templates or upload your own. When you commit documents, if the template doesn't exists, the API will return an error.

1. Install Insomnia.
2. Import the JSON into Insomnia.
3. On the request folder (top left), click on the down arrow > edit environment.
4. There set the "base_url" variable with the URL where EHRServer is running.
5. Save the environment.
6. Go to the "login" request a specify values for the username, password and organization number.
7. Click on "Send", you will get the API security token, copy it.
    a. Check page 19 for the documentation of the authentication API endpoint.
8. On Insomnia, find any request named "commit XML ...", click on it and go to "headers".
9. Put the API security token after the "Bearer " (yes the space after "Bearer" is needed).
10. Put the EHR UID in the URL, like: /api/v1/ehrs/:ehruid:/compositions
11. Click on "Send", if everything went OK, you will receive a success message.
12. Check the EHRServer Web Console, on the EHR used you will have a new clinical document!
13. To commit another document using the same request, you will need to change the version.contribution.uid.value and the version.uid.value on the XML under the body tab of the request on Insomnia. Try sending a request without changing those values, and the EHRServer will respond with an error message.

3.4. I want to query data!

Once you get your data in, you don't want it to be isolated in the EHR. You want to query data and use it in different ways. You can query for clinical documents or for data values.

Go to Queries > [+] (New Query). Assign a name and select a type: "composition" means you want to query documents, "datavalue" means you want to query data.

3.4.1. Querying documents

1) Assign a name and select type "composition".

2) Select a concept to define the criteria e.g. "Blood Pressure"

3) Select a data point to define the criteria e.g. Diastolic

4) Define the criteria e.g. magnitude > 90 and units = mm[Hg]

[4] https://insomnia.rest/
[5] https://github.com/ppazos/cabolabs-ehrserver/tree/master/api

You can define as more conditions as part of your query criteria. We also defined a condition over Systolic to have magnitude > 140 and units = mm[Hg]

5) Select the criteria logic “and” or “or”, this will define how to interpret all your conditions. We selected “or”.

6) Our final criteria match all the clinical documents that have a record of high blood pressure. You can create other queries to match documents based on your requirements.

7) Click on “Create” to save your query.

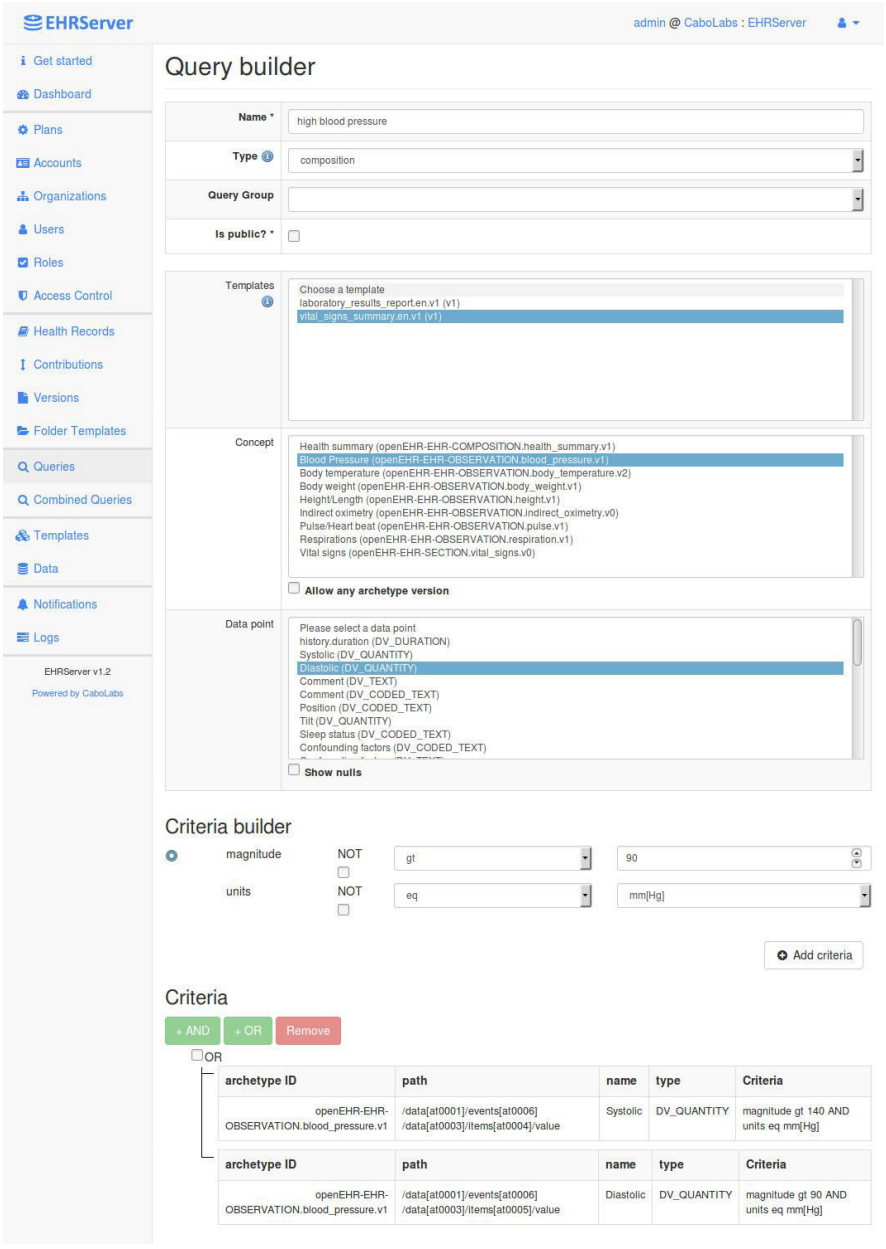


Figure 1: Query Builder – clinical document query creation

From the Query Builder you can also test your query to see if it was correctly defined and you get the data you expect. Click on “Test” and 1) Select an EHR (if no EHR is selected, the query will be executed over all the available EHRs, 2) then click on “Execute” and 3) Review the results (results are indexes, i.e. pointers to clinical docs, if you want the data, select “Retrieve data”).

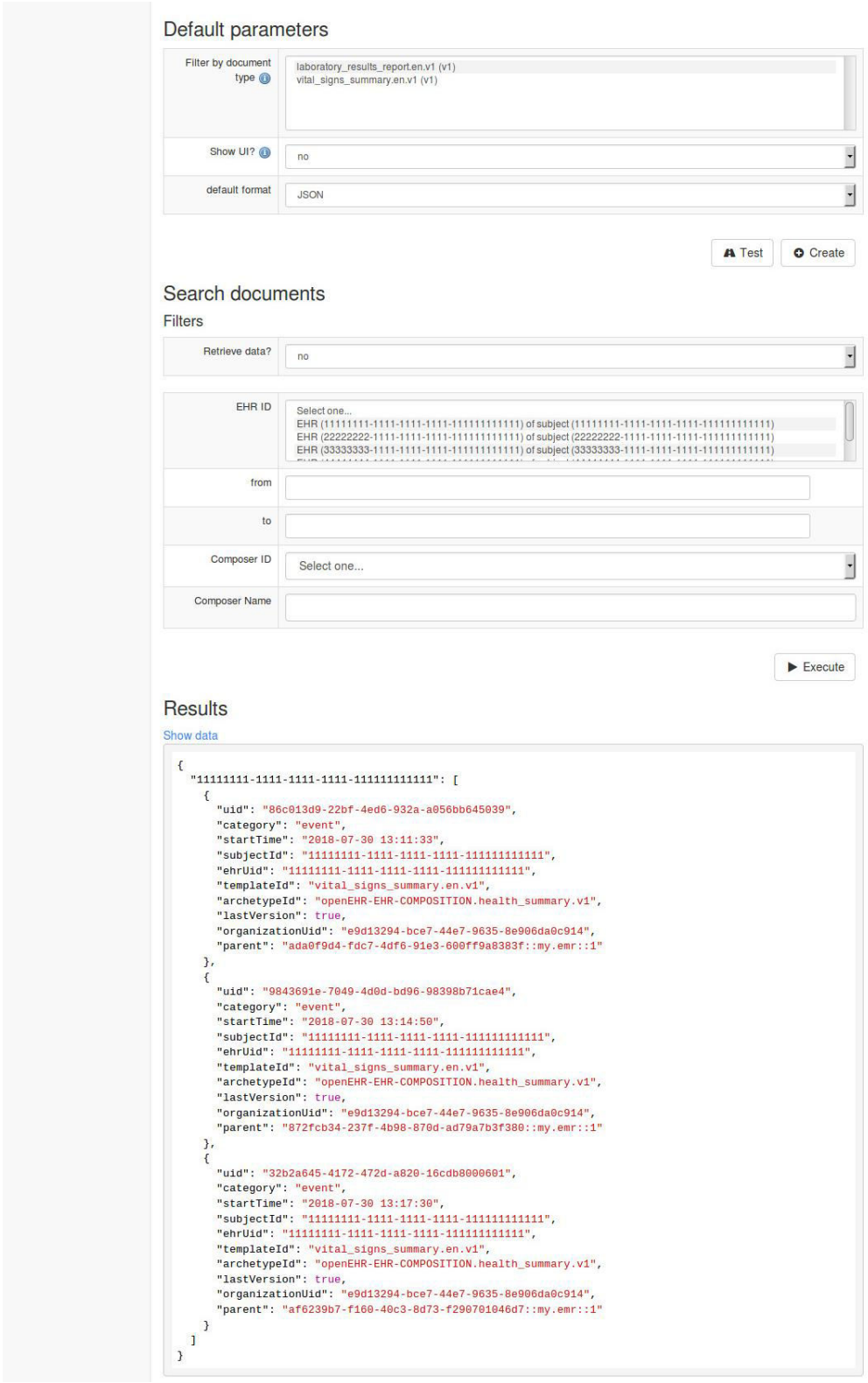


Figure 2: Query builder – clinical document query testing

3.4.2. Querying data

Querying data is pretty easy, just select Concept and Data points and click on “Add projection”, that means that you want that data in the results. In the sample below we have selected Systolic and Diastolic Blood Pressure, Body Temperature, Body Weight, Respiration Rate, and Heart Rate (Pulse), so this is a pretty complete vital sign query.

There are some output options like the output format (JSON or XML) and the default group (no grouping, group by composition or group by path).

Group by composition: data will be grouped by the clinical doc that contains the data.
Group by path: data will be grouped by the type of data, e.g. all Heart Rates will be contained on the same series (easy to chart). You can also test this query, and the process is the same as the composition query test.

If you selected JSON as the result format, grouped the results by path, and the result includes numeric data, EHRServer will generate a chart and display the results in a graphical way. This is to verify at a glance that the results are the expected ones.

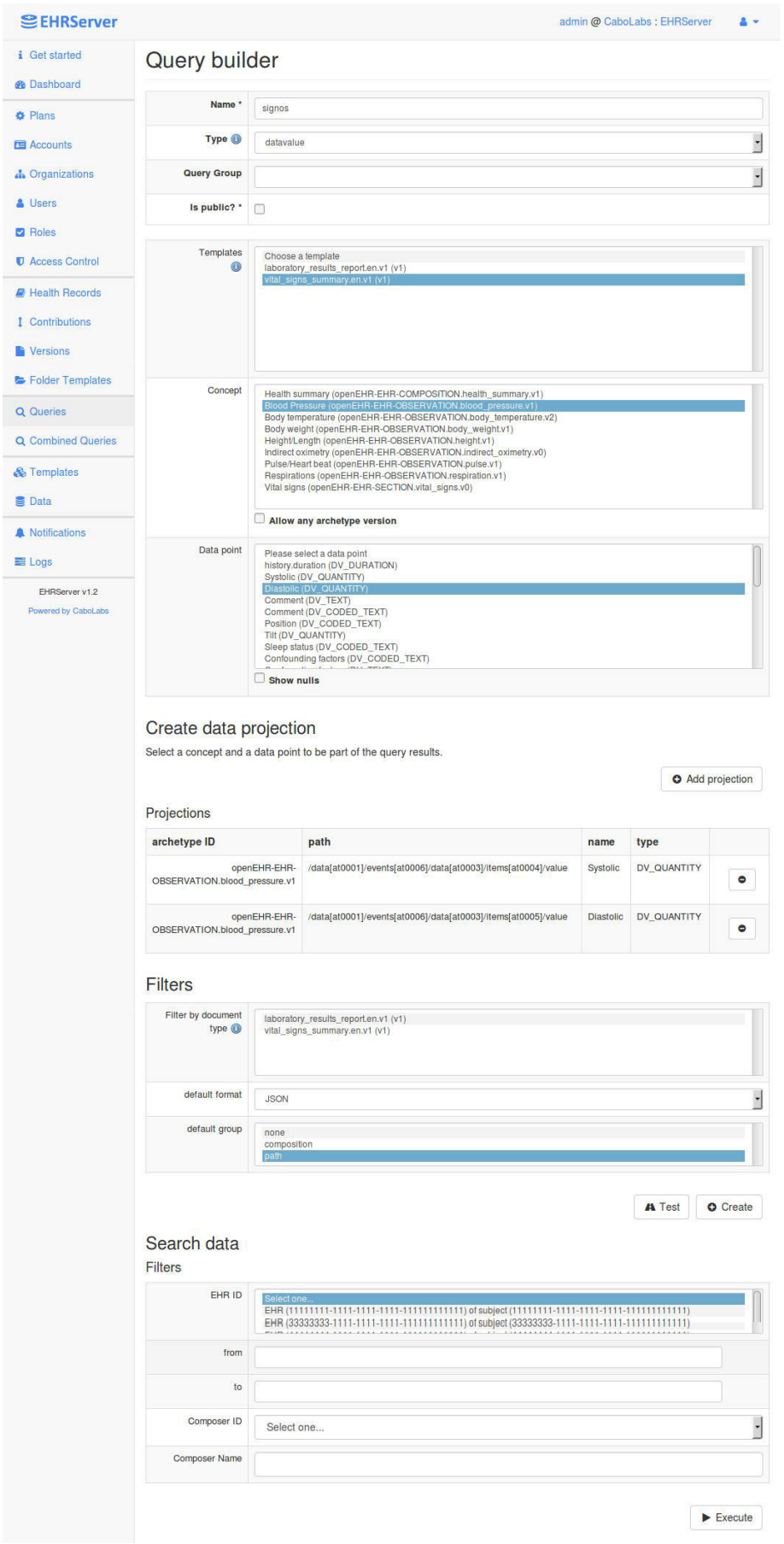


Figure 3: Query builder – datavalue query creation

Results

Show data


```
{
  "11111111-1111-1111-1111-111111111111": {
    "openEHR-EHR-OBSERVATION.blood_pressure.v1/data[at0001]/events[at0006]/data[at0003]/items[at0004]/
value<DV_QUANTITY>": {
      "type": "DV_QUANTITY",
      "name": {
        "en": "Systolic"
      },
      "serie": [
        {
          "magnitude": 142,
          "units": "mm[Hg]",
          "date": "2018-07-30 13:11:33",
          "instanceTemplatePath": "vital_signs_summary.en.v1/content[archetype_id=openEHR-EHR-SECTION.
vital_signs.v0](0)/items[archetype_id=openEHR-EHR-OBSERVATION.blood_pressure.v1](2)/data[at0001]/event
s[at0006](0)/data[at0003]/items[at0004](0)/value"
        },
        {
          "magnitude": 142,
          "units": "mm[Hg]",
          "date": "2018-07-30 13:14:50",
          "instanceTemplatePath": "vital_signs_summary.en.v1/content[archetype_id=openEHR-EHR-SECTION.
vital_signs.v0](0)/items[archetype_id=openEHR-EHR-OBSERVATION.blood_pressure.v1](2)/data[at0001]/event
s[at0006](0)/data[at0003]/items[at0004](0)/value"
        },
        {
          "magnitude": 150,
          "units": "mm[Hg]",
          "date": "2018-07-30 13:17:30",
          "instanceTemplatePath": "vital_signs_summary.en.v1/content[archetype_id=openEHR-EHR-SECTION.
vital_signs.v0](0)/items[archetype_id=openEHR-EHR-OBSERVATION.blood_pressure.v1](2)/data[at0001]/event
s[at0006](0)/data[at0003]/items[at0004](0)/value"
        }
      ]
    },
    "openEHR-EHR-OBSERVATION.blood_pressure.v1/data[at0001]/events[at0006]/data[at0003]/items[at0005]/
value<DV_QUANTITY>": {
      "type": "DV_QUANTITY",
      "name": {
        "en": "Diastolic"
      },
      "serie": [
        {
          "magnitude": 96,
          "units": "mm[Hg]",
          "date": "2018-07-30 13:11:33",
          "instanceTemplatePath": "vital_signs_summary.en.v1/content[archetype_id=openEHR-EHR-SECTION.
vital_signs.v0](0)/items[archetype_id=openEHR-EHR-OBSERVATION.blood_pressure.v1](2)/data[at0001]/event
s[at0006](0)/data[at0003]/items[at0005](1)/value"
        },
        {
          "magnitude": 96,
          "units": "mm[Hg]",
          "date": "2018-07-30 13:14:50",
          "instanceTemplatePath": "vital_signs_summary.en.v1/content[archetype_id=openEHR-EHR-SECTION.
vital_signs.v0](0)/items[archetype_id=openEHR-EHR-OBSERVATION.blood_pressure.v1](2)/data[at0001]/event
s[at0006](0)/data[at0003]/items[at0005](1)/value"
        },
        {
          "magnitude": 102,
          "units": "mm[Hg]",
          "date": "2018-07-30 13:17:30",
          "instanceTemplatePath": "vital_signs_summary.en.v1/content[archetype_id=openEHR-EHR-SECTION.
vital_signs.v0](0)/items[archetype_id=openEHR-EHR-OBSERVATION.blood_pressure.v1](2)/data[at0001]/event
s[at0006](0)/data[at0003]/items[at0005](1)/value"
        }
      ]
    }
  }
}
```

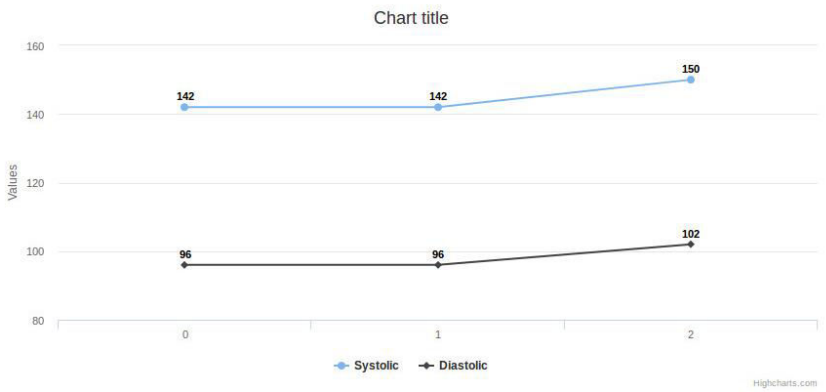
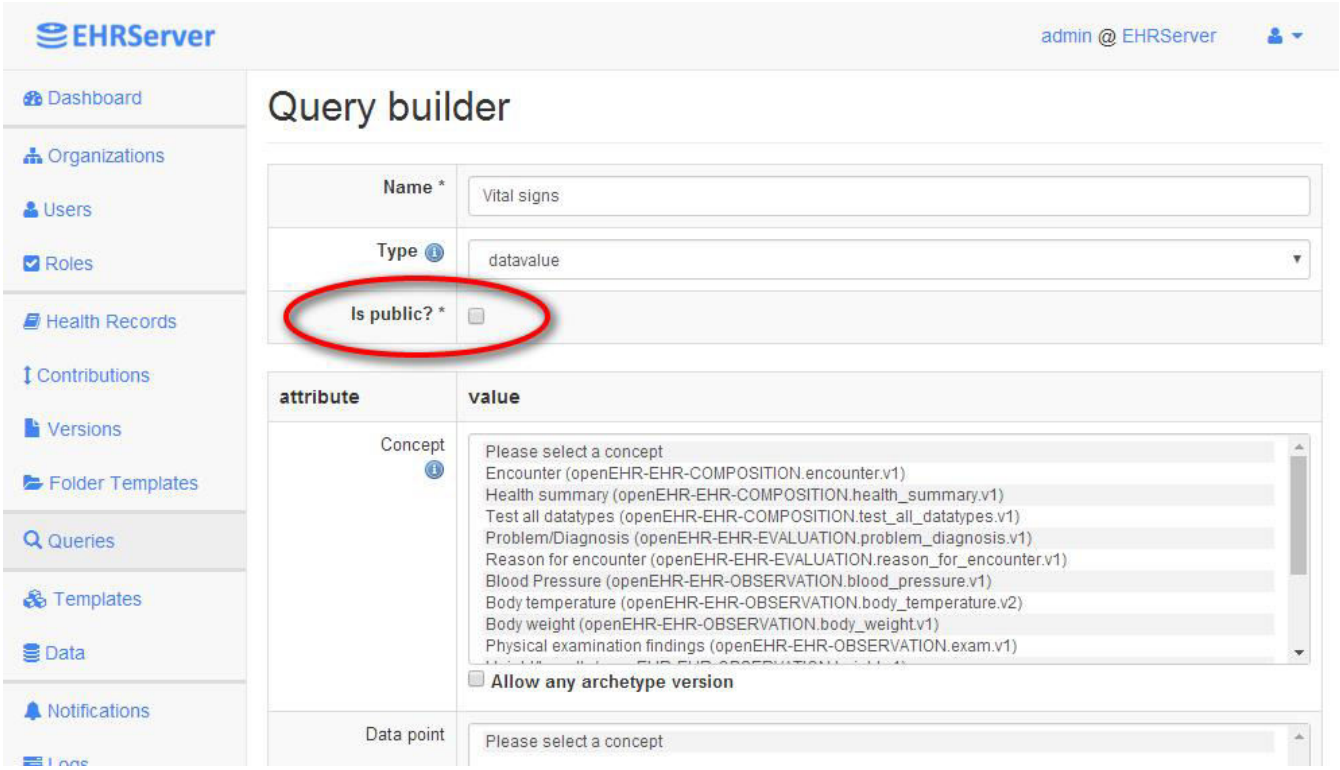



Figure 4: Query builder – datavalue query results

3.5. Managing queries

Queries can be created by any role with access to the EHRServer's Web Console. Queries can be public or private. Public queries can be used by any organization, but are created and updated only by administrators. Other roles (organization managers, account managers) can only create private queries. Private queries can be shared between many organizations under the control of the manager users.

When creating a new query, administrators will have an extra field to mark the query as public, so it will be seen by all the organizations in the EHRServer. This allows creating sample queries or very common queries once, which can be used by many, without much effort.



Query builder – admins can create public queries

For query sharing between organizations under your control, on the Query details screen there is "Share" button, there you can choose which organizations can access that query.

4. EHRServer Management

4.1. Supporting more clinical documents

Before committing any data, you need an Operational Template (OPT) that specifies the structure, semantics, constraints and terminology of your clinical documents. To upload new OPTs, you need to login and go to the Templates section > Upload Templates.

You can create your own OPTs by creating archetypes or using archetypes from the openEHR CKM (https://ckm.openehr.org/ckm/), and aggregating those archetypes in a Template using the Template Designer (https://www.openehr.org/downloads/modellingtools). You can find some OPT samples in our GitHub repo[6].

OPTs should comply with this XSD to be accepted by EHRServer:

https://github.com/ppazos/cabolabs-ehrserver/blob/master/src/main/webapp/xsd/OperationalTemplate.xsd

After you have an OPT loaded in the EHRServer, you can start committing compositions that follow the OPT definition. This way EHRServer can be extended indefinitely to support more and more clinical document structures, to be stored and queried through the EHRServer REST API.

4.2. Updating existing document definitions (OPTs)

On the Templates section of the Web Console, if an OPT is uploaded, there is a check for the template ID and the template UID. When you upload a template that has the same UID or Template ID as an existing one, the EHRServer will ask you to select that OPT and will create an internal version. This process can continue when you are developing new OPTs and updating new revisions of the same OPT for testing. Internally, the EHRServer will keep all the revisions together, and will use the last one.

Since openEHR v1.0.2 don't define a way to version OPTs, we defined that internal versioning for revisions, but for releasing a new version of an OPT after several revisions, we propose to use this format for the Template ID: concept.language.version

We also recommend to use the Template ID as the name of the OPT file, so for an encounter OPT, we can have this file name: encounter.en.v1.opt (.opt is the file extension).

Note: this is not part of the openEHR specification, but an implementation recommendation because the lack of definition on this area in the openEHR specs.

[6] https://github.com/ppazos/cabolabs-ehrserver/tree/master/opts

5. EHRServer REST API

The URLs of the endpoints documented on this section are relative to a base URL. The base URL depends on a domain or an IP address. If you have a domain like **ehrserver.cabolabs.com** (It's just an example), the base URL for the API endpoints will be:

https://ehrserver.cabolabs.com/api/v1

Where "v1" is the version of the REST API. This is to allow the evolution of the API while being able to use older versions of it.

If you run the server locally, let's say on port 8080, then the base URL would be:

http://127.0.0.1:8080/api/v1

So, if you want to invoke the GET /ehrs endpoint from a local deployment, the complete URL will look like this:

http://127.0.0.1:8080/api/v1/ehrs

Dev mode and stack traces

When running the EHRServer locally in development mode, the API errors will contain a stack trace useful for testing and debugging. The stack trace will not appear on production. To run the EHRServer in dev mode the command is "grails run-app", for production is "grails prod run-app". Also will run in production mode when deploying the EHRServer as WAR in Tomcat or other server.

5.1. POST /login

Get an authorization token to be used on all the other endpoints.

Parameters:

- email: email address associated with your account
- password: password associated with your account
- organization: organization number associated with your account

Result sample: (Content-Type: application/json)

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im9yZ21hbiIsIm..."
}
```

That token should be used to send requests to ALL the endpoints described below, by adding this header to the request:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im9yZ21hbiIsIm...
```

5.2. GET /organizations

Get the organizations associated with the authenticated user.

Parameters:

- format: result format, xml or json

Sample Result: (Content-Type: application/json)

```
[
    {
        "uid": "4f9cfbe4-8cdd-43fc-b0de-5544305fcdd5",
        "name": "Hospital CaboLabs",
        "number": "186456"
    },
    {
        "uid": "4ea72e8c-9707-4e1e-8acc-9e10e61b2037",
        "name": "Clinic Max Health",
        "number": "567895"
    }
]
```

5.3. GET /users/

Retrieves basic user data of all the users from the current organization with role equals or lower than the current user (from the API token).

Parameters:

- format: output format, valid values are “xml” or “json”.

Result sample: (Content-Type: application/json)

```
[
	{
		"username": "admin",
		"email": "admin@cabolabs.com"
	},
	{
		"username": "orgman",
		"email": "orgman@cabolabs.com"
	},
	{
		"username": "user",
		"email": "user@cabolabs.com"
	}
]
```

5.4. GET /users/$username

Get data about the user with username = $username.

Parameters:

- format: output format, valid values are “xml” or “json”.

Result sample: (Content-Type: application/json)

```
{
    "username": "orgman",
    "email": "pablo.swp+orgman@gmail.com",
    "organizations": [
        {
            "name": "Hospital de Clinicas",
            "number": "123456",
            "uid": "3edb4053-d248-46cd-87a5-d5906e1e6e32"
        }
    ]
}
```

5.5. POST /users

Register a new user for the current organization, through the API. For security reasons, the passwords can’t be set though the API and should be set by each user. The organization is taken from the Authorization Token.

Parameters:

- username: username for the new user, spaces are not allowed
- email: email of the new user, a notification to reset the password will be sent to this email
- format: result format, xml or json

Sample Result: (Content-Type: application/json)


```
{
    "username": "pablo.pazos",
    "email": "pablo.pazos@cabolabs.com",
    "organizations": [
        {
            "uid": "67559293-738c-4597-b1b0-68b6da0d8f32",
            "name": "Hospital Better Health",
            "number": "456789"
        }
    ]
}
```

5.6. GET /ehrs

Get the EHRs associated with the organization used on /login

Parameters:

- format: output format, valid values are “xml” or “json”.
- max: maximum number of ehrs to be retrieved from the offset.
- offset: results will be retrieved from the offset, default is 0 (with offset 0, ehrs will be retrieved from the first one, to the “max” one, with offset “max”, ehrs will be retrieved from “max” to “2*max”).

Result sample: (Content-Type: application/json)

```
{
    "ehrs": [
        {
            "uid": "11111111-1111-1111-1111-111111111111",
            "dateCreated": "20151125T015252,000+0000",
            "subjectUid": "11111111-1111-1111-1111-111111111111",
            "systemId": "CABOLABS_EHR_SERVER",
            "organizationUid": "cd69aa7c-0a11-46db-89c8-64435615536f"
        },
        {
            "uid": "22222222-1111-1111-1111-111111111111",
            "dateCreated": "20151125T015252,000+0000",
            "subjectUid": "22222222-1111-1111-1111-111111111111",
            "systemId": "CABOLABS_EHR_SERVER",
            "organizationUid": "cd69aa7c-0a11-46db-89c8-64435615536f"
        },
        ...
    ],
    "pagination": {
        "max": 15,
        "offset": 0,
        "nextOffset": 15,
        "prevOffset": 0
    }
}
```

5.7. GET /ehrs/$uid

Get one EHR which UID match $uid.

Parameters:

- format: output format, valid values are “xml” or “json”.
- uid: can be passed as parameter or in the URL.

Result sample: (Content-Type: application/json)

```
{
    "uid": "22222222-1111-1111-1111-111111111111",
    "dateCreated": "20151125T015252,000+0000",
    "subjectUid": "22222222-1111-1111-1111-111111111111",
```

```
    "systemId": "CABOLABS_EHR_SERVER",
    "organizationUid": "cd69aa7c-0a11-46db-89c8-64435615536f"
}
```

5.8. GET /ehrs/subjectUid/$subjectUid

Get one EHR which patient's UID match $subjectUid.

Parameters:

- format: output format, valid values are "xml" or "json".

Result sample: (Content-Type: application/json)

```
{
    "uid": "22222222-1111-1111-1111-111111111111",
    "dateCreated": "20151125T015252,000+0000",
    "subjectUid": "22222222-1111-1111-1111-111111111111",
    "systemId": "CABOLABS_EHR_SERVER",
    "organizationUid": "cd69aa7c-0a11-46db-89c8-64435615536f"
}
```

5.9. POST /ehrs

Creates a new EHR for an externally managed patient.

Parameters:

- format: output format, valid values are "xml" or "json".
- uid: optional unique identifier for the EHR, if not set the EHRServer will generate it.
- subjectUid: external identifier/reference to the patient's demographic record maintained in an external system like a Master Patient Index.

Result sample: (Content-Type: application/json)

```
{
    "uid": "42d7477c-6d01-42cf-ac35-9560f25d6ff1",
    "dateCreated": "2016-11-26 22:58:53",
    "subjectUid": "327e3a5d-dd5c-4158-88e9-a77072a5d3ce",
    "systemId": "CABOLABS_EHR_SERVER",
    "organizationUid": "77cfcb26-b9b2-4fdd-9413-14bdbaab0218"
}
```

5.10. GET /ehrs/$ehrUid/contributions

Get the audit logs for an EHR.

Parameters:

- format: output format, valid values are "xml" or "json".
- ehrUid: mandatory UID of the EHR to get the compositions from.

- from: date filter “from”, with format yyyyMMdd
- to: date filter “to”, with format yyyyMMdd
- max: maximum number of contributions to be retrieved from the offset.
- offset: results will be retrieved from the offset, default is 0 (with offset 0, contributions will be retrieved from the first one, to the “max” one, with offset “max”, contributions will be retrieved from “max” to “2*max”).

Result sample: (Content-Type: application/json)


```
{
    "contributions": [
        {
            "uid": "30de11b0-7ff8-440e-83e5-dec2a1206709",
            "organizationUid": "a82201e7-f197-4fe3-8d37-8e1fe6b33dc4",
            "ehrUid": "11111111-1111-1111-1111-111111111111",
            "versions": [
                "6f06e7f5-eccf-4e40-b7a3-9018ccaf0199::EMR::1"
            ],
            "audit": {
                "timeCommitted": "2016-06-25 06:47:37",
                "systemId": "EMR",
                "committer": {
                    "namespace": "local",
                    "type": "PERSON",
                    "value": "1324566",
                    "name": "Dr. House"
                }
            }
        },
        {
            "uid": "96d7cbb5-60b7-4714-ad36-0bbff989412b",
            "organizationUid": "a82201e7-f197-4fe3-8d37-8e1fe6b33dc4",
            "ehrUid": "11111111-1111-1111-1111-111111111111",
            "versions": [
                "53105d43-d849-463b-8a6d-f96150bc32cc::EMR::1"
            ],
            "audit": {
                "timeCommitted": "2016-06-25 06:47:47",
                "systemId": "EMR",
                "committer": {
                    "namespace": "local",
                    "type": "PERSON",
                    "value": "1324566",
                    "name": "Dr. House"
                }
            }
        }
    ],
    "pagination": {
        "max": 20,
        "offset": 0,
        "nextOffset": 20,
        "prevOffset": 0
    }
}
```

5.11. GET /compositions

Get the clinical documents from an EHR.

Parameters:

- format: output format, valid values are “xml” or “json”.
- ehrUid: UID of the EHR to get the compositions from.
- max: maximum number of compositions to be retrieved from the offset.
- offset: results will be retrieved from the offset, default is 0 (with offset 0, compositions will be retrieved from the first one, to the “max” one, with offset “max”, compositions will be retrieved from “max” to “2*max”).
- fromDate: results will have their creation date (for event) or commit date (for persistent) greater than fromDate.
- toDate: results will have their creation date (for event) or commit date (for persistent) lower than toDate.
- archetypeId: an specific COMPOSITION archetype ID to filter the results.
- category: “persistent” or “event”.

Result sample: composition index object (Content-Type: application/json)

```
{
    "result": [
        {
            "uid": "0f78e043-aa09-4212-9669-fcef0adaf470",
            "category": "event",
            "startTime": "2016-06-25 07:29:28",
            "subjectId": "11111111-1111-1111-1111-111111111111",
            "ehrUid": "11111111-1111-1111-1111-111111111111",
            "templateId": "Signos",
            "archetypeId": "openEHR-EHR-COMPOSITION.signos.v1",
            "lastVersion": true,
            "organizationUid": "d04809ca-08dc-454a-8390-96a0b125abf1",
            "parent": "90120202-e7a6-4032-a935-fe91f6e7fd28::EMR::1"
        },
        ...
    ],
    "pagination": {
        "max": 20,
        "offset": 0,
        "nextOffset": 20,
        "prevOffset": 0
    }
}
```

Note: parent has the UID of the VERSION that contains the COMPOSITION referenced by the index.

5.12. GET /compositions/$uid

Get the clinical document with UID = $uid.

Parameters:

- format: output format, valid values are “xml”, “json” or “html”.
- uid: UID if the composition (path parameter)

Result sample: composition version object (Content-Type: application/json)


```
{
    "version": {
        "@xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
        "@xmlns": "http://schemas.openehr.org/v1",
        "@xsi:type": "ORIGINAL_VERSION",
        "contribution": {
            "id": {
                "@xsi:type": "HIER_OBJECT_ID",
                "value": "ad6866e1-fb08-4e9b-a93b-5095a2563775"
            },
            "namespace": "EHR::COMMON",
            "type": "CONTRIBUTION"
        },
        "commit_audit": {
            "system_id": "CABOLABS_EHR",
            "committer": {
                "@xsi:type": "PARTY_IDENTIFIED",
                "name": "Dr. Pablo Pazos"
            },
            "time_committed": {
                "value": "20140901T233114,065-0300"
            },
            "change_type": {
                "value": "creation",
                "defining_code": {
                    "terminology_id": {
                        "value": "openehr"
                    },
                    "code_string": 249
                }
            }
        },
        "uid": {
            "value": "91cf9ded-e926-4848-aa3f-3257c1d89554::EMR_APP::1"
        },
        "data": {
            "@archetype_node_id": "openEHR-EHR-COMPOSITION.test_all_datatypes.v1",
            "@xsi:type": "COMPOSITION",
            "name": {
                "value": "Test all datatypes"
            },
            "uid": {
                "@xsi:type": "HIER_OBJECT_ID",
                "value": "d6fa1aa6-cfc7-4c28-ba51-555ee55b0ae1"
            },
     ...
}
```

5.13. POST /ehrs/$ehrUid/compositions

Commits a set of compositions to an EHR.

Parameters:

- ehrUid (in URL): UID of the EHR to commit the compositions to (path parameter)
- auditCommitter: name of the person or system that commits the composition, for audit purposes.

Preconditions

All the referenced Operational Templates should be loaded in the EHRServer to accept the commit. If any composition in the commit contains a reference to an OPT that is not loaded, an HTTP Status "412 Precondition Failed" will be returned.

Request body

The body should contain a set of versions in XML or JSON format. Each XML version should be compliant with this XSD: https://github.com/ppazos/cabolabs-ehrserver/blob/master/xsd/Version.xsd
JSON versions are transformed internally to XML and validated against the same XML Schema.

Rules (derived from the openEHR specs)

1. The EHRServer will assign version.uid.value for each version on the commit

Before EHRServer v1.3, the client needed to set the version.uid.value, now that is not required because it will be assigned by the EHRServer.

The version.uid.value will have this format: versioned_object_id::creating_system_id::version_tree_id

Where:

- versioned_object_id: is an UUID, generated by the EHRServer.
- creating_system_id: will be set to the version.commit_audit.system_id
- version_tree_id: will be 1 for new documents, or the value given by the EHRServer from the checkout endpoint.

2. Versioned documents

The commit of a new document, will generate a versioned object in the EHRServer, and will be a container for all the versions of the same document. The uid of that object will be the versioned_object_id portion of the version uid, all the versions of the same document will reference the same versioned_object_id and have incremental version_tree_id.

3. Create a new version of an existing document

In order to create a new version of an existing document, client apps should checkout the specific version, using the checkout service. Then make some changes to the document, and commit the modified document, that should include the preceding_version_uid attribute in the committed XML or JSON, and that should be set to the version.uid that should be versioned. For an example of this, check the "ehrs/$uid/compositions (new version)" request on the Insomnia script[7].

And the commit_audit.change_type information should be amendment or modification. Check the codes in the openEHR terminology (https://github.com/ppazos/openEHR-OPT/blob/master/resources/terminology/openehr_terminology_en.xml#L26-L34). The new commit will generate a new version, associate it with the versioned object and increase the version_tree_id.

4. Encoding

Client applications should encode clinical documents using UTF-8. Other encoding support can be added in the future. For now we need to keep things simple using just UTF-8.

5. Root archetype id

Remember to send the root openEHR archetype id in the data.@archetype_node_id attribute, and all the other nodes that correspond to resolved archetype slots.

6. Commit time

The version.commit_audit.time_committed that is set by client apps will be overridden by the server to be compliant with this rule from the openEHR specs:

"The time_committed attribute in both the Contribution and Version audits should reflect the time of committal to an EHR server, i.e. the time of availability to other users in the same system. It should therefore be computed on the server in implementations where the data are created in a separate client context."

Valid ISO 8601[8] dates will be parsed correctly, for example:

[7] https://github.com/ppazos/cabolabs-ehrserver/blob/master/api/EHRServer_v1.3_insomnia_5.8.4.json
[8] https://en.wikipedia.org/wiki/ISO_8601

- compact format: 20140901T233114.065-0300
- extended format: 2017-10-05T01:30:52.503Z

The time zone is needed to have a fully specified date without any ambiguity. Time zone Z is equivalent to +0000 or -0000.

***Note**: This applies to any date time data type present on the composition, including the commit time.*

If you are using XML, the valid datetime format is only the extended format[9].

7. Contributions

The parameter auditCommitter is used to create the CONTRIBUTION for each commit. To be compliant with the openEHR specs, the client system should use that data to create the VERSION.commit_audit structure. So this rule is met:
"CONTRIBUTION.audit captures to the time, place and committer of the committal act; these three attributes (system_id, committer, time_committed of AUDIT_DETAILS) should be copied into the corresponding attributes of the commit_audit of each VERSION included in the CONTRIBUTION...".

8. Empty data

In the committed XML or JSON, empty values on datatypes are not valid. If there are no values for a datatype, the whole datatype structure should not be in the committed data. Empty value on ELEMENT types can be signaled by the use of the null_flavour attribute[10], that is a DV_CODED_TEXT.

Sample XML commit

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<versions xmlns="http://schemas.openehr.org/v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <version xsi:type="ORIGINAL_VERSION">
    <contribution>
       <id xsi:type="HIER_OBJECT_ID">
         <value>ad6866e1-fb08-4e9b-a93b-5095a2563779</value>
       </id>
       <namespace>EHR::COMMON</namespace>
       <type>CONTRIBUTION</type>
    </contribution>
    <commit_audit>
       <system_id>CABOLABS_EHR</system_id>
       <committer xsi:type="PARTY_IDENTIFIED">
         <name>Dr. Pablo Pazos</name>
       </committer>
       <time_committed>
         <value>20140901T233114,065-0300</value>
       </time_committed>
       <change_type>
         <value>creation</value>
```

[9] https://www.w3.org/TR/xmlschema-2/#dateTime
[10] https://www.openehr.org/releases/RM/Release-1.0.2/docs/data_structures/data_structures.html#_overview

```
      <defining_code>
        <terminology_id>
          <value>openehr</value>
        </terminology_id>
        <code_string>249</code_string>
      </defining_code>
    </change_type>
</commit_audit>
<data xsi:type="COMPOSITION" archetype_node_id="openEHR-EHR-COMPOSITION.test_all_datatypes.v1">
  <name>
     <value>Test all datatypes</value>
  </name>
  <archetype_details>
     <archetype_id>
       <value>openEHR-EHR-COMPOSITION.test_all_datatypes.v1</value>
     </archetype_id>
     <template_id>
       <value>Test all datatypes</value>
     </template_id>
     <rm_version>1.0.2</rm_version>
  </archetype_details>
  <language>
     <terminology_id>
       <value>ISO_639-1</value>
     </terminology_id>
     <code_string>es</code_string>
  </language>
  <territory>
     <terminology_id>
       <value>ISO_3166-1</value>
     </terminology_id>
     <code_string>UY</code_string>
  </territory>
  <category>
    <value>event</value>
    <defining_code>
      <terminology_id>
        <value>openehr</value>
      </terminology_id>
      <code_string>443</code_string>
    </defining_code>
  </category>
  <composer xsi:type="PARTY_IDENTIFIED">
    <name>Dr. Pablo Pazos</name>
  </composer>
  <context>
    <start_time>
      <value>20140901T232600,304-0300</value>
    </start_time>
    <setting>
      <value>Hospital Montevideo</value>
      <defining_code>
        <terminology_id>
          <value>openehr</value>
        </terminology_id>
        <code_string>229</code_string>
      </defining_code>
    </setting>
  </context>
  <content xsi:type="OBSERVATION" archetype_node_id="openEHR-EHR-OBSERVATION.test_all_datatypes.v1">
    <name>
      <value>Blood Pressure</value>
    </name>
    <language>
      <terminology_id>
         <value>ISO_639-1</value>
      </terminology_id>
      <code_string>es</code_string>
    </language>
    <encoding>
      <terminology_id>
        <value>UNICODE</value>
      </terminology_id>
      <code_string>UTF-8</code_string>
```

```
        </encoding>
        <subject xsi:type="PARTY_IDENTIFIED">
          <external_ref>
            <id xsi:type="HIER_OBJECT_ID">
              <value>[PATIENT_UID]</value>
            </id>
            <namespace>DEMOGRAPHIC</namespace>
            <type>PERSON</type>
          </external_ref>
        </subject>
        <data xsi:type="HISTORY" archetype_node_id="at0001">
          <name>
            <value>history</value>
          </name>
          <origin>
            <value>20140101</value>
          </origin>
          <events xsi:type="POINT_EVENT" archetype_node_id="at0002">
            <name>
              <value>any event</value>
            </name>
            <time><value>20140101</value></time>
            <data xsi:type="ITEM_TREE" archetype_node_id="at0003">
              <name>
                <value>Arbol</value>
              </name>
              <items xsi:type="ELEMENT" archetype_node_id="at0011">
                <name>
                  <value>Count</value>
                </name>
                <value xsi:type="DV_COUNT">
                  <magnitude>3</magnitude>
                </value>
              </items>
            </data>
          </events>
        </data>
      </content>
    </data>
    <lifecycle_state>
      <value>completed</value>
      <defining_code>
        <terminology_id>
          <value>openehr</value>
        </terminology_id>
        <code_string>532</code_string>
      </defining_code>
    </lifecycle_state>
  </version>
</versions>
```

You can find this XML at: https://github.com/ppazos/cabolabs-ehrserver/blob/master/test/resources/commit/test_commit_1.xml

Considerations about the XML to commit:

- versions.version.commit_audit.committer.name is mandatory (required by EHRServer).
- versions.version.contribution should be the same for all the versions.version (a commit represents one contribution).
- If versions.version.data.uid is empty, the EHRServer will assign an UID to the composition.

Sample JSON commit

```
{
  "versions" : [
    {
      "version": {
        "@xmlns": "http://schemas.openehr.org/v1",
        "@xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
        "@xsi:type": "ORIGINAL_VERSION",
        "contribution": {
          "id": {
            "@xsi:type": "HIER_OBJECT_ID",
            "value": "f65421a5-6698-4603-976e-aa3dc5413534"
          },
          "namespace": "EHR::COMMON",
          "type": "CONTRIBUTION"
        },
        "commit_audit": {
          "system_id": "CABOLABS_EHR",
          "committer": {
            "@xsi:type": "PARTY_IDENTIFIED",
            "name": "Dr. Pablo Pazos"
          },
          "time_committed": {
            "value": "2018-04-20T04:01:48.717Z"
          },
          "change_type": {
            "value": "creation",
            "defining_code": {
              "terminology_id": {
                "value": "openehr"
              },
              "code_string": 249
            }
          }
        },
        "data": {
          "@xsi:type": "COMPOSITION",
          "@archetype_node_id": "openEHR-EHR-COMPOSITION.test_all_datatypes.v1",
          "name": {
            "value": "Test all datatypes"
           },
           "uid": {
             "@xsi:type": "HIER_OBJECT_ID",
             "value": "961e7542-43aa-4396-9cd9-3ffae2c36abd"
           },
           "archetype_details": {
             "archetype_id": {
               "value": "openEHR-EHR-COMPOSITION.test_all_datatypes.v1"
             },
             "template_id": {
               "value": "test_all_datatypes.es.v1"
             },
             "rm_version": "1.0.2"
           },
           "language": {
             "terminology_id": {
               "value": "ISO_639-1"
             },
             "code_string": "es"
           },
           "territory": {
             "terminology_id": {
               "value": "ISO_3166-1"
             },
             "code_string": "UY"
           },
           "category": {
             "value": "event",
             "defining_code": {
               "terminology_id": {
                 "value": "openehr"
               },
               "code_string": 433
             }
           },
```

```
  "composer": {
    "@xsi:type": "PARTY_IDENTIFIED",
    "name": "Dr. Pablo Pazos"
  },
  "context": {
    "start_time": {
      "value": "2018-04-20T04:01:48.717Z"
    },
    "setting": {
      "value": "Hospital Montevideo",
      "defining_code": {
        "terminology_id": {
        "value": "openehr"
      },
      "code_string": 229
    }
  }
},
"content": {
  "@xsi:type": "OBSERVATION",
  "@archetype_node_id": "openEHR-EHR-OBSERVATION.test_all_datatypes.v1",
  "name": {
    "value": "Blood Pressure"
  },
  "language": {
    "terminology_id": {
      "value": "ISO_639-1"
    },
    "code_string": "es"
  },
  "encoding": {
    "terminology_id": {
      "value": "UNICODE"
    },
    "code_string": "UTF-8"
  },
  "subject": {
    "@xsi:type": "PARTY_IDENTIFIED",
    "external_ref": {
      "id": {
        "@xsi:type": "HIER_OBJECT_ID",
        "value": "11111111-1111-1111-1111-111111111111"
      },
      "namespace": "DEMOGRAPHIC",
      "type": "PERSON"
    }
  },
  "data": {
    "@xsi:type": "HISTORY",
    "@archetype_node_id": "at0001",
    "name": {
      "value": "history"
    },
    "origin": {
      "value": "2018-04-20T04:01:48.718Z"
    },
    "events": {
      "@xsi:type": "POINT_EVENT",
      "@archetype_node_id": "at0002",
      "name": {
        "value": "any event"
      },
      "time": {
        "value": "2018-04-20T04:01:48.718Z"
      },
      "data": {
        "@xsi:type": "ITEM_TREE",
        "@archetype_node_id": "at0003",
        "name": {
          "value": "Arbol"
        },
        "items": {
          "@xsi:type": "ELEMENT",
          "@archetype_node_id": "at0011",
```

```
                    "name": {
                      "value": "Count"
                    },
                    "value": {
                      "@xsi:type": "DV_COUNT",
                      "magnitude": 3
                    }
                  }
                }
              }
            }
          }
        },
        "lifecycle_state": {
          "value": "complete",
          "defining_code": {
            "terminology_id": {
              "value": "openehr"
            },
            "code_string": 532
          }
        }
      }
    }
  ]
}
```

5.14. GET /queries

Get the list of queries created in the EHRServer.

Parameters:

- format: output format, valid values are “xml” or “json”.
- max: maximum number of queries in the results.
- offset: results will be retrieved from the offset, default is 0.

Result sample: (Content-Type: application/json)


```
{
    "queries": [
        {
            "uid": "7b8762ac-eaf4-435b-9fde-d59730b6641f",
            "name": "documents",
            "format": "xml",
            "type": "datavalue",
            "group": "none",
            "projections": [
                {
                    "archetypeId": "openEHR-EHR-OBSERVATION.respiration.v1",
                    "path": "/data[at0001]/events[at0002]/data[at0003]/items[at0004]/value"
                },
                {
                    "archetypeId": "openEHR-EHR-OBSERVATION.terminology_ref.v1",
                    "path": "/data[at0001]/events[at0002]/data[at0003]/items[at0004]/value"
                }
            ]
        },
        {
            "uid": "1133fa73-4bf8-43eb-95a5-e69c7a91ffc9",
            "name": "data",
            "format": "json",
            "type": "composition",
            "criteria": [
                {
                    "archetypeId": "openEHR-EHR-OBSERVATION.blood_pressure.v1",
                    "path": "/data[at0001]/events[at0006]/data[at0003]/items[at0004]/value",
                    "conditions": {
                        "magnitude": { "gt": [140.0] },
                        "units": { "eq": "mm[Hg]" }
                    }
                },
                {
                    "archetypeId": "openEHR-EHR-OBSERVATION.blood_pressure.v1",
                    "path": "/data[at0001]/events[at0006]/data[at0003]/items[at0005]/value",
                    "conditions": {
                        "magnitude": { "gt": [90.0] },
                        "units": { "eq": "mm[Hg]" }
                    }
                }
            ]
        }
    ],
    "pagination": {
        "max": 15, "offset": 0, "nextOffset": 15, "prevoffset": 0
    }
}
```

5.15. GET /queries/$queryUid

Get the query with the UID $queyrUid.

Parameters:

- format: output format, valid values are “xml” or “json”.

Result sample: (Content-Type: application/json)

```
{
   "uid": "7b8762ac-eaf4-435b-9fde-d59730b6641f",
   "name": "documents",
   "format": "xml",
   "type": "datavalue",
   "group": "none",
   "projections": [
      {
         "archetypeId": "openEHR-EHR-OBSERVATION.respiration.v1",
         "path": "/data[at0001]/events[at0002]/data[at0003]/items[at0004]/value"
      },
      {
         "archetypeId": "openEHR-EHR-OBSERVATION.terminology_ref.v1",
         "path": "/data[at0001]/events[at0002]/data[at0003]/items[at0004]/value"
      }
   ]
}
```

5.16. GET /queries/$queryUid/execute

Executes the query with the UID $queyrUid.

Parameters:

- format: output format, valid values are “xml” or “json”.
- ehrUid: UID of the EHR we want to query. If no ehrUid is specified, the result will contain results for multiple EHRs.
- organizationUid: UID of the organization that owns the EHRs we want to query.
- retrieveData: this parameter specifies if the composition query should return data if the value is “true”
- group: overrides the grouping of the datavalue query, valid values are: “none”, “composition” or “path”.
- fromDate: filter for the results, to get results after this date. Expected format is: yyyyMMdd.
- toDate: filter for the results, to get results before this date. Expected format is: yyyyMMdd.
- composerUid: filter results by the UID of the composer (author)
- composerName: filter results by the name of the composer (author), can be a partial name.

Result sample for datavalue query: (Content-Type: application/json)

```
{
  "openEHR-EHR-
OBSERVATION.blood_pressure.v1/data[at0001]/events[at0006]/data[at0003]/items[at0004]/value<DV_QUANTITY>": {
    "type": "DV_QUANTITY",
```

```
    "name": "Sistólica",
    "serie": [
      {
        "magnitude": 106,
        "units": "mm[Hg]",
        "date": "2016-01-14 07:34:59"
      }
    ]
  },
  "openEHR-EHR-
OBSERVATION.blood_pressure.v1/data[at0001]/events[at0006]/data[at0003]/items[at0005]/value<DV_QUANTITY>": {
    "type": "DV_QUANTITY",
    "name": "Diastólica",
    "serie": [
      {
        "magnitude": 56,
        "units": "mm[Hg]",
        "date": "2016-01-14 07:34:59"
      }
    ]
  },
  "timing": "10 ms"
}
```


Result sample for composition query: (Content-Type: application/json)


```
{
   "results": [
      {
         "uid": "0f78e043-aa09-4212-9669-fcef0adaf470",
         "category": "event",
         "startTime": "2016-06-25 07:29:28",
         "subjectId": "11111111-1111-1111-1111-111111111111",
         "ehrUid": "11111111-1111-1111-1111-111111111111",
         "templateId": "Signos",
         "archetypeId": "openEHR-EHR-COMPOSITION.signos.v1",
         "lastVersion": true,
         "organizationUid": "d04809ca-08dc-454a-8390-96a0b125abf1",
         "parent": "90120202-e7a6-4032-a935-fe91f6e7fd28::EMR::1"
      },
      ...
   ],
   "timing": "312 ms"
}
```


Notes:

For datavalue queries, the result structure will depend on the selected grouping. On the example above, the grouping by path is shown. On both, group by path or composition the path associated with the results will end with <datatype>, this is to avoid ambiguities between results for the same archetype and path that have alternative datatypes in the template (e.g. a node can be DV_TEXT or DV_CODED_TEXT).

For composition queries, the result is a list of indexes of compositions, like the result of the /compositions endpoint.

For composition queries, if retrieveData=true, instead of indexes of compositions, the result will include the complete compositions. We discourage the use of this parameter if the filters are too wide a lots of compositions can be retrieved (can take a while). A better approach would be to query composition indexes and then get the data for specific compositions from /compositions/$uid

The filter parameters composerUid and composerName are matched against the values that come in the composition.composer element like this:

```
<composer xsi:type="PARTY_IDENTIFIED">
  <external_ref>
    <id xsi:type="HIER_OBJECT_ID">
      <value>850609af-a0e9-4c2b-975d-51ca2ae4fec4</value>
    </id>
    <namespace>DEMOGRAPHIC</namespace>
    <type>PERSON</type>
  </external_ref>
  <name>Dr. House</name>
</composer>
```

The composerUid parameter will be matched against the composer.external_ref.id.value, the whole id should match the parameter value. And the composerName parameter will be matched against composer.name accepting partial names as parameter values, so composerName="house" will match the "Dr. House" value. The matching is case insensitive.

5.17. GET /ehrs/$ehrUid/compositions/$compositionUid/checkout

Gets the latest version of a clinical document (composition) with the aim of creating a new version of it (due a correction or an amendment of the information contained in it).

Parameters:

- ehrUid (in URL): identifier of the EHR that contains the composition.
- compositionUid (in URL): identifier of the composition to be modified, should be the latest version of the document (use GET /compositions to get the UIDs of the last versions of existing clinical documents)
- format: output format, can be xml or json

Sample Result: (Content-Type: text/xml)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<versions xmlns="http://schemas.openehr.org/v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <version xsi:type="ORIGINAL_VERSION">
     <contribution>
       <id xsi:type="HIER_OBJECT_ID">
         <value>ad6866e1-fb08-4e9b-a93b-5095a2563779</value>
       </id>
       <namespace>EHR::COMMON</namespace>
       <type>CONTRIBUTION</type>
     </contribution>
     <commit_audit>
       <system_id>CABOLABS_EHR</system_id>
       <committer xsi:type="PARTY_IDENTIFIED">
         <name>Dr. Pablo Pazos</name>
       </committer>
       <time_committed>
         <value>20140901T233114,065-0300</value>
       </time_committed>
       <change_type>
         <value>creation</value>
         <defining_code>
           <terminology_id>
             <value>openehr</value>
           </terminology_id>
           <code_string>249</code_string>
         </defining_code>
```

```
      </change_type>
    </commit_audit>
    <uid>
      <value>91cf9ded-e926-4848-aa3f-3257c1d89e37::EMR_APP::1</value>
    </uid>
        ...
```

Notes:

The returned XML will have the same structure as the documents sent in the request to POST /commit. For now /checkout supports XML only results, in the future we will add JSON support to this endpoint.

When the document is modified on a client application, it should be committed as it is (the version.uid should not be changed by the client app), and the EHRServer will generate the new version for the document, and associate the new version with the previous one, in the POST /commit call.

5.18. GET /templates

Returns a list of Operational Templates that can be accessed by the authenticated user. It returns the OPT metadata, not the OPT instances.

Parameters:

- format: result format, xml or json
- max: maximum number of results for paginated results
- offset: initial record for paginated results (for next value check the nextOffset on the result)

Sample Result: (Content-Type: application/json)

```
{
    "templates": [
    {
        "templateId": "Simple Lab Order EN",
        "concept": "Simple Lab Order",
        "language": "ISO_639-1::en",
        "uid": "77e35fd9-6ad3-4e93-a8fd-3e8aecea0f4f",
        "archetypeId": "openEHR-EHR-COMPOSITION.request.v1",
        "archetypeConcept": "Request for service",
        "isPublic": false
    },
    {
        "templateId": "Simple Encounter EN",
        "concept": "Simple Encounter",
        "language": "ISO_639-1::en",
        "uid": "47c9f4f3-5a61-44f2-aad7-7279b87814a6",
        "archetypeId": "openEHR-EHR-COMPOSITION.encounter.v1",
        "archetypeConcept": "Encounter",
        "isPublic": false
    },
    {
        "templateId": "Vital Signs",
        "concept": "Vital Signs",
        "language": "ISO_639-1::en",
        "uid": "cdbd8d8b-7c51-4508-883a-262dd11fdda1",
        "archetypeId": "openEHR-EHR-COMPOSITION.signos.v1",
        "archetypeConcept": "Vital signs",
        "isPublic": false
    },
    {
        "templateId": "Simple Vaccination Record EN",
        "concept": "Simple Vaccination Record",
```

```
        "language": "ISO_639-1::en",
        "uid": "62eaeb34-fb59-4666-b608-eb02d8abc056",
        "archetypeId": "openEHR-EHR-COMPOSITION.vaccination_list.v1",
        "archetypeConcept": "Vaccination List",
        "isPublic": false
    }
    ],
    "pagination": {
        "max": 15,
        "offset": 0,
        "nextOffset": 15,
        "prevOffset": 0
    }
}
```

5.19. GET /templates/$uid

Returns an Operational Template instance in XML. The result is just the XML of an OPT. Find examples here: https://github.com/ppazos/cabolabs-ehrserver/tree/master/opts

5.20. (experimental) POST /query/composition/execute

Executes a given composition query and returns the result. This is useful when a client system has all the information to create queries dynamically and want to execute a query without creating it from the Web Console (stored queries).

Parameters:

- format: result format, xml or json

Body: only accepts JSON queries, with this format:

```
{
  "query": {
    "name": "High blood pressure",
    "type": "composition",
    "format": "json",
    "where": {
      "_type": "OR",
      "left": {
        "_type": "COND",
        "archetypeId": "openEHR-EHR-OBSERVATION.blood_pressure.v1",
        "path": "/data[at0001]/events[at0006]/data[at0003]/items[at0004]/value",
        "rmTypeName": "DV_QUANTITY",
        "class": "DataCriteriaDV_QUANTITY",
        "allowAnyArchetypeVersion": false,
        "name": "Systolic",
        "magnitudeValue": "140.0",
        "magnitudeOperand": "gt",
        "magnitudeNegation": false,
        "unitsValue": "mm[Hg]",
        "unitsOperand": "eq",
        "unitsNegation": false,
        "spec": 0
      },
      "right": {
        "_type": "COND",
        "archetypeId": "openEHR-EHR-OBSERVATION.blood_pressure.v1",
        "path": "/data[at0001]/events[at0006]/data[at0003]/items[at0005]/value",
```

```
        "rmTypeName": "DV_QUANTITY",
        "class": "DataCriteriaDV_QUANTITY",
        "allowAnyArchetypeVersion": false,
        "name": "Diastolic",
        "magnitudeValue": "90.0",
        "magnitudeOperand": "gt",
        "magnitudeNegation": false,
        "unitsValue": "mm[Hg]",
        "unitsOperand": "eq",
        "unitsNegation": false,
        "spec": 0
      }
    },
    "select": [],
    "group": "none"
  },
  "fromDate": "20170101",
  "toDate": "",
  "format": "json",
  "qehrId": "11111111-1111-1111-1111-111111111111",
  "composerUid": "",
  "composerName": "",
  "max": 10,
  "offset": 0,
  "retrieveData": false
}
```

This is a composition query that returns all the compositions that have a “basic_demographic” record that contains the gender ("path": "/data[at0001]/items[at0002]/value") that is equals to feminine ("codeValue": "at0004").

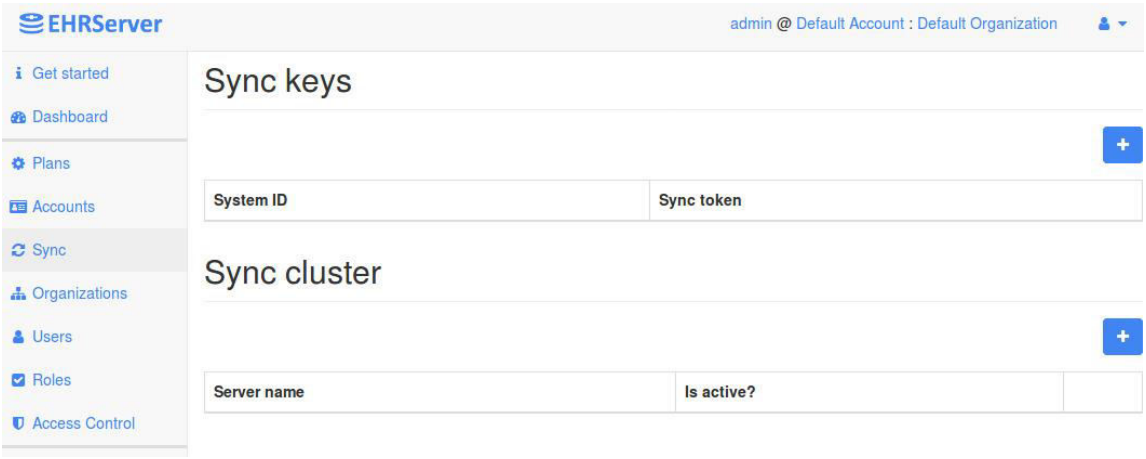
To create such query it is needed that you have the Operational Template that contains all the archetype ids, and that you can calculate the paths and get the values to use in the query conditions.

6. EHRServer Cluster Synchronization (experimental)

EHRServer v1.4 is the first release to support synchronization between different instances of EHRServer. The idea is to provide a core set of functionalities to support high availability and load balancing on a distributed network of EHRServer instances, supporting the backend of EHR systems.

The first and simplest use case for this is to have a backup server. So after configuring the cluster, when an EHR is created, an OPT uploaded, a CONTRIBUTION committed or a query created, that is automatically synchronized from a master server to one or many slave servers.

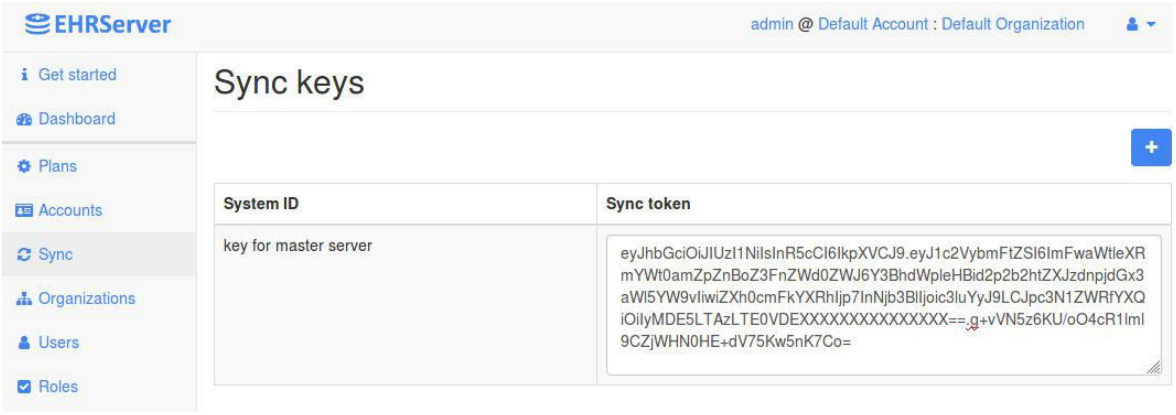
Only administrators can configure an EHRServer cluster, and setup is pretty straight forward. All the configuration is done from the Web Console. Admins will find a new Sync section in the menu, from where Sync keys can be created on slave servers, and Sync cluster configuration can be done on the master server.



Sync configuration

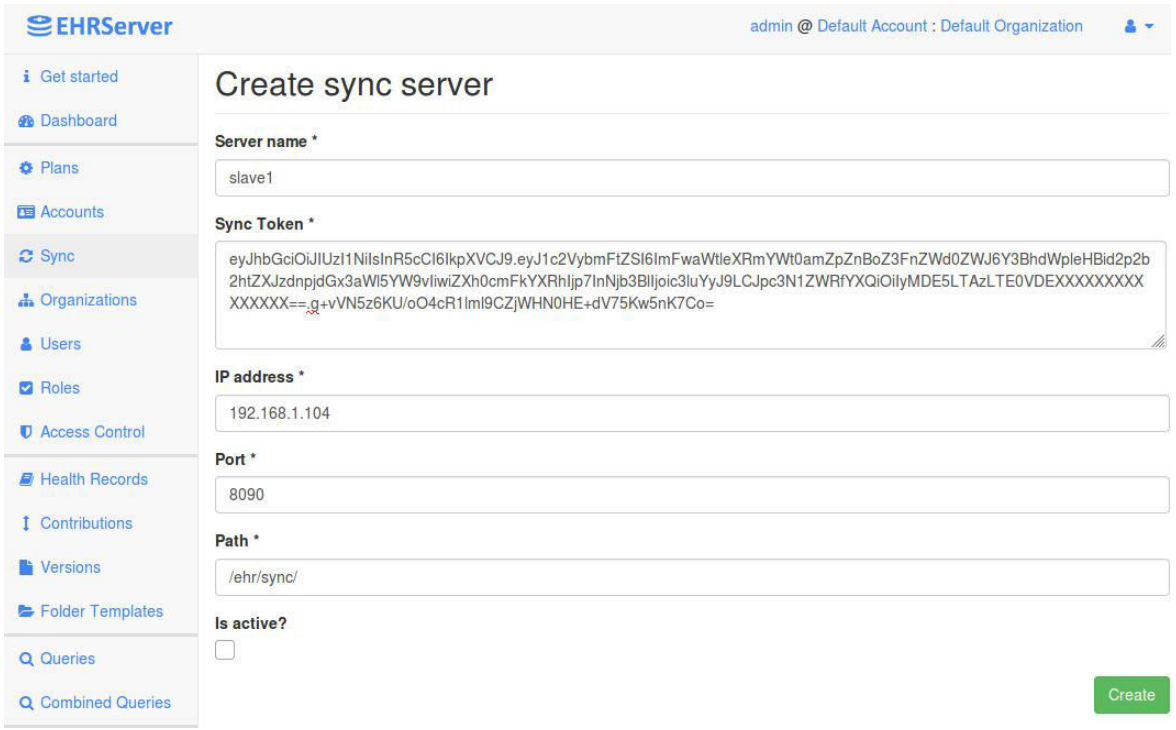
When you have two EHRServer instances running, first go to the slave one and add a new sync key. You will need that key to configure the master server, basically is the what the master server uses to authenticate itself on the slave servers to allow it to send data to the slave servers and the slave servers to accept it.

Sync key creation



Sync key created

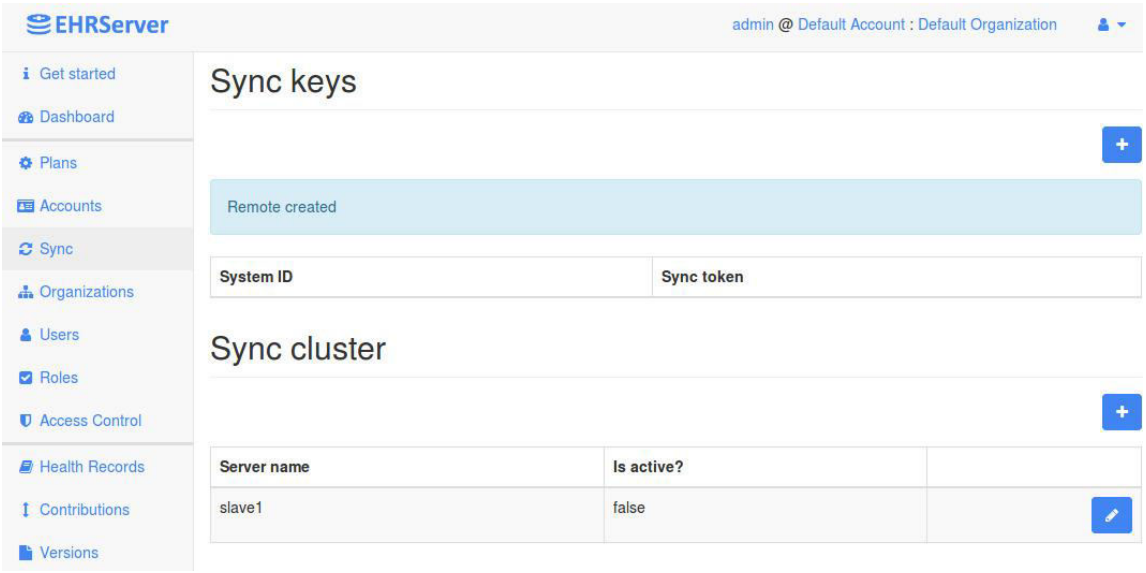
Then on the master server, you need to configure the connection to the slave servers, using the Sync key created on each one of the slaves.



Sync connection to slave

The required data is:

1. Server name: anything that you want to call the slave server
2. Sync token: the sync key created on the slave server
3. IP address: IP or domain of the slave server
4. Port: in which the slave server is listening
5. Path: is the URL path to the sync API, in development is “/ehr/sync”, in production might change but “/ehr” should be the path where the server is running (e.g. https://ip:port/base_path_to_ehrserver/sync this will be how the final URL for sync looks like).
6. Is active?: if that is checked, the server will start synchronizing with the slave.



Slave config created on the master server