Gamebus FHIR Layer

Release v0.0.1

unknown

Jul 17, 2023

CONTENTS

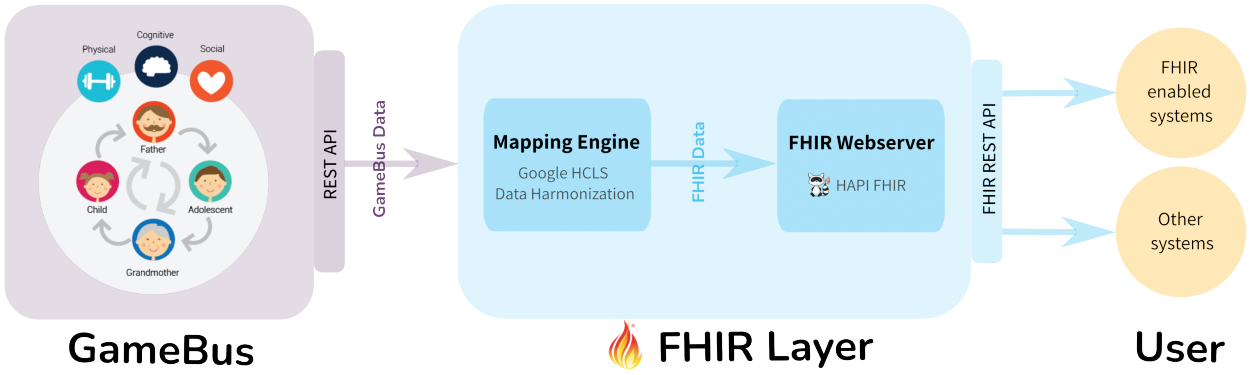
GameBus is a digital platform where you can, together with your family, friends or team, play healthy social, cognitive, and physical games in a personalized gaming experience. Like many other healthcare platforms, GameBus uses its in-house schemas to represent data and offers a specific REST API to share these data. However, this approach can lead to barriers to information exchange between platforms that do not use the same schema or API. It's like the communication challenge between two people who do not understand each other's language. A better solution would be for the different platforms to speak the same language. Here, a popular "language" is FHIR.

FHIR (Fast Healthcare Interoperability Resources) is a standard for exchanging healthcare information electronically. It describes healthcare data formats and elements and API. FHIR has been more and more widely used in industry and academia, becoming the de-facto standard.

To enable FHIR service for GameBus system, we have developed GameBus FHIR layer. It is built on a technology stack of open source software and consists of two main parts: a mapping engine and a FHIR web server. The two components are integrated into the FHIR layer in order for GameBus to provide FHIR compliant data. The FHIR layer can be deployed as a microservice. And more importantly, applying the FHIR layer to GameBus does not need any change on GameBus system.

Though the FHIR layer is developed for GameBus platform, it can be easily reused for other healthcare systems with some adaptations to the details.

The diagram below shows the relationship between GameBus system, FHIR layer, and end users.



Highlights:

- Open source
- Smooth deployment as microservice
- Adding/changing mappings with ease
- Adding/changing operations for FHIR REST API with ease
- Not only for GameBus, easy to adapt for other healthcare platforms

CHAPTER

ONE

TUTORIALS

1.1 Tutorial

1.1.1 Requirements

Note for Mac users

Mac with Apple silicon (e.g. M1, M2 cpu) is not supported

Installation

- Chrome browser (it's required because of Chrome DevTools)
- Docker, see how to install Docker

1.1.2 Start FHIR server

GameBus FHIR layer is open-source and available as a Docker image (id: `nlesc/gamebus-fhir-layer`). The most convenient way to deploy or run it is using Docker.

This tutorial will show you how to run GameBus FHIR layer using Docker locally.

Run Docker container

To start the server, type the command below in a terminal:

```
docker run -p 8080:8080 nlesc/gamebus-fhir-layer start_fhir_server https://api3-new.
↪gamebus.eu/v2
```

This command runs a Docker container using the `nlesc/gamebus-fhir-layer` image,

- `-p 8080:8080` binds the container's port `8080` to host machine's port `8080`,
- `start_fhir_server` is the command to start the HAPI FHIR server in container,
- `https://api3-new.gamebus.eu/v2` is the endpoint of GameBus API.

For more details about running a Docker container, see Docker's doc.

When the server starts, it'll be served locally on http://localhost:8080.

Now the FHIR server is ready to use. Before using it, we have to create an account on GameBus platform and add some test data to GameBus for us to request later using FHIR REST API, for details see the next section.

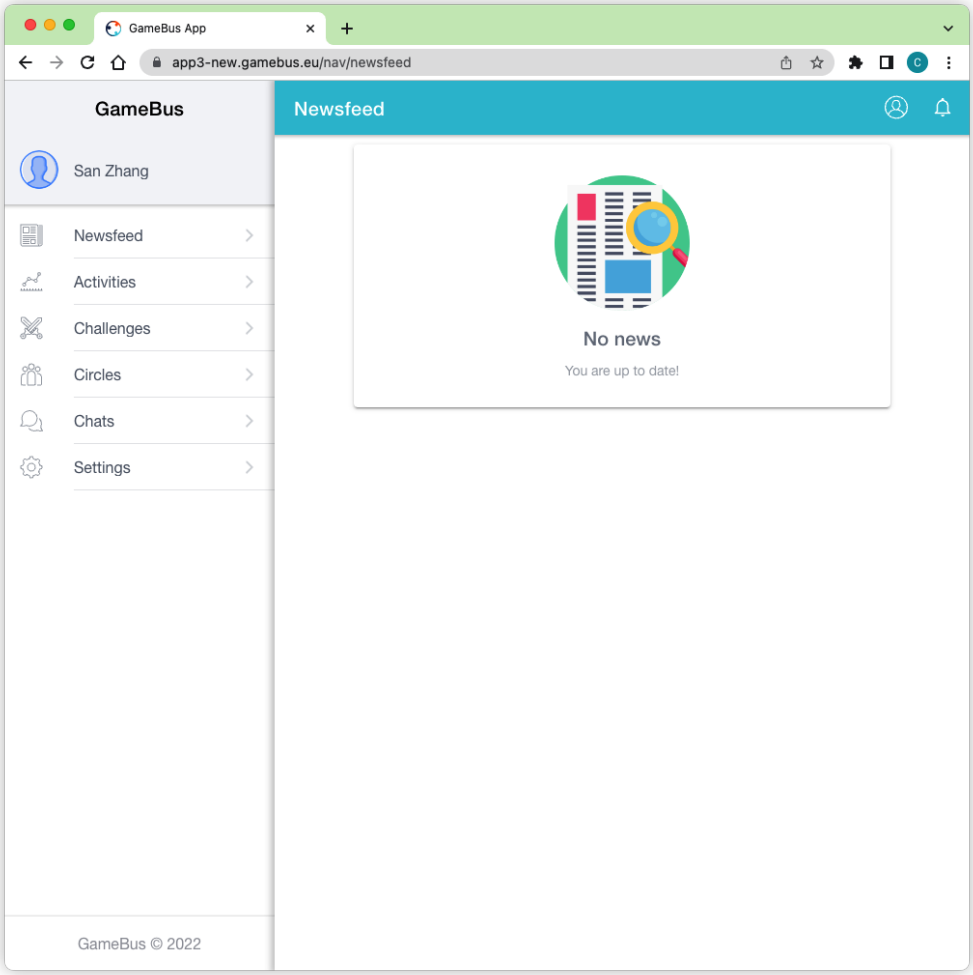
1.1.3 GameBus

GameBus FHIR layer is a wrapper on top of GameBus platform, acting like a translator. The layer itself does not have a database, so not storing any data. GameBus platform is where the data is stored.

To use the FHIR layer, users need to create an account on GameBus platform (for free) and then add some data to it for requesting later with FHIR REST API.

Create a GameBus account

You can create a GameBus account at https://app3-new.gamebus.eu. Then you will see a web app like the screenshot below.

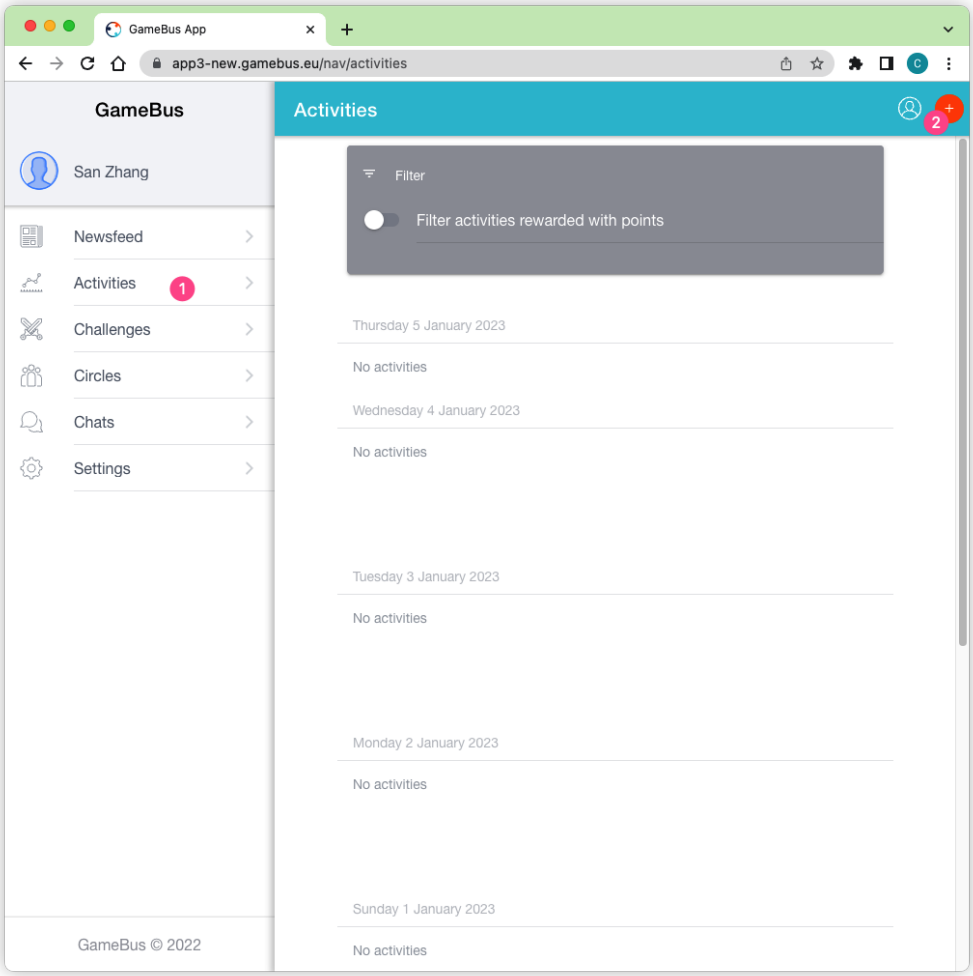


If you want to explore the full functionality of GameBus, you could check its manual at https://blog.gamebus.eu. Note that it's NOT required for this tutorial.

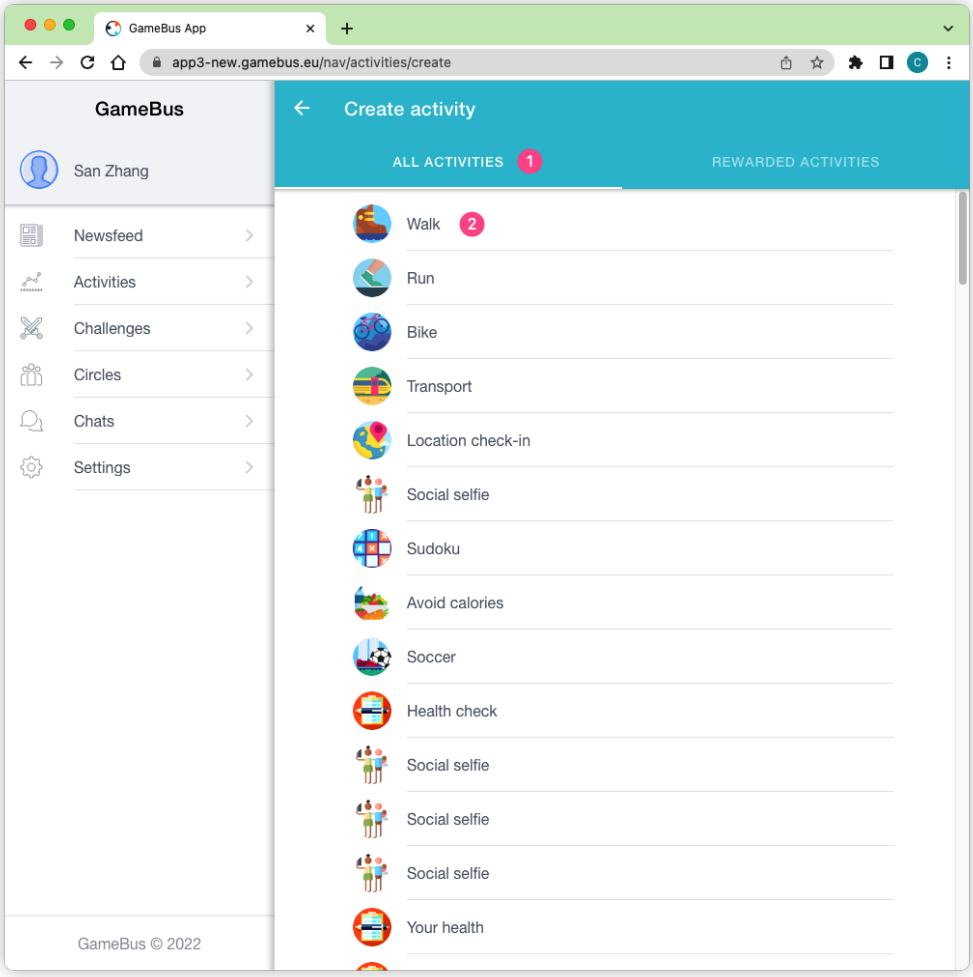
Add data manually

There are various ways to add activity data to GameBus. Here we'll manually fill some data using its web app.

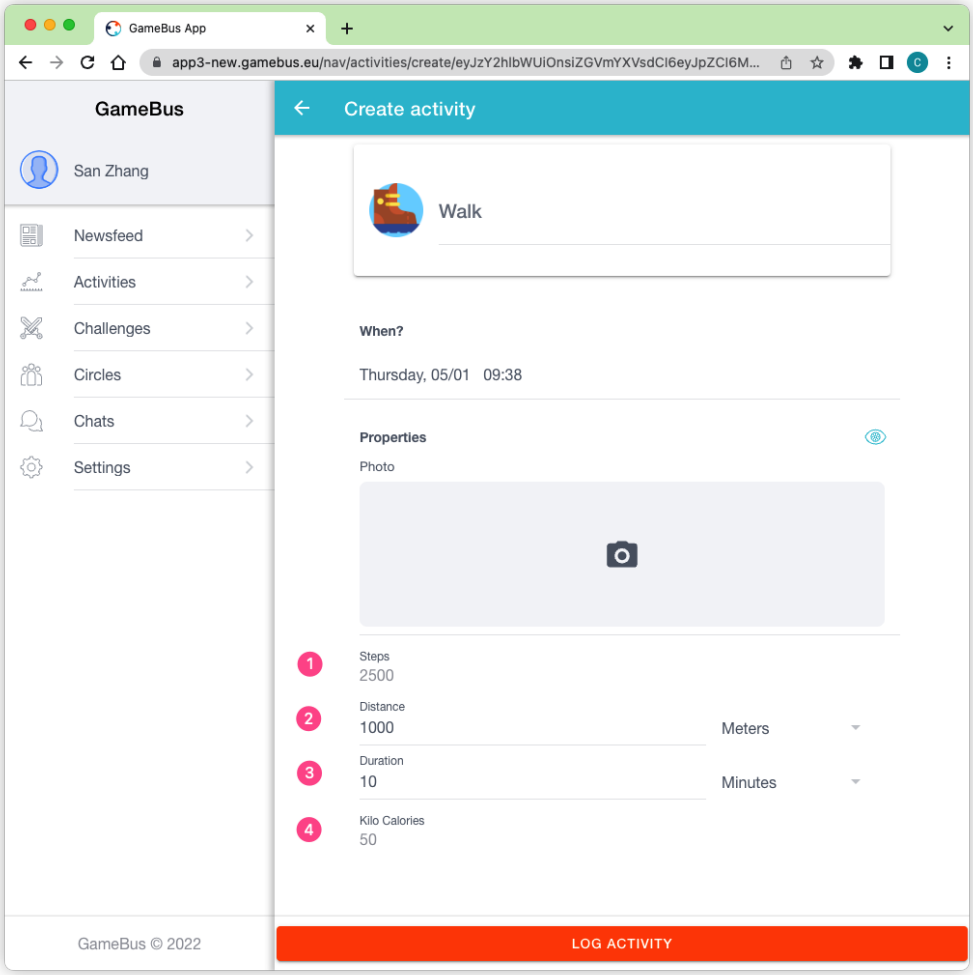
1. Click sidebar *Activities* and then click the plus button



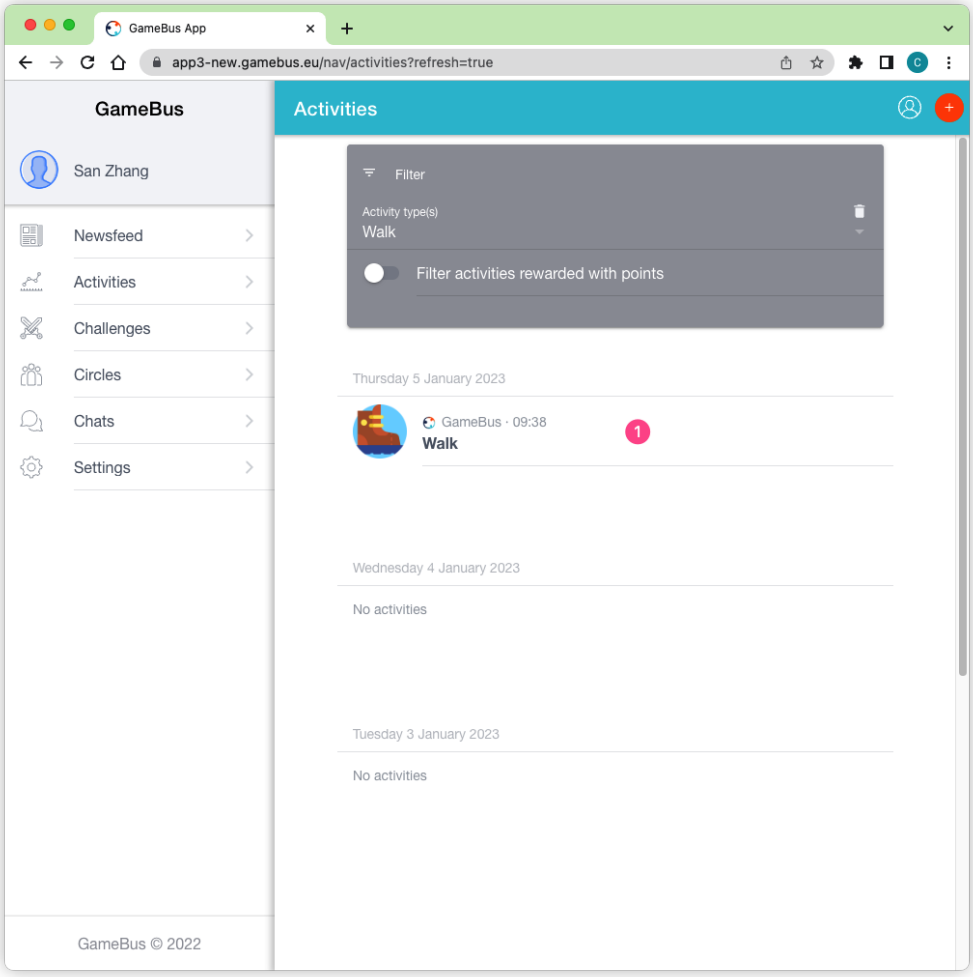
2. Click *ALL ACTIVITIES* and then choose *Walk* activity



3. Add walk data manually. For example, here we add 2500 steps, 1000 meters, 10 minutes and 50 Kcal. After filling, click *LOG ACTIVITY* to complete the filling.

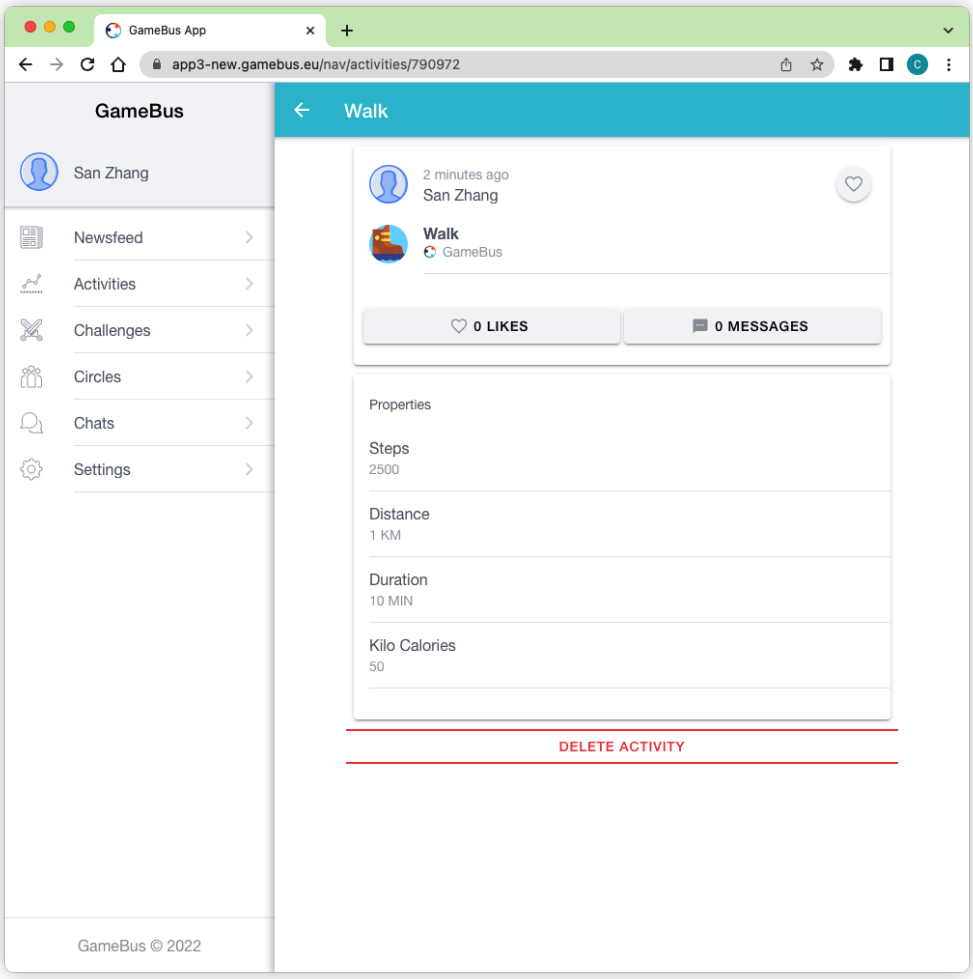


1. Now you can see the walk activity in the *Activities* page, and click it to view details.



Get activity id

We can get the id of the walk activity from url, the id here is `790972`. Check your url for the id, and it will be used in next section when sending request to FHIR server.



Add more data

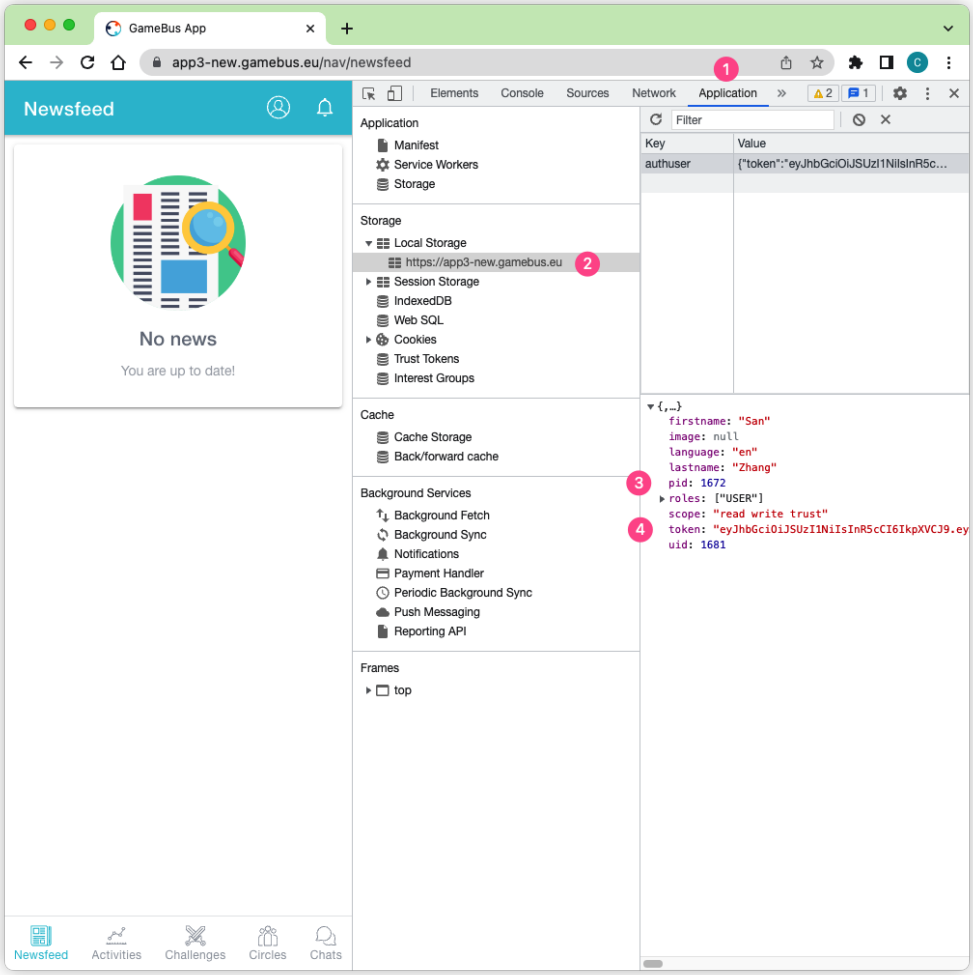
Likewise, you can add more data for other activities, e.g. run, bike, BP measurement.

Also, remember to take a note of the id of each activity, you'll need them in next section.

Get GameBus token and player id

Chrome browser is required in this part since Chrome DevTools will be used.

1. Login GameBus web app (https://app3-new.gamebus.eu) using Chrome
2. Open Chrome DevTools by pressing key `F12`. See the guide on how to open Chrome DevTools in different ways.
3. Click the `Application` panel and then the `Local Storage`, you will see player id `pid` and token `token` on the right region. The player id is the id for your GameBus account.
4. Take a note of the player id and token, they will be used in next section.



1.1.4 Request on FHIR API

After *starting FHIR server* and *adding data to GameBus platform* in previous sections, now it's ready to try the service of GameBus FHIR layer.

The FHIR layer is a service layer on top of the GameBus platform, providing the FHIR REST API for communicating FHIR-compliant data with the outside. For example, when a user sends an HTTP GET request to FHIR REST API, the FHIR layer will transform this request and forward the transformed request to GameBus's API; GameBus will process the request, e.g. read the requested data from its database, and then return the relevant data in GameBus format to the FHIR layer; the FHIR layer will then transform the GameBus data to data in FHIR-compliant format, and then return the FHIR data to user as a HTTP response. That is how the FHIR layer works and how it enables GameBus to provide FHIR service.

To try and test the FHIR API, we need API clients to help send HTTP requests. Various API clients exist, e.g. httpie, Postman, Hoppscotch and Insomnia.

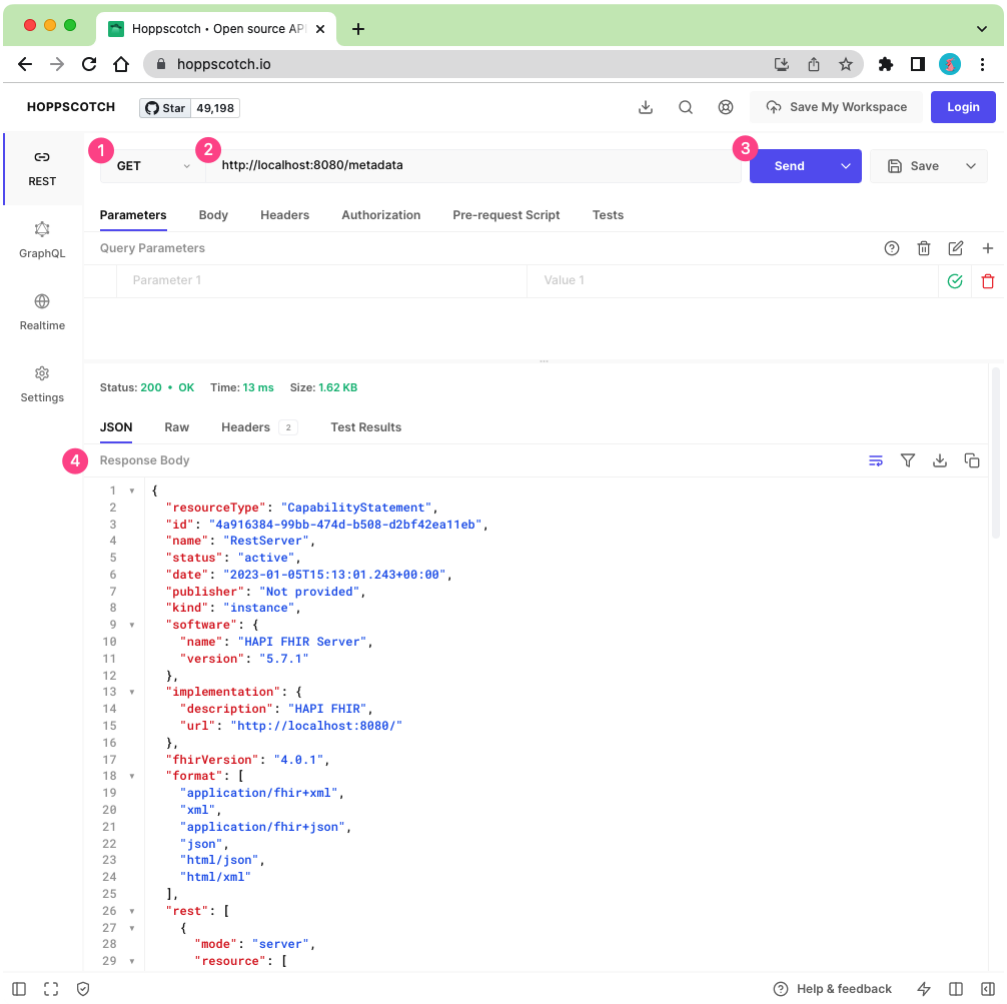
In this tutorial, we will use Hoppscotch. It's a web-based API client. Open its website(https://hoppscotch.io/) in a browser, then you can start sending HTTP requests.

Check the capability of FHIR server

We can check the capability of the FHIR server by sending a request to `[base]/metadata` API. We started the server of FHIR layer locally, so `[base]` here is `localhost:8080`.

Open Hoppscotch website (https://hoppscotch.io/), check the screenshot below, and do the following steps:

1. Use HTTP method `GET`
2. Fill URL box with `http://localhost:8080/metadata`
3. Click button `Send`
4. You should see the response of FHIR server.



The response is FHIR CapabilityStatement resource, which describes the available FHIR resources, operations, and functionalities on this FHIR server.

Check the resource body to see what FHIR resources and interactions are supported by the current FHIR layer.

Read FHIR resources

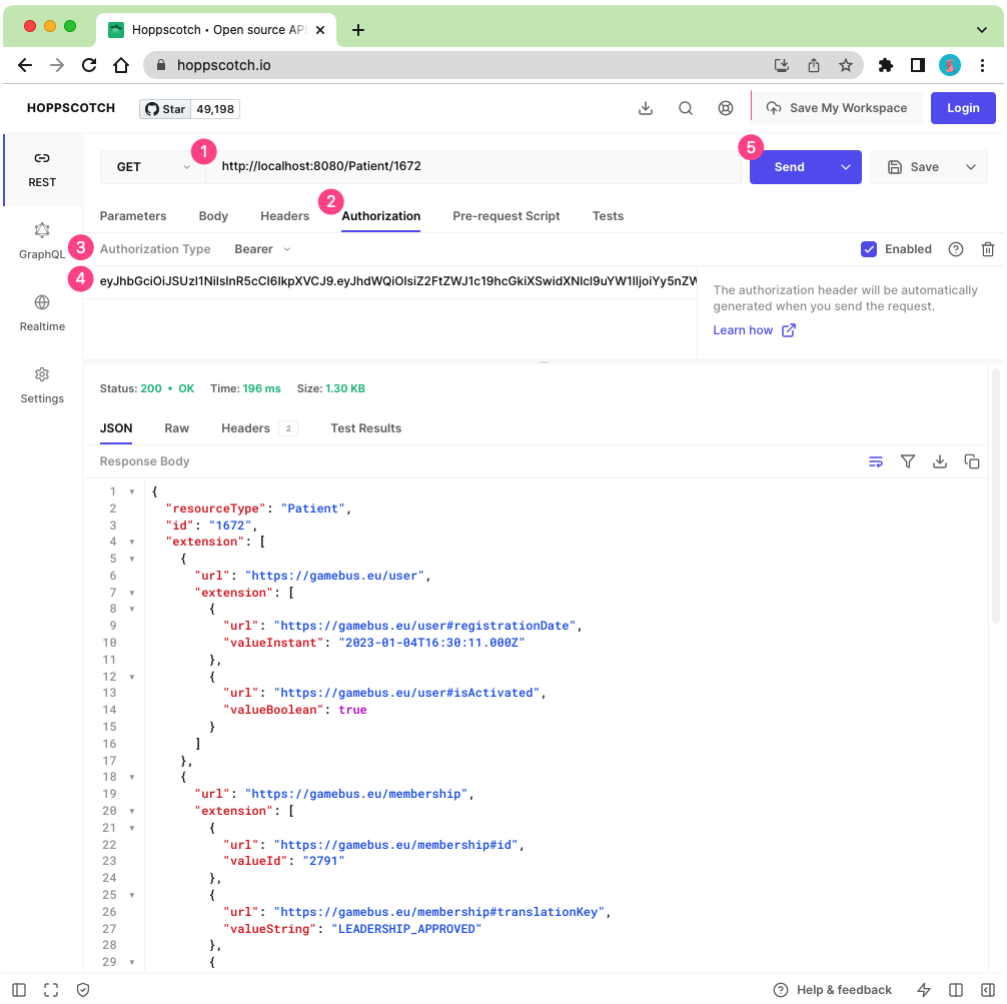
This part will show how to read FHIR Patient and Observation resources. GameBus token, player id, and activity id will be required to send HTTP requests, check *GameBus section* to get them.

Read FHIR Patient resource

FHIR Patient resource describes the profile of the patient. FHIR layer maps the user or player profile of GameBus to the FHIR Patient data.

Check the screenshot below and follow the steps:

1. Fill URL box with `http://localhost:8080/Patient/[pid]`. You need to replace `[pid]` with your player id.
2. Click *Authorization* button
3. Select authorization type `Bearer`
4. Fill in the token box with your GameBus token
5. Send the request



The response is a FHIR Patient resource, which is transformed from the player data of GameBus by the FHIR server.

Check the detail of the response body to see if the information is consistent with what you provided to GameBus when creating an account, e.g. first name, last name and email.

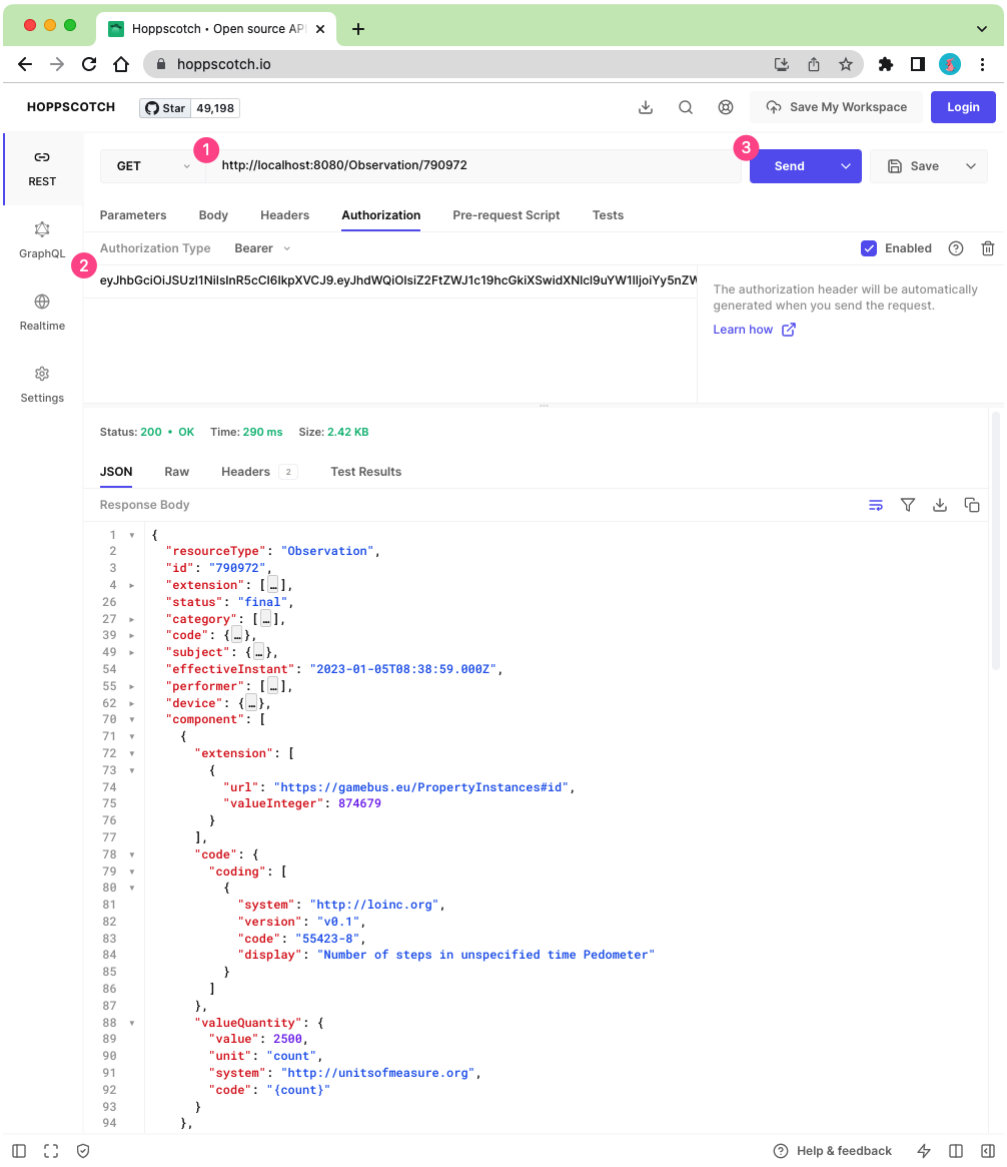
Read FHIR Observation resource

Some activity data (e.g. walk) were added to GameBus in the previous section. These activities will be mapped to FHIR Observation resource by the FHIR layer.

To request the FHIR Observation data, the activity id is required, e.g. the id of walk activity. Check *previous section* to get it.

Check the screenshot below and follow the steps:

1. Fill URL box with `http://localhost:8080/Observation/[activity_id]`. You need to replace `[activity id]` with GameBus activity id, e.g. walk activity id is `790972`.
2. Fill in Bearer token if it's empty
3. Send the request



The response is a FHIR Observation resource. Check the detail of the response body to see if the information is consistent with the activity data you added to GameBus.

Search FHIR Observation resources

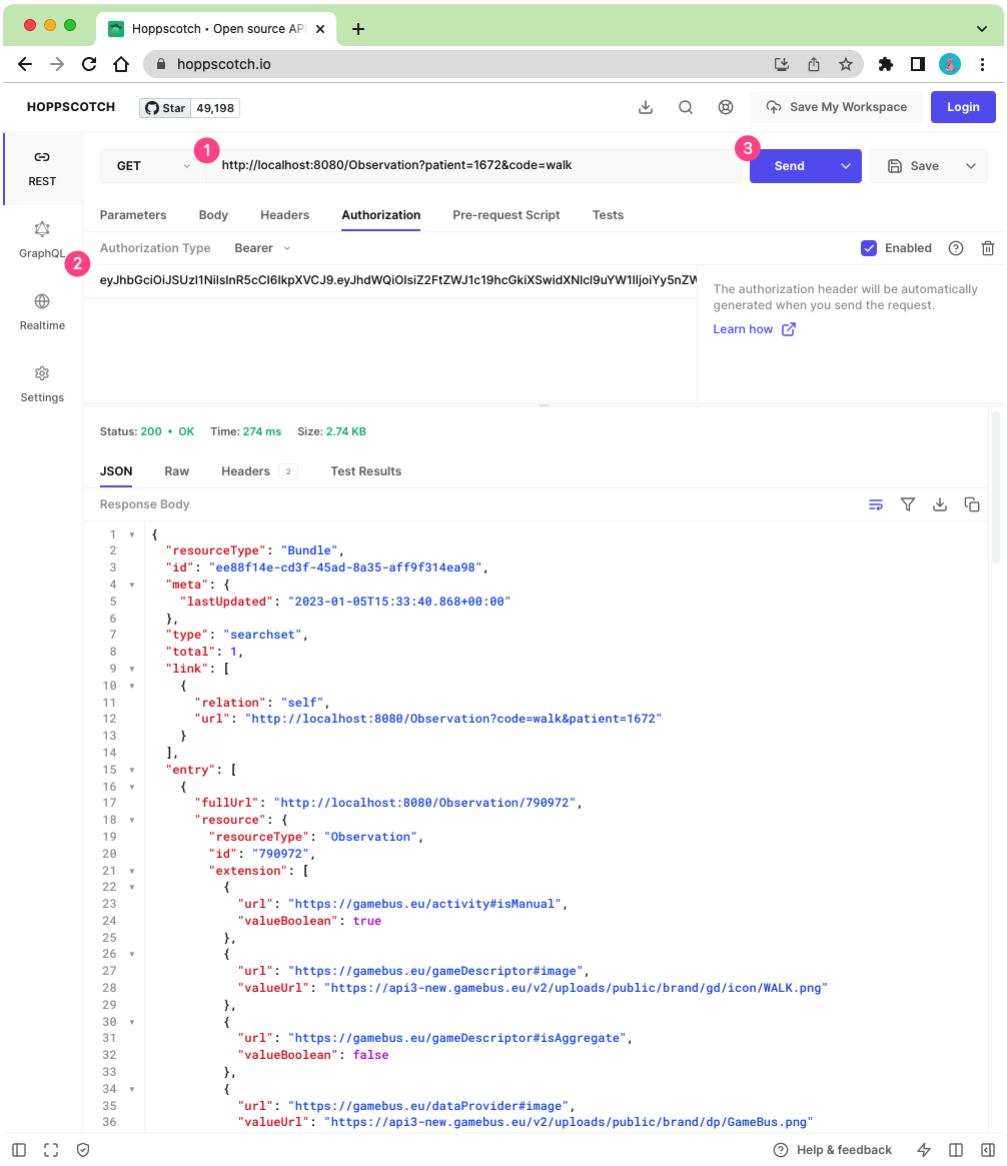
Besides requesting FHIR observations with a specific id, FHIR layer also supports searching based on e.g. observation type and/or date. To get the full list of supported search parameters, you can check the `CapabilityStatement` data in the *section above*.

Search specific types of observations

As an example, here we'd like to search all observations related to walk activity.

Check the screenshot below and follow the steps:

1. Fill URL box with `http://localhost:8080/Observation?patient=[pid]&code=walk`. You need to replace `[pid]` with GameBus player id.
2. Fill in Bearer token if it's empty
3. Send the request



The response is FHIR Bundle resource, it's a bundle of requested walk activities (FHIR Observation data) with full URL to each observation.

Try other search parameters

Observation supports other search parameters besides `code`.

Here are some examples you could try:

Request URL	Comment
http://localhost:8080/Observation?patient={[}pid{]}&code=walk,run,bike	search all observations related to walk, run and bike activities
http://localhost:8080/Observation?patient={[}pid{]}&date=gt2022-12-01	search all observations created after 1st December, 2022
http://localhost:8080/Observation?patient={[}pid{]}&code=walk&date=gt2022-12-01	search all *walk* observations created after 1st December, 2022
http://localhost:8080/Observation?patient={[}pid{]}&code=walk&_format=json	search all walk observations and set response format to json
http://localhost:8080/Observation?patient={[}pid{]}&code=walk&_sort=date	search all walk observations that are sorted by date
http://localhost:8080/Observation?patient={[}pid{]}&code=walk&_elements=code,subject.reference	search all walk observations and return only "code" and "subject.reference" parts of the Observation resource

Note that the search parameter `patient` is always required to specify which patient (GameBus player) to query. When changing the patient (i.e. player id), you also need to change the token to the one associated with that patient (GameBus player).

Useful links

Here is a cheat sheet for FHIR REST APIs.

For a detailed explanation of all FHIR APIs and search parameters, please check FHIR specification.

CHAPTER

TWO

GUIDE FOR DEVELOPMENT

2.1 Guide for developers

2.1.1 Introduction to the architecture

GameBus FHIR layer is built on a tech stack of open-source softwares (see diagram below), comprising two major components:

- **Mapping engine**

  Mapping engine is a component to convert data from one format to another, e.g. from GameBus JSON data to FHIR JSON data.

  Google HCLS Data Harmonization is used as the mapping engine of GameBus FHIR layer. The engine supports transformation between any two formats or schemas by configuring mapping rules. The mapping rules can be configured using protobuf format or Whistle Data Transformation Language that will be automatically transpiled to protobuf. Because of that, we use **Google Whistle** or **GW** to refer to this mapping engine.

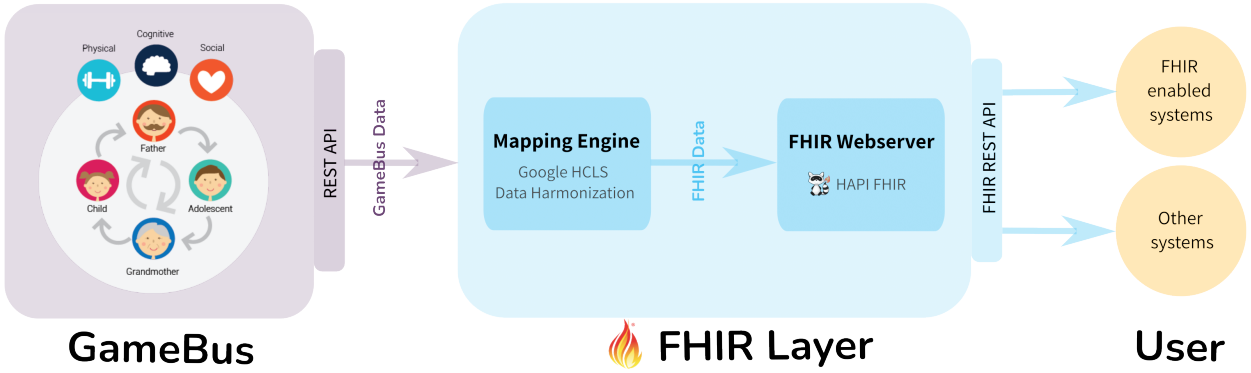
- **FHIR web server**

  FHIR web server is the server to provide the capabilities FHIR REST API.

  HAPI FHIR framework is used to add these capabilities to GameBus FHIR layer. With this framework, a HAPI FHIR plain server was created and some resource providers were defined to serve up FHIR resources, e.g. Patient and Observation.

  The FHIR server has two functionalities in GameBus FHIR layer:

  - It forwards the user's HTTP request to GameBus REST API after transforming the request for FHIR REST API to the request for GameBus API.
  - It gets FHIR-compliant data from mapping engine and sends these data to the user as HTTP response through FHIR REST API.



One of the advantages of this tech stack is that it does not change any code or schema of GameBus platform, but just add one more layer on the existing platform to add the capabilities of FHIR REST API. Moreover, though this FHIR layer is developed for GameBus platform, the tech stack can be easily applied to other healthcare platforms to enable FHIR service.

2.1.2 Build mapping engine

This section will show you how to build Google Whistle (GW) mapping engine (C shared library) for GameBus FHIR layer.

Build mapping engine shared library

Step 1, install the following dependencies

1. Golang (>=1.7)
2. Java JDK (>= 8)
3. Protobuf Compiler protoc (>= 3.11.4)
4. Clang (>= 11.0.1-2)

Step 2, download adapted Google Whistle codebase.
See *subsection below* for more info about the codebase.

```
git clone https://github.com/nwo-strap/healthcare-data-harmonization
cd healthcare-data-harmonization/mapping_engine
```

Step 3, build C shared library

```
./build_exports.sh
```

This script will generate C shared library `libgoogle_whistle.so` (for Linux) or `libgoogle_whistle.dylib` (for macOS) and create a corresponding symbolic link in the path `/usr/local/lib`. The GameBus FHIR layer will seek the mapping engine library in this path.

Google Whistle codebase

Compared with original Google Whistle codebase, the codebase used above is a forked codebase and updated with three new scripts:

- **mapping_engine/build_exports.sh**
  It's a helper BASH script to generate Go code from protobuf files and then compile Go code (e.g. export functions) to C shared library.
- **mapping_engine/main/exports.go**
  The export function `RunMapping` is defined in this Go file, which converts JSON string of one structure to another.

  This Go file can be updated to add other export functions.
- Dockerfile The dockerfile to build a docker image of the mapping engine.

2.1.3 Develop FHIR server

This section guides you on how to set up the development environment for GameBus FHIR layer and how to start a local test FHIR server.

Set up dev environment

Step 1, install the following dependencies

1. Java JDK (>=17)
2. Apache Maven (>=3.8)

Step 2, build mapping engine
Check *this section* to build Google Whistle mapping engine (C shared library), which is used by FHIR server to convert data.

Step 3, set mapping configs

Mapping configs are the rules used by Google Whistle mapping engine to convert data from one format to another. The repo of mapping configs has pre-defined rules for data conversion between GameBus and FHIR. It should be set properly to be used in GameBus FHIR layer, see the steps below:

First, clone mapping configs repo

```
git clone https://github.com/nwo-strap/mapping_configs.git
```

Let's assume the path of this clone is `MAPPING_CONFIG_PATH`, e.g. `/home/mapping_configs`.

Then, update all `local_path` variables in `gamebus_fhir_r4/configurations/*.textproto` files.

If the path of the cloned repo (your `MAPPING_CONFIG_PATH`) is `/mapping_configs`, you don't need to do anything; Otherwise, you MUST update all `local_path` with absolute path.

Step 4, clone source code

```
git clone https://github.com/nwo-strap/gamebus-fhir-layer.git
```

The `gamebus-fhir-layer` repo contains the implementation of the FHIR server by taking advantage of HAPI FHIR framework.

Resource for developing FHIR server

Now it's ready for further development of FHIR server based on HAPI FHIR framework, e.g. adding or changing FHIR resources or operations.

HAPI FHIR website has great documentation for developers to build an FHIR server. Also, you could check existing code in `gamebus-fhir-layer` repo to get a sense of how the HAPI FHIR framework works.

Start a local test server

Start a local FHIR server to test new functionalities:

```
# make sure you are working in the gamebus-fhir-layer repo
cd gamebus-fhir-layer

# Replace "[GAMEBUS_API_URL]" and "[mapping_configs_ABSOLUTE_PATH]" with real␣
↪values
mvn -D="jna.library.path=/usr/local/lib" \
    -Dgb.url="[GAMEBUS_API_URL]" \
    -Dgwc.player="[mapping_configs_ABSOLUTE_PATH]/gamebus_fhir_r4/
↪configurations/player.textproto" \
    -Dgwc.activity="[mapping_configs_ABSOLUTE_PATH]/gamebus_fhir_r4/
↪configurations/activity.textproto" \
    jetty:run
```

- `-D="jna.library.path=/usr/local/lib"` sets the path of mapping engine shared library
- `-Dgb.url="[GAMEBUS_API_URL]` sets the URL of GameBus REST API, which is https://api3-new.gamebus.eu/v2.
- `-Dgwc.player="[mapping_configs_ABSOLUTE_PATH]/gamebus_fhir_r4/configurations/player.textproto"` sets the mapping config for GameBus player data. Replace the `[mapping_configs_ABSOLUTE_PATH]` with the real value of the absolute path of the mapping configs repo (see step 3 above).
- `-Dgwc.activity="[mapping_configs_ABSOLUTE_PATH]/gamebus_fhir_r4/configurations/activity.textproto"` sets the mapping config for GameBus activity data.

By default, the server will be served at the base "http://localhost:8080".

To test the FHIR server, it's required to add test data to GameBus, check *the tutorial* to see how to add data to GameBus.

Then you can send HTTP requests to test the new functionalities of the FHIR server. See this tutorial about *how to request on FHIR REST API*.

2.1.4 Develop mapping configs

The `mapping_configs` repo contains the mapping rules for data conversion between GameBus and FHIR (v4). These mapping rules are coded with a domain specific language, i.e. Google HCLS Whistle Data Transformation Language.

This section will show you how to set up the development environment, how to run mapping using Google Whistle mapping engine (which is compiled as an executable but not a shared library), and how to validate mapping results.

Note that the development of mapping configs here is independent of FHIR server, you don't need to run an FHIR server.

Set up dev environment

Step 1. Build mapping engine

This part is similar to the section "*Build mapping engine*", but it is not going to build a C shared library. Instead, we just need to build an executable from the Go source code.

- First, install the following dependencies
  1. Golang (>=1.7)
  2. Java JDK (>= 8)
  3. Protobuf Compiler protoc (>= 3.11.4)
  4. Clang (>= 11.0.1-2)
- Second, clone mapping engine codebase

```
git clone https://github.com/nwo-strap/healthcare-data-harmonization
```

- Lastly, build it

```
cd healthcare-data-harmonization
./build_all.sh
```

The executable `healthcare-data-harmonization/mapping_engine/main/main` will be generated after building. Check if it exists.

Step 2. Clone mapping_configs repo

```
git clone https://github.com/nwo-strap/mapping_configs.git
```

Let's call the path of this clone repo `MAPPING_CONFIG_PATH`, e.g. `/home/mapping_configs`.

Now the environment is ready. You can then work with the mapping rules in the local repo.

Resources for development

To work with mapping rules, you need to be familiar with the Google Whistle language. Here are some resources for you to get started:

- Mini guide on language basics
- Language reference
- Builtin functions

Run mapping and validate mapping results

In this part, we take the `gamebus_fhir_r4` mapping rules in the repo as an example. To run and validate the mapping rules, you need to do the following:

- Update all `local_path` variables in `mapping_configs/gamebus_fhir_r4/configurations/*.textproto` files.

  If the path of the cloned repo (your `MAPPING_CONFIG_PATH`) is `/mapping_configs`, you don't need to do anything; Otherwise, you MUST update all `local_path` with absolute path.

- Run mapping

  Try the following command to convert GameBus player data to FHIR Patient:

  ```
  cd mapping_configs

  [BASE_PATH]/healthcare-data-harmonization/mapping_engine/main/main \
      -data_harmonization_config_file_spec=./gamebus_fhir_r4/configurations/
  ↪player.textproto \
      -input_file_spec=./gamebus_fhir_r4/example/gb_player.json
      -output_dir=.
  ```

  - The `[BASE_PATH]` is the path to the mapping engine repo, replace it with real value.
  - `-data_harmonization_config_file_spec` sets which mapping config file to use. Take a look at all config files and try others for different types of input.
  - `-input_file_spec` sets the path to the input JSON file.
  - `-output_dir` sets the path to the output directory,

  The output file is `./gb_player.output.json`, which is named based on the name of the input file.

  Try other mapping configs and input files, you can find reference output files in the folder `gamebus_fhir_r4/example/output`.

- Validate mapping results

  To make sure the mapping output conforms to FHIR specification, the fhir-validator-app or its free service can help you validate the results. It is just needed to paste the content of the mapping output to the app or service.

2.1.5 Build docker image

This section shows how to build a docker image of GameBus FHIR layer from three repos of healthcare-data-harmonization (mapping engine), mapping_config and gamebus-fhir-layer (FHIR server).

Requirement

Install the following software

- docker (20.10.14)

After installation, check the version of `buildx` by running

```
docker buildx version
```

Make sure it has a version >=0.8.

Using remote code

We can build a docker image of GameBus FHIR layer from the remote GitHub repos:

- Google Whistle mapping engine https://github.com/nwo-strap/healthcare-data-harmonization
- Mapping configs https://github.com/nwo-strap/mapping_configs
- FHIR server https://github.com/nwo-strap/gamebus-fhir-layer

The last repo contains the Dockerfile used to build the docker image. So first we need to clone this repo

```
git clone https://github.com/nwo-strap/gamebus-fhir-layer.git

# make sure you work in the "gamebus-fhir-layer" repo
cd gamebus-fhir-layer
```

Then, build the docker image

```
docker buildx build --no-cache=true -t gamebus-fhir-layer .
```

This command will build a docker image with the name `gambus-fhir-layer`.

By default, the source code from the latest commit of `main` or `master` branch of each repo will be used to build the docker image.

To use the source code of different versions, you could provide a branch name, commit , or tag name to the `buildx` command:

```
docker buildx build --no-cache=true -t gamebus-fhir-layer \
    --build-arg GW_VERSION=[gitBranch_orCommit_orTag] \
    --build-arg GW_CONFIG_VERSION=[gitBranch_orCommit_orTag] \
    --build-arg GAMEBUS_FHIR_VERSION=[gitBranch_orCommit_orTag] \
    .
```

Replace the `[gitBranch_orCommit_orTag]` with real values.

Using local code

It's also possible to build a docker image directly from local repos. It helps a lot to test new code before pushing them to remote.

First, make sure the three repos locate in the same place. You can clone them if needed

```
git clone https://github.com/nwo-strap/healthcare-data-harmonization
git clone https://github.com/nwo-strap/mapping_configs
git clone https://github.com/nwo-strap/gamebus-fhir-layer
```

Then, build the docker image

```
# make sure you work in the "gamebus-fhir-layer" repo
cd gambus-fhir-layer

# build docker image
docker buildx build --no-cache=true -t gamebus-fhir-layer \
    --build-context gw-src=../healthcare-data-harmonization \
    --build-context gw-config-src=../mapping_configs \
```

(continues on next page)

(continued from previous page)

```
    --build-context gamebus-fhir-src=. \
    .
```

The `--build-context` argument sets which local repo to use. You need to set an absolute path to this argument if your repos are not in the same place.