**CSIRO Project**

**Author: Ayelén Stefanía Andrade**

**Description**

The project was built using Java 11, Spring framework and microservice architecture pattern. The java application works as an Adapter, and is capable of receiving documents in two different formats, established as the examples given in the task. One of the document is a JSON file, and the other is a CSV file.

The software application is able to convert each of the documents’ format to FHIR validated JSON format.

The chosen FHIR format is a Diagnostic Report, as it includes most of the attributes of the original files: patient ID, date issued, test type (specimen) and final diagnosis. Differential diagnosis was not included, as those reports should be addressed in independent reports.

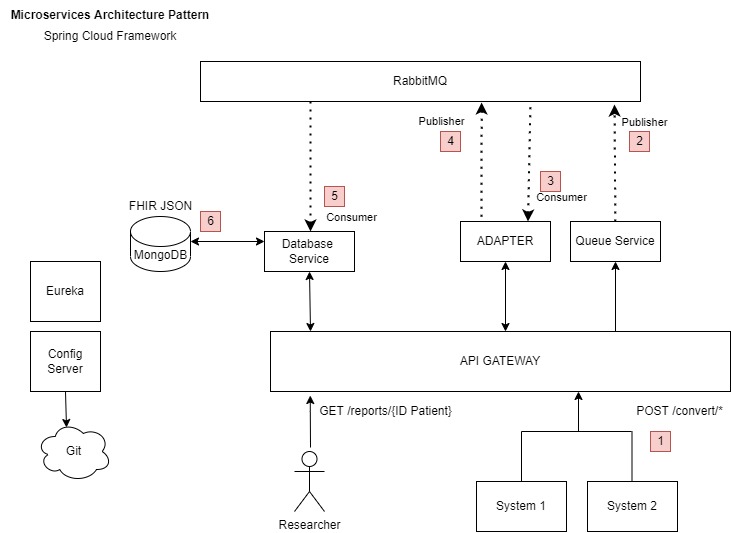
API methods and descriptions are detailed in this table:

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /reports/converter/json | Recieves a JSON file sent in the body and returns a response with FHIR JSON format of the file. |
| POST | /reports/converter/csv | Recieves a CSV file sent in the body and returns a response with FHIR JSON format of the file. |

Documentation of API requests and examples is available in: <https://documenter.getpostman.com/view/20744743/Uz5AreLx> .

The project can be deployed with docker compose using the next command in the root directory (“../ResearcherData/”):

docker compose up -d

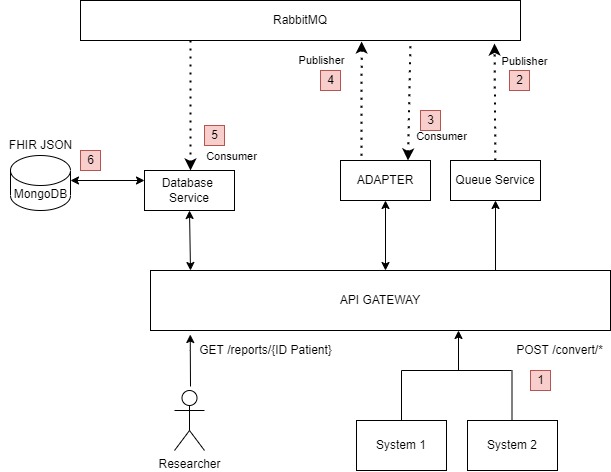
**Architecture Diagram**

The architecture designed follows the **Microservice** pattern. **Spring Cloud framework** can be implemented**,**with a **Config Server** to store and serve distributed configurations across multiple applications and environments.  It uses a repository layer that supports VCS (Git) or local storage.

**Netflix Eureka** would be used a registration and discovery service. Each service will be registered with the Eureka server and provide metadata such as host and port, health indicator URL, home page, etc. Eureka receives heartbeat messages from each instance belonging to a service and registering the service within the server.

**Spring Cloud Gateway** will be used to build an API to help simplify the communication between a client and a service. The API will sit between a requester and a resource that’s being requested, where it intercepts, analyzes, and modifies every request. Spring Cloud Security can be implemented in this API to control which role each route requires a user to have. Spring Boot Actuator can also be implemented, which would allow to easily emit metrics of the application performance.

To explain the architecture in the next picture, there is a series of events that are enumerated.



1. The systems 1 and 2 produce JSON or CSV files respectively, and each file can be sent in the body of a POST request to the Gateway API, which would redirect it to the Queue Service.
2. The Queue Service will receive the files and set them up in a queue via a message broker (RabbitMQ).
3. The Adapter Service will act as a consumer of this queue, and each request will be processed asynchronously. Each file will be converted to a validated FHIR Diagnostic Report in JSON format. If an error came up during the conversion of the documents, they would be assessed correctly with the corresponding HTTP response.
4. The adapter will generate FHIR JSON files and also set them in a different queue in Rabbit MQ.
5. The database microservice will act as a Consumer and receive those files from the respective queue.
6. The files will be then saved in a mongoDB database.

Researchers will be able to query the data by sending a request to the Gateway API, which will redirect it to the Database Service to retrieve the data needed. For example, a GET BY PATIENT ID request will be responded with a list of the resources depending on the patient id.

This architecture can be built with AWS, storing the backend with ECS (Elastic Container Service) to deploy, manage and scale containerized applications. Amazon DocumentDB can be used for MongoDB.