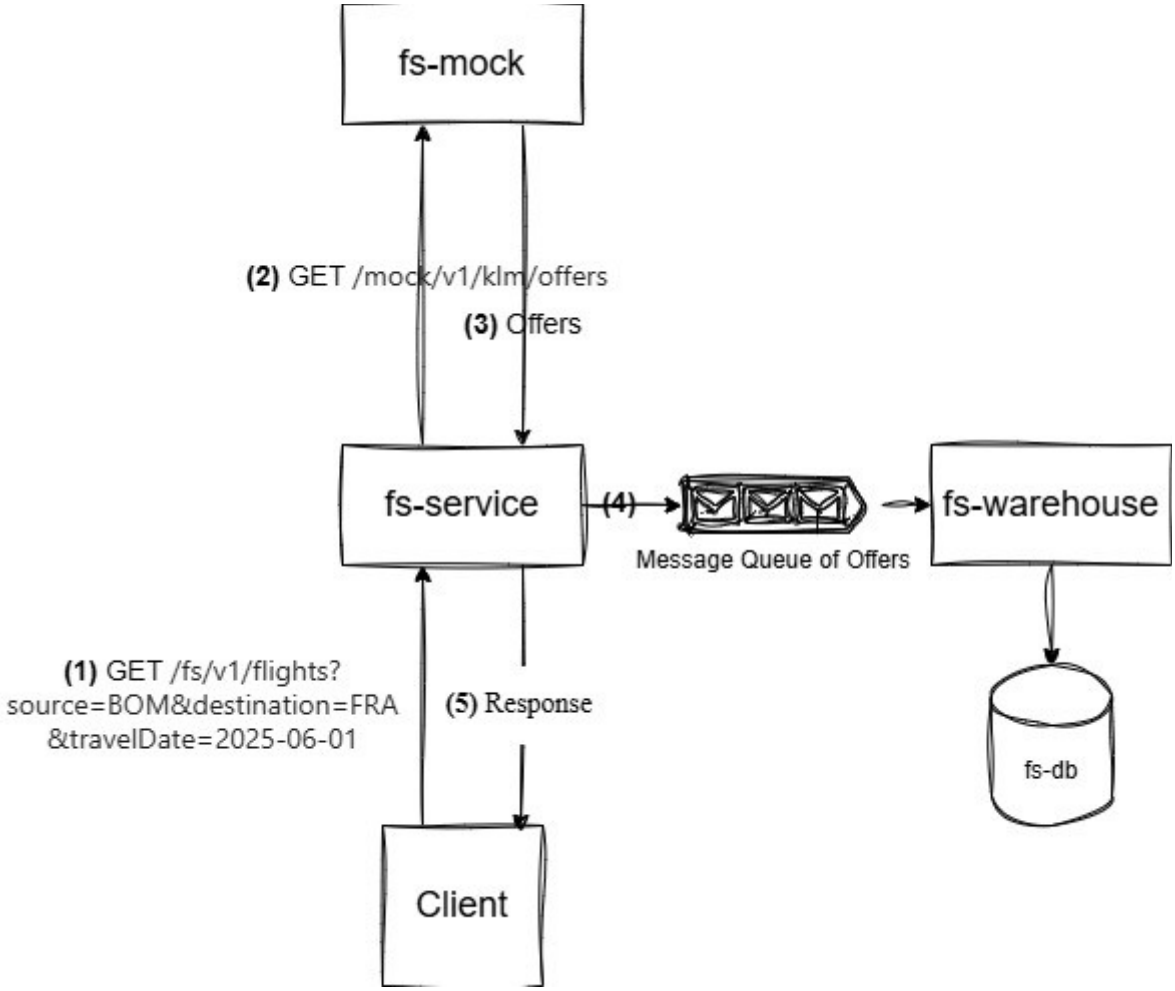
Motivation

This project is aimed at demonstrating skills related to Quarkus and Java EE. This is a small copy version of Google Flights. Like Google Flights, it tries to fetch prices real time and returns them to the user.

Planning

The project will have modules to mock the functionality of airlines to respond with prices (fs-mock), a main module which recieved request from client and responsible for querying airlines for prices and aggregating them (fs-service) and a module to store offer prices in a warehouse so that the data can be used for training AI models for dynamic price prediction (fs-warehouse). We used Postgres to persist those offers and RabbitMQ as the connecting component between fs-service and fs-warehouse. The common model classes shared by all modules are extracted in fs-models.

The overall architecture will look like this:



Implementation:

1. Design and add Swagger API documentation for the mock module. I got to learn about configurations and nitty gritty of it maven plugins in this step. Add some biolerplate code to the mock to respond to the REST requests.
2. Create a Swagger API and create aDocker image creation workflow for fs-service module. The native image building option provided by Quarkus is interesting when we want to do stress test this project.
3. Create and call REST clients for Airlines using @RegisterRestClient from Microprofile. This is a very good option and saves a lot of time writing a REST client from scratch.

4. Add support for caching responses from Airlines. This should help us to avoid calling Airlines for same flight details in short amount of times. This also helped me to learn about caching.
5. Add RabbitMQ as a message broker between fs-service and fs-warehouse. I decided not to use synchronous call as that adds unnecssary wait before returnining response to the client.
6. Add fs-warehouse module which will act as a data warehouse. Connect it to RabbitMQ and persist offers in the Postgres database. The Hibernate Panache's repository removes need of writing a DAO layer which makes life easy and makes the code more maintainable.

Next step:

I would like to add fs-ai module which has an AI model pretrained on offers datapoints in the warehouse. This model can be used to give price prediction for the flight details handled by fs-service. Additionally, it should provide explanation of its predition (explanianable AI).