HLA High Level Analysis

Triathlon Insurance Product

- **Executive Summary**

The Executive Summary **provides** a clear and concise overview of the technical solution and its purpose. In this case, it addresses a specific need in the insurance market, particularly in the realm of sports-related insurance products. Below is a detailed description of what should be included in this section:

- o **Purpose of the Document**

- **Objective:** The purpose of this document is to detail the technical solution developed for a new insurance product tailored for triathlon athletes. This product aims to fill a gap in the market by offering coverage not only for personal injuries but also for damages to high-value equipment, such as bicycles, wetsuits, and helmets, during competitions.

- **Target Audience:** The document is intended for business stakeholders, project managers, and technical teams who need to understand both the technical and business implications of the solution.

- o **Business Problem**

- **Market Gap:** Currently, there is a lack of insurance products that cater specifically to the needs of triathletes. While existing insurance policies may cover personal injuries, they often do not cover equipment damage, which can be substantial given the high cost of triathlon gear.

- **Challenges for Athletes:** For example, if an athlete suffers a bicycle accident during a competition, traditional insurance might cover medical expenses but not the repair or replacement of the bicycle, which can be extremely costly. This lack of coverage poses a significant financial risk to athletes.

  - **Proposed Solution Overview**

- **Solution Overview:** The solution involves developing a web application using Angular 18 for the frontend, supported by a backend architecture composed of multiple microservices. These microservices handle master data management, quote calculation, and weather data integration to assess risk factors.

- **Key Features:**

  - **Event and Equipment Selection:** Users can select the triathlon event they are participating in and the equipment they wish to insure, such as their bicycle, wetsuit, and helmet.

  - **Weather Integration:** The application integrates with a public weather service to obtain the probability of rain at the event location on the scheduled date. If the probability of rain exceeds 50%, the insurance premium is adjusted accordingly to reflect the higher risk of accidents.

- **Introduction**

  - **Background**

- **Context:** Triathlon is a growing sport with a dedicated audience, many of whom have a high disposable income and invest significantly in their equipment. However, the insurance market has not yet fully capitalized on this demographic by offering specialized products that meet their unique needs.

- **Need for Innovation:** This application addresses the need for a comprehensive insurance product that provides coverage beyond standard personal injury, extending to the protection of valuable sporting equipment.

  - **Objetives**

- **Primary Objective:** To develop a technically robust and user-friendly application that allows triathletes to insure both themselves and their equipment, with premiums that reflect the actual risk based on real-time weather data.

- **Secondary Objectives:**

  - Enhance the customer experience by providing a seamless and intuitive platform.

  - Integrate with existing public APIs to offer dynamic risk assessment and premium calculation.

  - Leverage cloud-based technologies for scalability and data management.

3. System Architecture Overview

- **3.1 Architecture Diagram**

  - Visual representation of the overall system architecture.
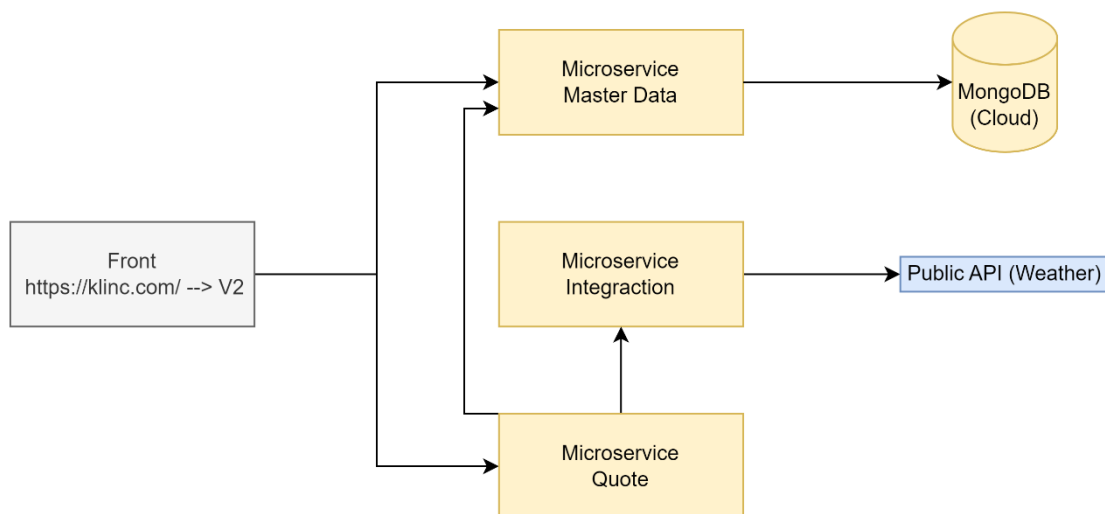
o Visual representation of the call sequence.
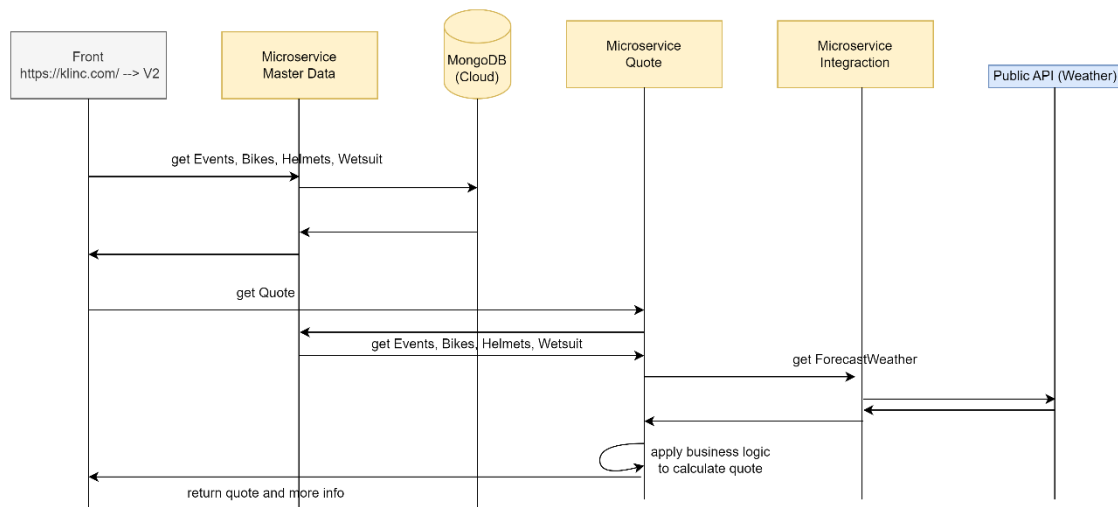
- **3.2 Component Overview**

  o Description of each component in the architecture:

    ▪ **Frontend Application (Angular 18):** User interface and user experience considerations.

    ▪ **Master Data Microservice:** Manages master data and retrieves information from MongoDB.

    ▪ **Quote Microservice:** Apply business logic to calculate the quote based on various inputs.

    ▪ **Integration Microservice:** Integrates with an external weather API to obtain real-time data for quotes.

4. Frontend (Angular 18)

- **4.1 Technology Stack**

  o Angular version 18.2

- **4.2 Component Structure**

  o Home component to represent home page

  o poliza-triatlon component to represent the page where we call master data microservice to load the combo box of Events, Bikes, Helmets,

Wetsuit, and the button "presupuesto" to call Quote Microservice to return the quote and more info to show in the page.

- **4.4 Communication with Backend**

    o The frontend communicates with the microservices via REST API calls).

        ▪ localhost:8080/event ← get all the events

        ▪ localhost:8080/bike ← get all the bikes

        ▪ localhost:8080/wetsuit ← get all the wetsuit

        ▪ localhost:8080/helmet ← get all the helmets

    o if we need the information about one event, we could call like this (id of the event) localhost:8080/event?id=1 or by name of the event.

5. Backend Microservices

Three microservices have been developed in Java (SpringBoot 3 with graalVM to improve the performance loading different components of the application).

- JVM 22.0.2
- Dependencies
    o Spring-boot-starter-web
    o Spring-boot-starter-actuator
    o Javax.annotation-api
    o Jakarta.validation-api
    o spring-boot-starter-data-mongodb
    o de.flapdoodle.embed.mongo.spring3x

- **5.1 Master Data Microservice**

    o **5.1.1 Functionality**

        ▪ Manages and retrieves master data required for insurance products. For this example, we only develop the following functionalities.

ZURICH

```
Ⓜ getBike(String, String, String):ResponseEntity<List<Bike>>
Ⓜ getEvent(String, String):ResponseEntity<List<Event>>
Ⓜ getHelmet(String, String, String):ResponseEntity<List<Helmet>>
Ⓜ getWetsuit(String, String, String):ResponseEntity<List<Wetsuit>>
```
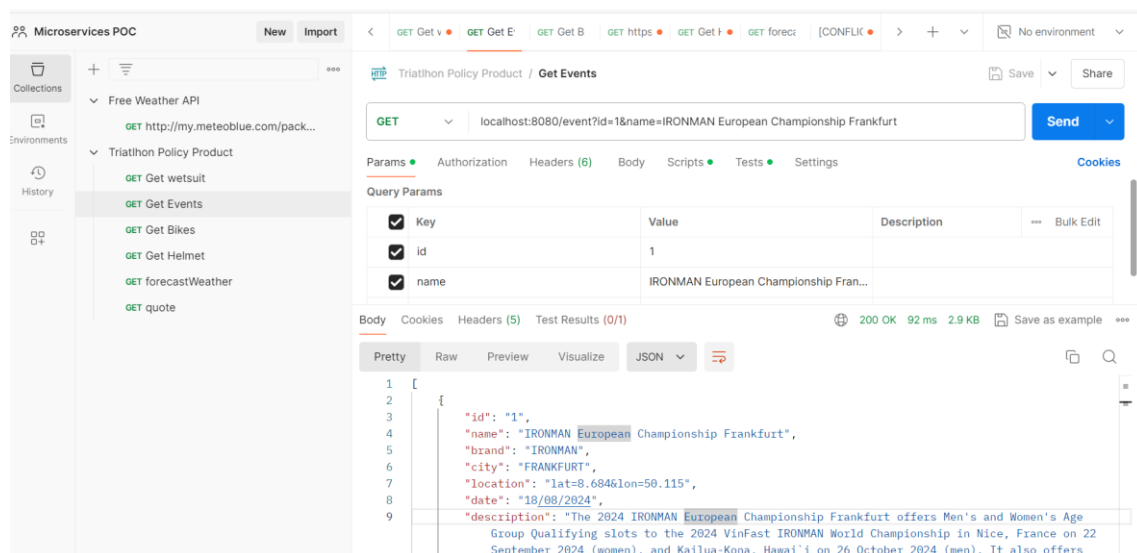
        ◦ **5.1.2 Database Integration**

As you can see in the project in GitHub, I've separated the layers in different folders (controller, service, repository) and I use **MongoTemplate** object to execute the queries with the criteria

        ◦ **5.1.3 API Endpoints**

Like an example here is the postman collection to test all the endpoints

- **5.2 Quote Calculation Microservice**

        ◦ **5.2.1 Functionality**

- Handles business logic for calculating quote. This business logic is **homemade**, because I don't have any specification of that.

  - **5.2.2 Workflow**

Microservice define only one endpoint /quote with some parameters

localhost:8082/quote?eventId=2&bikeId=3&wetsuitId=1&helmetId=1

the IDs of the master data information selected by the user. The business logic will call the master data microservice to get all the information of the ID 2 /event?id=2. Then we will evaluate if we have an Id of the bike. If we have that ID, we will call to the master data microservice to get all the information of the id 3 /bike?id=3. Then we know the market value of the bike, and we have the location of the event.

  - **5.2.3 Integration with Weather API**

    - I've developed the integration with the integration microservices with RestTempalte object sending the parameters related with the location of the event (latitude and longitude).

    - If the value returned by the service is more than a constant that I've defined (50%) I will multiply que bike quote with 1.5

    - The normal quote of a gear (bike, helmet or wetsuit) is 5% of the market value.

  - **5.2.4 API Endpoints**

Triatlhon Policy Product ^

GET	/quote

- **5.3 Integration Microservice**

  - **5.3.1 Functionality**

    - Retrieves weather data from an external public API. The public API is the following

https://docs.meteoblue.com/en/weather-apis/forecast-api/forecast-data

And to use this API you have to signup to the system, and then use an API-KEY to call this service. I've used RestTemplate to call to this service like this

This service will return a lot of information related with the weather, and I will use precipitation_probability field for the next 8 days.

```
"data_day": {
  "time": [
    "2024-08-19",
    "2024-08-20",
    "2024-08-21",
    "2024-08-22",
    "2024-08-23",
    "2024-08-24",
    "2024-08-25",
    "2024-08-26"
  ],
  "temperature_instant": [17.5, 18.66, 18.83, 14.67, 17.54, 19.55, 22.57, 16.07],
  "precipitation": [1.8, 0, 7.3, 0, 0, 0, 22.5, 0],
  "predictability": [62, 83, 58, 85, 82, 54, 16, 29],
  "temperature_max": [23.84, 25.82, 22.97, 26.26, 30.13, 34.06, 23.02, 22.63],
  "sealevelpressure_mean": [1016, 1014, 1018, 1015, 1013, 1012, 1020, 1021],
  "windspeed_mean": [1.7, 1.28, 1.96, 0.97, 1.7, 1.94, 1.82, 1.4],
  "precipitation_hours": [6, 0, 3, 0, 0, 0, 7, 0],
  "sealevelpressure_min": [1015, 1012, 1016, 1012, 1012, 1009, 1015, 1020],
  "pictocode": [16, 3, 7, 2, 2, 3, 8, 3],
  "snowfraction": [0, 0, 0, 0, 0, 0, 0, 0],
  "humiditygreater90_hours": [0, 0, 0, 0, 0.08, 0, 0, 0.21],
  "convective_precipitation": [0, 0, 0, 0, 0, 0, 22.5, 0],
  "relativehumidity_max": [96, 94, 84, 80, 87, 85, 91, 90],
  "temperature_min": [16.68, 15.06, 14.67, 12.69, 14.32, 16.23, 16.07, 14.8],
  "winddirection": [0, 270, 270, 90, 270, 135, 270, 0],
  "felttemperature_max": [24.21, 27.44, 22.29, 27.05, 30.1, 34.1, 23.07, 21.62],
  "indexto1hvalues_end": [23, 47, 71, 95, 119, 143, 167, 191],
  "relativehumidity_min": [55, 55, 45, 43, 34, 29, 56, 47],
  "felttemperature_mean": [20.47, 21.31, 18.36, 19.08, 21.87, 25.02, 20.38, 17.87],
  "windspeed_min": [0.92, 0.85, 1.09, 0.7, 1.18, 1.5, 1.04, 1.02],
  "felttemperature_min": [17.94, 15.66, 13.79, 12, 13.84, 16.36, 16.88, 14.23],
  "precipitation_probability": [0, 0, 19, 0, 0, 32, 74, 32],
```

I will use de maximum value of all the array, to know it the probability to rain is higher than the maximum value that I've defined.

- ○ **5.3.4 API Endpoints**

Triatlhon Policy Product ⌃

GET	/forecastWeather

6. Data Management

For this exercise, I have different options to manage the information about the master data. I needed some place to store the events, and different gear related with triathlon.

- Some static files (properties) with the information because this is a POC
- Use a Relational Database (like PostgreSQL)
- Use a non-Relational Database like MongoDB to store the info in Collections

Finally, I've decided to use MongoDB because it's a more realistic case than the first option and have more performance than the second option.

Other option that I thought was a PostgreSQL with a cache to get the data faster.

- **6.1 MongoDB Cloud Integration**

    - ○ For Details on the MongoDB cloud setup, including clusters, collections, and data access patterns.

        I've configured a cluster on Cloud (AWS Paris Region) to do this POC
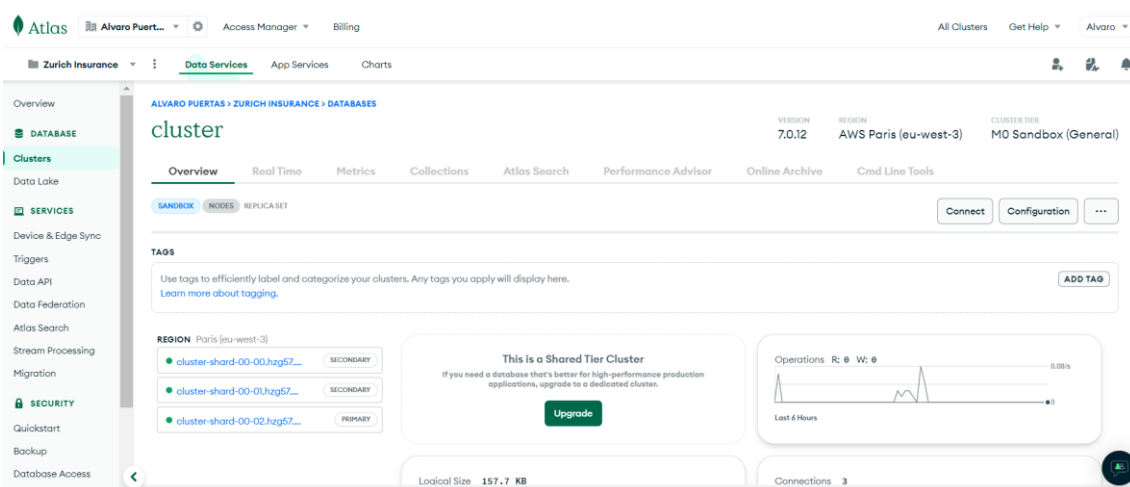
And I configure the following collections.

Here I describe the data that we have in each collection:

- Event Collection

```
_id: ObjectId('66c14e8c410985d51f883272')
name : "IRONMAN European Championship Frankfurt"
brand : "IRONMAN"
city : "FRANKFURT"
location : "lat=8.684&lon=50.115"
URLImage : "https://ironman.kleecks-cdn.com/cdn1/attachments/photo/7525-199685611/…"
description : "The 2024 IRONMAN European Championship Frankfurt offers Men's and Wome…"
date : "18/08/2024"
searchId : "1"
```

- Bike Collection

```
_id: ObjectId('66c17ccdf89a5b4d99958869')
searchId : "1"
brand : "TREK"
model : "Speed Concept SLR 9 AXS"
year : "2024"
marketValue : "12499.00"
currency : "EUR"
description : "La Speed Concept SLR 9 AXS es una bicicleta aerodinámica de triatlón h…"
URLImage : "https://media.trekbikes.com/image/upload/f_auto,fl_progressive:semi,q_…"
```

- Wetsuit Collection

```
_id: ObjectId('66c172adf89a5b4d99958866')
searchId : "1"
brand : "HUUB"
model : "Brownlee Agilis 4:4"
year : "2022"
marketValue : "799.00"
currency : "EUR"
description : "Huub ha estado trabajando con los hermanos Brownlee para crear el traj…"
URLImage : "https://img.fruugo.com/product/8/00/1391053008_max.jpg"
```

- Helmet Collection

```
_id: ObjectId('66c18558f89a5b4d9995886c')
searchId : "1"
brand : "RUDY PROJECT"
model : "The Wing"
year : "2024"
marketValue : "399.90"
currency : "EUR"
description : "The Wing revolutionizes the concept of aerodynamics in cycling helmets…"
URLImage : "https://www.rudyproject.com/sites/default/files/2024-02/rudyproject-th…"
```

For this POC I've stored 3 documents for each collection.

## 7. API Design and Documentation

- **7.1 RESTful API Principles**

  I've used API First to define the APIs to communicate frontend application with the data / integrations or business logic

  - The API is the first user interface of the application
  - I define the API first, then the implementation
  - The API is described (self-descriptive)
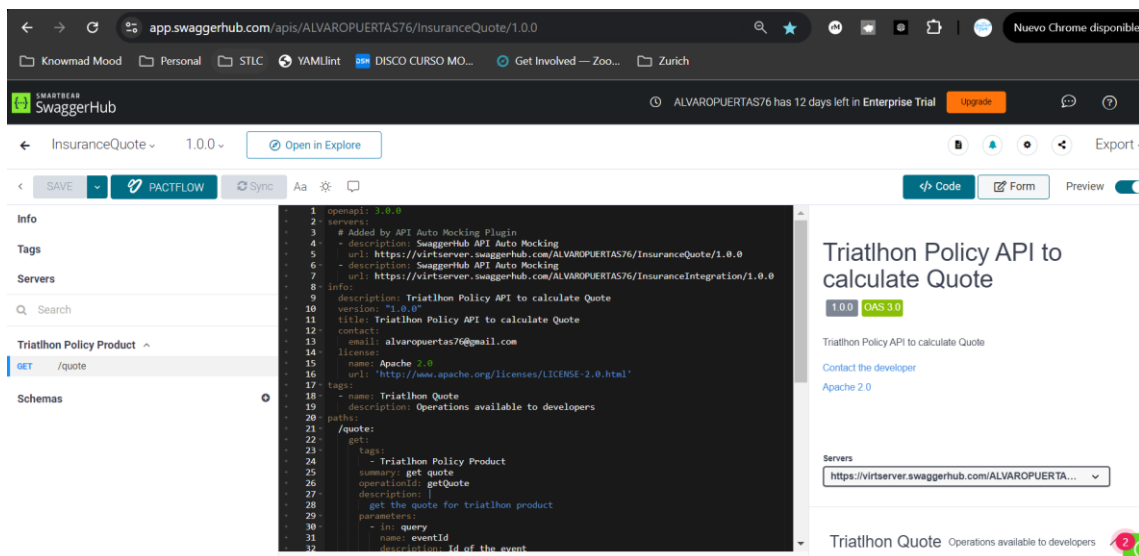
- **7.2 API Specifications**

  - Detailed documentation of all APIs, including request/response formats, status codes, and example calls.

    - Master Data API Definition

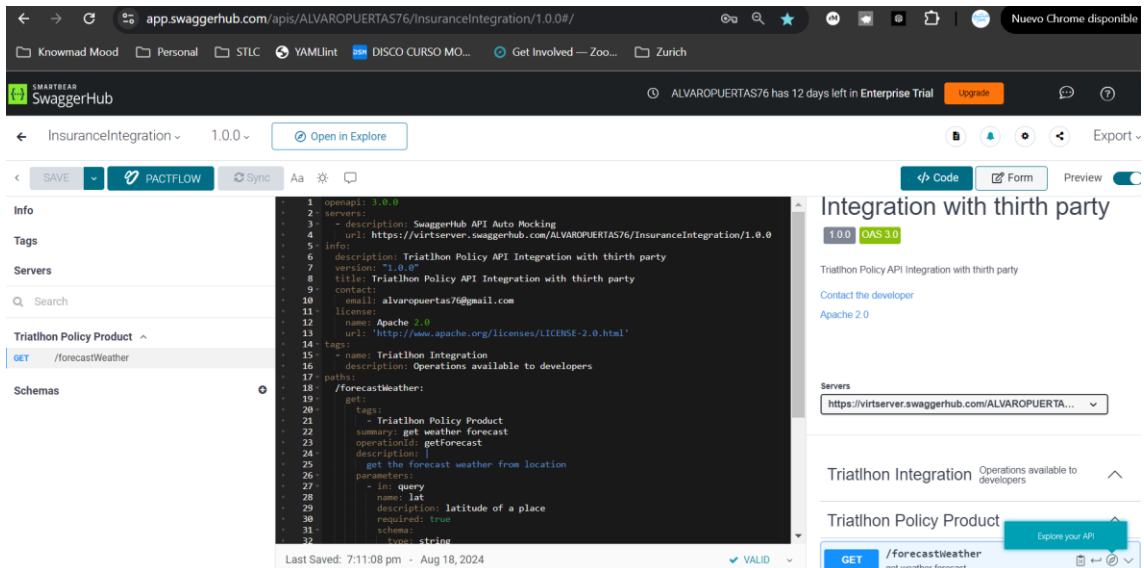You can see the yaml resolved definition in the following link : InsuranceMasterData-1.0.0.yaml
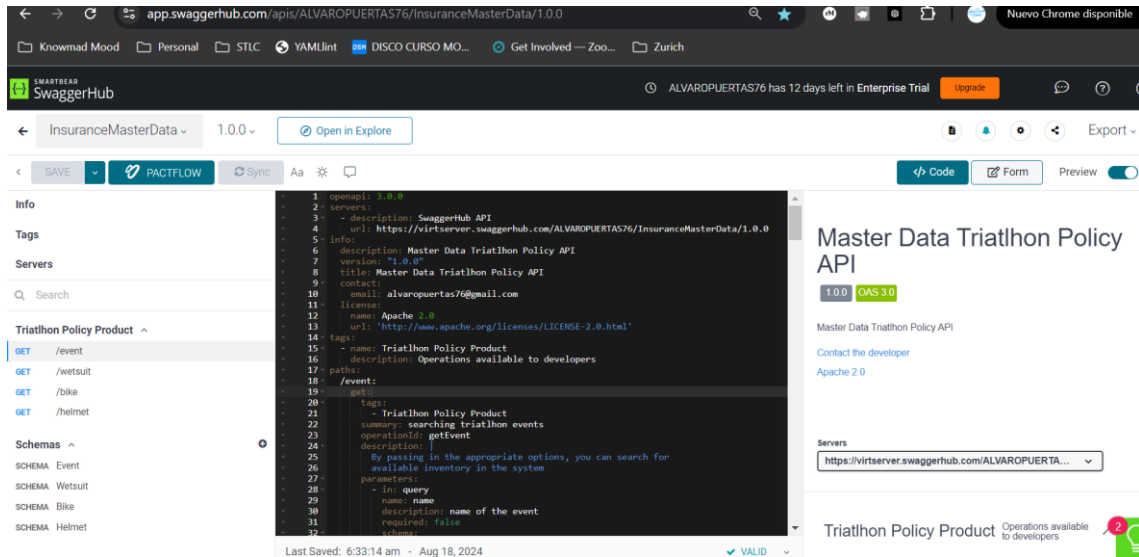
- Quote API definition

You can see the yaml resolved definition in the following link: InsuranceQuote-1.0.0.yaml

- Integration API Definition

You can see the yaml resolved definition in the following link: InsuranceIntegration-1.0.0.yaml

I have added Spring Actuator, which exposes the following endpoints to determine the health of the microservice, and we can view interesting metrics

```
{"_links":{"self":{"href":"http://localhost:8080/actuator","templated":false},"beans":
{"href":"http://localhost:8080/actuator/beans","templated":false},"caches":
{"href":"http://localhost:8080/actuator/caches","templated":false},"caches-cache":
{"href":"http://localhost:8080/actuator/caches/{cache}","templated":true},"health-path":
{"href":"http://localhost:8080/actuator/health/{*path}","templated":true},"health":
{"href":"http://localhost:8080/actuator/health","templated":false},"info":
{"href":"http://localhost:8080/actuator/info","templated":false},"conditions":
{"href":"http://localhost:8080/actuator/conditions","templated":false},"shutdown":
{"href":"http://localhost:8080/actuator/shutdown","templated":false},"configprops":
{"href":"http://localhost:8080/actuator/configprops","templated":false},"configprops-prefix":
{"href":"http://localhost:8080/actuator/configprops/{prefix}","templated":true},"env-toMatch":
{"href":"http://localhost:8080/actuator/env/{toMatch}","templated":true},"env":
{"href":"http://localhost:8080/actuator/env","templated":false},"loggers-name":
{"href":"http://localhost:8080/actuator/loggers/{name}","templated":true},"loggers":
{"href":"http://localhost:8080/actuator/loggers","templated":false},"heapdump":
{"href":"http://localhost:8080/actuator/heapdump","templated":false},"threaddump":
{"href":"http://localhost:8080/actuator/threaddump","templated":false},"metrics-requiredMetricName":
{"href":"http://localhost:8080/actuator/metrics/{requiredMetricName}","templated":true},"metrics":
{"href":"http://localhost:8080/actuator/metrics","templated":false},"sbom":
{"href":"http://localhost:8080/actuator/sbom","templated":false},"sbom-id":
{"href":"http://localhost:8080/actuator/sbom/{id}","templated":true},"scheduledtasks":
{"href":"http://localhost:8080/actuator/scheduledtasks","templated":false},"mappings":
{"href":"http://localhost:8080/actuator/mappings","templated":false}}}
```

8. Deployment and Scalability

- **8.1 Deployment Strategy**
  - For this exercise, frontend application and 3 microservices are deployed in localhost.
    - Frontend: localhost:4200/
    - MasterData: localhost:8080/XXXXX
    - Integration: localhost:8081/forecastWeather?lat=47.558&lon=7.573
    - Quote: localhost:8082/quote?eventId=2&bikeId=3&wetsuitId=1&helmetId=1

- I use a MongoDB database to store all the data related with master data and is deployed on Cloud (AWS).