Software Engineering

Specification

AeroTelcel

Revision 1.0

Table of Contents

Revision History

Version	Name	Reason For Changes	Date
0.1		System's Definition	
0.1.1	Javier Ortega	Requirement Elicitation	07/11/2023
0.1.2	Marcos Gonzalez	Requirement Prioritization	21/11/2023
0.1.3	Javier Ortega	System Description	14/11/2023
0.1.4	Jorge Ramirez	System Overview	15/11/2023
0.1.5	Marcos Gonzalez	UML Use Case Diagram	03/12/2023
0.2		System's Design	
0.2.1	Jorge Ramirez	System's Architectural Design	03/12/2023
0.2.2	Jorge Ramirez	UML Component Diagram	04/12/2023
0.2.3	Marcos Gonzales	UML Sequence Diagram	06/12/2023
0.2.4	Javier Ortega	Define Database Structure	06/12/2023
0.2.5	Javier Ortega	UML ER Diagram	07/12/2023

Approved By

Approvals should be obtained for project manager, and all developers working on the project.

Name	Department	Date
Jorge Ramirez de Diego	AeroTelcel Development	16/12/2023
Javier Ortega Mendoza	AeroTelcel Development	16/12
Marcos Gonzalez Fernandez	AeroTelcel Development	16/12/2023

1. Introduction

1.1 Purpose of the System

1.1.1 Introduction:

The “AeroTelcel” application aims to fulfill the requirement of a user-centered flight tracking system. Existing solutions, such as “flightradar24”, usually aim to satisfy the aviation enthusiast market, which, in our opinion, is an extremely niche market. That is why, by implementing a user-centered application, not only would it be able to provide another option to the enthusiast market, but it would also be able to cover the average traveler's needs.

Being able to provide updated, necessary, and clear information on what an average traveler might need, the application will utilize both user-provided information and up to date airport and active airplanes information through the usage of the OpenSky API, to provide the user not only with its flight information, but also with extra features such as live data on the flight, crowdsourced feedback, opinions, and delays based on airport, airline, route, historical, and active data.

1.1.2 Objectives:

The system will attempt to fill the existing market gap when it comes to flight tracking. The goal is to provide users a way of predicting their travel experience based on, mostly, crowdsourced intelligence.

1.1.3 Scope:

The scope of the application is to develop a user-centered tracking system for flights that can address the limitations imposed by the flight enthusiast focused applications that might leave a great number of potential users that might need a service like this one whenever they need to travel. It’s essential to focus not only on aviation enthusiasts, but also to match the needs of the average traveler. Through the help of the OpenSky API to gather both airport and aircraft information, mixed with user-provided data to present a comprehensive set of data and features that benefit the end user.

1.1.4 Users:

The potential user not only covers the aviation enthusiast, but potentially could be any person that is about to travel anywhere by flight. The main objective is to provide the necessary information contained in an understandable format, where the user finds the prioritized information while also being capable of looking at the details after in case of any interest. This way, the user is not being encaged into the format we designed, but it's also allowed to explore and figure out all the features and information present.

1.1.5 Benefits:

The benefits of having a microservices architecture mixed with a layered architecture, is that whenever the system needs maintenance, or any issue has been found in a particular area of the system, it’s not necessary to put the entire system offline. It’s possible to just maintain the area being maintained or fixed for the necessary time, and the rest of the system doesn’t have to be affected necessarily. This not only enables us to have an overall specified control over the system, but also enhances the system with a higher chance of having minimal downtime.

1.2 System Overview

1.2.1 Constraints:

One of the main constraints for the project, with a possibility of enhancement in future versions, is the source of our solid data, this being the flight and active aircraft information that is retrieved by the OpenSky API. While being opensource, the API only allows 4,000 request credits per day, each categorized by the size of the data being requested (from 1 to 4 points). From this, we’d be able to use, on average, 2.7 credits per minute within a day, where the current solution arranged by the team is to devise a certain amount of these credits for frequent database updates, while also having a “reserve” for emergency requests.

This constraint could be solved in future versions of the system, by exchanging this free and opensource API, into a paid API, which would provide us with the necessary calls that could ease up the constraints provided. Of course, this exchange would mean that the project would have funds

to maintain this kind of expense, but it is a possibility considering the success of this version, and possible investor attraction.

1.2.2 Security Considerations

- User Data Protection:
    - Storage of user data through internal cookies to prevent data centralization.
    - Necessary user data is anonymously stored and deleted when no longer in use.
- Application Data Protection:
    - Multi-layer path from client to DB.
    - DB is only accessible through a defined interface.
    - Access to web application utilizes SSH encryption.

1.3 Definitions and Acronyms

UML = Unified Modeling Language
DB = Database
OpenSky = API to be used to provide information to the databases.

2. Software Requirements Specifications

2.1.1 Functional Requirements

- **Flight Search**
  - The system shall be capable on letting users search based on flight number
  - The system shall be capable on letting users search based on arrival city
  - The system shall be capable on letting users search based on departure city
  - The system shall provide an average delay for flight whenever retrieving one
- **Interactive map**
  - The system's interactive map shall display the airport status
  - The system's interactive map shall display the flight route
- **Flight Notifications Subscription**
  - The system shall allow the user to subscribe to receive notifications from a particular flight.
    - This subscription will result in receiving an e-mail upon any change from the flight.
    - The subscription can be revoked at any time.
- **Crowdsourcing**
  - The system shall allow users to report delays for specific incidents with specific flight number or airport name.
  - The information provided by the users shall be a source for new delay average calculation to be displayed in the system.
- **Airport status**
  - The system shall display weather conditions in the selected airport.
  - The system shall provide the user with the flight's depart gate.
    - The system shall also push a notification whenever the depart gate is changed.
  - Airport Intelligence-Related Requirements
    - Average flight delays at specific airports.
      - Times of day with the most frequency of delays.

        - Common causes for delays at specific/major airports.
        - Prepare edge case where the airport doesn't have available intelligence.
    - Real-Time airport delay trend monitoring.
- **Flight status**
    - The system shall provide the live location of the plane.
    - Flight Intelligence-Related Requirements
        - Flight expected delay based on:
            - Real-Time delay trends related to airlines.
            - Real-Time delay trends related to flight route.
            - Historical flight delays from specific flight.
        - Real-Time flight delay monitoring.
- **Notifications**
    - The system shall provide real-time notifications on cancellations, delays or gate changes.
- **Feedback and Ratings**
    - The system shall allow user feedback and ratings on flights.

2.1.2 Non-Functional Requirements

- **Security**
    - Provide end-to-end encryption for user's data security.
    - SSH Certificates for HTTP/s encryption.
- **Adaptability**
    - The system shall be compatible with different browsers and operating systems.
    - The system shall provide learnability for ease of use and practicality.
- **Performance**
    - The database data shall be updated with a short response time for the live tracking feature.
    - The system shall support high throughput for both data intake and user presence.
- **Reliability**

- Since the system is based on microservices, it shall be prepared for a service critical failure while the application is still running.
- The system shall be prepared for a critical failure or crash.
- The system shall have no full down-time
- The system shall only allow down-time per service

– **Scalability**
- The system shall be able to handle a growing number of flights and users.

– **Usability**
- The system shall provide contextual help and tooltips for new users.
- The system shall provide an easy-to-use design while also maintaining the relevant information on display.

– **Availability**
- The system should have minimal downtime.
- The system should advise users in advance whenever maintenance is necessary.

– **Data Backup and Recovery**
- There should be a data recovery mechanism in case of system failure.

– **Compliance**
- The system should comply with aviation regulations and the GDPR's data regulations.

2.1.3 Prioritized Functional Requirements

- **Flight Search:**

  The system shall be capable of letting **users search based on flight number.**
- **Flight Status:**

  The system shall provide the live location of the plane.
- **Crowdsourcing:**

  The system shall allow users to report delays for specific incidents with a specific flight number or airport name.
- **Notifications:**

The system shall provide real-time notifications on cancellations, delays, or gate changes.

These 4 requirements are the most important because they define the core of the project. They follow the functionality, usability, and responsiveness of the flight management system AeroTelcel. The flight search makes the user find the specific flight to track. The Flight Status gives us accurate data of how the trip is going (real-time data) or will go based on predictions. The crowdsourcing ensures data reliability by making users use real-time data. The notifications have information on the status of the flight (eg. Cancelled, suspended, boarding, etc.).

2.1.4 Prioritized Non-Functional Requirements

- **Security:**

  Provide end-to-end encryption for user's data security.

- **Performance:**

  The database data shall be updated with a short response time for the live tracking feature.

- **Reliability:**

  Since the system is based on microservices, it shall be prepared for a service critical failure while the application is still running.

  The system shall have no full downtime.

- **Scalability:**

  The system shall be able to handle a growing number of flights and users.

These 4 Non-Functional requirements are the main focus of the project. Security is important to provide a secure environment to our users, since personal data is going to be used. Performance is important since we are using real-time data and the project cannot be slow. Reliability is important because when down the response time shall be quick since we need this data reach our customers.

2.1.5 Requirements Representation with UML Use Case Diagram

UML Use Case Diagrams aim to illustrate the ways in which a particular user will intract with a system. They help define the scope of the system, as well as the goals the application should fulfill. UML Use Case Diagrams work as a starting point for the system's architecture, as they are created well before any thought goes into the system's specifics.

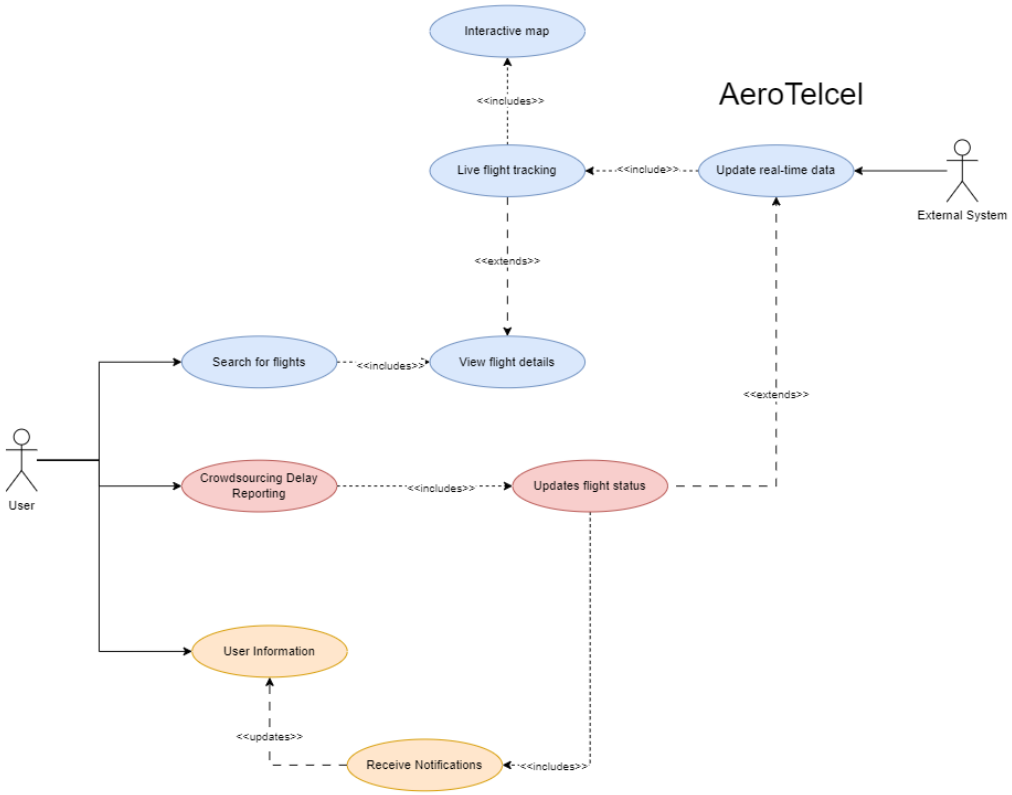

Figure 1 - UML Use Case Diagram

3. Overall Architecture of the System

3.1.1 High-Level Architecture

By creating a shared center of knowledge, defining a clear, well described, and fitting architecture will facilitate the development process of the system. Based on the pre-defined requirements, a list of relevant and desired architectural patterns were selected.

- **Layered Architecture**
    - 3 clearly divided layers: Front-end Layer, Logic Layer, Database Layer.
    - Each layer has a specific role and responsibility.
    - Separation allows for modularity, along with fast and independent development of each layer.
- **Client-Server Architecture.**
    - The Front-end Layer acts as the client, who makes requests to the server.
    - The Logic Layer and the Database Layer can be considered server-side, processing the client's requests, and sending responses through APIs.
- **Proxy Pattern**
    - Utilized to connect the User Interface Layer to the Logic Layer through a single point of contact.
    - Proxy will then forward the request to the corresponding service.
    - Used to enable development of User Interface Layer whilst the Logic Layer is under development.
- **Repository Pattern**
    - Utilized to interface the Logic Layer to the Database Layer.
    - Used to remove the dependency of the Logic Layer of the Database Layer.
- **Service-Oriented Architecture**
    - The Logic Layer will offer services to other components.
    - Well-defined interfaces and communication protocols between layers will be defined before the development of the components even commences.
- **Microservices Architecture**

- All components are independently scalable and deployable.
- They interact with each other through lightweight communication protocols, like REST.

- **Database-Centric Architecture**
  - The Database Layer is a critical component of the system.
  - It handles all data storage and retrieval, which is essential for the functionality of the rest of the system.

3.1.2 System Components

In order to comply with the selected architectures for the system, the following components are required.

- Front-End Layer:
  - User Interface
    - Responsible for receiving and responding to interactions from users.
    - Will request data from inferior layers in order to build-up the required response.
- Logic Layer:
  - Logic Interface
    - Responsible for receiving and responding to requests from the User Interface.
    - Will act as an API Gateway between the rest of the logic components, and the User Interface.
  - Flight & Map Logic
    - Calculates, retrieves, and processes the requested airport and flight data.
  - Subscription Management
    - Manages Subscriptions of Users.
  - Notifications and Feedback Logic
    - Manages real-time notifications.
    - Handles user feedback and reports on flights and airports.
- Database Layer:
  - Database Interface
    - Responsible for receiving and responding to requests from the Logic Layer.

- Will act as an API Gateway between the Logic Layer, and the Database.

- Database
  - Stores and retrieves airport and flight data, user subscriptions, user feedback, etc.

3.1.3 Software Design with UML Component Diagram

UML Component Diagrams aim to show the structure of the software system that is to be created, along with the desired interfaces and internal systems. At a high level, component diagrams help as an initial definition of the system's architecture, and aid in the future development of deployment diagrams, class diagrams and sequence diagrams. They play a vital role in microservice-centered applications, like "AeroTelcel", given the complexity of interfacing between services.

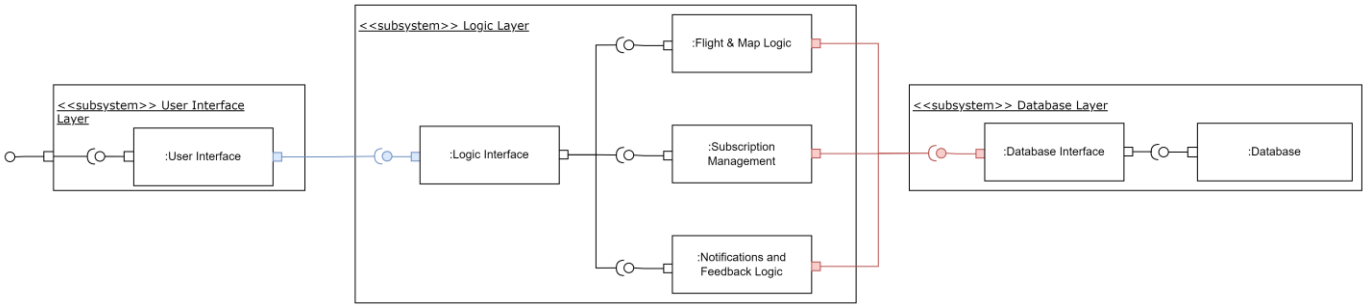

Figure 2 - UML Component Diagram

3.1.4 Software Design with UML Class Diagram

This UML Class Diagram aims to explain, with a specification perspective, to display an early-stage concept of how the system is going to work and the overall planned structure of it. The result is a layered architecture where in the presentation layer, the user receives each part of the information handled by both business-logic and database layers. The microservices, with the database interface, are in the business-logic layer, handling data input and output from both presentation and database layers. And in the end, the database layer is in charge of handling the connection to the API and updating its values so whenever any service from business-logic wishes to request any data, the database is capable of being up to date so it's able to return results in time.

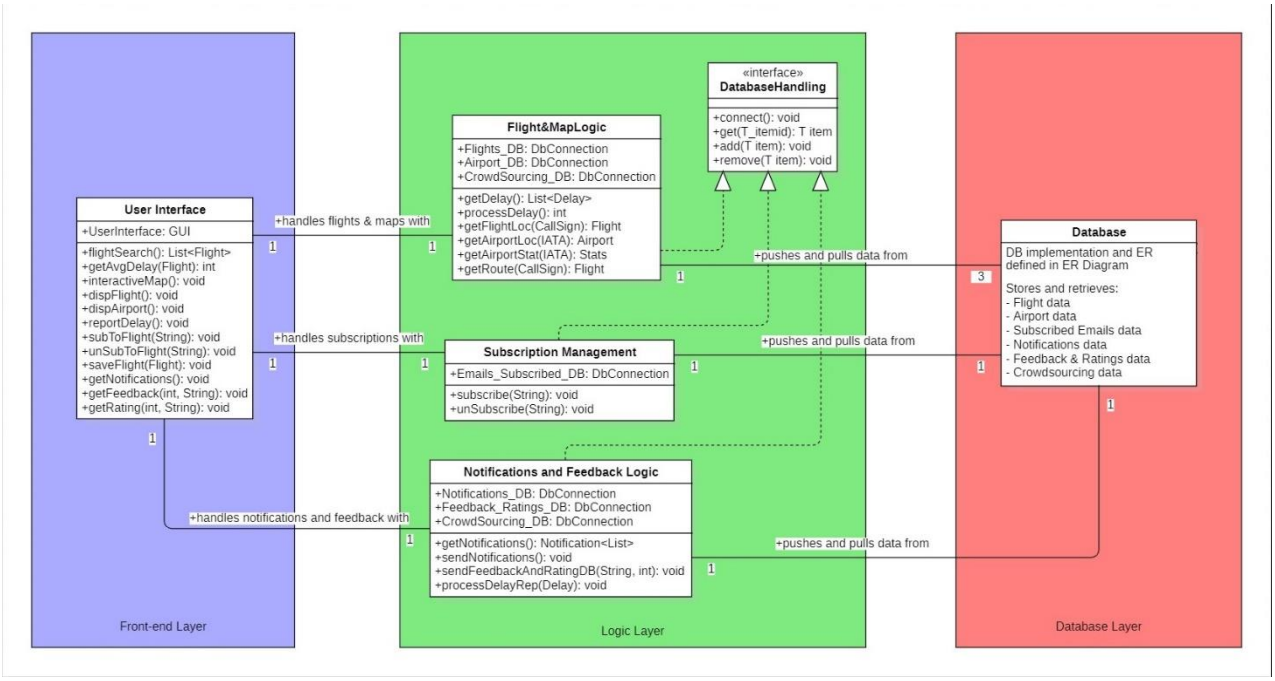

Figure 3 - UML Class Diagram made with free version of StarUML

3.1.5 Software Design with UML Sequence Diagram

UML Sequence Diagrams aim to illustrate the sequence of messages in between the components of a system when an interaction occurs.

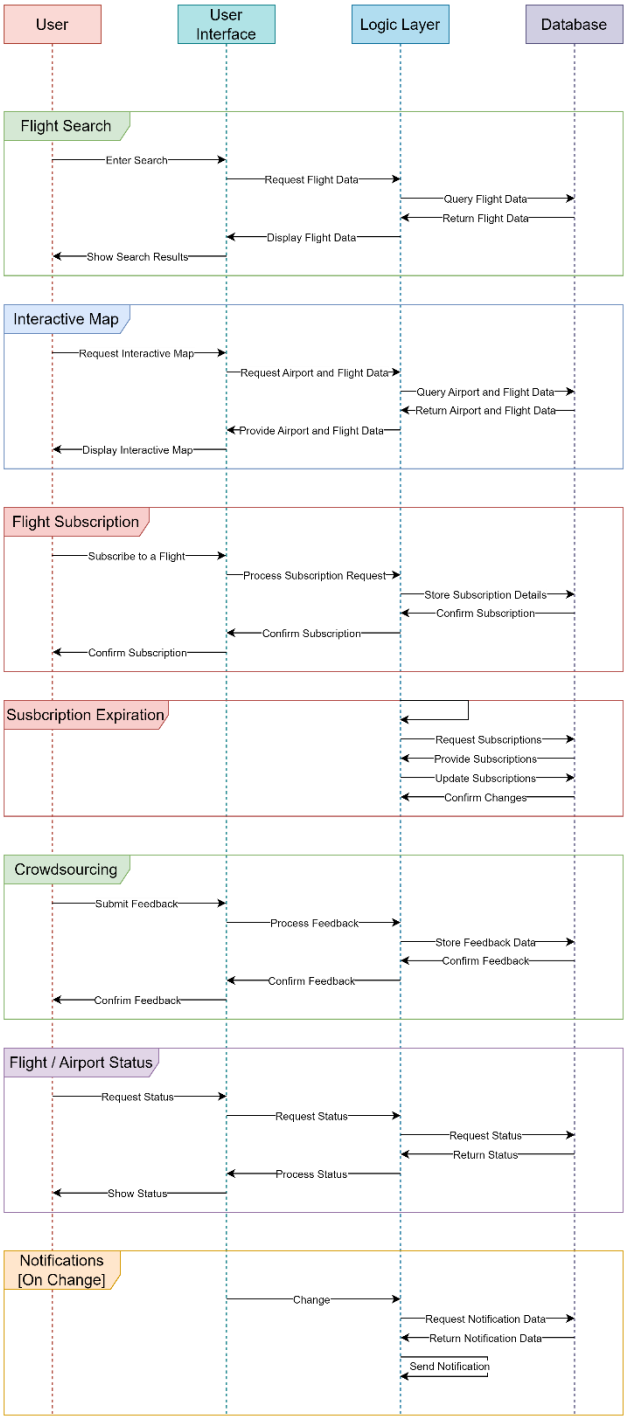

Figure 4 - UML Sequence Diagram

3.1.6 Database Design with UML ER Diagram

Besides the data provided by user interaction (subscriptions, crowdsourcing, feedback, etc.), an essential factor to determine the main structure of this project's UML Entity Relationship Diagram was determined by the OpenSky's API structure that is to be used within this system. For both airports and active airplanes, the results returned by the API consist in a JSON object counting with a timestamp determined by the time of the request, and an array of 'states' as the documentation indicates. This array, containing most fields of our own database, will be unraveled by our code, and assigned to each of its spaces within our database, and update the data as follows:

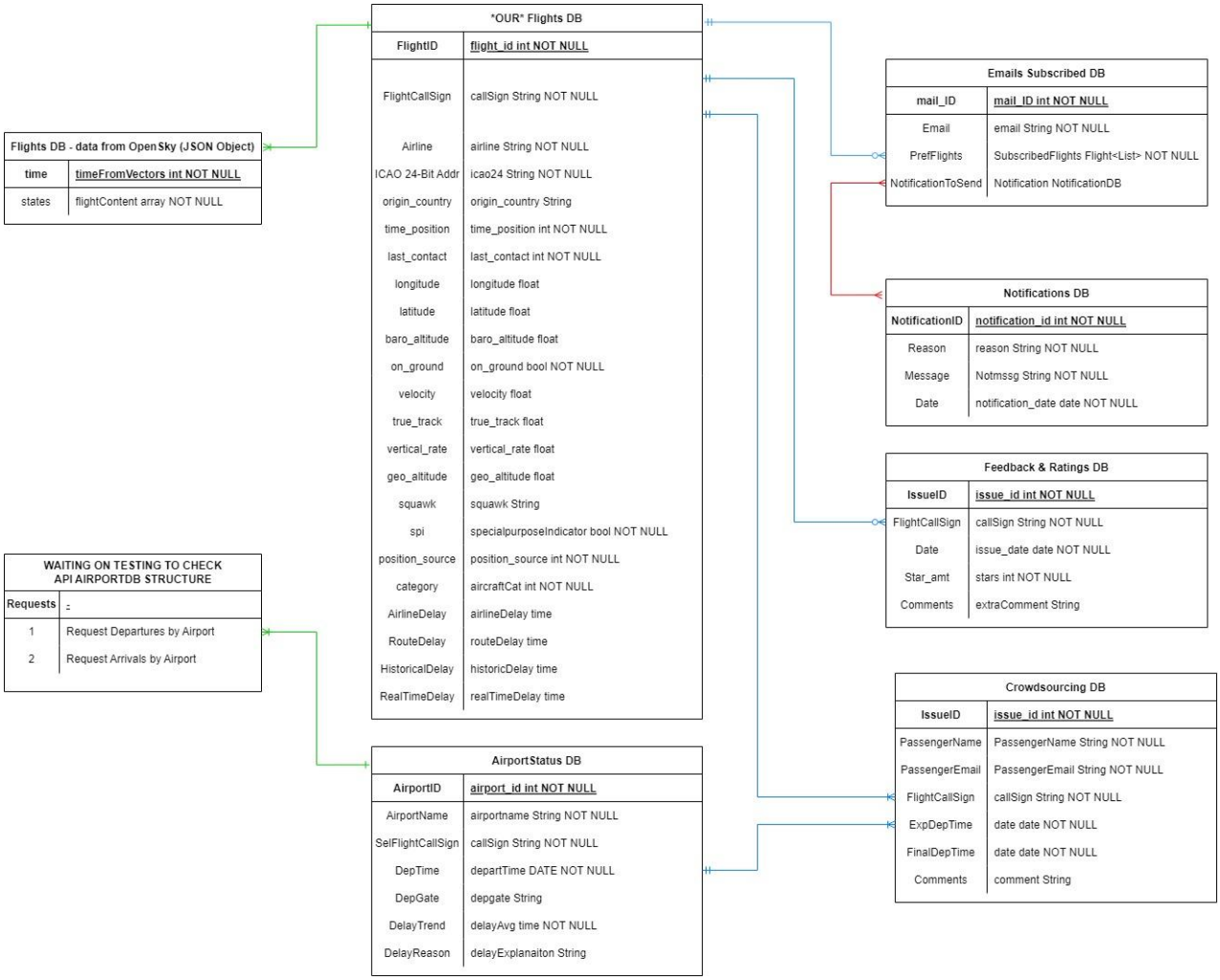

Figure 5 - UML ER Diagram with free version of StarUML

4. References (APA)

- Schäfer, M., Strohmeier, M., Lenders, V., Martinovic, I., & Wilhelm, M. (2014). *Bringing Up OpenSky: A Large-scale ADS-B Sensor Network for Research. In Proceedings of the 13th IEEE/ACM International Symposium on Information Processing in Sensor Networks (IPSN)*. Retrieved November 10, 2023, from https://opensky-network.org
- Schäfer, M., Strohmeier, M., Lenders, V., Martinovic, I., & Wilhelm, M. (2014). The OpenSky Network API documentation. Retrieved November 10, 2023, from https://openskynetwork.github.io/opensky-api/

5. Appendices

Figure 1 - UML Use Case Diagram
https://drive.google.com/file/d/1xqLPc6VIRmaWcpZkHk-STf-z3d0fJtTl/view?usp=sharing

Figure 2 - UML Component Diagram
https://drive.google.com/file/d/1vxFc-iB3fuTjx3s2lBi4A8pa0fYLdbym/view?usp=sharing

Figure 4 - UML Sequence Diagram
https://drive.google.com/file/d/1G0HRlvd1cgI7_EFrxI1yaWz8C_CogAbK/view?usp=sharing