Software Engineering Specification

Aerotelcel

https://github.com/JRamirezDD/Aerotelcel

Revision 1.0

Revision History

Version	Name	Reason For Changes	Date
1.0	Jorge Ramirez de Diego	Documentation of: Software Requirements Specifications - Non-Functional Requirements - UML Use Case Diagram Software Design Specifications - Architecture Design o High-Level Architecture o System Components o General Component Design o System UML Component Diagram o System Design with UML Sequence Diagram Software Implementation - Tools and Technologies Used o Back-end - Internal API Design o Subscription Handler - Integration Tests o Subscription Lifecycle - Automated Testing o Build-Triggered Test Execution - Application Installation and Launch Procedure o Requirements o Configuration Instructions o Launch Procedure	*24/03/2024*
2.0	Javier Ortega Mendoza	Documentation of: Purpose of the System - Scope Software Requirements Specifications - Functional Requirements Software Design Specifications - Database design - UML ER Diagram - Use Case Diagram Explanation Software Implementation - Tools and Technologies Used o Back-end, OpenSky API - Internal API Design	*24/03/2024*

		○ Flight Data Handler ○ Flight Data Receiver ○ Notifications Handler ○ Subscription Redis - Stress Tests ○ doSearch() Method Test - Application Installation Launch Procedure ○ Requirements ○ Configuration Instructions ○ Launch Procedure	
3.0	Marcos Gonzalez Fernandez	Documentation of: ● Purpose of the system: ○ Introduction of system ○ Objectives ○ Users ● System Overview: ○ Constraints. ○ Security and considerations. ● Definitions and acronyms ● Functional Requirements: ○ Flights and Airports ○ Subscriptions and notifications ○ User feedback/ Reports ● Prioritized Requirements ○ Functional Requirements and explanation. ● UML Class Diagram ● Design Patterns ● Tools and technologies: ○ Front-End. ● Screenshots of the application ● Application installation for launch procedure ○ Front-End ● Unit test: ○ Front-End	*24/03/2024*

Approved By

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

Name	Signature	Department	Date
Marcos Gonzalez Fernandez	[illegible]	AeroTelcel Development	21.03.2024
Jorge Ramirez de Diego	[illegible]	AeroTelcel Development	21.03.2024
Javier Ortega Mendoza	[illegible]	AeroTelcel Development	21.03.2024

1. Introduction

1.1 Purpose of the System

1.1.1 Introduction of the system

The ‘AeroTelcel’ application aims to fulfil the requirement of a user-centered flight tracking system. Current solutions such as ‘flightradar24’ or ‘flightaware’ are over-complicated or simply not enough. We have the vision of making a simple-to-use and precise flight-tracking system with new features that will interest potential users. This application is not only a flight tracker but also searches airports. This is a huge differentiation from other applications since we allow users to plan their trips with better and more precise information.

Being able to provide real-time information is what the traveller needs, the application utilizes the information provided by OpenSky API to update the users. Here we want to make our users happy and want to know their experiences, that is why we have questionnaires to know how their results with those flights or airports were. Also, the system lets the user subscribe to these flights or airports, making the information available in their emails.

1.1.2 Objectives

The system will attempt to fill the existing market gap regarding flight tracking. The goal is to provide users with a way of predicting their travel experience with information of flights and airports.

1.1.3 Users

The potential user is not only the aviation enthusiast, but we also want every person who takes a flight or is going to receive someone from it to use it. The main objective is to provide the necessary information in an easy-to-read format, where the prioritized information is what the user can see. We don’t want data that the user can potentially confuse, we want the simplest data but still give what they need. Also, a simple design of the interface is key for every new user to quickly understand the features of the application.

1.2 System Overview

1.2.1 Scope

The scope of this project is to be able to provide the user with a readable, reliable, easy to use and understandable system that enables it to lookup flight and airport information. Whether the user wishes to look for an airport or flight, the system should be able to deliver current information on it. On the side of the system, it should be able to handle high information throughput for flight data, while also being able to handle data production to the end user concurrently. Part of this would be supported on a microservices architecture, where the user will be able to perform different tasks asking information from different services, while they each communicate information between each other. This design will not only improve data accessibility features but will also enhance the product's reliability. Since each module will be a different service, at times when source code modification or additions are needed, or any of the services experience any type of issues, the system in its entirety will not have to undergo maintenance, allowing the management team to handle the issue affecting the crashed/modified service and allow them to re-enable it while still having a functional system. Other features within the project include map visualization of flight or airport, capacity for user to subscribe to an airport or flight, the capability to look at an airport's arrivals and departures, and the capacity to file reports for both airports and flights.
The system will be limited to a web application accessible via web browser to the end user, while it is all going to be able to be run as pod containers through Docker or as SpringBoot applications. The system will also mask project's details through the usage of intermediary layers to mask the main services utilities, code, and features, it will also have the capability to refresh the flights added at a given set of time, being able to checkup, and update or delete, already existing flights depending on the time of landing or departure within the database.
While other flight applications might be using paid APIs with different, faster, and more efficient features, this project, through the usage of the OpenSky API, will retrieve all flights, limiting the system with the API's database reach and API calls efficiency. Issue which can be addressed in future instances with the help of other paid APIs.

1.2.2 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. Also, the data supplied by this API is often missing different

pieces of information required for our system to function, so extra implementations towards data integrity and sanitation had to be added so the information in the database could be in a functioning state and not wasting physical memory.
This constraint could be solved in future versions of the system, by exchanging this free and opensource API, into a paid API, which would be more reliable and could 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.

1.2.3 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 applications utilizes SSH encryption.

1.3 Definitions and Acronyms

UML = Unified Modeling Language
DB = Database
API = Application Programming Interface
IATA = 3 letter location / definition code used for airline and airport definition and recognition.
ICAO (Airport) = 4 letter code used for airports for defining an airport's name based on region, country, and location.
ICAO (Flight) = categorization code for identifying aircrafts written in 24-bits.
Callsign = unique identifiers referring to an aircraft. For commercial flights, the first three letters belong to the Airline.

2. Software Requirements Specifications

2.1 Functional Requirements

Flights and Airports

- The system shall provide real-time information about flights and airports.

- The system shall be capable of letting users search based on flight number or airport code.
- The system shall be capable of letting users search for a flight based on whether it is an airport's departure or arrival.
- The system shall assign an airline to all flights read with the API's response.
- The system shall be able to remove old flight instances according to the last time they were updated.
- The system shall be able to handle a great number of flight data per API call.
- The system shall be able to perform data validation and sanitation when intaking from the API call for database integrity.
- The system shall be able to match Flights with both Airport's Arrivals and Departures stored flights.
- The system shall be able to bring to the front-end a final FlightData object with matched data from Flights, Airports, Arrivals, Departures and Airline entities for ease of data display.
- The system shall show a map with the coordinates of both airports and flights.
- The system shall be able to read output from python files to retrieve flight information.
- The system shall be able to auto update all flight data after a determined amount of time.

Subscriptions and Notifications

- The system shall allow users to subscribe to receive notifications from a particular flight or airport.
- The system shall allow users to unsubscribe to stop receiving notifications from a particular flight or airport.

User Feedback/Reports

- The system shall allow users to rate specific flights or airports based on various qualities.

2.2 Non-Functional Requirements

- Performance:
  - The system shall support high throughput for external API Data Intake.
  - The database data shall be updated with a short response time for the live tracking feature.
- Reliability:
  - The system shall have minimal downtime, it shall work 23.5 hours per day.
- Availability:
  - The system shall be available 95% of the day.
- Security:
  - The system must comply with GDPR's user data regulations.
  - The system shall prioritize flight data validation and sanitation to ensure database integrity.

- Maintainability:
  - The system shall adhere to coding standards and best practices documented in the project's coding guidelines.
- Portability:
  - The system shall be Containerizable.
- Usability:
  - The system shall be accessible with different browsers and operating systems.
  - The system shall provide learnability for ease of use and practicality.
- Scalability:
  - The system shall be scalable with the number of flights and airports.
- Interoperability:
  - The system shall be hostable on different platforms.
- Compliance:
  - The system shall adhere to coding standards and best practices documented in the project's coding guidelines.
- Efficiency:
  - The system shall optimize database queries and indexing strategies to ensure efficient data retrieval and processing.
- Resilience:
  - The system shall be prepared for a critical failure or crash.
- Data Integrity
  - The system shall ensure that the flight data taken from each API call undergoes a validation and sanitation process for safe database upload.

2.3 Prioritized Requirements

2.3.1 Functional Requirements

- The system shall provide real-time information about flights and airports.
- The system shall be capable of letting users search based on flight number or airport code.
- The system shall show a map with the coordinates of both airports and flights.
- The system shall allow users to subscribe to receive notifications from a particular flight or airport.
- The system shall allow users to rate specific flights or airports based on various qualities.

These Functional Requirements are the core of the functionality of the application, and we based our design on meeting these requirements. The importance of providing real-time information about flights and airports is the core for our users to have the best experience with the application. Searching for flight numbers or airport codes is a way we thought was practical since normally in a flight ticker or when a user is booking a flight they have that information at their disposal. The map is important for us because we think that the users will like not only having the information as words or phrases but also an illustration to represent the data. We want our users to be informed about either the flights or airports, the way we thought was by making them subscribe to this information and receive notifications about it. Finally, the rating of flights and airports is important for us because we want our users to share their experiences by answering simple questions about it.

2.3.2 Non-Functional requirements:

- The system shall be hostable on different platforms.
- The system shall be containerizable.
- The system shall support high throughput for external API intake.
- The system shall be scalable.
- The system shall be accessible with different operating systems and browsers.
- The system shall have minimal downtime, it shall work 23.5 hours per day.
- The system shall comply with GDPRs user data regulations.

The Non-Functional Requirements are important for the application since they evaluate the engine of the application. Being hostable represents our idea of being multiplatform, which enhances the scalability of the application. Containerizable is for the ease of transportation of the application. Supporting high throughput is important for the performance of the system. Being scalable is core due to the fact that we want the maximum reach for the application. Being accessible by different browsers or operating systems is important to increase the scope of users. Minimal downtime is important to have the maximum performance of the application. GDPR regulations are important because we want our users to be protected and secure by the application.

2.4 UML Diagrams to represent the requirements.

2.4.1 UML Use Case Diagram

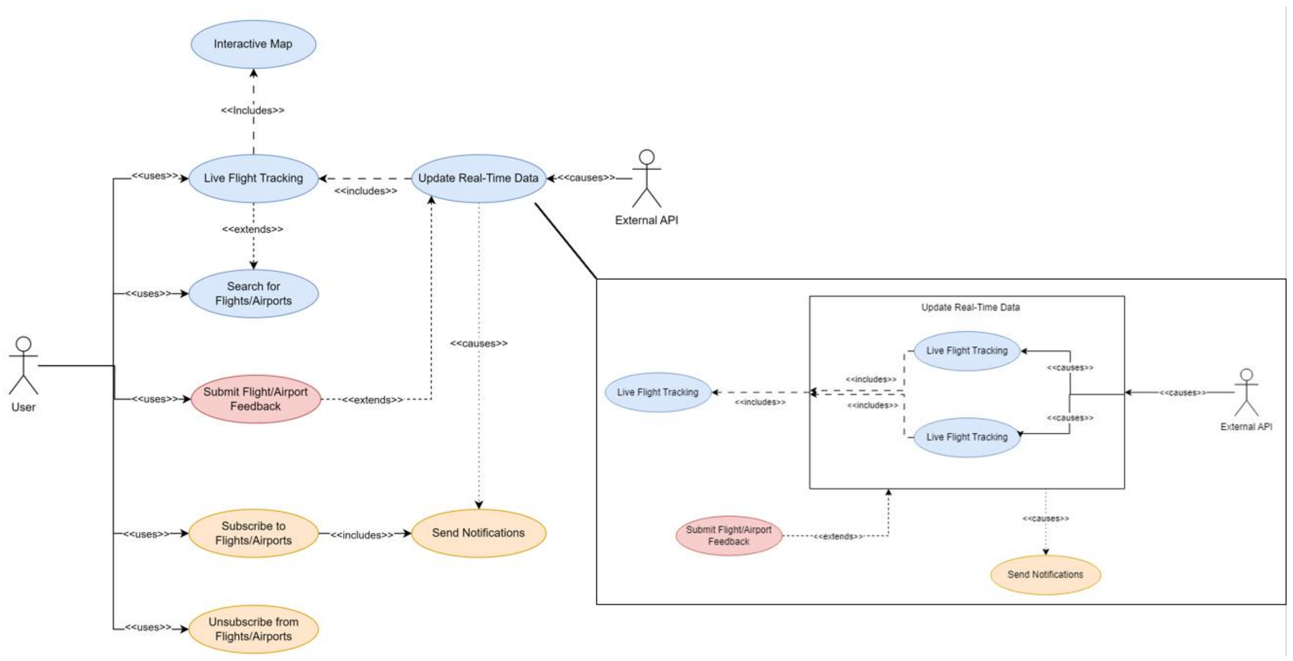


Figure 1 - Use Case Diagram

User can use life flight tracking for looking up current active flights retrieved by the database, also through looking up a specific flight or an airport, the user can consult the desired airport or flight code.
If there is a live flight within the time of search, and is available in database, the user will be displayed specific flight data, such as airport of arrival / departure, time of flight, current flight state and consult each of the airport's data, while also being able to see the flight's live location on the map.
Whenever looking up a specific airport, the user will be able to find current flights arriving or departing into the airport while also being able to locate them within the map with extra pieces of data.
The user can also submit feedback for either airport or flights, which will then be stored in the database for admin reinvestment into the system. This information helps the system advice users of possible changes.
The user can also subscribe / unsubscribe from specific airports or flights with an email address, which will then be notified of any changes the database's updated data receives. They could be created on delays, landings, departures, and is open for more types of notification for future implementation.
Finally, when it comes to the live flight data update, the OpenSky API is called to perform the methods `get_states`, `get_arrivals_by_airport`, and

`get_departures_by_airport`, which will then be read, formatted, and assigned to their corresponding databases by the flight data handler service, and its AirportDataController or FlightDataController respectively. Once uploaded, the front-end will receive the new information to display, or proceed with notifications on the rest of the back-end services.

3. Software Design Specifications

3.1 Architectural Design

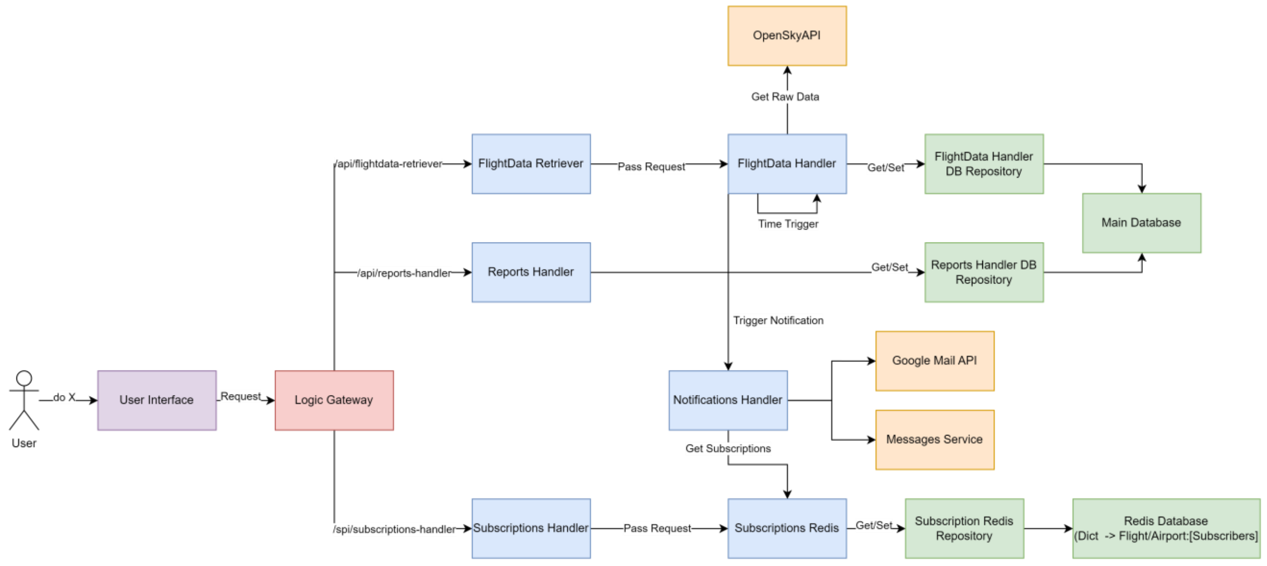
3.1.1 High-Level Architecture

- **Layered Architecture (General System)**
    - 3 clearly divided layers: Front-end Layer, Logic Layer, Database Layer.
    - Each layer has a specific role and responsibility.
    - Benefits of the pattern:
        - Separation allows for modularity and improved scalability.
        - At the time of development, allows for defining a person responsible for each layer, as well as distributing tasks fairly.
        - Teams can work on different layers at the same time without significant overlap or intra-component dependencies, significantly improving development efficiency.
- **Client-Server Architecture**
    - Front-end Layer acts as the client, who makes requests to the server.
    - Logic Layer of the application can be considered the server, as it processes the client's requests, and sends responses through the implemented REST APIs.
- **Broker Pattern**
    - Application's microservices can be accessed by the client through an intermediary service (Logic Gateway).
    - Logic Gateway is responsible for receiving client requests and forwarding them to the appropriate component.
    - Allows for complete decoupling between the Front-end and Back-end layers.
- **Service-Oriented Architecture**
    - Self-contained components provide one particular business service.
    - The inner workings of the components are irrelevant to their clients.
    - Components can be accessed through pre-defined communication protocols.
- **Microservices Architecture:**
    - Enhances modularity provided from Layered Architecture.
    - All components are independently scalable and deployable.

        - The use of service-oriented architecture and containerization fulfills this statement.
    - Responsibility for each component can be granted for one team member, ensuring fair development distribution.
- **REST**
    - Components are interacted with through their implemented API Interfaces.
    - Requests to API Interfaces are made through HTTP Requests.
        - Consisting of:
            - HTTP Verb
            - Pat
            - Data
                - Submitted in JSON format.
    - Requests to API Interfaces result in HTTP Responses.
        - Consisting of:
            - Response Code
            - Data
                - Returned in JSON format.

3.1.2 System Components

The following superficial system design was ideated with the goal of complying with the selected system architectures. This system design will be further described in the form of a component diagram in order to guide the development of microservices.



https://drive.google.com/file/d/1ItSofJOJR3fhpaJfh2e0GwrMP6lkkPf7/view?usp=sharing

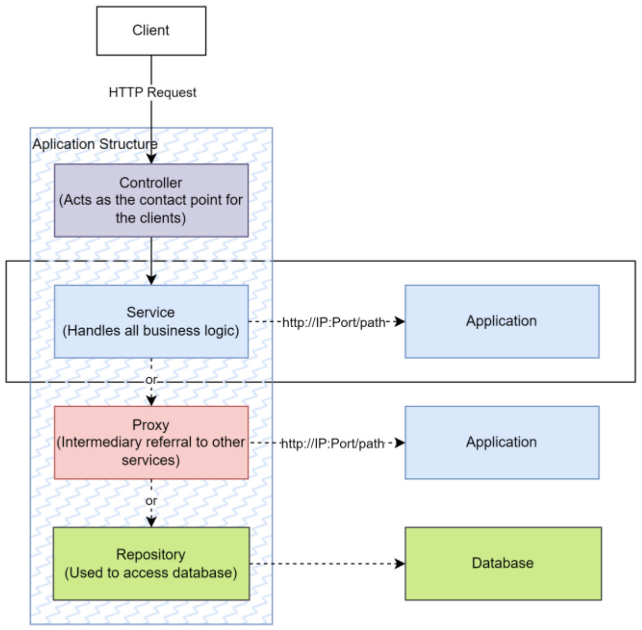
Figure 2 - Superficial System Design

- Front-End Layer:
  - User Interface:
    - Allows users to search for Flights and Airports.
    - Gives user information regarding selected Flights and Airports.
    - Hosts an interactive map through the external MapBox API.
    - Displays airport and flight locations through the interactive map.
    - Allows users to submit Flight/Airport feedback.
    - Allows users to subscribe and unsubscribe to flight/airport notifications through e-mail.
- Logic Layer:
  - Logic Gateway:
    - Responsible for receiving and responding to requests from the User Interface.
    - Provides a fixed access endpoint for requests external to the server.
    - Routes API requests to specific components:
      - FlightData Retriever
      - Reports Handler
      - Subscriptions Handler
  - Reports Handler:

        - Receives and stores Flight/Airport reports.
        - Retrieves Flight/Airport reports from Main Database.
    - FlightData Retriever:
        - Intermediary layer used to only allow user-required requests to FlightData Handler.
    - FlightData Handler:
        - Retrieves and stores Flight/Airport information.
            - Data undergoes a sanitation and validation process for safe database upload.
        - Updates Flight/Airport information through external OpenSky API.
            - Generates and forwards events for updated Flights/Airports.
            - Updates old information
        - Provides FlightData on request by matching both Airport and Flight information from database.
    - Subscriptions Handler:
        - Intermediary layer used to only allow user-required requests to Subscriptions Redis.
    - Subscriptions Redis:
        - Retrieves and stores Subscriptions.
    - Notifications Handler:
        - Receives events for updated Flights/Airports.
        - Sends corresponding notifications based on these events.
            - Able to send email and sms notifications depending on configuration.
- Database Layer:
    - Main Database:
        - Stores Airport information.
            - For each airport, store two external entity sets for both Arrivals and Departures.
        - Stores Flight information.
        - Stores Airport Reports.
        - Stores Flight Reports.
    - Redis Database:
        - Stores subscriptions in a Key->Value structure.

3.1.3 System Component Design

Given that the application was developed with microservices in mind, we have the advantage that we are able to define a standardized approach to each of the microservices' design. This approach allows us to better align the system to the specified requirements, as well as take the selected architectural patterns into account.



Client
HTTP Request
Aplication Structure
Controller (Acts as the contact point for the clients)
Service (Handles all business logic)
http://IP:Port/path
Application
or
Proxy (Intermediary referral to other services)
http://IP:Port/path
Application
or
Repository (Used to access database)
Database

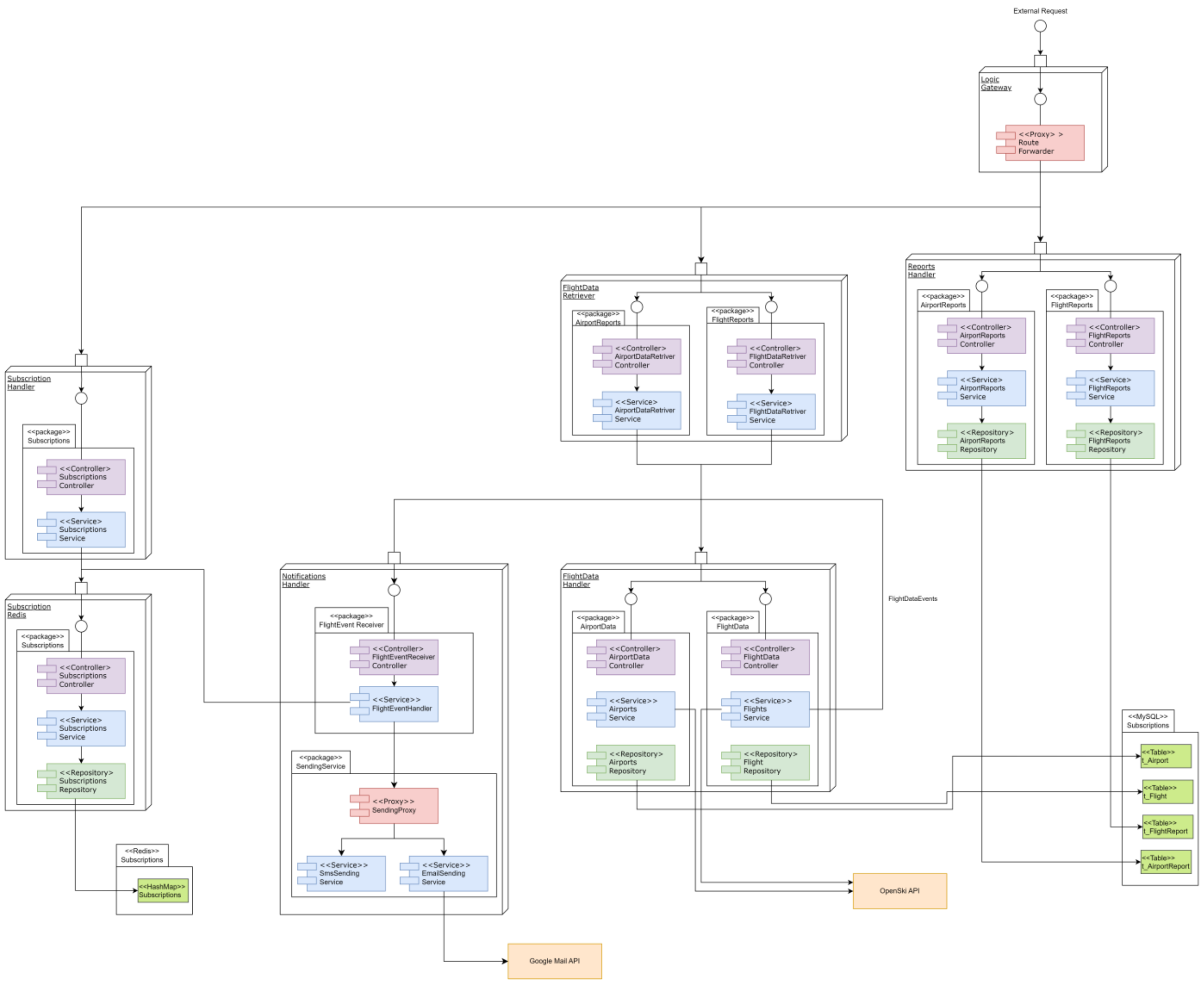
Figure 3 - General Microservice Design

- A layered architecture pattern was utilized for the general microservice design.
    - Microservices are divided into 2 or more layers, depending on the required used cases.
        - Controller Layer
            - Central contact point for incoming requests.
            - Delegates requests to corresponding service.
        - Service Layer
            - Handles the microservice's business logic.
            - Acts as mediator between the microservice with:
                - Other microservices
                - A proxy layer.
                - A repository layer.
        - Proxy Layer
            - Acts as a mediator between the service with other microservices.
            - Adds complexity but improves modularity.
            - Beneficial when called microservices change constantly.
        - Repository Layer
            - Acts as a single point of contact with a utilized database.

3.1.4 System UML Component Diagram

The functionality of each of the system's components has already been described in the System Components Section. In this section, we aim to explain and justify the internal design of each of the microservices.

UML Component Diagrams aid us in designing how components in an application should interact with each other. They are extremely fitting in microservices architectures, where instead of the internal design of each of the components being relevant, like is the case with class diagrams, the more general intractability between the system's components and microservices is required to gain a good understanding of the system.



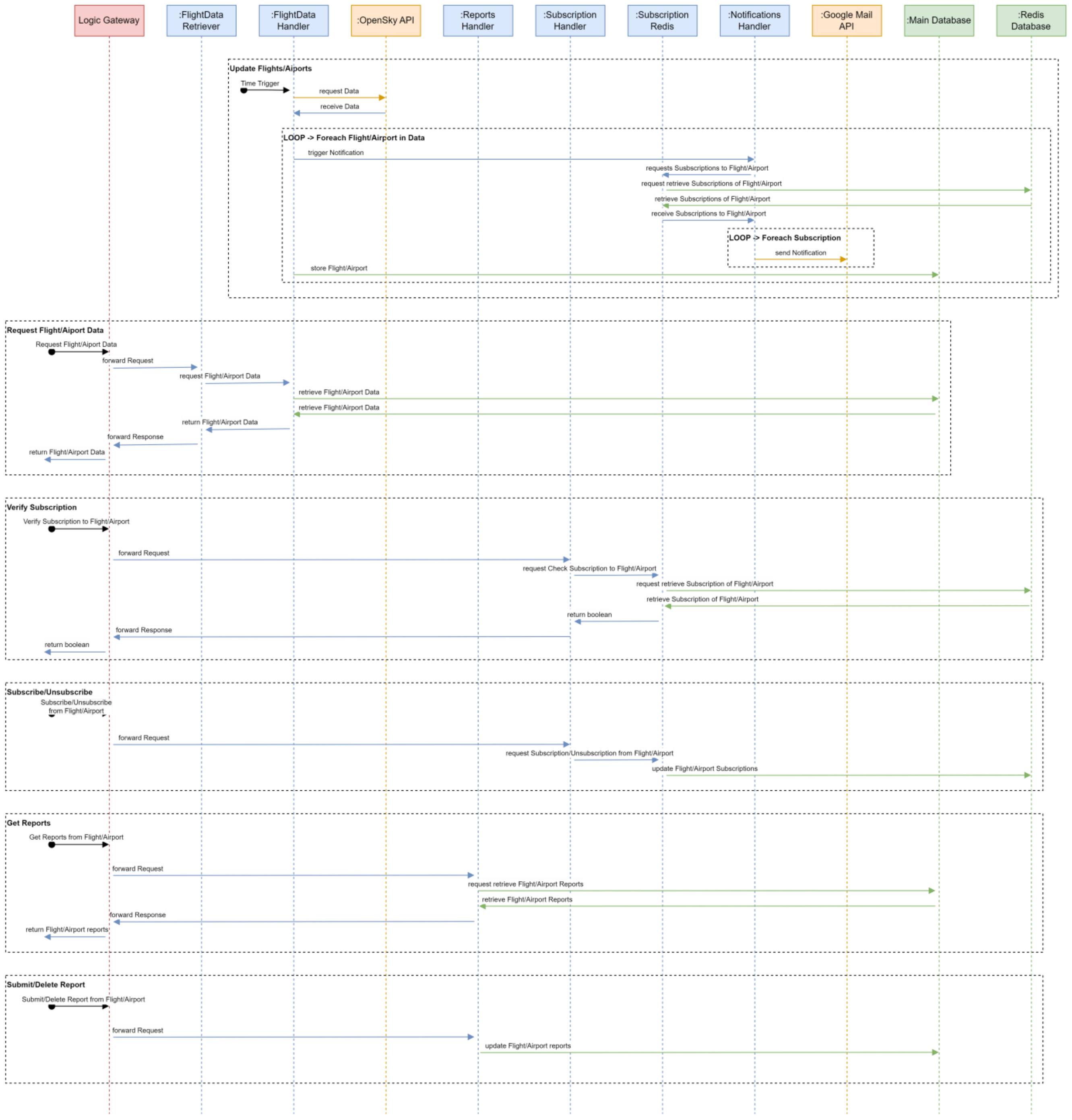
External Request
Logic Gateway
<<Proxy> > Route Forwarder
FlightData Retriever
<<package>> AirportReports
<<Controller> AirportDataRetriver Controller
<<Service> AirportDataRetriver Service
<<package>> FlightReports
<<Controller> FlightDataRetriver Controller
<<Service> FlightDataRetriver Service
Reports Handler
<<package>> AirportReports
<<Controller> AirportReports Controller
<<Service> AirportReports Service
<<Repository> AirportReports Repository
<<package>> FlightReports
<<Controller> FlightReports Controller
<<Service> FlightReports Service
<<Repository> FlightReports Repository
Subscription Handler
<<package>> Subscriptions
<<Controller> Subscriptions Controller
<<Service> Subscriptions Service
Subscription Redis
<<package>> Subscriptions
<<Controller> Subscriptions Controller
<<Service> Subscriptions Service
<<Repository> Subscriptions Repository
<<Redis>> Subscriptions
<<HashMap>> Subscriptions
Notifications Handler
<<package>> FlightEvent Receiver
<<Controller> FlightEventReceiver Controller
<<Service>> FlightEventHandler
<<package>> SendingService
<<Proxy>> SendingProxy
<<Service>> SmsSending Service
<<Service>> EmailSending Service
Google Mail API
FlightData Handler
<<package>> AirportData
<<Controller> AirportData Controller
<<Service>> Airports Service
<<Repository> Airports Repository
<<package>> FlightData
<<Controller> FlightData Controller
<<Service>> Flights Service
<<Repository> Flight Repository
FlightDataEvents
OpenSki API
<<MySQL>> Subscriptions
<<Table>> t_Airport
<<Table>> t_Flight
<<Table>> t_FlightReport
<<Table>> t_AirportReport

https://drive.google.com/file/d/1vxFc-iB3fuTjx3s2lBi4A8pa0fYLdbym/view?usp=sharing

Figure 4 - Application Component Diagram

3.1.5 System Design with UML Sequence Diagram

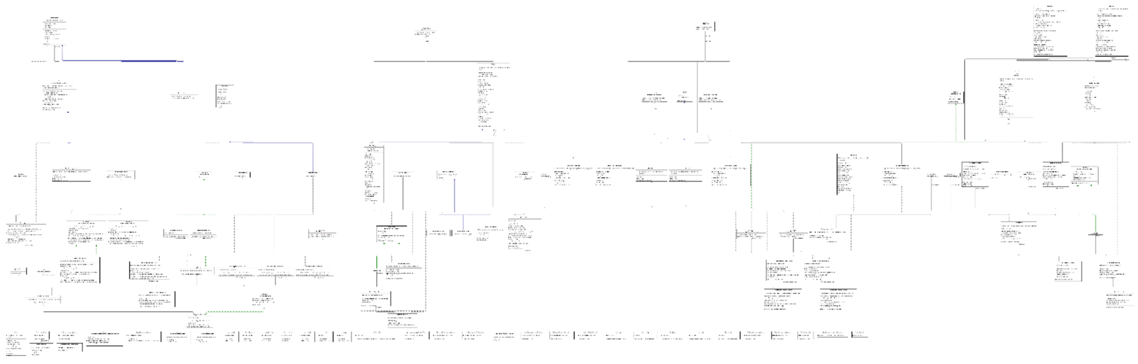
This sequence diagram from the microservices view allows us to visualize how and when interactions between components are expected. Although the internal processes might be significantly more complicated, only the inputs and outputs are relevant for acquiring a high-level overview of the system.


Logic Gateway
:FlightData Retriever
:FlightData Handler
:OpenSky API
:Reports Handler
:Subscription Handler
:Subscription Redis
:Notifications Handler
:Google Mail API
:Main Database
:Redis Database
Update Flights/Aiports
Time Trigger
request Data
receive Data
LOOP -> Foreach Flight/Airport in Data
trigger Notification
requests Susbscriptions to Flight/Airport
request retrieve Subscriptions of Flight/Airport
retrieve Subscriptions of Flight/Airport
receive Subscriptions to Flight/Airport
LOOP -> Foreach Subscription
send Notification
store Flight/Airport
Request Flight/Aiport Data
Request Flight/Aiport Data
forward Request
request Flight/Airport Data
retrieve Flight/Airport Data
retrieve Flight/Airport Data
return Flight/Airport Data
forward Response
return Flight/Airport Data
Verify Subscription
Verify Subscription to Flight/Airport
forward Request
request Check Subscription to Flight/Airport
request retrieve Subscription of Flight/Airport
retrieve Subscription of Flight/Airport
return boolean
forward Response
return boolean
Subscribe/Unsubscribe
Subscribe/Unsubscribe from Flight/Airport
forward Request
request Subscription/Unsubscription from Flight/Airport
update Flight/Airport Subscriptions
Get Reports
Get Reports from Flight/Airport
forward Request
request retrieve Flight/Airport Reports
retrieve Flight/Airport Reports
forward Response
return Flight/Airport reports
Submit/Delete Report
Submit/Delete Report from Flight/Airport
forward Request
update Flight/Airport reports

https://drive.google.com/file/d/1I2Yxkrs7XmSE9M07hrirUtLNX3i4G0FG/view?usp=sharing

Figure 5 - UML Sequence Diagram

3.2 Component Design with UML Class Diagram



https://drive.google.com/file/d/10YFg78CAHWF299Vz8ljXaDnFB6q-W4ty/view

Figure 6 - UML Class Diagram

Due to this diagram's length, we will provide a link for better observation.

3.3 Design Patterns

- Proxy Pattern
  - Forwards requests to the required destination.

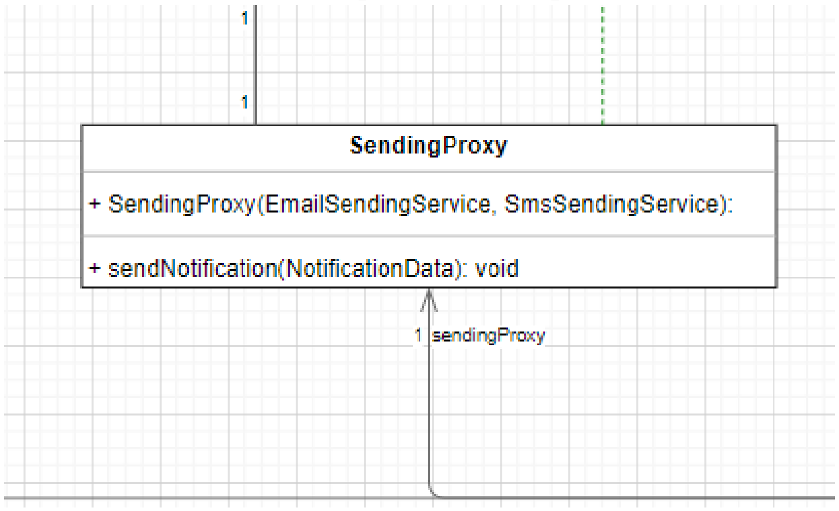

1
1
SendingProxy
+ SendingProxy(EmailSendingService, SmsSendingService):
+ sendNotification(NotificationData): void
1
sendingProxy

Figure 7 - Proxy Pattern

Here we can see the proxy pattern for the notification handler to request the important information that the user or the system needs.

- Repository Pattern
  - Acts as a single point of contact with the application's database

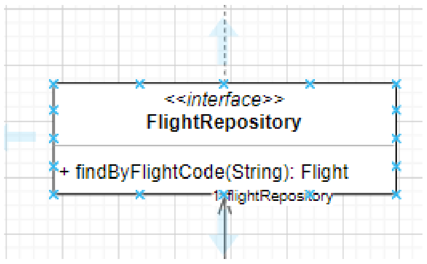
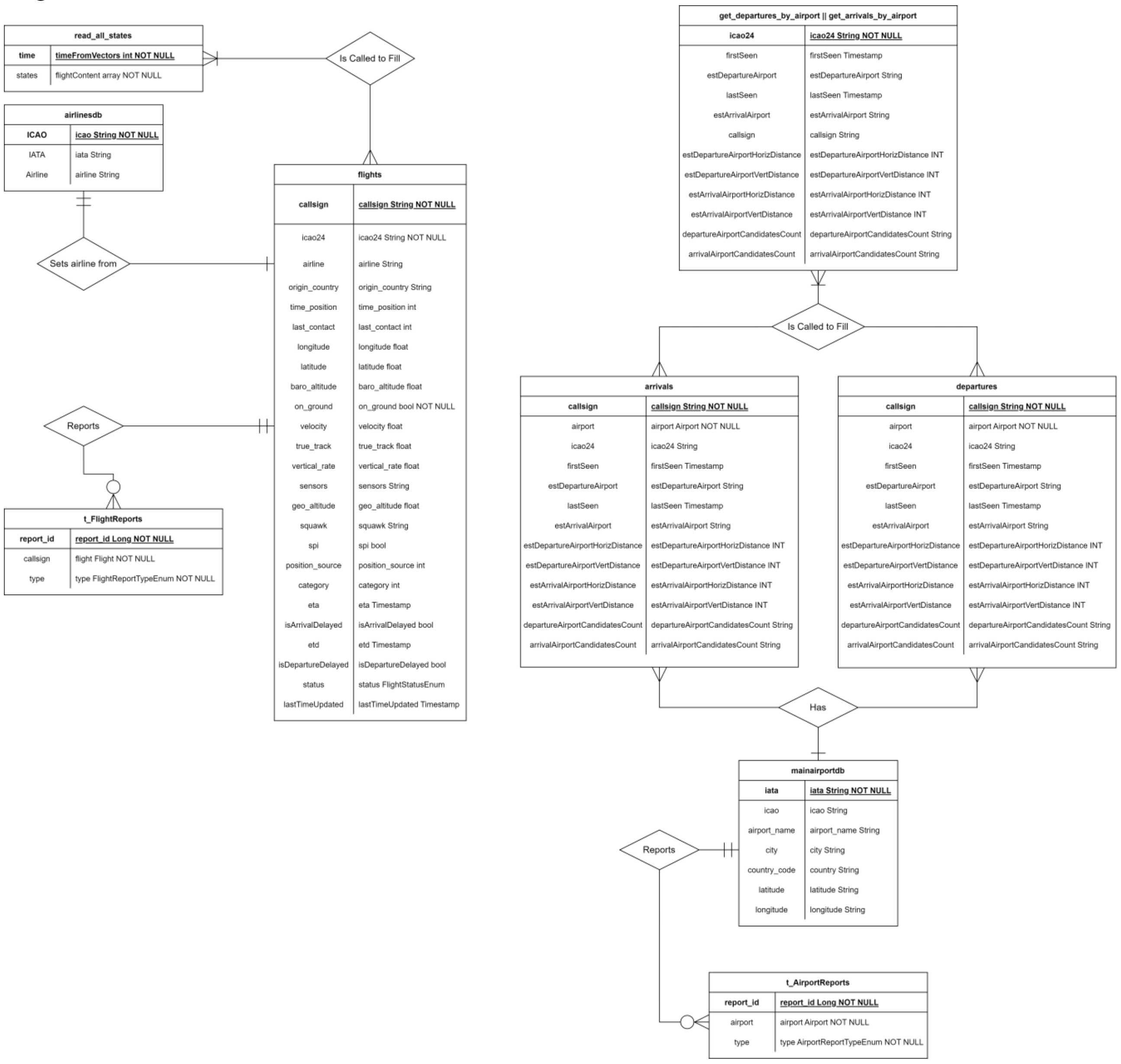


Figure 8 - Repository Pattern

In the class diagram this repository means the contact with the GET request of the user and the database in order to retrieve the information of the flight with the number.

3.4 Database Design with ER Diagram

Flights:


read_all_states
time | timeFromVectors int NOT NULL
states | flightContent array NOT NULL
Is Called to Fill
airlinesdb
ICAO | icao String NOT NULL
IATA | iata String
Airline | airline String
Sets airline from
flights
callsign | callsign String NOT NULL
icao24 | icao24 String NOT NULL
airline | airline String
origin_country | origin_country String
time_position | time_position int
last_contact | last_contact int
longitude | longitude float
latitude | latitude float
baro_altitude | baro_altitude float
on_ground | on_ground bool NOT NULL
velocity | velocity float
true_track | true_track float
vertical_rate | vertical_rate float
sensors | sensors String
geo_altitude | geo_altitude float
squawk | squawk String
spi | spi bool
position_source | position_source int
category | category int
eta | eta Timestamp
isArrivalDelayed | isArrivalDelayed bool
etd | etd Timestamp
isDepartureDelayed | isDepartureDelayed bool
status | status FlightStatusEnum
lastTimeUpdated | lastTimeUpdated Timestamp
Reports
t_FlightReports
report_id | report_id Long NOT NULL
callsign | flight Flight NOT NULL
type | type FlightReportTypeEnum NOT NULL
get_departures_by_airport || get_arrivals_by_airport
icao24 | icao24 String NOT NULL
firstSeen | firstSeen Timestamp
estDepartureAirport | estDepartureAirport String
lastSeen | lastSeen Timestamp
estArrivalAirport | estArrivalAirport String
callsign | callsign String
estDepartureAirportHorizDistance | estDepartureAirportHorizDistance INT
estDepartureAirportVertDistance | estDepartureAirportVertDistance INT
estArrivalAirportHorizDistance | estArrivalAirportHorizDistance INT
estArrivalAirportVertDistance | estArrivalAirportVertDistance INT
departureAirportCandidatesCount | departureAirportCandidatesCount String
arrivalAirportCandidatesCount | arrivalAirportCandidatesCount String
Is Called to Fill
arrivals
callsign | callsign String NOT NULL
airport | airport Airport NOT NULL
icao24 | icao24 String
firstSeen | firstSeen Timestamp
estDepartureAirport | estDepartureAirport String
lastSeen | lastSeen Timestamp
estArrivalAirport | estArrivalAirport String
estDepartureAirportHorizDistance | estDepartureAirportHorizDistance INT
estDepartureAirportVertDistance | estDepartureAirportVertDistance INT
estArrivalAirportHorizDistance | estArrivalAirportHorizDistance INT
estArrivalAirportVertDistance | estArrivalAirportVertDistance INT
departureAirportCandidatesCount | departureAirportCandidatesCount String
arrivalAirportCandidatesCount | arrivalAirportCandidatesCount String
departures
callsign | callsign String NOT NULL
airport | airport Airport NOT NULL
icao24 | icao24 String
firstSeen | firstSeen Timestamp
estDepartureAirport | estDepartureAirport String
lastSeen | lastSeen Timestamp
estArrivalAirport | estArrivalAirport String
estDepartureAirportHorizDistance | estDepartureAirportHorizDistance INT
estDepartureAirportVertDistance | estDepartureAirportVertDistance INT
estArrivalAirportHorizDistance | estArrivalAirportHorizDistance INT
estArrivalAirportVertDistance | estArrivalAirportVertDistance INT
departureAirportCandidatesCount | departureAirportCandidatesCount String
arrivalAirportCandidatesCount | arrivalAirportCandidatesCount String
Has
mainairportdb
iata | iata String NOT NULL
icao | icao String
airport_name | airport_name String
city | city String
country_code | country String
latitude | latitude String
longitude | longitude String
Reports
t_AirportReports
report_id | report_id Long NOT NULL
airport | airport Airport NOT NULL
type | type AirportReportTypeEnum NOT NULL

https://drive.google.com/file/d/1cBk93KEmHtU5hx5H-wg9i0lCnD5pmIPB/view?usp=sharing

Figure 9 - ER Diagram Part 1, Flights

Airports:

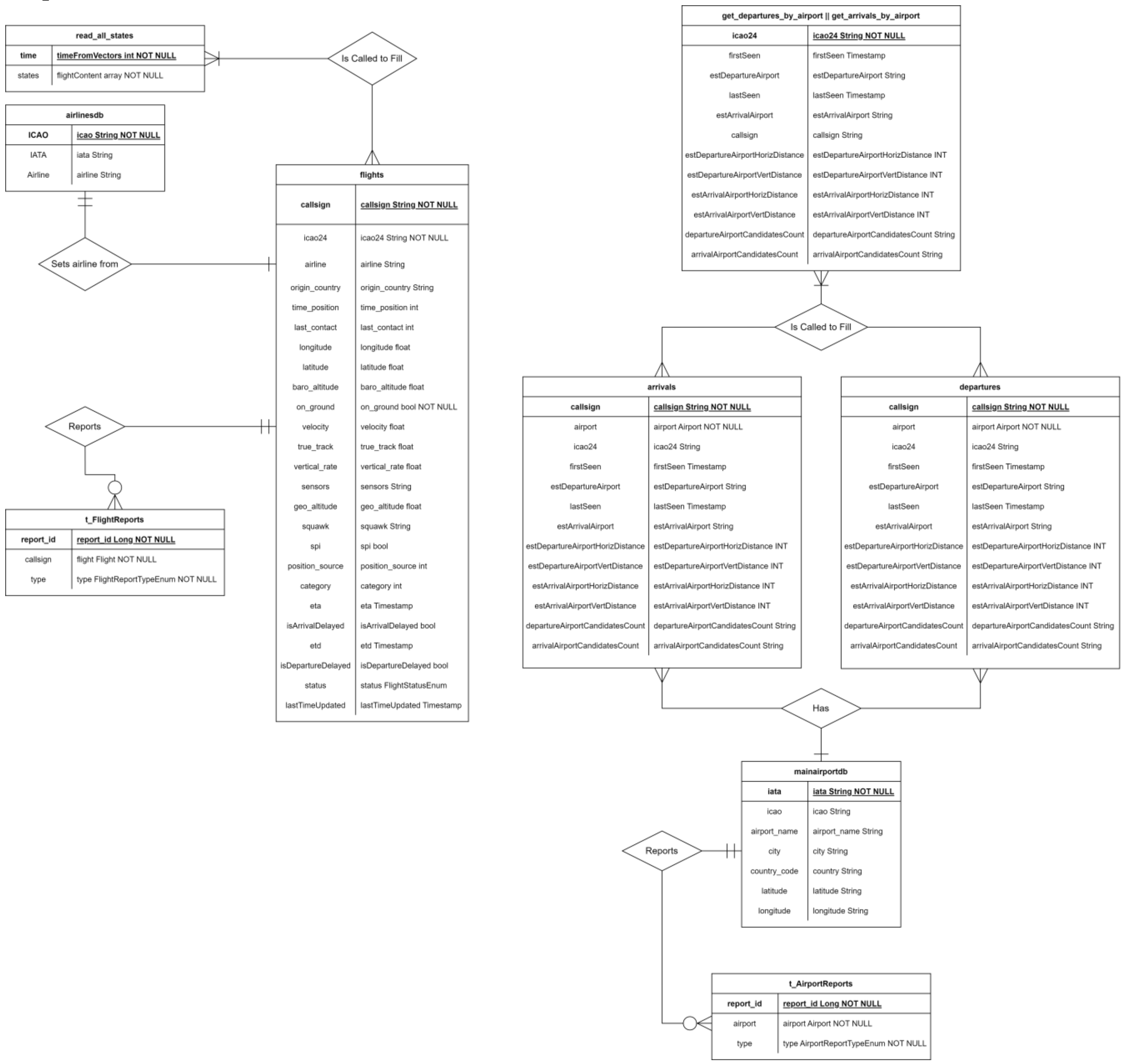

read_all_states
time | timeFromVectors int NOT NULL
states | flightContent array NOT NULL
Is Called to Fill
airlinesdb
ICAO | icao String NOT NULL
IATA | iata String
Airline | airline String
Sets airline from
flights
callsign | callsign String NOT NULL
icao24 | icao24 String NOT NULL
airline | airline String
origin_country | origin_country String
time_position | time_position int
last_contact | last_contact int
longitude | longitude float
latitude | latitude float
baro_altitude | baro_altitude float
on_ground | on_ground bool NOT NULL
velocity | velocity float
true_track | true_track float
vertical_rate | vertical_rate float
sensors | sensors String
geo_altitude | geo_altitude float
squawk | squawk String
spi | spi bool
position_source | position_source int
category | category int
eta | eta Timestamp
isArrivalDelayed | isArrivalDelayed bool
etd | etd Timestamp
isDepartureDelayed | isDepartureDelayed bool
status | status FlightStatusEnum
lastTimeUpdated | lastTimeUpdated Timestamp
Reports
t_FlightReports
report_id | report_id Long NOT NULL
callsign | flight Flight NOT NULL
type | type FlightReportTypeEnum NOT NULL
get_departures_by_airport || get_arrivals_by_airport
icao24 | icao24 String NOT NULL
firstSeen | firstSeen Timestamp
estDepartureAirport | estDepartureAirport String
lastSeen | lastSeen Timestamp
estArrivalAirport | estArrivalAirport String
callsign | callsign String
estDepartureAirportHorizDistance | estDepartureAirportHorizDistance INT
estDepartureAirportVertDistance | estDepartureAirportVertDistance INT
estArrivalAirportHorizDistance | estArrivalAirportHorizDistance INT
estArrivalAirportVertDistance | estArrivalAirportVertDistance INT
departureAirportCandidatesCount | departureAirportCandidatesCount String
arrivalAirportCandidatesCount | arrivalAirportCandidatesCount String
Is Called to Fill
arrivals
callsign | callsign String NOT NULL
airport | airport Airport NOT NULL
icao24 | icao24 String
firstSeen | firstSeen Timestamp
estDepartureAirport | estDepartureAirport String
lastSeen | lastSeen Timestamp
estArrivalAirport | estArrivalAirport String
estDepartureAirportHorizDistance | estDepartureAirportHorizDistance INT
estDepartureAirportVertDistance | estDepartureAirportVertDistance INT
estArrivalAirportHorizDistance | estArrivalAirportHorizDistance INT
estArrivalAirportVertDistance | estArrivalAirportVertDistance INT
departureAirportCandidatesCount | departureAirportCandidatesCount String
arrivalAirportCandidatesCount | arrivalAirportCandidatesCount String
departures
callsign | callsign String NOT NULL
airport | airport Airport NOT NULL
icao24 | icao24 String
firstSeen | firstSeen Timestamp
estDepartureAirport | estDepartureAirport String
lastSeen | lastSeen Timestamp
estArrivalAirport | estArrivalAirport String
estDepartureAirportHorizDistance | estDepartureAirportHorizDistance INT
estDepartureAirportVertDistance | estDepartureAirportVertDistance INT
estArrivalAirportHorizDistance | estArrivalAirportHorizDistance INT
estArrivalAirportVertDistance | estArrivalAirportVertDistance INT
departureAirportCandidatesCount | departureAirportCandidatesCount String
arrivalAirportCandidatesCount | arrivalAirportCandidatesCount String
Has
mainairportdb
iata | iata String NOT NULL
icao | icao String
airport_name | airport_name String
city | city String
country_code | country String
latitude | latitude String
longitude | longitude String
Reports
t_AirportReports
report_id | report_id Long NOT NULL
airport | airport Airport NOT NULL
type | type AirportReportTypeEnum NOT NULL

Figure 10 - ER Diagram Part 2, Airports

Explanation:

The structure for both higher entities *Flights* and *Airports* are defined in this matter mainly due to the result given by the API calls: `get_all_states`, `get_departures_by_airport` and `get_arrivals_by_airport`. The format on how the data is received is also represented in the diagram, clarifying data origin for *arrivals*, *departures*, and *flights* entities. Also, two entities' values come from external, static databases, these being *airlinesdb* and *mainairportdb*, which hold non-changing data sets, that are only necessary for API calls (icao from *mainairportdb*) and airline assignation (IATA from *airlinesdb*).

Flights holds the information provided by the API call, plus, adding the additional airline title whenever reading the API call and matching it to *airlinesdb*. Also, all data related to delay and *lastTimeUpdated* are added on read, declaring the local data, or updating on previous data.

Mainairportdb on the other hand, holds a pair of one-to-many relationships, both being groups of data from *Arrivals* and *Departures*, which are matched by the airport's icao code before upload to set an Airport object on the database row as well. *Mainairportdb*'s particular data has the purpose of helping front-end to represent the airport in the map and display extra data such as city, airport codes, name and country.

Finally, both *t_AirportReports* and *t_FlightReports* function separately. Both being created on call whenever a user wants to report either a flight or airport. Each of these databases holds either a Flight or Airport object in their rows, holding it for future crowdsourcing features. Also, for updating *flights, Arrivals and Departures*, each of these API calls are expected to be done around every 2 hours, while also having the capability to replace old data thanks to features such as *lastTimeUpdated.*

4. Software Implementation

4.1 Tools and Technologies Used

4.1.1 Front-End

For the front-end of the application, we used the following tools and technologies: React and React-Router-Dom. React is the core of our front-end code since is the skeleton of it, here we use some imports such as 'useEffect' or 'useState' for its functionality. React-Router-Dom was used to change from one page to another page, also it was important to pass certain Key values for enhancing its performance, 'useNavigate' and 'useLocation' were the imports that were used to pass the information.

4.1.2 Back-end

- MySQL
  - Used as Main Database.
- Redis
  - Used as Subscription Database.
- Gradle
  - Dependency Management.
  - Java version control.
- Springboot
  - Back-end microservices development framework.
  - Database handling through built-in repositories.
  - Intra-service communication via REST Clients.
  - Ensure proper logging tools for either action or error recognition.
  - Testing features for unit testing and integration testing.
- Docker
  - Provide containerization of each service, alongside configuration details.
  - Allow to deploy individual containerized microservices in the form of pods.
- Docker Compose
  - Deploy all system pods, including databases, in a cluster.
  - Provide an internal network for microservices intra-service communication.
  - Define container characteristics via cluster configuration file.
- Scripting Languages (Shell & Bash)
  - Automate Gradle Project building.
  - Automate Containerization of microservices.
  - Automate deployment of individual microservices in the form of docker containers.

- Python
    - Scripts for API flight retrieval.

4.2 External APIs

- Mapbox API:
    - Mapbox was used to position the map in the application, this tool is helpful in order to check where the flights are and where the airports are located.
- OpenSky API:
    - Utilized for handling all data intake for Flights and Airports. Called frequently to update data to be handled by flightdata_handler service.
- Google Mail API:
    - Utilized for sending email notifications to users when Flights and Airports are modified.

4.3 Internal API Design

Given the great extent of possible REST requests to the various APIs, instead of providing request and response schemas, we decided to provide the specific objects being transmitted, along with the source code reference.

4.3.1 FlightData Retriever

Endpoint	HTTP Method	Description	Request Object	Response Status	Response Object
/api/airportDataController/updateAllAirports	GET	Asks flight data handler to update all airports in the database. WARNING: Very large data request for API/ API doesn't recognize all airport codes. Works properly but behavior is broken by API and system's capabilities.	List<Airport.java>	OK	void
/api/airportDataController/updateSelectedAirport/{iata}	GET	Asks flight data handler to update chosen airport on both Arrivals and Departures repositories	Airport.java	OK	void
/api/airportDataController/getAllAirports	GET	Asks flight data handler to get all airports with their data	List<AirportResponse.java>	OK	List<Airport Response>
/api/airportDataController/getAllAirports/getAirportByCode/{iata}	GET	Asks flight data handler to get specific requested airport for	AirportResponse.java	OK	Airport Response

		display.			
/api/flightController/updateAllStates	GET	Asks flight data handler to update all the current flights in the database. If there's flights older than a day, these are deleted while current ones are updated with new data.	Flight.java	OK	Void
/api/flightController/getAllStates	GET	Asks flight data handler to return a list of all current Flight Objects	List<FlightResponse.java>	OK	List<FlightResponse>
/api/flightController/getFlightByCallsign/{callsign}	GET	Asks flight data handler to get a specific flight format including departure and arrival data	FlightDataResponse.java	OK	FlightDataResponse
/api/flightController/deleteAllStates	DELETE	Asks flight data handler to delete all flights in the database	FlightRepository.java	OK	void

4.3.2 FlightData Handler

Endpoint	HTTP Method	Description	Request Object	Response Status	Response Object
/api/airportDataController/updateAllAirports	GET	Updates all airports in the database. WARNING: Very large data request for API/	List<Airport.java>	OK	void

		API doesn't recognize all airport codes. Works properly but behavior is broken by API and system's capabilities.			
/api/airportDataController/updateSelectedAirport/{iata}	GET	Updates chosen airport on both Arrivals and Departures repositories	Airport.java	OK	void
/api/airportDataController/getAllAirports	GET	Gets all airports with their data	List<AirportResponse.java>	OK	List<AirportResponse>
/api/airportDataController/getAllAirports/getAirportByCode/{iata}	GET	Gets specific requested airport for display.	AirportResponse.java	OK	Airport Response
/api/flightController/updateAllStates	GET	Updates all the current flights in the database. If there's flights older than a day, these are deleted while current ones are updated with new data.	Flight.java	OK	Void
/api/flightController/getAllStates	GET	Returns a list of all current Flight Objects	List<FlightResponse.java>	OK	List<FlightResponse>
/api/flightController/getFlightByCallsign/{callsign}	GET	Gets a specific flight format including departure and arrival data	FlightDataResponse.java	OK	FlightDataResponse
/api/flightController/deleteAllStates	DELETE	Deletes all flights in the database	FlightRepository.java	OK	void

4.3.3 Notifications Handler

Endpoint	HTTP Method	Description	Request Object	Response Status	Response Object
/api/notifications-handler/flightEvent/delayed	POST	Mentod notifies that a flight has been delayed	FlightDelayedEvent.java	OK	Void
/api/notifications-handler/flightEvent/landed	POST	Method notifies that a flight has landed	FlightLandedEvent.java	OK	Void
/api/notifications-handler/flightEvent/takenoff	POST	Method notifies that a flight has taken off	FlightTakenoffEvent.java	OK	Void

4.3.4 Reports Handler

Endpoint	HTTP Method	Description	Request Object	Response Status	Response Object
/api/reports-handler/reports/airports	POST	Creates an Airport Report for the solicited airport.	AirportReportRequest.java	CREATED	Void
/api/reports-handler/reports/airports/{reportId}	GET	Gets a specific airport report by its id	AirportReport.java	DELETED	AirportReportResponse.java
/api/reports-handler/reports/airports/count	GET	Counts all the reports stored	AirportReport.java	OK	
/api/reports-handler/reports/airports	GET	Gets a list of all the airport reports.	AirportReport.java	GET	List<AirportRespose>
/api/reports-handler/reports/airports/iata/{iata}	GET	Finds a list of reports according to an airport's IATA code.	AirportReport.java	GET	List<AirportReportResponse>
/api/reports-handler/reports/airports/iata	GET	Gets the amount of all reports in the database	AirportReport.java	GET	Long

/{iata}/count		according to an IATA code.			
/api/reports-handler/reports/flights	POST	Sets the flightReportHandlerService to store a FlightReportRequest	FlightReportRequest.java	CREATED	Void
/api/reports-handler/reports/flights{reportId}	GET	Gets a flight report according to a report's ID.	FlightReport.java	OK	FlightReportResponse.java
/api/reports-handler/reports/flights/delete	DELETE	Deletes a report according to its ID.	FlightReport.java	OK	Void
/api/reports-handler/reports/flights/count	GET	Counts all the flight reports in the system.	FlightReport.java	OK	Long
/api/reports-handler/reports/flights/get	GET	Gets all the reports in the system.	FlightReport.java	OK	List<FlightReportResponse>
/api/reports-handler/reports/flights/callsign/{callsign}	GET	Gets a list of all flight reports according to a specific callsign (flight).	FlightReport.java	OK	List<FlightReportResponse>
/api/reports-handler/reports/flights/callsign/{callsign}/count	GET	Gets the amount of reports declared towards a specific callsign (flight).	FlightReport.java	OK	Long

4.3.5 Subscription Handler

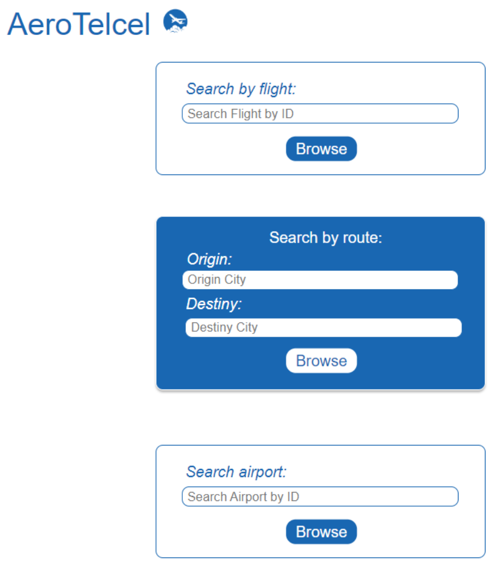
Endpoint	HTTP Method	Description	Request Object	Response Status	Response Object
/api/subscription-handler/subscriptions/determine-subscription	POST	Determines whether email is subscribed or not	FindSubscriptionRequest.java	OK	boolean
/api/subscription-handler/subscriptions	POST	Subscribes to Flight/Airport	SubscriptionRequest.java	CREATED	void
/api/subscription-handler/subscriptions/unsubscribe	DELETE	Unsubscribes from Flight/Airport	UnsubscriptionRequest.java	OK	void
/api/subscription-handler/subscriptions/unsubscribe-from-all	DELETE	Removes all Subscriptions	UnsubscriptionFromAllRequest.java	OK	void

4.3.6 Subscription Redis

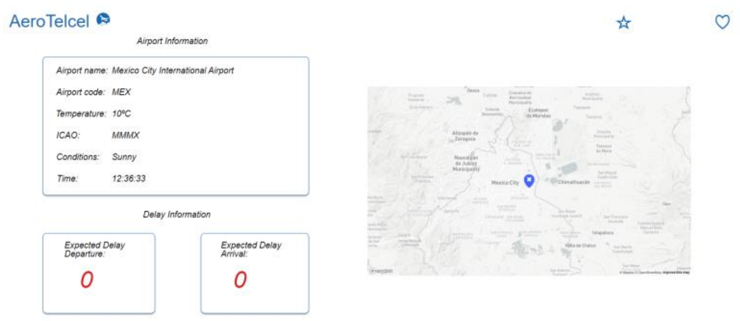
Endpoint	HTTP Method	Description	Request Object	Response Status	Response Object
/api/subscription-redis/subscriptions/POST	POST	Creates a subscription request	SubscriptionRequest.java	CREATED	Void
/api/subscription-redis/subscriptions/GET	GET	Gets a list of all subscriptions	AviationDataSubscription.java	OK	List<AviationDataSubscriptionsResponse>
/api/subscription-redis/subscriptions /{aviationDataID}	GET	Gets a list of all subscriptiosn related to one flight.	AviationDataSubscription.java	OK	AviationDataSubscriptionResponse
/api/subscription-redis/subscriptions/determine-subscription	POST	Find a subscription request	FindSubscriptionRequest.java	OK	boolean
/subscriptions/determine-subscription/subscriptions/find	POST	Finds a subscription by	Subscription.ja	OK	Subscription

-subscription		using a FindSubscription Request	va		Response
/subscriptions/determine-subscription/subscriptions/{aviationDataID}	DELETE	Delete a subscription according to an aviation data ID, or callsign	Subscription.java	OK	Void
/subscriptions/determine-subscription/subscriptions/unsubscribe	DELETE	Asks for an unsubscription request	UnsubscriptionRequest.java	OK	Void
/subscriptions/determine-subscription/subscriptions/unsubscribe-from-all	DELETE	Unsubscribes from all flights	UnsubscriptionFromAllRequest.java	OK	Void
/subscriptions/determine-subscription/subscriptions/DELETE	DELETE	Deletes all subscriptions		OK	Void

4.4 Screenshots of the Application



This is the first page a user will see when the user clicks on the application. Here it can observe 3 different browsers. The first browser is the search by flight number which is unique per flight. The second browser is searching for the flights from an airport to another airport with their respective codes, it returns a list of flights. Finally, the last browser searches the airports by their codes.



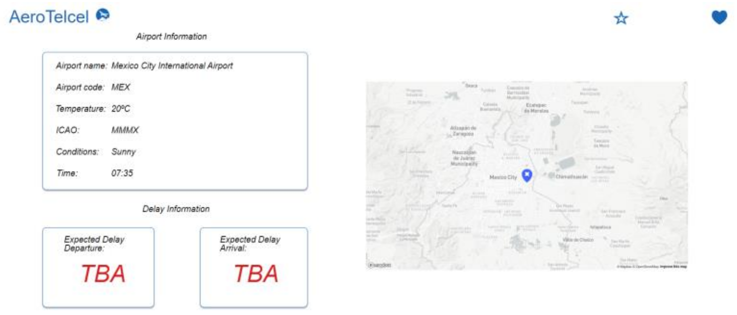
AeroTelcel

Airport Information

Airport name:	Mexico City International Airport
Airport code:	MEX
Temperature:	10ºC
ICAO:	MMMX
Conditions:	Sunny
Time:	12:36:33

Delay Information

Expected Delay Departure:	Expected Delay Arrival:
0	0



AeroTelcel

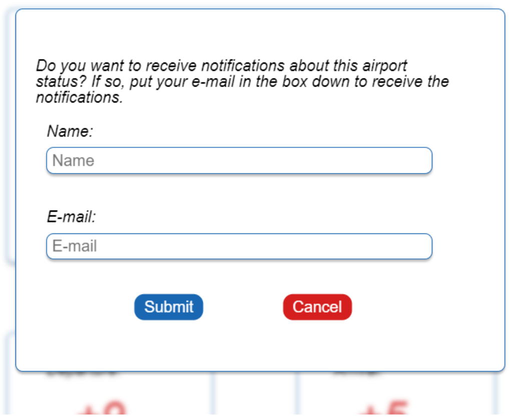
Airport Information

Airport name:	Mexico City International Airport
Airport code:	MEX
Temperature:	20ºC
ICAO:	MMMX
Conditions:	Sunny
Time:	07:35

Delay Information

Expected Delay Departure:	Expected Delay Arrival:
TBA	TBA

These two images represent the airport information page, here you can see the information that we provide to the user. It gives the airport name, code, temperature, ICAO (another code), the current conditions, the time, and the expected delays. You can also see the map on which there is a mark at the exact position of the airport. This information is displayed since it receives the airport code from the previous site and does a GET method from the flight handler.



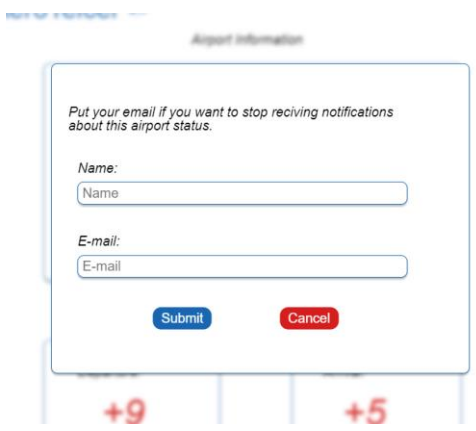
Do you want to receive notifications about this airport status? If so, put your e-mail in the box down to receive the notifications.

Name:

Name

E-mail:

E-mail



Put your email if you want to stop reciving notifications about this airport status.

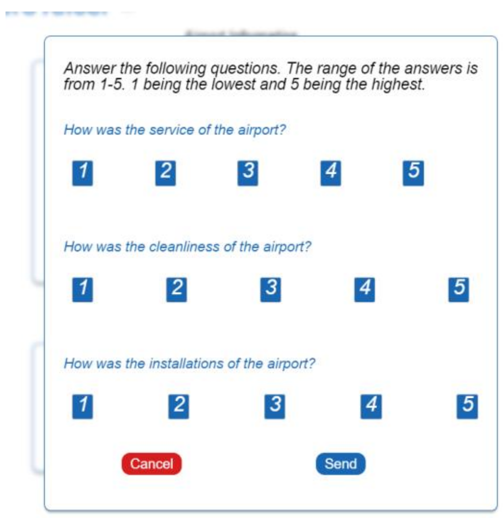
Name:

Name

E-mail:

E-mail

These two images are the messages that the user sees after clicking the heart in the airport pages, each heart has its message to subscribe to the airport or unsubscribe from it. Here we recollect the data input by the user and add the airport code that passes from the airport page to commit a POST to the subscription handler.



Answer the following questions. The range of the answers is from 1-5. 1 being the lowest and 5 being the highest.

How was the service of the airport?

1 2 3 4 5

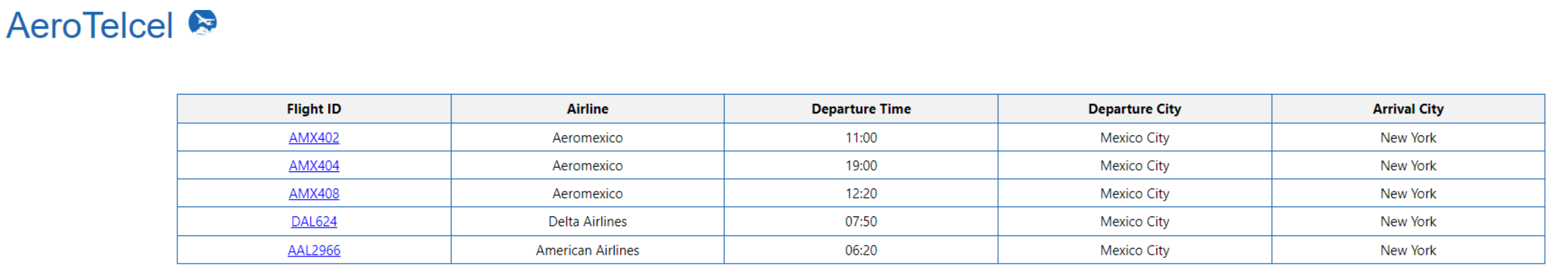
How was the cleanliness of the airport?

1 2 3 4 5

How was the installations of the airport?

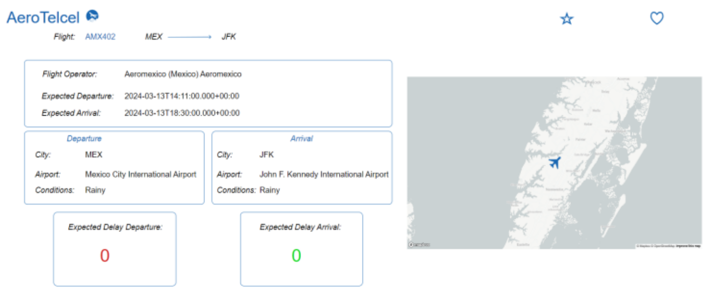
1 2 3 4 5

This questionnaire is obtained after clicking the star on the airport page. Here we receive the airport code from the airport page and make a questionnaire in order to see the satisfaction of the users with the airport. Here we make a POST method to the reports handler sending the results of the questionnaire.



Flight ID	Airline	Departure Time	Departure City	Arrival City
AMX402	Aeromexico	11:00	Mexico City	New York
AMX404	Aeromexico	19:00	Mexico City	New York
AMX408	Aeromexico	12:20	Mexico City	New York
DAL624	Delta Airlines	07:50	Mexico City	New York
AAL2966	American Airlines	06:20	Mexico City	New York

When the user does not have the information at hand of the flight code, it can search with the airport codes from departure and arrival, in the image we see information displayed in order for the user to choose which flight is the one he wants. It can click on the flight on this page, and it re-directs to the flight page.



Flight: AMX402 MEX → JFK

Flight Operator:	Aeromexico (Mexico) Aeromexico
Expected Departure:	2024-03-13T14:11:00.000+00:00
Expected Arrival:	2024-03-13T18:30:00.000+00:00

Departure

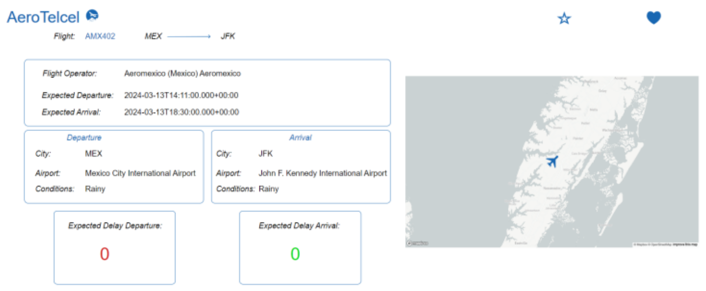
City:	MEX
Airport:	Mexico City International Airport
Conditions:	Rainy

Arrival

City:	JFK
Airport:	John F. Kennedy International Airport
Conditions:	Rainy

Expected Delay Departure: 0

Expected Delay Arrival: 0



Flight: AMX402 MEX → JFK

Flight Operator:	Aeromexico (Mexico) Aeromexico
Expected Departure:	2024-03-13T14:11:00.000+00:00
Expected Arrival:	2024-03-13T18:30:00.000+00:00

Departure

City:	MEX
Airport:	Mexico City International Airport
Conditions:	Rainy

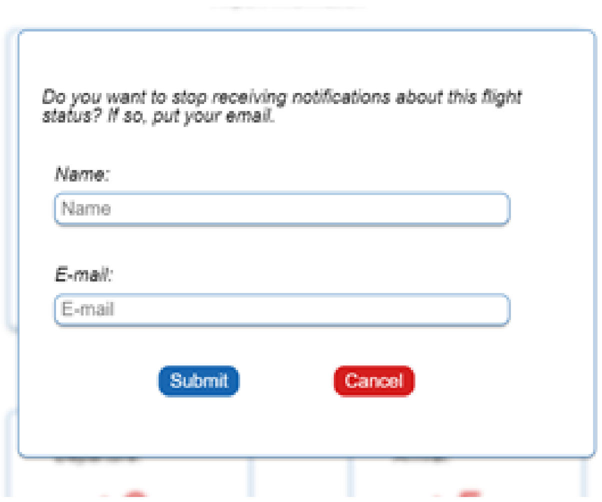
Arrival

City:	JFK
Airport:	John F. Kennedy International Airport
Conditions:	Rainy

Expected Delay Departure: 0

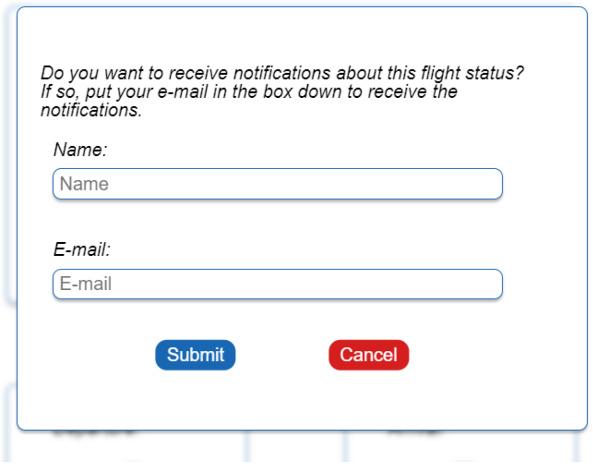
Expected Delay Arrival: 0

The two images from above are the information of the flight page. Here the user can observe the flight information such as the code, airport of departure, airport of arrival, the estimated departure and arrival, etc. As can be seen we have a map with a line between both airports simulating the route of the flight and it also has markers from the airports and a little plane for the flight position.


Do you want to stop receiving notifications about this flight status? If so, put your email.

Name:

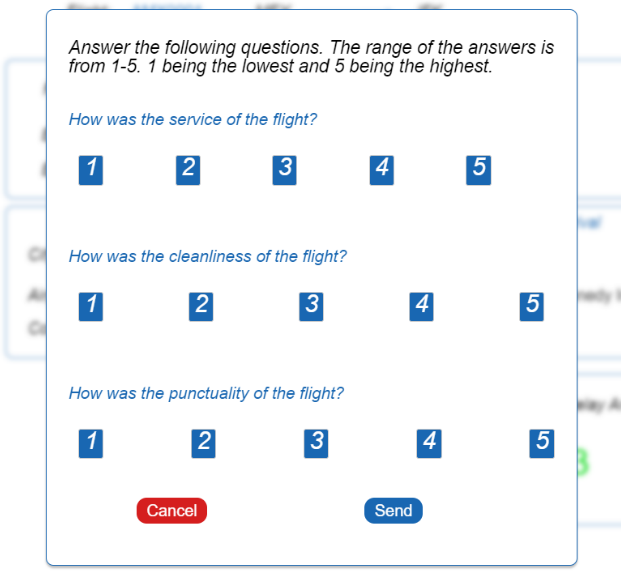
E-mail:


Do you want to receive notifications about this flight status? If so, put your e-mail in the box down to receive the notifications.

Name:

E-mail:

These two images are the messages that the user sees after clicking the heart in the flight pages, each heart has its message to subscribe to the flight or unsubscribe from it. Here we recollect the data input by the user and add the flight code that passes from the flight page to commit a POST to the subscription handler.



Answer the following questions. The range of the answers is from 1-5. 1 being the lowest and 5 being the highest.

How was the service of the flight?

How was the cleanliness of the flight?

How was the punctuality of the flight?

This questionnaire is obtained after clicking the star on the flight page. Here we receive the flight code from the flight page and make a questionnaire to see the satisfaction of the users with the flight. Here we make a POST method to the reports handler sending the results of the questionnaire.

4.5 APIs and libraries

4.5.1 Frond-end APIs and Libraries

- MapBox API was used in the front-end to put a map in the application.
- React and React-Router-Dom are the libraries used by the application in the Front-end.

4.5.2 Back-end APIs and Libraries

4.5.2.1 Development APIs and Libraries

- Open Sky API: was utilized to retrieve all flight information for active flights, departures, and arrivals. With the help of externally added databases for airports and airlines, the system was able to enhance the information retrieved by the API with additional data such as being able to assign an airline to each flight or assign each arrival and departure to its respective airport.
- Spring Framework Boot: Part of the larger Spring Framework. Spring Boot simplifies the process to build a new Spring application.
    - spring-boot-starter-web
        - Facilitates building RESTful applications by providing an MVC model to work with.
    - spring-boot-starter-data-jpa
        - Simplifies the required configuration for building applications that use JPA for interactions with databases.
    - spring-boot-starter-data-redis
        - Provides the necessary tools for building applications that interact with a Redis database.
    - spring-cloud-starter-openfeign
        - Utilized for building Web Service Clients that interact with other microservices.
- Spring Framework Cloud:
    - spring-cloud-starter-gateway
        - Provides a simple way to route requests to corresponding APIs.
- Lombok
    - Used for annotations that remove boilerplate code, as well as for logging purposes.
- Mysql Connector
    - Provides required JDBC drivers for JPA to interact with MySQL.

4.5.2.2 Testing APIs and Libraries

- Spring Framework Boot:

    - spring-boot-starter-test
        - Offers an extensive set of tools for testing Spring Boot applications.
    - spring-boot-testcontainers
        - Provides disposable instances of applications that can run in a Docker container. In our case, it is used for testing system's interactions with databases.

4.6 Application Installation and Launch Procedure

4.6.1 Frond-end

4.6.1.1 Requirements

- React.js
- React-Router-dom
    - Version 6.22.1
- Mapbox Gl
    - Version 1.13.1
- React-mapbox
    - Version 5.1.1

4.6.1.2 Configuration Instructions

- To run the application correctly, run the back-end first and check that each microservice is running smoothly. Then follow the launch procedure to initiate the application in the react server. Modifications can be made inside each JavaScript file for each page of the website.

4.6.1.3 Launch Procedure

To utilize the AeroTelcel app the user must install the following dependencies: React-React-Router-Dom and Mapbox GL. Run the following script in the terminal: "npm install".

After this, you should run all the back-end services and check out that the services are working without any problem. Then, position yourself inside the application folder by stating in the terminal:

```
cd .\Web\\aerotelcel
```

Finally, run the application script in the terminal:

```
npm start
```

4.6.2 Back-end

4.6.2.1 Default Launch Procedure (Preferred Launch Procedure)

4.6.2.1.1 Requirements

- Java Development Kit (JDK) - https://adoptium.net/temurin/
  - Version 17.0.10+7
- Docker Desktop - https://www.docker.com/products/docker-desktop/
  - Version 4.28.0
- MySQL Instance running on localhost.
  - Expected Configuration:
    - Exposed Port 3306
    - Root Password: password
- Redis Instance running on localhost.
  - Expected Configuration:
    - Exposed Port: 6379
- Python 3.9.7
  - Download Python - https://www.python.org/downloads/
- OpenSky API
  - Clone the APIs repository from https://github.com/openskynetwork/opensky-api
  - Install the API using the following command:
  - `pip install -e /path/to/repository/python`
  - Consult API documentation for further information.

4.6.2.1.2 Configuration Instructions

- Environment Variables located in the "application.properties" file for each microservice can be individually modified in case there are changes in the expected configurations.

4.6.2.1.3 Launch Procedure

- Navigate to the Aerotelcel/DevOps folder.
- Build up docker compose for databases (docker-compose-dbservices.yaml), ``docker-compose -f docker-compose-dbservices.yaml up``.
- Execute build_all.bat if in Windows, or build_all.sh if in Linux.
- Execute run_all.bat if in Windows, or run_all.sh if in Linux.

4.6.2.2 Docker-Compose Launch Procedure (Partially Functional)

4.6.2.2.1 Requirements

- Docker Desktop - https://www.docker.com/products/docker-desktop/
  - Version 4.28.0

4.6.2.2.2 Configuration Instructions

- Environment Variables for all microservices can be modified directly in the Aerotelcel/DevOps/docker-compose.yaml file.

4.6.2.2.3 Launch Procedure

- Navigate to the Aerotelcel/DevOps folder.
- Execute build_all.bat if in Windows, or build_all.sh if in Linux.
- Run command "docker-compose up --build".

4.7 Unit Test Case

Front End Unit Testing:

For the front end, there was a unit test that put into practice sending the string input by the user from the browser page to the airport page. In Appendix B.1, it can be seen the table with the Unit Test, here we perform a series of tests trying to pass the input with examples of airport codes such as 'MEX' or 'JFK' to see the result on the next page. After observing that the airport page was getting updated when the input was an airport that existed in the database we tried with an airport code that didn't. 'ZZZ' is the airport used and the string did correctly pass from one page to another, but the airport page didn't put the information making it an error. These tests were done to prove that 'useNavigate' and 'useLocation' worked by passing the String input by the user. Also, there is an image of the code with the case example 'MEX' to see that the test passed without any problem.

4.8 Stress Test Case

Logic and Database Layer Stress Testing – `doSearch()`

One of the essential methods for the overall system functionality relies on the `doSearch()` method, implemented in both Flight retrieval (ReadAllStates) and Airport retrieval (ReadAirportArrivals & ReadAirportDepartures) services. This method is in charge of reading the JSON flight or arrival/departure objects printed by the API calls: `get_states`, `get_departures_by_airport` and `get_arrivals_by_airport`, who are then read by their respective service and verified / sanitized for their proper database upload, while also adding extra data such as airline and last time updated for the case of flights, and an airport of departure / arrival for Departures and Arrivals respectively.

This method is essential, since the information received from these API calls is the backbone of the system when it comes to data. Because of this, it's essential that data is assigned properly for future display in the system and that it can handle large data input. The relevance of this data integrity got

enlarged at the realization that the API's information was occasionally incomplete or lacking essential data for our system such as the callsign of a flight, or a location of departure or arrival. Because of this, before uploading the data, it must be cleaned, verified, and managed by different sectors of the code to ensure that flights are complete, and dispose the information that gives us no usage and would waste physical space if stored.
This testing focuses on the reliability of the `doSearch()` method, which handles this data intake and verifies the information, assigns extra data (such as airports or airlines), and gets rid of data outliers and flights that were not updated recently. It is essential that this method works appropriately, since each API call can return between 7K to 20K flight objects that the system must be able to handle.

The test consists of ensuring that a flight object can be read properly, converted, assigned an airline, and sanitized, since at different calls, some pieces of essential data, like callsigns, were received with empty spaces (e.g., "`ACA1074__`"), were fully lacking one, or simply didn't belong to commercial aircrafts, which is out of focus from what the project intends to work for. In both development and testing, the system proved to be able to handle a great data intake from each API call, being able to handle proper flights, dispose the ones that proved to not be helpful, and sanitize each of the accepted ones for proper database upload. Throughout development, the method was changed in different parameters to become more effective each time it was utilized, and while its time complexity enlarged, the team chose to prioritize data integrity, also because this piece of code functions on a separate service from the one retrieving the information, enabling us to provide better data quality without affecting the end user completely.

4.9 Integration Testing

4.9.1 Subscription Lifecycle

The goal of the test is to ensure that the Subscription Redis application is capable of performing all operations required for managing Subscriptions in the Redis Database.
The test is executed with the use of the following technologies:

- SpringBoot:
  - Simplifies development of application tests by providing the option to encapsulate the application in a test environment (SpringBootTest).
- TestContiners
  - Provides a solution to customized docker containers for the duration of the test. Real instances of dependencies are used, which ensures that the application will perform as expected when finally deployed.

Together, these libraries are used to prove that the application performs as expected with its database dependency.

	Project Name:	Aerotelcel	Test Designed by:	Jorge	Ramirez de Diego
	Module/Component Name:	Subscription Redis Redis Database	Test Designed date:	08	03/2024
	Release Version:	1.0	Test Executed by:	Jorge	Ramirez de Diego
			Test Execution date:	16	03/2024
Pre-conditions:	Application Launch Dependencies are installed. redis:latest image is available in Docker. Internet connection is required for downloading the test's dependencies. Port 6379 is available. No pre-existing subscriptions in the Redis Database utilized by the test.				
Postconditions:	No postconditions. Testing environment is cleaned.				
Test Steps:	1. **Test Setup:** • Parallel Spring Boot application is executed for the purpose of the test. • Configuration of the application is dynamically overwritten from the default values.				

	• Redis Container is automatically created using the TestContainers library. **2. Test Execution:** • ***Subscription Creation***: Subscription request with dummy data for aviationDataID, name, and email is created. • ***Subscription Saving***: Subscription request is saved using the subscriptionService.saveSusbcription(request) method. This action should cause the subscription to be stored in the Redis database. • ***Subscription Retrieval***: After saving, the test retrieves the subscriptions for the provided mock aviation data ID using the subscriptionService.fetchSubscriptions(request.getAviationDataID()) method. The retrieved data is expected to contain the previously saved subscription. • ***Unsubscription***: The test then performs an unsubscribe action by calling subscriptionService.unsubscribe(request.getAviationDataID(), request.getEmail()). The subscription should now be deleted from the Redis database. **3. Test Cleanup:** • When the test is completed, Testcontainers automatically removes the Redis database container before stopping, ensuring that no data persists after the execution.
Breakout Points:	- **Failure to Retrieve:** The test asserts that after saving the subscription, there is exactly one subscription stored in the Redis database, which matches the data from the subscription that was saved. In case there are more subscriptions, or the data from the found subscription doesn't match, the test fails. - **Failure to Unsubscribe:** After performing the unsubscribe action, the test attempts to retrieve subscriptions once again, but it expects a 'NullPointerException', which would indicate that the subscription no longer exists in the Redis database. This would verify that the subscription was correctly removed. In case the subscription was located, the test fails.
Safe Points:	- Before Any Network Communication: Before initiating the server and any client requests, preventing any unwanted side effects on the network. - After Each Major Step: After each key testing step, the server and the client will be in an expected state. - Post Test Execution: After all resources are released and server is stopped, regardless of the testing outcome.

Test Case#	Test Title	Test Summary	Expected Result	Status
1	Implementation Test	Test ran to ensure that the Integration Test was implemented correctly.	Successful (OK)	Pass
n	Pre-Build Test	Test executed upon every build process of the component.		

4.10 Automated Testing

4.10.1 Gradle Build Test Automation

Integrating application tests with build processes can provide many benefits that end up improving the general system's quality. These advantages are provided in the following manner:

- Early detection of defects
    - Tests ran when building an application provide immediate feedback on the effects that the changes made have, which allows developers to quickly identify and remediate any defects that come up.
- Improved Code Quality and Reliability
    - By constantly testing the application, a certain quality is ensured. This prevents issues appearing further up in the development cycle.
    - With automated testing more extensive test coverage can be provided, as tests that might be too tedious or time-consuming can be developed once and run upon every build cycle.
- Development Efficiency
    - By reducing the time spent testing and debugging, developers can spend significantly more time designing and developing new parts of the system.

Gradle can be configured to run Spring Boot tests automatically as part of the build process. Gradle has built-in support for tests built in various testing frameworks, facilitating integration significantly.

With the currently implemented tests, only the Subscription Lifecycle integration test is performed upon building, but once more tests are created, they can be easily integrated with the application's build processes.

5. References

5.1 External Databases

- IP2Location. (2024) iata-icao.csv (1.0.13) [CSV] https://github.com/ip2location/ip2location-iata-icao
- Laboratoire Spécification et Vérification. (n.d.) airlines.txt [TXT] http://www.lsv.fr/~sirangel/teaching/dataset/airlines.txt

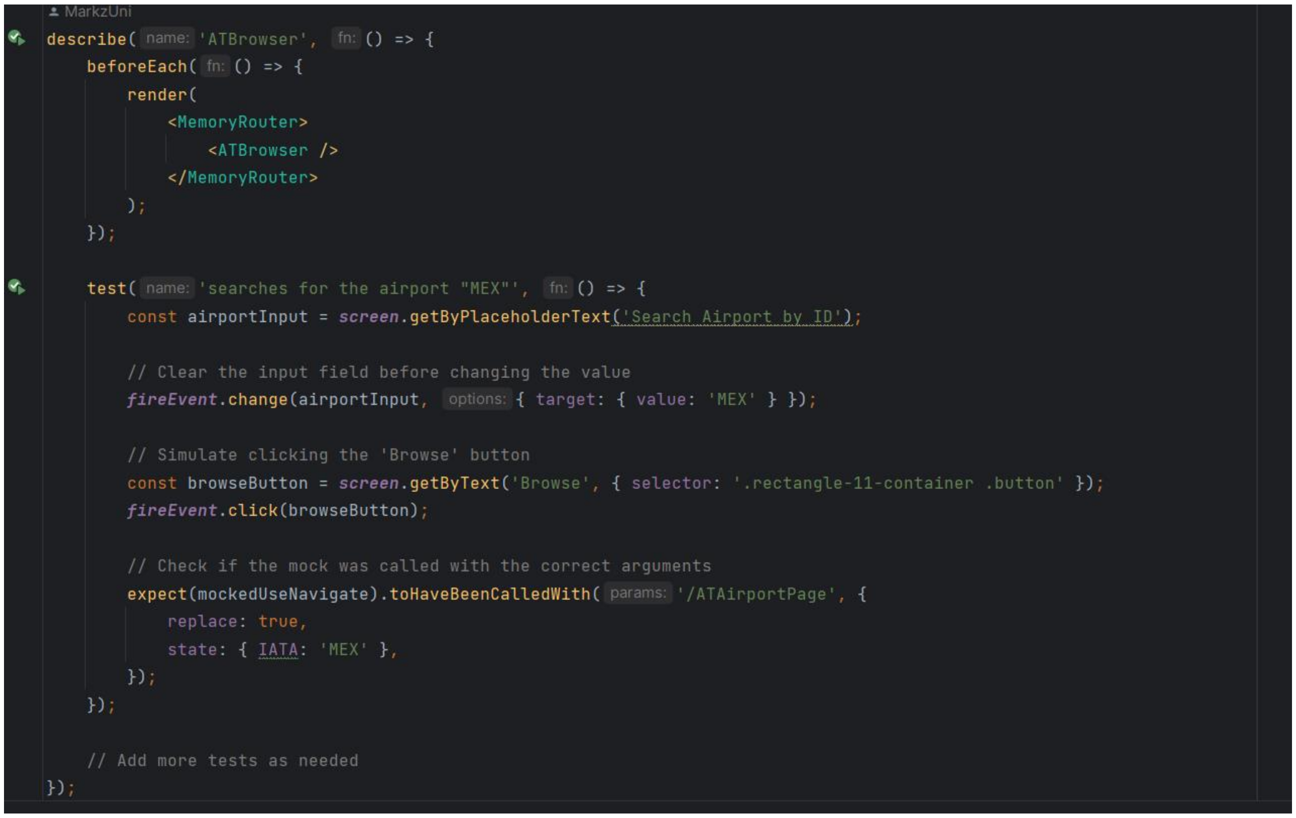
5.2 External Software Documentation

- 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
- Webb, P., Syer, D., Long, J., Nicoll, S., Winch, R., Wilkinson, A., Overdijk, M., Dupuis, C., Deleuze, S., Simons, M., Pavić, V., Bryant, J., Bhave, M., Meléndez, E., Frederick, S., & Halbritter, M. (2024). Spring Boot reference documentation (Version 3.2.4). Retrieved March 24, 2024, from https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

6. Appendices

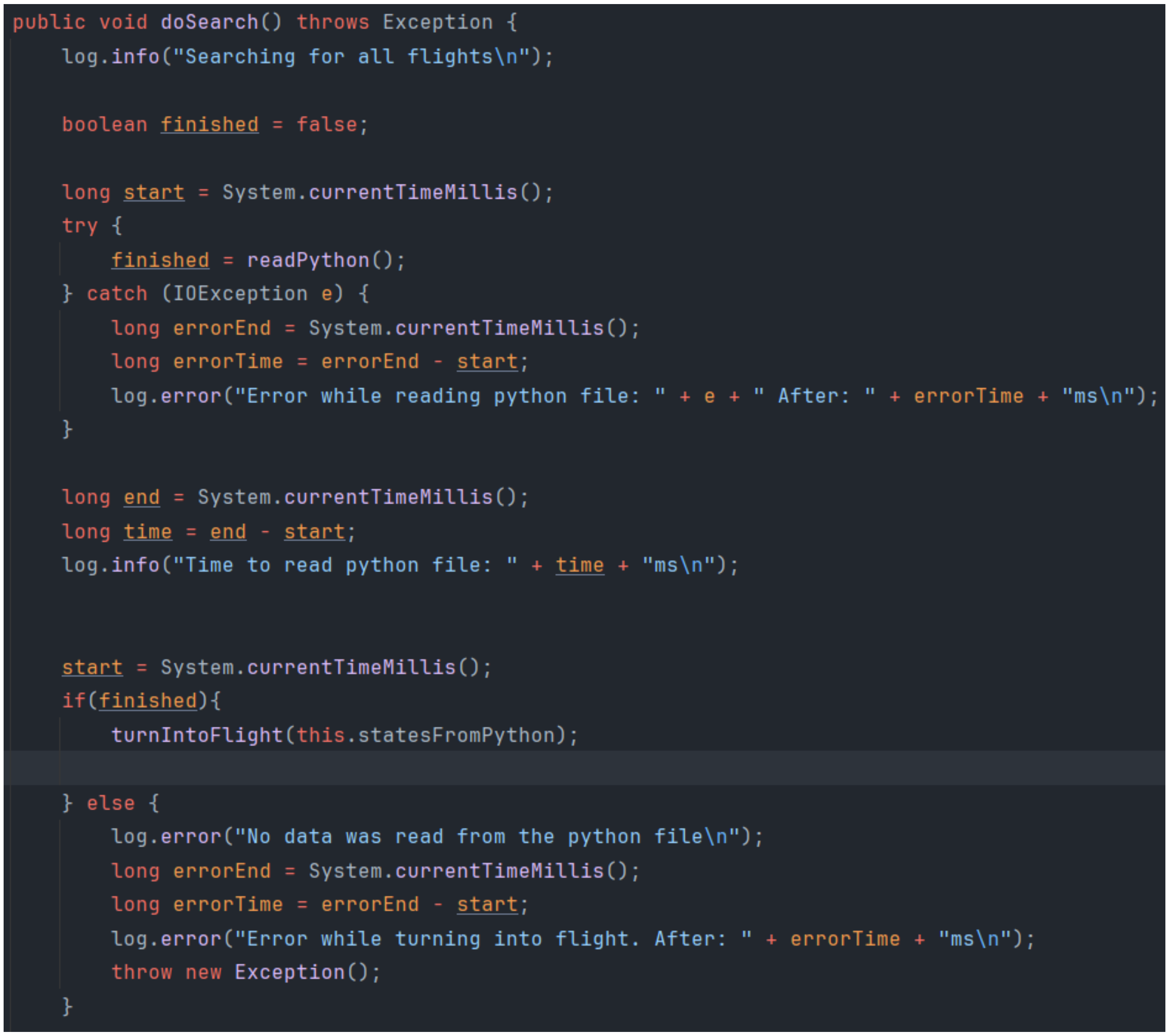
6.1 Appendix A: Source Code Snippets

6.1.1 Unit Test Front-end Image Src example with ‘MEX’

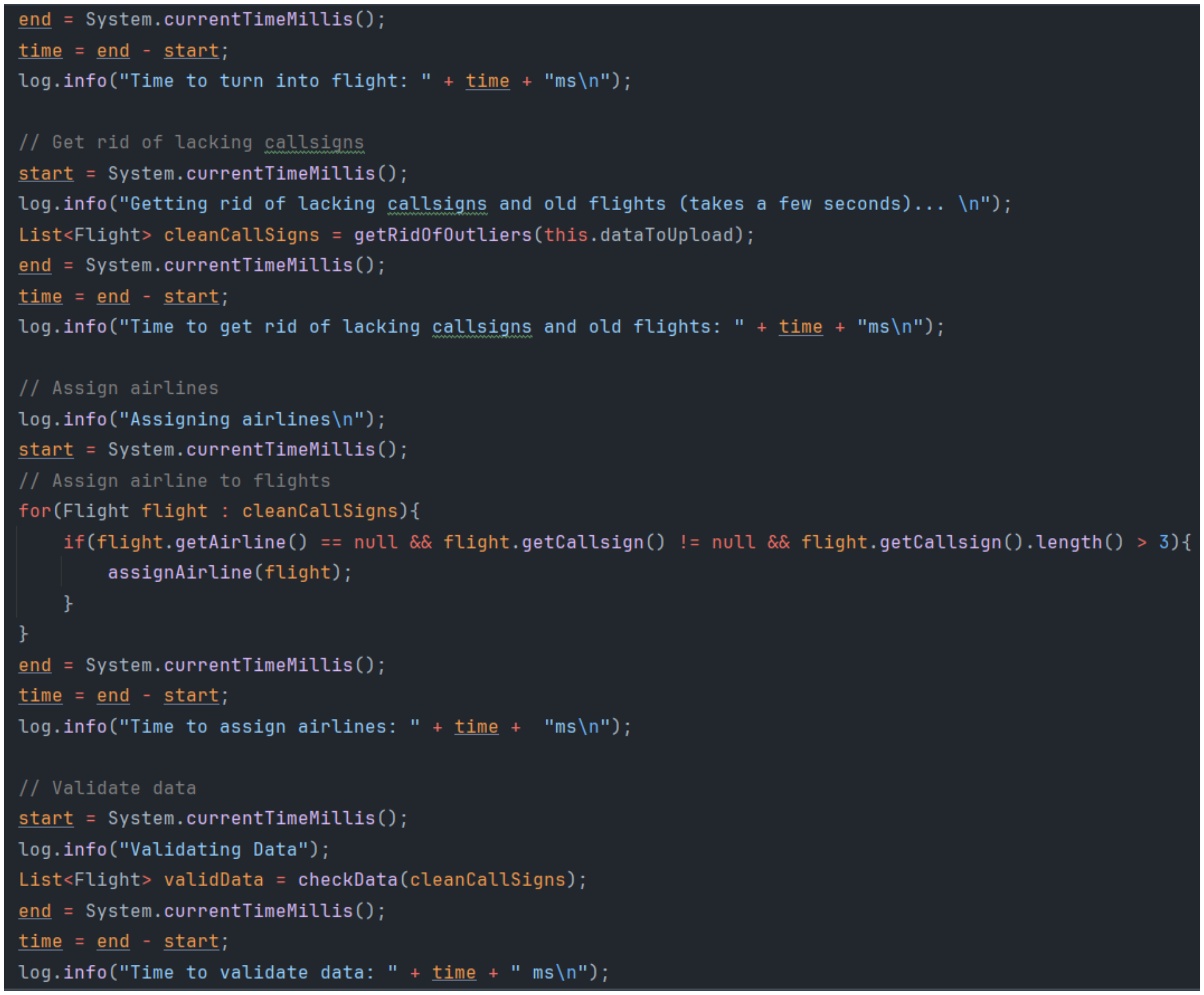


```
MarkzUni
describe( name: 'ATBrowser',  fn: () => {
    beforeEach( fn: () => {
        render(
            <MemoryRouter>
                <ATBrowser />
            </MemoryRouter>
        );
    });

    test( name: 'searches for the airport "MEX"',  fn: () => {
        const airportInput = screen.getByPlaceholderText('Search Airport by ID');

        // Clear the input field before changing the value
        fireEvent.change(airportInput,  options: { target: { value: 'MEX' } });

        // Simulate clicking the 'Browse' button
        const browseButton = screen.getByText('Browse', { selector: '.rectangle-11-container .button' });
        fireEvent.click(browseButton);

        // Check if the mock was called with the correct arguments
        expect(mockedUseNavigate).toHaveBeenCalledWith( params: '/ATAirportPage', {
            replace: true,
            state: { IATA: 'MEX' },
        });
    });

    // Add more tests as needed
});
```

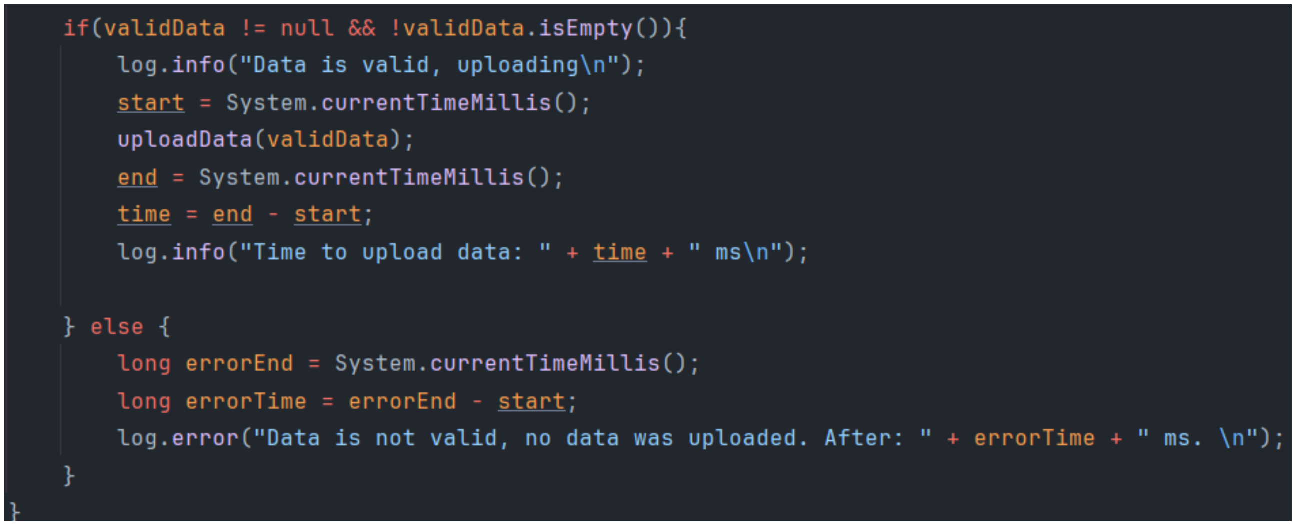
6.1.2 Unit test Logic / Database Layer Testing – doSearch()



```java
public void doSearch() throws Exception {
    log.info("Searching for all flights\n");

    boolean finished = false;

    long start = System.currentTimeMillis();
    try {
        finished = readPython();
    } catch (IOException e) {
        long errorEnd = System.currentTimeMillis();
        long errorTime = errorEnd - start;
        log.error("Error while reading python file: " + e + " After: " + errorTime + "ms\n");
    }

    long end = System.currentTimeMillis();
    long time = end - start;
    log.info("Time to read python file: " + time + "ms\n");


    start = System.currentTimeMillis();
    if(finished){
        turnIntoFlight(this.statesFromPython);

    } else {
        log.error("No data was read from the python file\n");
        long errorEnd = System.currentTimeMillis();
        long errorTime = errorEnd - start;
        log.error("Error while turning into flight. After: " + errorTime + "ms\n");
        throw new Exception();
    }
```

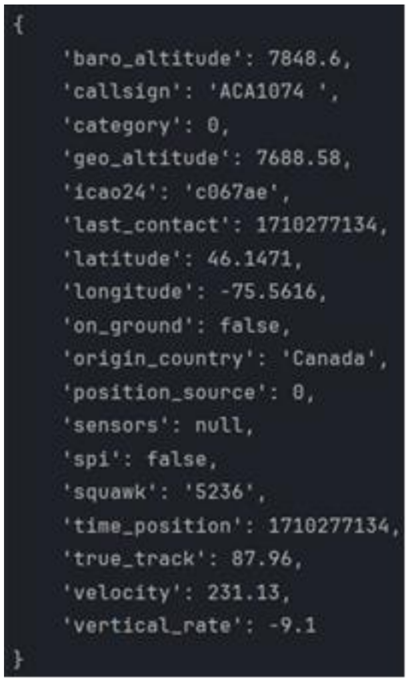


```
end = System.currentTimeMillis();
time = end - start;
log.info("Time to turn into flight: " + time + "ms\n");

// Get rid of lacking callsigns
start = System.currentTimeMillis();
log.info("Getting rid of lacking callsigns and old flights (takes a few seconds)... \n");
List<Flight> cleanCallSigns = getRidOfOutliers(this.dataToUpload);
end = System.currentTimeMillis();
time = end - start;
log.info("Time to get rid of lacking callsigns and old flights: " + time + "ms\n");

// Assign airlines
log.info("Assigning airlines\n");
start = System.currentTimeMillis();
// Assign airline to flights
for(Flight flight : cleanCallSigns){
    if(flight.getAirline() == null && flight.getCallsign() != null && flight.getCallsign().length() > 3){
        assignAirline(flight);
    }
}
end = System.currentTimeMillis();
time = end - start;
log.info("Time to assign airlines: " + time +  "ms\n");

// Validate data
start = System.currentTimeMillis();
log.info("Validating Data");
List<Flight> validData = checkData(cleanCallSigns);
end = System.currentTimeMillis();
time = end - start;
log.info("Time to validate data: " + time + " ms\n");
```



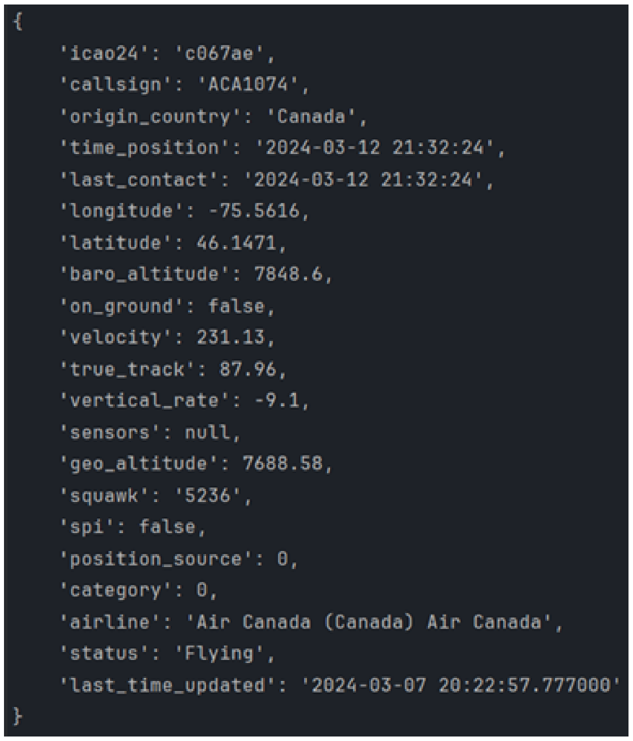
```
    if(validData != null && !validData.isEmpty()){
        log.info("Data is valid, uploading\n");
        start = System.currentTimeMillis();
        uploadData(validData);
        end = System.currentTimeMillis();
        time = end - start;
        log.info("Time to upload data: " + time + " ms\n");

    } else {
        long errorEnd = System.currentTimeMillis();
        long errorTime = errorEnd - start;
        log.error("Data is not valid, no data was uploaded. After: " + errorTime + " ms. \n");
    }
}
```

6.1.3 doSearch() expected flight object format when object is received



```
{
    'baro_altitude': 7848.6,
    'callsign': 'ACA1074 ',
    'category': 0,
    'geo_altitude': 7688.58,
    'icao24': 'c067ae',
    'last_contact': 1710277134,
    'latitude': 46.1471,
    'longitude': -75.5616,
    'on_ground': false,
    'origin_country': 'Canada',
    'position_source': 0,
    'sensors': null,
    'spi': false,
    'squawk': '5236',
    'time_position': 1710277134,
    'true_track': 87.96,
    'velocity': 231.13,
    'vertical_rate': -9.1
}
```

6.1.4 doSearch() expected final flight object format



```
{
    'icao24': 'c067ae',
    'callsign': 'ACA1074',
    'origin_country': 'Canada',
    'time_position': '2024-03-12 21:32:24',
    'last_contact': '2024-03-12 21:32:24',
    'longitude': -75.5616,
    'latitude': 46.1471,
    'baro_altitude': 7848.6,
    'on_ground': false,
    'velocity': 231.13,
    'true_track': 87.96,
    'vertical_rate': -9.1,
    'sensors': null,
    'geo_altitude': 7688.58,
    'squawk': '5236',
    'spi': false,
    'position_source': 0,
    'category': 0,
    'airline': 'Air Canada (Canada) Air Canada',
    'status': 'Flying',
    'last_time_updated': '2024-03-07 20:22:57.777000'
}
```

6.2 Appendix B: Test Tables

6.2.1 Appendix B.1: Unit Test Front-end Table

Unit Test Case

Project Name:	**Aerotelcel**	**Test Designed by:**	**Marcos**	
Module/Component Name:	**Front-end Image Src**	**Test Designed date:**	12.03.2024	
Release Version:	**3.0**	**Test Executed by:**	**Marcos**	
		Test Execution date:	12.03.2024	

Pre-condition	Started the flight_handler, write the airport code.
Postconditions:	Go to the next page and update the information with the airport code.

Test Case#	Test Title	Test Summary	Test Steps	Test Data	Expected Result	Status (Pass/Fail)	Notes (if any)
1	Write airport 'MEX'	Write this airport and update the information of the page	'MEX' is written, the browser button is clicked and the information of the IATA is passing to the next page updating the information.	IATA: 'MEX'	The IATA shall pass and update correctly the information.	PASS	N/A
2	Write airport 'JFK'	Write this airport and update the information of the page	'JFK' is written, the browser button is clicked and the information of the IATA is passing to the next page updating the information.	IATA: 'JFK'	The IATA shall pass and update correctly the information.	PASS	N/A
3	Write airport 'ZZZ'	Write this airport and update the information of the page	'ZZZ' is written, the browser button is clicked and the information of the IATA is passing to the next page but there is no airport found so it triggers an error.	IATA: 'ZZZ'	The airport should not be found making the test fail.	FAIL	N/A

6.2.2 Appendix B.2: Unit test Logic / Database Layer Testing – doSearch()

Stress Test Case

Project Name:	**Aerotelcel**	**Test Designed by:**	**Javier Ortega**	
Module/Component Name:	**Flightdata_handler**	**Test Designed date:**	12.03.2024	
Release Version:	**3.0**	**Test Executed by:**	**Javier Ortega**	
		Test Execution date:	12.03.2024	

Pre-condition	Run ReadAllStates Service and make a call for doSearch() method
Postconditions:	Deliver prepared Flight data object for upload, flight object should include airline, updated and formatted timestamps, status added and last time updated timestamp.

Test Case#	**Test Title**	**Test Summary**	**Test Steps**	**Test Data**	**Expected Result**	**Status (Pass/Fail)**	**Notes (if any)**
1	Single Flight Object Test	Verify the proper formatting of one flight data object	Make doSearch() read the single flight data (Appendix A 5.1.3), wait for doSearch to complete data information and display ready flight object for upload	Input done by hand, **Appendix 5.1.3**	A resulting full flight object must represent what appears in Appendix **5.1.4**	PASS	
2	One get_states() API call	Verify the method's effectiveness with a large dataset	Make doSearch() read the large dataset given by get_states Let doSearch assign, verify and sanitize data and display deleted flights, dropped out ones and show how many are ready for upload.	Call to get_states from OpenSky API	A log representing "All flights siccessfuly managed and ready for upload"	PASS	
3	Test app-level behaviour	Verify functionality in being able to delete old flights while also handling a large dataset	Perform the same behaviour as test 2, but in this case, verify doSearch()'s functionality at dropping out old data or updating another flight with new data	- Call to get_states from OpenSky API - Previous old data stored in database	A log representing "All flights siccessfuly managed and ready for upload" Also, logs for older flight deletion during managing	PASS	While the test doesn't run into any failure, the time taken to perform the task grows greatly since it looks up in the database for older callsigns