***Functional Specification:***

* **Preface:**

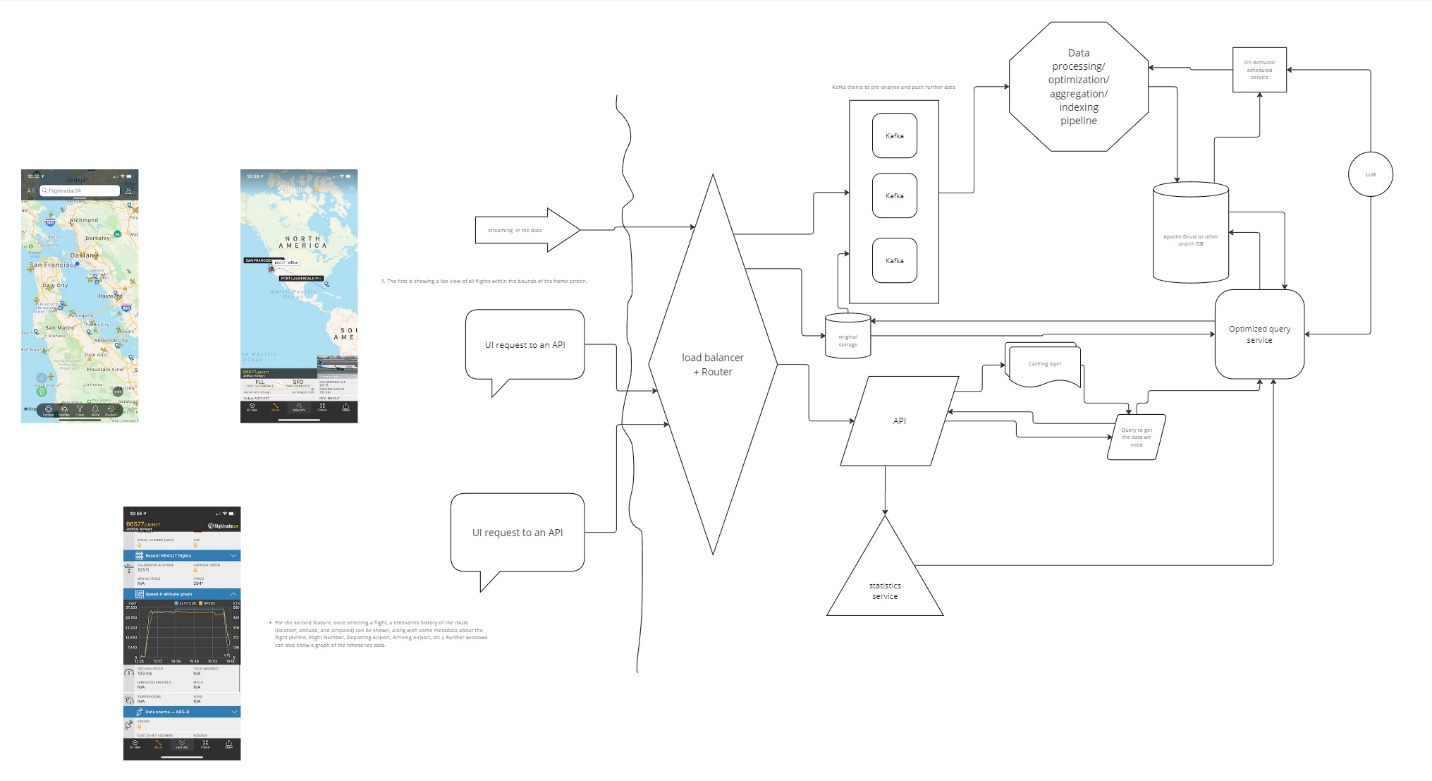
Design 2 features in regards to the tracking of all the airplane flights in the world:

* The first is showing a live view of all flights within the bounds of the home screen.
* For the second feature, once selecting a flight, a timeseries history of the route (location, altitude, and airspeed) can be shown, along with some metadata about the flight (Airline, Flight Number, Departing Airport, Arriving Airport, etc.). Further windows can also show a graph of the timeseries data.
* **Description:**

As an input, we assume a stream of live data is available for every flight airborne around the world and is providing a sample per flight **once every 5 seconds**. All of the data needed is in those streams. As the output, an **HTTP API** that the UI polls/pulls from is sufficient. External customers/clients must be able to access the data without much lag (return in 10's or 100's of milliseconds).

* **Design solution:**

Please, look at the design diagram in pdf for more details (or here):



The flow should as follows:

From the data side: the data enters the cloud through the load balancer + router and goes into 2 places at the same time. One place is an Elasticsearch and the other is a start of the ETL pipeline with Kafka clusters used as data stream navigation. Kafka allows some pre-processing of the data which can be handy here. Next data goes into the service to index/process/aggregate the data as we want/need. And finally data hits the Druid database.

From the client side: the request in both cases hits the load balancer + router first (assuming the authentication service has been already worked out) and goes to the API processing service. From there depending on the feature we are going to either directly get the list of the flights in a specified region or go and pull statistics as well as the data on a single flight with the specified ID.

* **APIs:**

1. *GET /api/flights*:

Request:

GET /api/flights

Body: scale, coordinates (4 points of the screen frame)

Filters: None by default or empty list (depends on the backend language of choice)

Response:

flights = [

{id, airline, flight number, departing airport, arrival airport, departure/arrival time},

…

]

A GET request to obtain a list of all the flights within the targeted area on the map. There should be mapping location provided as rectangle with corresponding parameters of scaling to calculate the area to show the flights. We naturally need to provide both inbound and outbound flights.

Need to have filtering options:

* Inbound/outbound
* Within time of leave/arrival (time frame other that default say 30 minutes from current timestamp)
* Target city/From city depending on inbound/outbound
* …

1. *GET /api/flights/[flight\_id]*:

Request:

GET /api/flight/flight\_id

Response:

flight = {

Airline

Flight number

Departing airport

Arrival airport

Speed\_and\_altitude = [

Speed, altitude, time

…

]

}

A GET request to obtain information of a flight by its’ ID – would suggest the airline + number of the flight. Here we return the full info on the flight if specified a correct ID. If the ID is incorrect – need to display a proper message with the suggestion to check the airline code and number.

We are to return the following info:

* Airline;
* Flight number;
* Departing airport;
* Arrival airport;
* The graph data for speed and altitude;
* …
* **Technology & Choises:**

For the backend services I’d choose a Go/Rust/C/C++ - lightweight, fast, low level, easily distributed across platforms code. For data science and RAGs Python stays unbeatable so far, but we might opt out for Golang/C++ in pursue of faster performance.

For the APIs I’d use JSON as request/response format – I think it will be especially handy in case of the second feature since we are returning a lot of data and JSON is easy and straight forward to navigate and access.

For the model I’m thinking of comparing **GPT-4o** vs **Llama 3 8B** vs **Gemini 1.5** (according to this comparison chart <https://artificialanalysis.ai/leaderboards/models> these models look promising).

When thinking about the datastore Apache Druid looks like a perfect match to me. It is literally created for the real-time analytics. It’s solving the task by having pre-calculated buckets like Splunk does. I would also like to back up with Elasticsearch (or similar in the GCP/Azure cloud). We do need that backup data in the pure form for many reasons: logging, running and re-running ETL, re-running indexing and such. We might want to have SQL DB somewhere in the picture for simple data retrieval. And even though it’s a data duplication which is advised to be avoided, the basic data that is not written as often, but read a lot might require such a solution.