SAED Assignment Report

By: Mohammed Al-Asadi

Multithreading solution

The problem statement required airports to give out flight requests and needed planes to fly between airports and get serviced. Multithreading was used in the Airport and Plane classes to facilitate this.
The Airport class was responsible for the creation of flight requests. These were held in a blocking queue to stop any potential race conditions or deadlock situations with multiple threads accessing the queue simultaneously.
The plane class is responsible for all the movement of the planes and handles the plane service functionality that triggers upon landing, making the plane thread sleep for a random amount of time.

Threads are used in the Airport class to have a stream of flight requests be run constantly across all airports. Threads are also used on all individual planes to allow all aircraft in the simulation to function independently of each other and be able to fly from airport to airport. Servicing is managed in the plane class to avoid resource issues by allowing all planes to call the plane service script when needed.

The planes have a currentAirport variable that changes upon landing at a new airport. By accessing this variable, they gain access to that airports blocking queue to get flight requests sent out to all the other airports.

The simulation ends through using the thread pool's function threadPool.shutdownNow() to end all threads where they are.

Flight Visualisation

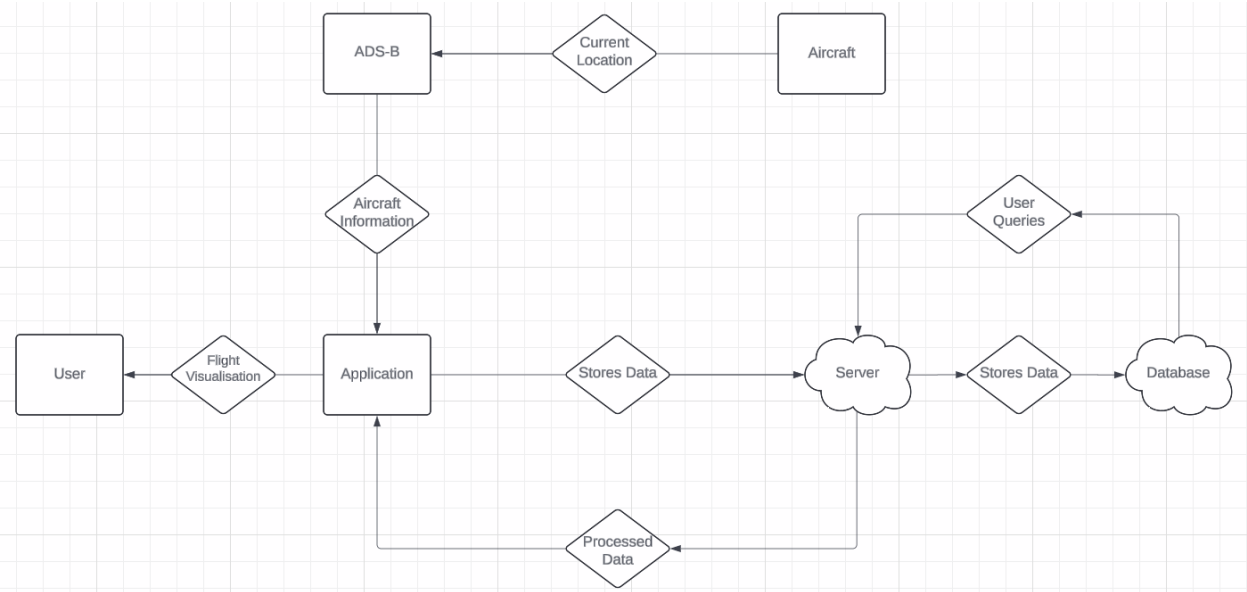
A modified version that uses real time data would be much harder to implement. The main issue for this center around the use of real-time data. It would require use of publicly available surveillance broadcast data or taking data from already existing flight trackers light FlightRadar24. This would be able to track most aircraft due to regulations mandating the use of aircraft to be equipped with ADS-B (Automatic Dependent Surveillance-Broadcast). Issues faced with this approach are potential latency delays due to needing to get aircraft GPS information transmitted to a

ground station which will then upload real time location data that can be used in the application. The latency would be made worse if information is pulled from a flight tracker instead as that would add an extra step in the process before data can be pulled into the application.
To allow the data to be viewed from a mobile app a There would need to be either a new UI created or have the current UI be adjusted to be scalable across different platforms and screen sizes. Also, due to the ease of access of a mobile app, the system would need to be scaled to handle many concurrent users simultaneously.
To allow users to visualise past flights dating back to a month maximum the application would need to be adjusted to store large amounts of flight data and would likely need servers to store the information. Query functionality would also need to be implemented to allow users to search specific flights or time frames.

Non-Functional Requirements

- System must be able to handle hundreds of thousands of concurrent users simultaneously.
- The system must be able to input hundreds of thousands of flight updates simultaneously.
- Flight data shown must be within at most a 3 second delay.
- The system must be able to store data of hundreds of thousands of different flights dating back to a month.
- Mobile app must be usable across different devices.

Architectural Approach



Aircrafts in flight transmit their GPS data to an ADS-B which is then taken directly into the application. Application sends the data to a cloud server which handles user load and data processing. The servers send back requested data which can range from a live global feed to requested dates or previous flights. All flight data in the server is put into a cloud database where it is held for a month. This data can be queried by the server before being sent to the application which presents the data to the user.

This method would lessen delay through taking direct data transmission from an ADS-B as opposed to a flight tracker which would add additional latency on displayed flight data.

Application can handle large amounts of users globally through its scalable server and database which can be adjusted to user demand. Scalable server is also capable of inputting large amounts of different flight updates simultaneously scaling depending on current air traffic.

A Scalable database would allow for thousands of flights to be saved dating back a month.

Server can send a visualisation specific for mobile output to allow for app access across a large number of different devices.