**Flight Booking & Management**

Our project consists of two separate APIs designed to efficiently organize flight reservation system management and user operations: the Admin API and the User API.

**Admin API**

The Admin API serves as an interface for managing and reporting flights. The admin can add new flights by specifying the departure and destination locations, the date range, and the days on which flights will operate. Additionally, the admin can set the capacity for each flight. A key feature is that flights with a capacity of 0 are excluded from reports, ensuring that only available flights are displayed. The Admin API also allows flight reports to be generated, and the flights can be queried based on a date range.

**User API**

The User API allows users to list available flights based on their desired departure and destination locations as well as the date range. Once a flight is selected, the capacity of that flight decreases by one. However, the user’s flight is not considered valid until the check-in process is completed. After purchasing the flight, the user can complete the check-in process using the ticketId and successfully finalize the ticket purchase.

**Paging and Performance**

Both the Admin and User APIs implement paging to handle large datasets efficiently. Paging allows users and admins to enter the page number and page size, which results in only the desired amount of data being fetched. This prevents unnecessary data load, ensuring efficient system performance and enhancing the user experience.

**TicketId and Uniqueness**

Users are allowed to generate their own ticketId. If the ticketId has already been used by another user, the system notifies the user that the ticketId is already taken and prompts them to choose a different one. This ensures that each user has a unique ticketId, maintaining the integrity of the system.

**Security**

For security purposes, authentication is applied to admin operations. Admins must authenticate with a username and password before being allowed to add or report flights. This increases the security of the application and ensures that only authorized users can perform admin tasks.

**Assumptions and Issues**

An important assumption in the design is that the flight capacity is updated correctly and that users can only access flights with available capacity. Additionally, users can only complete their check-in process after purchasing the ticket, which ensures the accuracy and security of the flight reservation process. Another key consideration was ensuring the accuracy of the flight dates and times and offering suitable time slots to users.

**Testing and Integration**

The project was tested in a Swagger environment, allowing both users and admins to easily test the API endpoints. Swagger provides an interactive tool for viewing and testing all the functionalities of the APIs. Operations like adding flights, listing flights, making reservations, and completing check-ins were verified through this environment.

**Database and Cloud Infrastructure**

The database used for the project is MongoDB. MongoDB’s flexible and scalable architecture allows for fast storage and processing of data. MongoDB has been efficiently used in this project to store flight data, user information, and ticket details.

Additionally, AWS(Amazon Web Services) was utilized for hosting the database and APIs. AWS provides a secure and scalable environment for the project’s cloud infrastructure. However, due to configuration challenges, I was unable to directly test the system on AWS. Despite this, the hosting process on AWS has been successfully completed, and with the correct configurations, the system is expected to function fully.

**Conclusion**

This design effectively organizes the management of flights and user operations, ensuring a clear division of roles between admins and users. The system offers secure and accurate ticketing processes for users while providing admins with the ability to manage flights in a secure manner.