Final Integrated Travel Booking System Proposal

Project Overview

This project is a scalable travel booking system similar to EgyptAir, allowing users to search flights, book tickets, manage bookings, and receive notifications. The system uses a microservices architecture to ensure high scalability, fault tolerance, and independent service deployment.

Team Structure

- **Total Teams**: 3 subteams (each with 4 members)
- **Total Microservices**: 4 Microservices

Subteam A - Flight Management Service

- **Members**: 4 (Ibrahim, Yousef Ashraf, Yaseen, Mohey)
- **Sub-Scrum Master**: Ibrahim Ashraf
- **Microservice**: FlightService

Subteam B - Booking Management Service

- **Members**: 4 (Youssef Amr, Ahmed Salah, Momen, Batwary)
- **Sub-Scrum Master**: Yousef Amr
- **Microservice**: BookingService

Subteam C - User and Notification Services

- **Members**: 4 (Haneen, Salma, Manar, Menna)
- **Sub-Scrum Master**: Haneen
- **Microservices**:
  - UserService
  - NotificationService

Microservices Design

1. Flight Management Service (Subteam A)

Database Entities

- **Flight**: flight_id (PK), aircraft (FK), origin, destination, departure_time, arrival_time, status, class_type, available_seats, gate_info
- **Aircraft**: aircraft_id (PK), model, capacity, airline_name
- **Seat**: seat_id (PK), flight_id (FK), seat_number, is_available, class_type
- **Price**: Id (PK), flight_id (FK), seat_id (FK), price

Design Patterns

- **Builder Pattern**: For creating flight objects with optional fields
- **Singleton Pattern**: For flight schedule manager to ensure a single instance

User Stories

- CRUD operations for flights
- Filter flights by destination/date/time
- Check seat availability
- Set or update flight status

Architecture Components

- Uses Redis caching for flight search results

2. Booking Management Service (Subteam B)

Database Entities

- **Booking**: bookingId (UUID), userId (UUID FK), Payment_id (UUID FK), status (Enum), createdAt (DateTime), updatedAt (DateTime)
- **FlightTicket**: FlightTicket (UUID), FullName (String), nationality (String), passportNumber (String), gender (String), dateOfBirth (Date), bookingId (UUID FK), FlightId (UUID FK), seatID (UUID FK)
- **Payment**: paymentId (UUID), bookingId (UUID FK), amount (Decimal), currency (String), status (Enum), paidAt (DateTime)

Entity Relationships

- Booking has many FlightTickets
- Booking has many Payments

Design Patterns

- **Factory Pattern**: For dynamically creating booking records
- **Command Pattern**: To encapsulate booking requests and queue them

User Stories

- Create a booking with flight segments
- View booking details
- Cancel a booking before departure
- Make a payment for a booking
- Receive confirmation after payment

Architecture Components

- Implements asynchronous booking processing with RabbitMQ

3. User Management Service (Subteam C)

Database Entities

- **User**: userId (Long PK), fullName (String), email (String), password (String), phone (String), registrationDate (LocalDateTime)
- **UserProfile**: userId (same as User), nationality (String), passportNumber (String), gender (String), dateOfBirth (Date)

Entity Relationships

- 1-to-1 between User and UserProfile (optional)

Design Patterns

- **Strategy Pattern**: For handling different login methods
- **Singleton Pattern**: For managing login sessions or shared AuthManager

User Stories

- Register a new user
- Log in
- Update profile info
- Change password
- Delete account
- View user info

4. Notification Service (Subteam C)

Database Entities (MongoDB Documents)

- **Notification**: id (MongoDB ID), userId (Long), bookingId (Long), type (EMAIL/SMS), message (String), timestamp (LocalDateTime)
- **NotificationTemplate**: templateId (String), type (EMAIL/SMS), title (String), content (String)

Design Patterns

- **Observer Pattern**: React to events received from RabbitMQ
- **Strategy Pattern**: Handle multiple sending channels (email, SMS)

User Stories

- Receive booking messages via RabbitMQ
- Store notifications as documents
- Simulate sending (email or SMS)
- Search messages by user, type, or date

Database Design

Microservice	DB Type	Justification
FlightService	PostgreSQL	Structured data, relational queries for routes and schedules
BookingService	PostgreSQL	Transactions need ACID guarantees
UserService	PostgreSQL	User authentication and relationships

Microservice	DB Type	Justification
NotificationService	MongoDB	Stores flexible notification templates and logs

Technological Stack

- **Backend**: Java Spring Boot
- **SQL Database**: PostgreSQL
- **NoSQL Database**: MongoDB (for NotificationService)
- **Load Balancer**: NGINX (static round-robin setup)
- **Caching**: Redis for flight search results
- **Message Queue**: RabbitMQ for Booking/Notification services

Microservices Communication

- **Synchronous**: RESTful APIs for core communication (UserService → BookingService)
- **Asynchronous**: RabbitMQ for booking confirmation → notification
    - **Why?** Reduces tight coupling and allows retry/queuing under high load

Reflection Usage

- **Where**: Reflection will be used in the BookingService
- **Why**: To dynamically instantiate booking strategies based on class name strings from config files or message queue

Summary of Microservices

Microservice	Team(s)	DB	Design Patterns	Message Queue
FlightService	A	PostgreSQL	Builder, Singleton	No
BookingService	B	PostgreSQL	Factory, Command	Yes (RabbitMQ)
UserService	C	PostgreSQL	Strategy, Singleton	No
NotificationService	C + B	MongoDB	Strategy, Observer	Yes (RabbitMQ)