College of Engineering

Department of Software Engineering

Distributed Systems

Title: Airlines Ticket Reservation Platform

Section: C

Group Members

No.	Name	ID
1	Helina Fisseha	ETS 0806/14
2	Kaleb Demisse	ETS 0869/14
3	Kirubel Eskender	ETS 0945/14
4	Meron Getaneh	ETS 1043/14
5	Meron Kassahun	ETS 1045/14
6	Mesud Ahmed	ETS 1054/14

Submitted to: Inst. Felix Edesa

Submission Date: November 21, 2025

Table of Contents

System Design Document

1. Introduction

1.1 Purpose

The purpose of this System Design Document (SDD) is to detail the architecture and contracts for the Airlines Ticket Reservation Platform. This document serves as a blueprint for the development team, outlining the components, data models, API definitions, and message flows required to build the system as defined in the project requirements.

1.2 Scope

The scope of this design covers the core functionality of an Airlines Ticket Reservation Platform, including:

- User account management (registration, login, profile).
- Flight data management (schedules, locations, seat capacity).
- Synchronous permanent flight booking and cancellation.
- Asynchronous user notifications (confirmation, cancellation).

Out of Scope: This system does not include any payment processing functionality, temporary seat-locking mechanisms, or expiration logic for bookings.

1.3 Assumptions and Constraints

- **Architecture:** The system must be built using a **Microservices Architecture**.
- **Persistence:** Each service is assumed to have its own dedicated database.
- **Communication:** Inter-service communication uses both synchronous (HTTP/REST) and asynchronous (Pub/Sub Events) methods.

2. Architectural Overview

The Airlines Ticket Reservation Platform utilizes a **Microservices Architecture** to ensure high decoupling, independent deployability, and scalability of individual business capabilities. It employs an **Event-Driven Architecture (EDA)** pattern for non-critical, delayed operations (like notifications).

Key Architectural Decisions:

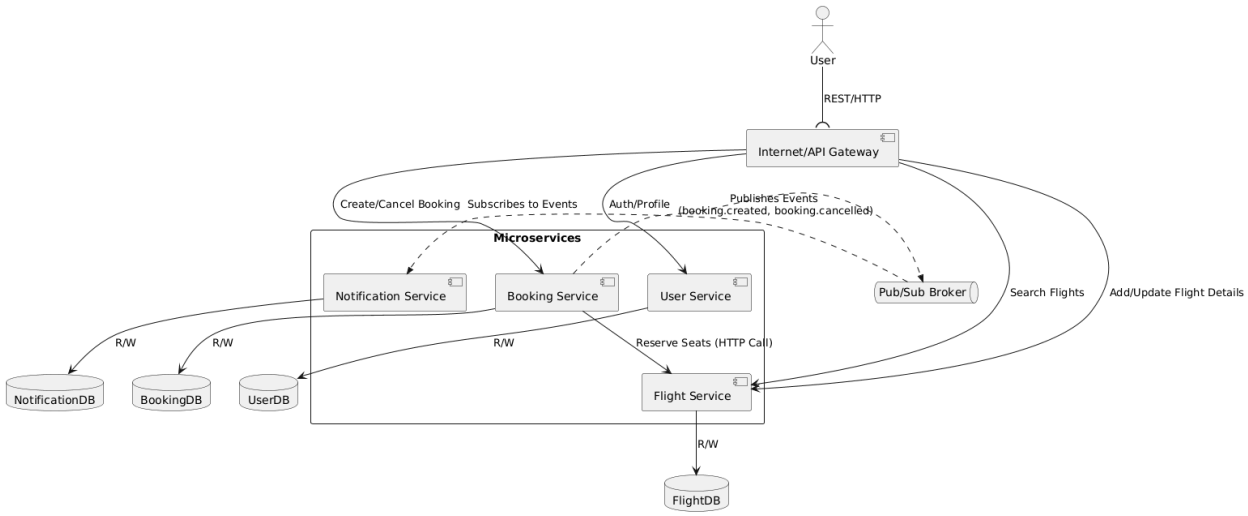
1. **Service Decoupling:** The system is divided into four distinct, vertically aligned microservices: User, Flight, Booking, and Notification.
2. **API Gateway :** An entry point (often referred to as an API Gateway, though not explicitly listed as a service) is assumed to route external traffic to the appropriate services.
3. **Synchronous Communication:** The critical path (e.g., booking a flight, which requires checking and reserving seats) uses synchronous communication.
4. **Asynchronous Communication:** Non-critical flows (e.g., confirming a booking or sending a cancellation notice) use a message broker (Pub/Sub) to achieve greater resilience and decoupling between the **Booking Service** and the **Notification Service**.

Microservices Included:

- **User Service:** Manages user identity and profile data.
- **Flight Service:** Manages all flight schedules and seat inventory.
- **Booking Service:** Manages the core business logic for creating and canceling reservations.
- **Notification Service:** Handles event subscription and sending communications (Email/SMS) to users.

3. Component View

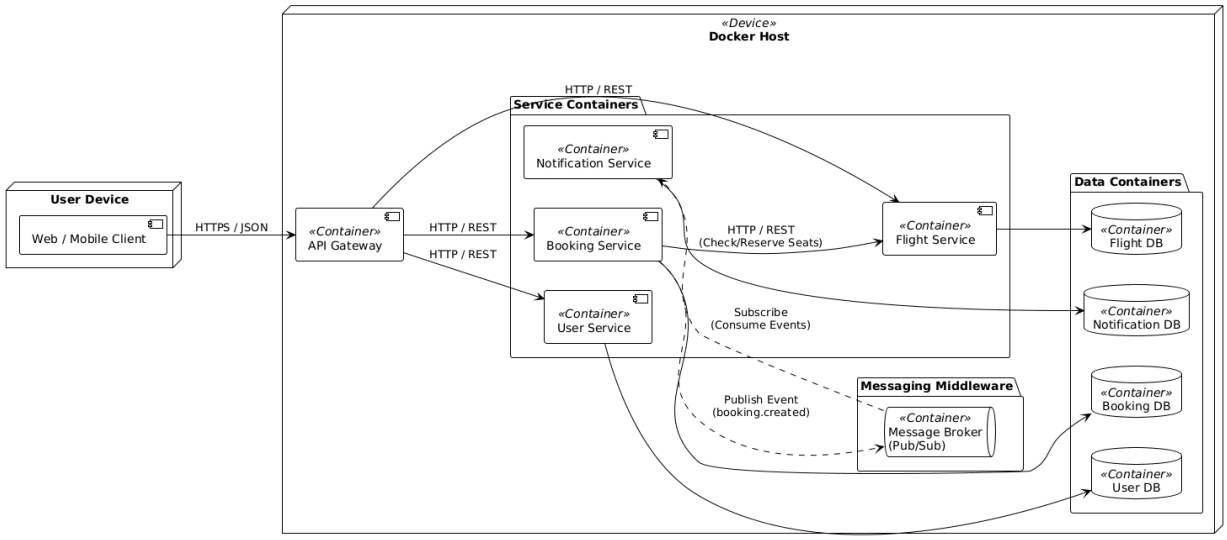
3.1 Component Diagram



3.2 Component Responsibilities

Component	Primary Purpose	Key Responsibilities	Persistence	Communication Type
User Service	User identity and profile management.	Register users, authenticate users	UserDB (users)	Synchronous (API)
Flight Service	Flight data and seat inventory management.	Store flight schedules, store locations, update available seat counts, handle Admin creation/updates of flight data, provide seat availability.	FlightDB (locations, flights)	Synchronous (API)
Booking Service	Core booking business logic.	Create/cancel bookings, validate seat availability (via Flight Service), reserve seats (via Flight Service), publish booking events.	BookingDB (bookings, booking_passengers)	Sync (Calls FS, US); Async (Publishes to Q)
Notification Service	Asynchronous user communication.	Subscribe to booking events (booking.created, booking.cancelled), send user notifications (Email/SMS), log delivery status.	NotificationDB (notifications)	Asynchronous (Subscribes to Q)

4. Deployment View



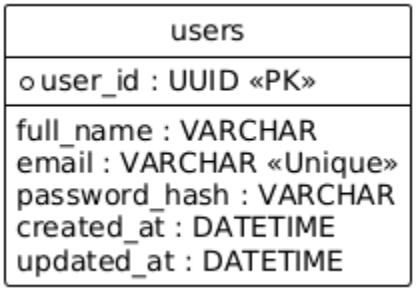
5. Data Model

This section outlines the data architecture for the Airlines Ticket Reservation System. Following the Microservices Architecture pattern, each service maintains its own dedicated database to ensure loose coupling. Data consistency across services is maintained through synchronous API calls for critical operations and asynchronous events for eventual consistency where applicable.

5.1 Entity-Relationship Diagram (ERD)

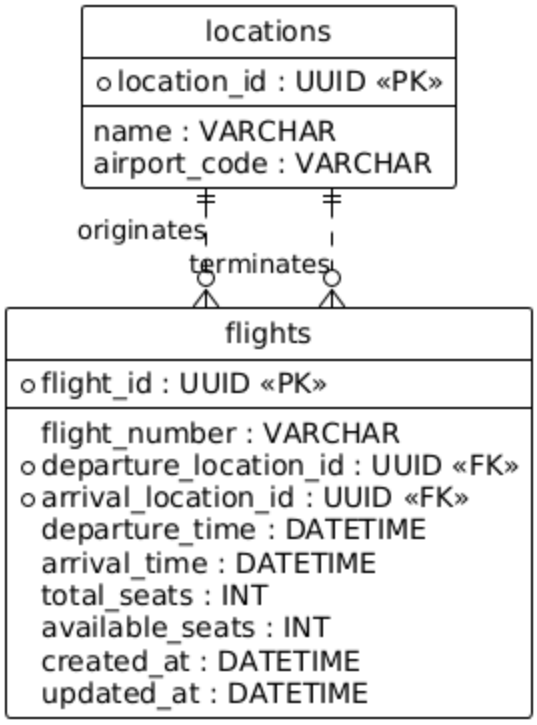
5.1.1 User Service Data Model

The User Service manages a single entity responsible for identity and authentication.



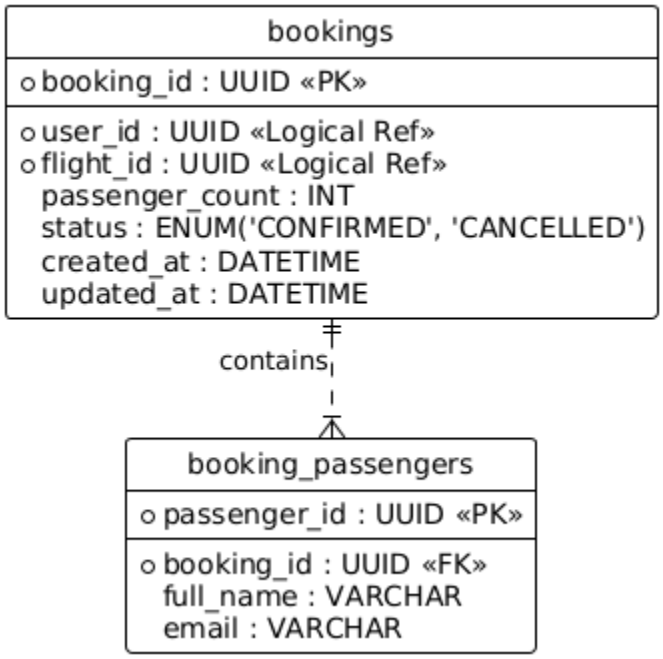
5.1.2 Flight Service Data Model

The Flight Service manages flight schedules and their relationship to airport locations.

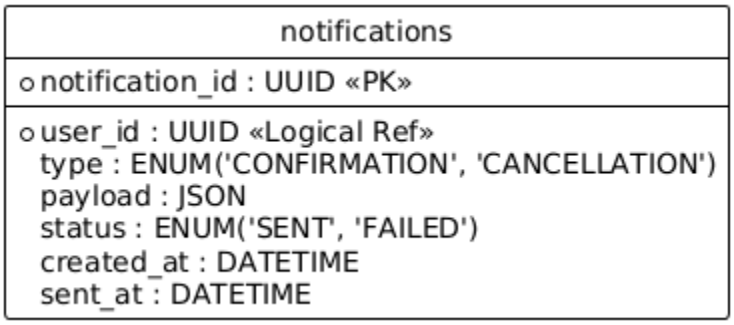


5.1.3 Booking Service Data Model

The Booking Service manages reservations and passenger details. It maintains logical references (UUIDs) to the User and Flight services but does not enforce physical foreign keys to them.



5.1.4 Notification Service Data Model



5.2 Core Schema Definitions

5.2.1 User Service Database

This database manages user identity and authentication credentials.

Table: users

Field	Type	Description
user_id	UUID (PK)	Unique identifier for the user.
full_name	VARCHAR	Full legal name of the user.
email	VARCHAR	User's email address (Unique). Used for login.
password_hash	VARCHAR	Securely hashed password.
created_at	DATETIME	Timestamp when the account was created.
updated_at	DATETIME	Timestamp when the profile was last updated.

5.2.2 Flight Service Database

This database manages static flight schedules, airport locations, and dynamic seat inventory.

Table: locations

Field	Type	Description
location_id	UUID (PK)	Unique identifier for the location.
name	VARCHAR	City or Location name (e.g., "Addis Ababa").
airport_code	VARCHAR	Standard airport code (e.g., "ADD").

Table: flights

Field	Type	Description
flight_id	UUID (PK)	Unique identifier for the flight.
flight_number	VARCHAR	Operational flight number (e.g., "ET102").
departure_location_id	UUID (FK)	Reference to the locations table.
arrival_location_id	UUID (FK)	Reference to the locations table.
departure_time	DATETIME	Scheduled departure time.
arrival_time	DATETIME	Scheduled arrival time.
total_seats	INT	Total capacity of the aircraft.
available_seats	INT	Current inventory count. Updated on booking/cancellation.
created_at	DATETIME	Record creation timestamp.
updated_at	DATETIME	Record update timestamp.

5.2.3 Booking Service Database

This database handles the core transactional data for reservations.

Table: bookings

Field	Type	Description
booking_id	UUID (PK)	Unique identifier for the reservation.
user_id	UUID	Logical reference to the User Service.
flight_id	UUID	Logical reference to the Flight Service.
passenger_count	INT	Number of seats reserved in this booking.
status	ENUM	Current state: CONFIRMED or CANCELLED.
created_at	DATETIME	Timestamp when the booking was made.
updated_at	DATETIME	Timestamp of the last status change.

Table: booking_passengers

Field	Type	Description
passenger_id	UUID (PK)	Unique identifier for the passenger detail record.
booking_id	UUID (FK)	Reference to the parent bookings record.
full_name	VARCHAR	Name of the specific passenger.

passport_id	VARCHAR	Passport Id of the passenger

5.2.4 Notification Service Database

This database acts as an audit log for communications sent to users.

Table: notifications

Field	Type	Description
notification_id	UUID (PK)	Unique identifier for the notification log.
user_id	UUID	Logical reference to the recipient (User Service).
type	ENUM	Type of event: BOOKING_CONFIRMATION, BOOKING_CANCELLATION.
payload	JSON	Snapshot of the data sent (e.g., flight details).
status	ENUM	Delivery status: SENT or FAILED.
created_at	DATETIME	Timestamp when the notification event was received.
sent_at	DATETIME	Timestamp when the message was successfully sent.

6. API Contracts (Service Interfaces)

This section defines the Application Programming Interfaces (APIs) for the Airlines Ticket Reservation System. The system exposes a RESTful API via a centralized API Gateway, which

routes traffic to the underlying microservices.

All API endpoints use standard HTTP methods (GET, POST, PUT, DELETE) and return responses in JSON format.

6.1 API Gateway Strategy

The API Gateway acts as the single entry point for all client applications. It is responsible for:

- **Routing:** Forwarding requests to the appropriate microservice (User, Flight, Booking, Notification).
- **Security:** handling SSL termination and initial request validation.

The following table summarizes the public-facing endpoints exposed by the Gateway and their destination services.

Method	Endpoint	Target Service	Description
Auth & Users			
POST	/api/v1/auth/register	User Service	Register a new user account.
POST	/api/v1/auth/login	User Service	Authenticate user and retrieve token.
GET	/api/v1/users/{id}	User Service	Retrieve user profile details.
PUT	/api/v1/users/{id}	User Service	Update user profile information.
DELETE	/api/v1/users/{id}	User Service	Delete a user account.
Flights			
GET	/api/v1/flights	Flight Service	Search for available flights.
GET	/api/v1/flights/{id}	Flight Service	Get details of a specific flight.

POST	/api/v1/flights	Flight Service	(Admin) Create a new flight schedule.
PUT	/api/v1/flights/{id}	Flight Service	(Admin) Update flight details.
DELETE	/api/v1/flights/{id}	Flight Service	(Admin) Cancel/Remove a flight.
GET	/api/v1/locations	Flight Service	List all airport locations.
Bookings			
POST	/api/v1/bookings	Booking Service	Create a new permanent booking.
GET	/api/v1/bookings/{id}	Booking Service	Retrieve booking details.
GET	/api/v1/bookings/user/{uid}	Booking Service	Retrieve booking history for a user.
DELETE	/api/v1/bookings/{id}	Booking Service	Cancel an existing booking.
Notifications			
GET	/api/v1/notifications/{uid}	Notif. Service	View notification history for a user.
DELETE	/api/v1/notifications/{id}	Notif. Service	Dismiss/Delete a specific notification.

6.2 Detailed OpenAPI/Swagger Specification

Refer to Appendix A or the github raw file.

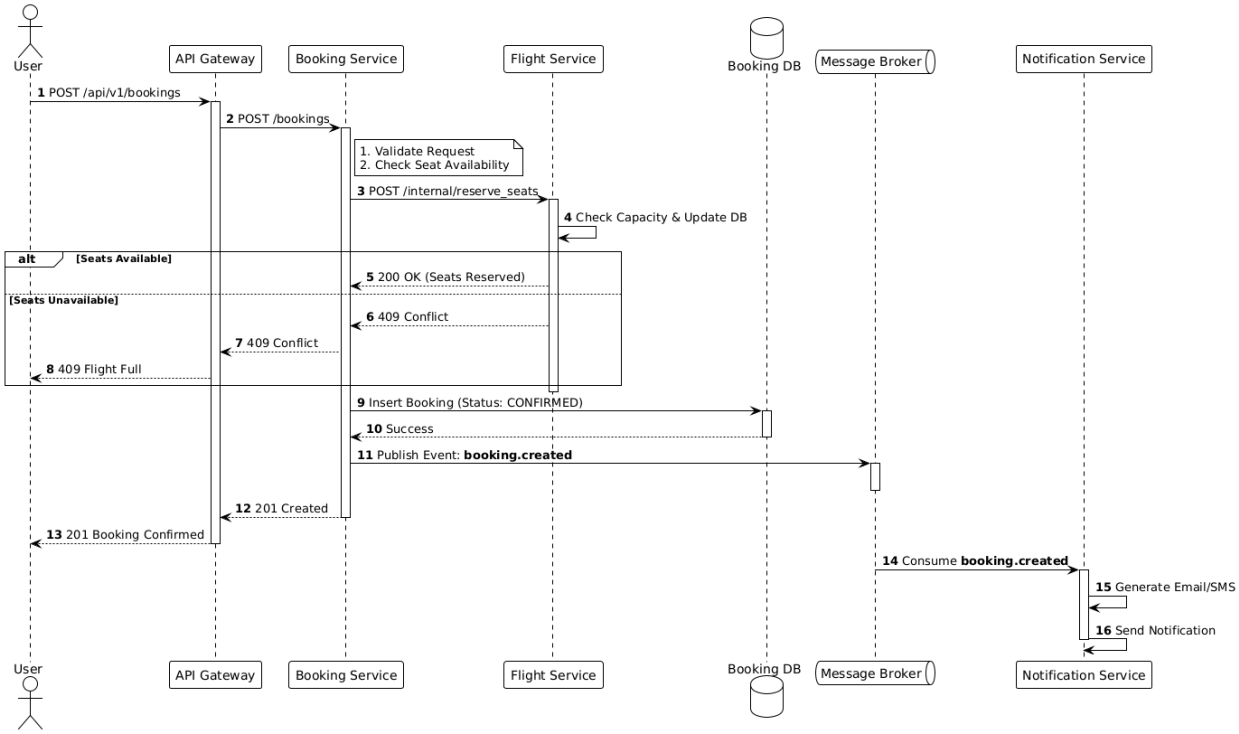
7. Message Flows and Event Schemas

7.1 Key Message Flows Diagram

The following sequence diagrams illustrate the primary interactions between the User, API

Gateway, and backend services.

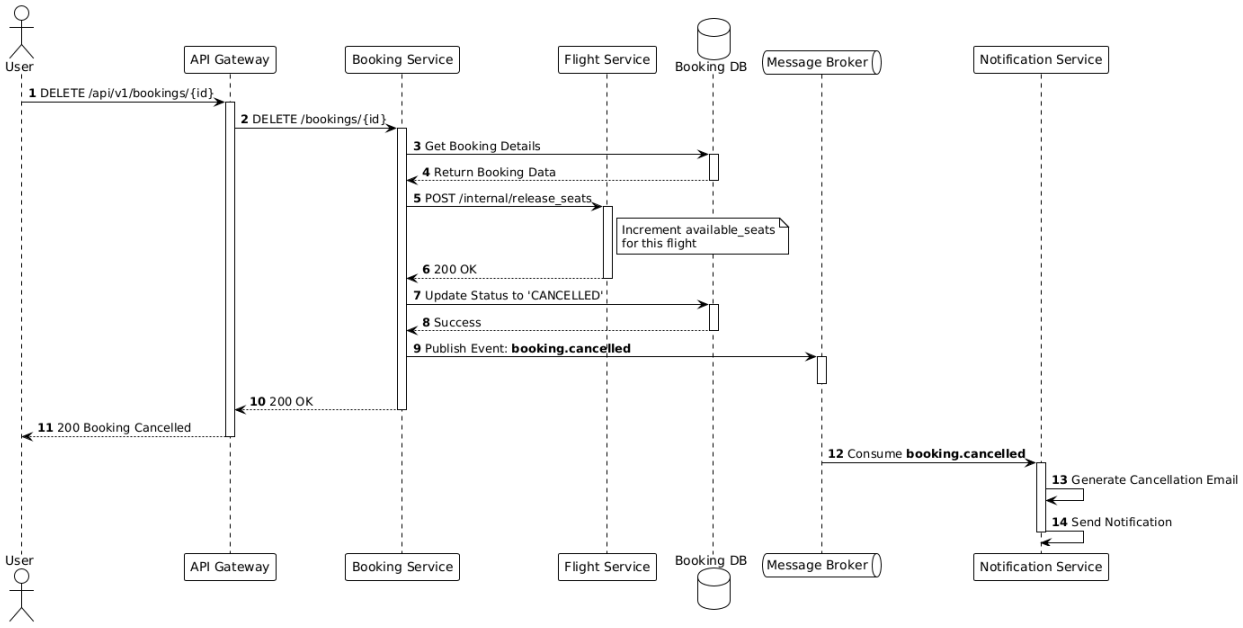
7.1.1 Sequence Diagram: Create Booking Flow



This flow represents the "Happy Path" for creating a booking. Note that the seat reservation with the Flight Service is synchronous to prevent overbooking, while the notification is handled asynchronously.

7.1.2 Sequence Diagram: Cancel Booking Flow

This flow handles the cancellation of an existing reservation. It ensures seats are released back to the inventory synchronously.



7.2 Pub/Sub Topic Definitions

The system uses a Message Broker to decouple the Booking Service from the Notification Service.

Topic Name	Producer	Consumer	Purpose	Retention Policy
booking.created	Booking Service	Notification Service	Triggered when a booking is successfully saved and confirmed. Used to generate confirmation emails.	Delete after acknowledgement
booking.cancelled	Booking Service	Notification Service	Triggered when a booking is successfully cancelled. Used to notify the user of the refund/cancellation.	Delete after acknowledgement

7.3 Detailed Event JSON Schemas

The following schemas define the data structure (payload) for the events published to the message broker.

Event: Booking Created

Topic: booking.created

Description: Contains details required to send a confirmation email, including passenger list and flight summary.

Payload (JSON):

```
{
  "eventId": "evt_1234567890",
  "eventType": "BOOKING_CREATED",
  "timestamp": "2025-11-21T14:30:00Z",
  "data": {
    "bookingId": "bkg_987654321",
    "userId": "usr_11223344",
    "flightId": "flt_55667788",
    "flightNumber": "ET-302",
    "passengerCount": 2,
    "totalPrice": 450.00,
    "passengers": [
      {
        "fullName": "Abebe Bikila",
        "email": "abebe@example.com"
      },
      {
        "fullName": "Kebede Tadesse",
        "email": "kebede@example.com"
      }
    ]
  }
}
```

Event: Booking Cancelled

Topic: booking.cancelled

Description: Contains minimal details required to notify the user that their reservation has been removed.

Payload (JSON):

```
{
```

```
  "eventId": "evt_0987654321",
  "eventType": "BOOKING_CANCELLED",
  "timestamp": "2025-11-22T09:15:00Z",
  "data": {
    "bookingId": "bkg_987654321",
    "userId": "usr_11223344",
    "flightId": "flt_55667788",
    "reason": "User requested cancellation via portal"
  }
}
```

8. Conclusion

The Airlines Ticket Reservation System architecture leverages a containerized Microservices approach to ensure scalability, maintainability, and independent deployment. By isolating core domains: Users, Flights, Bookings, and Notifications into distinct services with dedicated databases, the system achieves high decoupling.

The hybrid communication strategy ensures data integrity for critical booking operations through synchronous REST calls, while providing resilience and responsiveness for user communications via asynchronous Pub/Sub messaging.

Appendix

Appendix A

```
openapi: 3.0.3
info:
  title: Flight Ticket Booking System API
  description: >
    API specification for the Flight Ticket Booking System.
    Routes traffic via an API Gateway to User, Flight, Booking, and Notification microservices.
  version: 1.0.0

paths:
  # ----------------------------
  # AUTHENTICATION & USER MANAGEMENT
  # ----------------------------
  /auth/register:
    post:
      tags: [Auth]
      summary: Register a new user
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UserRegistration'
      responses:
        '201':
          description: User created successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/UserResponse'

  /auth/login:
    post:
      tags: [Auth]
      summary: Authenticate user
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
```

```
            required: [email, password]
            properties:
              email: { type: string, format: email }
              password: { type: string, format: password }
    responses:
      '200':
        description: Login successful
        content:
          application/json:
            schema:
              type: object
              properties:
                token: { type: string }
                user: { $ref: '#/components/schemas/UserResponse' }

/users/{id}:
  parameters:
    - name: id
      in: path
      required: true
      schema: { type: string, format: uuid }
  get:
    tags: [Users]
    summary: Get user profile details
    responses:
      '200':
        description: User found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UserResponse'
  put:
    tags: [Users]
    summary: Update user profile
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              full_name: { type: string }
              password: { type: string }
    responses:
      '200':
```

```
        description: Profile updated
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UserResponse'
  delete:
    tags: [Users]
    summary: Delete user account
    responses:
      '204':
        description: User deleted

# ----------------------------
# FLIGHT MANAGEMENT
# ----------------------------
/locations:
  get:
    tags: [Flights]
    summary: List all airport locations
    responses:
      '200':
        description: List of locations
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Location'

/flights:
  get:
    tags: [Flights]
    summary: Search for available flights
    parameters:
      - name: from
        in: query
        description: Departure Location ID
        schema: { type: string, format: uuid }
      - name: to
        in: query
        description: Arrival Location ID
        schema: { type: string, format: uuid }
      - name: date
        in: query
```

```
        schema: { type: string, format: date }
    responses:
      '200':
        description: Search results
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/FlightResponse'
  post:
    tags: [Flights]
    summary: (Admin) Create a new flight schedule
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/FlightCreateRequest'
    responses:
      '201':
        description: Flight created
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/FlightResponse'

/flights/{id}:
  parameters:
    - name: id
      in: path
      required: true
      schema: { type: string, format: uuid }
  get:
    tags: [Flights]
    summary: Get flight details
    responses:
      '200':
        description: Flight details
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/FlightResponse'
  put:
```

```
    tags: [Flights]
    summary: (Admin) Update flight details
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/FlightCreateRequest'
    responses:
      '200':
        description: Flight updated
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/FlightResponse'
  delete:
    tags: [Flights]
    summary: (Admin) Cancel/Remove a flight
    responses:
      '204':
        description: Flight removed

# ----------------------------
# BOOKING MANAGEMENT
# ----------------------------
/bookings:
  post:
    tags: [Bookings]
    summary: Create a permanent booking
    description: Synchronously reserves seats. Returns error if full.
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/BookingRequest'
    responses:
      '201':
        description: Booking Confirmed
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BookingResponse'
      '409':
        description: Flight fully booked or unavailable
```

```
/bookings/{id}:
  parameters:
    - name: id
      in: path
      required: true
      schema: { type: string, format: uuid }
  get:
    tags: [Bookings]
    summary: Get booking details
    responses:
      '200':
        description: Booking found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BookingResponse'
  delete:
    tags: [Bookings]
    summary: Cancel a booking
    responses:
      '200':
        description: Booking cancelled
        content:
          application/json:
            schema:
              type: object
              properties:
                status: { type: string, example: "CANCELLED" }

/bookings/user/{uid}:
  parameters:
    - name: uid
      in: path
      required: true
      schema: { type: string, format: uuid }
  get:
    tags: [Bookings]
    summary: Retrieve booking history for a user
    responses:
      '200':
        description: User booking history
        content:
          application/json:
```

```
            schema:
              type: array
              items:
                $ref: '#/components/schemas/BookingResponse'

  # ----------------------------
  # NOTIFICATIONS
  # ----------------------------
  /notifications/{uid}:
    parameters:
      - name: uid
        in: path
        required: true
        schema: { type: string, format: uuid }
    get:
      tags: [Notifications]
      summary: View notification history for a user
      responses:
        '200':
          description: List of notifications
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Notification'

  /notifications/{id}:
    parameters:
      - name: id
        in: path
        required: true
        schema: { type: string, format: uuid }
    delete:
      tags: [Notifications]
      summary: Dismiss/Delete a notification
      responses:
        '204':
          description: Notification deleted

# ----------------------------
# DATA MODELS
# ----------------------------
components:
```

```
  schemas:
    # User Schemas
    UserRegistration:
      type: object
      required: [full_name, email, password]
      properties:
        full_name: { type: string, example: "John Doe" }
        email: { type: string, format: email, example: "john@example.com" }
        password: { type: string, format: password, example: "securePass123" }

    UserResponse:
      type: object
      properties:
        user_id: { type: string, format: uuid }
        full_name: { type: string }
        email: { type: string, format: email }
        created_at: { type: string, format: date-time }

    # Flight Schemas
    Location:
      type: object
      properties:
        location_id: { type: string, format: uuid }
        name: { type: string, example: "Addis Ababa" }
        airport_code: { type: string, example: "ADD" }

    FlightCreateRequest:
      type: object
      required: [flight_number, departure_location_id, arrival_location_id, departure_time,
total_seats]
      properties:
        flight_number: { type: string, example: "ET-500" }
        departure_location_id: { type: string, format: uuid }
        arrival_location_id: { type: string, format: uuid }
        departure_time: { type: string, format: date-time }
        arrival_time: { type: string, format: date-time }
        total_seats: { type: integer, example: 150 }

    FlightResponse:
      allOf:
        - $ref: '#/components/schemas/FlightCreateRequest'
        - type: object
          properties:
            flight_id: { type: string, format: uuid }
```

```
        available_seats: { type: integer }

# Booking Schemas
BookingRequest:
  type: object
  required: [user_id, flight_id, passengers]
  properties:
    user_id: { type: string, format: uuid }
    flight_id: { type: string, format: uuid }
    passengers:
      type: array
      items:
        type: object
        properties:
          full_name: { type: string }
          email: { type: string }

BookingResponse:
  type: object
  properties:
    booking_id: { type: string, format: uuid }
    flight_id: { type: string, format: uuid }
    status: { type: string, enum: [CONFIRMED, CANCELLED] }
    passenger_count: { type: integer }
    created_at: { type: string, format: date-time }

# Notification Schemas
Notification:
  type: object
  properties:
    notification_id: { type: string, format: uuid }
    type: { type: string, enum: [BOOKING_CONFIRMATION, BOOKING_CANCELLATION] }
    message: { type: string }
    sent_at: { type: string, format: date-time }
```