Flight Status Tracker Workflow

Workflow Overview

1. **User Login and Flight Details**
    - **User Login:** Passengers log in with their PNR (Passenger Name Record).
    - **User Portal:** After login, the user is redirected to their portal where they can view the latest details of their flight.
2. **Admin Login and Flight Management**
    - **Admin Login:** Admins log in to access flight management functionalities.
    - **Flight Management:** Admins can view all flights, search for specific flights, and update their status.
3. **Updating Flight Details**
    - **Edit Flight Details:** When an admin selects a flight to update, an edit modal opens allowing them to modify flight details such as arrival time, departure time, date, and boarding gate.
    - **Save Status:** Once the admin saves the updated details, the changes are recorded in the database.
4. **Notification Process**
    - **Trigger Notification:** After saving the flight status, notifications are triggered for passengers who have booked the flight.
    - **Notification Delivery:** Notifications are sent via email to the passengers using RabbitMQ.

Detailed API Workflow

1. User Portal

- **API Call:** When the user logs in with their PNR, the frontend (React.js) makes an API call to the backend (Spring Boot) to fetch the flight details associated with the PNR.
- **Endpoint Example:**
    - **GET /api/flight/{pnr}**
    - **Request:** `{ pnr: "PNR12345" }`
    - **Response:** Flight details including current status, arrival time, departure time, date, and boarding gate.

2. Admin Portal

- **API Call:** After the admin logs in, the frontend makes an API call to retrieve all flights from the backend.
- **Endpoint Example:**
    - **GET /api/flights**
    - **Response:** List of all flights with their details.
- **API Call:** To search for a specific flight, the frontend sends a search request to the backend.
- **Endpoint Example:**
    - **GET /api/flights/search?query={flightNumber}**

    - **Request:** `{ flightNumber: "FL123" }`
    - **Response:** Flight details matching the search criteria.
- **API Call:** When an admin selects a flight and opens the edit modal, the frontend requests the current details of that flight for editing.
- **Endpoint Example:**
    - **GET /api/flight/{flightId}**
    - **Request:** `{ flightId: "FL123" }`
    - **Response:** Current flight details for editing.
- **API Call:** After the admin updates the flight details and clicks "Save," the frontend sends an API call to the backend to update the flight status.
- **Endpoint Example:**
    - **PUT /api/flight/{flightId}**

Request Body:
json

```
{
  "arrivalTime": "2024-08-01T12:30:00Z",
  "departureTime": "2024-08-01T15:00:00Z",
  "date": "2024-08-01",
  "boardingGate": "G12"
}
```

    - **Response:** Confirmation of the update.

3. Notification Service

- **Triggering Notification:**
    - **API Call:** After updating the flight status, the backend triggers a notification service.
    - **Backend Action:** The backend publishes a message to RabbitMQ with details about the flight update.
- **RabbitMQ Processing:**
    - **Consumer Service:** A consumer service listens to RabbitMQ for flight update messages.
    - **Email Sending:** The consumer service retrieves the passenger details from the database and sends email notifications about the flight status changes.

Summary

1. **User logs in** with their PNR and fetches flight details from the backend.
2. **Admin logs in,** manages flights, and updates their status through the backend.
3. **Flight status updates** are processed, and a notification message is sent to RabbitMQ.
4. **RabbitMQ** triggers notifications to passengers via email based on the updates.

This workflow ensures that passengers receive timely updates about their flight status changes and admins can efficiently manage and update flight information.