Setup for flight booking system

Frontend (Java Swing)

The frontend provides the user interface, allowing interactions with the backend through buttons, forms, and display components. Here's how each action would work in detail:

1. Welcome Page

    Buttons: Two buttons labeled "Admin" and "Customer."

    Action:

    - Admin Button: Takes the user to an admin login page.
    - Customer Button: Takes the user to the flight search page directly.

2. Admin Login Page

    Form Fields: JTextField for username and JPasswordField for password.

    Login Button:

    - ActionListener: When clicked, it calls the UserService.login() method with the entered username, password, and role as "admin."
    - Backend Connection: If login returns true, proceed to the admin dashboard (where the admin can add or delete flights); if false, show an error message.

3. Admin Dashboard (Add Flight Page)

    Form Fields: JTextFields for flight origin, destination, date, time, seats, and price.

    Flight List Table:

        - Display the list of all flights in a JTable, including columns for origin, destination, date, time, seats available, and price.
        - Each row should have a unique flight_id, which will be helpful when updating or deleting a specific flight.

    Add Flight Button:

    - ActionListener: When clicked, it collects data from the fields, creates a Flight object, and calls FlightService.addFlight().
    - Backend Connection: The flight details are saved in the database if the insertion is successful. Display a confirmation message on success.

    Delete Flight Button:

    - Add a "Delete Flight" button. When clicked, it should delete the selected flight from the database.

    Delete Flight Action Flow

    - The admin selects a flight from the JTable.
    - Retrieve the flight_id from the selected row.
    - Delete Flight Button:
        - ActionListener: When clicked, call the deleteFlight method in FlightService with the flight_id of the selected flight.
        - Confirmation Dialog: Display a confirmation dialog before proceeding with deletion

    Update Flight Button:

- Add an "Update Flight" button. When clicked, it should bring up an update form pre-populated with the selected flight's details.

Update Flight Action Flow

- The admin selects a flight from the JTable.
- Retrieve the flight_id and other flight details from the selected row.
- Update Flight Button:
    - ActionListener: When clicked, open a new JFrame with a form pre-filled with the selected flight's details.
    - After updating details in the form, clicking "Save" calls the updateFlight method in FlightService.
- Pre-fill form with flight details, allowing the admin to modify them.

4. Customer Search Flights Page

Search Fields: JTextFields for origin, destination, date.
Search Button:

- ActionListener: When clicked, it retrieves the flight search criteria and calls a searchFlights method in FlightService to fetch available flights from the database.
- Results Table: Display the results in a JTable showing origin, destination, date, time, available seats, and price.

5. Customer Booking Page

Flight Selection: Customers select a flight from the search results.
Seat Selection: Choose a seat number if applicable.
Book Flight Button:

- ActionListener: When clicked, it captures the selected flight, seat, and price and calls BookingService.createBooking() to save the booking.
- Backend Connection: If the booking is successful, display a booking confirmation with flight details, seat number, and total price.

Backend (Java)

The backend will handle all the core logic, including user authentication, flight management (adding, updating, and deleting flights), and booking management. Here's how each feature would function in detail:

Core Classes:

1. Database Connectivity

Create a DatabaseConnection class that establishes a connection to the MySQL database.
This class should have a method like getConnection() that returns a Connection object.

2. User Authentication (Login for Admins)

Create a UserService class with a login method to validate admin credentials.
This method checks the users table in the database to verify the username, password, and role.

3. Flight Management (Admin)

Add Flight: Create a FlightService class with an addFlight method to insert new flight records. This method takes flight details as parameters and inserts them into the flights table. Other methods may include deleteFlight(int flightId) for removing a flight and updateFlight(int flightId) for modifying flight details.

4. Booking Management (Customer)

Create Booking: The BookingService class would have a createBooking method that records a booking in the bookings table, capturing the user ID, flight ID, seat number, and price.

Additional Backend Classes:

- FlightSearchService:
    - This class can be dedicated to handling searches for flights, allowing customers to query available flights based on their criteria (origin, destination, date).
- User Class:
    - To represent a user object, which could help in managing user details throughout the application.
- Booking Class:
    - Similar to the User class, a Booking class could encapsulate booking details, making it easier to handle bookings in an object-oriented way.
- Flight Class:
    - A Flight class to represent flight objects, which can help manage flight data effectively within the application.

Database

1. Users Table

The users table holds information about both admins and customers, including their roles for access control.

```
CREATE TABLE users (
   user_id INT AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(100) NOT NULL,
   password VARCHAR(100) NOT NULL,
   email VARCHAR(100),
   role ENUM('admin', 'customer') NOT NULL -- Specifies if user is an admin or customer
);
```

2. Flights Table

The flights table stores all flight details, including a price field to represent the current price per seat for each flight. In this setup, the admin role has privileges to add, update, and delete flight details (including price, available seats, date, etc.) in the flights table, while customers can view flight information and make bookings based on the current details.

```
CREATE TABLE flights (
   flight_id INT AUTO_INCREMENT PRIMARY KEY,
   origin VARCHAR(50) NOT NULL,
   destination VARCHAR(50) NOT NULL,
   date DATE NOT NULL,
   time TIME NOT NULL,
   seats_available INT NOT NULL,
   price DECIMAL(10, 2) NOT NULL -- Current price per seat for the flight
);
```

3. Bookings Table
The bookings table stores each booking made by customers, including the price per seat at the time of booking. This preserves the original price even if the flight price changes later.

```
CREATE TABLE bookings (
   booking_id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL, -- Links to the user making the booking
   flight_id INT NOT NULL, -- Links to the booked flight
   seat_number VARCHAR(5) NOT NULL,
   price DECIMAL(10, 2) NOT NULL, -- Price per seat at the time of booking
   FOREIGN KEY (user_id) REFERENCES users(user_id),
   FOREIGN KEY (flight_id) REFERENCES flights(flight_id)
);
```

Database Relationships Summary:
The user_id in bookings references users(user_id) to associate each booking with a specific customer.
The flight_id in bookings references flights(flight_id) to link each booking to a particular flight.
The price column in bookings allows each booking to retain its original cost, even if the flight's price changes later.
With this setup, your system can efficiently handle both current flight details and historical booking records.