Omar

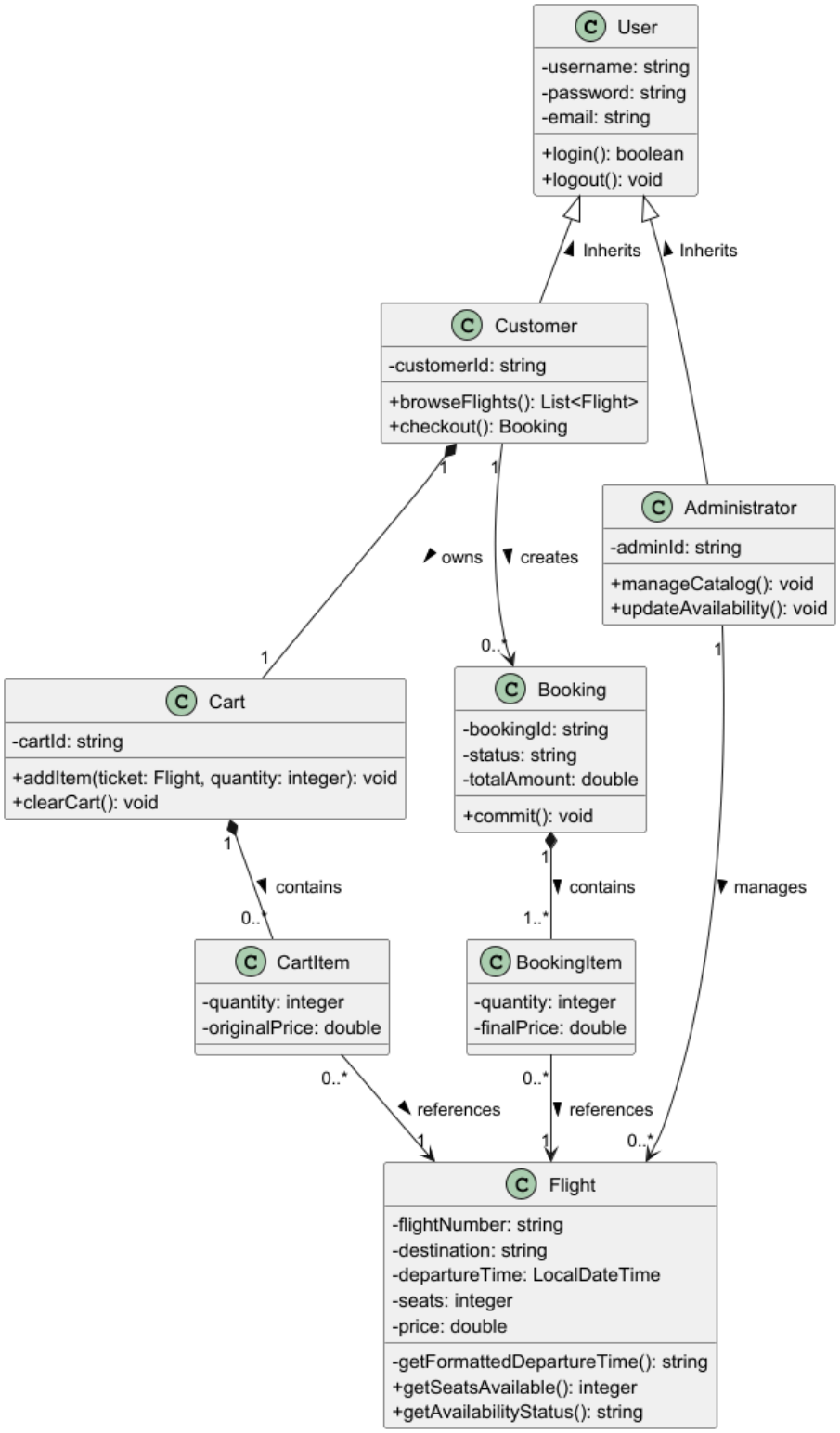
PlantUml & Domain analysis

Code:

```
@startuml

skinparam classAttributeIconSize 0

class User{

- username: string
- password: string
- email: string
+ login(): boolean
+ logout(): void

}

class Customer{

- customerId: string
+ browseFlights(): List<Flight>
+ checkout(): Booking

}

class Administrator{

- adminId: string
+ manageCatalog(): void
+ updateAvailability(): void

}

class Flight{

- flightNumber: string
- destination: string
- departureTime: LocalDateTime
- getFormattedDepartureTime(): string
- seats: integer
+ getSeatsAvailable(): integer
- price: double
+ getAvailabilityStatus(): string

}

class Cart{
```

```
- cartId: string
+ addItem(ticket: Flight, quantity: integer): void
+ clearCart(): void
}

class CartItem{
- quantity: integer
- originalPrice: double
}


class Booking{

- bookingId: string
- status: string
- totalAmount: double
+ commit(): void
}

class BookingItem{

- quantity: integer
- finalPrice: double
}


User <|-- Customer: Inherits <
User <|-- Administrator: Inherits <

Customer "1" *-- "1" Cart : owns >
Cart "1" *-- "0..*" CartItem : contains >
CartItem "0..*" --> "1" Flight : references >

Administrator "1" --> "0..*" Flight : manages >

Customer "1" --> "0..*" Booking : creates >
Booking "1" *-- "1..*" BookingItem: contains >
BookingItem "0..*" --> "1" Flight : references >



@enduml
```

The UML diagram:



Domain analysis

Core Entities

1. **User:** The default entity type that represents anyone using the system.

2. **Customer:** A specific entity type that represents a user that searches for available flights, using a booking cart, and reserves flights.

3. **Administrator:** A specific entity type that represents a user that is responsible for managing the flight catalog and updating ticket availability.

4. **Flight:** The catalog entry represents a plane ticket, and includes information such as flight number, destination, date, seat, availability, and time.

5. **Cart:** A private collection that is managed by the customer to add plane tickets to it. It will be persistently saved in the database.

6. **CartItem:** An entity representing specific flight tickets added to the customers cart, it tracks the quantity added and it shows the original price of the ticket at time it was added.

7. **Booking:** The final transaction record that is created when the customer checks out, this will clear the cart and update the availability of the tickets.

8. **BookingItem:** A record of the specific flights that have been purchased in a booking, and it shows the final price.

Relationships Between Entities

Inheritance: The Customer and Administrator both **inherit** attributes and authentication methods from the base User class.

Composition:

- A customer owns exactly 1 cart.
- A cart contains zero to many CartItems, and if the cart is cleared then the CartItems will be removed.
- A booking contains one to many BookingItems.

Association:

- An Administrator manages zero to many Flights in the catalog.
- A Customer creates zero to many Bookings over time.

- Both CartItem and BookingItem reference exactly 1 Flight from the central catalog to avoid data duplication.

Dhabya

FUNCTIONAL REQUIREMENTS

1. User Management

FR-001: The system shall allow users to log in using a valid username and password.

FR-002: The system shall allow users to log out then terminate their session.

FR-003: The system shall restrict functions based on user role (Customer or Administrator).

2. Flight Catalog

FR-010: The system shall display all the available flights.

FR-011: The system shall allow users to search flights based on the destination.

FR-012: The system shall allow users to search flights based on the travel date.

FR-013: The system shall display flight information in detail and this includes flight number, destination, date, and available seats.

FR-014: The system shall display the availability status of each flight.

3. Booking Cart

FR-020: The system shall assign each customer exactly one booking cart.

FR-021: The system shall allow customers to add flights into their booking cart.

FR-022: The system shall allow customers to remove flights from their booking cart.

FR-023: The system shall display all the flights that are currently in the customer's booking cart.

FR-024: The system shall persist cart items for logged-in customers in the database.

FR-025: The system shall restrict access to the cart for only the owning customer.

4. Booking Management

FR-030: The system shall allow customers to commit selected flights and create a booking.

FR-031: The system shall require that at least one flight is in the booking.

FR-032: The system shall update the availability of seats in the flight after a booking is committed.

FR-033: The system shall clear the booking cart after a booking was successful.

FR-034: The system shall store and manage a booking status value.

5. Administrator Flight Management

FR-040: The system shall allow administrators to be able to add new flights to the catalog.

FR-041: The system shall allow administrators to update flight details including destination, date, and seat availability.

FR-042: The system shall allow administrators to remove flights from the catalog.

6. Non-Functional Requirements

NFR-001: The system shall securely store user passwords using hashing.
NFR-002: The system shall restrict unauthorized access to protected endpoints.
NFR-003: The system shall persist all data in a MariaDB database.
NFR-004: The system shall provide acceptable response time under normal load.

7. Constraints

- The backend must use Spring Boot.
- Frontend must use Mustache.
- The database must use MariaDB.

8. Assumptions

- Each user has a unique username.
- Booking requires at least one flight.
- Payment processing is outside current scope.

Glossary

User

Definition: A registered account in the Wandria system that can authenticate and access system features.

Attributes:

- username (unique identifier)
- password (securely stored)
- email

Synonyms: Account

Related Terms: Customer, Administrator, Authentication

Business Rules:

- Each User must have a unique username.
- Passwords must be securely stored (not plain text).

Example: Username: Dalsuwaidi Email: dibrahim34@gmail.com

Customer

Definition: A User who can browse flights, manage a booking cart, and commit bookings.

Attributes:

- customerId (unique)

- Associated Cart
- Associated Bookings

Synonyms: Traveler, Client

Related Terms: Cart, Booking, Flight

Business Rules:

- Each Customer has exactly one Cart.
- Customers can only access their own Cart and Bookings.

Example: Customer ID: C1023

Administrator

Definition: A User with elevated permissions who manages the flight catalog and availability.

Attributes:

- adminId (unique)

Synonyms: Admin

Related Terms: Flight, Catalog, Availability

Business Rules:

- Only Administrators can add, update, or remove Flights.

Example: Admin ID: A001

Flight

Definition: A catalog entry representing a scheduled travel route that can be booked by Customers.

Attributes:

- flightNumber (unique identifier)
- destination
- date
- seats (available seat count)

Synonyms: Route, Trip

Related Terms: Booking, Cart, Availability

Business Rules:

- Each Flight must have a unique flightNumber.
- Available seats cannot be negative.
- A Flight can only be reserved if seats > 0.

Example: Flight Number: EK202 Destination: Dubai Date: 2026-03-15 Seats: 120

Cart (Booking Cart)

Definition: A collection of selected Flights associated with a Customer prior to booking.

Attributes:

- cartId (unique)
- Contains one or more Flights

Synonyms: Booking Cart, Shopping Cart

Related Terms: Customer, Booking, Flight

Business Rules:

- Each Customer manages exactly one Cart.
- Cart contents are persistent in the database.
- Only the owning Customer can access their Cart.

Example: Cart ID: CART445

Booking

Definition: A finalized reservation created when a Customer commits selected Flights from their Cart.

Attributes:

- bookingId (unique)
- status (Pending, Confirmed)

Synonyms: Reservation

Related Terms: Customer, Flight, Commit

Business Rules:

- A Booking must contain at least one Flight.
- Booking updates Flight availability.
- After Booking is committed, the Cart is cleared.

Example: Booking ID: BKG778 Status: Confirmed

Commit

Definition: The action performed by a Customer to finalize a Cart and create a Booking.

Synonyms: Finalize, Confirm

Related Terms: Booking, Availability

Business Rules:

- Commit reduces available seats of reserved Flights.
- Commit clears the Customer's Cart.

Availability

Definition: The number of remaining seats for a Flight.

Synonyms: Seat Count

Related Terms: Flight, Booking

Business Rules:

- Availability cannot be negative.
- Booking is only allowed if availability > 0.

Role-Based Access Control (RBAC)

Definition: A security mechanism that restricts system functionality based on user role.

Synonyms: Role Authorization

Related Terms: User, Customer, Administrator

Business Rules:

- Customers cannot modify Flights.
- Administrators cannot access Customer Carts.

Spring Boot

Definition: A Java-based backend framework used to build RESTful APIs and manage server-side logic.

Related Terms: Backend, REST API, Endpoint, Mustache

Mustache

Definition: A logic-less templating engine used in the Wandria system to generate dynamic HTML pages by inserting server-side data into predefined templates.

Attributes:

- Template files (.mustache)
- Placeholders for dynamic data

Synonyms: Templating Engine

Related Terms: Template, HTML, Frontend, Spring Boot

Business Rules:

- Mustache templates do not contain business logic.
- Data is provided by the backend before rendering the page.

Example: File: flights.mustache displays flight data dynamically.

Template

Definition: A predefined HTML file containing placeholders that are replaced with dynamic data when the page is rendered.

Attributes:

- templateName (unique identifier)
- placeholders for dynamic content

Synonyms: View

Related Terms: Mustache, HTML, Frontend

Business Rules:

- Templates must follow Mustache syntax.
- Templates are rendered by the backend before being sent to the user.

Example: Template: booking.mustache displays booking confirmation.

Frontend

Definition: The part of the system that users interact with directly, including web pages and user interface components.

Attributes:

- User interface pages
- Interactive elements

Synonyms: Client-side

Related Terms: Mustache, HTML, Backend

Business Rules:

- Frontend must display accurate data received from the backend.
- Users interact with the system through the frontend.

Example: The Wandria homepage where users search for flights.

Backend

Definition: The server-side part of the system responsible for processing requests, executing business logic, and interacting with the database.

Attributes:

- Server application
- API endpoints

Synonyms: Server-side

Related Terms: Spring Boot, MariaDB, REST API

Business Rules:

- Backend must validate user input.
- Backend manages communication with the database.

Example: Spring Boot server processing flight booking requests.

HTML

Definition: A markup language used to structure and display content on web pages.

Attributes:

- Tags and elements
- Page structure

Synonyms: HyperText Markup Language

Related Terms: Frontend, Template, Mustache

Business Rules:

- HTML must follow proper syntax.
- HTML pages are rendered in the user's browser.

Example: index.html displaying the Wandria homepage.

MariaDB

Definition: A relational database management system used to store persistent system data.

Synonyms: Database (DB)

Related Terms: Backend, Persistent Storage

REST API

Definition: A web service architecture used to enable communication between frontend and backend components.

Acronym: API – Application Programming Interface

Related Terms: Endpoint, HTTP, Backend

Endpoint

Definition: A specific URL that allows communication between client and server.

Related Terms: REST API, Backend, Frontend

Authentication

Definition: The process of verifying a user's identity using credentials such as username and password.

Related Terms: User, Authorization, RBAC

Authorization

Definition: The process of granting or denying access to system features based on user role.

Related Terms: Authentication, RBAC, User

Functional Requirement (FR)

Definition: A requirement that describes what the system must do.

Non-Functional Requirement (NFR)

Definition: A requirement that describes system quality attributes such as performance or security.

Joy

EPIC 1 – Authentication & Account Access

US-01 – Secure Login

As a user, I want to log into my account so that I can manage my bookings and flight selections.

US-02 – Secure Logout

As a user, I want to log out of my account so that my personal booking information remains secure.

EPIC 2 – Flight Catalog Browsing

US-03 – Browse Available Flights

As a user, I want to browse available flights so that I can compare destinations, schedules, and prices before booking.

US-04 – Search Flights by Destination or Date

As a user, I want to search flights by destination and travel date so that I can quickly find flights that match my travel plans.

US-05 – View Flight Details

As a user, I want to view detailed flight information so that I can make an informed booking decision.

EPIC 3 – Booking Cart (Collection)

US-06 – Add Flight to Booking Cart

As a user, I want to add selected flights to my booking cart so that I can review my choices before confirming.

US-07 – Remove Flight from Booking Cart

As a user, I want to remove flights from my booking cart so that I can adjust my travel plans before checkout.

US-08 – Finalize Booking (Commit)

As a user, I want to confirm my selected flights so that my seats are officially reserved.

US-09 – Persistent Booking Cart

As a user, I want my booking cart to remain saved even after I log out so that I can continue my booking later.

EPIC 4 – Airline Administrator (Owner)

US-10 – Add New Flight

As an airline administrator, I want to add new flights to the system so that travelers can book newly available routes.

US-11 – Update Flight Information

As an airline administrator, I want to update flight schedules, prices, and seat availability so that travelers always see accurate information.

US-12 – Remove Flight from Catalog

As an airline administrator, I want to remove flights from the catalog so that travelers cannot book unavailable routes.

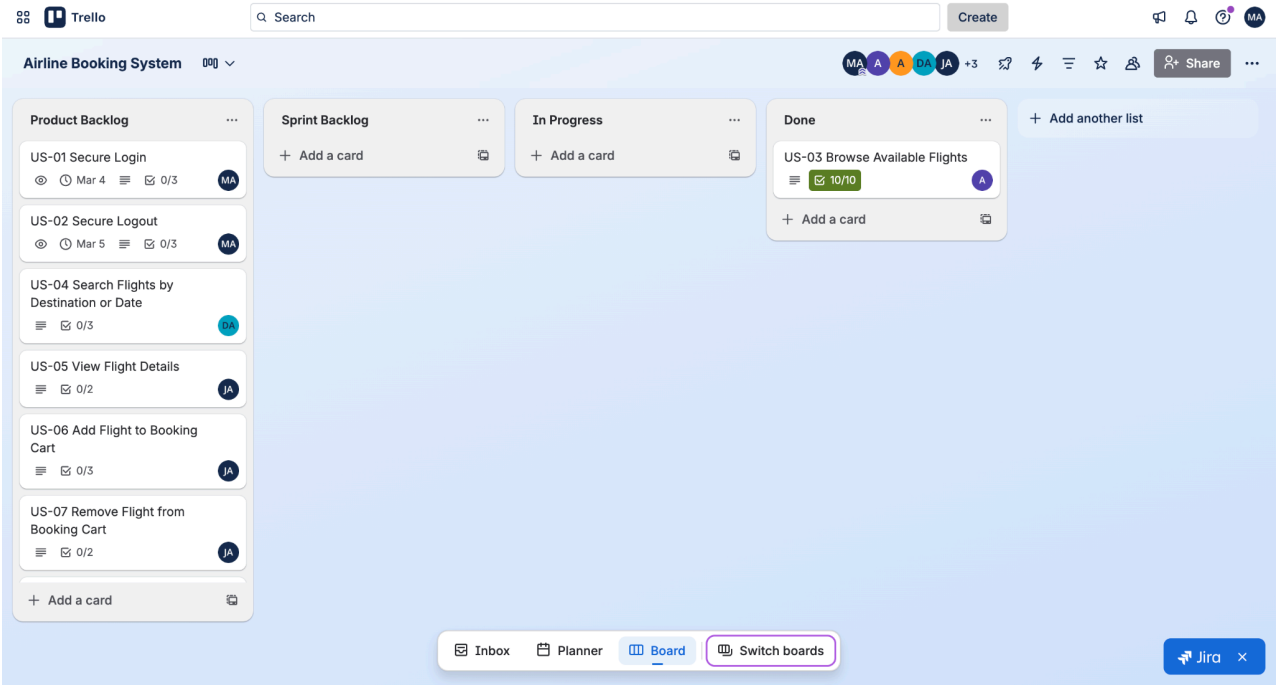
EPIC 5 – Security & Privacy

US-13 – Private Booking Cart

As a user, I want my booking cart to be private so that my selected flights are not visible to other users or administrators.

Madya

Trello:



https://trello.com/invite/b/6999c35404bfaa65066e2ae0/ATTI49b7cfb0e89566287b91fb7eb16cfdfb44C9F6EE/airline-booking-system

Traceability Matrix:

FR ID	Functional Requirement	US ID	User Story	Implementation Task
FR-001	System shall allow users to log in	US-01	Secure Login	Create login endpoint
FR-003	System shall restrict functions based on user role	US-01	Secure Login	Implement role-based access control
NFR-001	System shall securely store user passwords using encryption	US-01	Secure Login	Implement password hashing
FR-002	System shall allow users to log out	US-02	Secure Logout	Implement logout endpoint
FR-010	System will display all available flights	US-03	Browse Available Flights	Create Flight model class
FR-010	System will display all available flights	US-03	Browse Available Flights	Create FlightService
FR-010	System will display all available flights	US-03	Browse Available Flights	Create FlightController with @GetMapping('/flights')
FR-013	System will display flight number destination date and available seats	US-03	Browse Available Flights	Create flights.mustache template
FR-014	System shall display availability status	US-03	Browse Available Flights	Implement Available/Sold Out logic
NFR-004	System shall provide acceptable response time	US-03	Browse Available Flights	Optimize controller response logic
FR-010	System will display all available flights	US-03	Browse Available Flights	Add styling in style.css
FR-011	System will allow users to search flights by destination	US-04	Search Flights by Destination or Date	Implement search by destination endpoint
FR-012	System shall allow users to search flights by travel date	US-04	Search Flights by Destination or Date	Implement search by date endpoint
FR-013	System will display flight information in detail	US-05	View Flight Details	Create detailed flight view page
FR-020	System shall assign each customer exactly one booking cart	US-06	Add Flight to Booking Cart	Ensure cart is created automatically per customer
FR-021	System will allow customers to add flights to booking cart	US-06	Add Flight to Booking Cart	Create POST add-to-cart endpoint
FR-023	System will display all flights currently in the booking cart	US-06	Add Flight to Booking Cart	Create GET/cart endpoint
FR-022	System shall allow customers to remove flights from booking cart	US-07	Remove Flight from Booking Cart	Create DELETE remove-from-cart endpoint
FR-030	System shall allow customers to commit selected flights and create booking	US-08	Finalize Booking (Commit)	Implement commit booking logic
FR-031	System will require at least one flight in booking	US-08	Finalize Booking (Commit)	Validate cart is not empty before commit
FR-032	System will update seat availability after booking	US-08	Finalize Booking (Commit)	Reduce available seats after commit
FR-033	System shall clear booking cart after successful booking	US-08	Finalize Booking (Commit)	Clear cart after booking
FR-034	System shall store and manage booking status value	US-08	Finalize Booking (Commit)	Add booking status field (Pending/Confirmed)
FR-024	System shall persist cart items in database	US-09	Persistent Booking Cart	Store cart data in MariaDB
NFR-003	System shall persist all data in MariaDB	US-09	Persistent Booking Cart	Configure MariaDB persistence
FR-040	System shall allow administrators to add new flights	US-10	Add New Flight	Create POST /flights endpoint
FR-041	System shall allow administrators to update flight details	US-11	Update Flight Information	Create PUT /flights endpoint
FR-042	System shall allow administrators to remove flights	US-12	Remove Flight from Catalog	Create DELETE /flights endpoint
FR-025	System shall restrict cart access to owning customer	US-13	Private Booking Cart	Implement cart ownership validation
NFR-002	System shall restrict unauthorized access to protected endpoints	US-13	Private Booking Cart	Secure endpoints with authentication

https://docs.google.com/spreadsheets/d/1qqbs3zcrVFXm439ohfdp7rLJMMd8G21_1qi0N8dOSZ4/edit?usp=sharing

Ahmad

Ahmad Younes
Flight Class (Model)

```
package team2.wandria.model;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Flight {

   private String flightNumber;
   private String destination;
   private LocalDateTime departureTime;  // Date + Time
   private int seats;
   private double price;

   public Flight(String flightNumber, String destination,
                 LocalDateTime departureTime, int seats, double price) {
       this.flightNumber = flightNumber;
       this.destination = destination;
       this.departureTime = departureTime;
       this.seats = seats;
       this.price = price;
   }

   public String getFlightNumber() {
       return flightNumber;
   }

   public String getDestination() {
       return destination;
   }

   public LocalDateTime getDepartureTime() {
       return departureTime;
   }

   public int getSeatsAvailable() {
       return seats;
   }

   public double getPrice() {
       return price;
   }

   public String getFormattedDepartureTime() {
       DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MMM dd, yyyy
HH:mm");
       return departureTime.format(formatter);
```

```
    }

    public String getAvailabilityStatus() { //get availability for whether a
flight can hold more people based on whether there is seats or nor
        if (seats > 0) {
            return "Available";
        } else {
            return "Sold Out";
        }
    }
}
```

FlightController Class (Controller)

```java
package team2.wandria.controller;

import team2.wandria.service.FlightService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class FlightController {

   private final FlightService flightService;

   public FlightController(FlightService flightService) {
       this.flightService = flightService;
   }

   @GetMapping({"/flights"}) //return the view of available flights using
mustache templat
   public String getFlights(Model model) {
       model.addAttribute("flights", flightService.findAll());
       return "flights";
   }
}
```

FlightService

```
package team2.wandria.service;

import team2.wandria.model.Flight;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

@Service
public class FlightService {

   private final List<Flight> flights = new ArrayList<>();

   public FlightService() { //temporary memory data until a database is
implemented in future sprints
       flights.add(new Flight("EK202", "Dubai",
       LocalDateTime.of(2026, 3, 15, 14, 30),
       120, 950.0));

       flights.add(new Flight("QR811", "New York",
       LocalDateTime.of(2026, 3, 16, 9, 15),                0, 720.0));

       flights.add(new Flight("BA108", "London",
       LocalDateTime.of(2026, 3, 20, 18, 45),
       40, 1800.0));

       flights.add(new Flight("LH631", "America",
       LocalDateTime.of(2026, 3, 18, 7, 30),
       15, 1400.0));

       flights.add(new Flight("AF655", "Paris",
       LocalDateTime.of(2026, 3, 22, 21, 0),
       8, 1500.0));
   }

   public List<Flight> findAll() {

       return Collections.unmodifiableList(flights);
   }
}
```

flights.mustache (view)

```
<!DOCTYPE html>
<html>
<head>
   <title>Wandria Flights</title>
   <link rel="stylesheet" href="/style.css"></head>
<body>

<h1>Available Flights</h1>

<table>
   <tr>
       <th>Flight #</th>
       <th>Destination</th>
       <th>Date</th>
       <th>Seats</th>
       <th>Price</th>
       <th>Status</th>
   </tr>

   {{#flights}}
       <tr>
           <td>{{flightNumber}}</td>
           <td>{{destination}}</td>
           <td>{{formattedDepartureTime}}</td>
           <td>{{seats}}</td>
           <td>{{price}}</td>
           <td>{{availabilityStatus}}</td>
       </tr>
   {{/flights}}

</table>

</body>
</html>
```

Style.css for the mustache flights.mustache page

```
body {
   font-family: Arial, sans-serif;
   margin: 40px;
}

table {
   border-collapse: collapse;
   width: 100%;
}

th, td {
   border: 1px solid black;
```

```
    padding: 8px;
}
```