UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

Airline Reservation System

Submitted by:

Trisha Shukla	**PES1UG21CS676**
Tanishq Yaduka	**PES1UG21CS661**
Thanush Lodha	**PES1UG21CS672**
Swastika Sinha	**PES1UG21CS653**

6th Semester K Section

Prof. Bhargavi Mokashi
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Table of Contents

Introduction

The Airline Reservation System project aims to develop a comprehensive software solution that facilitates efficient management of airline operations, including passenger management, flight scheduling, and reservation processing. The system will offer an interactive user interface for creating, updating, and deleting passenger profiles, adding flights and their schedules, as well as enabling passengers to make and update reservations seamlessly.

Key Objectives:

Passenger Management:
- Create, update, and delete passenger profiles with relevant information such as name, contact details, and frequent flyer information.
- Implement authentication and authorization mechanisms to ensure data security and privacy.

Flight and Schedule Management:
- Add new flights to the system with details like flight number, origin, destination, departure, and arrival times.
- Manage flight schedules including regular schedules, special flights, and cancellations.

Reservation System:
- Provide an intuitive interface for passengers to search for flights based on criteria like date, time, origin, and destination.
- Enable passengers to make reservations by selecting available flights and specifying passenger details.
- Allow passengers to update or cancel their reservations as needed.

Interactive User Interface:
- Design a user-friendly interface for easy navigation and seamless interaction with the system.
- Incorporate feedback mechanisms to gather user inputs and improve system usability over time.

Use Cases

Flight Entity

1. Create a New Flight
    - Description: Allows administrators to add a new flight to the system.
    - Attributes Used: flightNumber, price, origin, destination, departureTime, arrivalTime, seatsLeft, description, plane, passengers
    - Operations Involved: Setter methods for setting flight details, including plane and passengers.
    - Constraints Handled: Ensures the uniqueness of flightNumber.
2. Update Flight Information
    - Description: Enables administrators to modify details of an existing flight.
    - Attributes Used: All flight attributes except flightNumber.
    - Operations Involved: Setter methods for updating flight details.
    - Constraints Handled: Validates input data such as price, seat availability, and departure/arrival times.
3. Retrieve Flight Information
    - Description: Allows users to view details of a specific flight.
    - Attributes Used: flightNumber, origin, destination, departureTime, arrivalTime, seatsLeft, description, plane, passengers
    - Operations Involved: Getter methods to retrieve flight details.

Plane Entity

1. Add a New Plane
    - Description: Administrators can add a new plane to the system.
    - Attributes Used: model, capacity, manufacturer, yearOfManufacture
    - Operations Involved: Setter methods for setting plane details.
    - Constraints Handled: Validates input data such as capacity and year of manufacture.
2. Update Plane Information
    - Description: Allows administrators to modify details of an existing plane.
    - Attributes Used: All plane attributes except id.
    - Operations Involved: Setter methods for updating plane details.
3. Retrieve Plane Information
    - Description: Users can view details of a specific plane.
    - Attributes Used: id, model, capacity, manufacturer, yearOfManufacture
    - Operations Involved: Getter methods to retrieve plane details.

Reservation Entity

1. Create a New Reservation
    - Description: Enables passengers to make a reservation for one or more flights.
    - Attributes Used: reservationNumber, origin, destination, price, passenger, flights
    - Operations Involved: Setter methods for setting reservation details, including passenger and flights.
2. Update Reservation Details

- Description: Allows passengers or administrators to modify reservation details such as origin, destination, or flights.
- Attributes Used: All reservation attributes except reservationNumber.
- Operations Involved: Setter methods for updating reservation details.

3. Retrieve Reservation Information
    - Description: Users can view details of a specific reservation.
    - Attributes Used: reservationNumber, origin, destination, price, passenger, flights
    - Operations Involved: Getter methods to retrieve reservation details.

Passenger Operations

1. View Passenger Profile
    - Description: Allows users to view their profile information.
    - HTTP Method: GET
    - Endpoint: /passenger/{id}
    - Parameters:
        - id: Passenger ID
        - xml (optional): XML format flag
    - Response: Returns the passenger's profile details.
    - Exception Handling: Handles NotFoundException by returning a bad request response with a custom error message.
2. Update Passenger Details
    - Description: Enables users to update their profile details.
    - HTTP Method: PUT
    - Endpoint: /passenger/{id}
    - Parameters:
        - id: Passenger ID
        - firstname, lastname, age, gender, phone: Updated profile details

        - xml (optional): XML format flag
    - Response: Returns a success message if the update is successful.
    - Exception Handling: Handles IllegalArgumentException and NotFoundException with appropriate error messages.
3. Delete Passenger Account
    - Description: Allows users to delete their account, including associated reservations.
    - HTTP Method: DELETE
    - Endpoint: /passenger/{id}
    - Parameters:
        - id: Passenger ID
        - xml (optional): XML format flag
    - Response: Returns a success message upon successful deletion.
    - Exception Handling: Handles NotFoundException by returning a bad request response with an error message.
4. Register New Passenger
    - Description: Enables new users to register by providing necessary information.
    - HTTP Method: POST
    - Endpoint: /passenger
    - Parameters:
        - firstname, lastname, age, gender, phone: New passenger details
        - xml (optional): XML format flag
    - Response: Returns a success message upon successful registration.
    - Exception Handling: Handles IllegalArgumentException with an appropriate error message.

Flight Operations

Description: Display all flights available.
HTTP Method: GET
Endpoint: /displayflights
Response: Returns a list of Flight objects.
Exception Handling: Handles NotFoundException with a 404 status code and an appropriate error message.

Operation 2:
Description: Get flight details by flight number.
HTTP Method: GET
Endpoint: /flight/{flightNumber}
Parameters:
- flightNumber: Path variable indicating the flight number.
- xml (optional): Flag for XML format.

Response: Returns details of the flight with the specified flight number.
Exception Handling: Handles NotFoundException with a 404 status code and an appropriate error message.

Operation 3:
Description: Delete a flight by flight number.
HTTP Method: DELETE
Endpoint: /airline/{flightNumber}
Parameters:
- flightNumber: Path variable indicating the flight number.
- xml (optional): Flag for XML format.

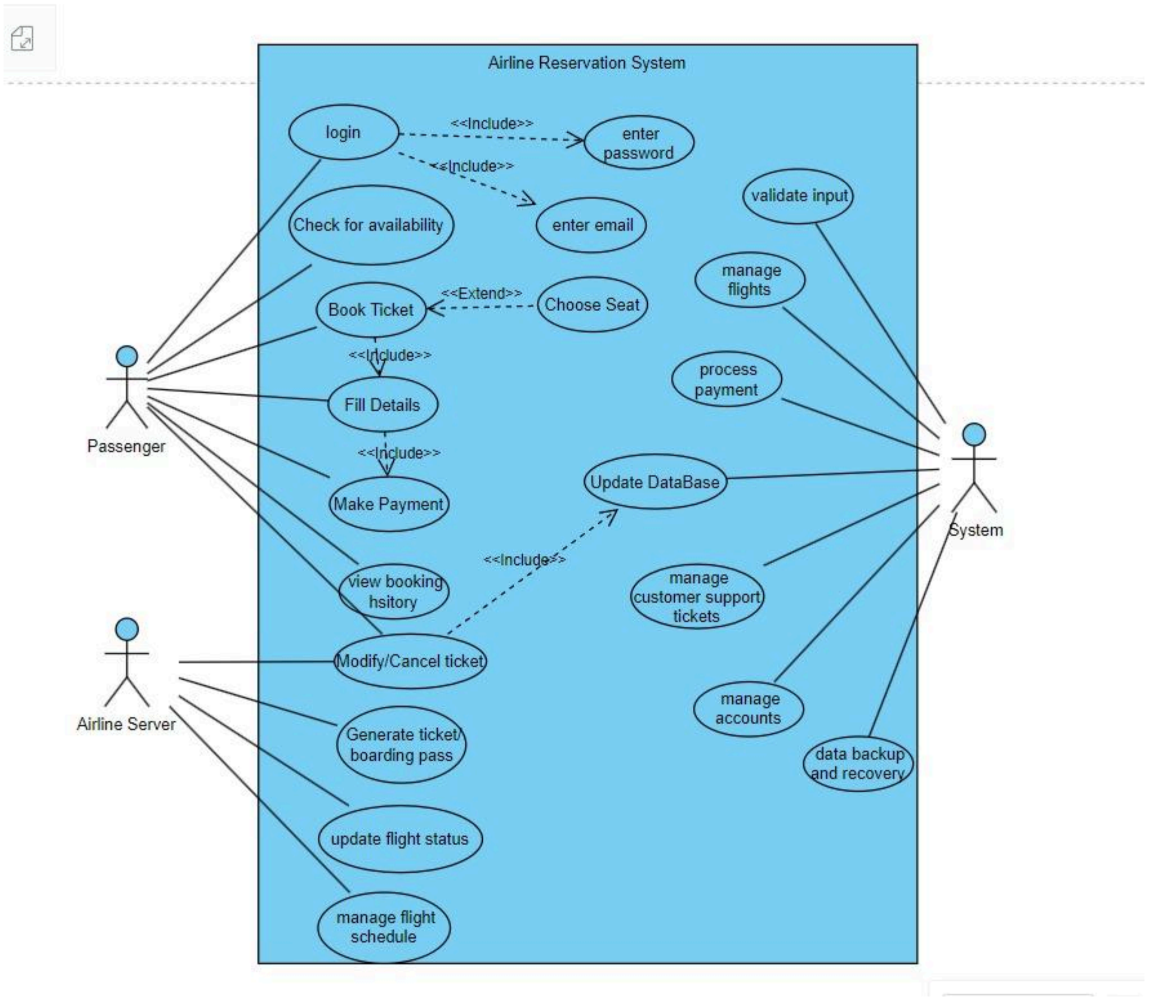
Response: Returns a success message upon successful deletion of the flight.
Exception Handling:
- Handles NotFoundException with a 404 status code and an appropriate error message.
- Handles IllegalArgumentException with a 400 status code and an appropriate error message.

Operation 4:
Description: Update flight details.
HTTP Method: POST

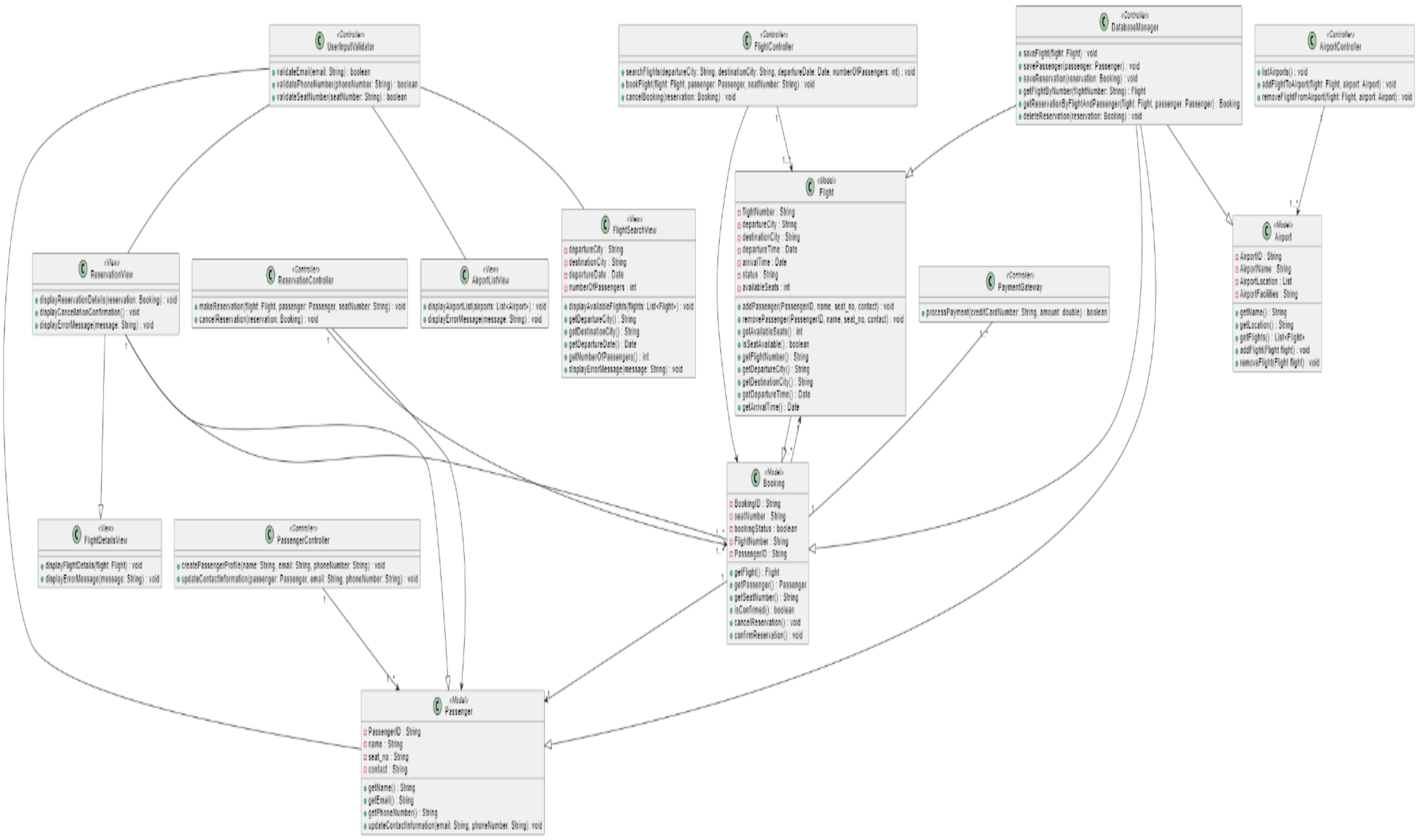
Endpoint: /flight/{flightNumber}
Parameters:
- flightNumber: Path variable indicating the flight number.
- price: Request parameter for the updated price.
- origin: Request parameter for the updated origin.
- destination: Request parameter for the updated destination.
- departureTime: Request parameter for the updated departure time.
- arrivalTime: Request parameter for the updated arrival time.
- description: Request parameter for the updated description.
- capacity: Request parameter for the updated capacity.
- model: Request parameter for the updated model.
- manufacturer: Request parameter for the updated manufacturer.
- yearOfManufacture: Request parameter for the updated year of manufacture.
- xml (optional): Flag for XML format.

Response: Returns updated details of the flight.
Exception Handling:
- Handles IllegalArgumentException with a 400 status code and an appropriate error message.
- Handles ParseException with a 400 status code and an appropriate error message.

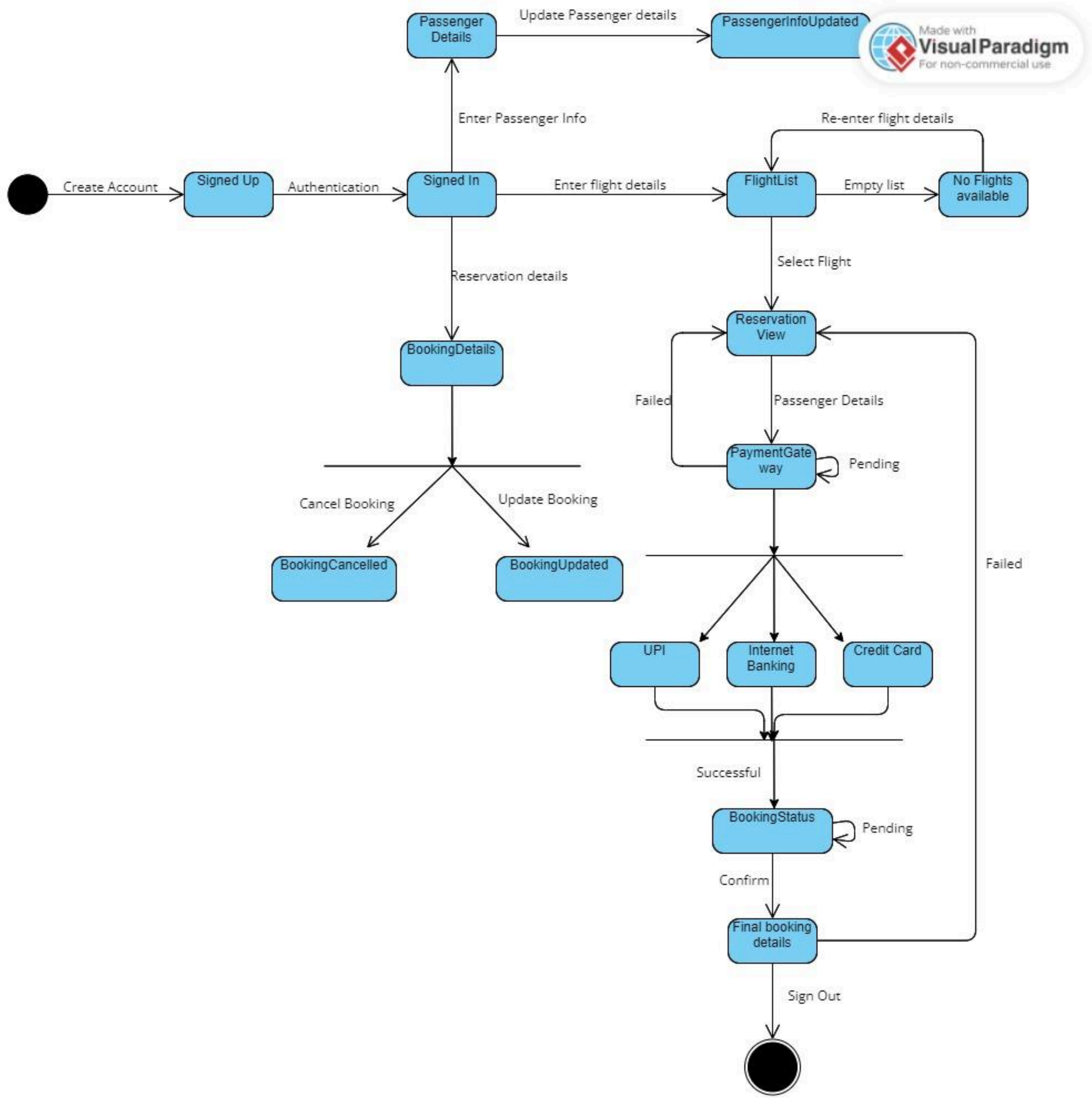
Use Case Diagram



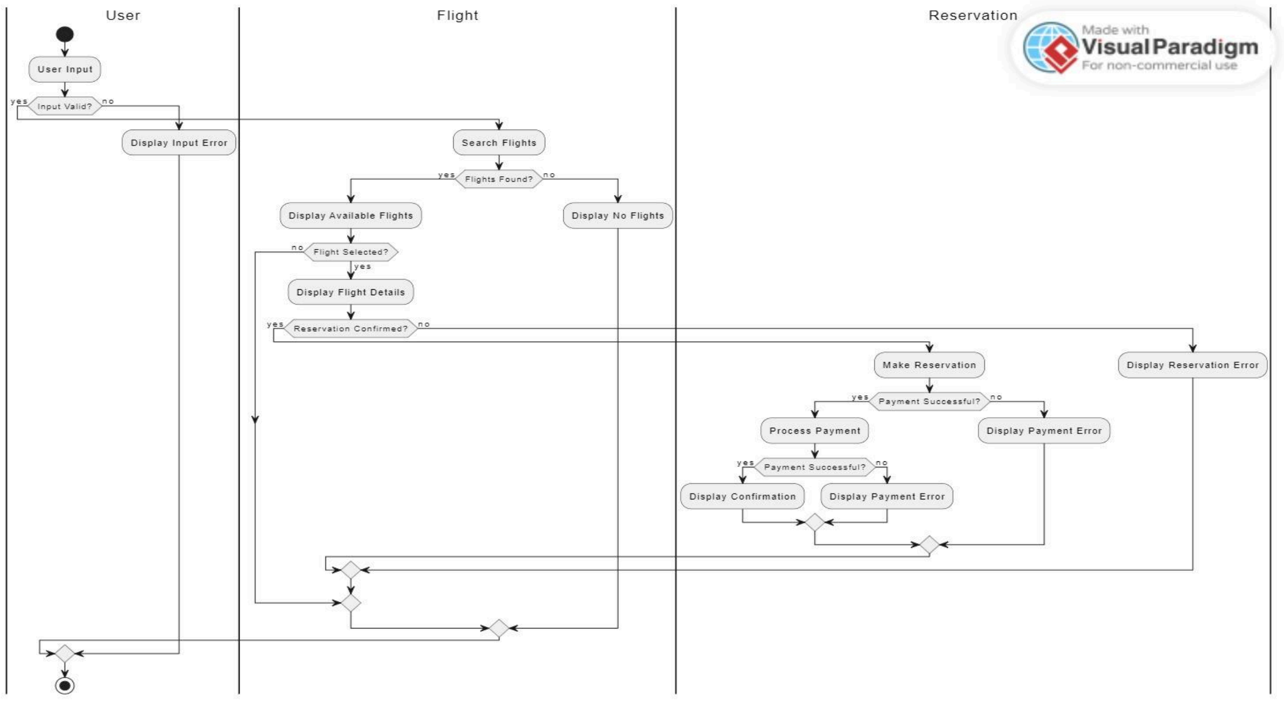
Class Diagram

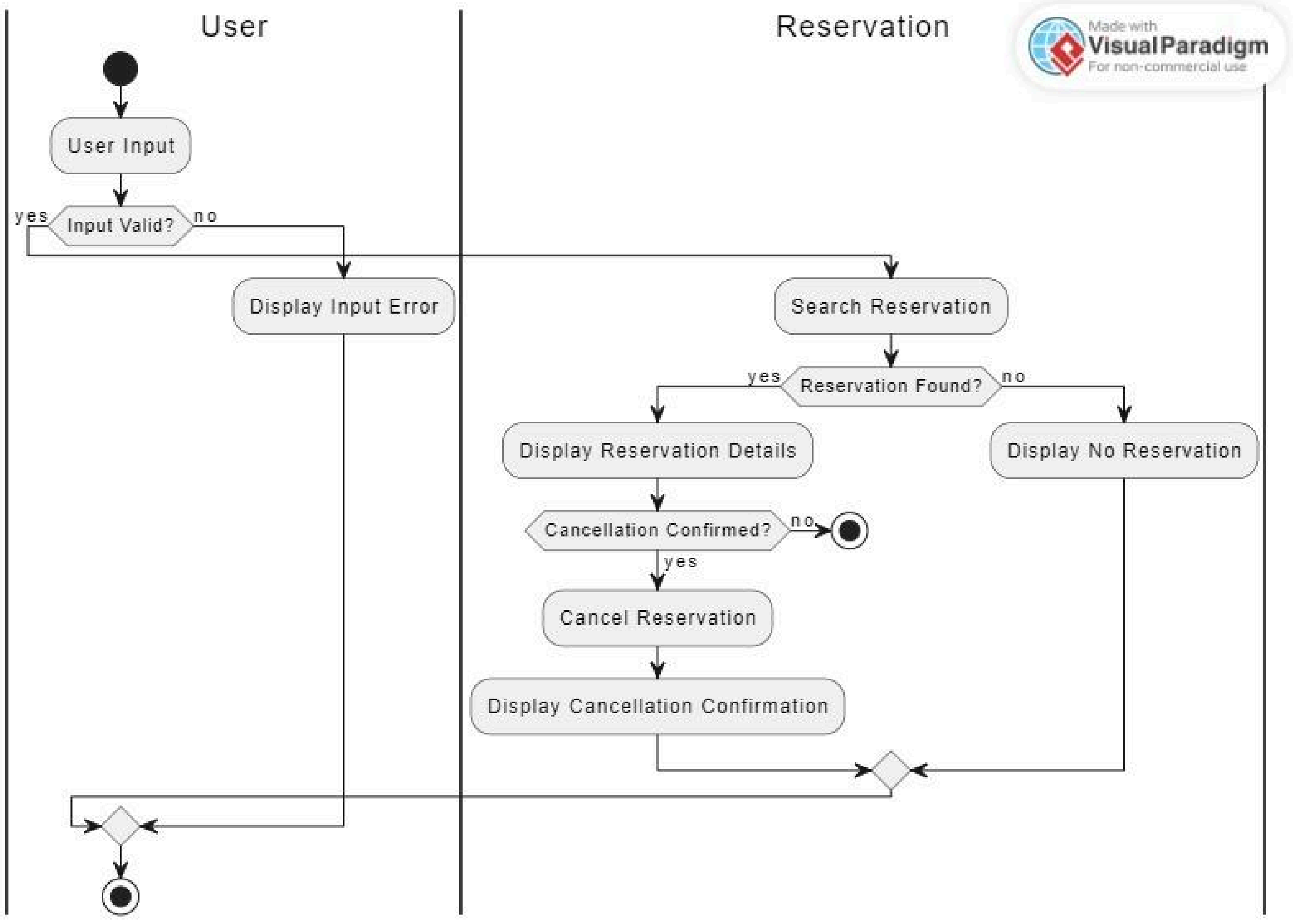

«Controller» UserInputValidator
validateEmail(email: String) : boolean
validatePhoneNumber(phoneNumber: String) : boolean
validateSeatNumber(seatNumber: String) : boolean
«Controller» FlightController
searchFlights(departureCity: String, destinationCity: String, departureDate: Date, numberOfPassengers: int) : void
bookFlight(flight: Flight, passenger: Passenger, seatNumber: String) : void
cancelBooking(reservation: Booking) : void
«Controller» DatabaseManager
saveFlight(flight: Flight) : void
savePassenger(passenger: Passenger) : void
saveReservation(reservation: Booking) : void
getFlightByNumber(flightNumber: String) : Flight
getReservationByFlightAndPassenger(flight: Flight, passenger: Passenger) : Booking
deleteReservation(reservation: Booking) : void
«Controller» AirportController
listAirports() : void
addFlightToAirport(flight: Flight, airport: Airport) : void
removeFlightFromAirport(flight: Flight, airport: Airport) : void
«View» ReservationView
displayReservationDetails(reservation: Booking) : void
displayCancellationConfirmation() : void
displayErrorMessage(message: String) : void
«Controller» ReservationController
makeReservation(flight: Flight, passenger: Passenger, seatNumber: String) : void
cancelReservation(reservation: Booking) : void
«View» AirportListView
displayAirportList(airports: List<Airport>) : void
displayErrorMessage(message: String) : void
«View» FlightSearchView
departureCity : String
destinationCity : String
departureDate : Date
numberOfPassengers : int
displayAvailableFlights(flights: List<Flight>) : void
getDepartureCity() : String
getDestinationCity() : String
getDepartureDate() : Date
getNumberOfPassengers() : int
displayErrorMessage(message: String) : void
«Model» Flight
flightNumber : String
departureCity : String
destinationCity : String
departureTime : Date
arrivalTime : Date
status : String
availableSeats : int
addPassenger(PassengerID, name, seat_no, contact) : void
removePassenger(PassengerID, name, seat_no, contact) : void
getAvailableSeats() : int
isSeatAvailable() : boolean
getFlightNumber() : String
getDepartureCity() : String
getDestinationCity() : String
getDepartureTime() : Date
getArrivalTime() : Date
«Controller» PaymentGateway
processPayment(creditCardNumber: String, amount: double) : boolean
«Model» Airport
AirportID : String
AirportName : String
AirportLocation : List
AirportFacilities : String
getName() : String
getLocation() : String
getFlights() : List<Flight>
addFlight(Flight flight) : void
removeFlight(Flight flight) : void
«Model» Booking
BookingID : String
seatNumber : String
bookingStatus : boolean
FlightNumber : String
PassengerID : String
getFlight() : Flight
getPassenger() : Passenger
getSeatNumber() : String
isConfirmed() : boolean
cancelReservation() : void
confirmReservation() : void
«View» FlightDetailsView
displayFlightDetails(flight: Flight) : void
displayErrorMessage(message: String) : void
«Controller» PassengerController
createPassengerProfile(name: String, email: String, phoneNumber: String) : void
updateContactInformation(passenger: Passenger, email: String, phoneNumber: String) : void
«Model» Passenger
PassengerID : String
name : String
seat_no : String
contact : String
getName() : String
getEmail() : String
getPhoneNumber() : String
updateContactInformation(email: String, phoneNumber: String): void
1
1..*

State Diagram



Activity Diagram





Architecture Patterns

MVC Architecture Pattern

In recent years web applications have become incredibly complex applications with lots of dynamic functionality driven by user-friendly interfaces, complex business logic and large databases. To work with these complex web applications developers use different architecture patterns to lay out their projects, to make the code less complex and easier to work with. The most popular of these patterns is MVC also known as Model View Controller. The Model-View-Controller (MVC) architectural pattern separates an application into three main logical components Model, View, and Controller. Each architectural component is built to handle specific development aspects of an application. It isolates the data, business logic and presentation layer from each other. There are many frameworks that support MVC architecture. Spring framework is one of the most popular MVC frameworks with support for Java language and many SQL and NoSQL databases. Spring MVC Framework follows the Model-View-Controller architectural design pattern which works around the Front Controller i.e. the Dispatcher Servlet. The Dispatcher Servlet handles and dispatches all the incoming HTTP requests to the appropriate controller. It uses @Controller and @RequestMapping as default request handlers. The @Controller annotation defines that a particular class is a controller. @RequestMapping annotation maps web requests to Spring Controller methods.

Model Components

1. Flight Entity
    - Attributes: flightNumber, price, origin, destination, departureTime, arrivalTime, seatsLeft, description, plane, passengers
    - Use Cases: Create, Update, Retrieve Flight Information
2. Plane Entity
    - Attributes: model, capacity, manufacturer, yearOfManufacture
    - Use Cases: Add New Plane, Update Plane Information, Retrieve Plane Information
3. Reservation Entity
    - Attributes: reservationNumber, origin, destination, price, passenger, flights
    - Use Cases: Create New Reservation, Update Reservation Details, Retrieve Reservation Information
4. BadRequest Class
    - Attributes: code, message
    - Use: Represents custom error responses for bad requests
5. ExceptionHandle Class
    - Attributes: badRequest
    - Use: Utility class for handling exceptions and generating error responses
6. Response Class
    - Attributes: code, message
    - Use: Represents response objects with code and message attributes

View Components

1. Error Response Views
   - Use: Display custom error messages for bad requests
2. Success Response Views
   - Use: Display success messages for operations like creating or updating entities

Controller Components

1. FlightController
   - Endpoints: /flight/{flightNumber}, /airline/{flightNumber}, /flight/{flightNumber}
   - Use Cases: Fetch flight details, delete flight, create/update flight
2. PassengerController
   - Endpoints: /passenger/{id}, /passenger, /passenger/{id}
   - Use Cases: Update passenger details, delete passenger, create passenger, get passenger
3. ReservationController
   - Endpoints: /reservation/{number}, /reservation, /reservation/{number}
   - Use Cases: Get reservation, create reservation, update reservation, cancel reservation

Additional Components

1. Utility Classes
   - ExceptionHandle: For exception handling and error responses
   - BadRequest: Custom error object for bad requests
   - Response: Response object with code and message attributes
2. Lombok Annotations

    - @Data, @RequiredArgsConstructor: Used for generating getter, setter, constructor, and other utility methods in entities and utility classes
3. Global Exception Handling
    - Use of @ControllerAdvice and @ResponseBody for handling exceptions globally and returning structured error responses

Design Principles

- **Single Responsibility Principle (SRP):** The project adheres to the Single Responsibility Principle by ensuring that each class has a single responsibility or concern. For example, entities such as Flight, Plane, and Passenger are responsible for data representation and persistence. Utility classes like BadRequest, ExceptionHandle, and Response handle error handling and response generation. Controllers such as FlightController, PassengerController, and ReservationController manage request handling and interaction with services. This adherence to SRP promotes modularity, code clarity, and easier maintenance by keeping classes focused on specific tasks.

- **Open/Closed Principle (OCP):** The project follows the Open/Closed Principle by designing components that are open for extension but closed for modification. Controllers and services are designed to be open for extension through additional methods or functionalities. Entities can be extended with new attributes or behaviors without changing their base structure.

This design approach promotes code reusability, scalability, and flexibility by minimizing the need for code modifications when extending or enhancing the system.

- **Dependency Inversion Principle (DIP)**: The Dependency Inversion Principle is applied through dependency injection and inversion of control mechanisms. Controllers depend on service interfaces rather than concrete implementations, promoting loose coupling and easier testing. This inversion of dependencies reduces direct dependencies between components, leading to a more modular, maintainable, and testable codebase.

- **Separation of Concerns (SoC):** The project embraces the Separation of Concerns principle by separating different concerns into distinct components. Entities are responsible for data representation and persistence, services encapsulate business logic and operations, and controllers manage request handling and response generation. This separation promotes code organization, readability, and easier collaboration among team members by clearly delineating responsibilities.

<u>Design Patterns</u>

- **Proxy Pattern:** In the provided code, the Plane class is annotated with @Entity, indicating that it's an entity class managed by a persistence context. This usage aligns with the Proxy Pattern conceptually.When you work with entities in a JPA (Java Persistence API) environment, the JPA provider (like Hibernate) creates proxy objects for entities. These proxy objects act as placeholders or surrogates for actual entity instances. They control access to the entity's state and manage operations like lazy loading of associations.

- **Iterator Pattern:** The Iterator Pattern is used to provide a way to access elements of an aggregate object sequentially without exposing its underlying representation. It simplifies the traversal of collections or data structures.The iteration over lists is handled using standard Java APIs (foreach, stream, etc.), so the Iterator Pattern isn't strictly implemented.

- **Front Controller Pattern:** The project architecture aligns with the Front Controller pattern, where Spring MVC acts as a front controller by routing incoming HTTP requests to appropriate controllers based on request mappings. This centralized request handling promotes reusable components, consistent request processing, and a streamlined approach to managing HTTP requests across the application.

- **Singleton Pattern:** Spring manages singleton beans by default, ensuring that certain components such as service beans are instantiated only once and shared throughout the application context. This adherence to the Singleton pattern promotes resource management, shared state access, and centralized control over singleton instances in the application context.

Github Link

https://github.com/tanishqyaduka/AirlineReservationSystem

Individual Contributions

Trisha - Plane, Response

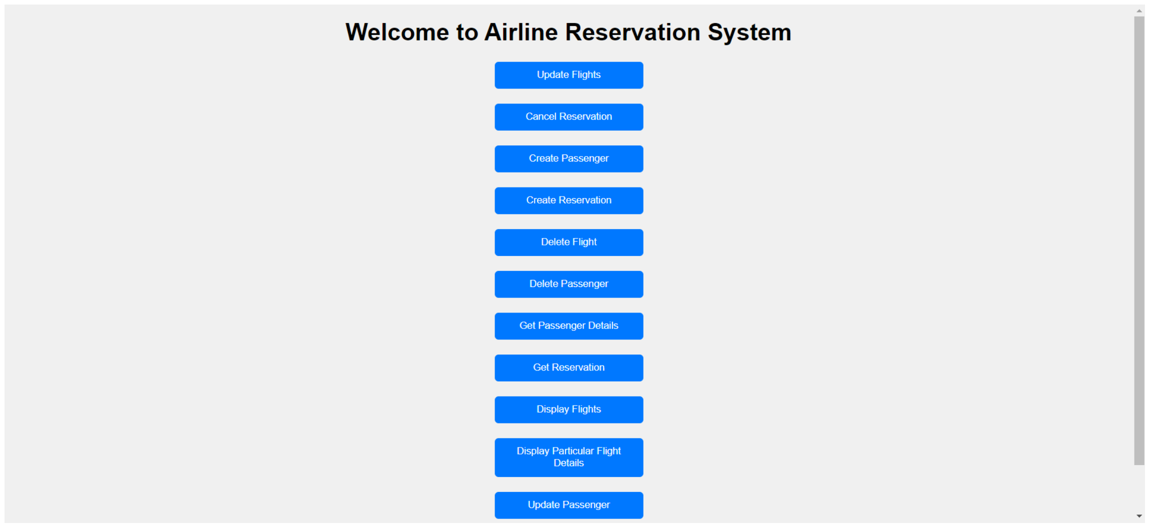
Tanishq - Passenger, Bad Request

Thanush - Flight, Exceptional Handle

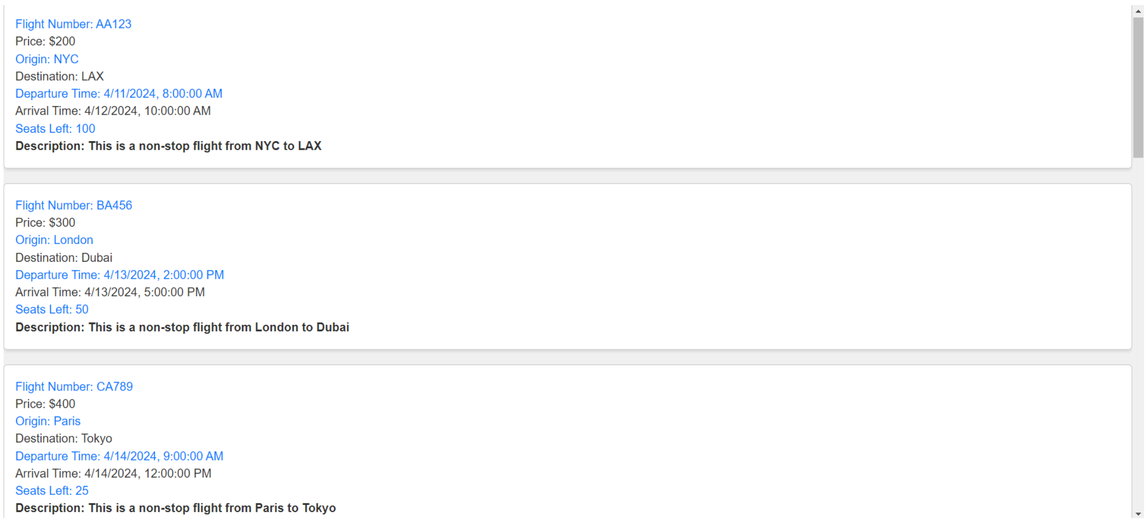
Swastika - Reservation, Home

Input and Output

- Home page



- Display Flights

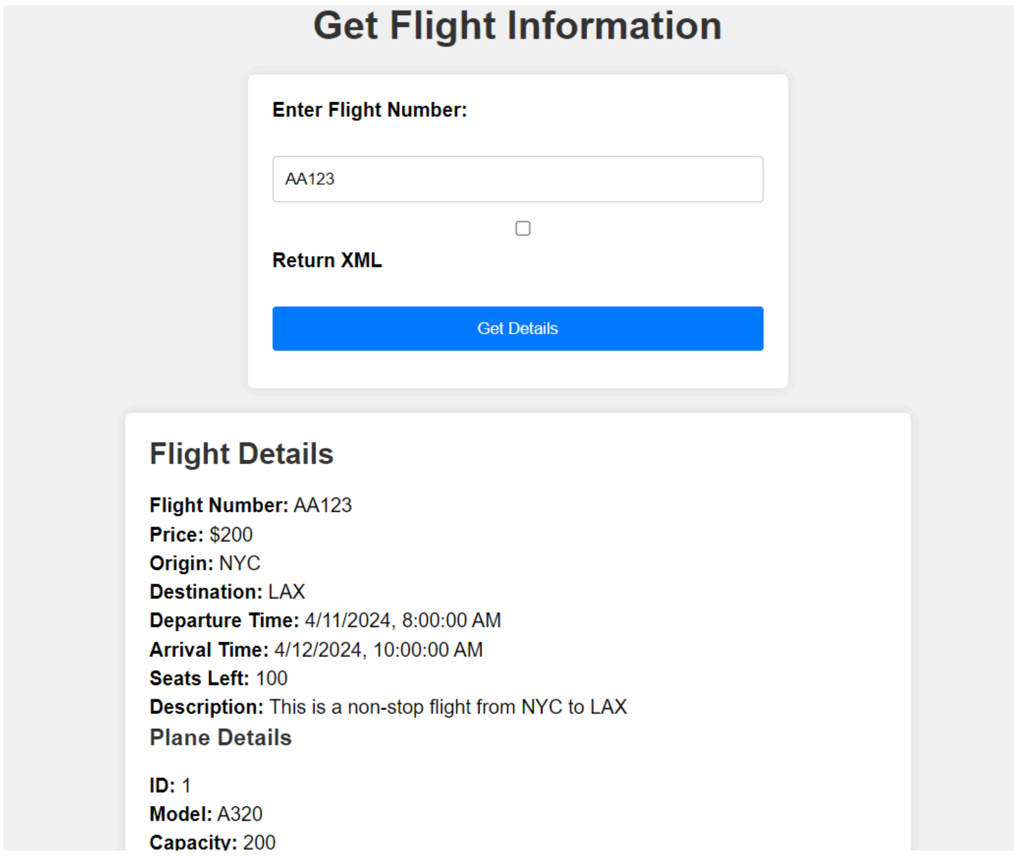


Flight Number: AA123
Price: $200
Origin: NYC
Destination: LAX
Departure Time: 4/11/2024, 8:00:00 AM
Arrival Time: 4/12/2024, 10:00:00 AM
Seats Left: 100
Description: This is a non-stop flight from NYC to LAX

Flight Number: BA456
Price: $300
Origin: London
Destination: Dubai
Departure Time: 4/13/2024, 2:00:00 PM
Arrival Time: 4/13/2024, 5:00:00 PM
Seats Left: 50
Description: This is a non-stop flight from London to Dubai

Flight Number: CA789
Price: $400
Origin: Paris
Destination: Tokyo
Departure Time: 4/14/2024, 9:00:00 AM
Arrival Time: 4/14/2024, 12:00:00 PM
Seats Left: 25
Description: This is a non-stop flight from Paris to Tokyo

- Get Flight Info



Get Flight Information

Enter Flight Number:

AA123

☐ **Return XML**

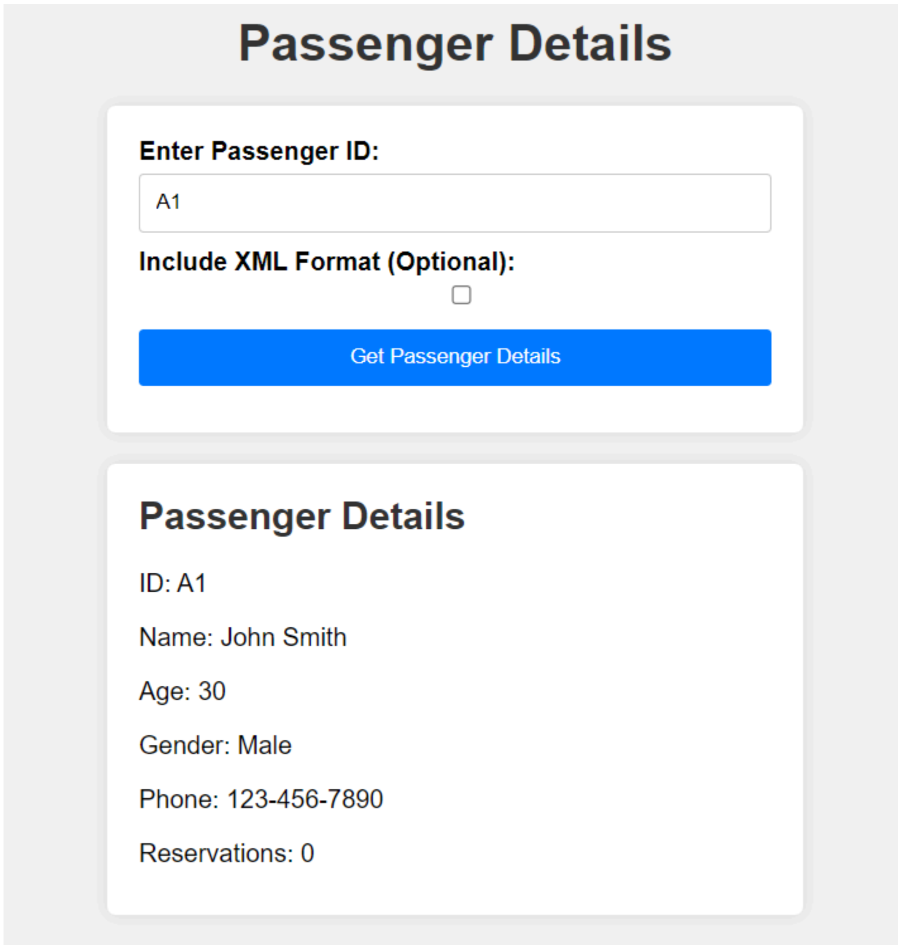
Flight Details

Flight Number: AA123
Price: $200
Origin: NYC
Destination: LAX
Departure Time: 4/11/2024, 8:00:00 AM
Arrival Time: 4/12/2024, 10:00:00 AM
Seats Left: 100
Description: This is a non-stop flight from NYC to LAX

Plane Details

ID: 1
Model: A320
Capacity: 200

- Get Passenger Details



Passenger Details

Enter Passenger ID:

A1

Include XML Format (Optional):

☐

Passenger Details

ID: A1

Name: John Smith

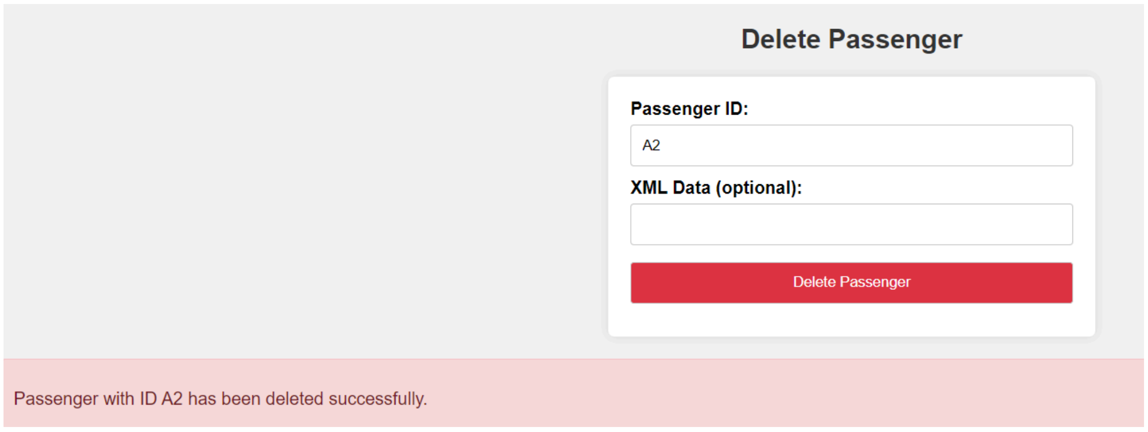
Age: 30

Gender: Male

Phone: 123-456-7890

Reservations: 0

- Delete Passenger



Delete Passenger

Passenger ID:

A2

XML Data (optional):

Passenger with ID A2 has been deleted successfully.