Picture 1 **FLIGHT MANAGEMENT SYSTEM**

**PREPARED FOR**

**JEE Full Stack BootCamp Programme**

**Capgemini Technology Services Pvt. Ltd.**

**PREPARED BY**

**Sanskar Dwivedi (Team Lead) Register+LogIn+Authentication Module**

**Rohit Agarwal (Team Member) Customer Modules**

**Ayushi Chaudhary (Team Member) Admin Modules**

**MENTORED BY**

**Ms. Varsha Lonkar**

**Abstract:**

The Flight Management System is a Java-based booking solution for flight tickets. It consolidates the data provided by different airline carriers and hence provides the user details and rates in real-time. Travellers may want to make changes in their bookings. ***The application allows them to book, cancel, view and update their bookings with ease.***Other than this, it eases the management of bookings too. All the bookings, flights, schedules and routes can be viewed, added and modified on a single application by the administrator.

**Scopes:**

**Inscope:**

Following is the functionality provided by the system:

There are two categories of people who would access the system: customer and administrator. Each of these would have some exclusive privileges.

1. The **customer** can:
   1. Create his user account.
   2. Login into the application.
   3. Check for available flights.
   4. Make a booking.
   5. View the bookings made.
   6. Cancel or modify a booking.
2. The **administrator** can:
   1. Login into the application.
   2. Add flight, schedule and route details.
   3. View the flight, schedule and route details.
   4. Cancel or modify the flight, schedule and route details.

**Outscope:**

The following functionalities have not been covered under the application:

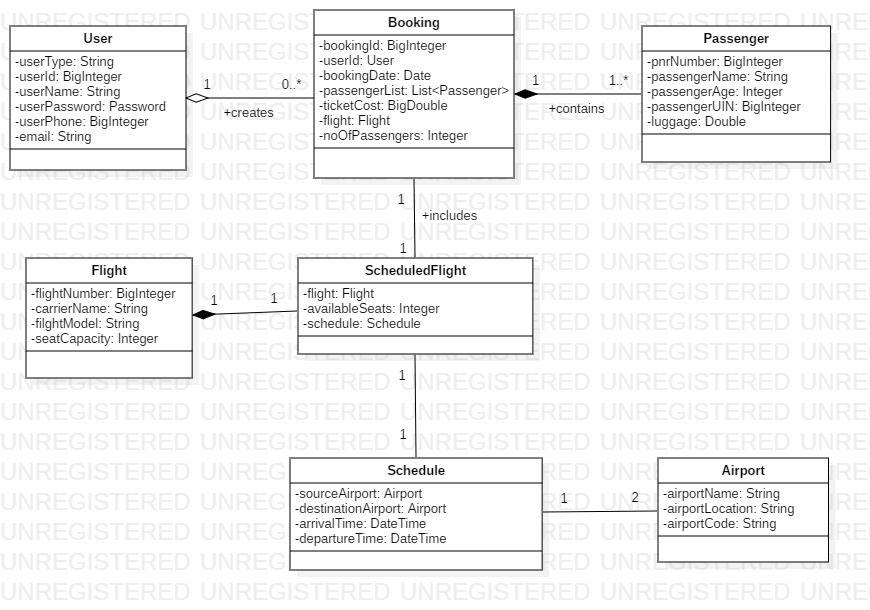
1. The application does not cover boarding pass generation and seating plans.
2. Third party applications like email & sms integrations.
3. Payments are not yet accepted by the application.

**Class Diagram:**

Class diagram is a static diagram. It represents the static view of an application. It is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

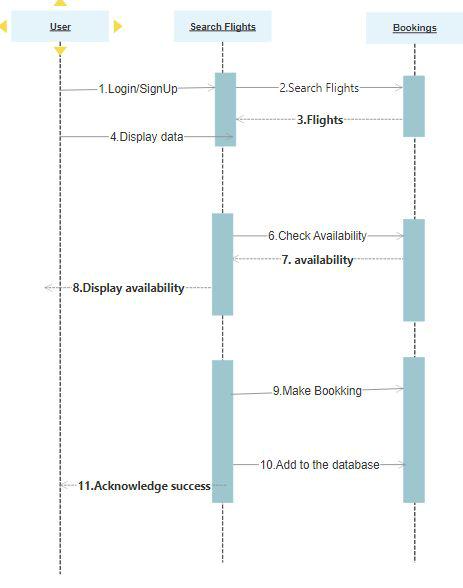
Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

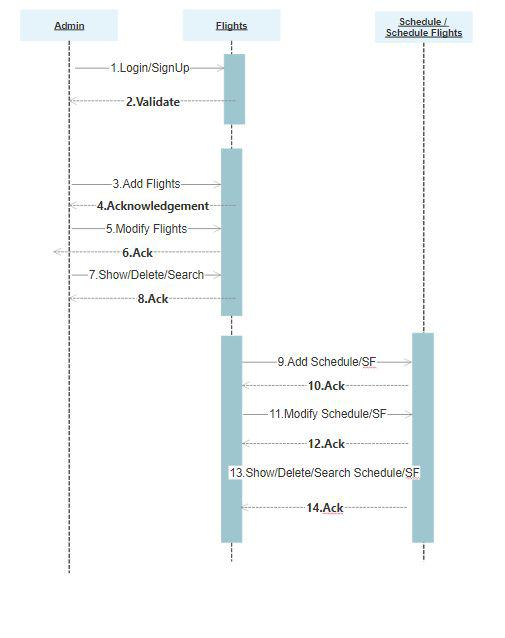
Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.



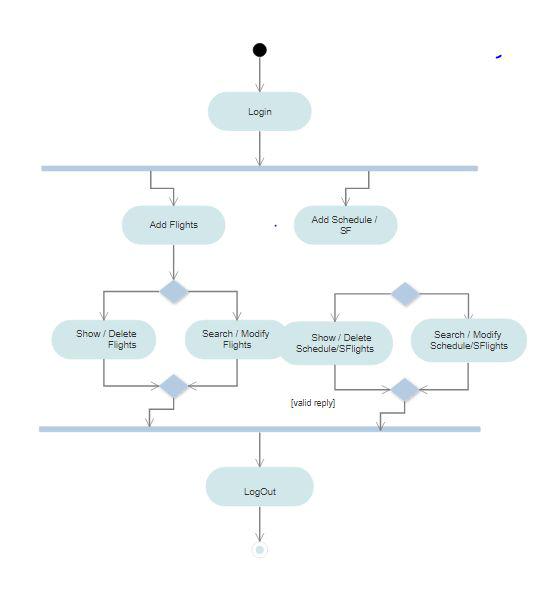
**Sequence Diagram for User:**

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function.

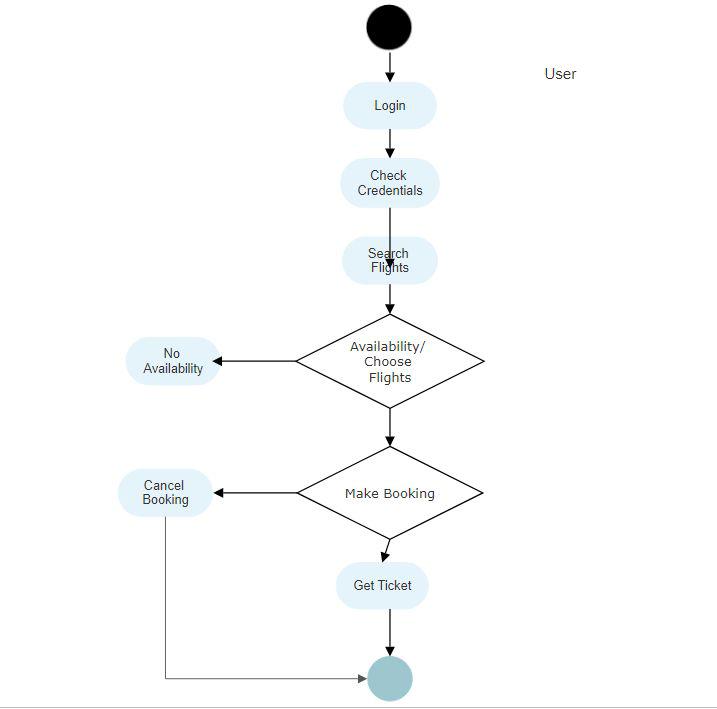


**Sequence Diagram for Admin:**

**Activity Diagram for Admin:**

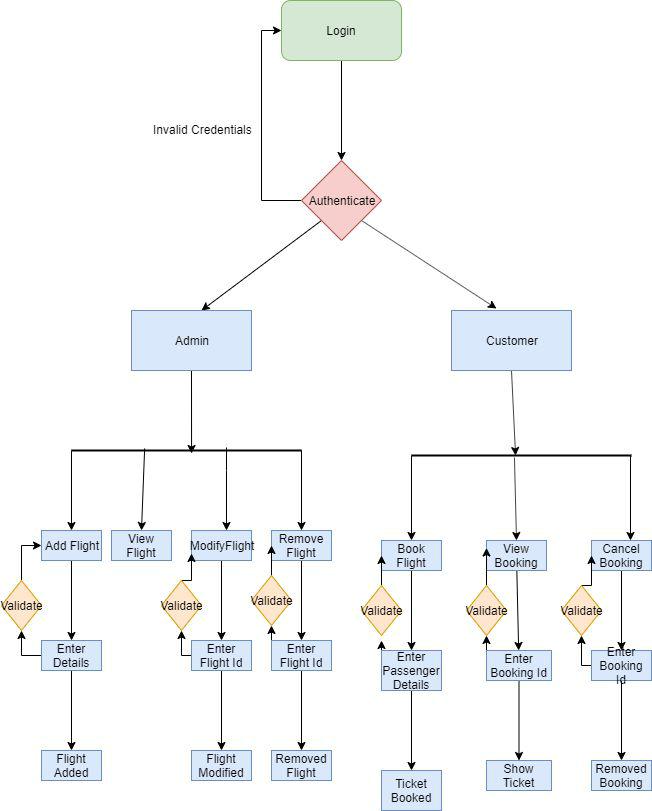
Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

**Activity Diagram for User:**



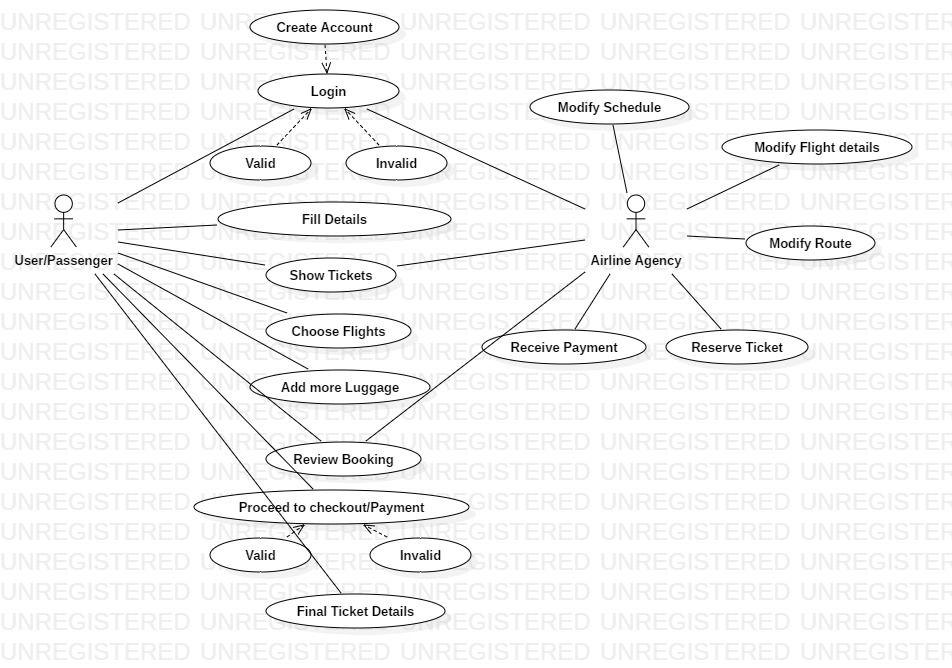
**Flow Chart:**

Flowcharts are used in designing and documenting simple processes or programs. Like other types of diagrams, they help visualise what is going on and thereby help understand a process, and perhaps also find less-obvious features within the process, like flaws and bottlenecks.



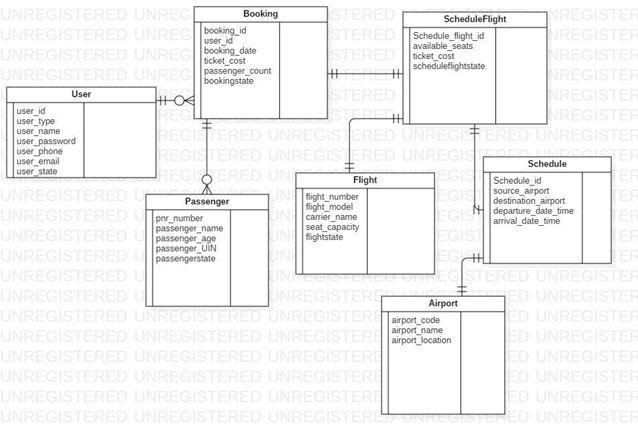
**Use Case Diagram:**

The purposes of use case diagrams can be said to be as follows−

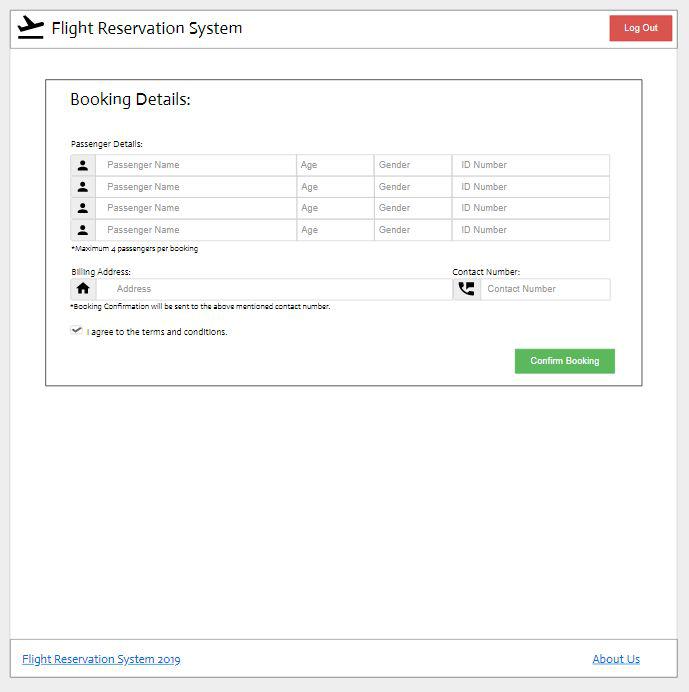
* Used to gather the requirements of a system.
* Used to get an outside view of a system.
* Identify the external and internal factors influencing the system.
* Show the interaction among the requirements are actors.

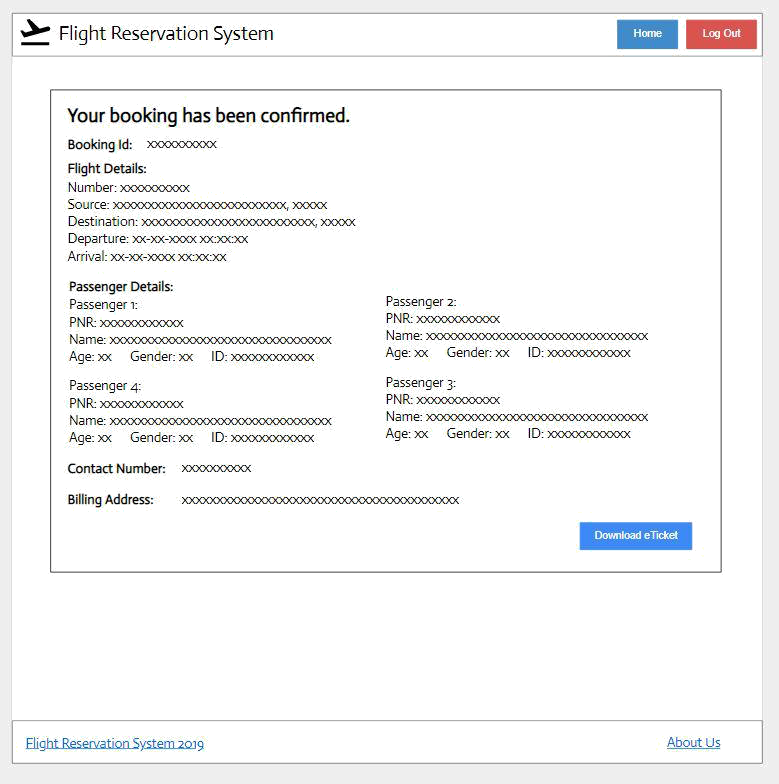
**Entity Relation Diagram:**

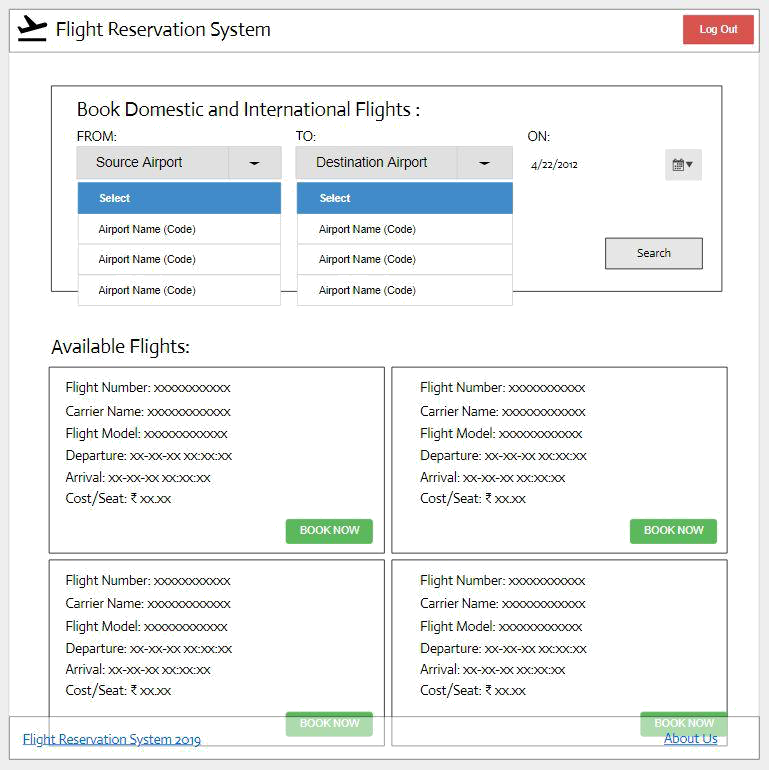
An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes.

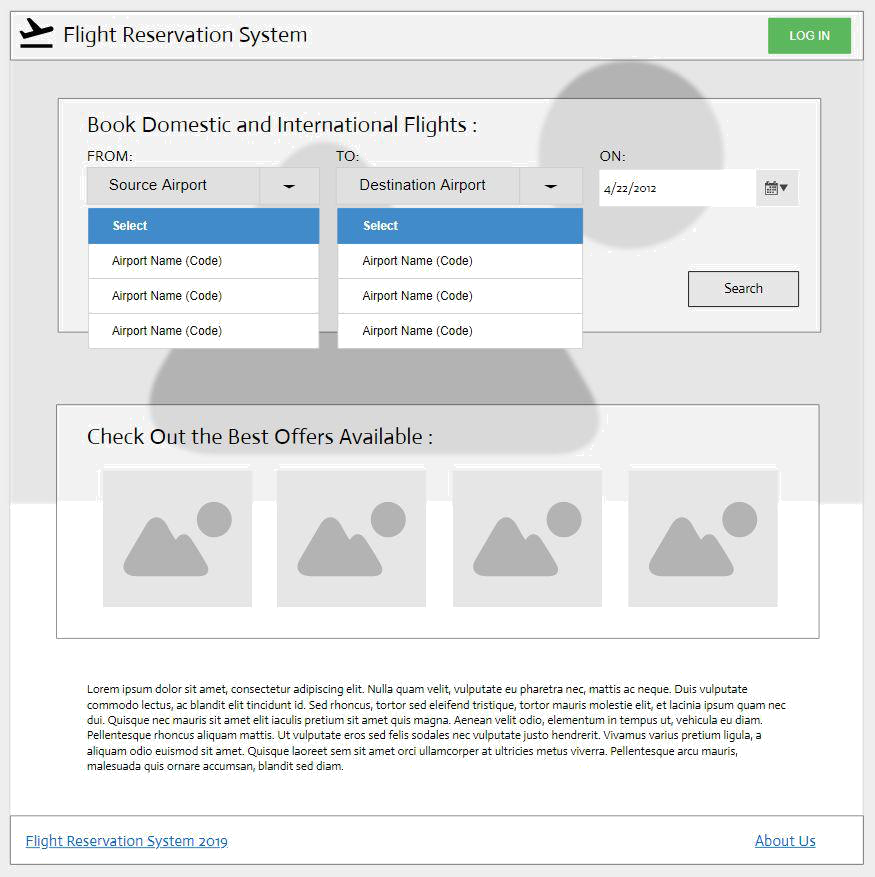


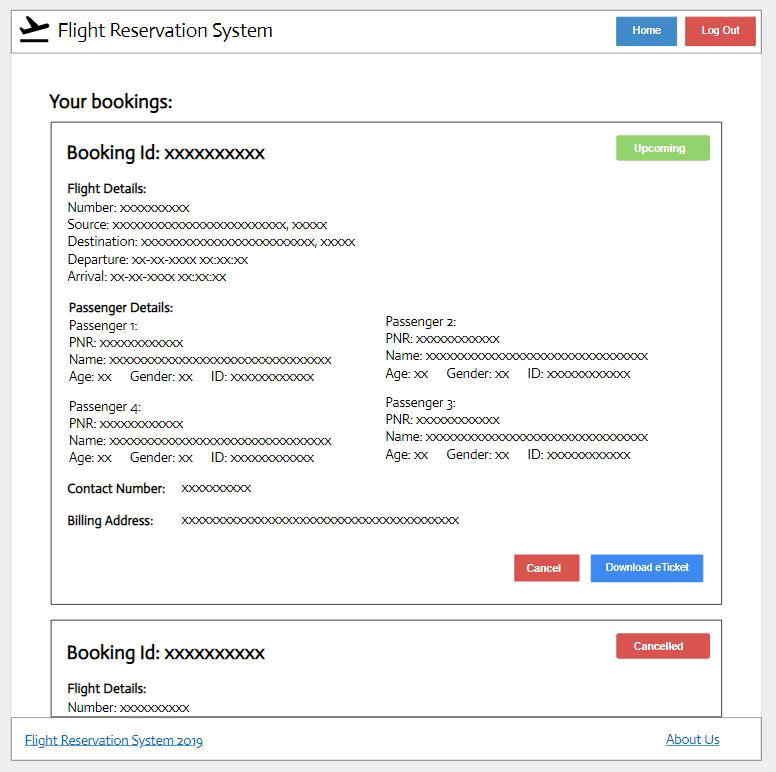
WIREFRAMES:

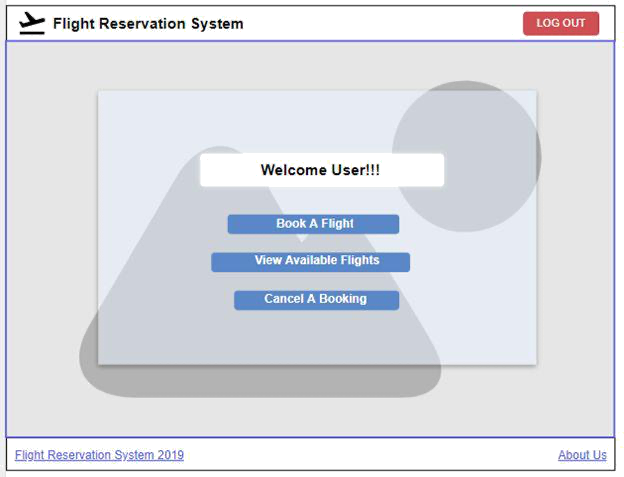


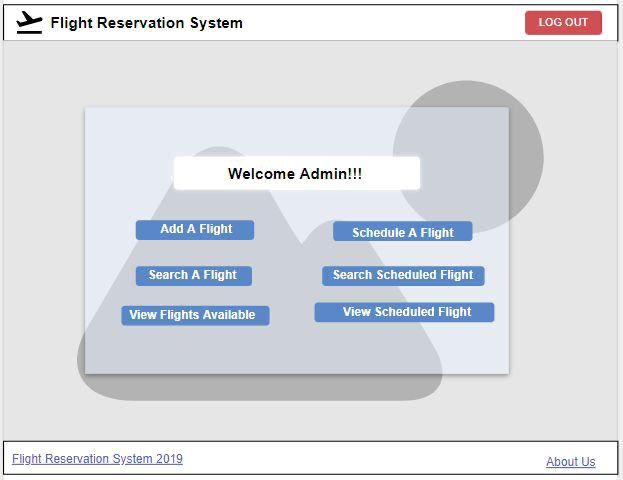


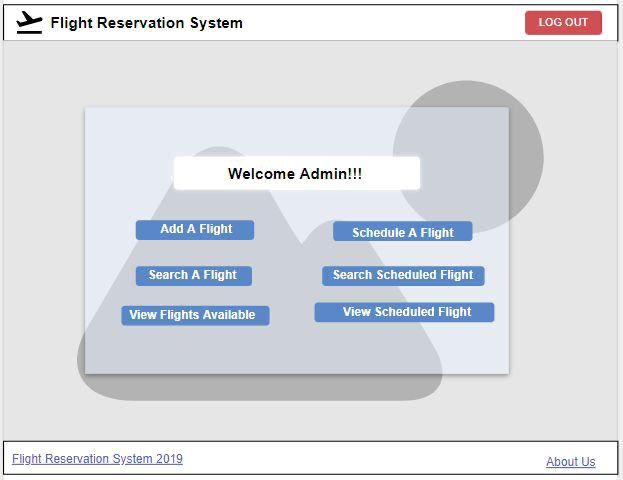




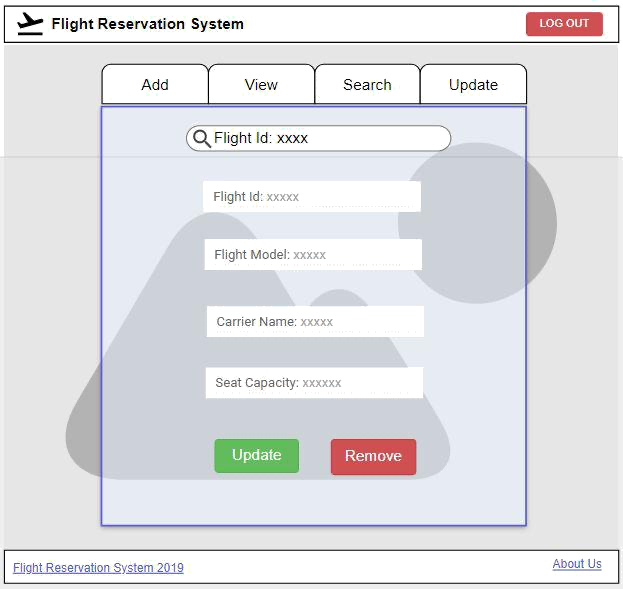




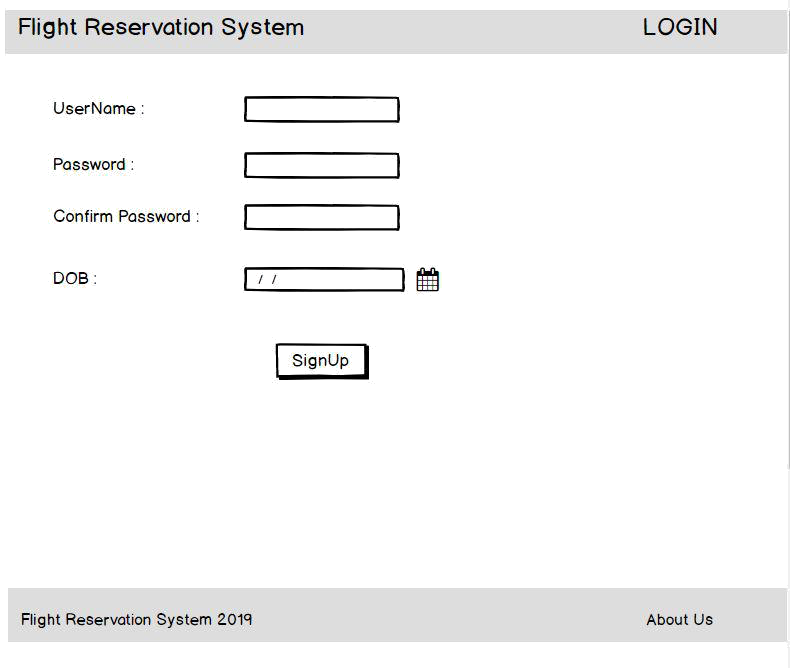


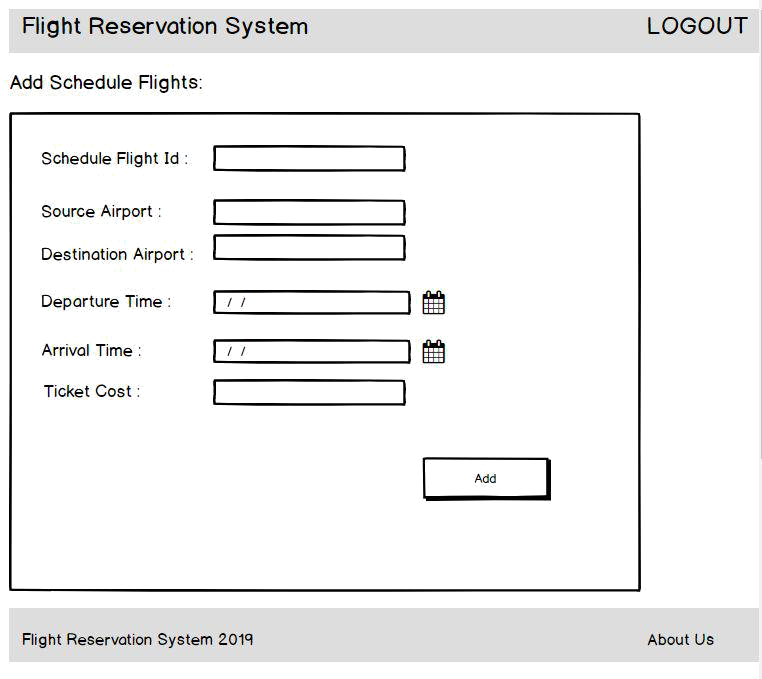


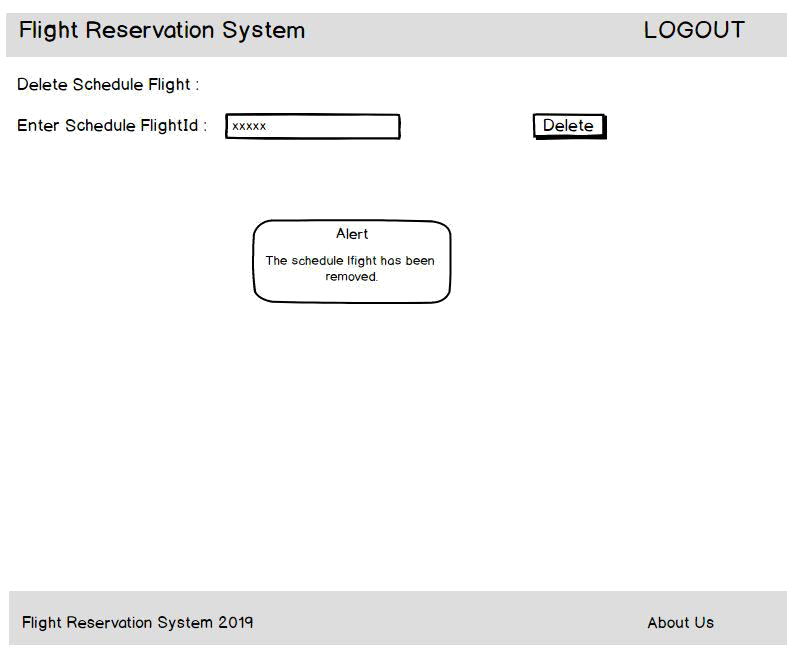


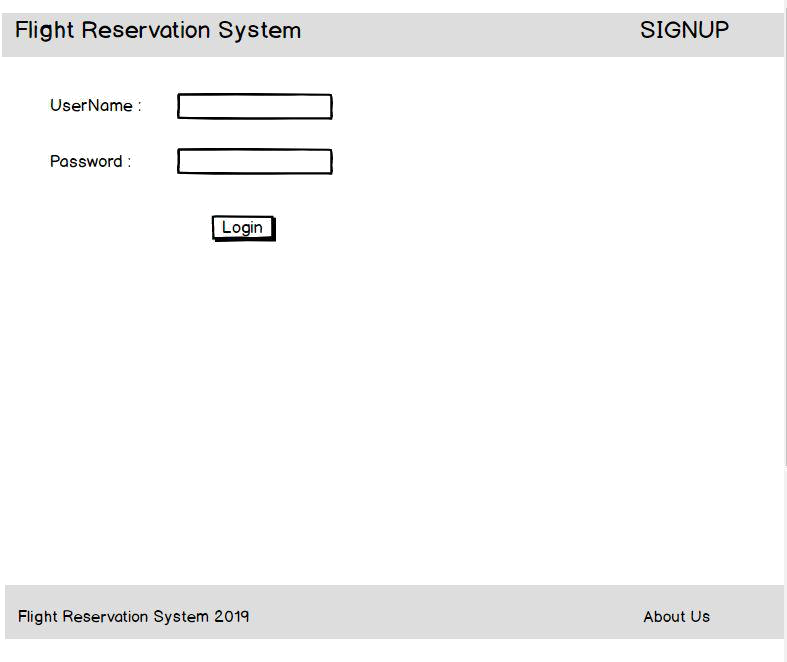


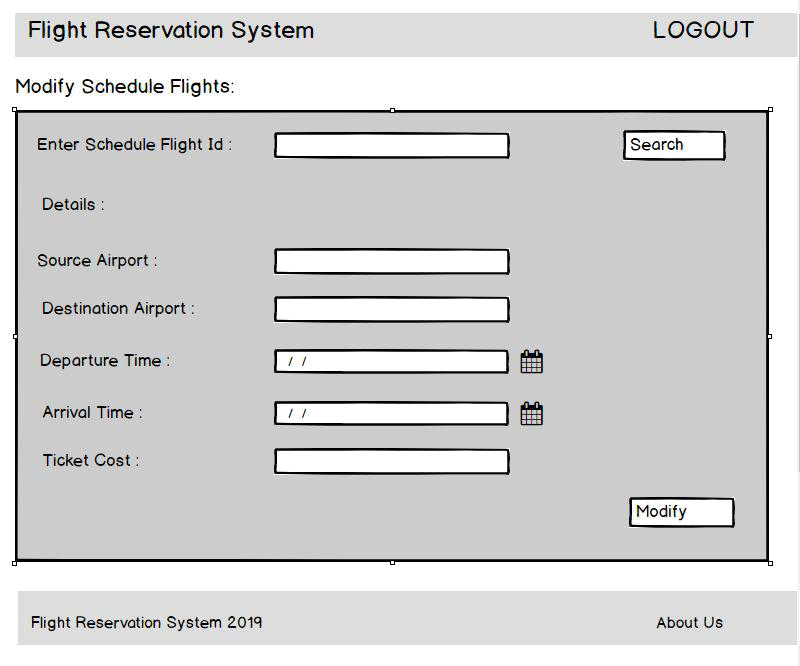


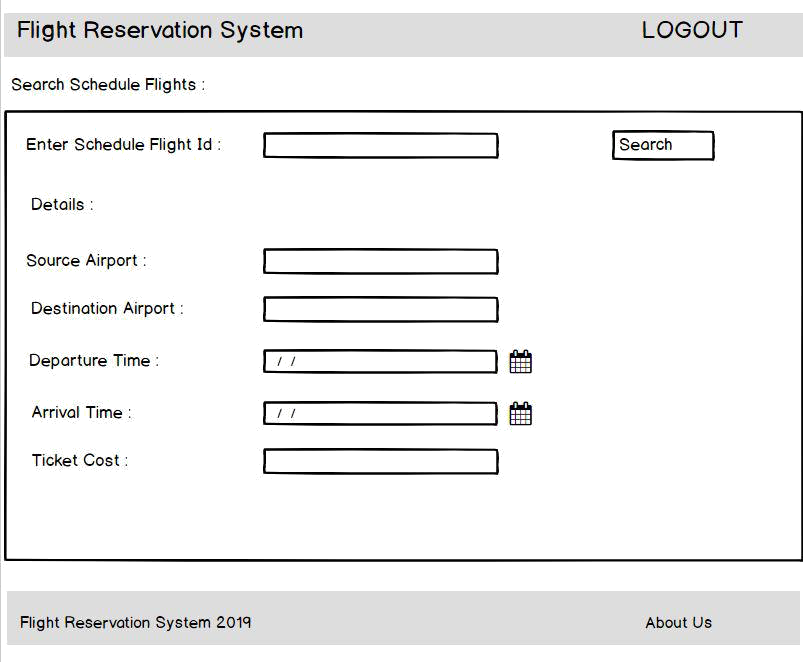












**Class and Method Description:**

**DTO Layer:**

1. **User:**​​This class stores the user type (admin or the customer) and all userinformation.

Attributes:​ userType: String. userId: BigInteger userName: String userPassword: Password userPhone:BigInteger userEmail: String

Methods:​ -

1. **Passenger**:​This class stores all the details of the travelling passenger.

Attributes:​

pnrNumber: BigInteger

passengerName: String

passengerAge: Integer

passengerUIN: BigInteger

Luggage: Double

Methods:​ -

1. **Booking**:​This class stores the details of a booking made by a particular userId. Everybooking stores a list of passengers travelling in it as well as the flight details.

Attributes:​

bookingId: BigInteger

userId: User

bookingDate: Date

passengerList: List<Passenger>

ticketCost: BigDouble

flight: Flight

noOfPassengers: Integer

Methods:​ -

1. **ScheduledFlight**:​This class stores a flight that is scheduled along with its scheduleand the vacancy.

Attributes:​

flight: Flight

availableSeats: Integer

schedule: Schedule

Methods:​ -

5. **Flight**:​ This class stores all the details of a flight.

Attributes:​

flightNumber: BigInteger

flightModel: String

carrierName: String

seatCapacity: Integer

Methods:​ -

6. **Schedule**:​ This class stores a flight schedule.

Attributes:​

sourceAirport: Airport

destinationAirport: Airport

arrivalTime: DateTime

departureTime: DateTime

Methods:​ -

1. **Airport**:​This class stores the details of an airport.Attributes:​

airportName: String

airportCode: String

airportLocation: String Methods:​ -

**Service Layer:**

1. **UserServiceImpl:**​Attributes:​ - Methods:

addUser(User):User :-Adds a new user.

viewUser(BigInteger):User :-

Shows the details of a user identifiable by the user id.

viewUser(): List<User> :-

Shows the details of all users.

updateUser(User):User :-

Updates the details of a user.

deleteUser(BigInteger):void

Removes a user as per the user id.

validateUser(User): void :-

Validates the attributes of a user.

1. **BookingServiceImpl:** Attributes:​ - Methods:

*addBooking(Booking):Booking* :*​*- Creates a new booking.

*modifyBooking(Booking): Booking* :*​*- Modifies a previous booking. Allinformation related to the booking except the booking id can be modified.

*viewBooking(BigInteger): List<Booking>* :*​*- Retrieves a booking madeby the user based on the booking id.

viewBooking(): List<Booking> :- Retrieves a list of all the bookings made.

*deleteBooking(BigInteger): void* :*​*-

Deletes a previous booking identifiable by the ‘bookingId’.

validateBooking(Booking): void :-

Validates the attributes of a booking.

validatePassenger(Passenger): void :-

Validates the attributes of a passenger.

1. **FlightServiceImpl:** Attributes:​ - Methods:

*addFlight(Flight): Flight :-*

Adds a new flight which can be scheduled.

*modifyFlight(Flight): Flight :-*

Modify the details of a flight.

viewFlight(BigInteger): Flight :-

Shows the details of a flight specified by the flight number.

*viewFlight(): List<Flight> :-*

View the details of all flights.

deleteFlight(BigInteger): void :-

Removes a flight.

validateFlight(Flight): void :-

Validates the attributes of a flight.

1. **ScheduleFlightServicesImpl:** Attributes:​ - Methods:

*scheduleFlight(ScheduledFlight): ScheduledFlight* :*​*-Schedules a flight alongwith its timings, locations and capacity

viewScheduledFlights(Airport, Airport, LocalDate): List<Scheduled

Flight> :-

Returns a list of flights between two airports on a specified date.

viewScheduledFlights(BigInteger):Flight :-

Returns a list of a scheduled flight identifiable by flight number.

*viewScheduledFlight(): List<ScheduledFlight>* :*​*-Shows all the details and status of all flights.

*modifyScheduledFlight(Flight,Schedule, Integer): ScheduledFlight* :*​*-Modifies the details of a scheduled flight.

*deleteScheduledFlight(BigInteger): void* :*​*-

Removes a flight from the available flights.

validateScheduledFlight(ScheduledFlight): void :-Validates the attributes of a scheduled Flight.

1. **AirportServiceImpl:**

Attributes:​ -

Methods :​

viewAirport(): List<Airport> :-

Returns the list of all airports.

viewAirport(String): Airport :-

Returns the details of an airport identifiable by the airport code.

**DAO Layer:**

1. **UserDaoImpl:**

Attributes:

userList: List<User>

Methods:

addUser(User):User :-

Adds a new user.

viewUser(BigInteger):User :-

Shows the details of a user identifiable by the user id.

viewUser(): List<User> :-

Shows the details of all users.

updateUser(User):User :-

Updates the details of a user.

deleteUser(BigInteger):void

Removes a user as per the user id.

1. **BookingDaoImpl:**

Attributes:

bookingList: List<Booking>

Methods:

*addBooking(Booking):Booking* :*​*- Creates a new booking.

*modifyBooking(Booking): Booking* :*​*- Modifies a previous booking. Allinformation related to the booking except the booking id can be modified.

*viewBooking(BigInteger): List<Booking>* :*​*- Retrieves a booking madeby the user based on the booking id.

viewBooking(): List<Booking> :- Retrieves a list of all the bookings made.

*deleteBooking(BigInteger): void* :*​*-

Deletes a previous booking identifiable by the ‘bookingId’.

1. **FlightDaoImpl:**

Attributes:

flightList: List<Flight>

Methods:

*addFlight(Flight): Flight :-*

Adds a new flight which can be scheduled.

*modifyFlight(Flight): Flight :-*

Modify the details of a flight.

viewFlight(BigInteger): Flight :-

Shows the details of a flight specified by the flight number.

*viewFlight(): List<Flight> :-*

View the details of all flights.

deleteFlight(BigInteger): void :-

Removes a flight.

1. **ScheduledFlightDaoImpl:**

Attributes:

scheduledFlightList: List<ScheduledFlight>

Methods:

*scheduleFlight(ScheduledFlight): ScheduledFlight* :*​*-Schedules a flight alongwith its timings, locations and capacity

viewScheduledFlights(Airport, Airport, LocalDate): List<Scheduled

Flight> :-

Returns a list of flights between two airports on a specified date.

viewScheduledFlights(BigInteger):Flight :-

Returns a list of a scheduled flight identifiable by flight number.

*viewScheduledFlight(): List<ScheduledFlight>* :*​*-Shows all the details and status of all flights.

*modifyScheduledFlight(Flight,Schedule,int): ScheduledFlight* :*​*-Modifies the details of a scheduled flight.

*deleteScheduledFlight(BigInteger): void* :*​*-

Removes a flight from the available flights.

1. **AirportDaoImpl:**

Attributes:

airportList: List<Airport>

Methods:

viewAirport(): List<Airport> :-

Returns the list of all airports.

viewAirport(String): Airport :-

Returns the details of an airport identifiable by the airport code.

**Validations:**

1. The ‘userPhone’ should have an exact 10 digit number and the number should not start with zero.
2. Date and Time should be valid i.e date and time that has already elapsed shouldn’t be entered
3. ‘noOfPassenger’ should always be less than equal to that of available seats.
4. The local part of the email should contain alphanumeric characters only. No special characters are to be present as the first character of the id.
5. The chosen airport’s name should be present inside the Airport database.
6. The Unique Identification Number should be of 12 digits.

**Assumptions:**

However, we have made a few assumptions with respect to the application, which are:

1. Administrator and customer are both Users. They are differentiated by a variable ‘userType’ in the User class.
2. Every passenger needs to enter a Unique Identification Number while booking is being made. For simplicity, we assume it to be a 12-digit Aadhaar Number.
3. All flights are direct flights.
4. No flight gets cancelled.
5. Number of airports is fixed and stored in database.
6. All the flights are considered to be domestic.