Ticket Reservation System

1.0 Introduction

1.1 purpose

The purpose of this document is to provide a comprehensive overview of the "Ticket Reservation and Services System." This document outlines the system's objectives, features, key stakeholders,
functionalities, operational constraints, and the interaction model between stakeholders and the
system. It also includes graphical notations to enhance understanding of the system's
components and workflow. The document is intended for both end-users, such as travelers making
bookings, and developers responsible for system implementation.

1.2 Scope of the project

The Ticket Reservation System is a comprehensive web-based application that enables event organizers to manage and sell tickets for their events, while providing customers with a user-friendly platform to search, reserve, and purchase tickets. The system aims to streamline the ticket reservation process, enhance the overall user experience, and provide valuable insights and analytics for event organizers.

This Software Design Document outlines the detailed scope of the Ticket Reservation System, covering the key features and functionalities that will be implemented.

Functional requirements

1. System must provide searching and selecting available flights based on various criteria (such as date, time, origin, destination, and class [Economy/Business/First Class], one way or round trip).
2. the system should allow users to select their seats during the booking process, considering factors such as availability, preferences, and any additional charges for specific seats.
3. The system should support secure payment processing for booking flights, including the acceptance of various payment methods and the generation of electronic tickets (QR code ticket) and a bill
4. System shall provide Users be able to view and manage their booked flights, make changes to their itineraries, and receive notifications about any updates or changes to their flights.
5. System shall support the application of loyalty program points, discounts, and promotional codes during the booking process(can use it to take discount25%).
6. System must facilitate the process of requesting refunds and cancellations, including managing refund policies.
7. The system should send email notifications to users for booking confirmations, itinerary changes, and other relevant updates.
8. Each Passenger has a limited sized of luggage by default that he can bring with him to the plane and the passenger should be able to pay an additional fee to exceed the size limit.(non)
9. The system should provide a feature for users to search and reserve available accommodations, including hotels and flats.
10. Each hotel or flat owner must have an account and should have a rating system for users to provide feedback and comments based on their experience with the owner.
11. The system should provide a feature for users to modify or cancel their accommodation bookings.
12. The system must support integration with travel services to provide additional travel information, such as weather forecasts, local events, and tourist attractions at the destination.(non)
13. The system should provide a customer support feature for users to contact customer service representatives for assistance with their bookings, refunds, or other inquiries.
14. The system should allow user to unlock extra services by paying more money.
15. The system should provide an admin view to manage/add (user accounts, flights, booking, refund) and have an overview of the system.
16. The system should provide a feature for users to purchase travel insurance during the booking process, offering coverage for medical emergencies, and lost luggage.(non)

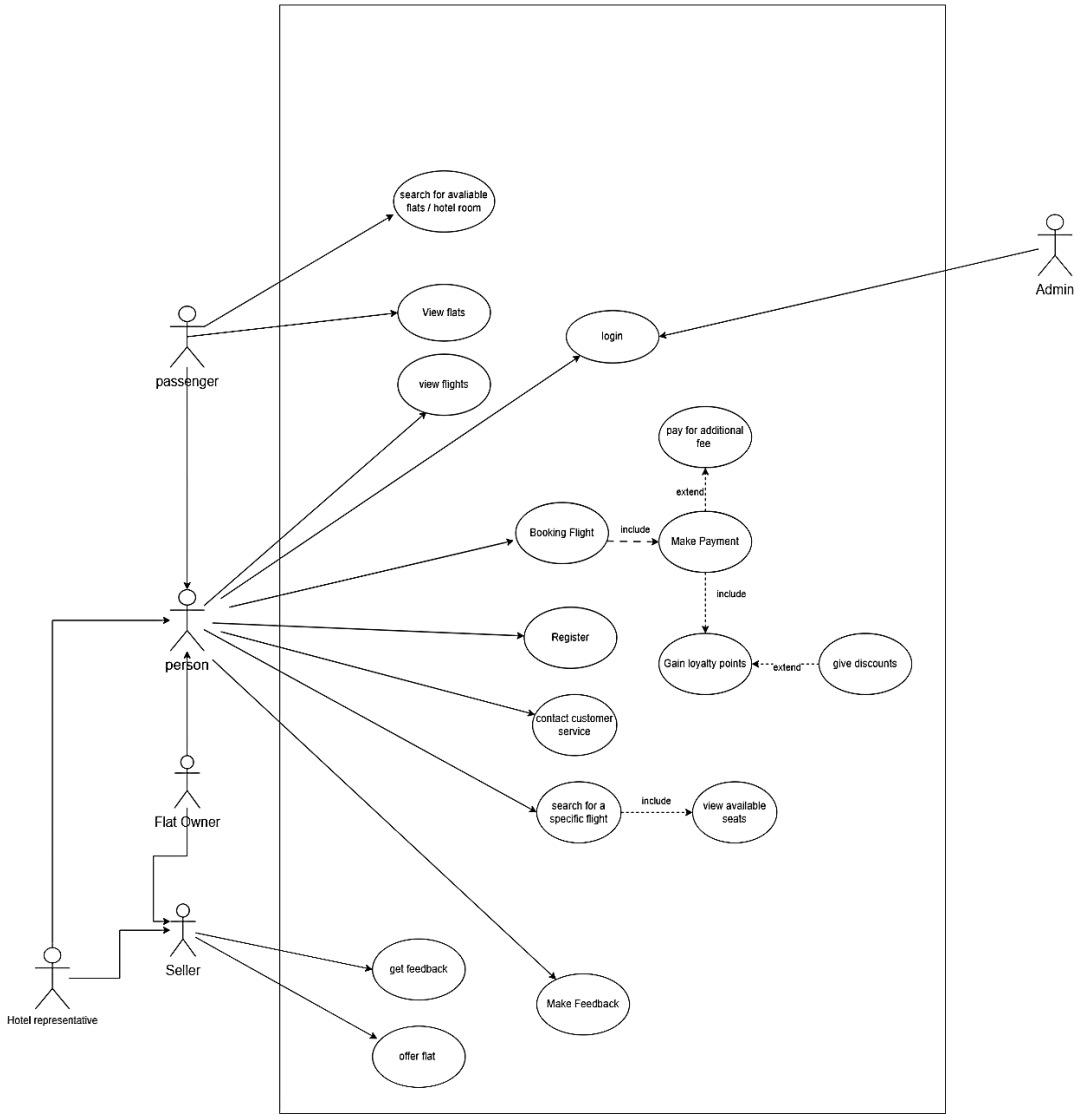
17. The system should provide a feature for users to search and reserve available airport lounges within a specific country, displaying information about their locations, amenities, and availability.
18. User should be allowed to select a preferred airline.
19. Nice to have: System shall include a customer support interface (chat bot) for users to seek assistance and submit inquiries.
20. System shall provide users with the ability to provide feedback and rate their booking experience.
21. System shall generate reports and provide analytics on booking trends, popular routes(most visited country), and revenue generation.

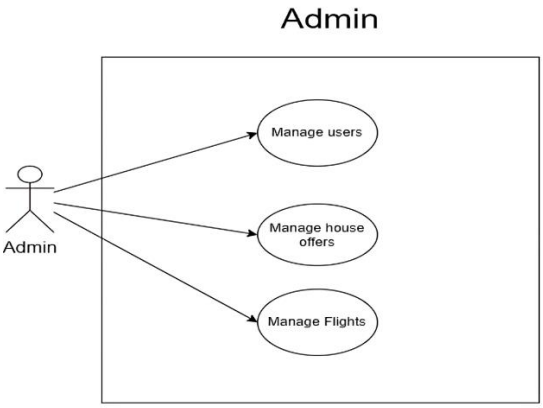
Non-Functional requirements

1. The system should show the countdown time of the plane that he has been booked before it takes off from the airport
2. System should have a section for customers feedback, Rating and reviews
3. System must provide a user-friendly interface
4. System must provide high security methods for the users sensitive data from unauthorized access.
5. The system must be highly scalable , which has the ability to handle increasing workloads and adapt to changing needs and demands.
6. The system must be available to users 24/7 , with a possible downtown of 30 minutes each month for the scheduled maintenance . Any unplanned downtown must not take more than 10 minutes and should happen less than twice in month.
7. The system should adhere to the accessibility standards , allowing the users with disabilities to access and use the system with assistive technologies .
8. The system should provide informative error messages and log errors to facilitate troubleshooting and support .
9. The system should accommodate large data storage capacity without impacting the performance

Diagrams

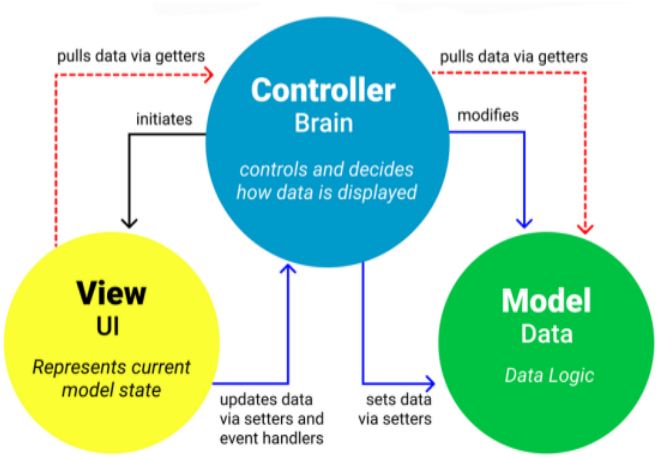
1.Use Case

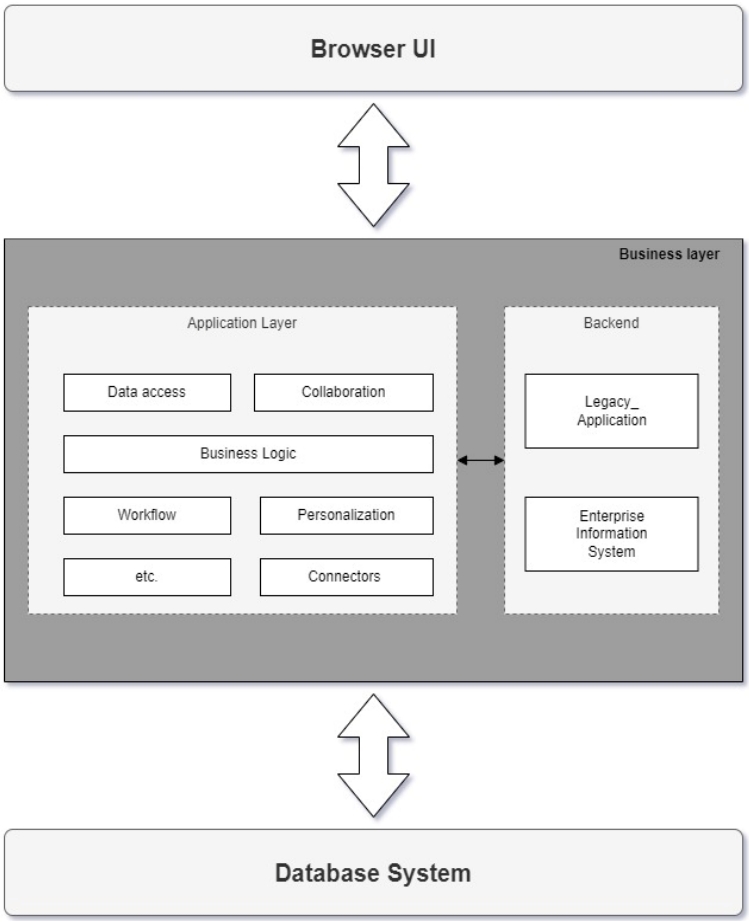




2.System Architecture Diagram:

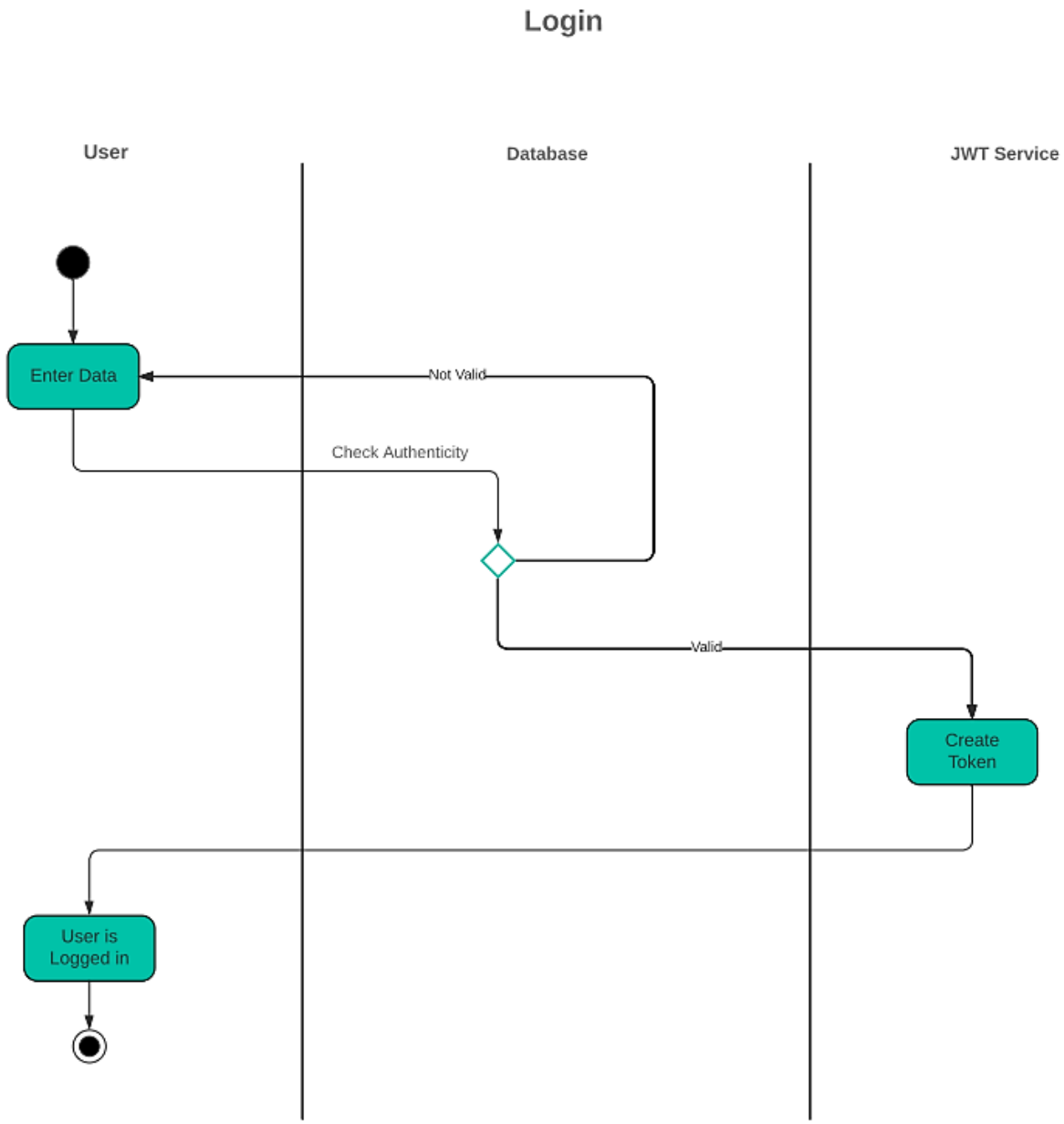
Name	Model-view-controller (MVC)
Context	Help keep the user interface separate from the rest of the model
Problem	Users need to see all updates that happened after making change.
Solution	Make 3 classes: 1- model class: contain all objects and DB. 2- view class: shows all updates, user can treat with it. 3-controller class: contain all if statements and methods which can change D.



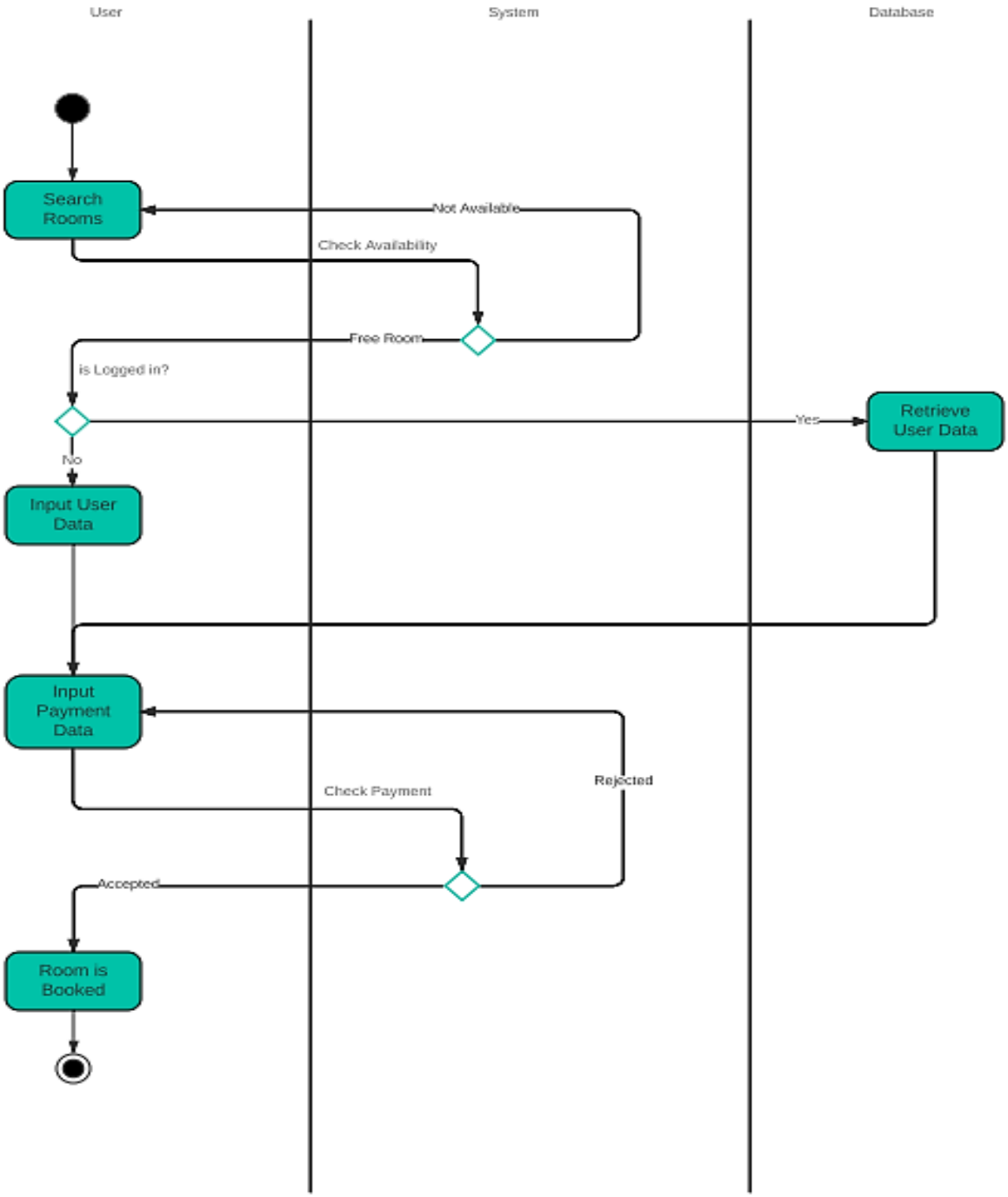


3.Activity Diagram:

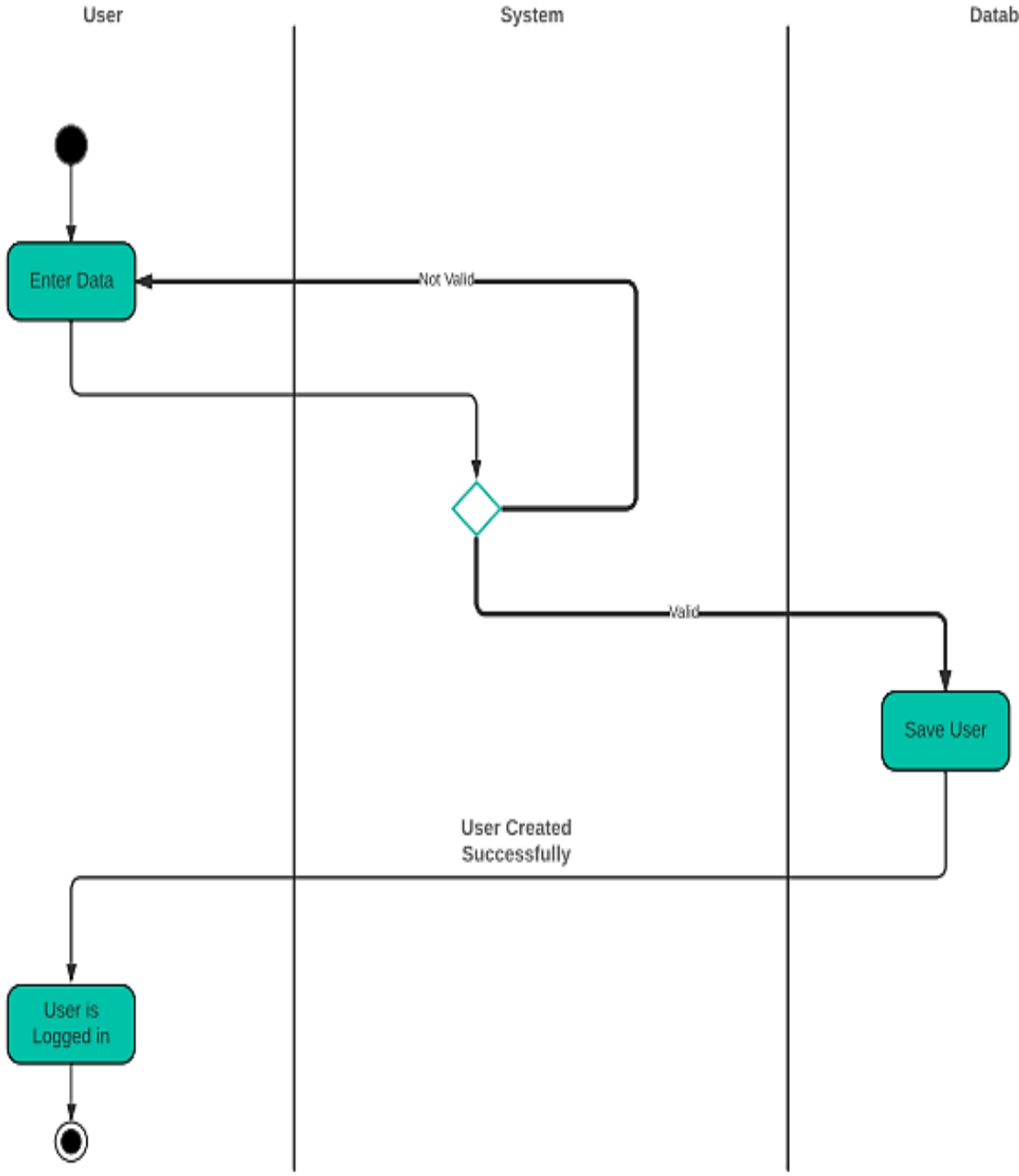
Login:



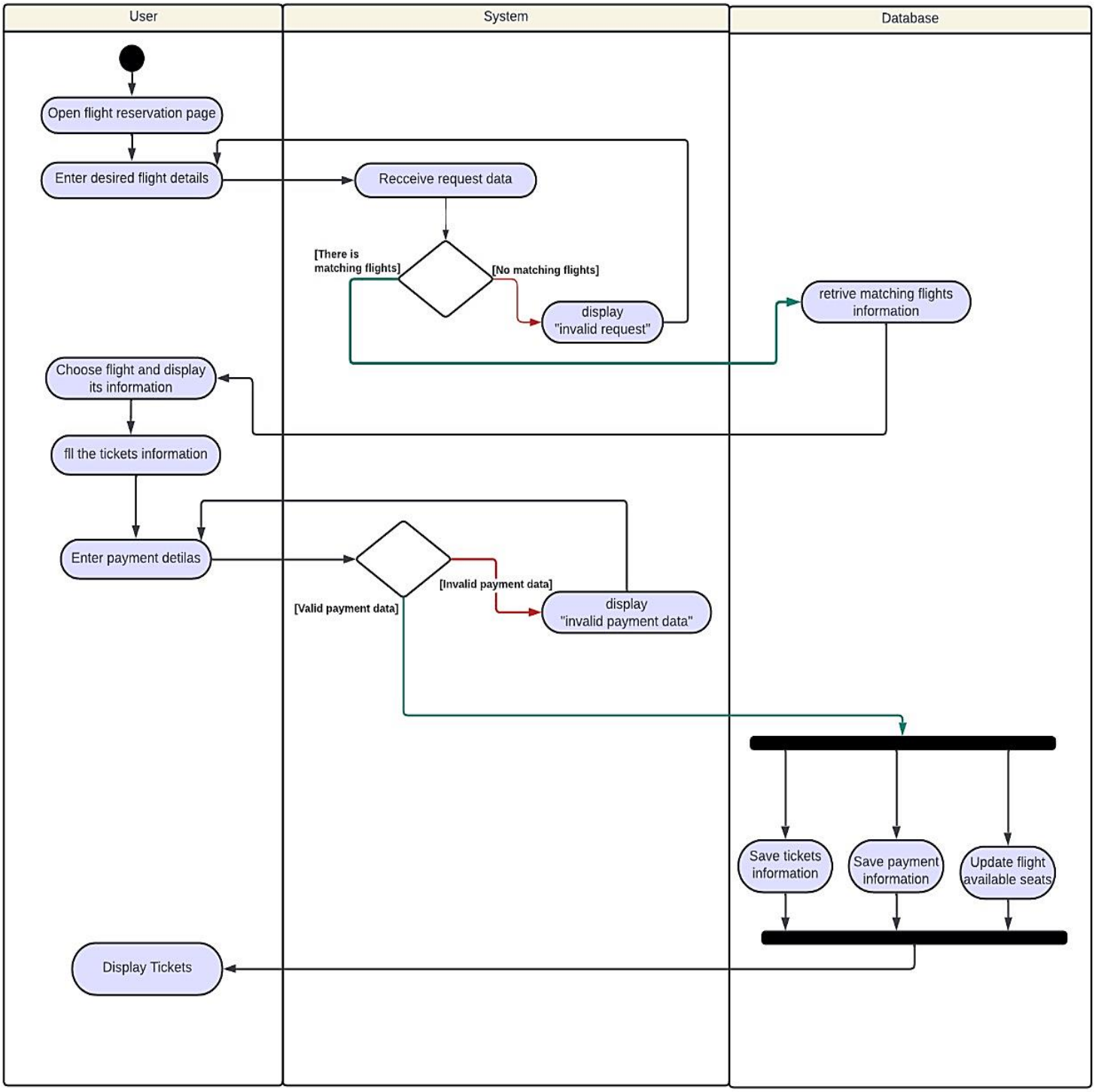
Search Accommodation:



Sign up:

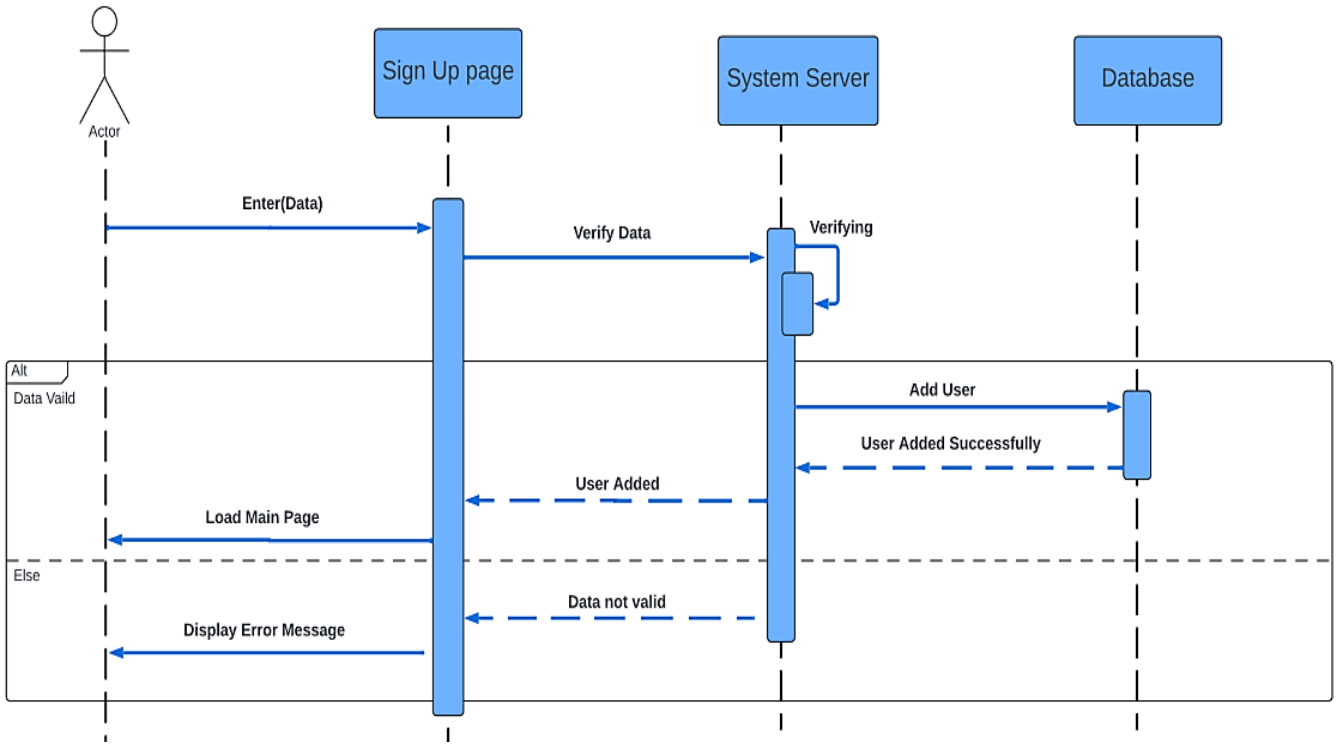

User
System
Database
Enter Data
Not Valid
Valid
Save User
User Created
Successfully
User is
Logged in

Flight, Entertainment Booking:

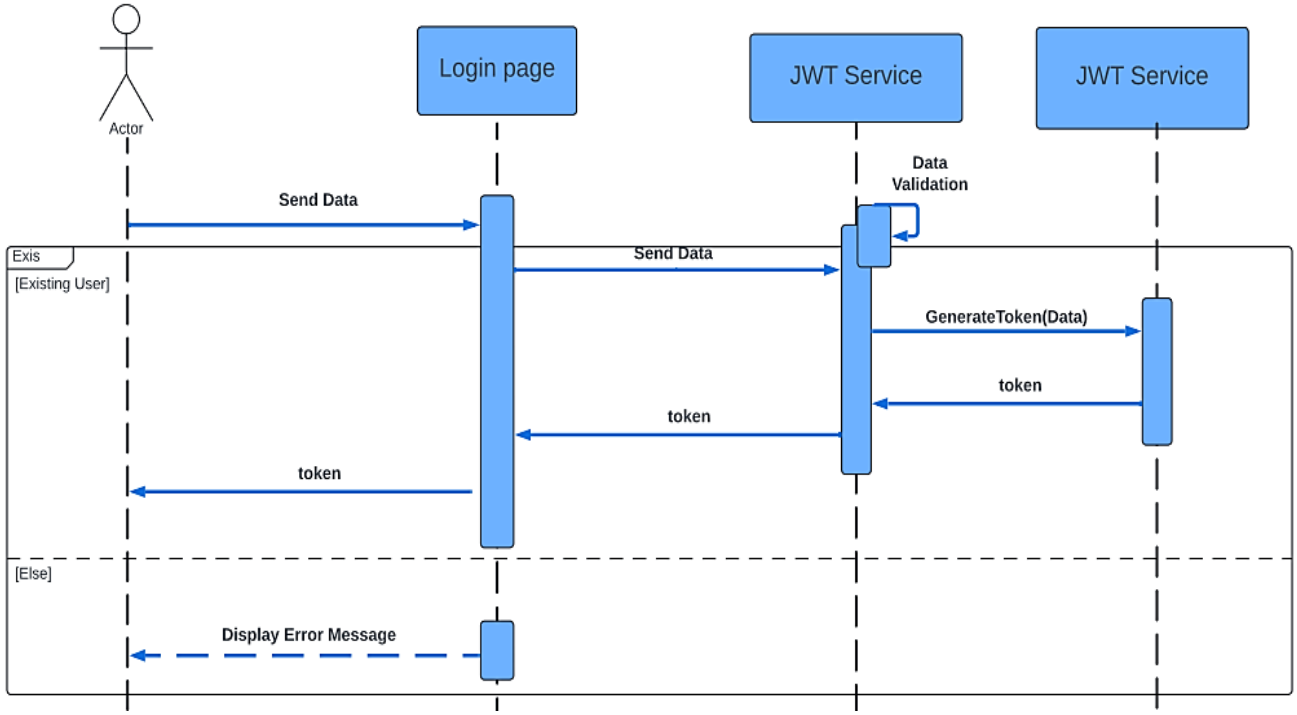


4.Sequence Diagram:

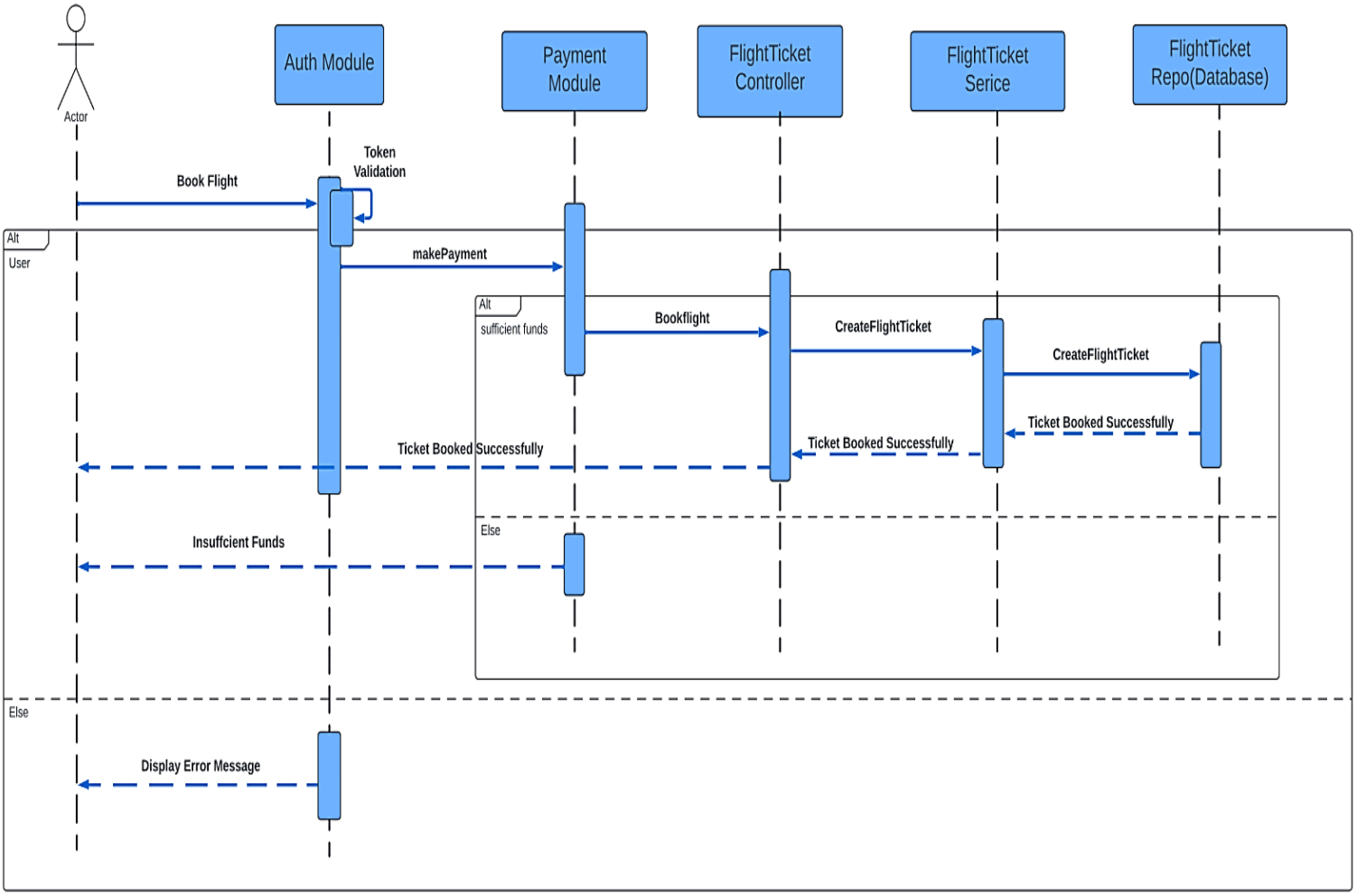
sign up:



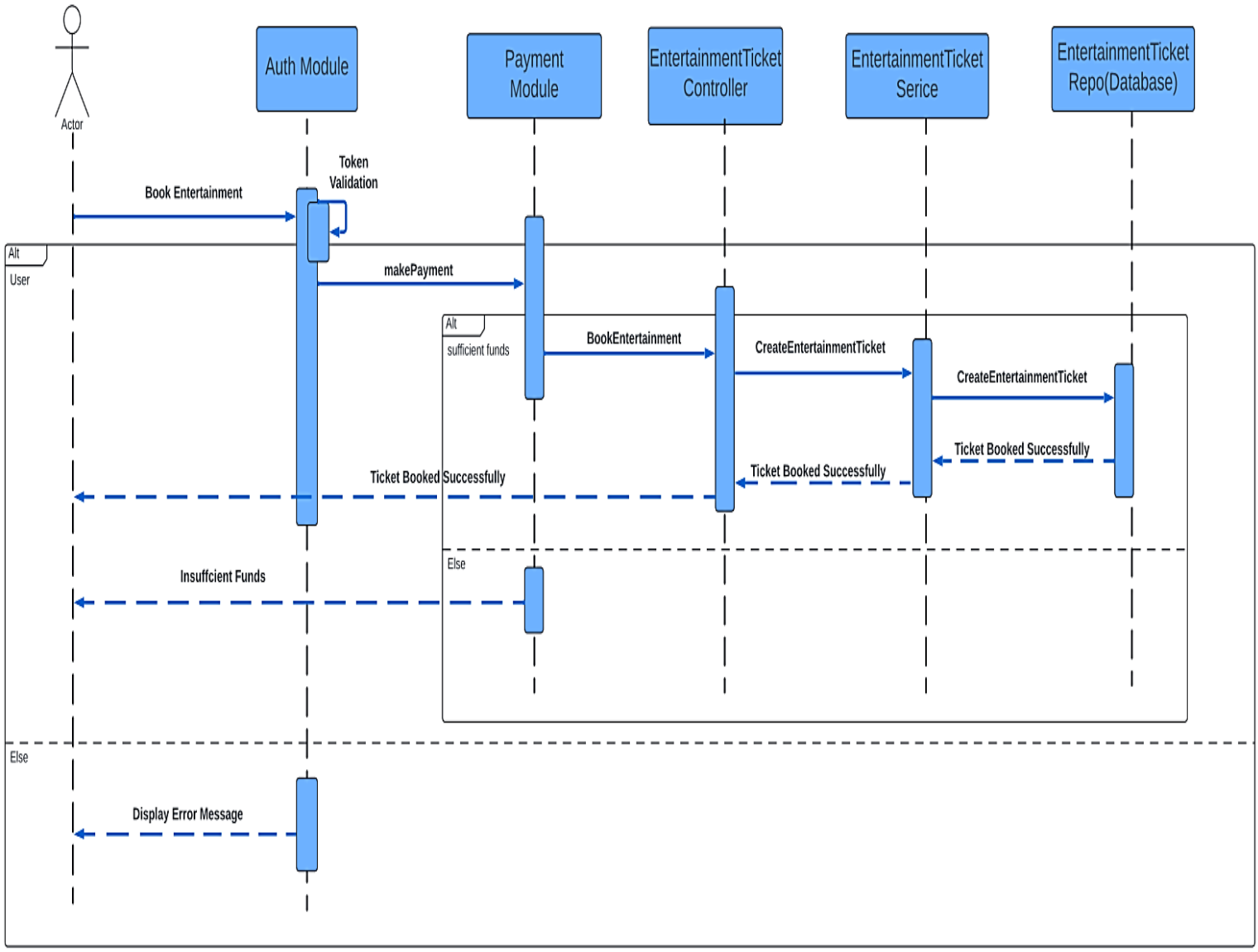
Loging:



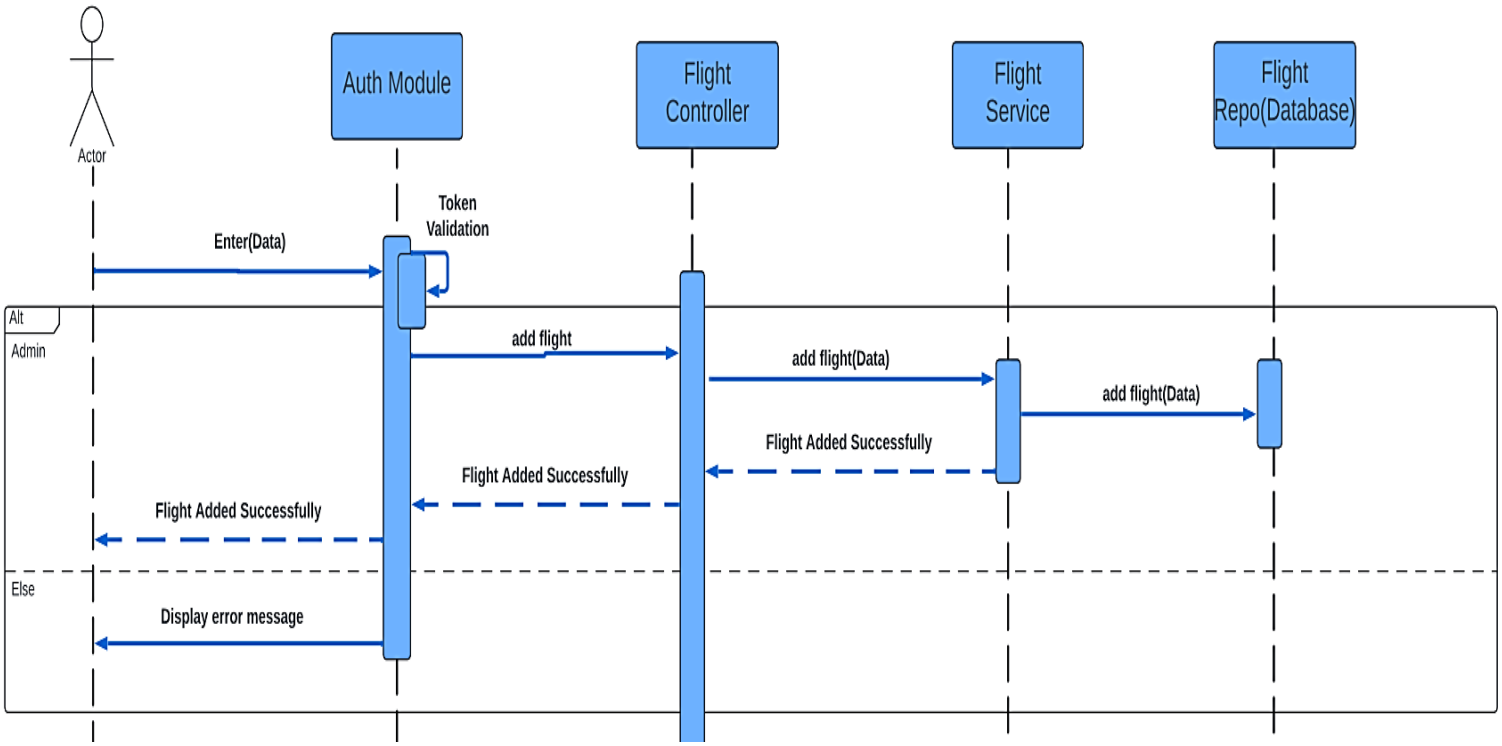
Flight ticket booking:



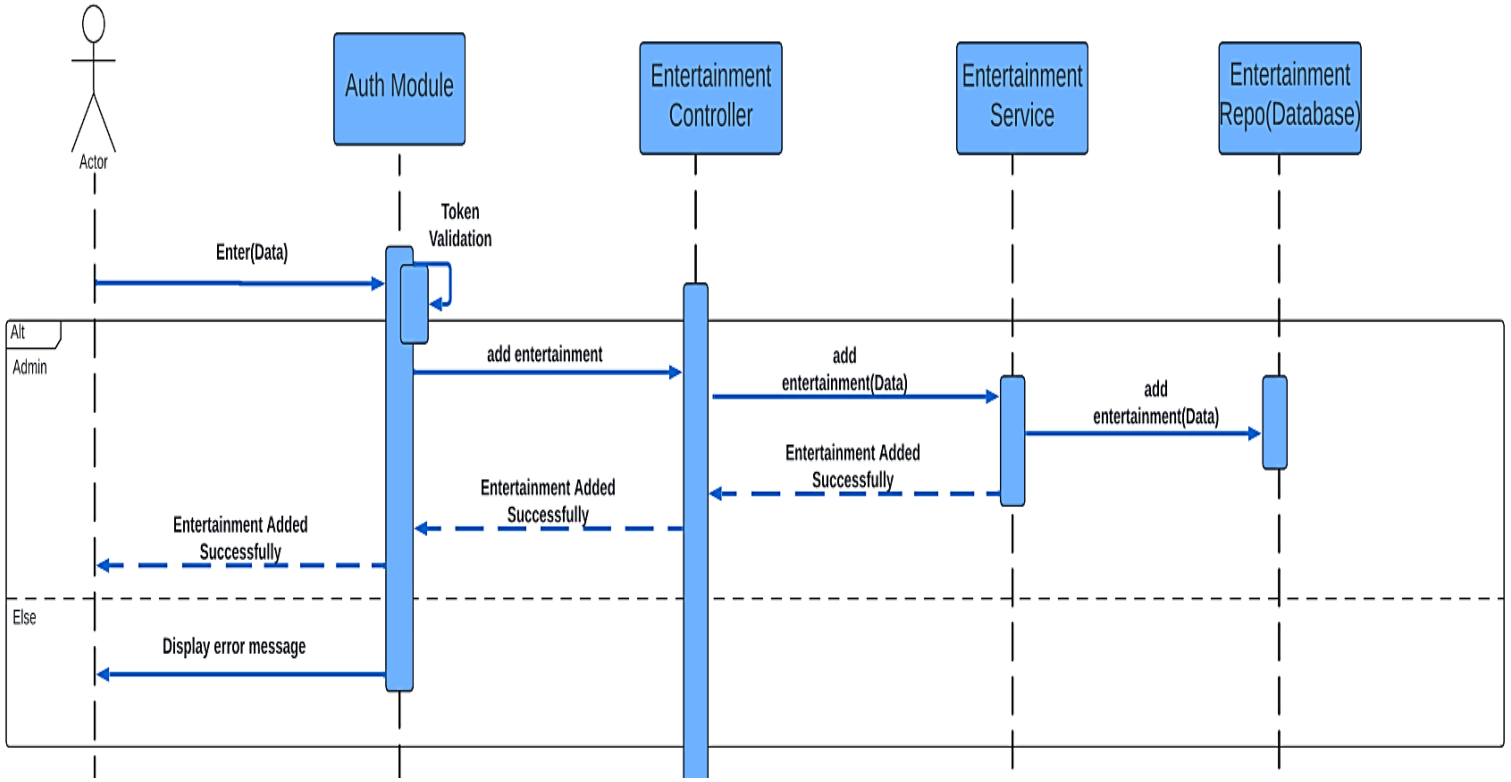
Entertainment ticket booking:



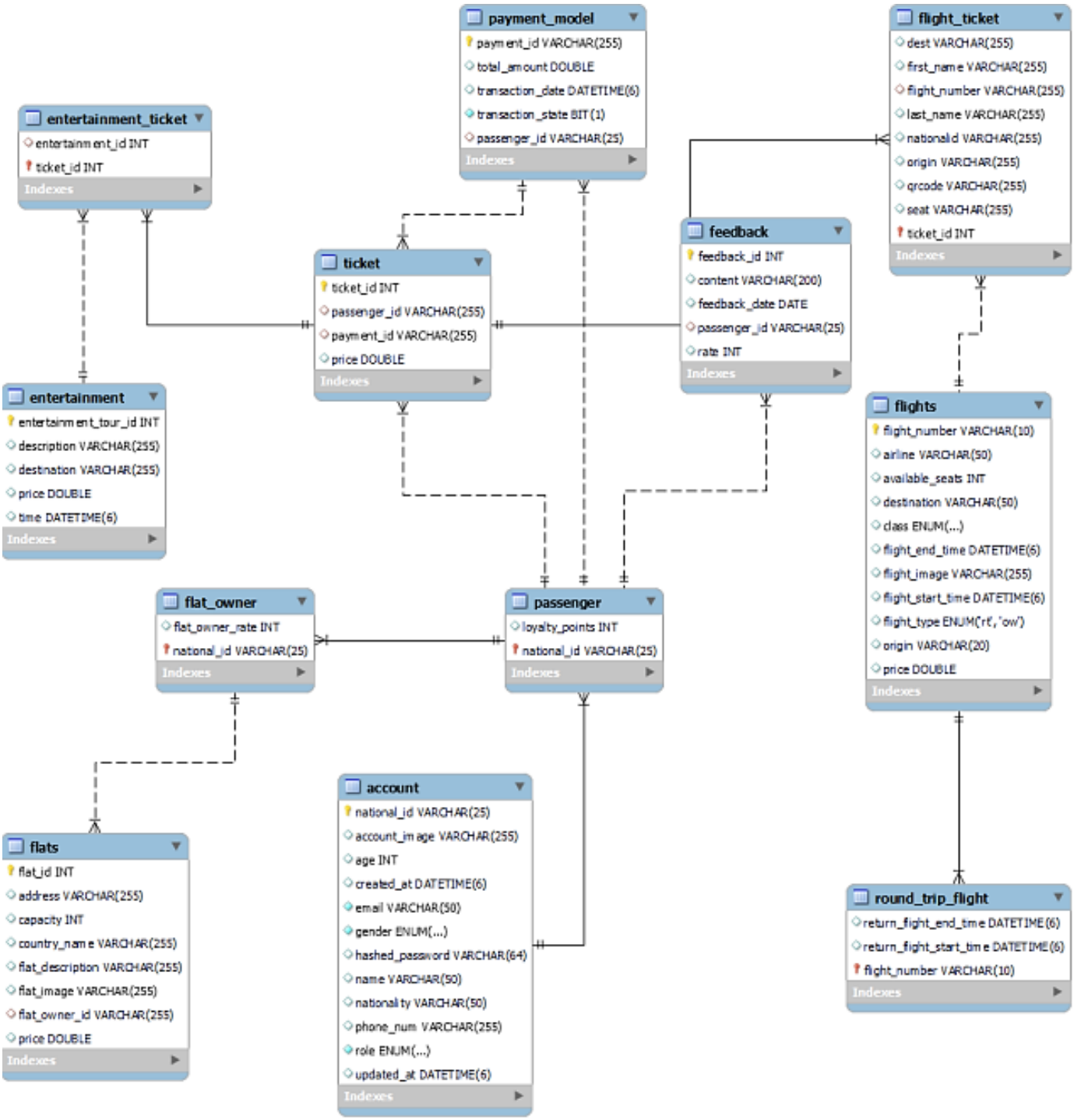
add flight:



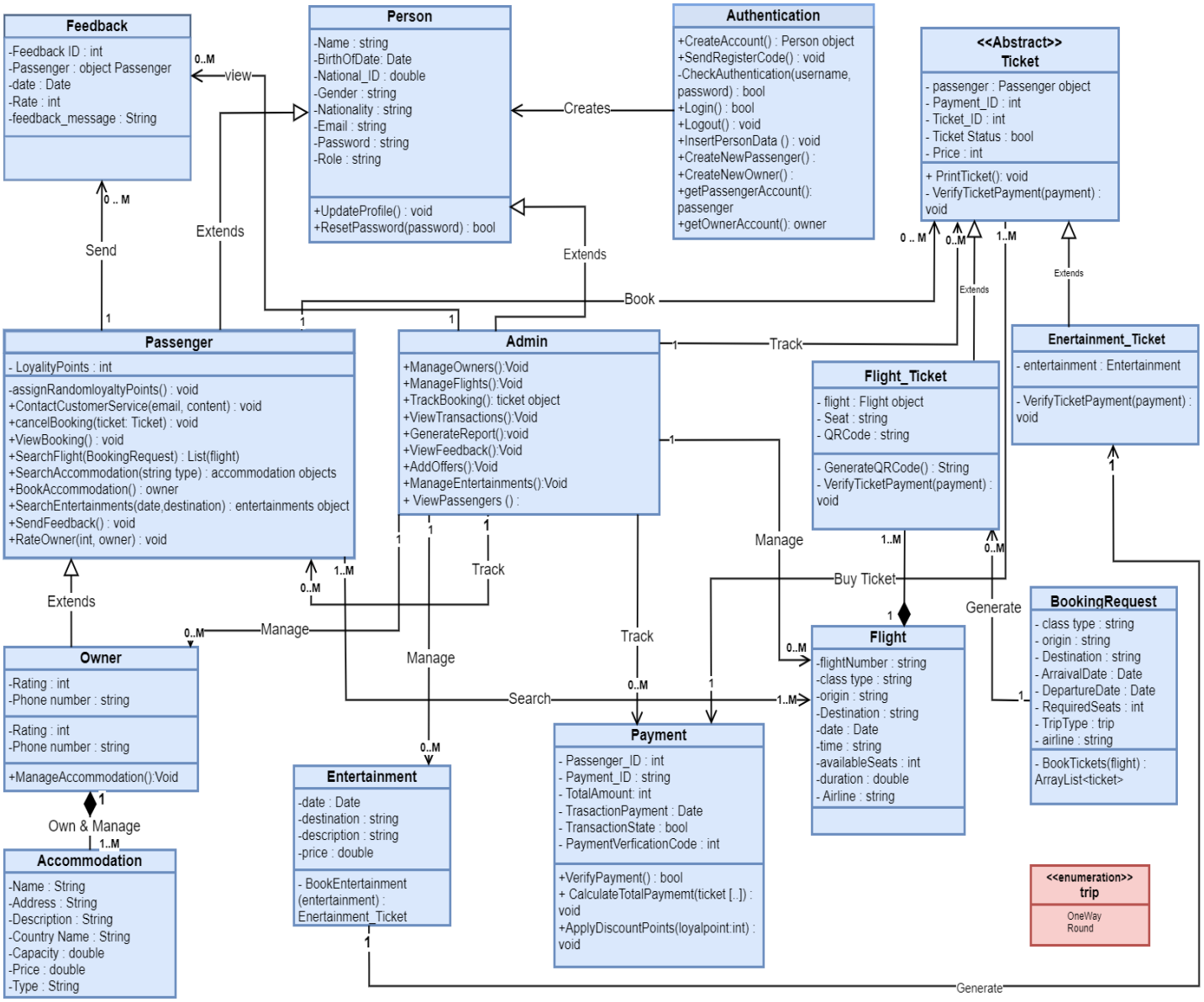
add entertainment:



5.ERD Diagram:



6.Class Diagram:



7.Object Constraint Language (OCL):

1- The person age should be greater than or equal to 18 years of age

```
context Person
invariant: self.BirthOfDate  >= 18
```

2- The number of tickets is less than or equal to available seats

```
context Flight
invariant: Flight_Ticket -> size  <=  availableSeat/
```

3- The passenger can't send feedback if he does not have an account

```
context Passenger
inv HasAccount: Passenger->notEmpty()          implies
self.SendFeedback() = true
```

4- If the passenger exceed 1000 points , he can use each 1000 points to get 10% discount

```
context Passenger
Inv PointDiscount: self.LoyalityPoints >= 1000
Implies
Self.Ticket.Payment. ApplyDiscountPoints()
```

5- The passenger can book maximum 5 of tickets

```
context Passenger
inv MaximumTickets: Ticket -> size() <= 5
```

6- The birth of date of person must be on or before the current date

```
 context Person
inv BirthDateBeforeCurrentDate:
self.BirthOfDate.isBefore(Today)
**Today:Date**
```

7- The departure date must be before the arrival date or the same(for round trip)

```
context BookingRequest
Inv DepartureDateBeforeArrivalDate:
self. departureDate >= self. ArraivalDate
```

8- The price of ticket must be greater than or equal to zero

```
context ticket
Inv TicketPriceNonNegative: self.price > 0.0
```

9- The passenger associated with the feedback must have attended at least one flight(the flight date on or before the feedback date)

```
context Feedback
inv PassengerHasAttendedFlight:
self.passenger -> ticket.payment->exists(f |
 f. TrasactionPayment <= self.feedbackDate)
```

10- All feedbacks associated with the passenger must have the passenger as the owner

```
context feedback
Invariant self.passenger.Person-> exists( Role = 'Owner')
```

11- The source can't be same as the destination

```
context BookingRequest
inv SourceNotEqualToDestination:
collection->reject(element | self.  origin.oclIsEqual(self.Destination))
```

12- The passenger can choose one airline preferred

```
context Passenger
inv: self.Flight.airline->size() = 1
```

13- When the person login should check if the person is already exists

```
context Person
inv PersonExists:
self. Person -> exists(p | p.username = self.username and    p.password = self.password)
```

14- The owner must have at least one accommodation

```
context Owner
inv OwnerHasAtLeastOneFlat:
self. Accommodation ->size() >= 1
```

15- The total payment must be equal to the ticket(s) price

```
context Payment
inv TotalPaymentEqualToTicketPrice:
self.totalPayment = self.tickets.price->sum()
```

16- The password must be at least 8 numbers with letters

```
context Person
inv PasswordRequirements:
self.password.size() >= 8
and self.password.matches('[a-zA-Z]+')
and self.password.matches('[0-9]+')
```

17- The admin can add offers for the flights that the departure date is close & have seats available on the flight

context Admin
inv CanAddOfferForFlightsWithAvailableSeats:
self.offersToAdd()->forAll
(self.today.addDays(3).isbefore(departureDate))

18- The passenger cancel booking takes 25% from the total price & return the seats and tickets available for sale again

context Passenger
inv CancelBookingWithFee:
self. cancelBooking(ticket) implies(self.Ticket.Payment.totalPayment = self.tickets->sum(t | t.price * 0.75),
self.Ticket.Ticket state = false)

⁂ Pre & Post Conditions:

context Passenger:: cancelBooking(ticket: Ticket)
Pre: cancelBooking: ticket.isDefined()
Post: cancelBooking: Ticket.canceled(ticket)

context Passenger:: assignRandomloyaltyPoints()
Pre: self. isDefined (Ticket)
Post: self. LoyalityPoints = self. LoyalityPoints@pre + RandomValue

Context Payment:: ApplyDiscountPoints(Loyality points:integer)
Pre: Loyalitypoints > 0
Post: self.TotalAmount = self.TotalAmount@Pre - Loyalitypoints

Context Person:: ResetPassword(password)
Pre: self.password@pre != self.password
Post: self.password = password

Context BookingRequest::BookTickets(f:Flight)
Pre: flights->isDefined()
Post: flights->include(f)

Context Passenger:: SearchFlight(booking:BookingRequest)
Pre: booking.isdfined()

Post: result -> exist(p | p = booking)

Context Authentication::
CheckAuthentication(username,password)
Pre: username.isdfined() & password.isdfined()
Post: result ->exist(username@pre = username & password@pre = password)

Context Ticket :: VerifyTicketPayment(payment)
Pre: self.Ticket.isdfined & self.TicketStatus = false
Post: self.TicketStatus = true