ENSF 480 Final Project Design Document

Flight Reservation Application

1. Introduction to the System

The Flight Reservation Application is a standalone desktop application written in Java. The application is a role-based system which has different views for the following three user types: Customer, Agent, and Admin. The system is built using object-oriented principles and common design patterns such as Singleton, Observer, Strategy, etc. The system puts high emphasis on modularity and reusability, as is outlined in the rest of this report.

Customers can independently search for flights based on criteria (origin, destination, date, etc.), select seats for a selected flight, enter their details, and complete bookings through the simulated payment process. Flight agents can assist customers with bookings, manage customer profiles, modify reservations, and view flight schedules. Admins have control over flights and aircraft, including the ability to add, update, or remove flights, and manage aircraft and route information.The system sends booking confirmations and monthly promotional notifications to registered customers via the promotions page on Customer view. The application uses a MySQL database and Java Swing for the GUI. The application is optimized for extensibility and has used proper design patterns where necessary. A summary of patterns used:

1. Singleton pattern for database initialization
2. Observer pattern for newsletter promotions to customers
3. Factory pattern for creating different users (admin, agent, customer)
4. Decorator and Factory for custom buttons on the GUI
5. Strategy for different logins and payments (extendable for future)
6. MVC for entire codebase - View = GUI, Control = controllers, DAO = model

1.1 Major Processes

This section describes four major processes in the Flight Reservation System, each represented by an activity diagram.

Login Process

Users login via their credentials, which are assigned a role of AGENT, ADMIN, and CUSTOMER via the role enumeration. Error messages are displayed for invalid usernames and passwords. Based on the role, the landing page is displayed upon successful login.

Browse Flights Process

Users enter search criteria (start and destination, as well as specific date) and a grid of all available flights is displayed. If the input format is incorrect, comprehensive error messages are displayed.

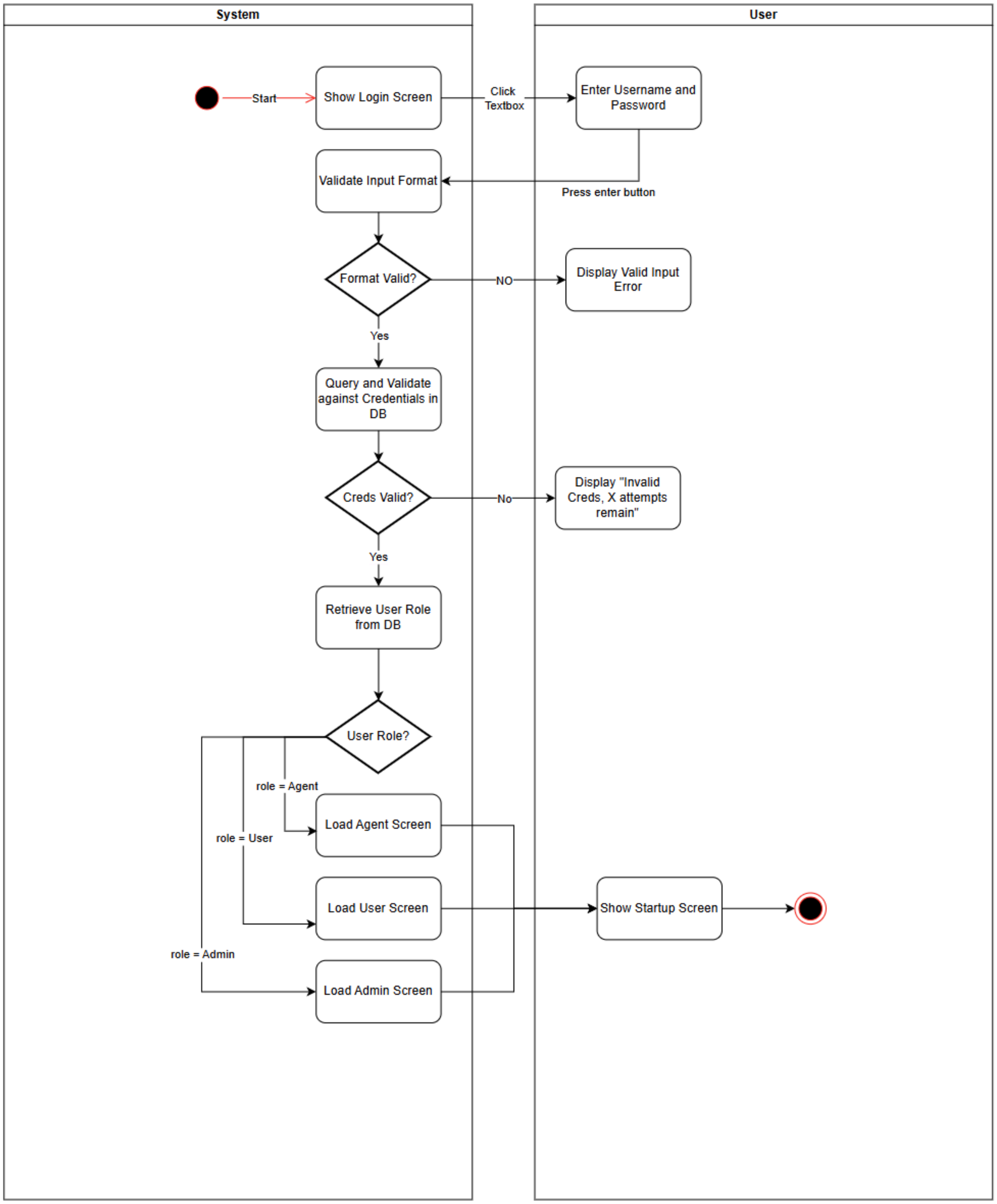
Make Payment Process

Users select a flight and view seats. After selecting a seat and entering passenger information, the system validates the input and displays any errors. The total fare is displayed, and the payment method is displayed. Users review the booking summary showing flight details, passengers, seats, and total cost. Upon confirmation, a preliminary booking record is created with status "Pending Payment," and the user proceeds to the payment process.

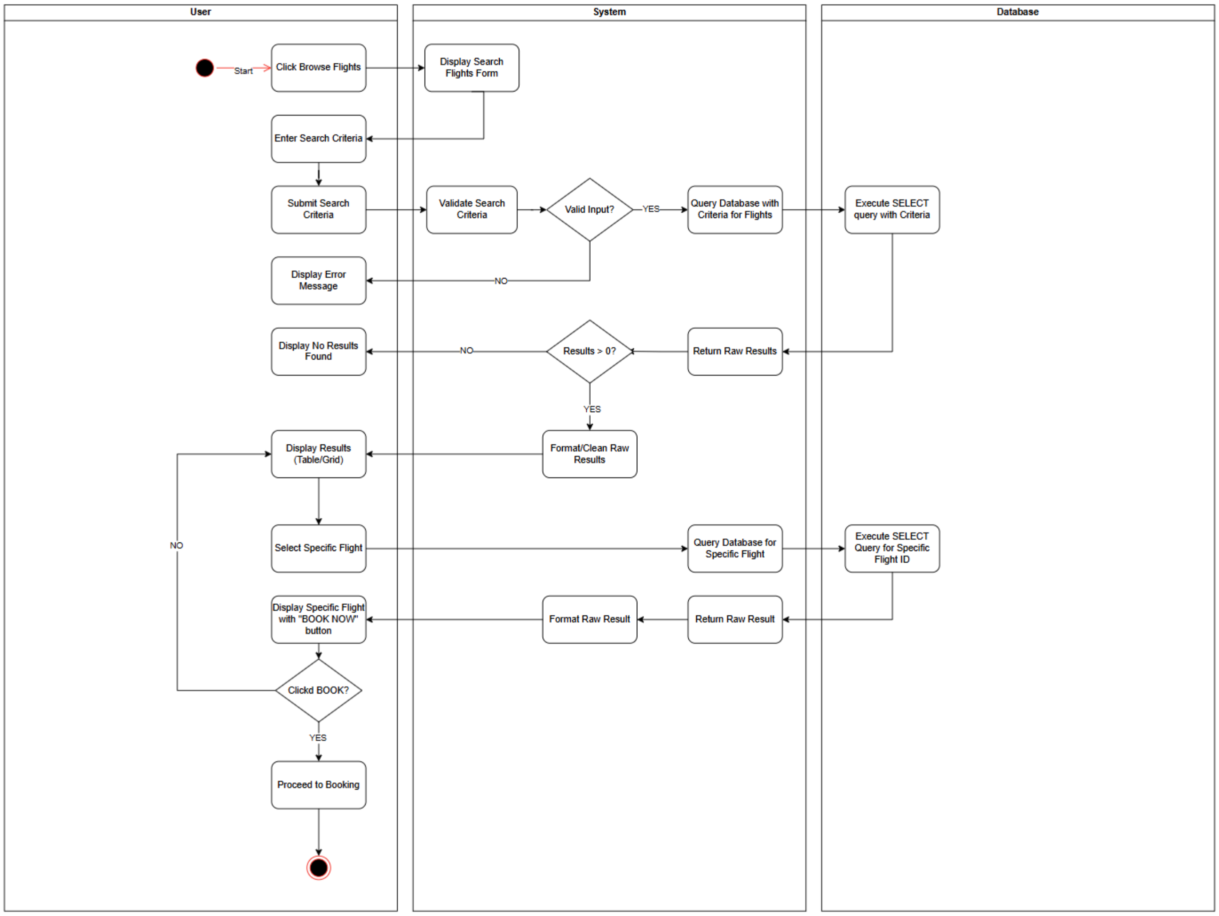
Make Payment Process

Users select a payment method (Credit Card, Debit Card, or PayPal) and enter payment details. The system validates the format of the information needed for that payment method, displaying specific error messages for invalid entries. The system confirms the payment (which will always be successful for our current implementation) and reserves the seat for the customer in the database.

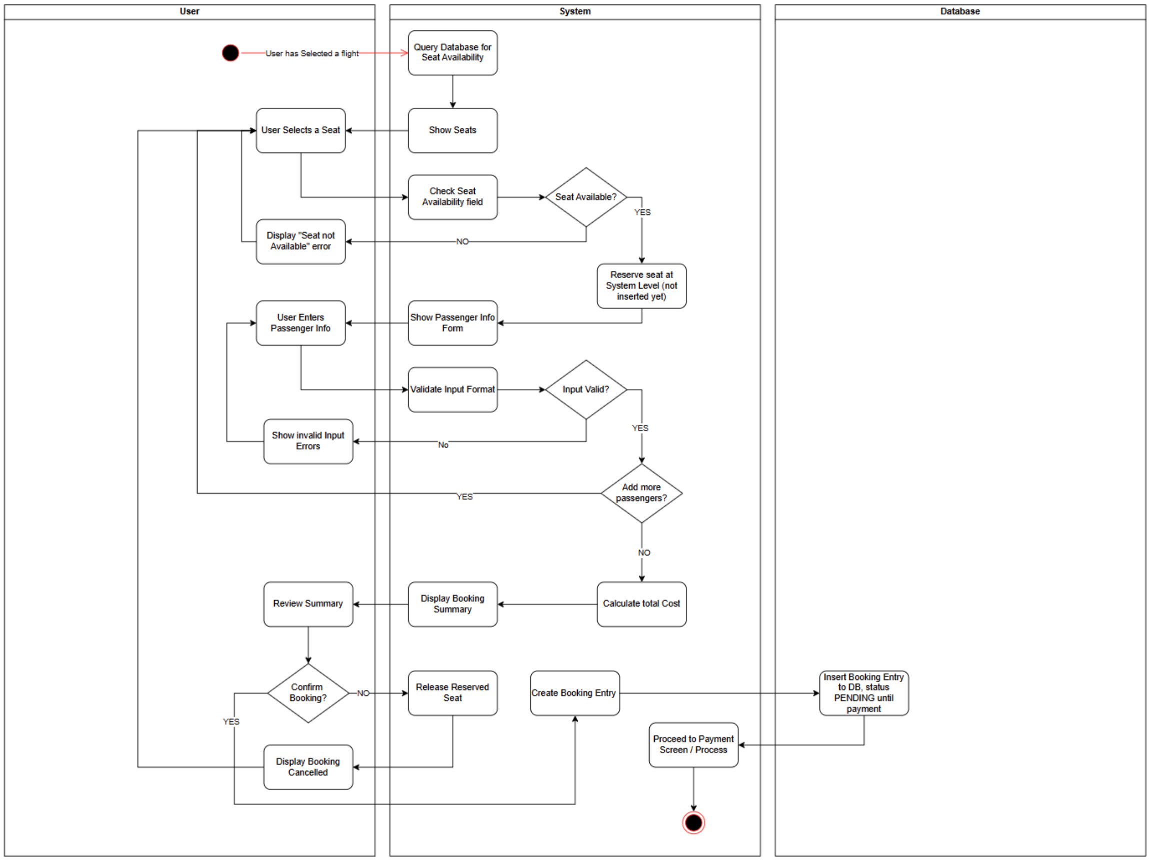
Login Activity Diagram



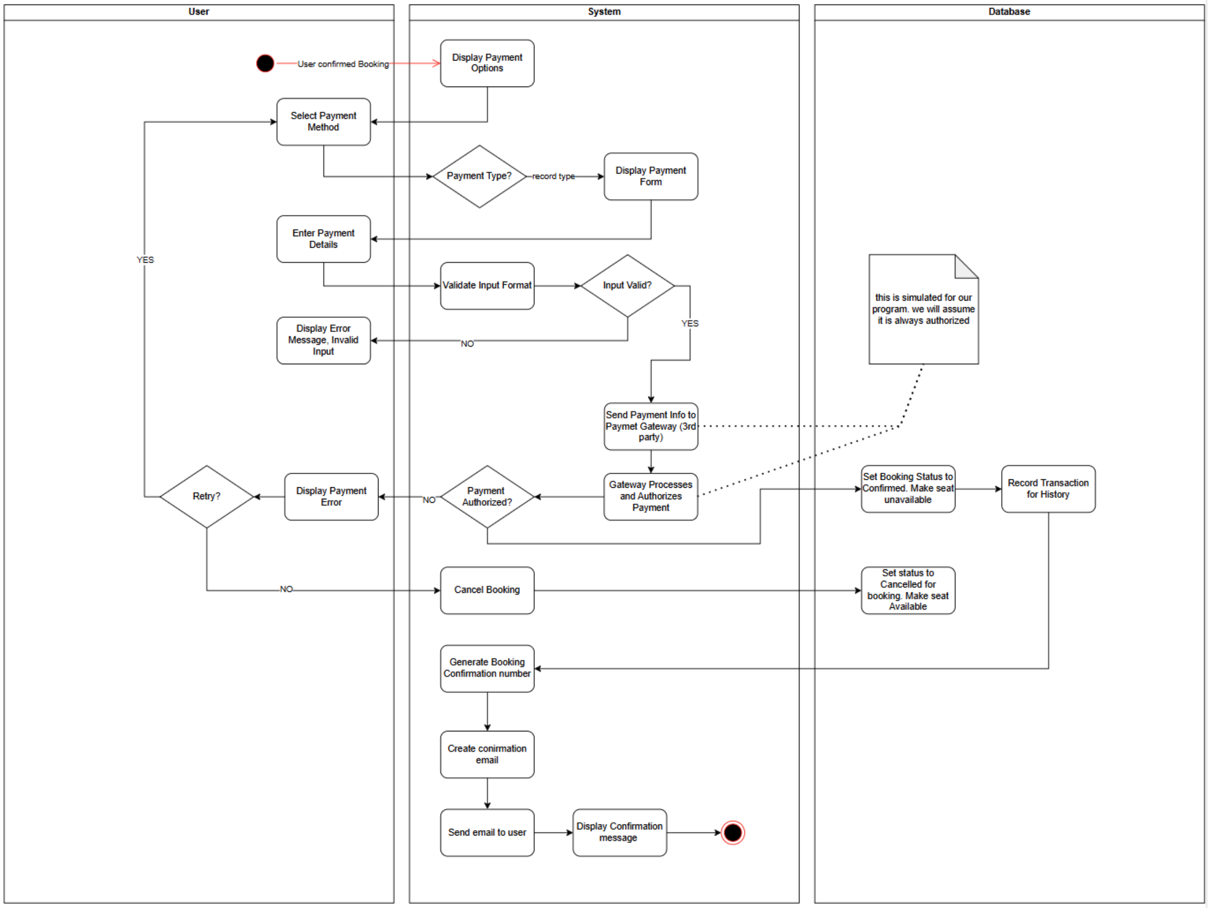
Browse Flights Activity Diagram



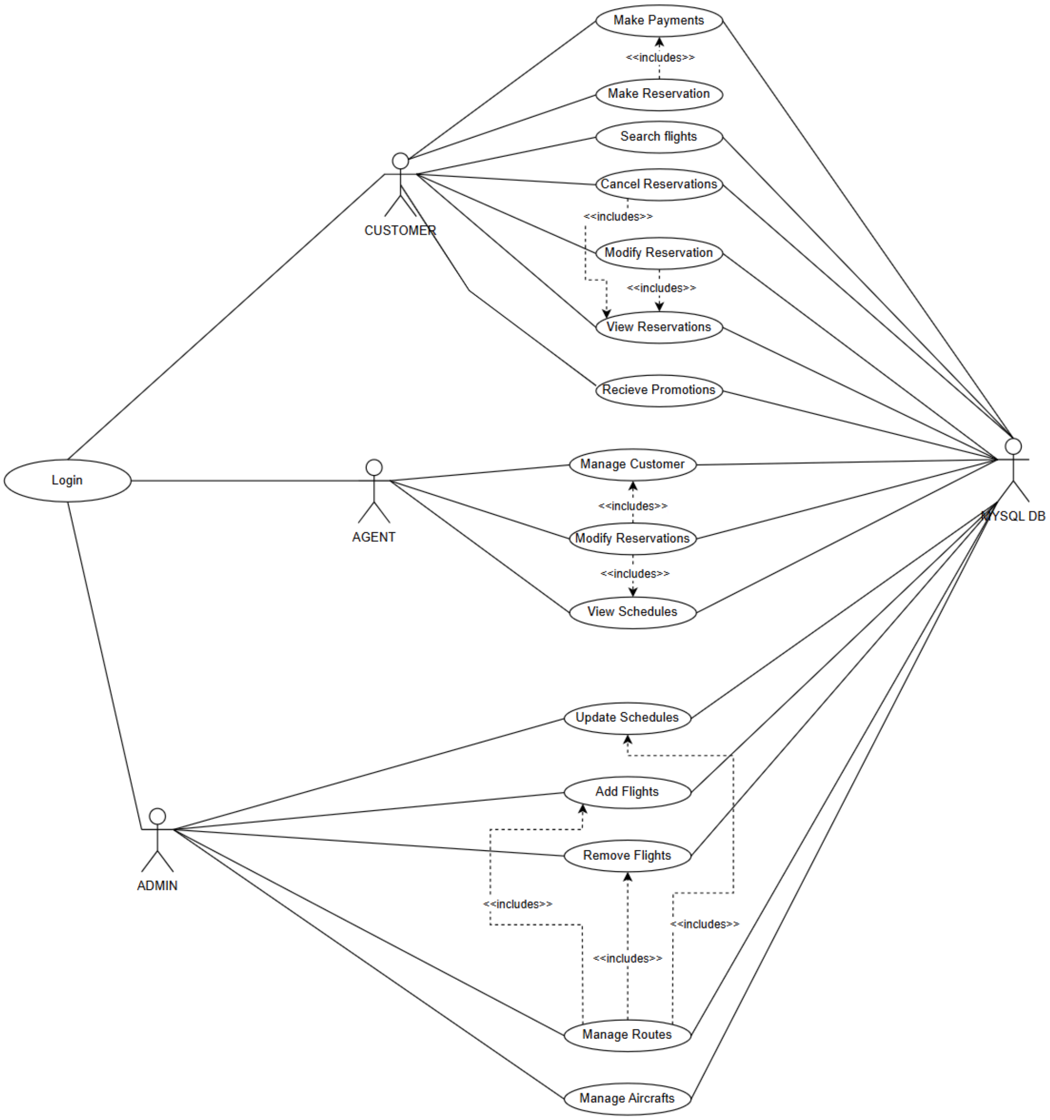
Book Flights Activity Diagram


User
System
Database
User has Selected a flight
Query Database for Seat Availability
Show Seats
User Selects a Seat
Check Seat Availability field
Seat Available?
YES
NO
Display "Seat not Available" error
Reserve seat at System Level (not inserted yet)
Show Passenger Info Form
User Enters Passenger Info
Validate Input Format
Input Valid?
YES
No
Show invalid Input Errors
Add more passengers?
YES
NO
Calculate total Cost
Display Booking Summary
Review Summary
Confirm Booking?
NO
YES
Release Reserved Seat
Create Booking Entry
Insert Booking Entry to DB, status PENDING until payment
Proceed to Payment Screen / Process
Display Booking Cancelled

Make Payment Activity Diagram


User
System
Database
User confirmed Booking
Display Payment Options
Select Payment Method
Payment Type?
record type
Display Payment Form
Enter Payment Details
YES
Validate Input Format
Input Valid?
this is simulated for our program. we will assume it is always authorized
YES
Display Error Message, Invalid Input
NO
Send Payment Info to Paymet Gateway (3rd party)
Retry?
Display Payment Error
NO
Payment Authorized?
Gateway Processes and Authorizes Payment
Set Booking Status to Confirmed. Make seat unavailable
Record Transaction for History
NO
Cancel Booking
Set status to Cancelled for booking. Make seat Available
Generate Booking Confirmation number
Create conirmation email
Send email to user
Display Confirmation message

2. Use Case Diagram



3. Scenarios for Use Cases

‘Login Scenario’

Preconditions:

- User is not currently logged into the system.
- A valid User Account exists in the System Database for the user.

The User opens the application and *selects* the Login option. The system *displays* the Login Form, and the user *enters* their username and password and *submits* the credentials. The system *retrieves* the corresponding User Account record from the Database, *validates* the credentials, and *determines* the user's role as either Customer, Flight Agent, or System Admin. If validation succeeds, the system *creates* a new Session object and *loads* the appropriate dashboard view for that role. If validation fails, the system *displays* an error message and *prompts* the User to try again.

Postconditions:

- A valid Session is established and associated with the logged-in User and their role.
- The appropriate role-specific home screen is displayed.

'Search a Flight Scenario'

Preconditions:

- User is logged into the System as an Agent or Customer.
- There is at least one Flight logged into the system that is available to book.

The Agent/Customer logs into the system and *selects* to search for a flight. The system *displays* the 'search flights' form and the user *enters* the search criteria, then *presses* the 'Go' button. The system *validates* the input format of each search criterion and *displays* necessary messages to the user. Upon validation, the system *retrieves* the flights that match that criteria from the database and *displays* them on the GUI to the user. The display *shows* flight details and *displays* a 'Book Now' button to continue the reservation process.

Postconditions:

- Matching flights are displayed to the user.

'Make a Reservation Scenario'

Preconditions:

- User is logged into the System as an Agent or Customer.
- A search scenario has been performed.
- Flights matching the search criteria are displayed.
- There is at least one Flight being displayed that is available to be booked.

The Agent/Customer is logged into the system and has performed a valid *search* for flights. Upon browsing the flights, the user *selects* 'Book Now' for a certain flight. The system *loads* the reservation form showing the selected flight details and available seats. The user *selects* one or more seats, *enters* passenger information (name, contact details, and any required identification), and *confirms* the number of tickets. The system *validates* all entered data and *checks* that the selected seats are still available. If validation fails, the system *displays* appropriate error messages and *prompts* the user to correct the input. If validation succeeds, the system *creates* a new reservation for the selected flight, *associates* it with the current customer (or the customer chosen by the Agent), *updates* the flight's available seat count, and *sets* the reservation status to 'Pending Payment.' The system then *displays* a reservation summary (reservation ID, flight details, passengers, seats, and total fare) and *provides* an option to proceed to the Make Payment process.

Postconditions:

- A new reservation is created in the system and stored in the database.
- The selected seats are temporarily reserved and no longer shown as available for that flight.
- The reservation is associated with the appropriate customer account and marked as 'Pending'.

'Cancel Bookings Scenario'

Preconditions:

- Customer is logged into the system.
- The Customer has at least one active Reservation in the Database.
- Relevant Reservations are visible (e.g., via a previous View Reservations step).

The Customer navigates to the My Reservations page and *selects* a specific Reservation they wish to cancel. The system *displays* the Reservation Details, including flight information and cancellation rules, and then the Customer *clicks* the Cancel Booking action. The system *checks* the Reservation status and *verifies* that cancellation is allowed according to the fare rules and time constraints. If allowed, the system *updates* the Reservation status to 'Cancelled', *releases* the reserved Seat Inventory back to the Flight, and *records* the cancellation timestamp in the Database. The system then *displays* a cancellation confirmation message and any applicable refund information to the Customer.

Postconditions:

- The selected Reservation has status 'Cancelled' in the Database.
- Associated Seat counts for the Flight are updated.
- A cancellation confirmation is visible to the Customer.

'View Reservations Scenario'

Preconditions:

- Customer is logged into the system.

The Customer navigates to the View Reservations section from their dashboard and *requests* to see their upcoming and past Reservations. The system *queries* the Database for all Reservation records linked to the Customer's User Account. The system then *sorts* the results by date and status and *renders* a Reservation List showing key details such as Flight Number, Departure Time, Destination, and Reservation Status. For each entry, the system also *provides* actions such as 'View Details', 'Cancel', or 'Modify' where applicable.

Postconditions:

- A list of the Customer's Reservations is displayed, ready for further actions (view, cancel, modify).

'Manage Customer Scenario'

Preconditions:

- Flight Agent is logged into the system.
- Customer records exist in the Database.

The Flight Agent from their agent dashboard *selects* the Manage Customer function. The system *displays* a Customer Search Form, and the agent *enters* search criteria such as Customer ID, last name, or email and *submits* the query. The system *retrieves* matching Customer records and *displays* a Customer Results List. The agent *selects* a specific Customer, and the system *shows* detailed Customer Profile information, including contact details and past Reservations. The agent may *edit* fields such as phone number or address and *save* the updates. The system then *validates* the new data and *writes* the changes back to the Database, *confirming* success to the Flight Agent.

Postconditions:

- The selected Customer record is updated in the Database with any changes made by the Flight Agent.

'Modify Reservations Scenario'

Preconditions:

- Flight Agent is logged into the system.
- A valid Customer and associated Reservation exist.
- The Reservation is modifiable according to business rules.

The Flight Agent *selects* the Modify Reservations option and *searches* for a Reservation using Reservation ID or Customer details. The system *retrieves* the matching Reservation and *displays* current flight, seat, and fare details. The agent *chooses* a modification action, such as changing the Flight

Date or Seat Class. The system *prompts* for new search criteria, *searches* for alternative Flights, and *displays* the available flight options. The agent *selects* a new Flight and *confirms* the change. The system *validates* seat availability and any fare differences, then *updates* the Reservation record, *releases* the old seat, *reserves* a new seat, and *stores* all changes in the Database. A confirmation of the updated reservation is then *shown* to the Flight Agent.

Postconditions:

- The Reservation details (flight/seat/etc.) are updated in the Database.
- The old seat inventory is released, and new seat inventory is allocated correctly.

‘View Schedules Scenario’

Preconditions:

- Flight Agent is logged into the system.
- At least one Flight schedule exists in the Database.

The Flight Agent *selects* View Schedules from the agent dashboard. The system *displays* a Schedule Filter Form allowing the agent to *enter* criteria such as origin, destination, and date. After the agent *submits* the filter, the system *queries* the Database for Flights that match the criteria and *constructs* a Schedule List. The system *renders* the list showing departure and arrival times, flight numbers, aircraft types, and seat availability. The Flight Agent may *scroll* through the results or *select* a particular Flight to view detailed schedule information.

Postconditions:

- A filtered list of Flight Schedules is presented to the Flight Agent for reference or further actions.

‘Add Flights Scenario’

Preconditions:

- System Admin is logged into the system.
- Required Route and Aircraft records exist in the Database.

The System Admin *selects* the Add Flights function from the admin dashboard. The system *displays* an Add Flight Form where the admin *enters* data such as Route (origin and destination), departure date and time, arrival time, Aircraft ID, and base fare. The system *validates* that the selected Route and Aircraft exist and that there are no conflicting schedules for that aircraft. If all checks pass, the system *creates* a new Flight object, *assigns* it a unique Flight Number, and *persists* it to the Database.

The system then *confirms* that the flight has been created and *makes* it available for Search Flights, View Schedules, and booking use cases.

Postconditions:

- A new Flight record with associated schedule is stored in the Database.
- The new Flight becomes available to Search flights, View Schedules, and booking use cases.

‘Remove Flights Scenario’

Preconditions:

- System Admin is logged into the system.
- The target Flight exists in the Database.
- The Flight meets business rules for removal (e.g., no active Reservations or an allowed cancellation window).

The System Admin *selects* the Remove Flights use case and *searches* for a Flight by Flight Number, Route, or date. The system *retrieves* matching Flight records and *shows* them to the admin. The admin *selects* the specific Flight to remove. The system *checks* for any active Reservations tied to that Flight. In this basic scenario, there are none, so the system *displays* a confirmation prompt. The admin *confirms* the removal, and the system *marks* the Flight as “Inactive” or *deletes* it physically from the Database, depending on business rules. The system *ensures* the flight is no longer returned by Search Queries or Schedule Queries and *displays* a success message.

Postconditions:

- The selected Flight is no longer available for new bookings or schedule display.
- The Database reflects the removal (inactive status or deleted record).

‘Manage Routes Scenario’

Preconditions:

- System Admin is logged into the system.

The System Admin *chooses* the Manage Routes function. The system *displays* a Route Management Screen listing existing Routes and offering actions to add or update them. The admin *selects* the option to create a new Route. The system *prompts* for origin airport, destination airport, and any metadata such as distance or standard duration. The System Admin *enters* the details and *submits* the route form. The system *checks* that a Route with the same origin and destination does not already exist and then *creates* a new Route entity in the Database. A confirmation message is *displayed*, and the new route becomes selectable when adding Flights.

Postconditions:

- A new Route record exists in the Database.
- The new Route is available for use in the Add Flights use case.

‘Manage Aircrafts Scenario’

Preconditions:

- System Admin is logged into the system.

The System Admin *selects* the Manage Aircrafts option from the admin dashboard. The system *displays* a Fleet Management Screen listing existing Aircraft records with details such as tail number, model, and seat capacity. The admin *chooses* to add a new Aircraft, and the system *presents* an Add Aircraft Form. The admin *enters* details like Aircraft ID, model, manufacturer, and seat configuration, then *submits* the form. The system *validates* that the Aircraft ID is unique and that mandatory fields are provided, then *creates* a new Aircraft entity and *stores* it in the Database. The system *confirms* the addition, and the new Aircraft appears in the fleet list and becomes assignable when adding Flights.

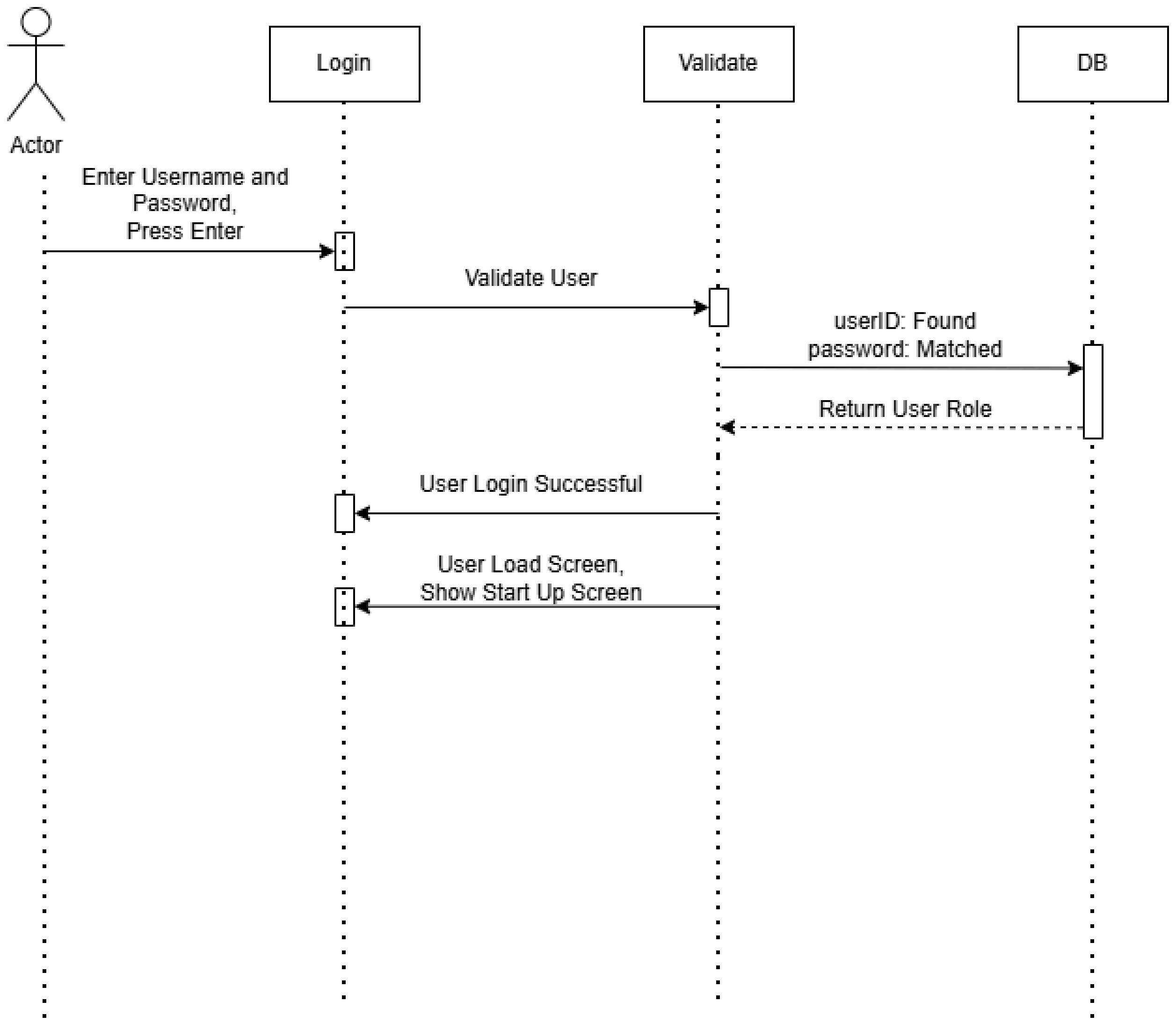
Postconditions:

- A new Aircraft Record is persisted in the Database.
- The Aircraft is available for selection in the Add Flights use case.

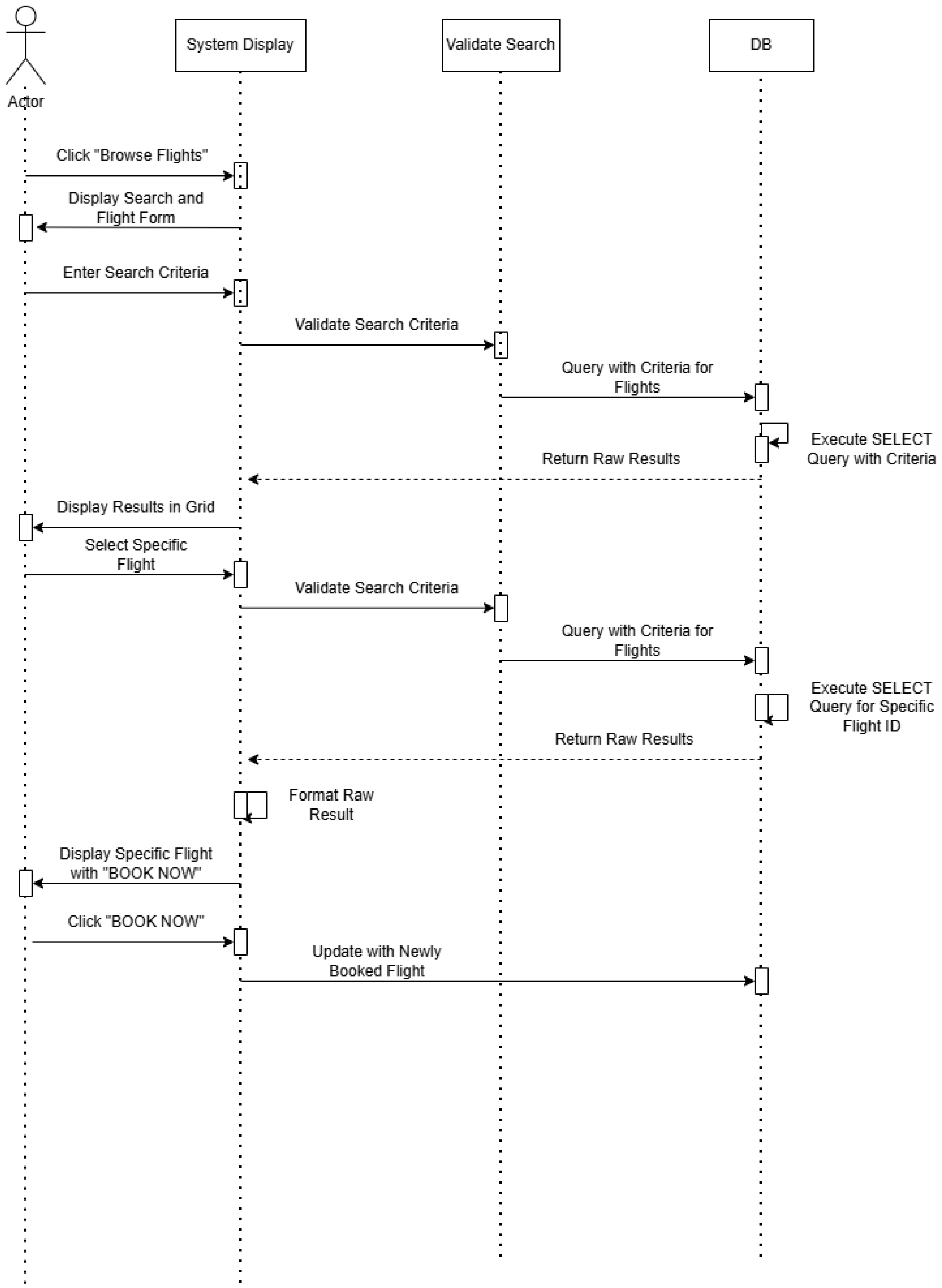
4. Sequence Diagrams for Scenarios

The following sequence diagrams show the sequence of major processes in the application.

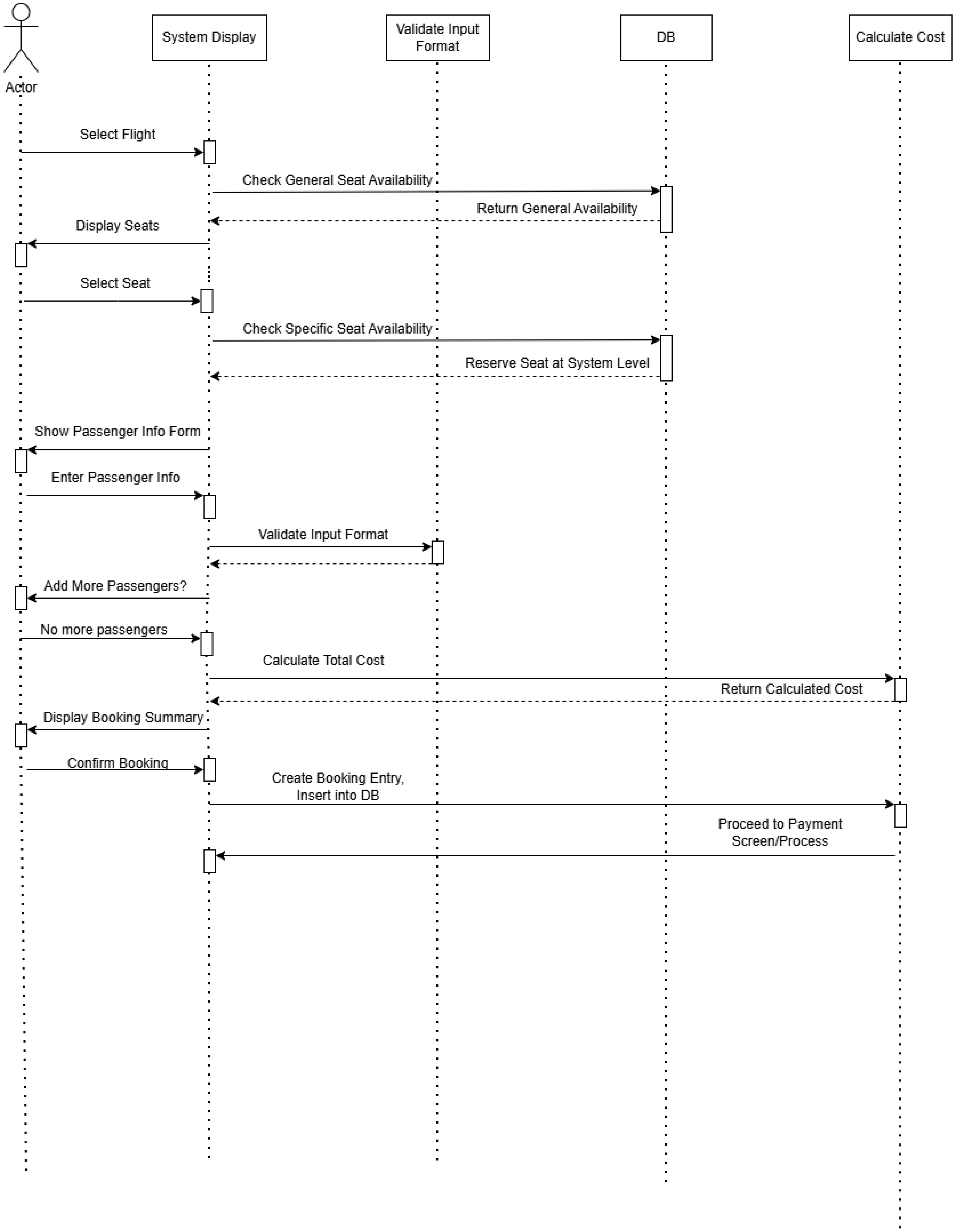
Login Sequence Diagram



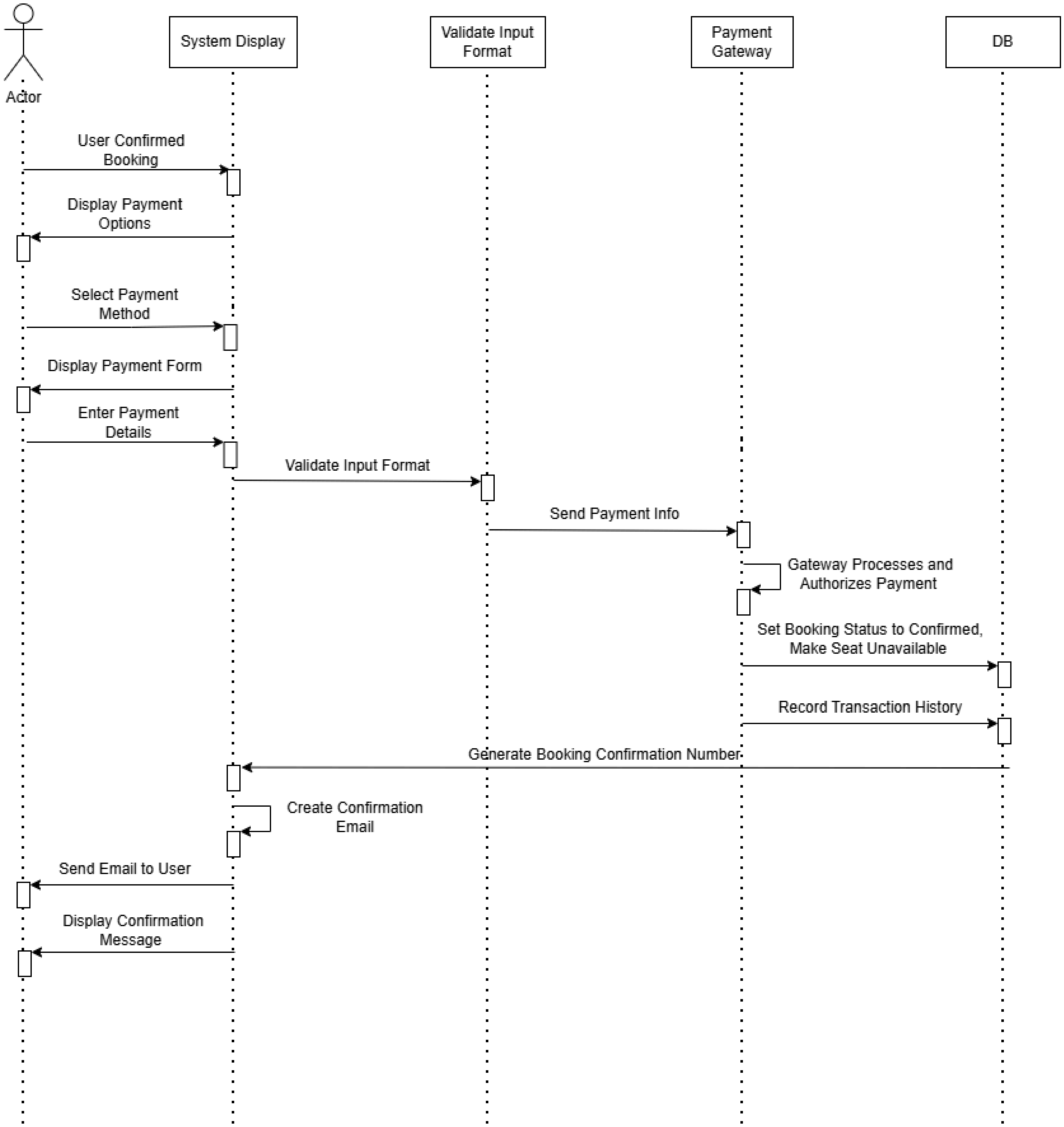
Browse Flights Sequence Diagram


Actor
System Display
Validate Search
DB
Click "Browse Flights"
Display Search and Flight Form
Enter Search Criteria
Validate Search Criteria
Query with Criteria for Flights
Execute SELECT Query with Criteria
Return Raw Results
Display Results in Grid
Select Specific Flight
Validate Search Criteria
Query with Criteria for Flights
Execute SELECT Query for Specific Flight ID
Return Raw Results
Format Raw Result
Display Specific Flight with "BOOK NOW"
Click "BOOK NOW"
Update with Newly Booked Flight

Reserve Flights Sequence Diagram



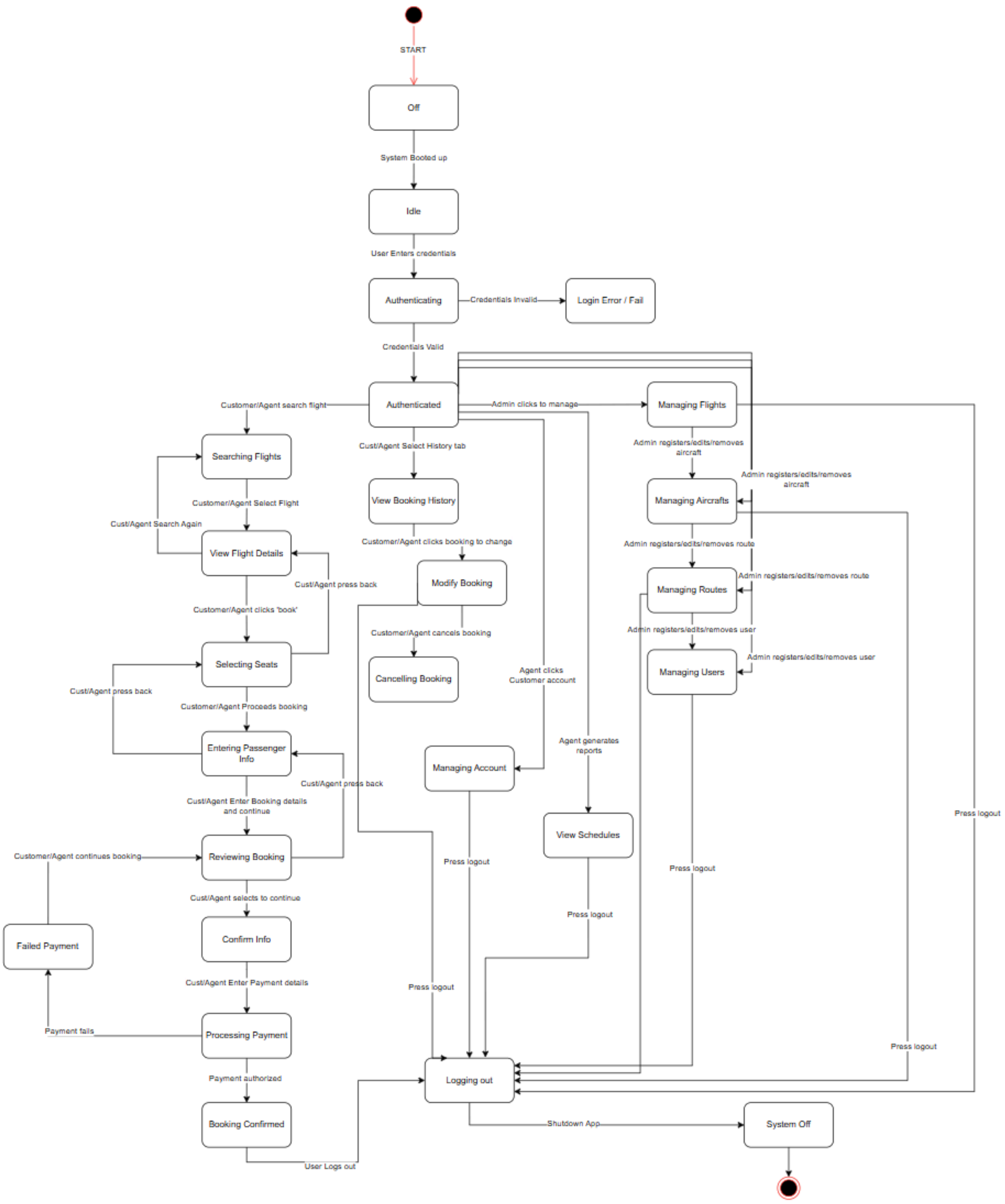
Make Payment Sequence Diagram



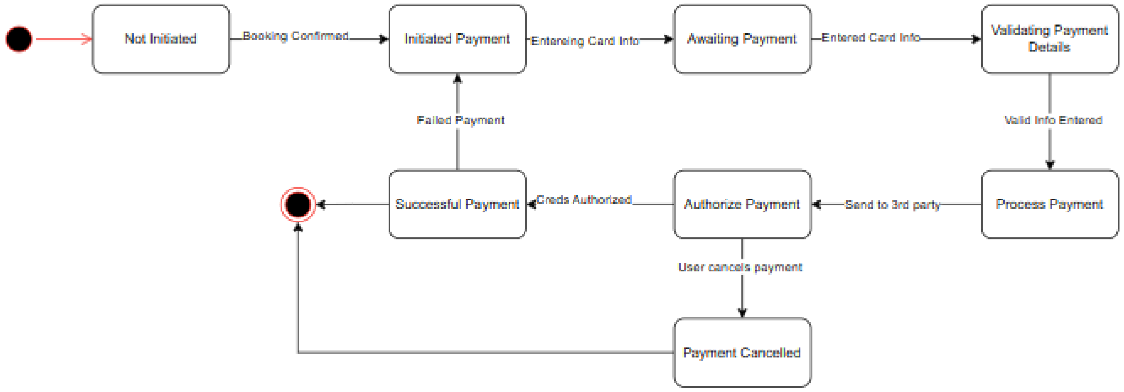
5. State Transition Diagrams

The following state transition diagrams show the state of the system for many major processes in the application.

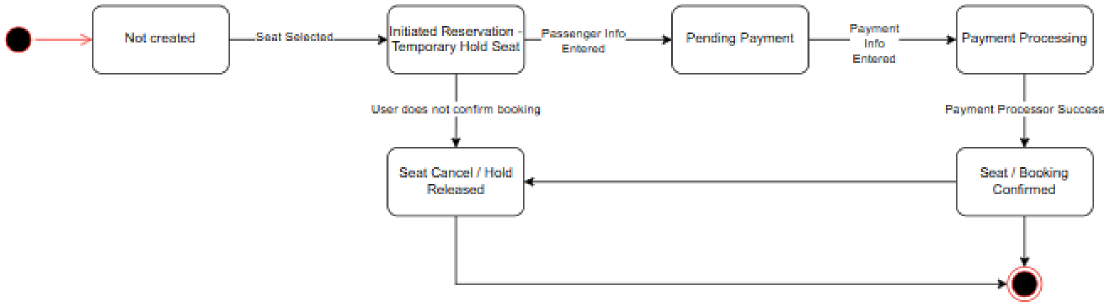
Entire System State Diagram



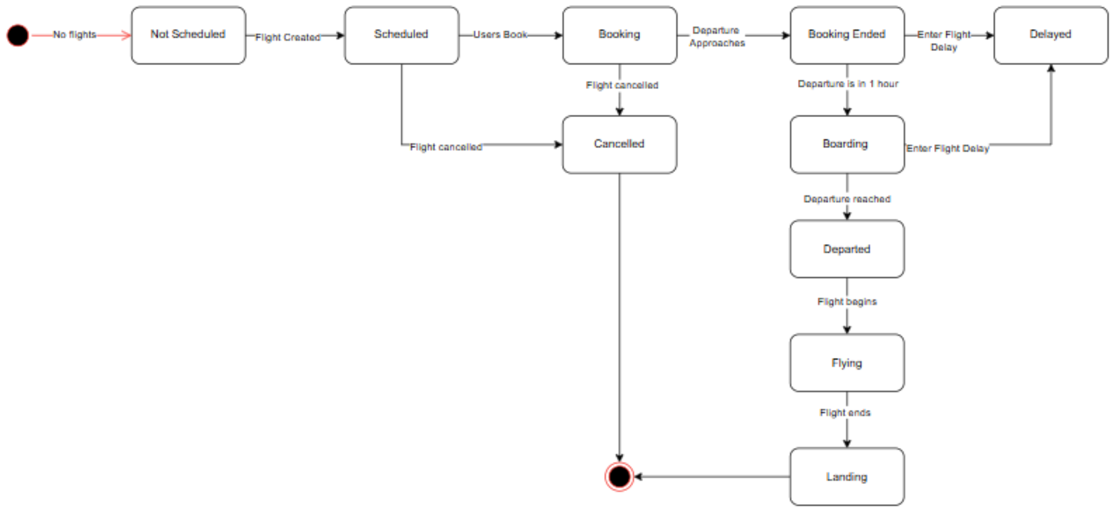
Payment State Diagram



Reservation State Diagram

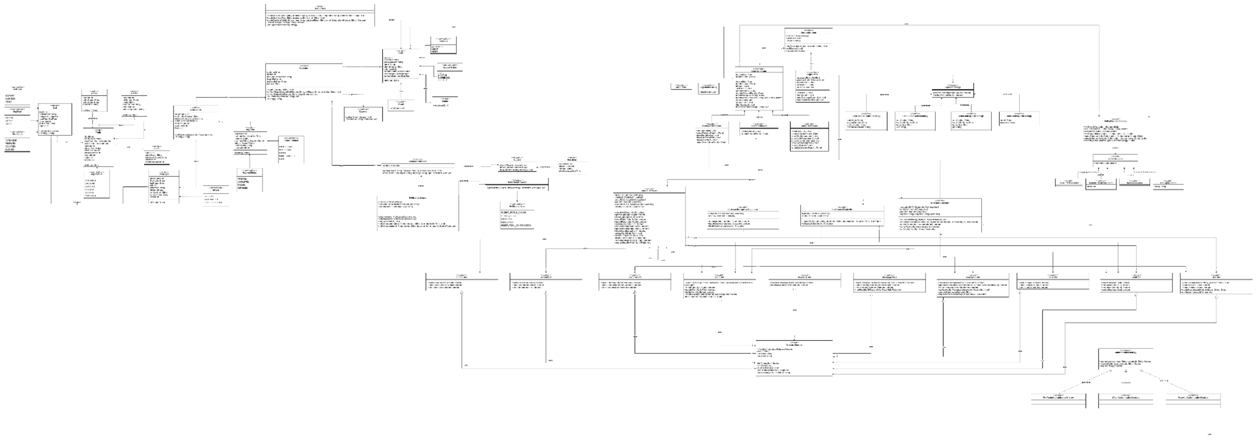

Not created
Seat Selected
Initiated Reservation - Temporary Hold Seat
Passenger Info Entered
Pending Payment
Payment Info Entered
Payment Processing
Payment Processor Success
Seat / Booking Confirmed
User does not confirm booking
Seat Cancel / Hold Released

Flight State Diagram



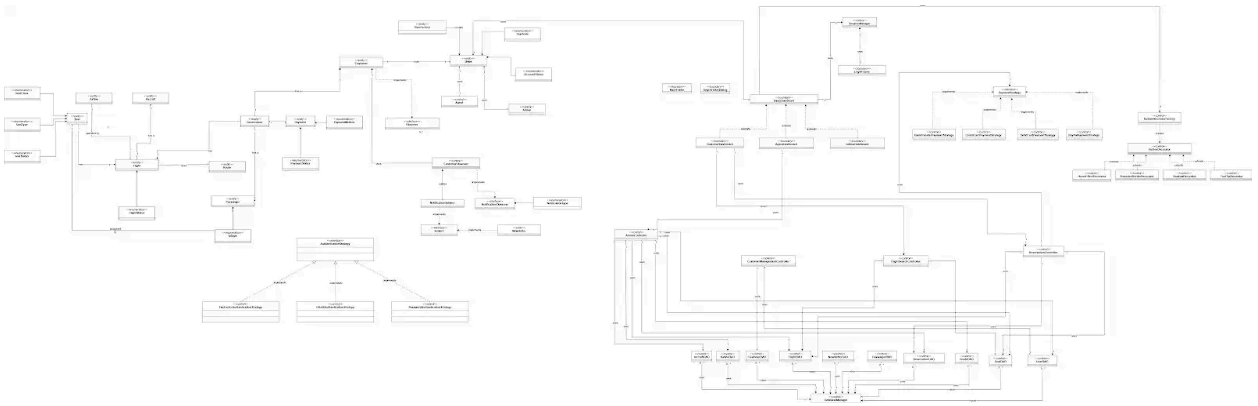
6. UML Class Diagram

Below is the entire UML. The two separates versions are found directly under this one.
Click to See Full Image



Below is the UML showing only relations between classes:

UML Class Diagram Relationships:
UML Class Diagram Relationships

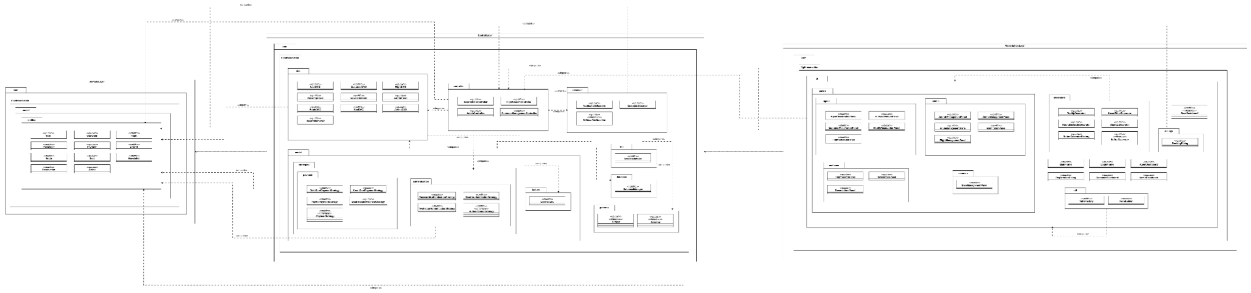


Below is the UML of all classes ordered alphabetically:

UML Class Diagram Classes:
UML Class Diagram Classes

7. Package Diagram

Click to See Full Image



Three-Layer System: Package and Class Breakdown

Domain Layer

Package: com.flightreservation.model.entities

Class Name	Type	Visibility
User	Class	Public
Customer	Class	Public

Flight	Class	Public
Reservation	Class	Public
Passenger	Class	Public
Payment	Class	Public
Aircraft	Class	Public
Airline	Class	Public
Route	Class	Public
Seat	Class	Public
Newsletter	Class	Public

Domain Layer Dependencies

- Depends on: None (pure domain model)
- Does NOT depend on: Control or Presentation layers

Control Layer

Package: com.flightreservation.controller

Class Name	Type	Visibility
ReservationController	Class	Public
FlightSearchController	Class	Public
AdminController	Class	Public
CustomerManagementController	Class	Public

Package: com.flightreservation.dao

Class Name	Type	Visibility
ReservationDAO	Class	Public
FlightDAO	Class	Public
CustomerDAO	Class	Public
UserDAO	Class	Public
PassengerDAO	Class	Public

SeatDAO	Class	Public
AircraftDAO	Class	Public
AirlineDAO	Class	Public
RouteDAO	Class	Public
NewsletterDAO	Class	Public

Package: com.flightreservation.model.strategies.payment

Class Name	Type	Visibility
PaymentStrategy	Interface	Public
CreditCardPaymentStrategy	Class	Public
DebitCardPaymentStrategy	Class	Public
PayPalPaymentStrategy	Class	Public
BankTransferPaymentStrategy	Class	Public

Package: com.flightreservation.model.strategies.authentication

Class Name	Type	Visibility
AuthenticationStrategy	Interface	Public
PasswordAuthenticationStrategy	Class	Public
OAuthAuthenticationStrategy	Class	Public
TwoFactorAuthenticationStrategy	Class	Public

Package: com.flightreservation.model.factory

Class Name	Type	Visibility
UserFactory	Class	Public

Package: com.flightreservation.observer

Class Name	Type	Visibility
NotificationObserver	Interface	Public
NotificationSubject	Class	Public

CustomerObserver	Class	Public

Package: com.flightreservation.patterns

Class Name	Type	Visibility
Observer	Interface	Public
Subject	Interface	Public

Package: com.flightreservation.database

Class Name	Type	Visibility
DatabaseManager	Class	Public

Package: com.flightreservation.util

Class Name	Type	Visibility
SessionManager	Class	Public

Control Layer Dependencies

- Depends on: Domain Layer (entities)
- Can depend on: Other Control packages
- Does NOT depend on: Presentation Layer

Presentation Layer

Package: com.flightreservation.ui

Class Name	Type	Visibility
BaseDashboard	Abstract Class	Public
CustomerDashboard	Class	Public
AdminDashboard	Class	Public
AgentDashboard	Class	Public
LoginFrame	Class	Public
RegistrationDialog	Class	Public
MainFrame	Class	Public

Package: com.flightreservation.ui.decorators

Class Name	Type	Visibility
ButtonDecorator	Abstract Class	Public
HoverEffectDecorator	Class	Public
TooltipDecorator	Class	Public
ShadowDecorator	Class	Public
RoundedBorderDecorator	Class	Public
ButtonDecoratorFactory	Class	Public

Package: com.flightreservation.ui.panels.admin

Class Name	Type	Visibility
AircraftManagementPanel	Class	Public
AirlineManagementPanel	Class	Public
FlightManagementPanel	Class	Public
NotificationPanel	Class	Public
RouteManagementPanel	Class	Public

Package: com.flightreservation.ui.panels.agent

Class Name	Type	Visibility
AgentReservationPanel	Class	Public
AllReservationsPanel	Class	Public
CustomerManagementPanel	Class	Public
FlightSchedulePanel	Class	Public
ModifyReservationPanel	Class	Public

Package: com.flightreservation.ui.panels.customer

Class Name	Type	Visibility
FlightSearchPanel	Class	Public
NewslettersPanel	Class	Public

ReservationsPanel	Class	Public

Package: com.flightreservation.ui.panels.common

Class Name	Type	Visibility
BaseManagementPanel	Class	Public

Package: com.flightreservation.ui.dialogs

Class Name	Type	Visibility
BookingDialog	Class	Public

Package: com.flightreservation.ui.util

Class Name	Type	Visibility
FormBuilder	Class	Public
TableFactory	Class	Public

Presentation Layer Dependencies

- Depends on: Control Layer (controllers, utilities)
- Does NOT depend on: Domain Layer directly

Public Classes Identification

All classes in the system are marked as public to allow proper access across layers, following the layering architecture:

- Domain Layer: All entity classes and enums are public to be accessible by the Control Layer
- Control Layer: All controllers, DAOs, strategies, factories, and utilities are public to be accessible by the Presentation Layer
- Presentation Layer: All UI components are public for proper component interaction