Term Project

Flight Reservation Web Application

Course: ENSF 614 – Fall 2023

Instructor: Mahmood Moussavi

Submission Date: December 3rd, 2023

Group 5: Mehreen Akmal 10134470, Mustayeen Abedin 30021056,
Romil Dhagat 30025587, Yene Irvine 30225899

Table Of Contents

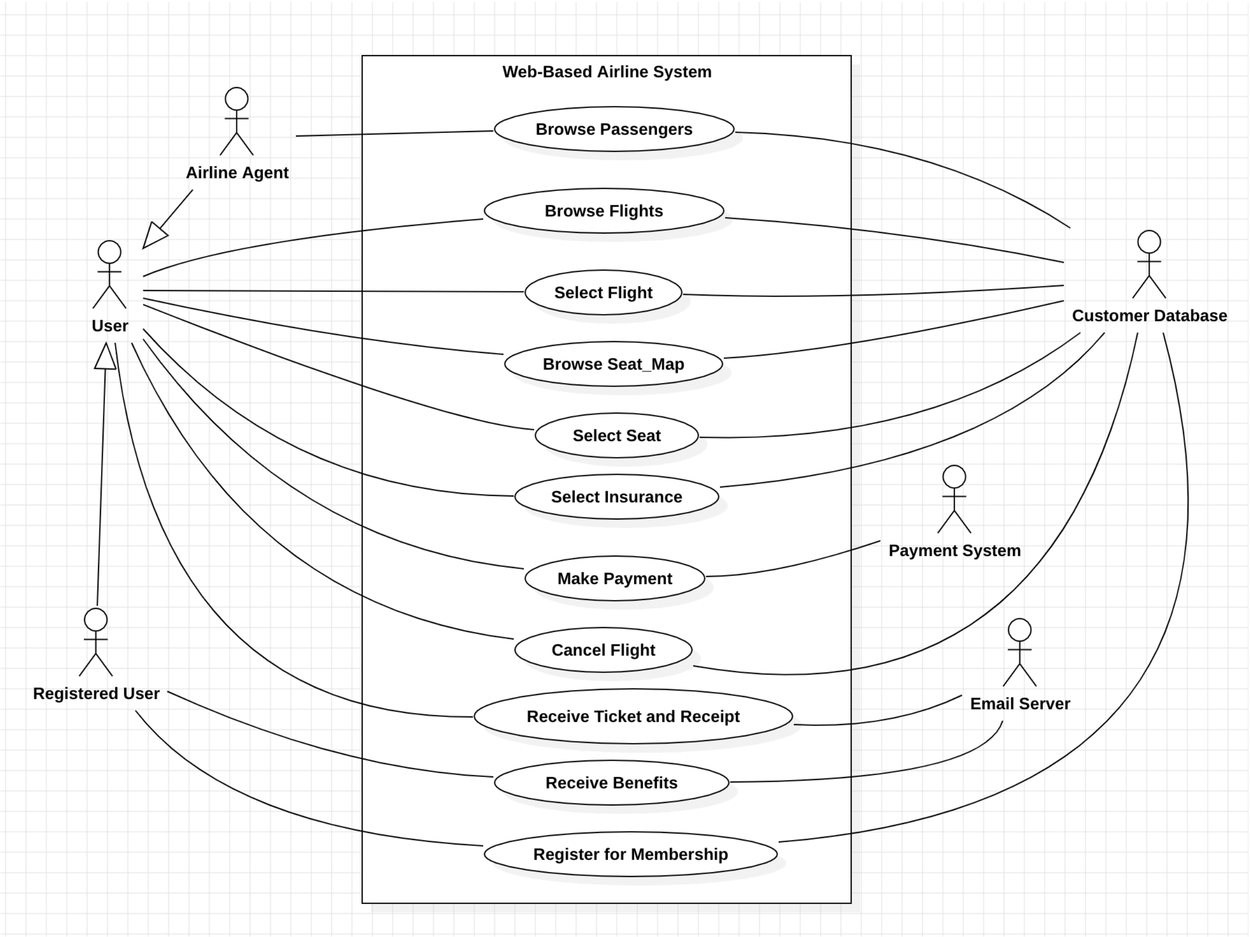
System Analysis

System Description

The Flight Reservation Web Application is a sophisticated and user-centric platform designed to ease the process of flight booking and management for a leading airline company. With a diverse range of user roles, including tourism agents, airline agents, registered users, and system administrators, the application provides a seamless and efficient experience for all stakeholders. Customers can easily browse and book one-way flights, select preferred seats from a graphical seat map, and make secure payments through credit cards, while registered users enjoy exclusive benefits such as monthly promotions and discounted airport lounge access. The system's dynamic pricing ensures fair and competitive seat pricing, and automated email notifications to keep users informed throughout the reservation process, from e-ticket delivery to payment receipts.

Behind the scenes, airline agents can efficiently manage passenger lists and provide superior customer service, while system administrators have complete control over the application's core functionalities, ensuring the smooth operation of flight schedules, crew assignments, and overall data management. With a focus on security, scalability, and ongoing development, the Flight Reservation Web Application stands as a robust solution that not only enhances the customer experience but also optimizes the airline's operational efficiency, making it a cornerstone in the aviation industry.

System’s Use Case Diagram



System Use Case Scenarios

As a team, we selected the 4 most important Use cases to write a scenario. The candidate objects are underlined (Not All Nouns) .

Select Seat Use Case Scenario

“Select Seat”

Diana selects the desired flight destination, clicks “proceed to seat selection” and is then prompted to the seat selection webpage.

The seat selection page displays a map of the plane showing where each seat is located, its type (Ordinary, Business and Comfort) and its availability, red for unavailable.

Diana now has the option to choose a seat by clicking an available seat, which appears green on click. if she chooses a seat not in the ordinary section then she is shown the additional amount for the upgraded seat.

If Diana has chosen a valid seat for the flight, she will be able to click “proceed to payment” that was grayed out prior to seat selection. Once Diana has confirmed she will be moved to the payment page.

Make Payment Use Case Scenario

“Make Payment”

After Emma selects the desired flight, and seat that she would like to purchase and clicks “proceed to payment”.

System moves to the payment page which displays the total cost of the ticket along with input fields for payment details. The system has the following input fields for payment:

“Card Number”, “Cardholder Name”, “Email”, “Expiry Month”, “Expiry Year”, “CVV”.

Emma enters the following information:

”4520883009165160”, “Emma Smith”, “emma@gmail.com”, “03”, “2026”, “566”.

Emma selects the option “Submit Payment”.

The system determines Emma has entered all the necessary and valid information.
System checks input entered meets the following conditions:

- Card number entered has 16 digits
- Cardholder name consists of a first and last name
- Expiry year has dropdown with only valid options
- CVV entered is 3 digits

The system indicates activity is complete, generates a booking reference number and returns to the home page. The system sends payment information to the payment system for processing. The system sends the ticket and payment receipt by email.

Member Registration Use Case Scenario

"Register for Membership"

Tom clicks the "Register for Membership" tab on the airline website. He is taken to the 'Membership Signup' webpage. The webpage prompts him to input his First Name, Last Name, email address and password to create an account. He inputs, respectively:

'Tom', 'Birds', 'tom123@gmail.com', 'Tomspassword2!''

Tom clicks " Apply for Membership". The system checks that all required entries are entered with a password consisting of at least 8 characters long, one uppercase letter, one lowercase letter, one digit and one symbol. If there is, the webpage informs Tom of this. For this scenario, we assume there is no existing account with that email and he successfully creates a new account.

On successfully creating a new account, an email is sent to the user with confirmation of account creation and benefits of being a member.

This account information is sent to the airline database. The system is now programmed to email Tom monthly promotion news on the first day of each month. He additionally has a discount applied when he uses airport lounges by showing the employee his membership status. He is also allowed to apply for a special airline credit card. Lastly, the system permits him one companion ticket each calendar year.

Manage Flight Information Use Case Scenario

"Adding New Flight Destination"

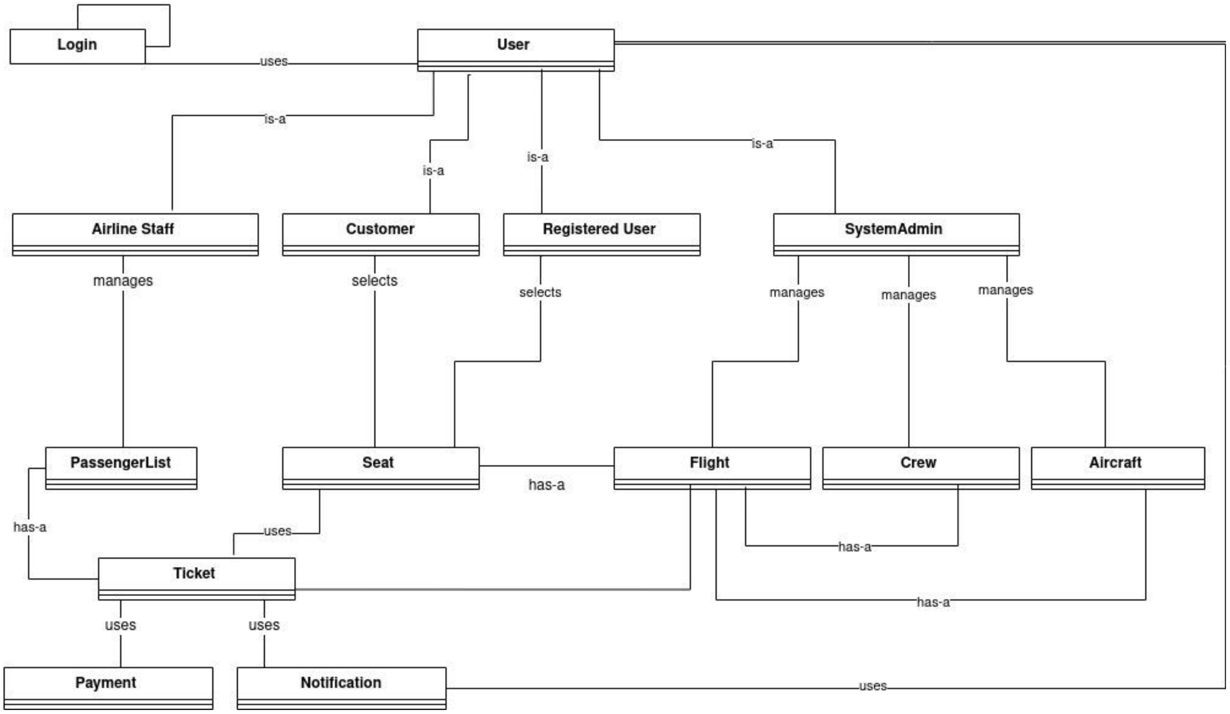
Randall is a system admin who is looking to add in a new flight destination. After a successful login, the system navigates him to the System Administration Portal where he selects the "View Flight Information" option. The system navigates him to a page displaying the current list of flights with some action options : "Add Flight", "Update", and "Delete Flight". He selects "Add Flight".

The system prompts him to enter the details of the flight, with the mandatory fields being: FlightID, Origin, Destination, Departure Date and Price.

Randall inputs “Calgary” as the Origin name, “456” as the FlightID, “Vancouver” as the Destination and the standard price as “500”. After reviewing the details, he submits the form. The system will then validate to check if this flight doesn’t already exist and all the required fields are filled.

After the system is satisfied with the request, it adds flight 456 with Vancouver as the list of flight destinations and displays the added flight at the bottom of the list. Randall proceeds to log out.

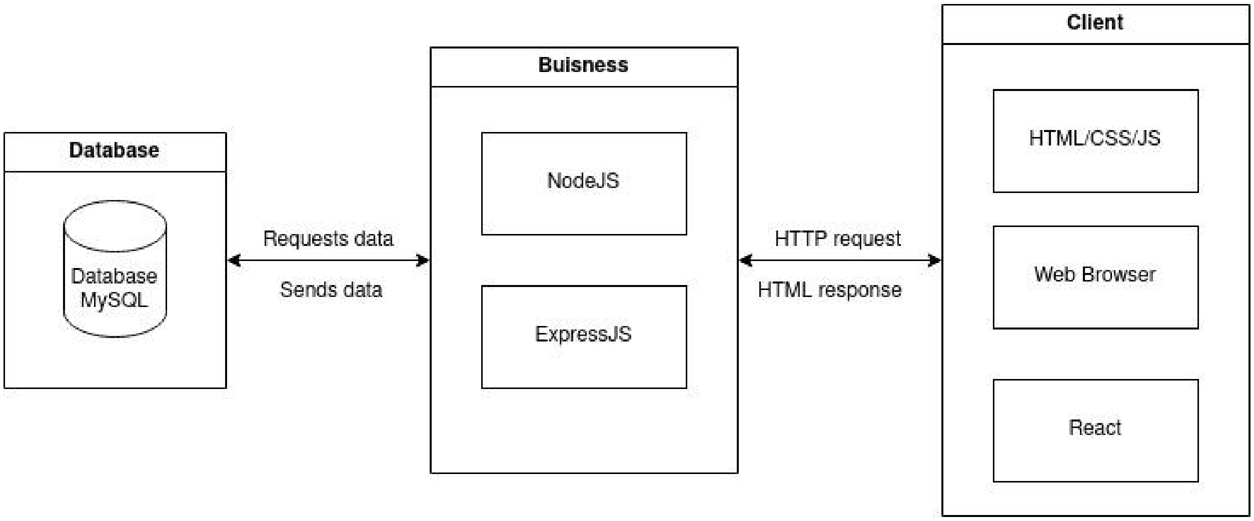
System Conceptual Model



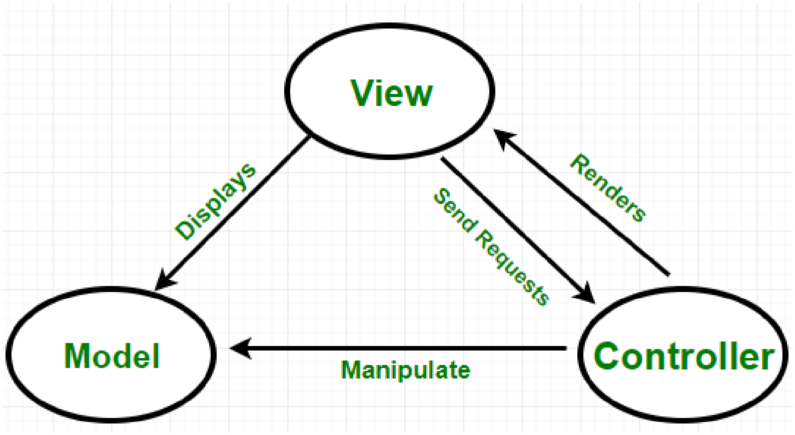
Domain Diagrams

Highlights of the System’s Architecture

The airline web application's system architecture applies a monolith SQL database and employs the Singleton, Observer (MVC), and Strategy design patterns. In the backend, Express.js serves as the routing and middleware web framework controller, ensuring effective communication between the server and client. The system layers are organized into the client (frontend) and server, where components in the frontend, acting as the presentation layer, seamlessly integrate user interaction and business logic. The data layer, represented by the monolith SQL database, serves as the persistence layer, handling data storage and retrieval.

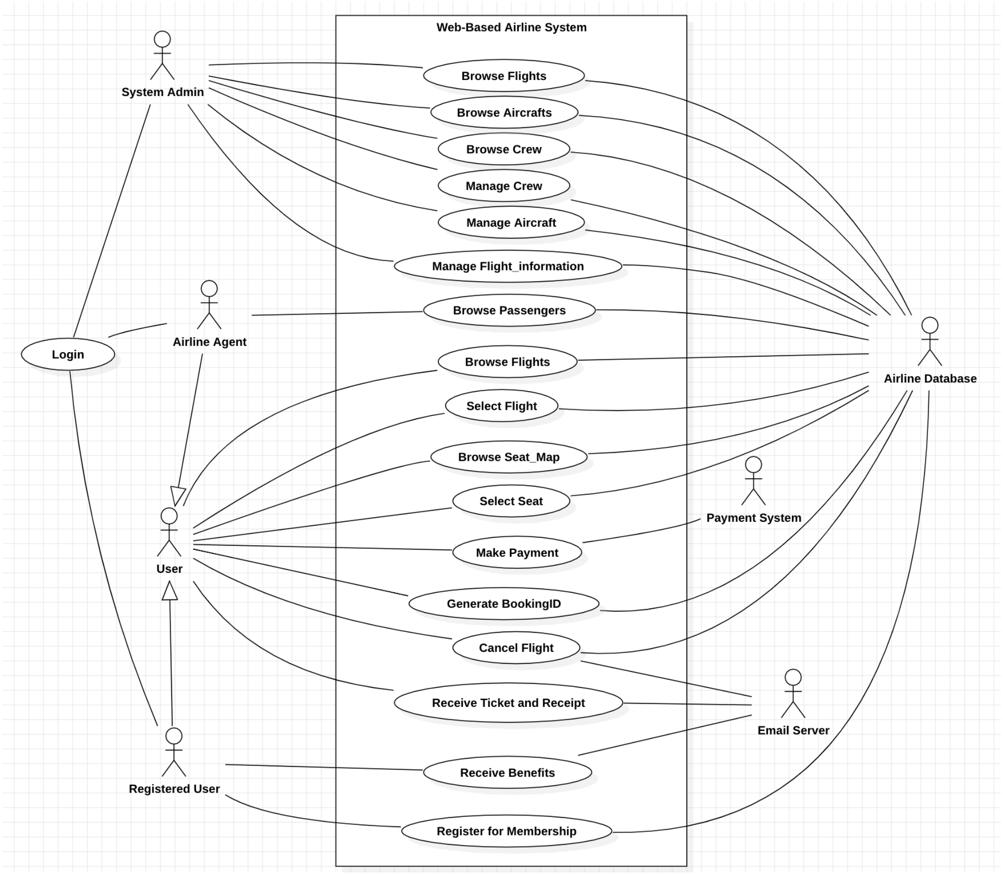


The key design patterns play a pivotal role in system functionality. The Singleton pattern ensures that the login maintains a single, globally available instance throughout the application, providing consistency for each user. The Observer pattern, integrated into the Model-View-Controller (MVC) structure, facilitates efficient communication between the user interface components and the underlying data model. In the Model-View-Controller (MVC) architecture, the Observer Pattern is showcased through the Model acting as the subject, maintaining the application's state. The Views, serving as observers, are notified of any changes in the Model's state and update their presentation accordingly. Finally the Controllers, acting as an intermediary, updating the Model in response to user input. Additionally, the Strategy pattern is implemented for email content, depending on the context the strategy for sending the email is evoked and concrete strategy is implemented based on the context for example sending an receipt has a different email sent than registering for promos.

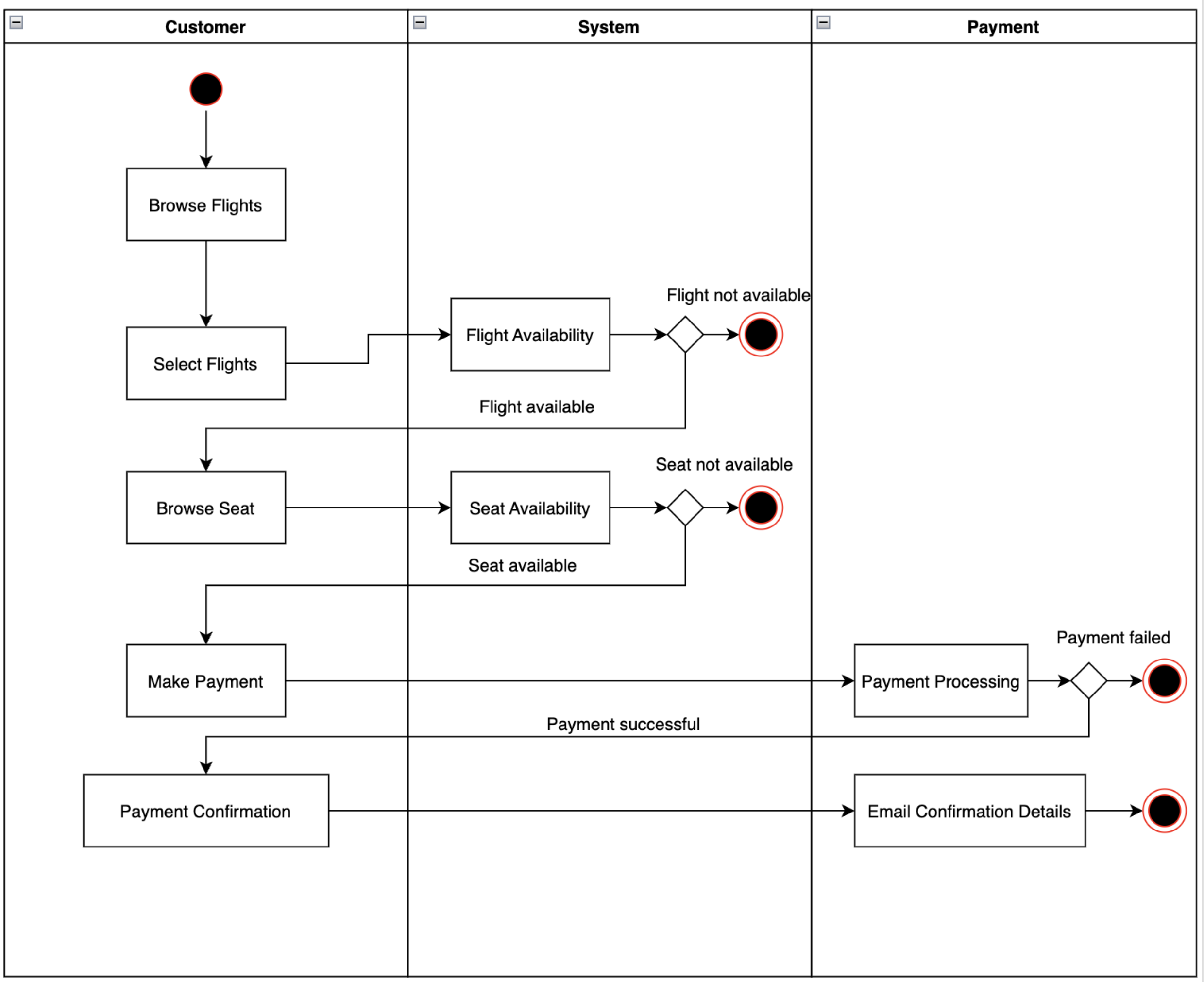


In terms of scalability and performance, React's modular structure allows for easy scaling of individual components within the client frontend. This modularity enables independent scaling, bug fixing, and feature additions without affecting other components. The combination of React and Node.js supports handling multiple concurrent client requisitions, contributing to optimal performance. The security measures include dedicated validation processes for payments and logins, ensuring the integrity of user data. The technology stack, comprising React for the frontend, Node.js for the backend, and MySQL for the database, forms a robust foundation for the airline web application's architecture.

Updated Use Case Diagram

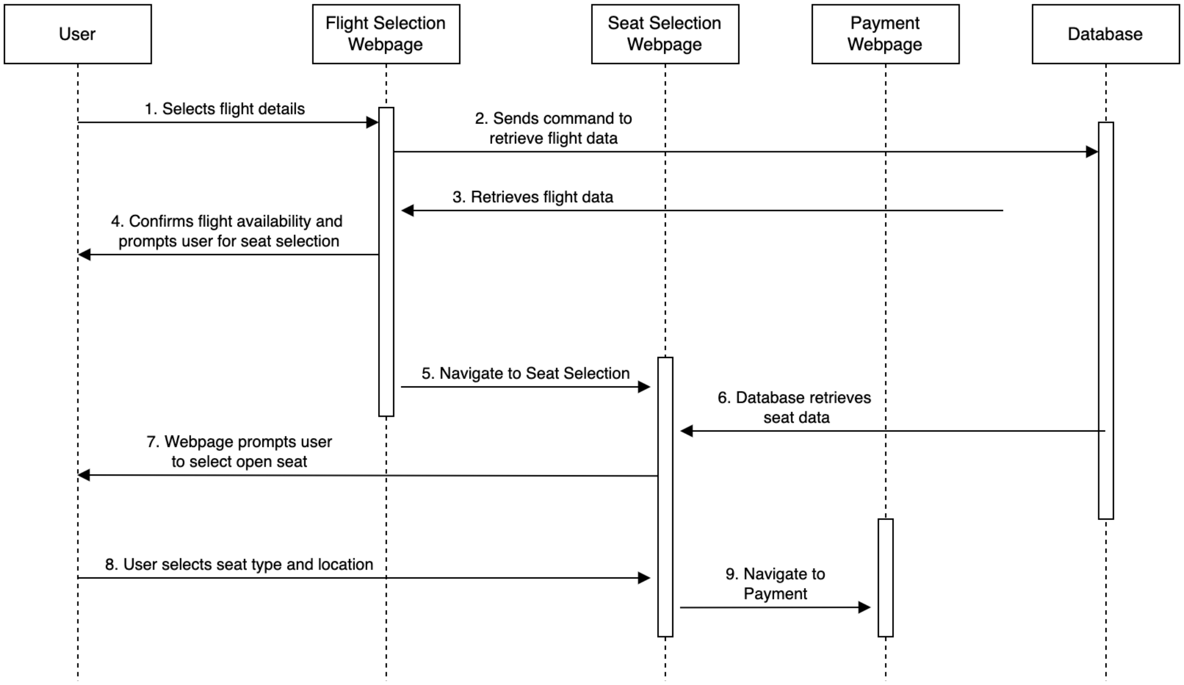


Systems Activity Diagram

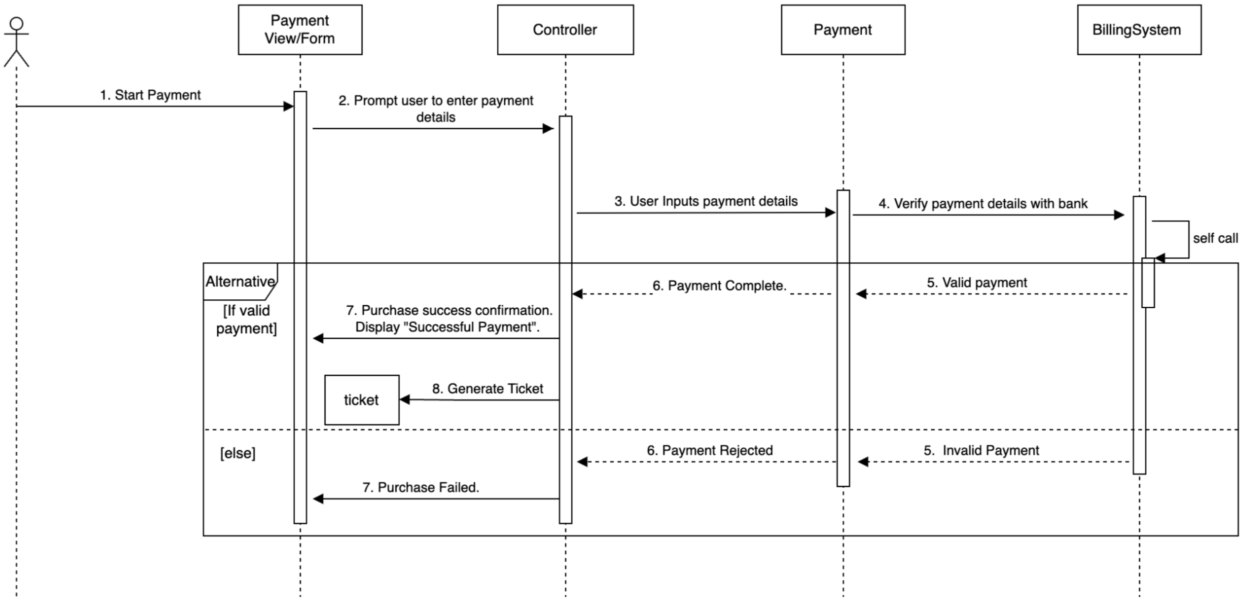


Sequence Diagram

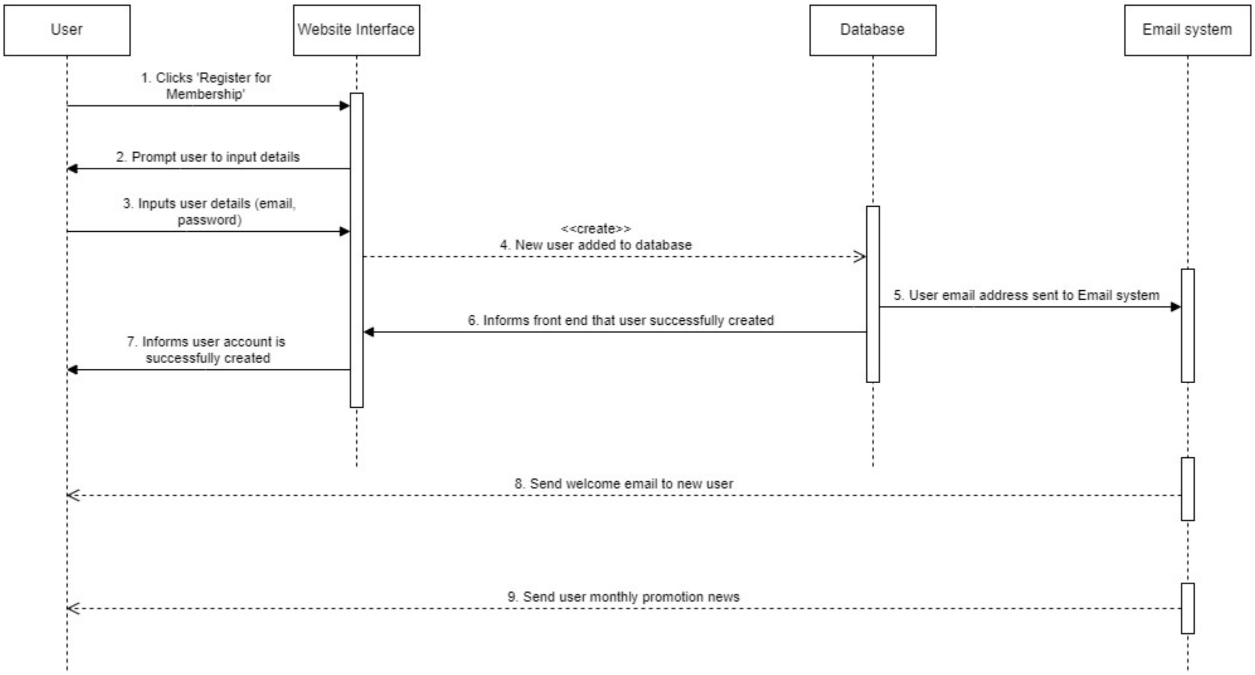
Select Seat Sequence Diagram


User
Flight Selection Webpage
Seat Selection Webpage
Payment Webpage
Database
1. Selects flight details
2. Sends command to retrieve flight data
3. Retrieves flight data
4. Confirms flight availability and prompts user for seat selection
5. Navigate to Seat Selection
6. Database retrieves seat data
7. Webpage prompts user to select open seat
8. User selects seat type and location
9. Navigate to Payment

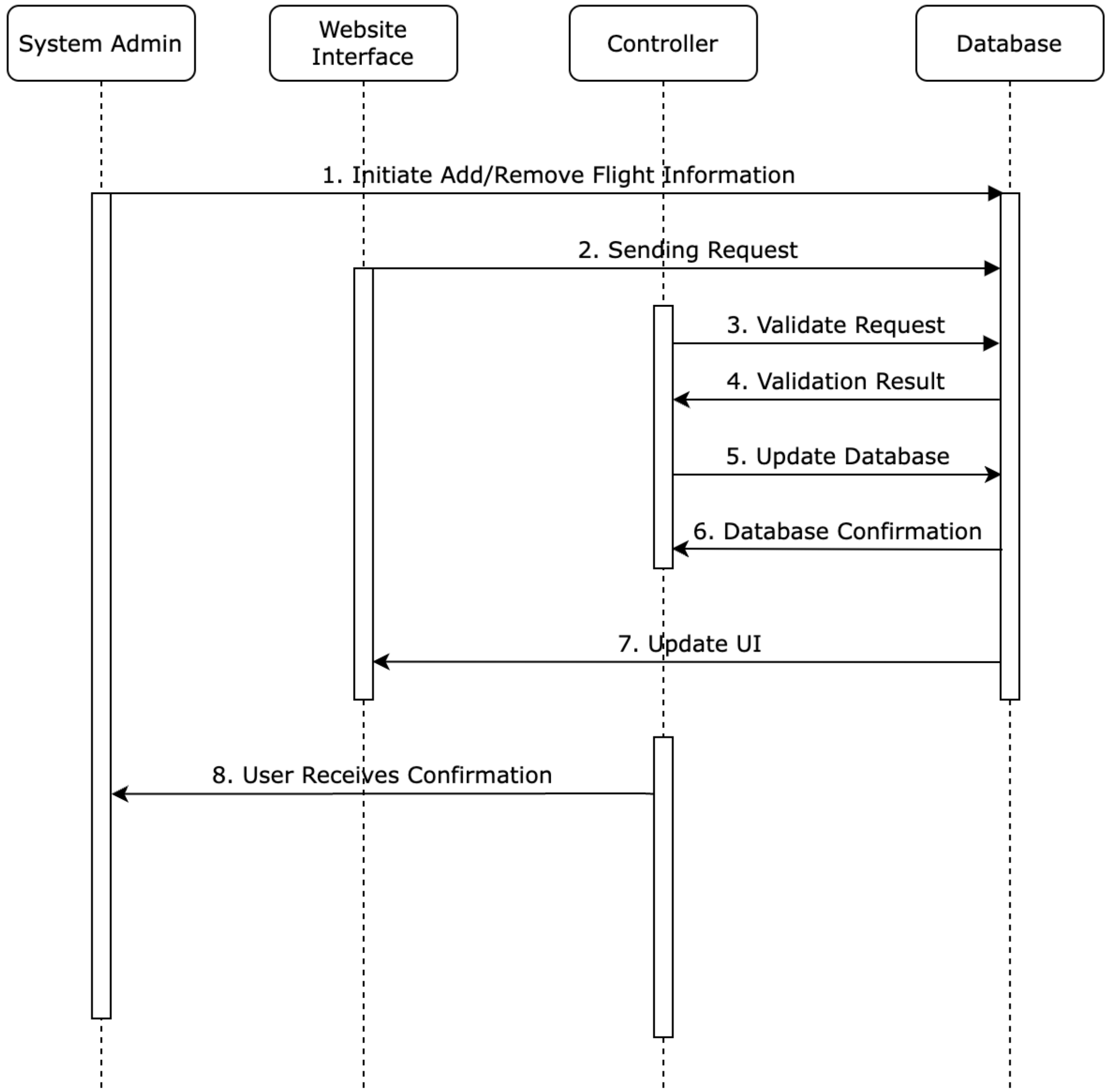
Make Payment Sequence Diagram



Member Registration Sequence Diagram

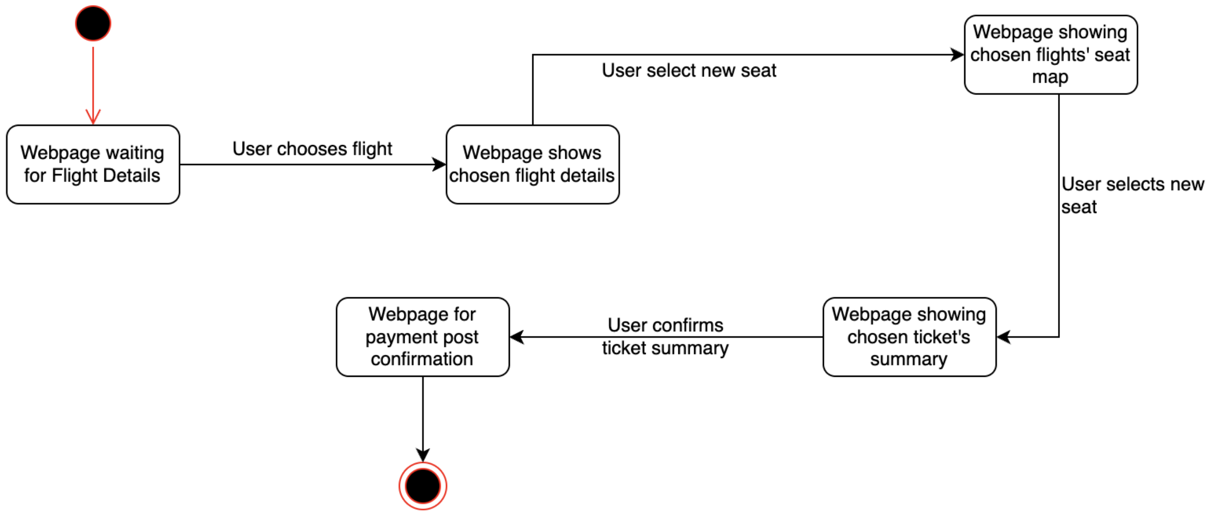

User
Website Interface
Database
Email system
1. Clicks 'Register for Membership'
2. Prompt user to input details
3. Inputs user details (email, password)
<<create>>
4. New user added to database
5. User email address sent to Email system
6. Informs front end that user successfully created
7. Informs user account is successfully created
8. Send welcome email to new user
9. Send user monthly promotion news

Manage Flight Sequence Diagram

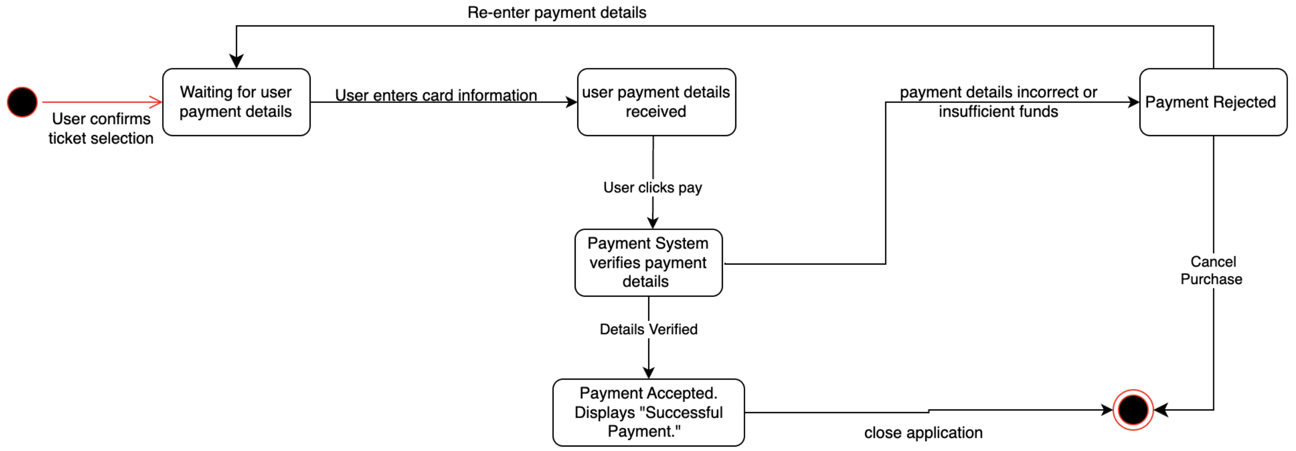


State Transition Diagram

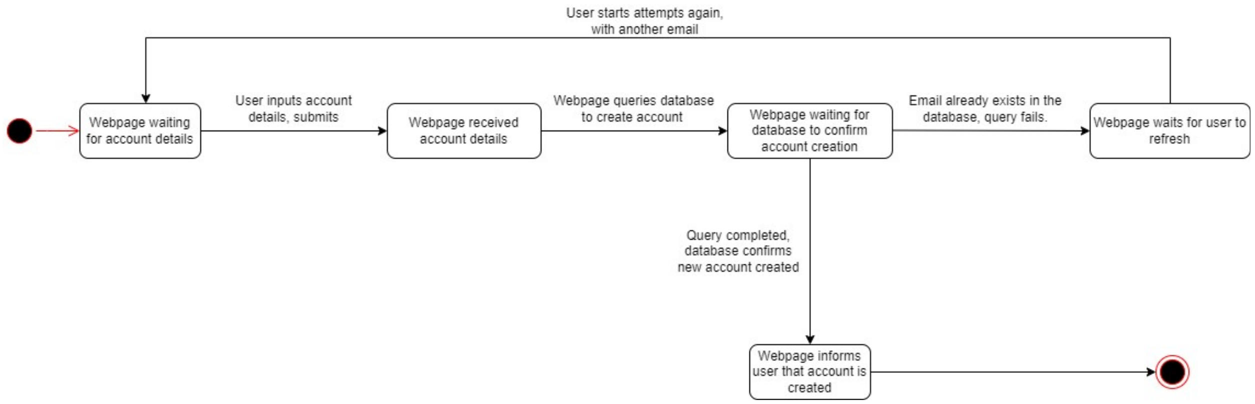
Select Seat State Transition Diagram


Webpage waiting for Flight Details
User chooses flight
Webpage shows chosen flight details
User select new seat
Webpage showing chosen flights' seat map
User selects new seat
Webpage showing chosen ticket's summary
User confirms ticket summary
Webpage for payment post confirmation

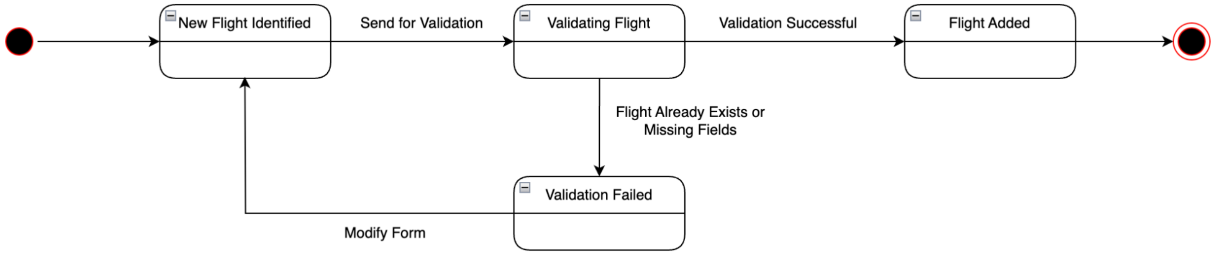
Make Payment State Transition Diagram


Re-enter payment details
User confirms ticket selection
Waiting for user payment details
User enters card information
user payment details received
payment details incorrect or insufficient funds
Payment Rejected
User clicks pay
Payment System verifies payment details
Cancel Purchase
Details Verified
Payment Accepted. Displays "Successful Payment."
close application

Member Registration State Transition Diagram

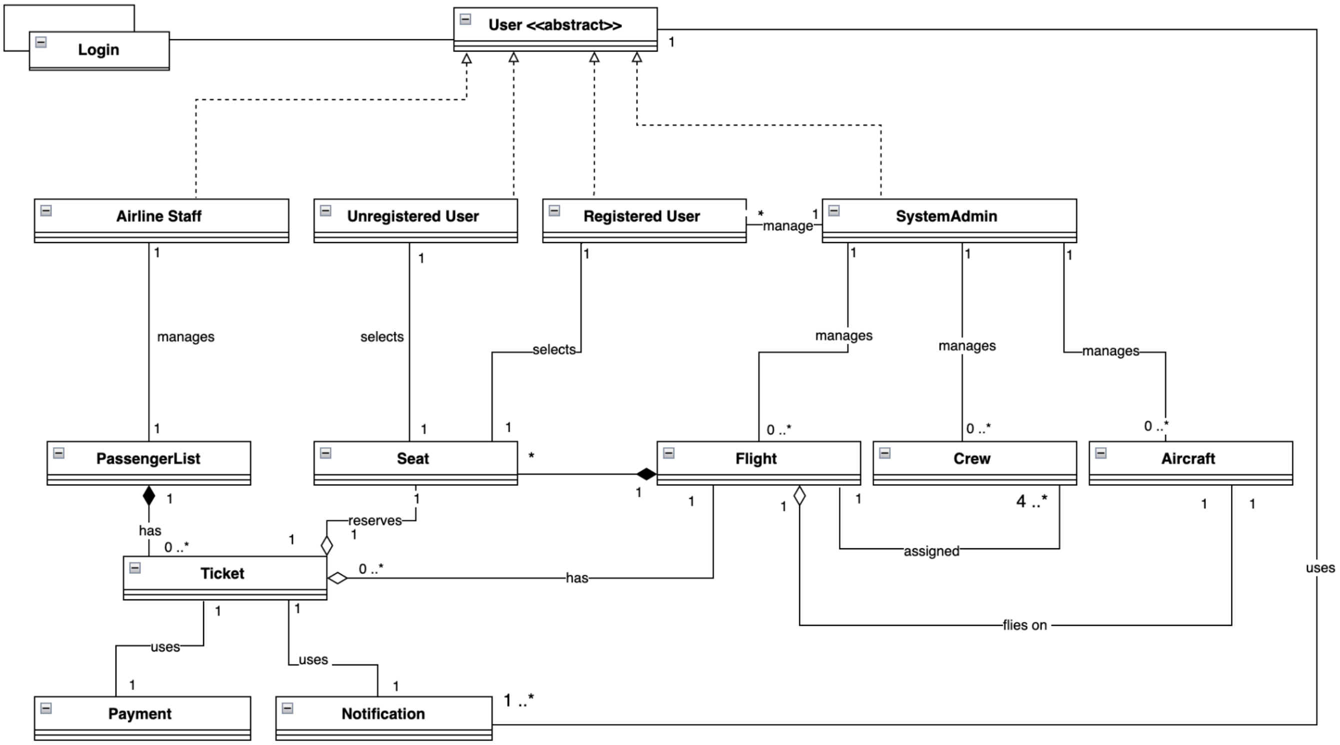

User starts attempts again, with another email
Webpage waiting for account details
User inputs account details, submits
Webpage received account details
Webpage queries database to create account
Webpage waiting for database to confirm account creation
Email already exists in the database, query fails.
Webpage waits for user to refresh
Query completed, database confirms new account created
Webpage informs user that account is created

Manage Flight State Transition Diagram



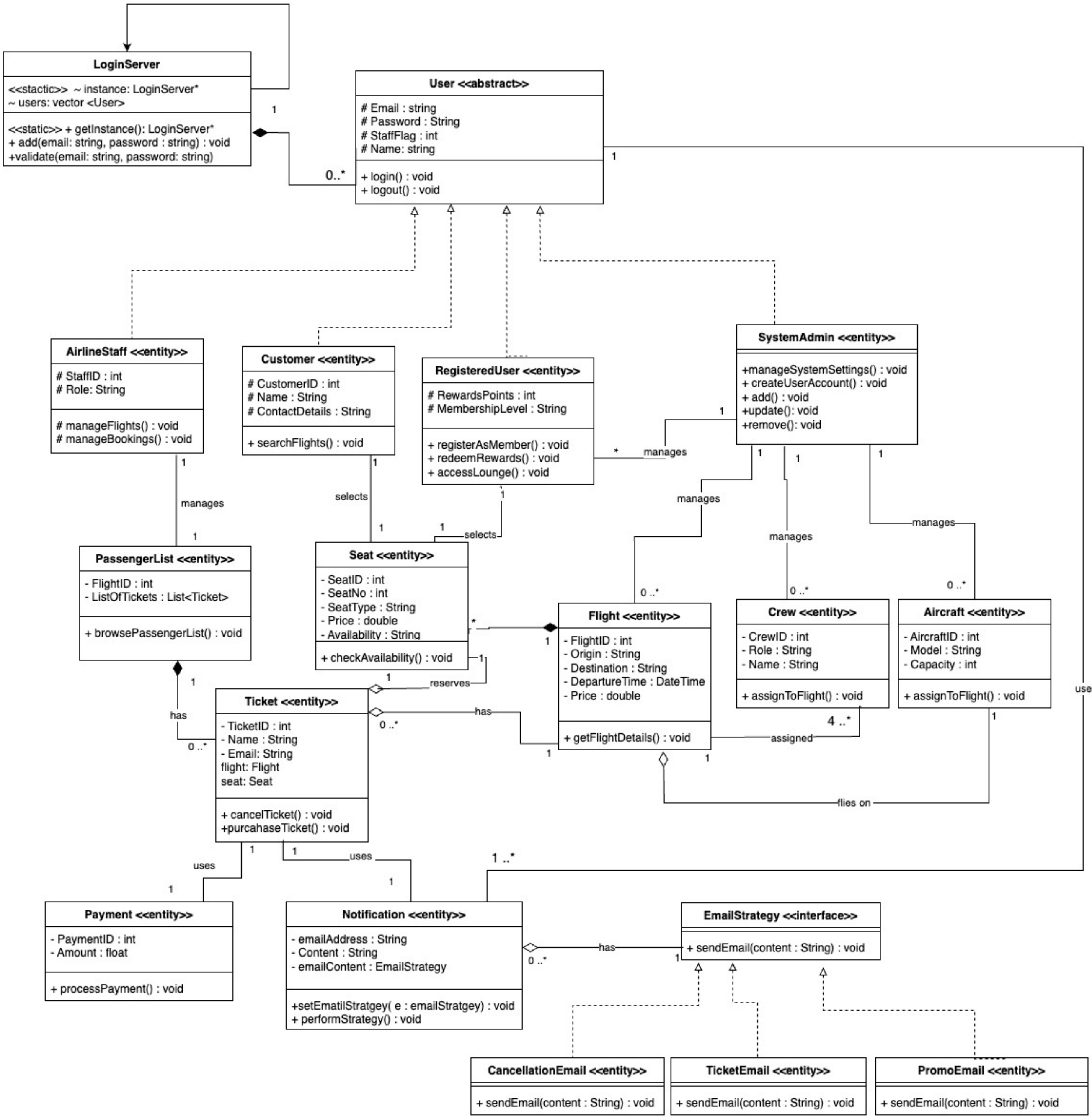
System’s Domain Class Diagram

without attributes and functionalities(only relationships and functionalities)



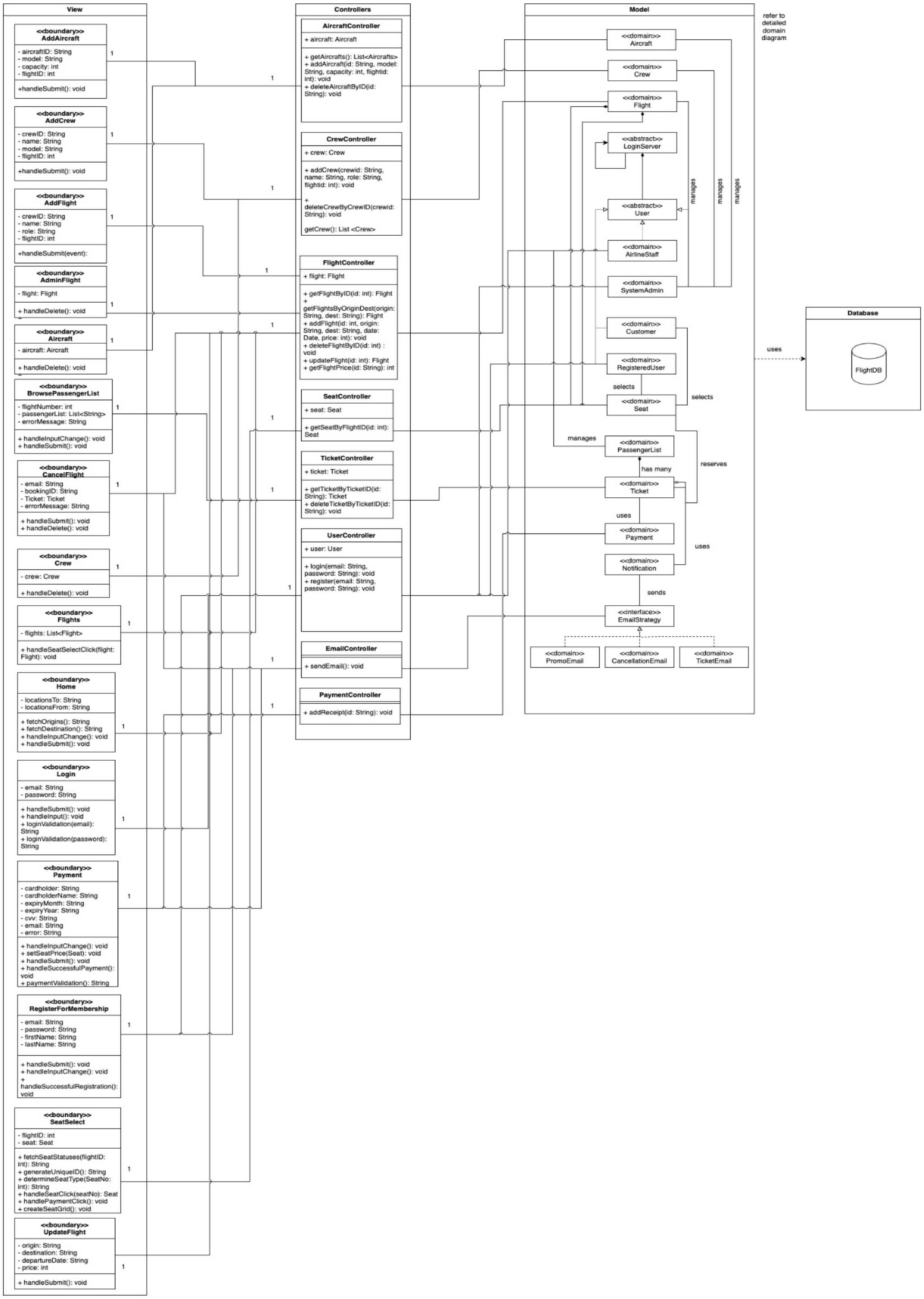
System’s Detailed Domain Class Diagram

with attributes, functionalities and relationships/multiplicities.



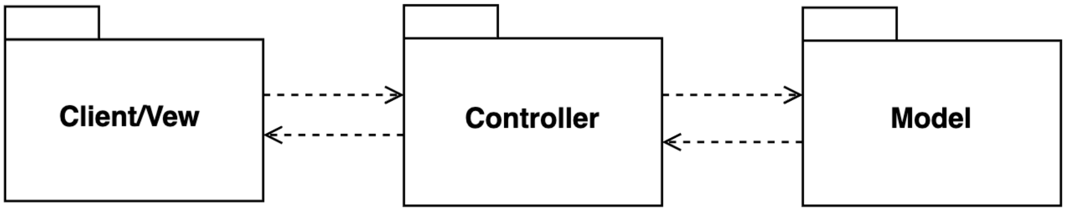
System’s Detailed Design-Class Diagram

Zoomed out full diagram attached to the next page. Please see appendix A for larger size of drawing.

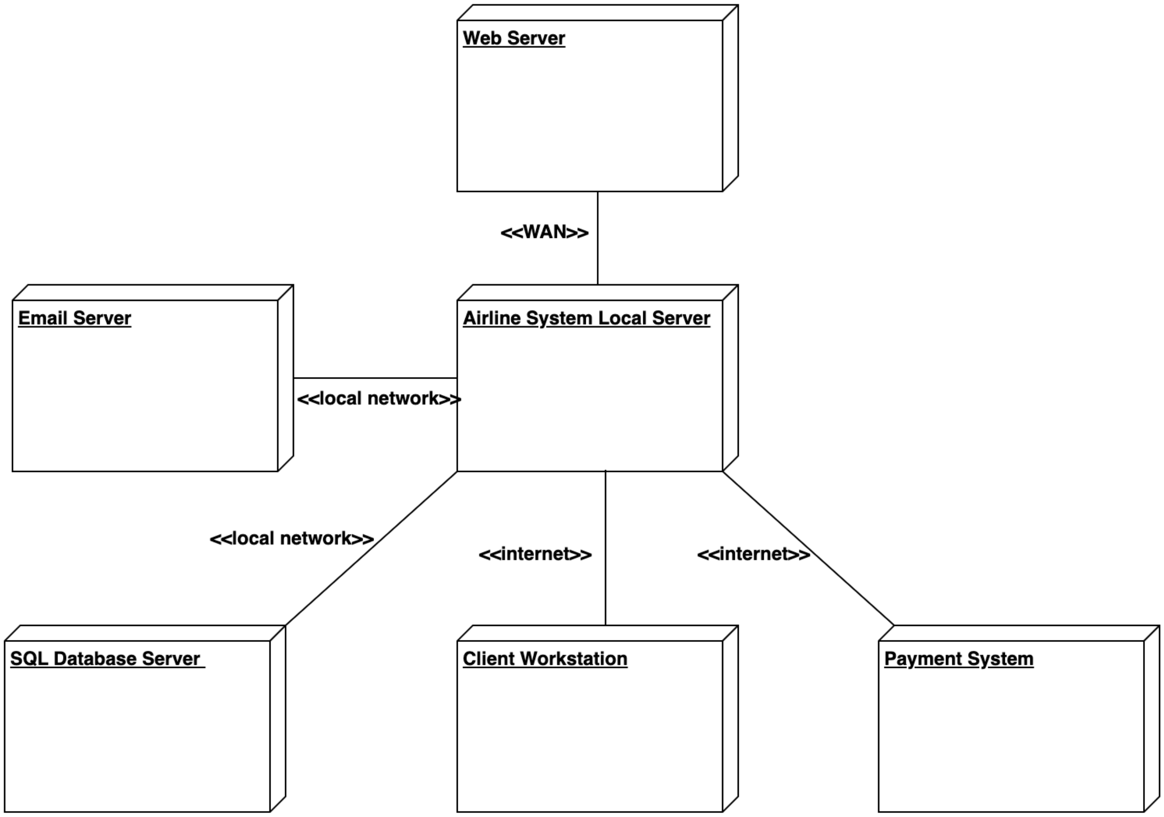


High-Level System's Architecture

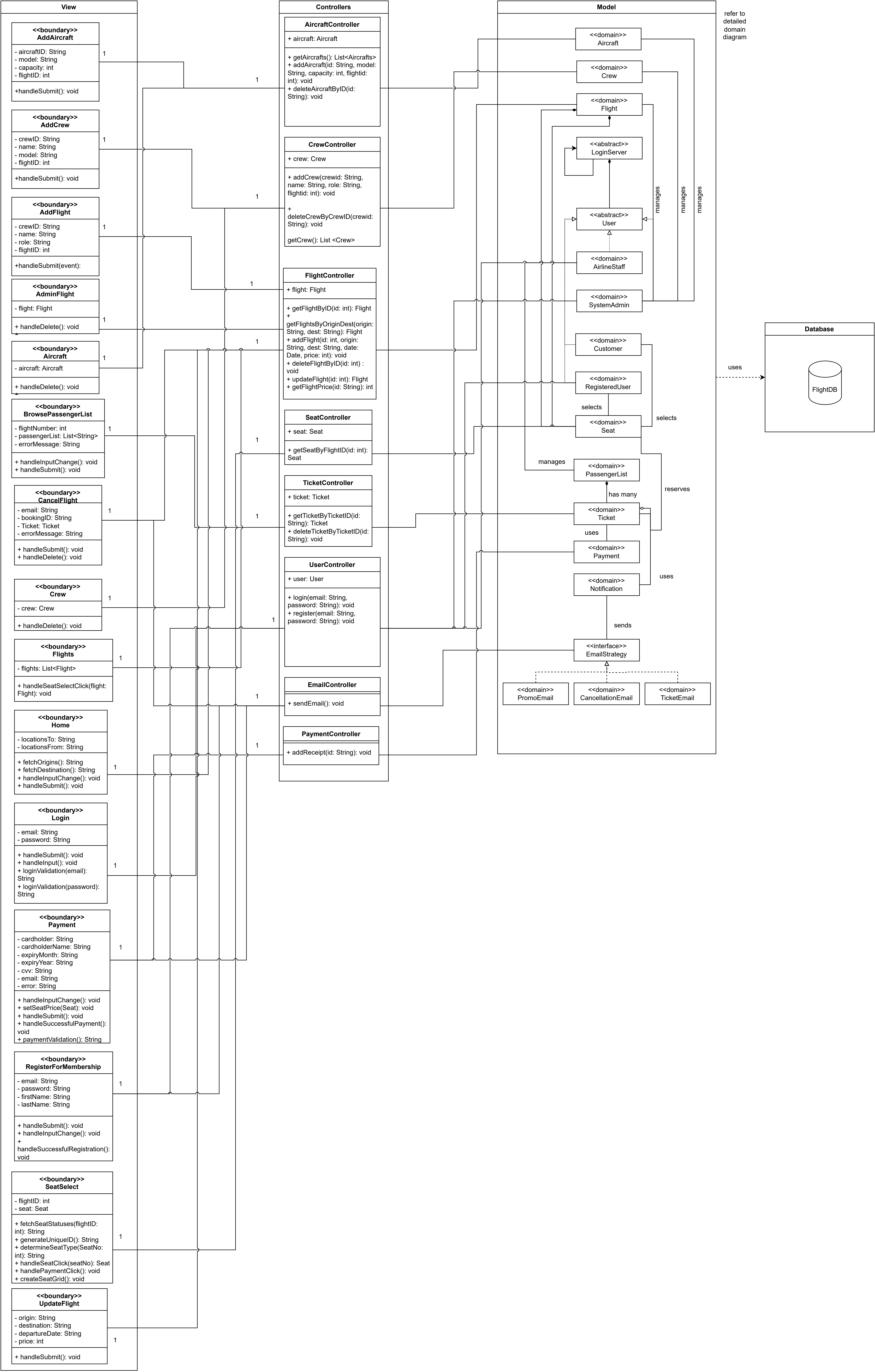
Package Diagram



Deployment Diagram



Appendix


View
<<boundary>>
AddAircraft
- aircraftID: String
- model: String
- capacity: int
- flightID: int
+handleSubmit(): void
<<boundary>>
AddCrew
- crewID: String
- name: String
- model: String
- flightID: int
+handleSubmit(): void
<<boundary>>
AddFlight
- crewID: String
- name: String
- role: String
- flightID: int
+handleSubmit(event):
<<boundary>>
AdminFlight
- flight: Flight
+ handleDelete(): void
<<boundary>>
Aircraft
- aircraft: Aircraft
+ handleDelete(): void
<<boundary>>
BrowsePassengerList
- flightNumber: int
- passengerList: List<String>
- errorMessage: String
+ handleInputChange(): void
+ handleSubmit(): void
<<boundary>>
CancelFlight
- email: String
- bookingID: String
- Ticket: Ticket
- errorMessage: String
+ handleSubmit(): void
+ handleDelete(): void
<<boundary>>
Crew
- crew: Crew
+ handleDelete(): void
<<boundary>>
Flights
- flights: List<Flight>
+ handleSeatSelectClick(flight: Flight): void
<<boundary>>
Home
- locationsTo: String
- locationsFrom: String
+ fetchOrigins(): String
+ fetchDestination(): String
+ handleInputChange(): void
+ handleSubmit(): void
<<boundary>>
Login
- email: String
- password: String
+ handleSubmit(): void
+ handleInput(): void
+ loginValidation(email): String
+ loginValidation(password): String
<<boundary>>
Payment
- cardholder: String
- cardholderName: String
- expiryMonth: String
- expiryYear: String
- cvv: String
- email: String
- error: String
+ handleInputChange(): void
+ setSeatPrice(Seat): void
+ handleSubmit(): void
+ handleSuccessfulPayment(): void
+ paymentValidation(): String
<<boundary>>
RegisterForMembership
- email: String
- password: String
- firstName: String
- lastName: String
+ handleSubmit(): void
+ handleInputChange(): void
+ handleSuccessfulRegistration(): void
<<boundary>>
SeatSelect
- flightID: int
- seat: Seat
+ fetchSeatStatuses(flightID: int): String
+ generateUniqueID(): String
+ determineSeatType(SeatNo: int): String
+ handleSeatClick(seatNo): Seat
+ handlePaymentClick(): void
+ createSeatGrid(): void
<<boundary>>
UpdateFlight
- origin: String
- destination: String
- departureDate: String
- price: int
+ handleSubmit(): void
1
Controllers
AircraftController
+ aircraft: Aircraft
+ getAircrafts(): List<Aircrafts>
+ addAircraft(id: String, model: String, capacity: int, flightid: int): void
+ deleteAircraftByID(id: String): void
CrewController
+ crew: Crew
+ addCrew(crewid: String, name: String, role: String, flightid: int): void
+ deleteCrewByCrewID(crewid: String): void
getCrew(): List <Crew>
FlightController
+ flight: Flight
+ getFlightByID(id: int): Flight
+ getFlightsByOriginDest(origin: String, dest: String): Flight
+ addFlight(id: int, origin: String, dest: String, date: Date, price: int): void
+ deleteFlightByID(id: int) : void
+ updateFlight(id: int): Flight
+ getFlightPrice(id: String): int
SeatController
+ seat: Seat
+ getSeatByFlightID(id: int): Seat
TicketController
+ ticket: Ticket
+ getTicketByTicketID(id: String): Ticket
+ deleteTicketByTicketID(id: String): void
UserController
+ user: User
+ login(email: String, password: String): void
+ register(email: String, password: String): void
EmailController
+ sendEmail(): void
PaymentController
+ addReceipt(id: String): void
Model
refer to detailed domain diagram
<<domain>>
Aircraft
<<domain>>
Crew
<<domain>>
Flight
<<abstract>>
LoginServer
<<abstract>>
User
<<domain>>
AirlineStaff
<<domain>>
SystemAdmin
<<domain>>
Customer
<<domain>>
RegisteredUser
<<domain>>
Seat
<<domain>>
PassengerList
<<domain>>
Ticket
<<domain>>
Payment
<<domain>>
Notification
<<interface>>
EmailStrategy
<<domain>>
PromoEmail
<<domain>>
CancellationEmail
<<domain>>
TicketEmail
manages
manages
manages
manages
selects
selects
reserves
has many
uses
uses
sends
uses
Database
FlightDB