ENSF 614

Term Project:
Design Document

Taran Bains

Design Phase

Part One - System Analysis

1. System Description

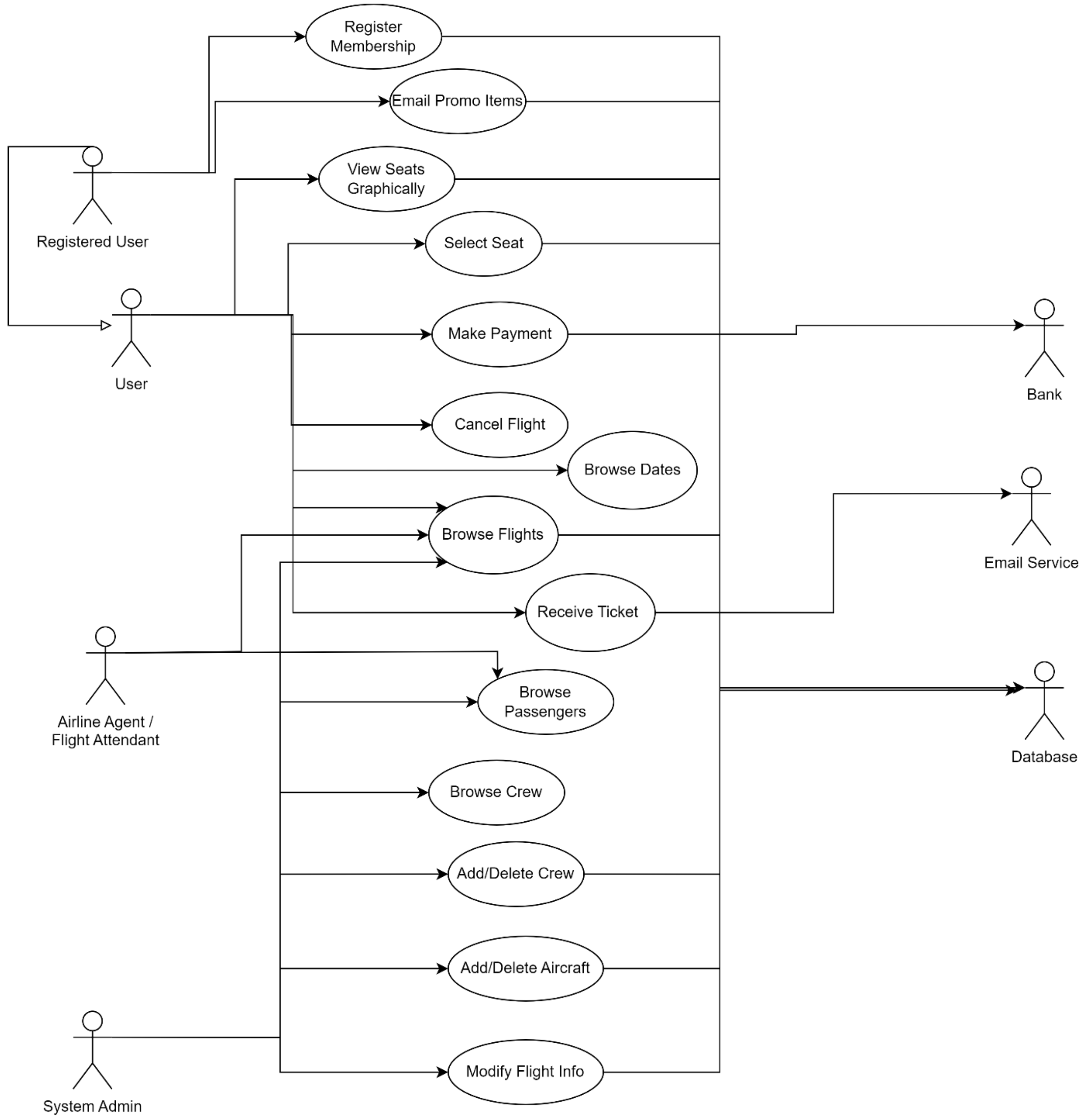
The Airline Web Application system is a tool that allows for facilitating flight reservations, managing booking, and providing essential information about the flights to specific users. One of the main functionalities of the application allows users, such as airline agents and travel agents, to browse flights, purchase tickets, select seats and cancel bookings. The other main functionality is to allow users that are part of the airline company, such as system admins and airline agents/flight attendants, to browse and update specific information about flights, such as flight destinations, crew information, and aircraft information.

There are 3 main user roles are customers, administrators, and crew. Customers are focused on purchasing tickets, crew has read access to browse information available to them and administrators are granted the highest level of permission to change information, such as changing flights and aircrafts. Key components of the system include a MySQL database and Java GUI and backend, that are connected through a JDBC (Java Database Connectivity) that is developed using a Model-View-Controller architectural pattern.

Specific Functionality:

- User registration and authentication
- User profile management
- Flight search
- Booking and reservation
- Payment processing
- Notification system, ie. email

2. System Use-Case Diagram



3. System Scenarios for Use Cases (5 Use Cases, Candidate Object Underlined)

Register as Member:

Jim selects an option to "Become Member" in the Airline Registration App.

The system provides input fields for the following information which Jim provides his information:

- First Name: Jim
- Last Name: Halpert
- Birth Date: 08/15/1991
- Email/Username: jim.halpert@dundermifflin.ca
- Password: DunderMifflin123

Jim selects the option to "Confirm and Register".

The system verifies that the email has not been used by another user and all other input fields are valid.

The system displays a confirmation page that indicates successful registration and allows Jim to continue the application.

Select A Flight:

John has either logged in as a "Registered User" or continued as a "Guest".

The system has now moved John to a search bar where he can search for flights by destination and dates, including "From" and "To".

John types 'Vancouver BC (YVR' as the from destination and 'Los Angeles, USA, (LAX)' as the to destination as well as the field for dates 'Fri, Dec 1 2023' to 'Sun, Dec 10 2023'.

The system moves onto the results page where a flight is shown if found or an error message suggesting to look at different dates for the same location.

John clicks on the Flight that was successfully found.

Graphically View Available Seats:

John has selected a flight, destination and dates.

The system moves onto a seat selection page where rows of seats are shown for the plane that display what is available, and what tier of seat they are: ordinary, comfort or business class.

John selects a window seat in comfort and selects "Confirmed Selection".

The system then moves onto the payment and confirmation page.

Make Payment:

John selects the flight/destination/dates/seat that he would like to purchase.

The system then moves onto an add-on ticket insurance page with the cost. John can either select insurance for his flight or decline and move onto payment.

The payment page comes up where all the ticket information (and optional insurance) is displayed and the input fields for credit card information and personal information such as billing address.

Input Fields for John to Fill Include:

- Billing Address: 907 1st Ave, Calgary S5M 0Y7

- Email: johndoe@gmail.com
- Credit or Debit: Credit
- Name on Card: John Doe
- Card Number: 1122 4499 0923 6598
- Card Expiration Date: 08/2025
- CSV: 098

John enters his information into the fields and selects “Confirm & Pay”.

The system verifies the card and shows the payment confirmation and sends receipt and ticket info to John’s email address.

Change Crew Member:

Jane Doe logs in as a System Administrator with her credentials.

The system has now moved Jane to a search bar where she can search for flights by destination and dates. Including “From” and “To”.

Jane types ‘Vancouver BC (YVR’ as the from destination and ‘Los Angeles, USA, (LAX)’ as the to destination as well as the dates ‘Fri, Dec 1 2023’ to ‘Sun, Dec 10 2023’.

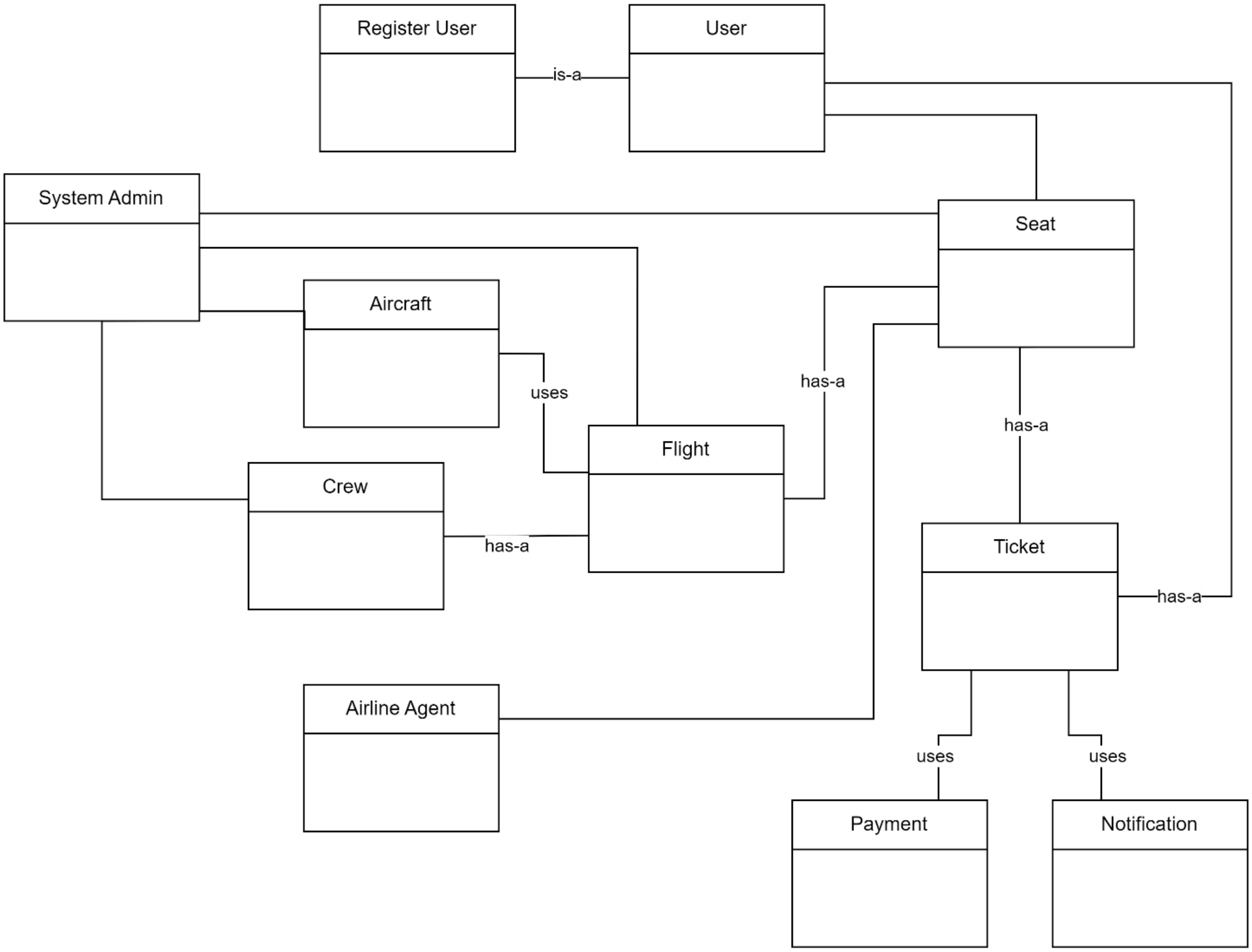
The system moves onto the results page where a flight is shown if found or an error message suggesting to look at different dates for the same location.

Jane selects the flight and the system displays options to see information about the crew, the passengers, the aircraft, and other flight information.

Jane selects the crew tab and the system displays a list of the crew assigned for the flight.

Jane adds Paul White to the crew for that flight.

4. System Conceptual Model



Part Two - Domain Diagrams

1. Highlights of the System Architecture

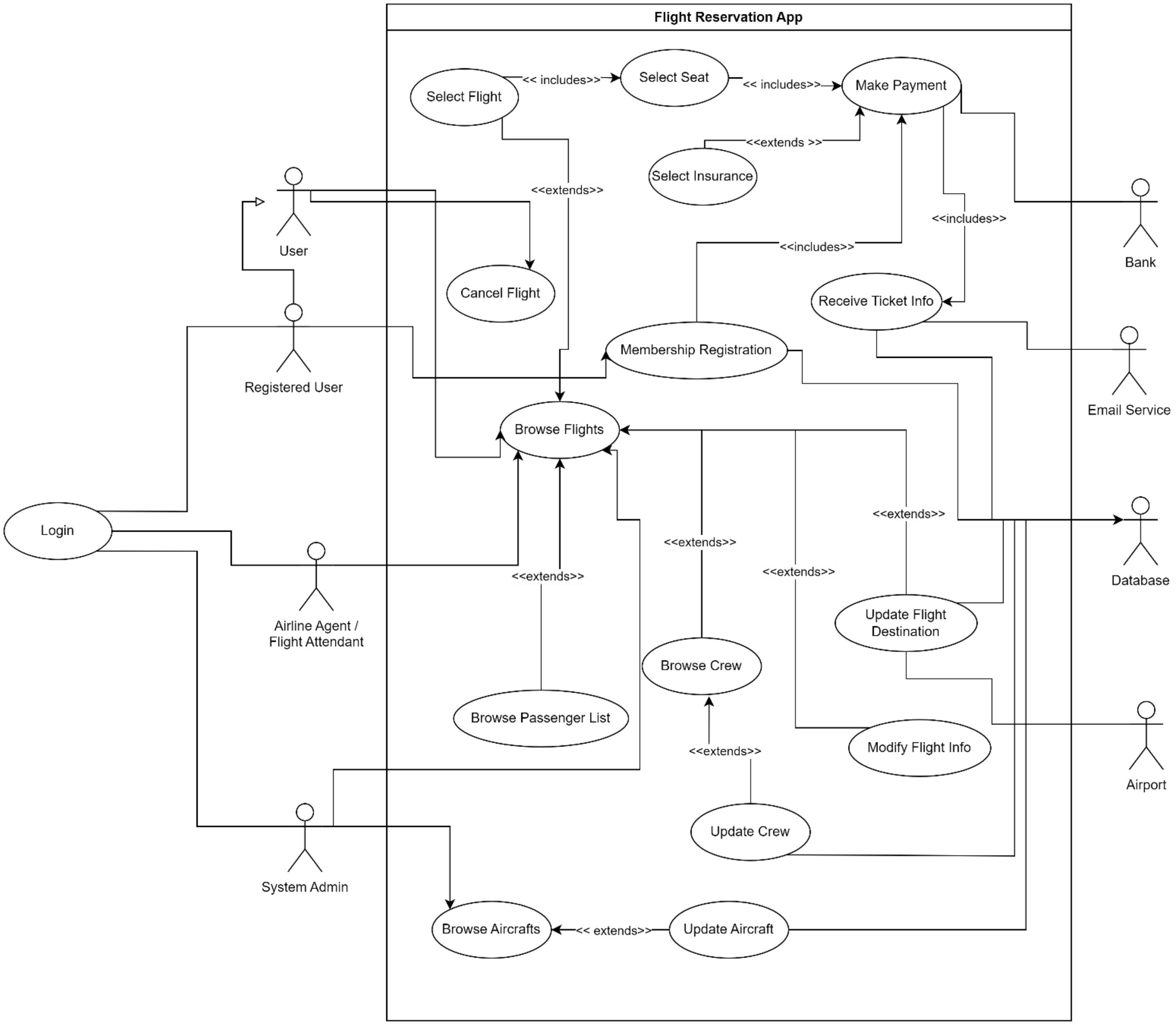
Client-Server Architecture: The system uses a graphical user interface, GUI, developed using Java Swing that handles user interactions including login, browsing and selecting flights, select seats, purchasing a ticket and cancelling a ticket. The system implements a Model-View-Controller design pattern where the user interacts with the View and the Controller interprets the user request and directs the Model to provide the requested information. A Controller also exists for the database to gather and save necessary information.

Database Connectivity: A MySQL database was chosen for the system that stores information such as flight details, aircraft details, user profiles for registered users and bookings or tickets. Java establishes a connection to the MySQL database using JDBC (Java Database Connectivity) and is implemented using a Singleton design pattern to maintain a consistent connection.

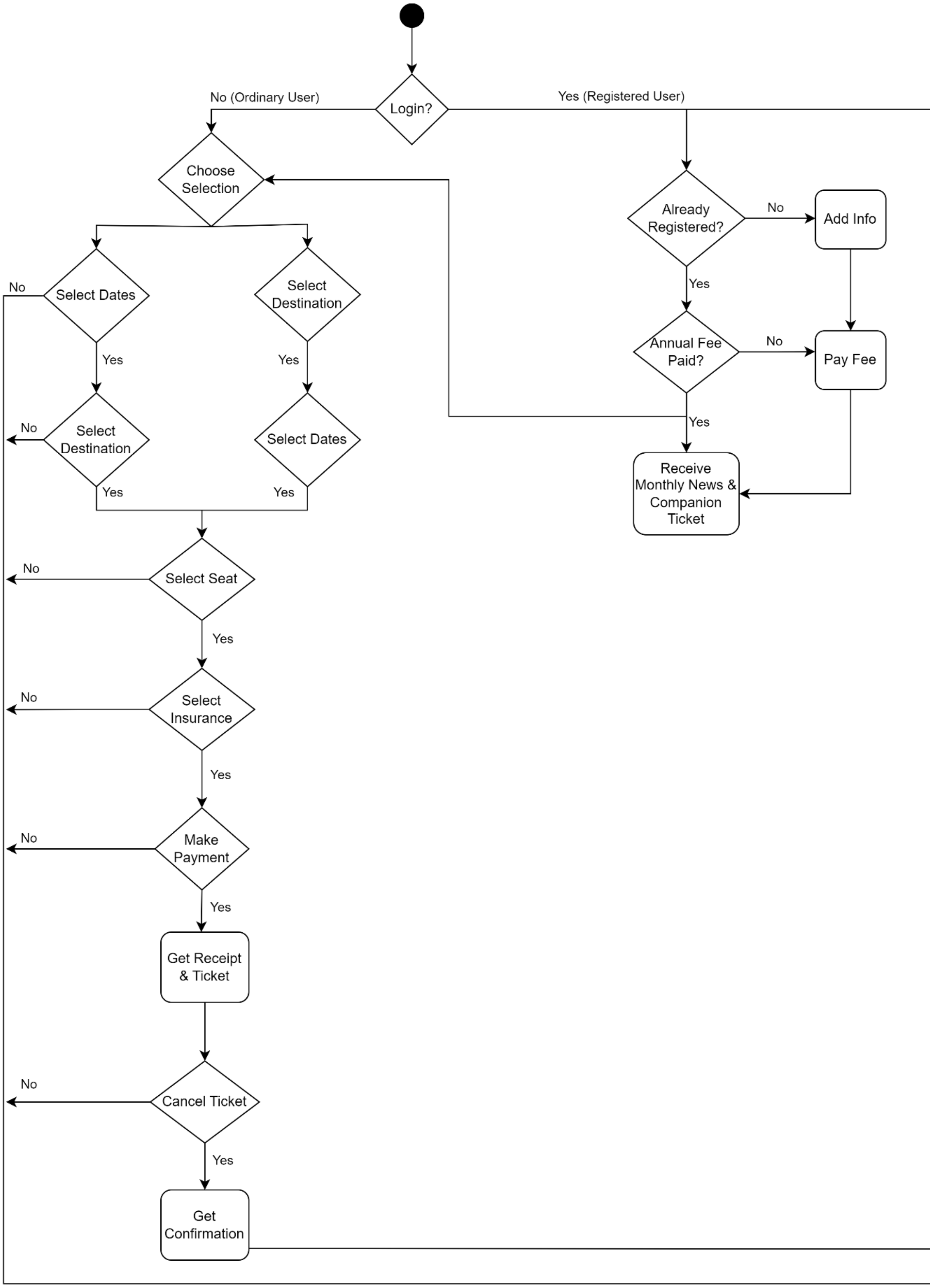
Key Design Patterns:

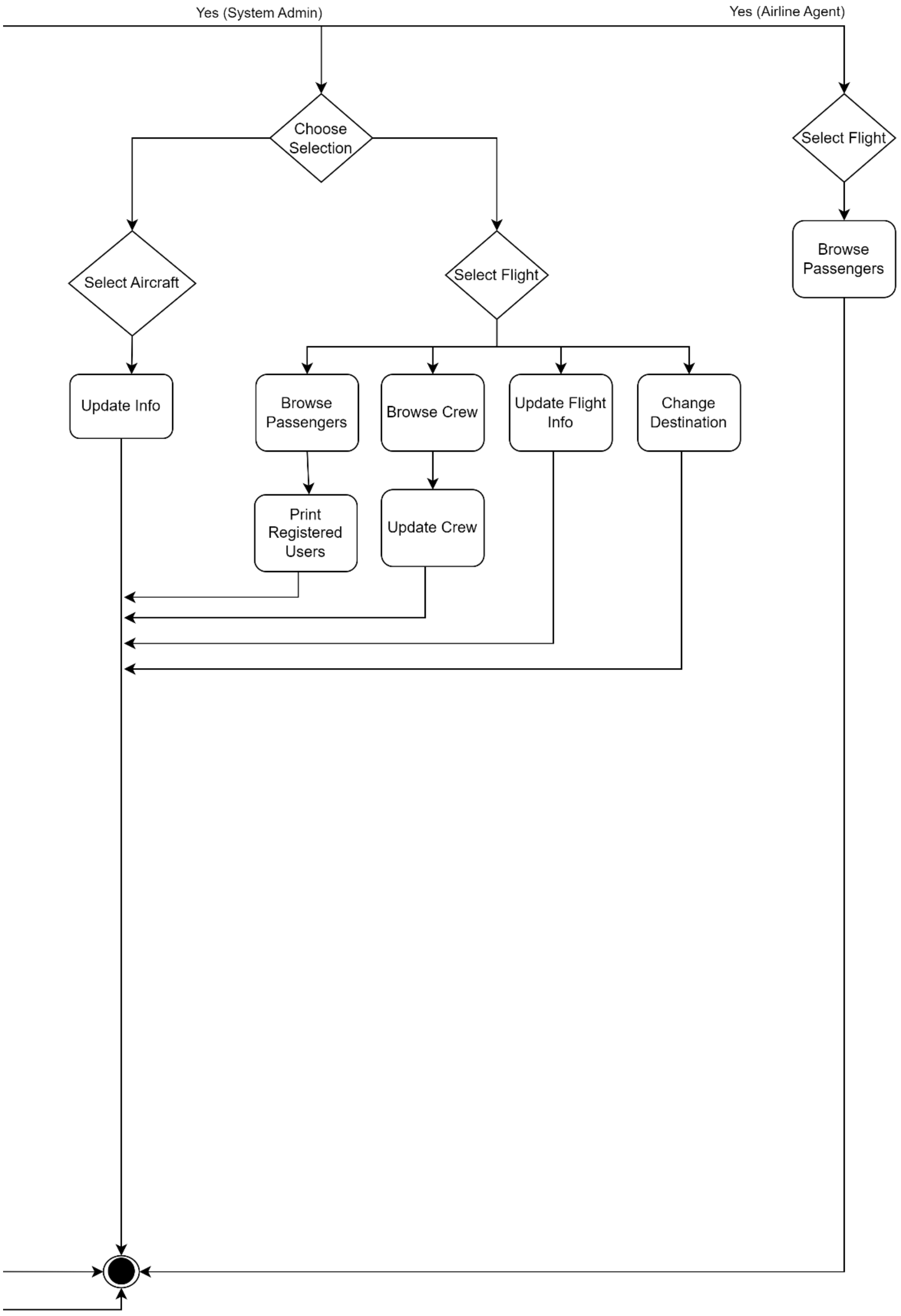
Model-View-Controller: The system follows this architectural pattern.
Singleton: The database connection is implemented in this pattern.
Decorator: The different types of employees follow this pattern. They were not implemented but are shown in the design.

2. Updated Use-Case Diagram



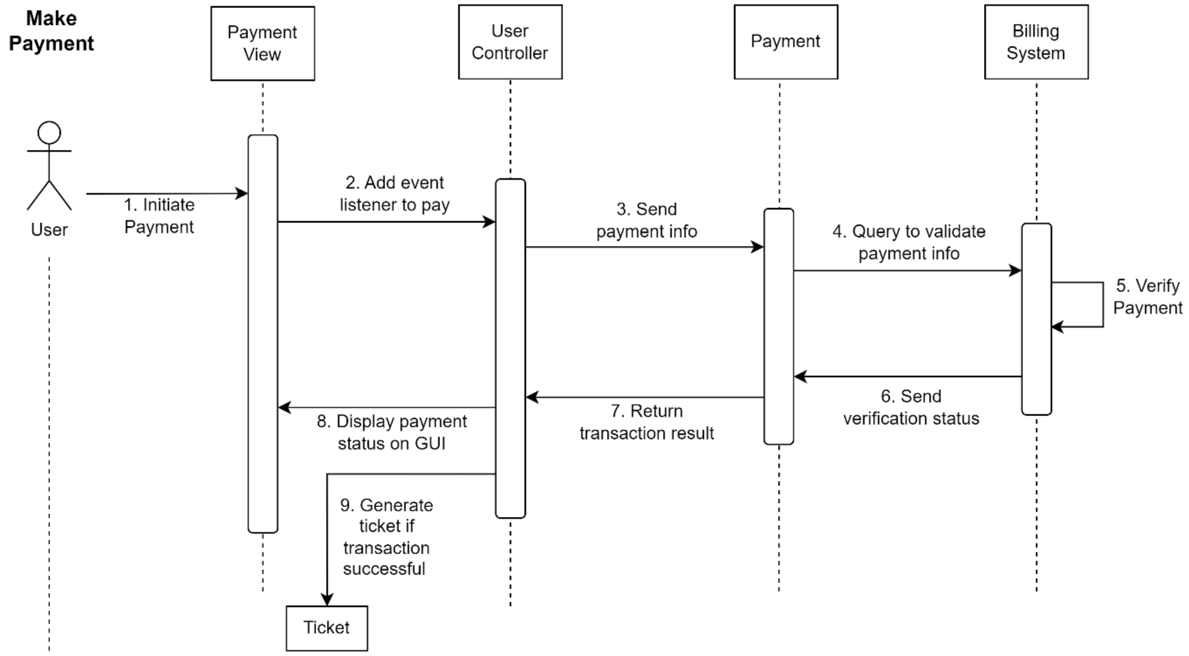
3. Systems Activity Diagram



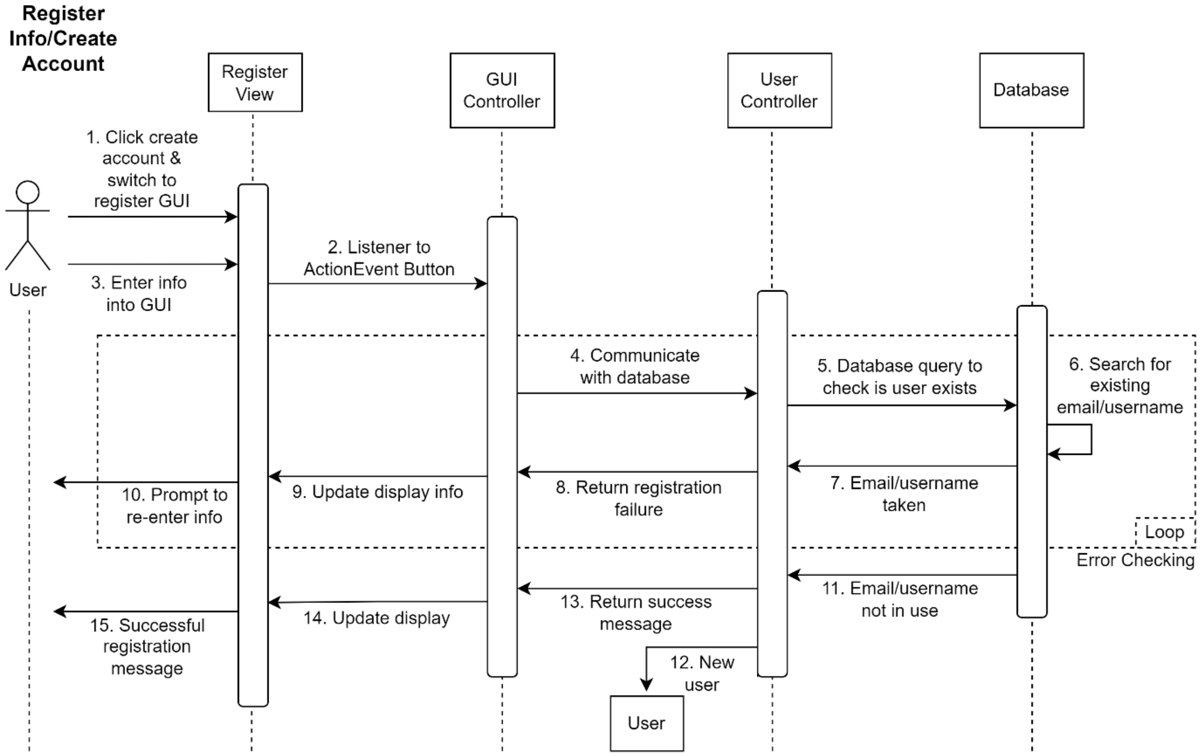

Yes (System Admin)
Yes (Airline Agent)
Choose Selection
Select Flight
Select Aircraft
Select Flight
Browse Passengers
Update Info
Browse Passengers
Browse Crew
Update Flight Info
Change Destination
Print Registered Users
Update Crew

4. Sequence Diagram (3 use cases)

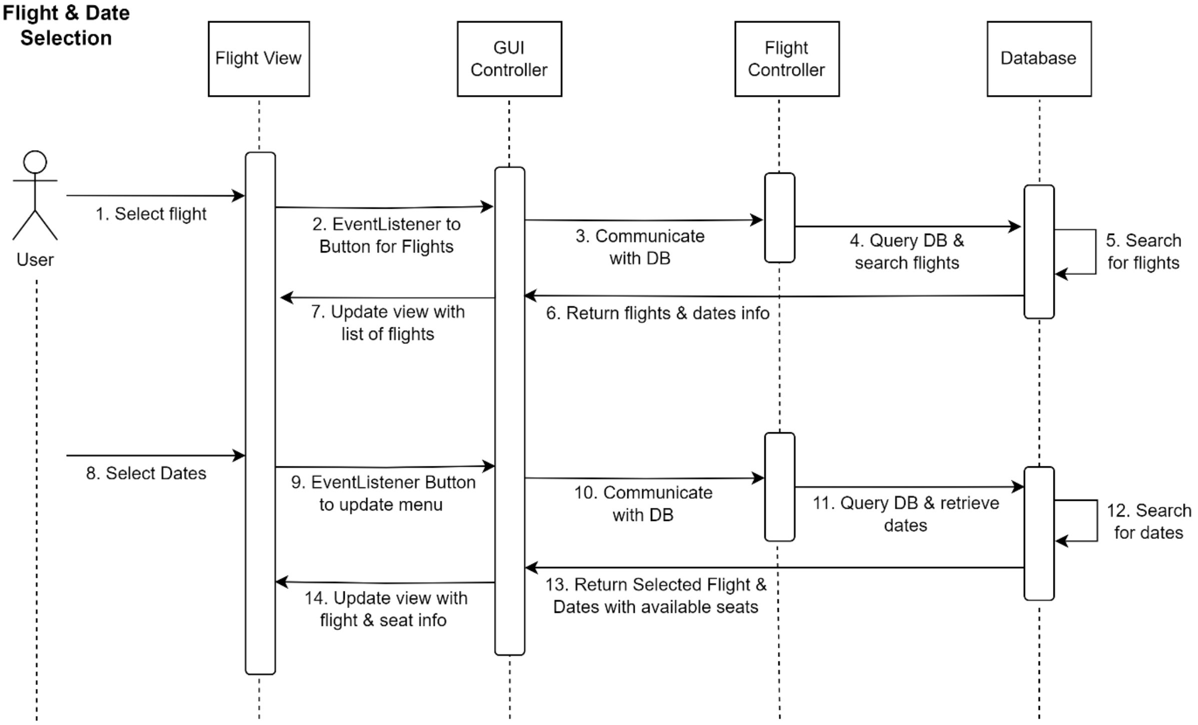
Make Payment


Payment View
User Controller
Payment
Billing System
User
1. Initiate Payment
2. Add event listener to pay
3. Send payment info
4. Query to validate payment info
5. Verify Payment
6. Send verification status
7. Return transaction result
8. Display payment status on GUI
9. Generate ticket if transaction successful
Ticket

Register Info/Create Account

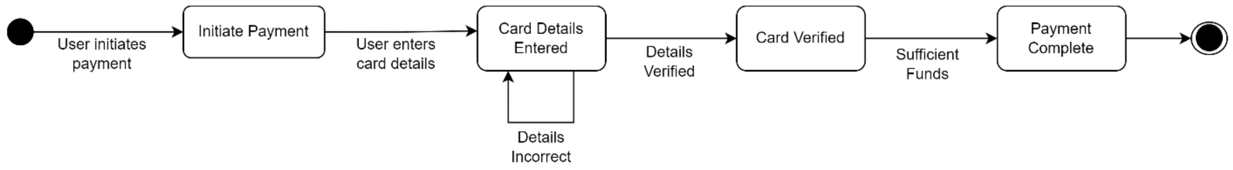

Register View
GUI Controller
User Controller
Database
User
1. Click create account & switch to register GUI
2. Listener to ActionEvent Button
3. Enter info into GUI
4. Communicate with database
5. Database query to check is user exists
6. Search for existing email/username
7. Email/username taken
8. Return registration failure
9. Update display info
10. Prompt to re-enter info
Loop
Error Checking
11. Email/username not in use
12. New user
User
13. Return success message
14. Update display
15. Successful registration message

Flight & Date Selection

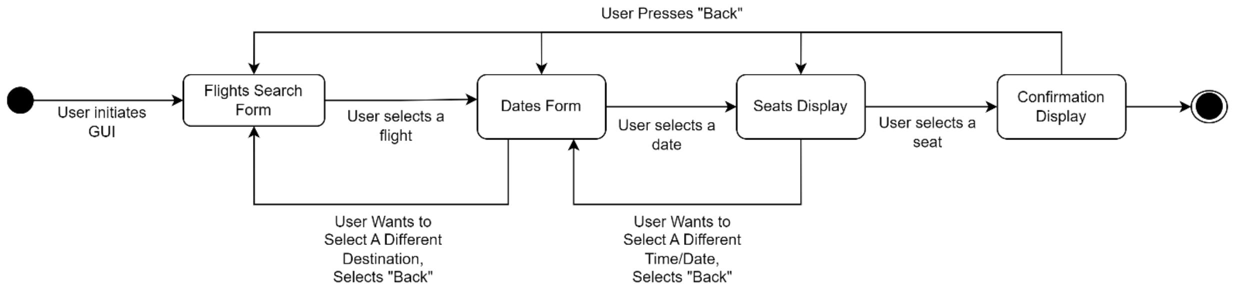

Flight View
GUI Controller
Flight Controller
Database
User
1. Select flight
2. EventListener to Button for Flights
3. Communicate with DB
4. Query DB & search flights
5. Search for flights
6. Return flights & dates info
7. Update view with list of flights
8. Select Dates
9. EventListener Button to update menu
10. Communicate with DB
11. Query DB & retrieve dates
12. Search for dates
13. Return Selected Flight & Dates with available seats
14. Update view with flight & seat info

5. State Transition Diagram (2 State Diagrams)

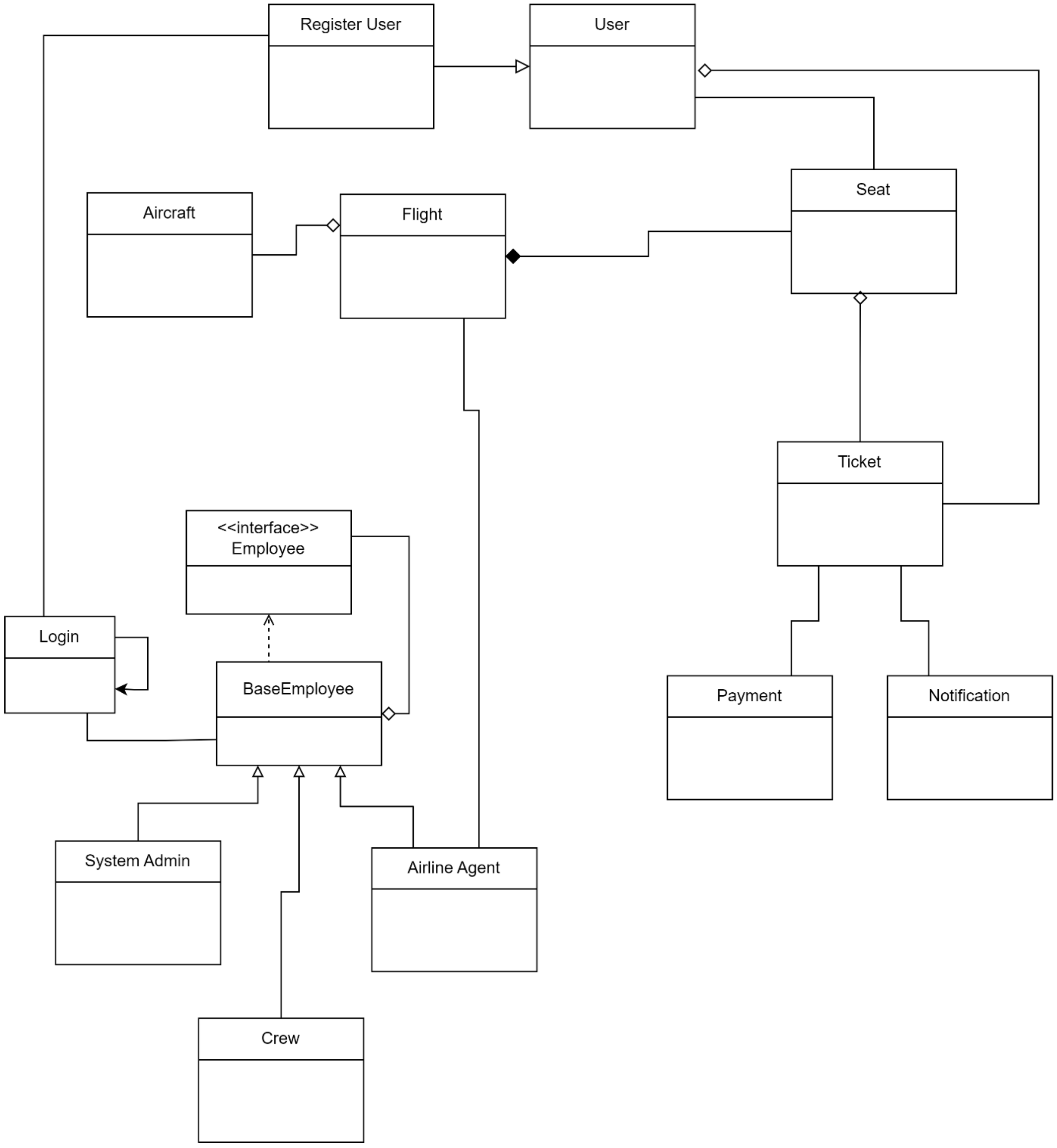
Make Payment



Select Flight

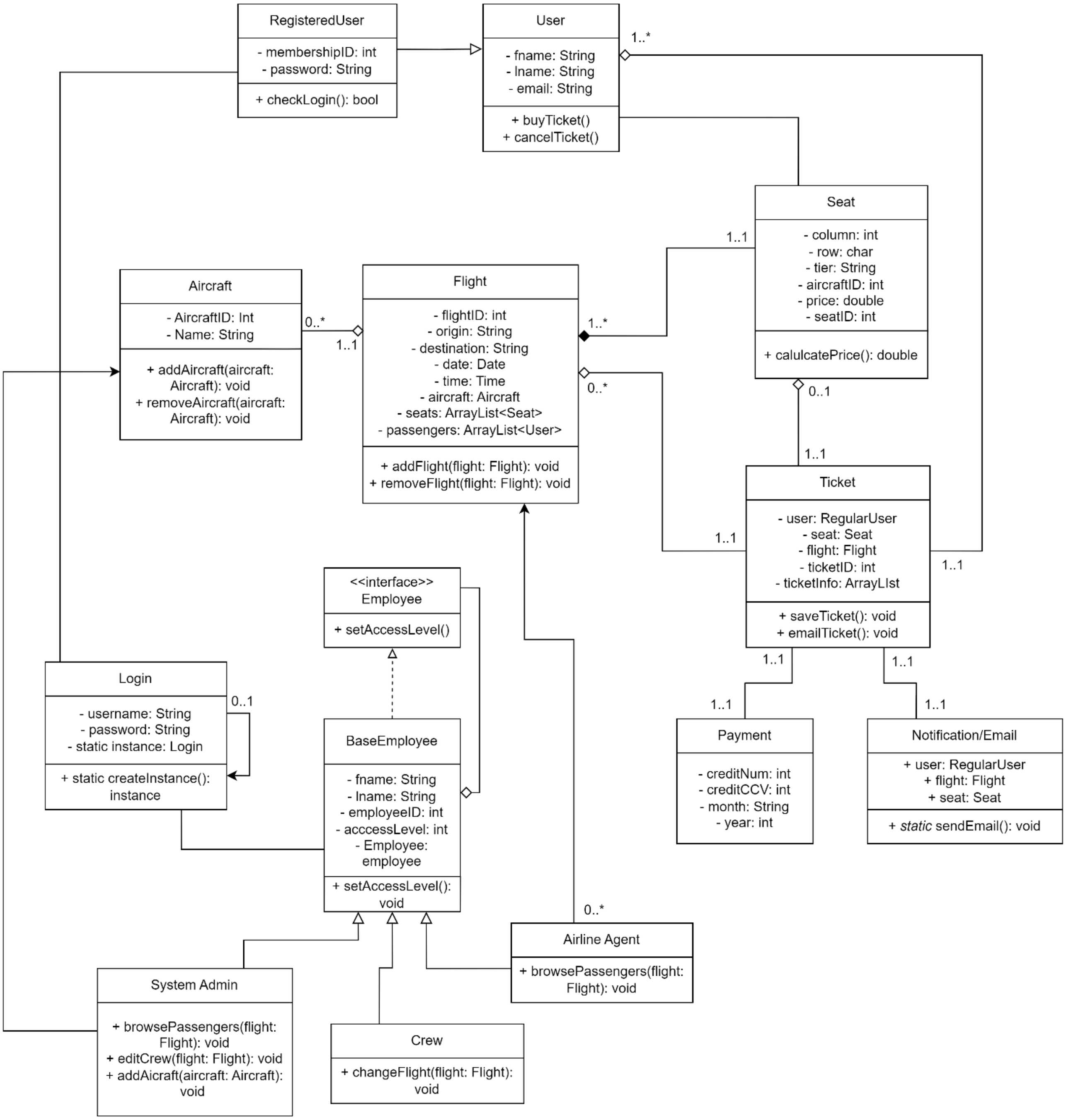


6. Systems Domain Class Diagram (Without Attributes & Functionalities)

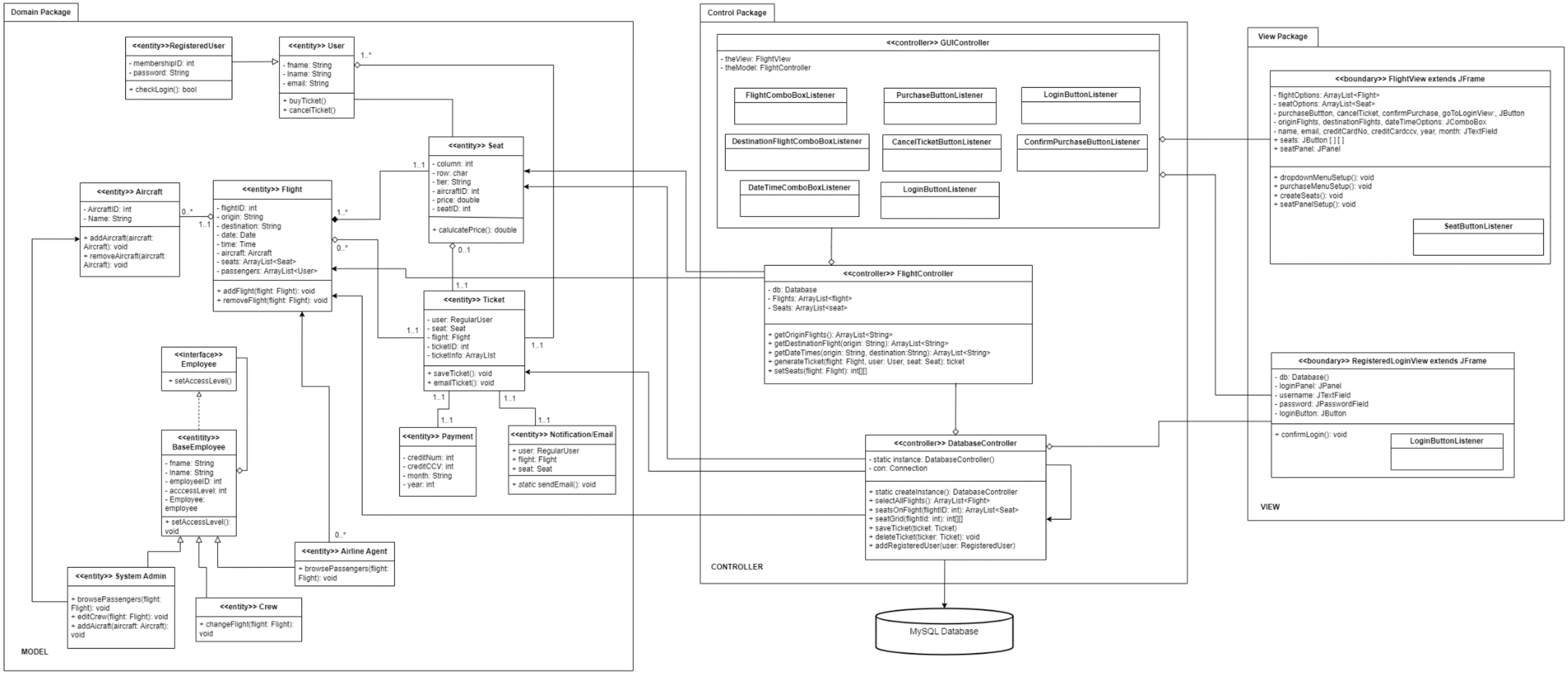


Login implements Singleton design pattern.
Employees implement a Decorator design pattern.

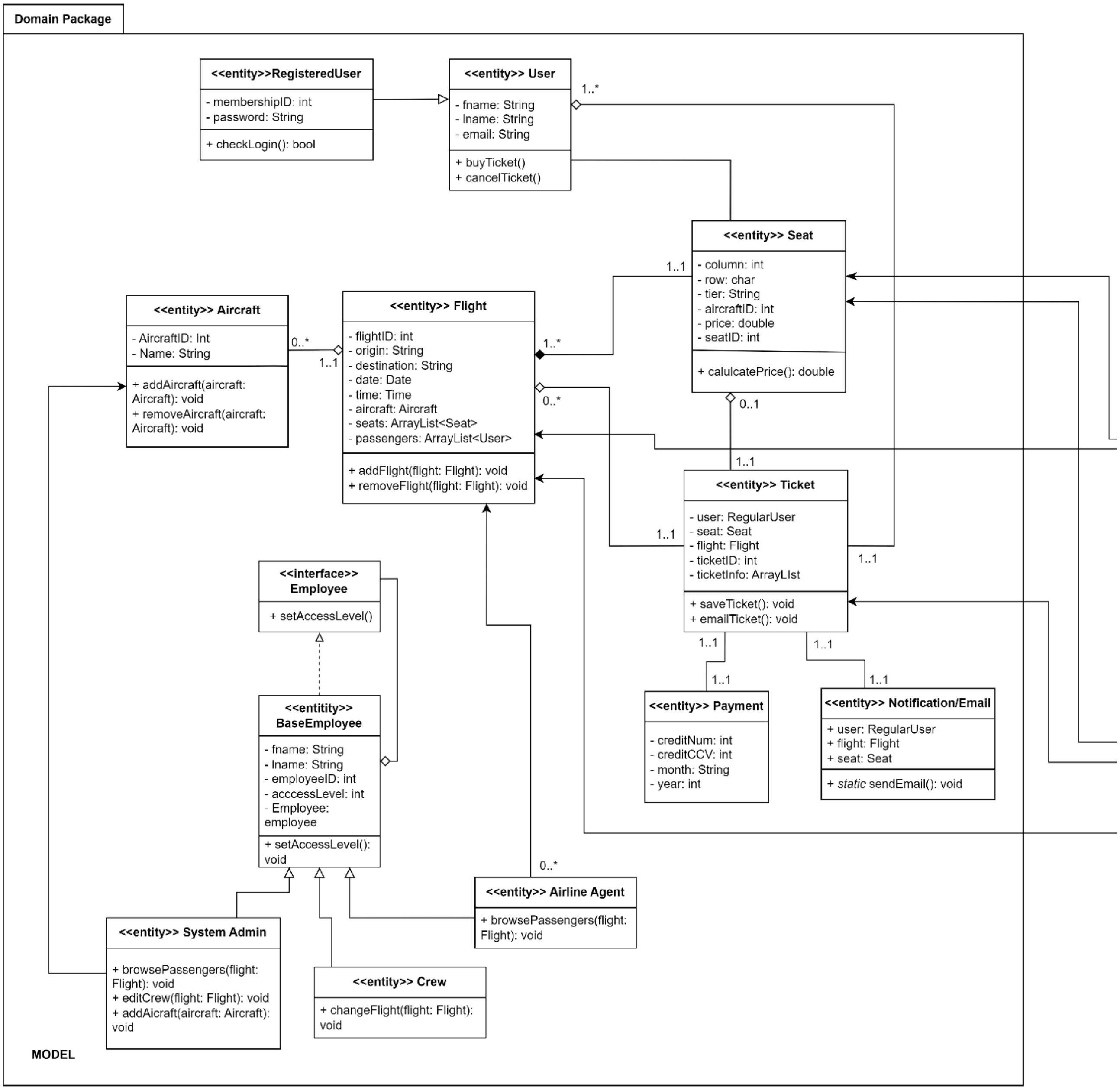
7. Systems Domain Class Diagram (With Attributes & Functionalities)



Part Three - System Detailed Design Class Diagram

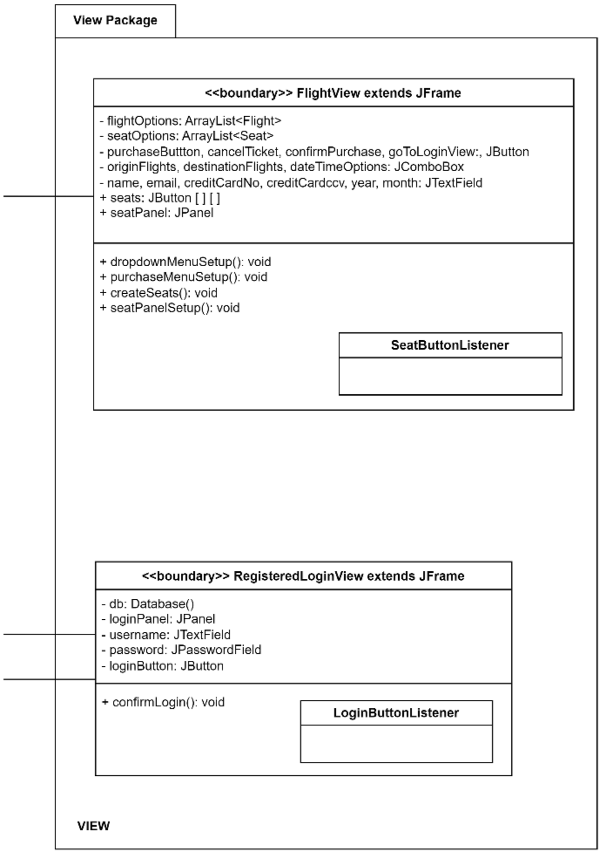

Domain Package
<<entity>>RegisteredUser
- membershipID: int
- password: String
+ checkLogin(): bool
<<entity>> User
- fname: String
- lname: String
- email: String
+ buyTicket()
+ cancelTicket()
1..*
<<entity>> Seat
- column: int
- row: char
- tier: String
- aircraftID: int
- price: double
- seatID: int
+ calulcatePrice(): double
1..1
0..1
<<entity>> Aircraft
- AircraftID: Int
- Name: String
+ addAircraft(aircraft: Aircraft): void
+ removeAircraft(aircraft: Aircraft): void
0..*
1..1
<<entity>> Flight
- flightID: int
- origin: String
- destination: String
- date: Date
- time: Time
- aircraft: Aircraft
- seats: ArrayList<Seat>
- passengers: ArrayList<User>
+ addFlight(flight: Flight): void
+ removeFlight(flight: Flight): void
1..*
0..*
1..1
<<entity>> Ticket
- user: RegularUser
- seat: Seat
- flight: Flight
- ticketID: int
- ticketInfo: ArrayList
+ saveTicket(): void
+ emailTicket(): void
1..1
1..1
1..1
1..1
1..1
<<entity>> Payment
- creditNum: int
- creditCCV: int
- month: String
- year: int
<<entity>> Notification/Email
+ user: RegularUser
+ flight: Flight
+ seat: Seat
+ static sendEmail(): void
<<interface>> Employee
+ setAccessLevel()
<<entitity>> BaseEmployee
- fname: String
- lname: String
- employeeID: int
- acccessLevel: int
- Employee: employee
+ setAccessLevel(): void
0..*
<<entity>> Airline Agent
+ browsePassengers(flight: Flight): void
<<entity>> System Admin
+ browsePassengers(flight: Flight): void
+ editCrew(flight: Flight): void
+ addAicraft(aircraft: Aircraft): void
<<entity>> Crew
+ changeFlight(flight: Flight): void
MODEL
Control Package
<<controller>> GUIController
- theView: FlightView
- theModel: FlightController
FlightComboBoxListener
PurchaseButtonListener
LoginButtonListener
DestinationFlightComboBoxListener
CancelTicketButtonListener
ConfirmPurchaseButtonListener
DateTimeComboBoxListener
LoginButtonListener
<<controller>> FlightController
- db: Database
- Flights: ArrayList<flight>
- Seats: ArrayList<seat>
+ getOriginFlights(): ArrayList<String>
+ getDestinationFlight(origin: String): ArrayList<String>
+ getDateTimes(origin: String, destination:String): ArrayList<String>
+ generateTicket(flight: Flight, user: User, seat: Seat): ticket
+ setSeats(flight: Flight): int[][]
<<controller>> DatabaseController
- static instance: DatabaseController()
- con: Connection
+ static createInstance(): DatabaseController
+ selectAllFlights(): ArrayList<Flight>
+ seatsOnFlight(flightID: int): ArrayList<Seat>
+ seatGrid(flightId: int): int[][]
+ saveTicket(ticket: Ticket)
+ deleteTicket(ticker: Ticket): void
+ addRegisteredUser(user: RegisteredUser)
CONTROLLER
MySQL Database
View Package
<<boundary>> FlightView extends JFrame
- flightOptions: ArrayList<Flight>
- seatOptions: ArrayList<Seat>
- purchaseButtton, cancelTicket, confirmPurchase, goToLoginView:, JButton
- originFlights, destinationFlights, dateTimeOptions: JComboBox
- name, email, creditCardNo, creditCardccv, year, month: JTextField
+ seats: JButton [ ] [ ]
+ seatPanel: JPanel
+ dropdownMenuSetup(): void
+ purchaseMenuSetup(): void
+ createSeats(): void
+ seatPanelSetup(): void
SeatButtonListener
<<boundary>> RegisteredLoginView extends JFrame
- db: Database()
- loginPanel: JPanel
- username: JTextField
- password: JPasswordField
- loginButton: JButton
+ confirmLogin(): void
LoginButtonListener
VIEW

1. Domain Classes

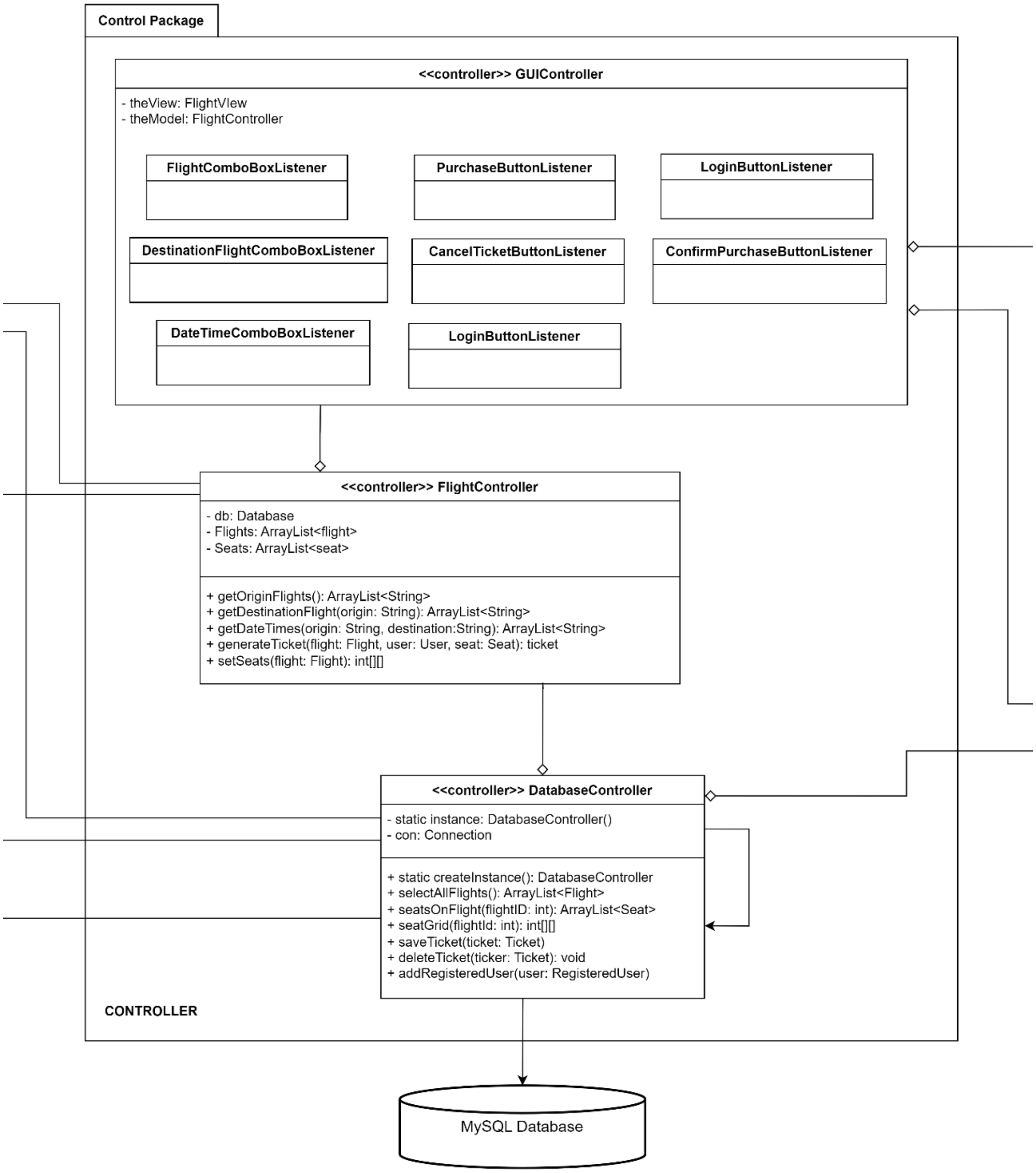


2. Boundary Classes

I am interpreting Boundary classes to be my view classes in my System.

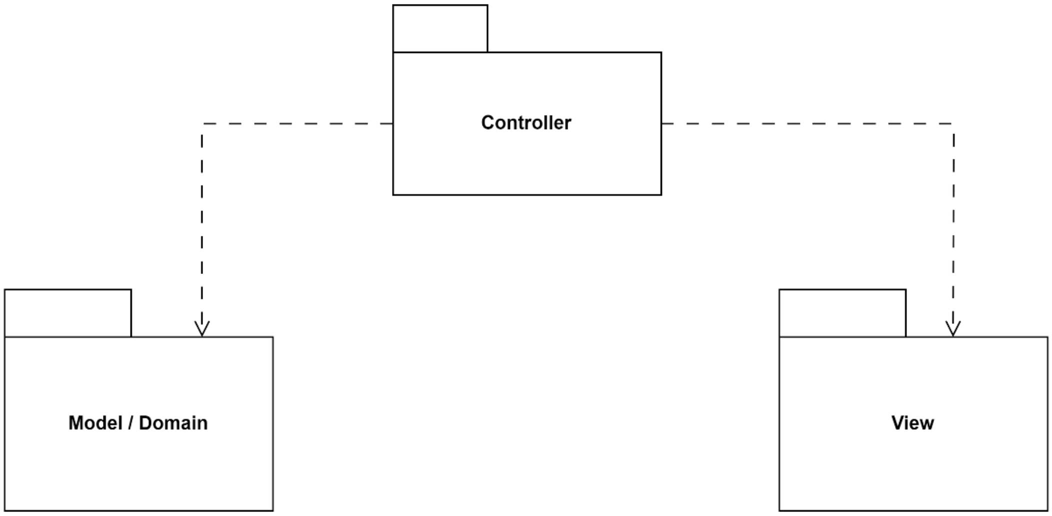

View Package
<<boundary>> FlightView extends JFrame
- flightOptions: ArrayList<Flight>
- seatOptions: ArrayList<Seat>
- purchaseButtton, cancelTicket, confirmPurchase, goToLoginView:, JButton
- originFlights, destinationFlights, dateTimeOptions: JComboBox
- name, email, creditCardNo, creditCardccv, year, month: JTextField
+ seats: JButton [ ] [ ]
+ seatPanel: JPanel
+ dropdownMenuSetup(): void
+ purchaseMenuSetup(): void
+ createSeats(): void
+ seatPanelSetup(): void
SeatButtonListener
<<boundary>> RegisteredLoginView extends JFrame
- db: Database()
- loginPanel: JPanel
- username: JTextField
- password: JPasswordField
- loginButton: JButton
+ confirmLogin(): void
LoginButtonListener
VIEW

3. Controller Classes

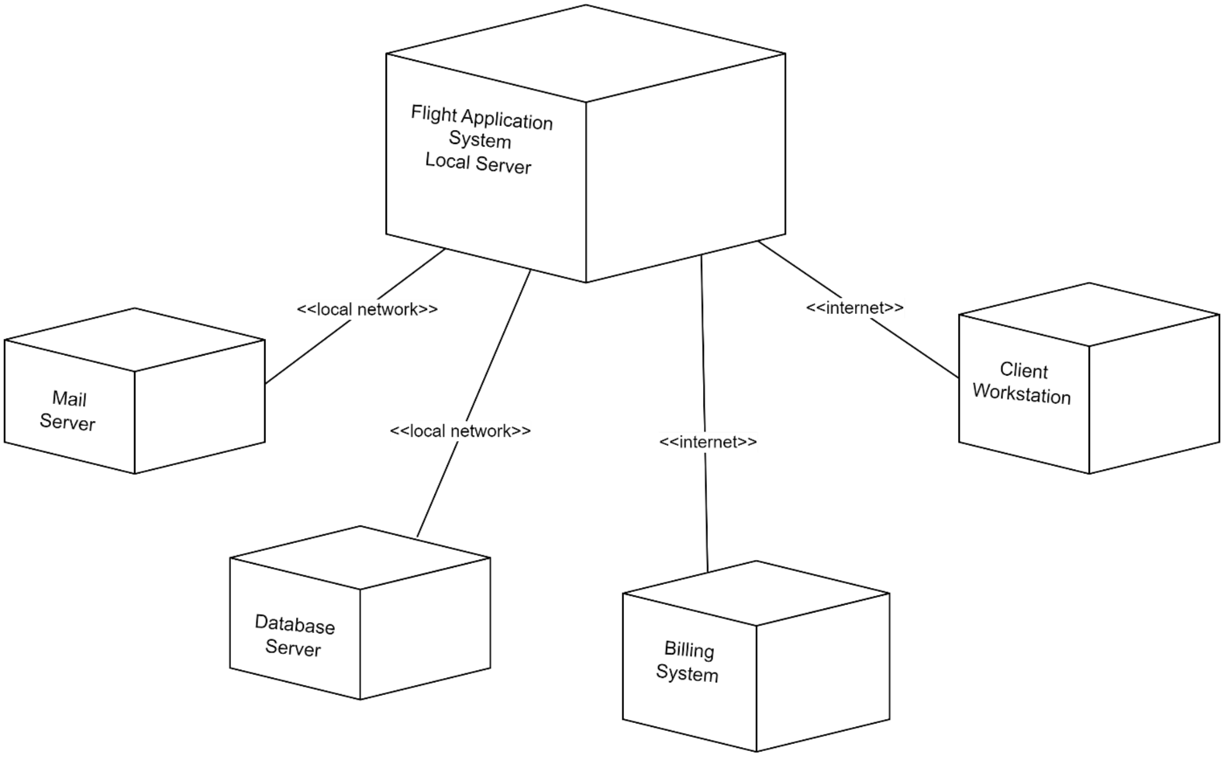

Control Package
<<controller>> GUIController
- theView: FlightView
- theModel: FlightController
FlightComboBoxListener
PurchaseButtonListener
LoginButtonListener
DestinationFlightComboBoxListener
CancelTicketButtonListener
ConfirmPurchaseButtonListener
DateTimeComboBoxListener
LoginButtonListener
<<controller>> FlightController
- db: Database
- Flights: ArrayList<flight>
- Seats: ArrayList<seat>
+ getOriginFlights(): ArrayList<String>
+ getDestinationFlight(origin: String): ArrayList<String>
+ getDateTimes(origin: String, destination:String): ArrayList<String>
+ generateTicket(flight: Flight, user: User, seat: Seat): ticket
+ setSeats(flight: Flight): int[][]
<<controller>> DatabaseController
- static instance: DatabaseController()
- con: Connection
+ static createInstance(): DatabaseController
+ selectAllFlights(): ArrayList<Flight>
+ seatsOnFlight(flightID: int): ArrayList<Seat>
+ seatGrid(flightId: int): int[][]
+ saveTicket(ticket: Ticket)
+ deleteTicket(ticker: Ticket): void
+ addRegisteredUser(user: RegisteredUser)
CONTROLLER
MySQL Database

Part Four – High Level Systems Architecture

1. Package Diagram



2. Deployment Diagram


Flight Application System Local Server
<<local network>>
<<local network>>
<<internet>>
<<internet>>
Mail Server
Database Server
Billing System
Client Workstation