ENSF 480 Term Project – Design

Flight Reservation Application
Use Case Model

Section L02 - Group 16

- Ebitimi Dambo, **30207054**
- Aadil Bashir, **30213437**
- Arol Nokam Wafo, **30174353**
- Erin Kim, **30211474**

System Introduction

In this project, we were tasked with developing a flight reservation application using the skills we acquired throughout the semester. At the beginning, our group was torn between two design approaches: creating a highly realistic reservation system supporting multiple airlines, or building a moderately realistic system intended for a startup airline that manages only its own flights. We evaluated both options and recognized that neither administrators nor agents would have the ability to manage third-party airlines or add new ones. As a result, we chose to design a reservation system focused on a single airline, modeled after the functionality of real-world airline platforms. We drew inspiration from websites such as KLM and Qatar Airways to shape the look and behavior of our application.

Our system begins with a login page where the user is prompted to enter their credentials. If the user is not yet registered, they have the option to create an account and become a customer. Registered users can simply enter their credentials and proceed. After logging in, the interface that appears depends on the user's role, which is stored in the database. This role is queried, and one of three user interfaces is displayed: the Customer UI, Agent UI, or Admin UI. Each of these interfaces offers different functionalities based on the permissions associated with the user's role, as illustrated in our use case diagram.

The next section of our report presents the activity diagram describing the logic flow of customer functionality, so I will briefly outline what happens when a customer logs in. The customer is greeted with a welcome screen that includes navigation options such as "View Reservations" and "Promotion News" in the header, as well as a flight search panel at the center of the screen. To make a booking, the customer enters their trip details into the search panel, which returns a list of matching flights. They can then select their flights and fill out the booking form. After completing the form, the customer proceeds to the payment step, and the reservation is created upon successful payment.

Once a reservation is made, the customer can view their bookings, generate a booking confirmation, and even cancel an existing reservation if necessary

With this overview in mind, we hope that the diagrams presented in the following sections will be easier to understand. Thank you.

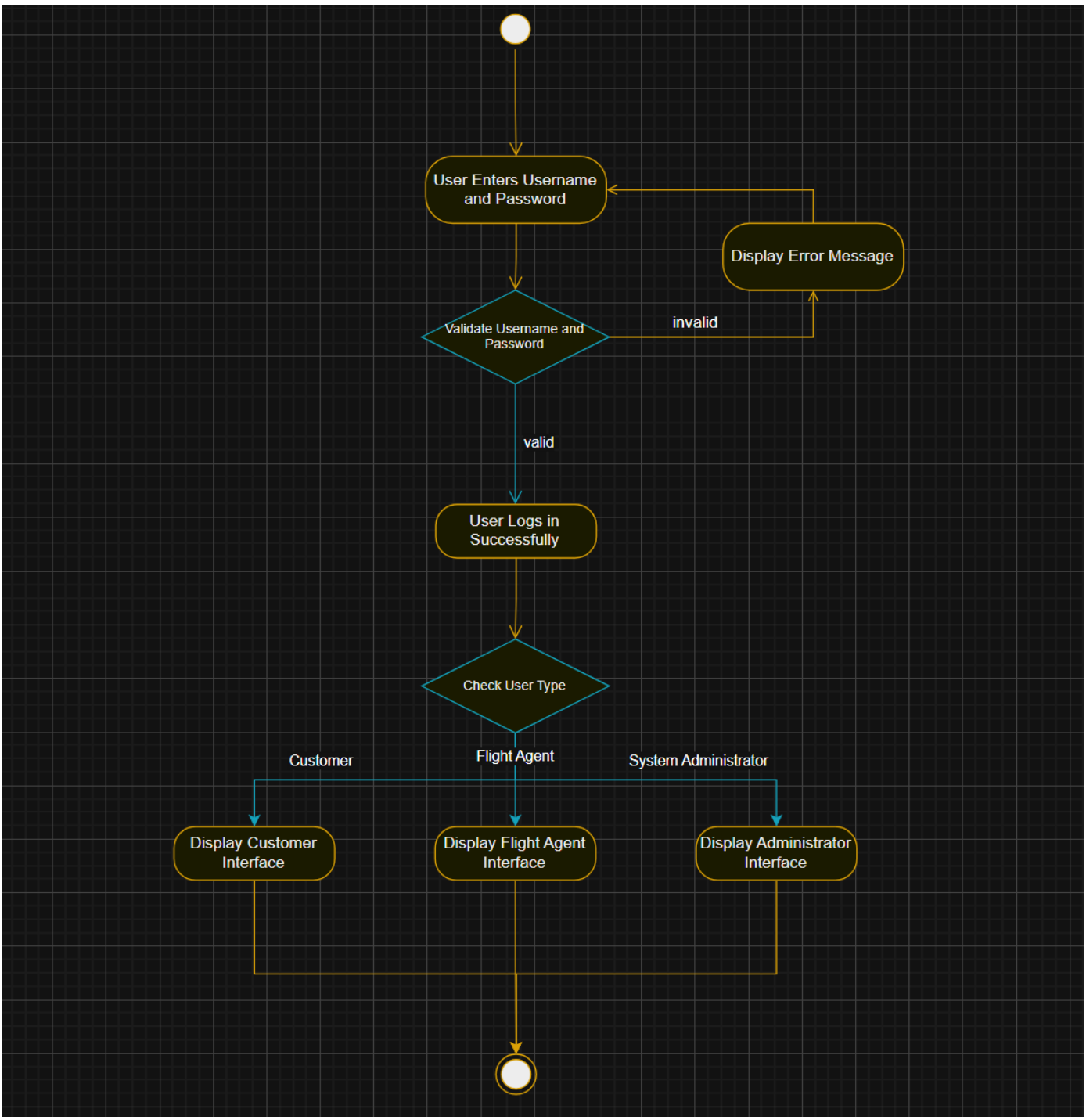
To run the system, you must meet the following requirements:

- Have jdk-25 or later installed
- Ensure the image files included in the zip file are in the right place (same directory as the src folder).
- Have MySQL and MySQL workbench installed
- Run attached SQL script in the workbench to get the required database for our system (data can be updated as you please, but column names must stay the same)
- Add mysql-connector-j-9.5.0.jar to java workspace and build path.

Activity Diagram that shows the Process Login

When the user starts up the application, they are met with a screen prompting them for a username and password. Once the username and password are entered, the system checks if they correspond to an existing user. If the details are invalid, an error message is displayed, and the user is prompted to enter their username and password again.

If the details are valid, the user is met with a success message, and the system checks the role associated with the user's credentials. The system then displays the interface that corresponds to the role of the user.

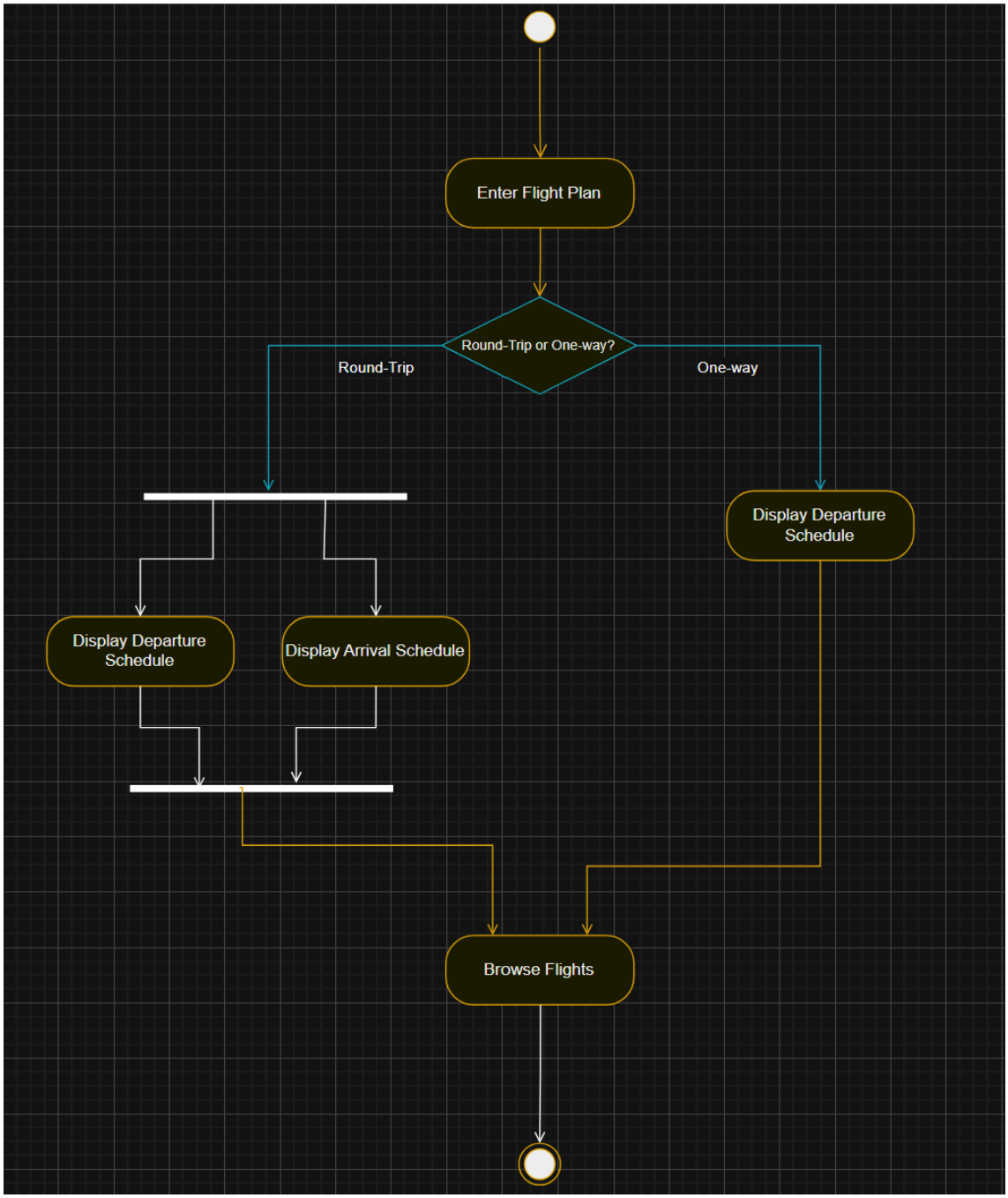


Activity Diagram that shows the Process of Browsing Flights

Once the customer has logged in, they are presented with an interface prompting them to enter their flight plan (trip details such as departure date, destination, and trip type). After submitting this information by clicking “Search Flights,” the system responds according to the type of trip selected.

If the plan is a round trip, the system retrieves both the departure and arrival schedules concurrently from the database and prepares them for display. If it is a one-way trip, only the departure schedule is retrieved and displayed.

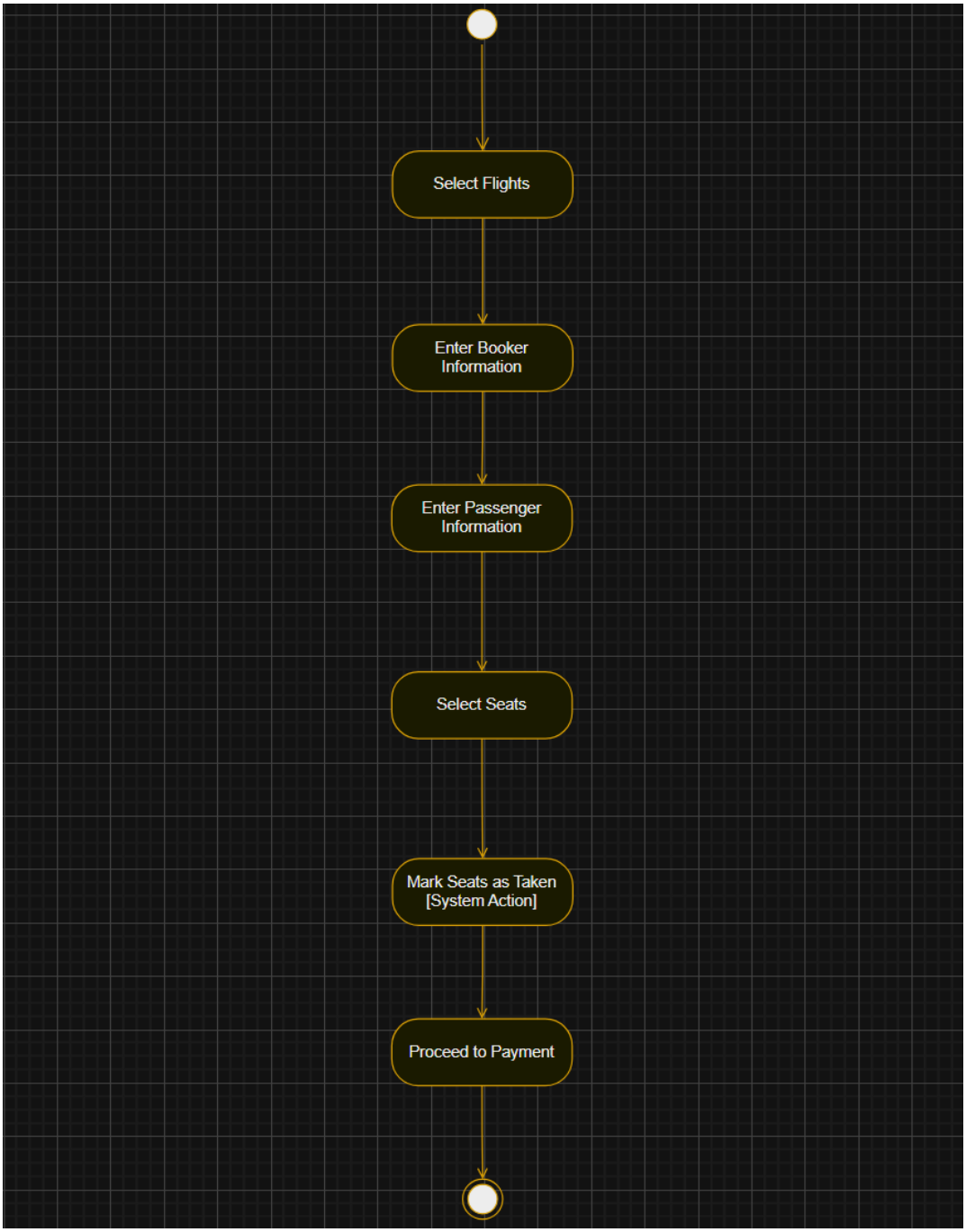
The customer can then browse through the available flights that match the details they entered and identify the one that best fits their travel plan.



Activity Diagram that shows the Process of Booking Flights

Once the customer has finished browsing and identified their desired flight, they can select it and will be presented with an interface to enter passenger details for the number of passengers entered on the search panel.

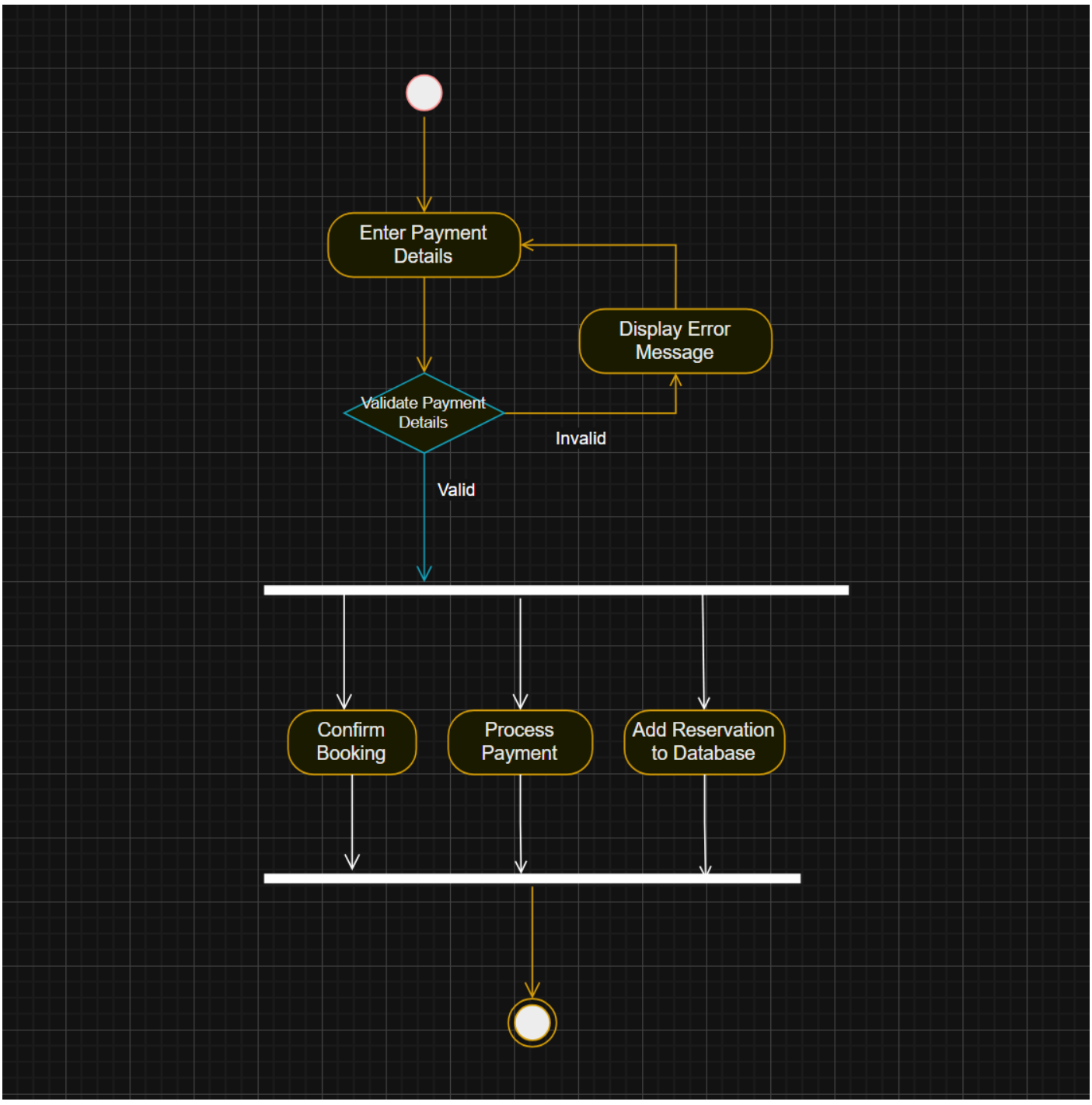
When all passenger details have been entered, the customer may select seats. Once seats are selected, the system automatically marks them as taken. The customer can then proceed to the payment stage to complete the booking.



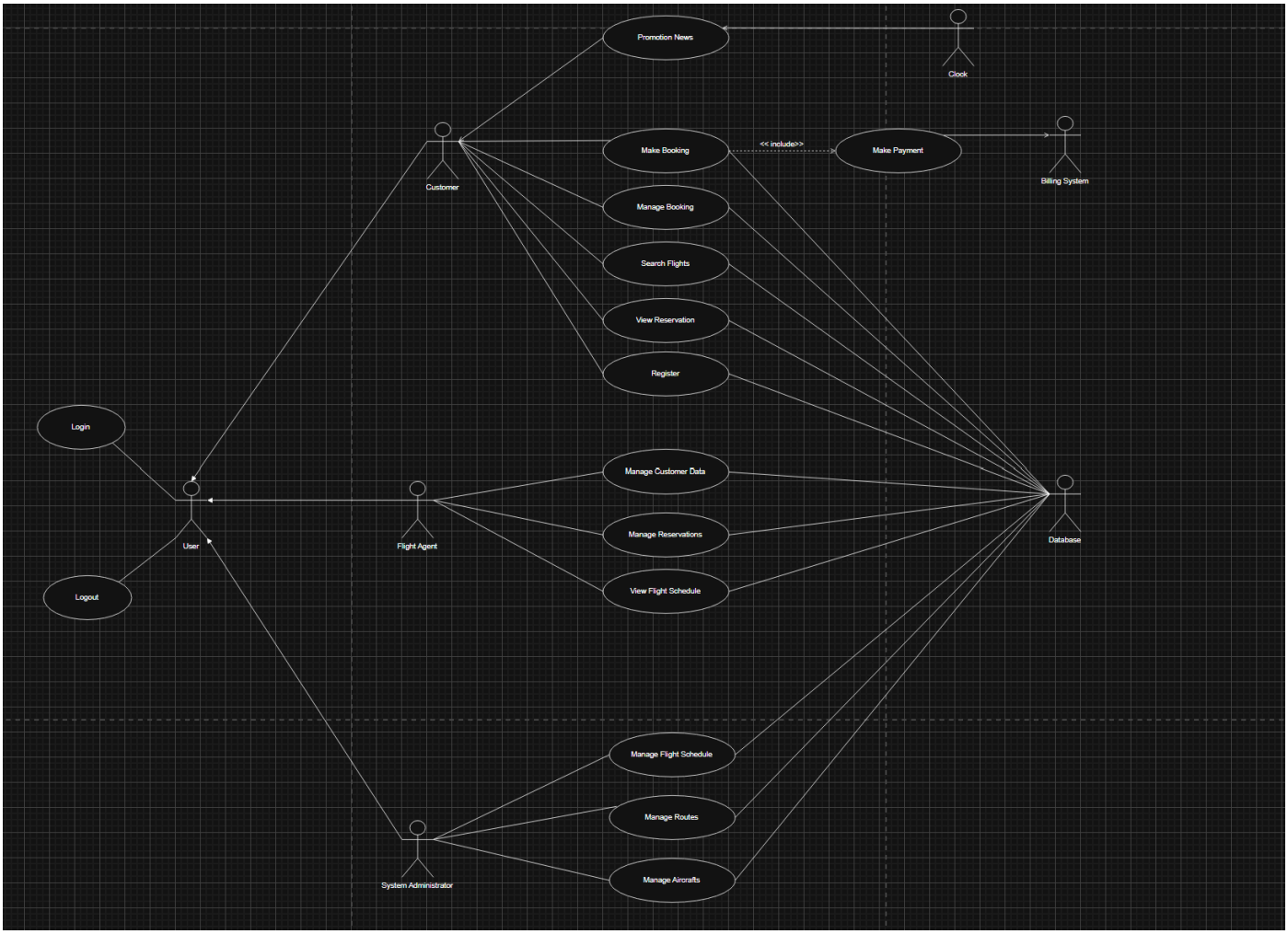
Activity Diagram that shows the Process of Making Payment

Once the booking process is complete, the customer will be prompted to pay for their flight. They must first enter their payment details, after which the system validates the information. If the details are invalid, an error message will be displayed, and the user will be prompted to re-enter their payment details.

If the payment details are valid, three processes will occur concurrently: the booking confirmation will be displayed to the customer, the payment will be processed, and the reservation will be added to the database.



Use Case Diagram



Use Case Scenarios

- Login:
  The user must successfully login to the flight reservation system before accessing any other system features or performing operations such as searching flights, making bookings, or managing reservations.

- Register:
  If a user who intends to become a customer attempts to login without an existing account, they must first register by providing the required personal and account details before attempting to login again.

- Search Flights:
  After a user, who is a customer, logs in, they can search for flights by entering their flight plan (Origin, Destination, and trip type). The System retrieves matching flights from the database and displays the list of available flights that satisfy the given criteria.

- Make Booking:
  After the list of flights matching the Customer's flight plan is displayed, the Customer may make a booking by selecting a desired flight, entering passenger information, and selecting available Seats. The System then proceeds to payment to finalize the transaction and creates a reservation for the selected flight and seat(s) on completion.

- Manage Bookings:
  After a user, who is a customer, has created a Reservation, they may manage bookings by performing actions such as canceling a reservation or generating a Booking Confirmation. The System marks the reservation as cancelled in the database upon deletion.

- Make Payment:
  After a user, who is a customer, fills the booking form, they must make a payment for the Reservation to be confirmed. The System simulates the payment process, verifies the transaction, and then stores the relevant Payment and Reservation details in the Database.

- View Reservation:
  After a user, who is a customer, has created a Reservation, they may view the details of the reservation by clicking on the "View Reservations" tab in the header. The System retrieves the Reservation details from the Database and displays it to the user.

- Manage Customer Data:
  After a user, who is a flight agent, logs in, they can manage customer data by adding new customer profiles or updating existing customer information. The System validates the input, then either creates a new customer record or updates the existing one in the database.

- Manage Reservations:
  After a user, who is a flight agent, logs in, they can make changes to existing reservations (e.g. seat changes or name changes). The System retrieves the Reservation details from the database, applies the requested modifications, and updates the Reservation record.

- View Flight Schedule:
  After a user, who is a flight agent, logs in, they can view the flight schedule to access current and upcoming Flights. The System retrieves all flight data from the Database and displays the schedule in a professional tabular format.

- Manage Flight Schedule:
  After a user, who is a system administrator logs in, they can manage the flight schedule by adding new Flights, removing existing ones, or updating flight information such as departure time, arrival time, or the assigned Aircraft. The System retrieves relevant Flight data from the database, applies the updates, and saves the modified information back to the database.

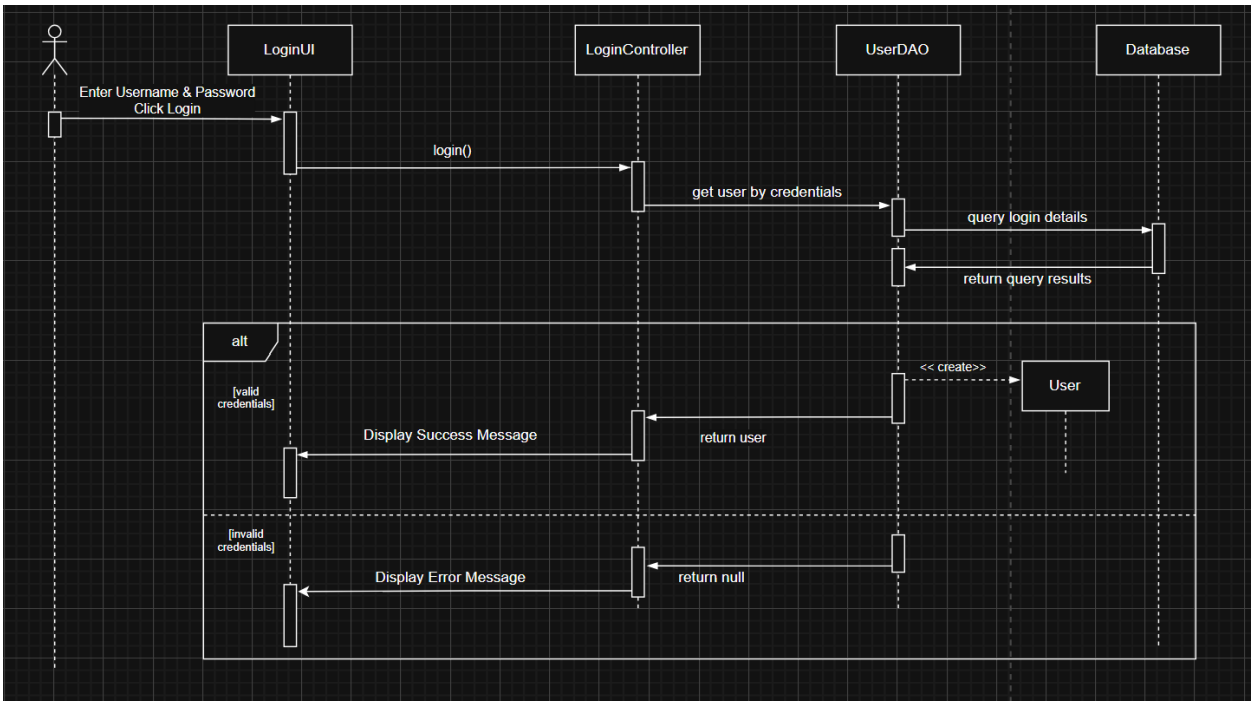
- Manage Routes:
  After a user, who is a system administrator, logs in, they can manage routes by adding new Routes, removing existing ones, or updating route information such as origin, destination, or airport details. The System retrieves relevant Route data from the database, applies the updates, and saves the modified information.

- Manage Aircraft:
  After a user, who is a system administrator, logs in, they can manage aircraft by adding new Aircraft to the fleet, removing existing ones, or updating Aircraft details. The System retrieves the relevant Aircraft data from the database, applies the updates, and saves the modified information.

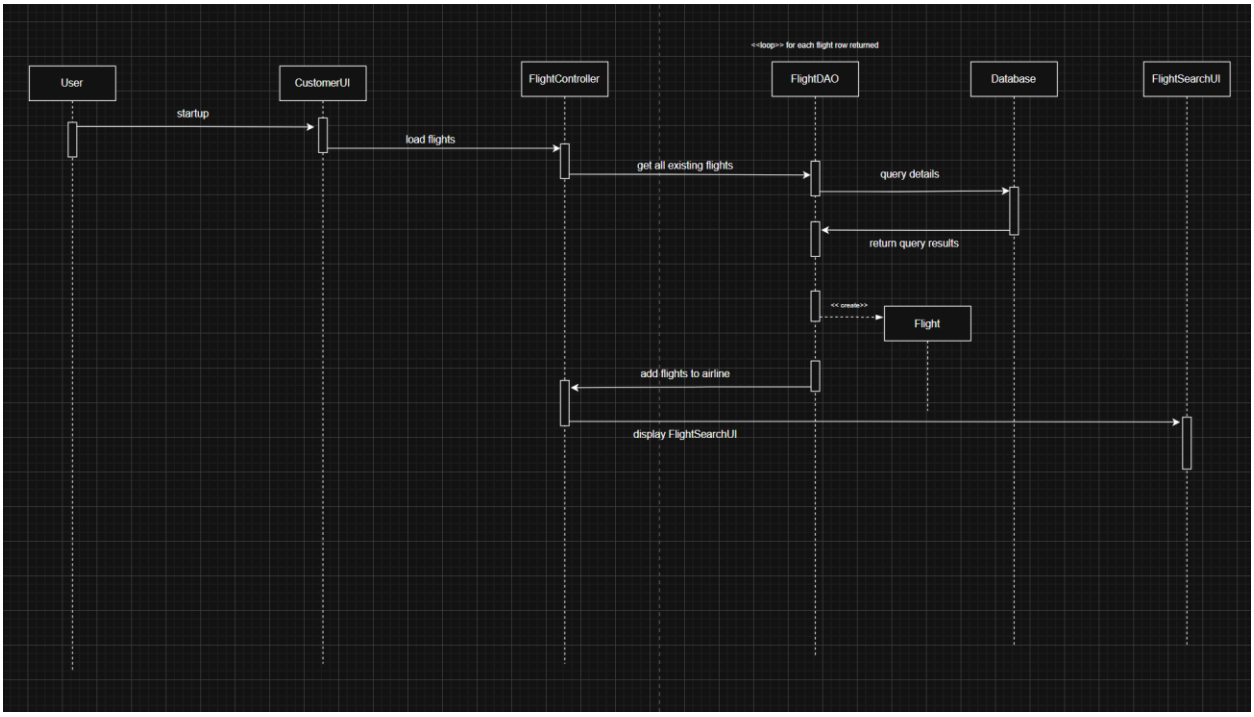
- Promotion News:
  After a Customer logs in, they can view promotion news to see ongoing flight deals or special offers. The System retrieves current Promotions from the promotion controller (since it is simulated) and alerts the user of a new promotion on the first day of each month.

Sequence Diagrams

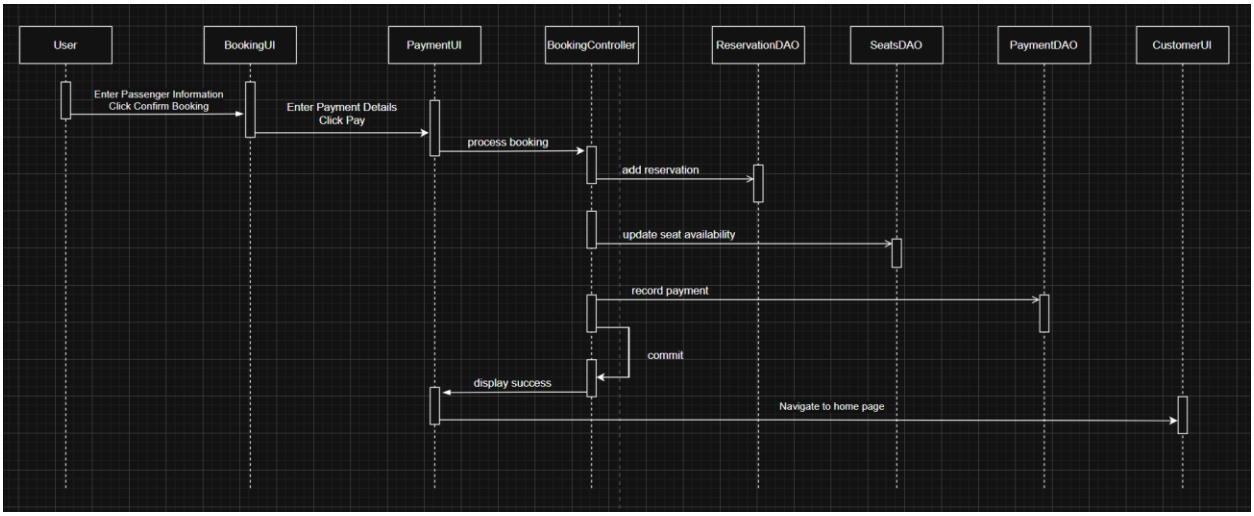
Login Use case:



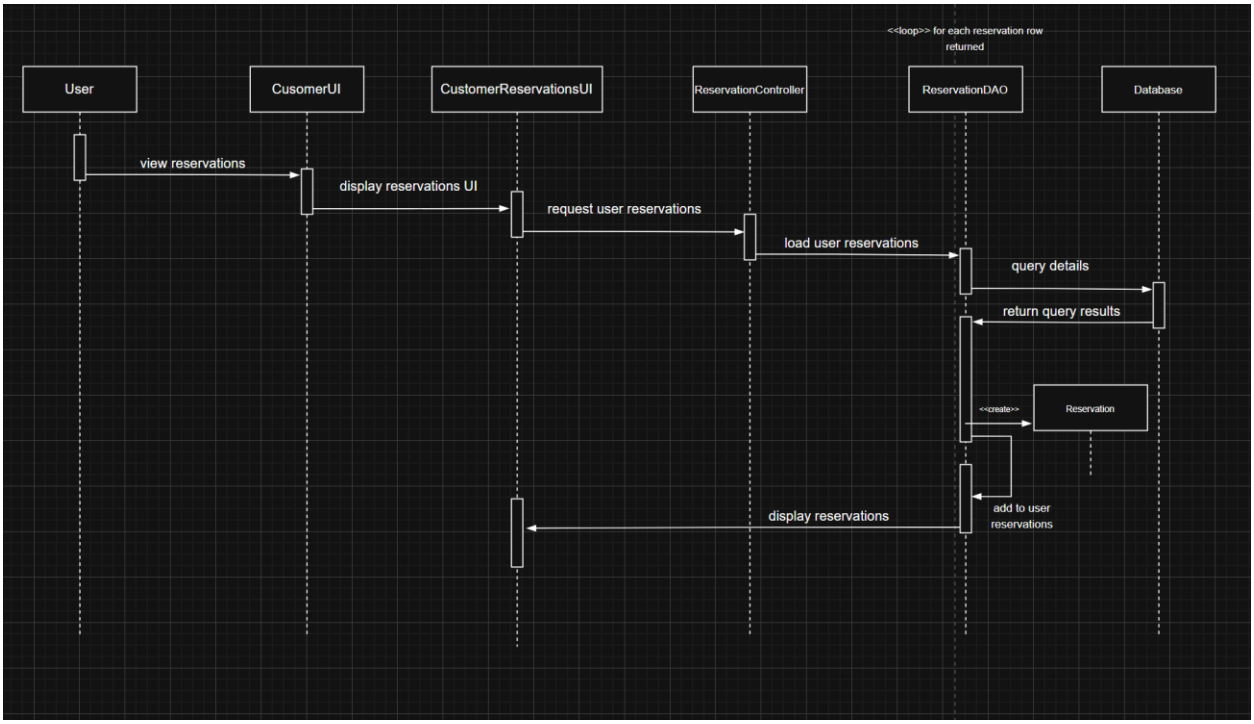
Search Flights Use case:



Make Booking:

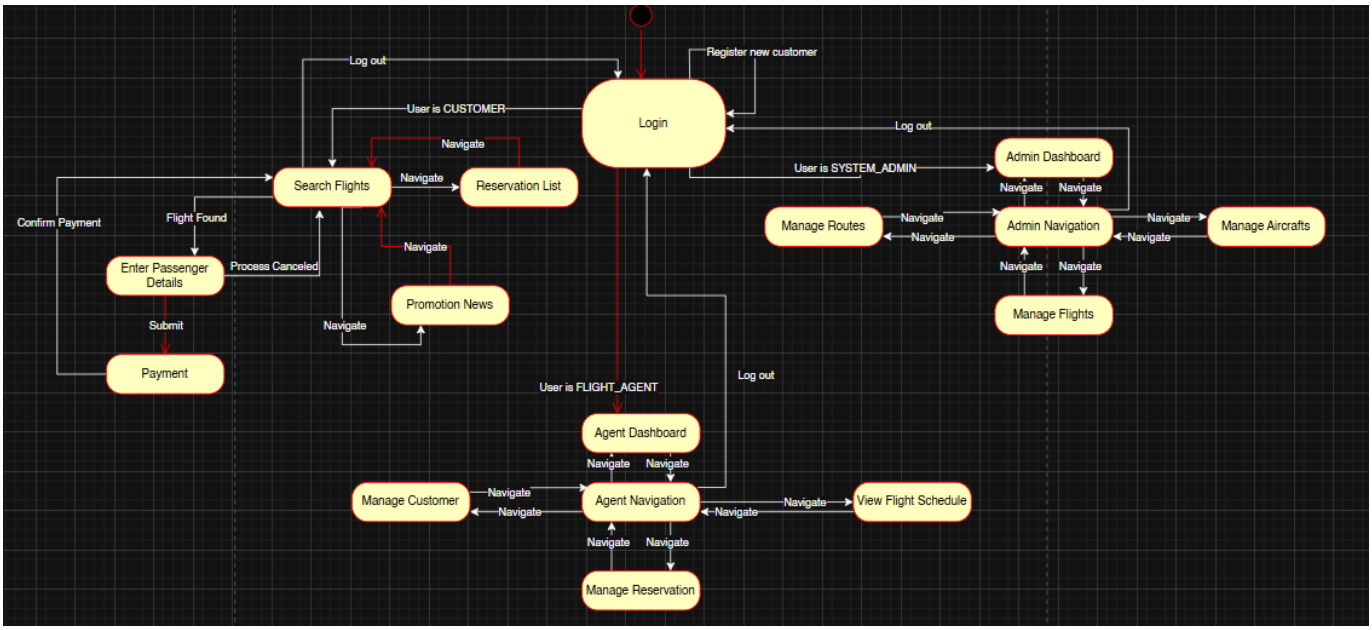


View Reservations:



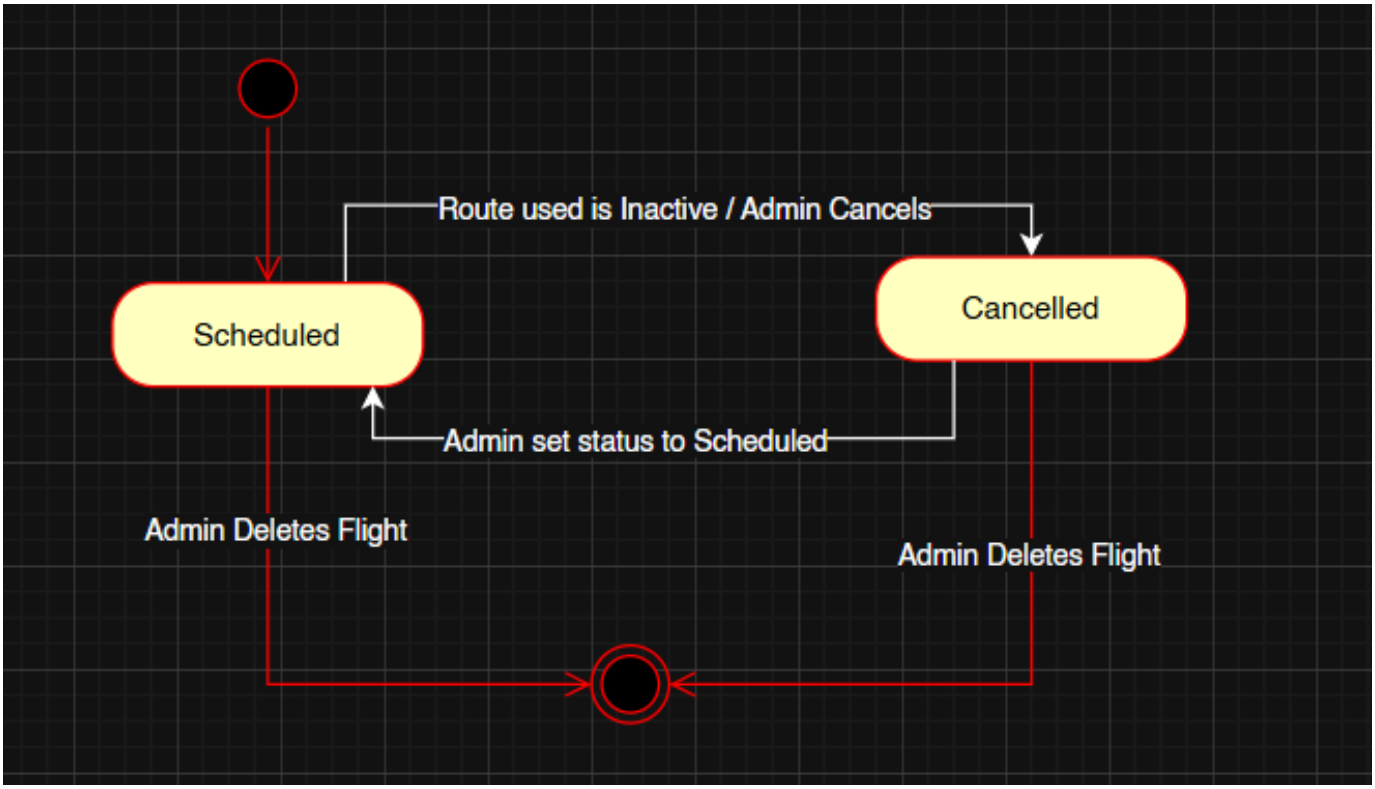
State Transition Diagrams

Whole System:

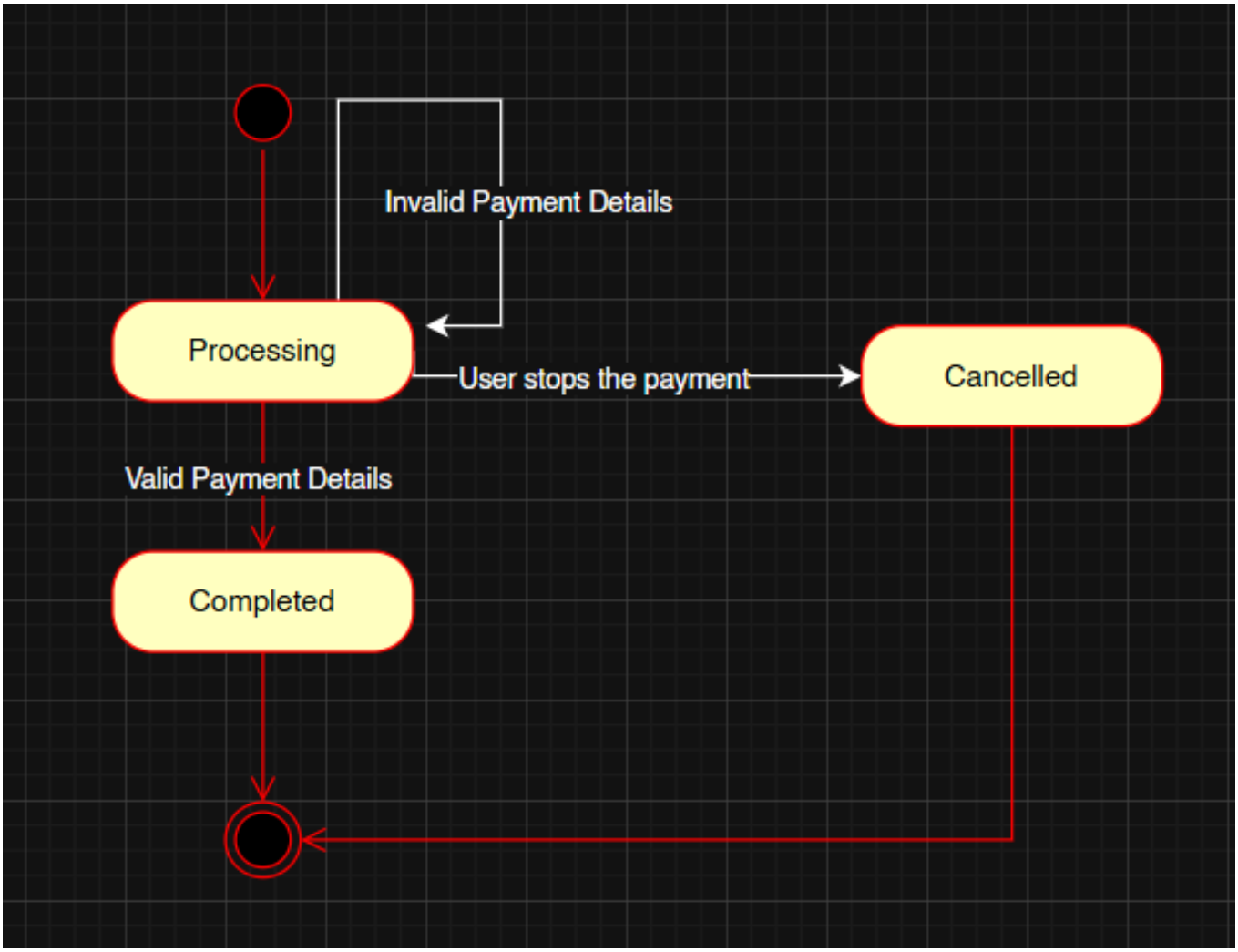


The system is basically a combination of different interfaces presented to the user. These are the states we believe they will be in at different points.

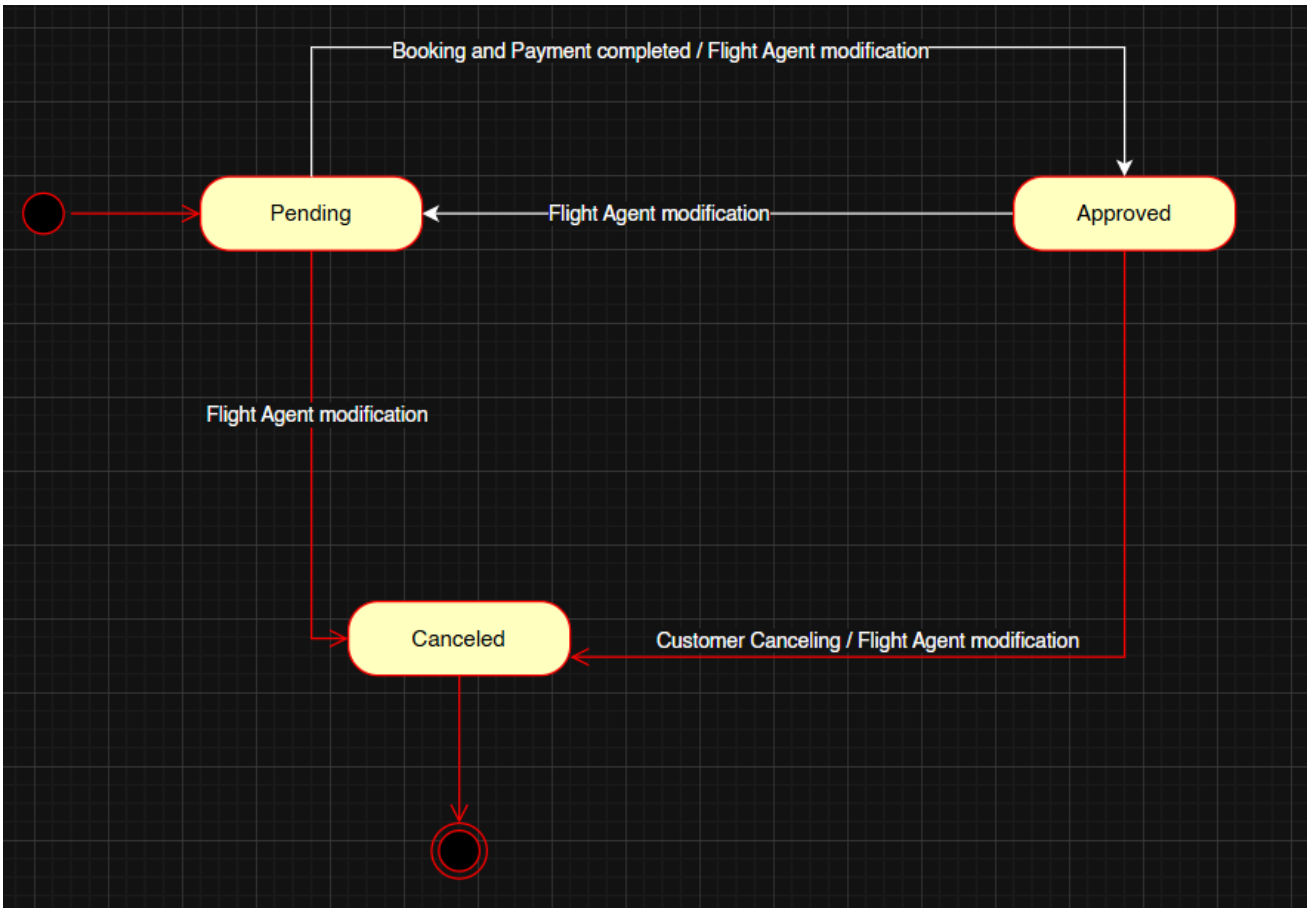
Flight:


Route used is Inactive / Admin Cancels
Scheduled
Cancelled
Admin set status to Scheduled
Admin Deletes Flight
Admin Deletes Flight

Payment:

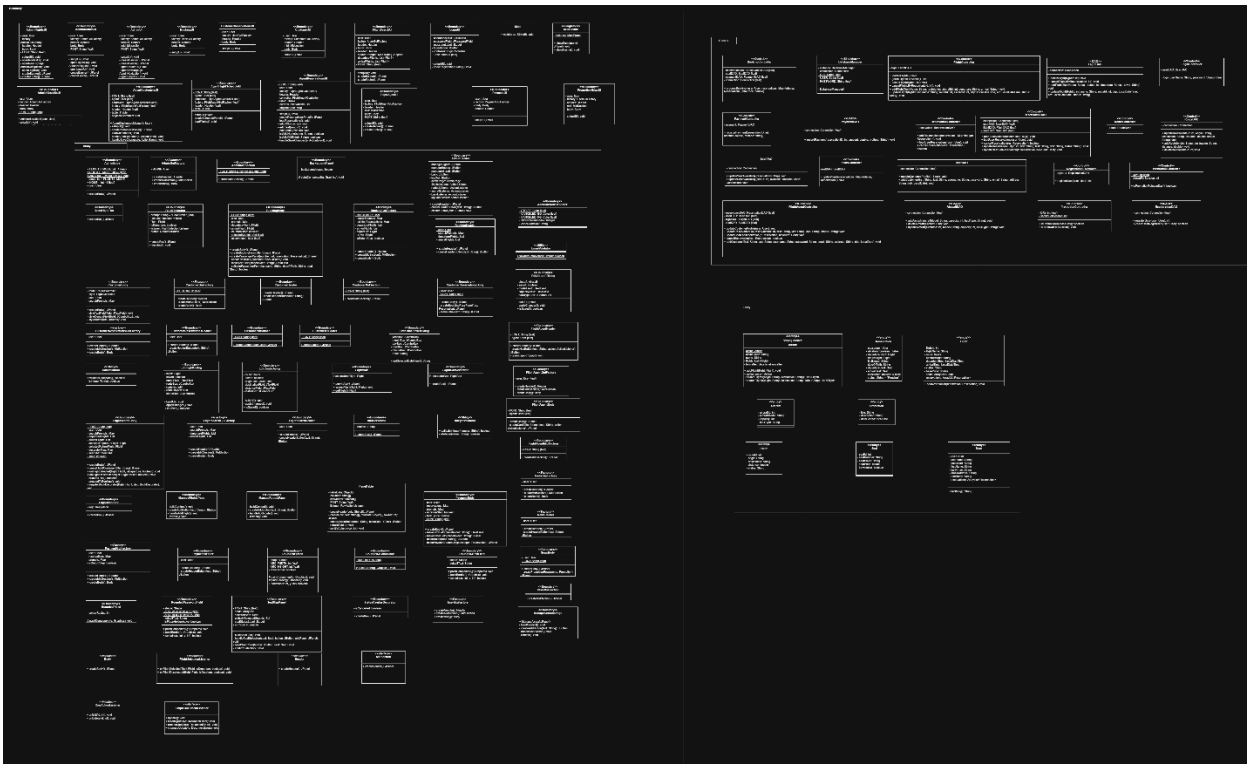

Invalid Payment Details
Processing
User stops the payment
Cancelled
Valid Payment Details
Completed

Reservation:



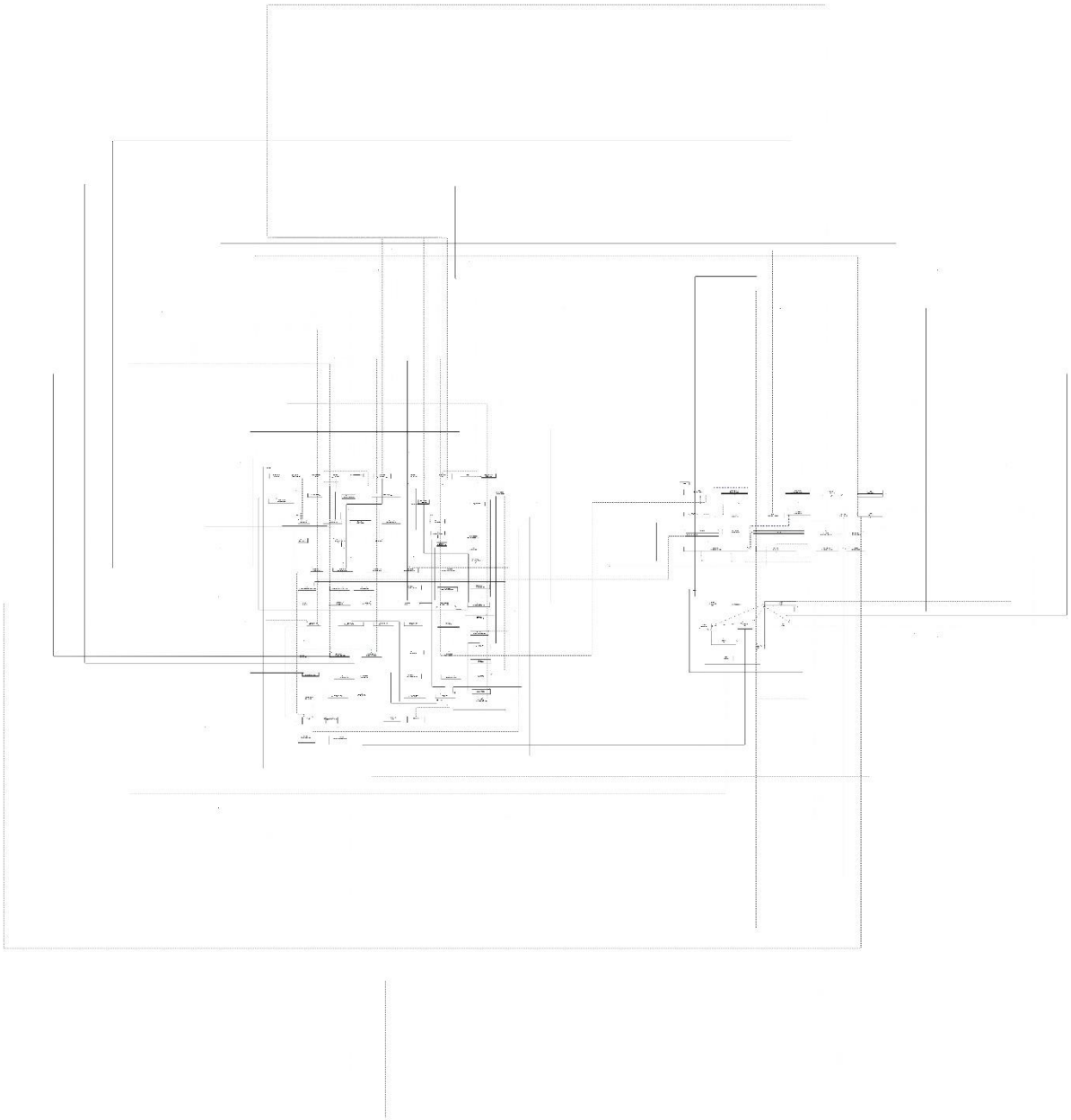
Class Diagrams:

Just important attributes, no relationships:



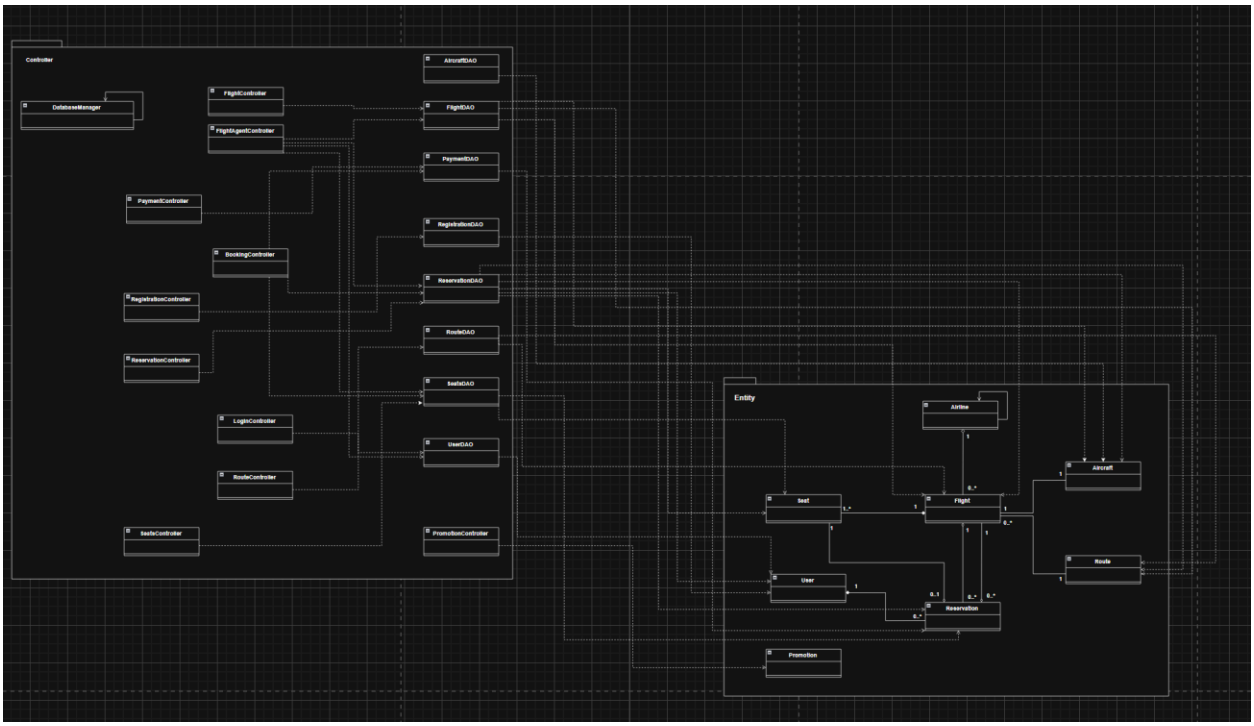
Note: We know you can’t see this (sorry), but we have attached the actual image so you can zoom in. Thank you!

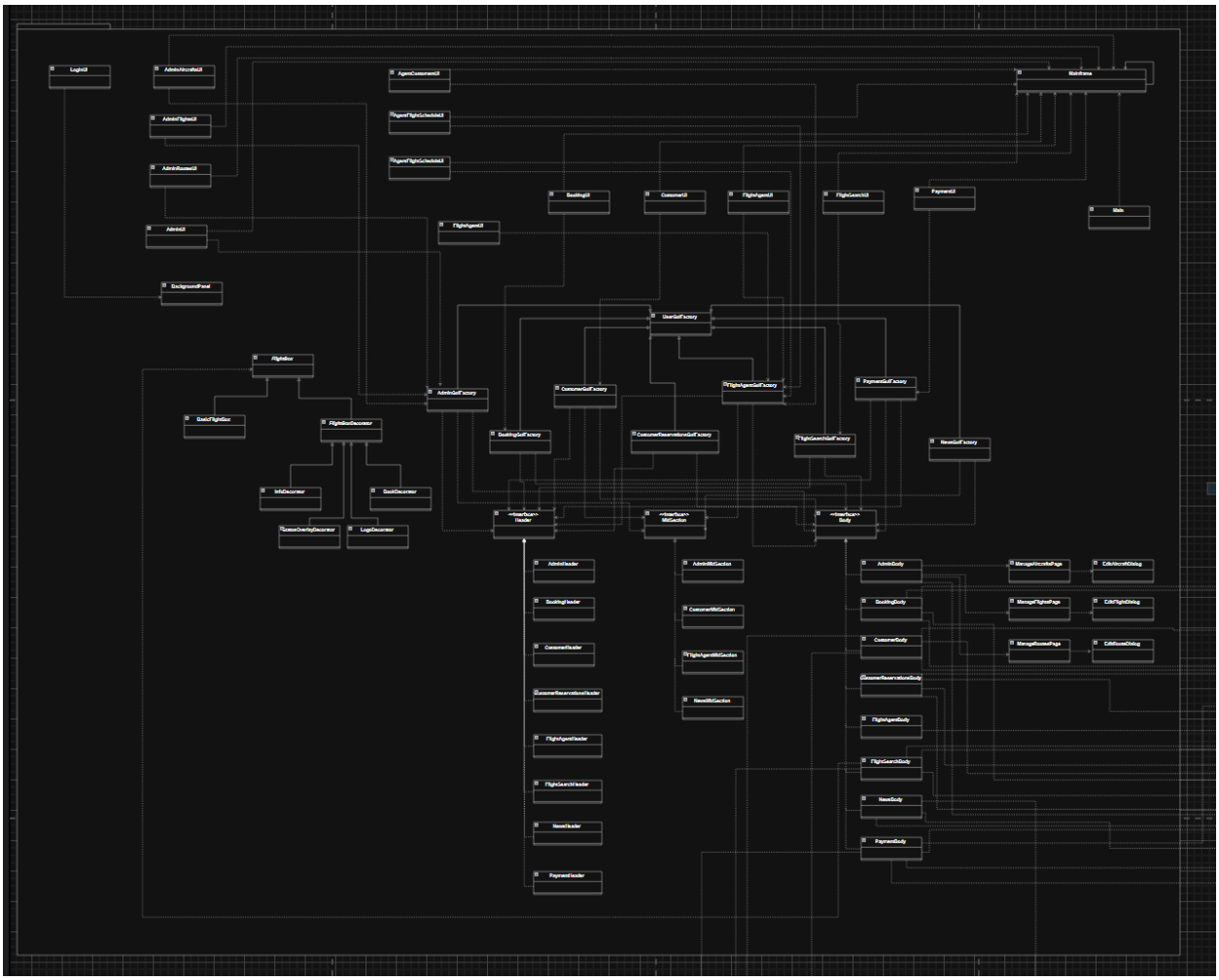
Class Diagram With Relationships:



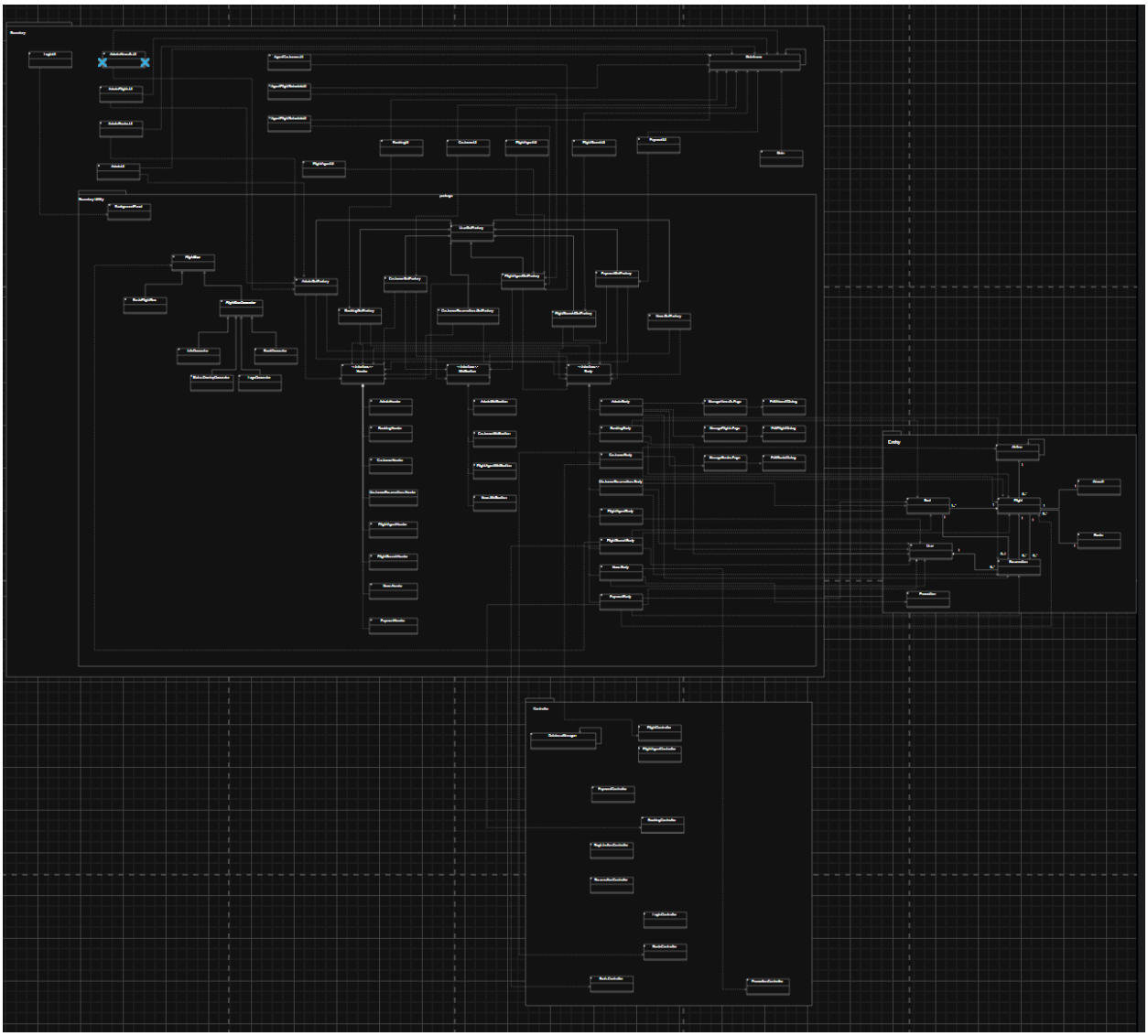
Our original class diagram is large and was not able to be exported as a clear image, so we provided, a draw.io file so that it can be viewed clearly (we have also included the image in case it renders on your device). Alternatively, we also created a much simpler version taking out classes that did not add to core functionality (purely for cosmetics). Here is the diagram:

The diagram below shows the interaction between the Controller and Entity packages, it also shows the details of the relationships between the entity classes themselves.

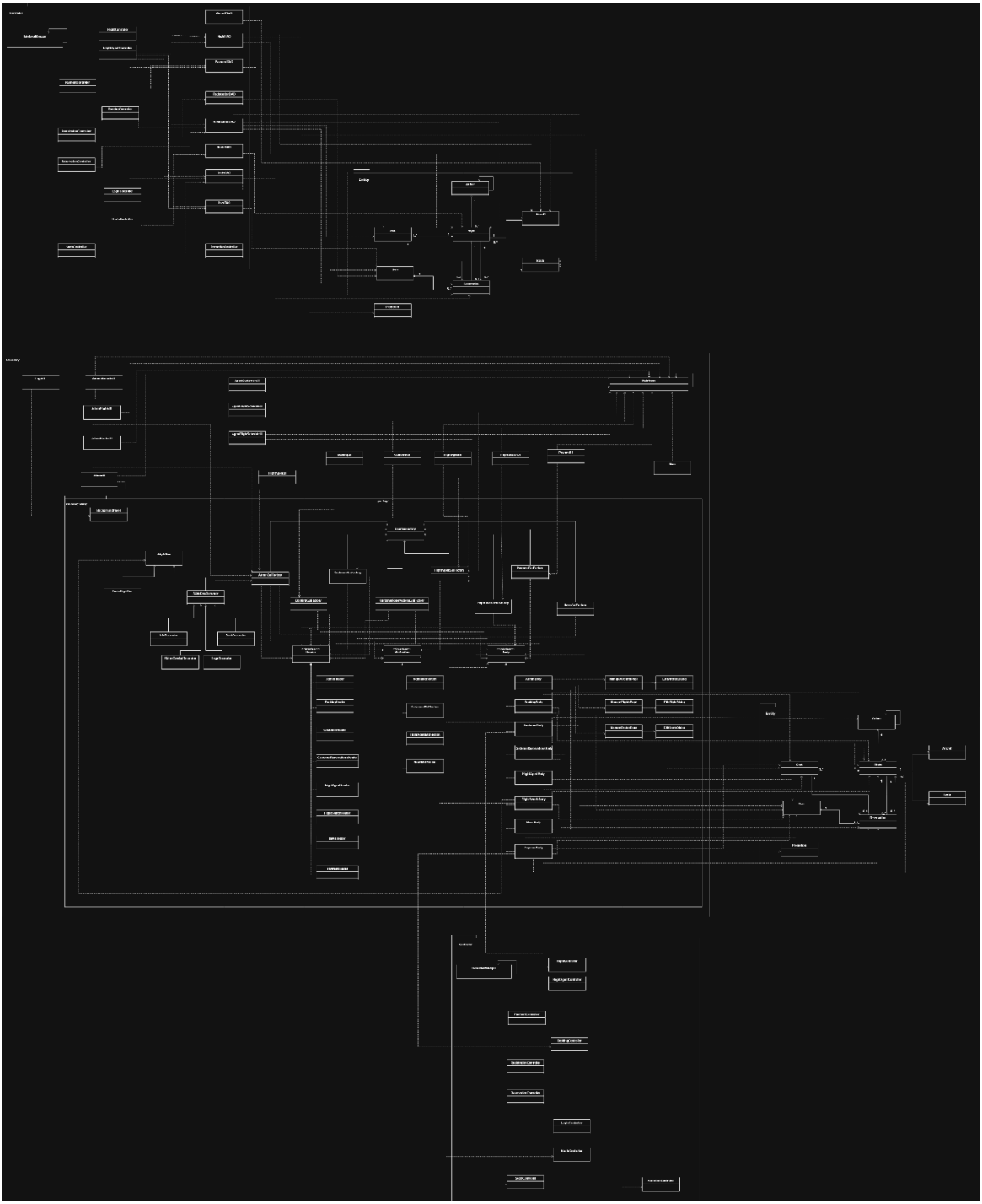

Controller
DatabaseManager
FlightController
FlightAgentController
PaymentController
BookingController
RegistrationController
ReservationController
LoginController
RouteController
SeatsController
AircraftDAO
FlightDAO
PaymentDAO
RegistrationDAO
ReservationDAO
RouteDAO
SeatDAO
UserDAO
PromotionController
Entity
Airline
Aircraft
Seat
Flight
Route
User
Reservation
Promotion


LogIn
AdminAircraftUI
AdminFlightUI
AdminRouteUI
AdminUI
BackgroundPanel
AgentCustomerUI
AgentFlightSchedulUI
AgentFlightSchedulUI
BookingUI
CustomerUI
FlightAgentUI
FlightSearchUI
PaymentUI
Main
MainFrame
UserGUIFactory
FlightBox
DeckFlightBox
FlightBoxDecorator
InfoDecorator
SeatDecorator
GlassOverlayDecorator
LogoDecorator
AdminGUIFactory
CustomerGUIFactory
FlightAgentGUIFactory
PaymentGUIFactory
BookingGUIFactory
CustomerReservationsGUIFactory
FlightSearchGUIFactory
NewsGUIFactory
<<interface>> Header
<<interface>> MidSection
<<interface>> Body
AdminHeader
BookingHeader
CustomerHeader
CustomerReservationsHeader
FlightAgentHeader
FlightSearchHeader
NewsHeader
PaymentHeader
AdminMidSection
CustomerMidSection
FlightAgentMidSection
NewsMidSection
AdminBody
BookingBody
CustomerBody
CustomerReservationsBody
FlightAgentBody
FlightSearchBody
NewsBody
PaymentBody
ManageAircraftPage
EditAircraftDialog
ManageFlightsPage
EditFlightDialog
ManageRoutesPage
EditRouteDialog

The diagram above is the boundary package, which is the biggest package in our system due to the application of GUI's. I will be showcasing its interaction with other system components below:



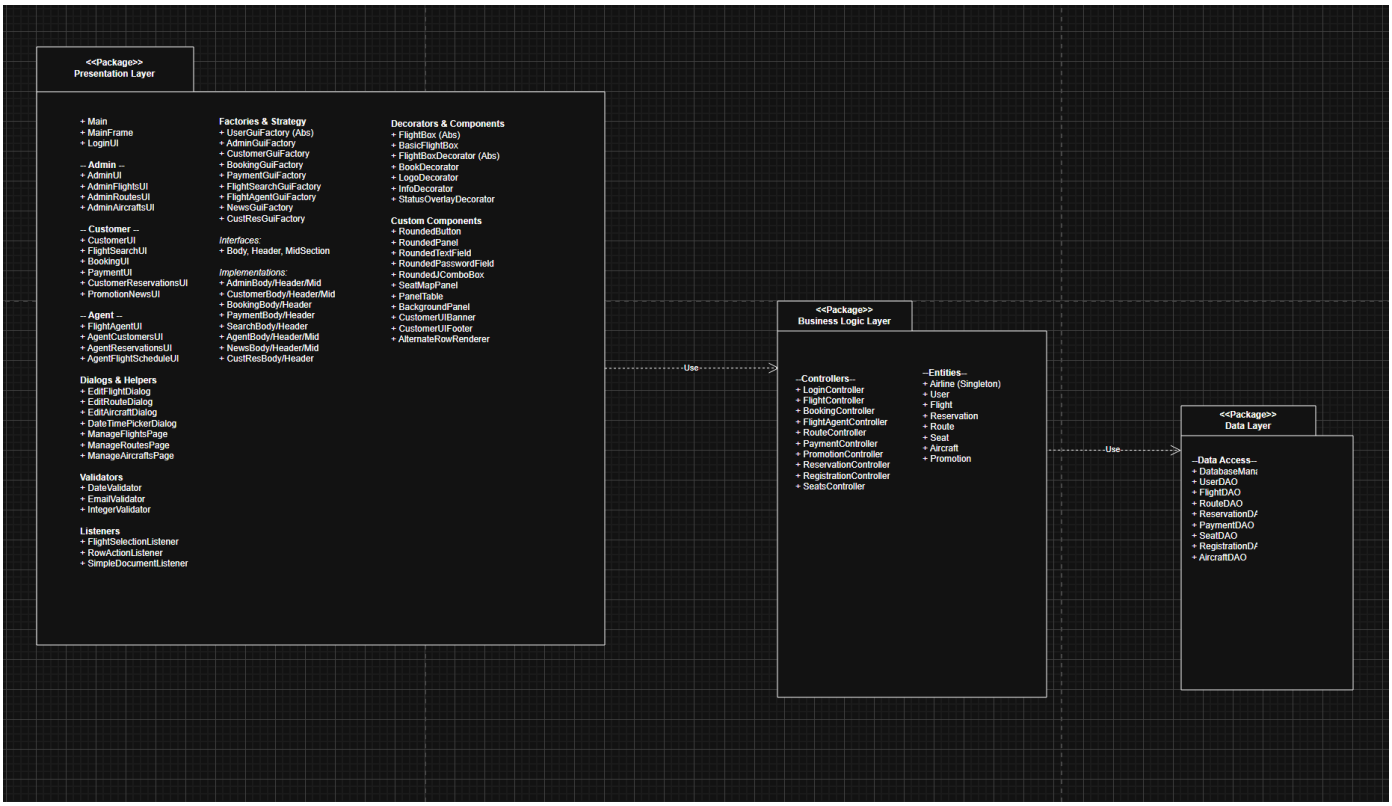
Now below, I will paste a screenshot of the entire diagram (note that I repeated entity and controller for boundary relationships, so it won't get very cluttered)

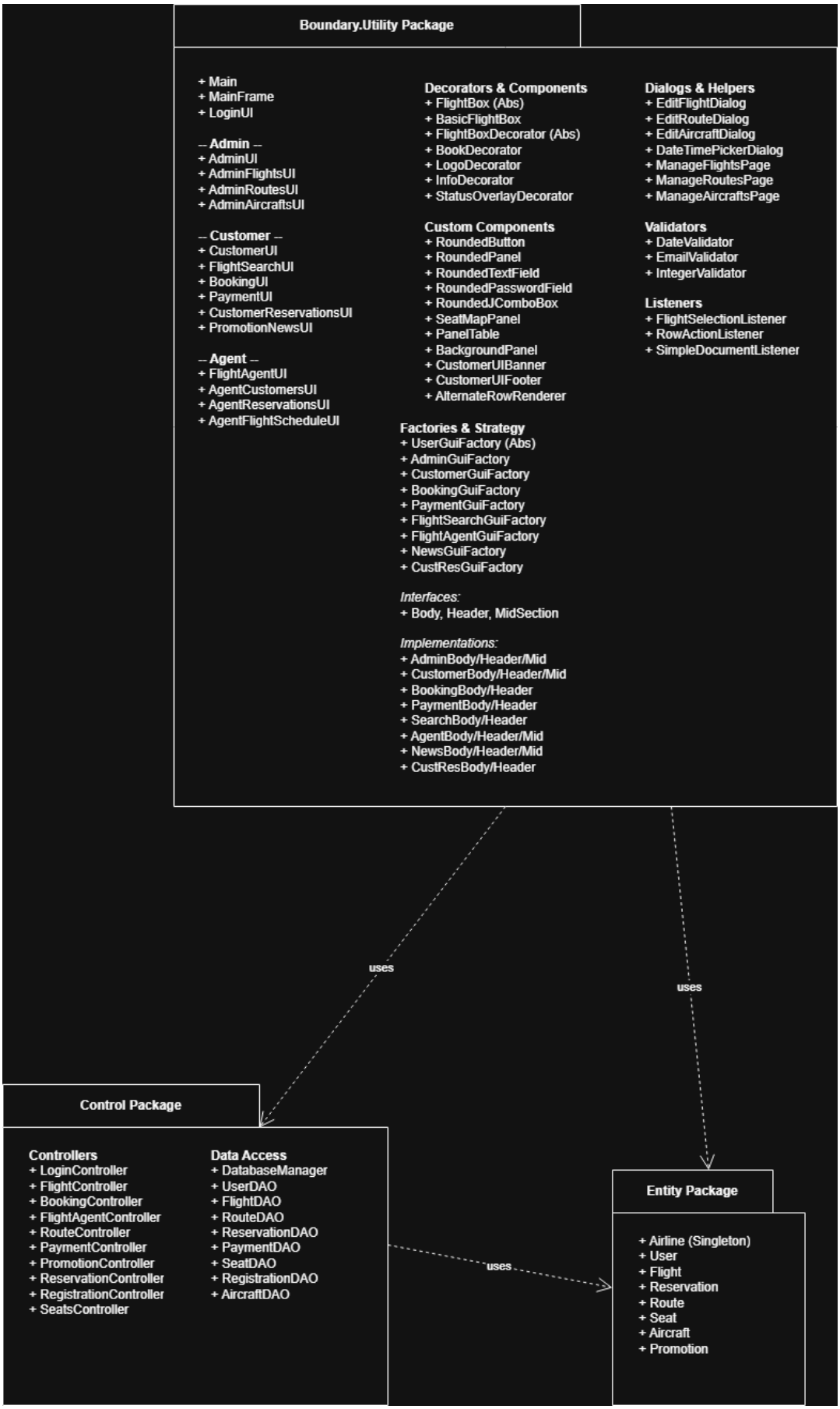


Note: Entity and Controller were repeated prevent congestion (as this was why the original diagram was very spread out).

Package Diagram:

Below is the package diagram of our flight reservation system.




Boundary.Utility Package
+ Main
+ MainFrame
+ LoginUI
-- Admin --
+ AdminUI
+ AdminFlightsUI
+ AdminRoutesUI
+ AdminAircraftsUI
-- Customer --
+ CustomerUI
+ FlightSearchUI
+ BookingUI
+ PaymentUI
+ CustomerReservationsUI
+ PromotionNewsUI
-- Agent --
+ FlightAgentUI
+ AgentCustomersUI
+ AgentReservationsUI
+ AgentFlightScheduleUI
Decorators & Components
+ FlightBox (Abs)
+ BasicFlightBox
+ FlightBoxDecorator (Abs)
+ BookDecorator
+ LogoDecorator
+ InfoDecorator
+ StatusOverlayDecorator
Custom Components
+ RoundedButton
+ RoundedPanel
+ RoundedTextField
+ RoundedPasswordField
+ RoundedJComboBox
+ SeatMapPanel
+ PanelTable
+ BackgroundPanel
+ CustomerUIBanner
+ CustomerUIFooter
+ AlternateRowRenderer
Dialogs & Helpers
+ EditFlightDialog
+ EditRouteDialog
+ EditAircraftDialog
+ DateTimePickerDialog
+ ManageFlightsPage
+ ManageRoutesPage
+ ManageAircraftsPage
Validators
+ DateValidator
+ EmailValidator
+ IntegerValidator
Listeners
+ FlightSelectionListener
+ RowActionListener
+ SimpleDocumentListener
Factories & Strategy
+ UserGuiFactory (Abs)
+ AdminGuiFactory
+ CustomerGuiFactory
+ BookingGuiFactory
+ PaymentGuiFactory
+ FlightSearchGuiFactory
+ FlightAgentGuiFactory
+ NewsGuiFactory
+ CustResGuiFactory
Interfaces:
+ Body, Header, MidSection
Implementations:
+ AdminBody/Header/Mid
+ CustomerBody/Header/Mid
+ BookingBody/Header
+ PaymentBody/Header
+ SearchBody/Header
+ AgentBody/Header/Mid
+ NewsBody/Header/Mid
+ CustResBody/Header
uses
uses
Control Package
Controllers
+ LoginController
+ FlightController
+ BookingController
+ FlightAgentController
+ RouteController
+ PaymentController
+ PromotionController
+ ReservationController
+ RegistrationController
+ SeatsController
Data Access
+ DatabaseManager
+ UserDAO
+ FlightDAO
+ RouteDAO
+ ReservationDAO
+ PaymentDAO
+ SeatDAO
+ RegistrationDAO
+ AircraftDAO
uses
Entity Package
+ Airline (Singleton)
+ User
+ Flight
+ Reservation
+ Route
+ Seat
+ Aircraft
+ Promotion

Database Steps

Download MySQL JDBC driver (V18.1) - https://dev.mysql.com/downloads/connector/j/

- Select "**Platform**" Independent.
- Download the "**ZIP Archive"** option
- Note that you will need to sign in to download the driver.
- Unzip the file, then copy and paste the "mysql-connector-j-9.5.0" at your desired destination.

Using "Eclipse IDE", make a folder in your java project called "lib", and paste the "mysql-connector-j-9.5.0.jar" file there. Right click the jar file, then select "Build path" -> "add to build path".

Open MYSQL workbench and select your "Local Instance" connection.

Copy and paste the file "flight_reservation_database.

Uses Synchronized for thread safety