UNIVERSITY OF CALGARY

ENSF 480 - Principles of Software Design

Flight Reservation Program

Contributors: Afrah Mohammad (#30144844)

Ella Boulanger (#30163384)

Fabiha Tuheen (#30140219)

Hamza Itani (#30091353)

Part A: System Analysis

System's Description:

System Initialization: Upon initialization, the Flight Reservation Application presents a Graphical User Interface (GUI) that serves as the entry point to the system, displaying the home page with navigation options.

Flight Reservation Process: The application allows all users to peruse the flight database, which is dynamically populated from the system's backend. However, to proceed with flight reservation, user authentication is required. This involves a registration process where users create a unique account tied to their personal information. Post-registration, users can authenticate themselves via the login module. Upon successful authentication, users can reserve a flight by selecting from the available options, choosing their preferred seat category (ordinary, comfort, or business-class), providing payment details, and finally confirming the booking. The system then generates a digital receipt, which is dispatched to the user's registered email address.

Registered User Interactions: Existing users, upon authentication, can access their personal dashboard which provides a consolidated view of their flight bookings. Additional features available to registered users include membership benefits and the option to apply for the airline's credit card.

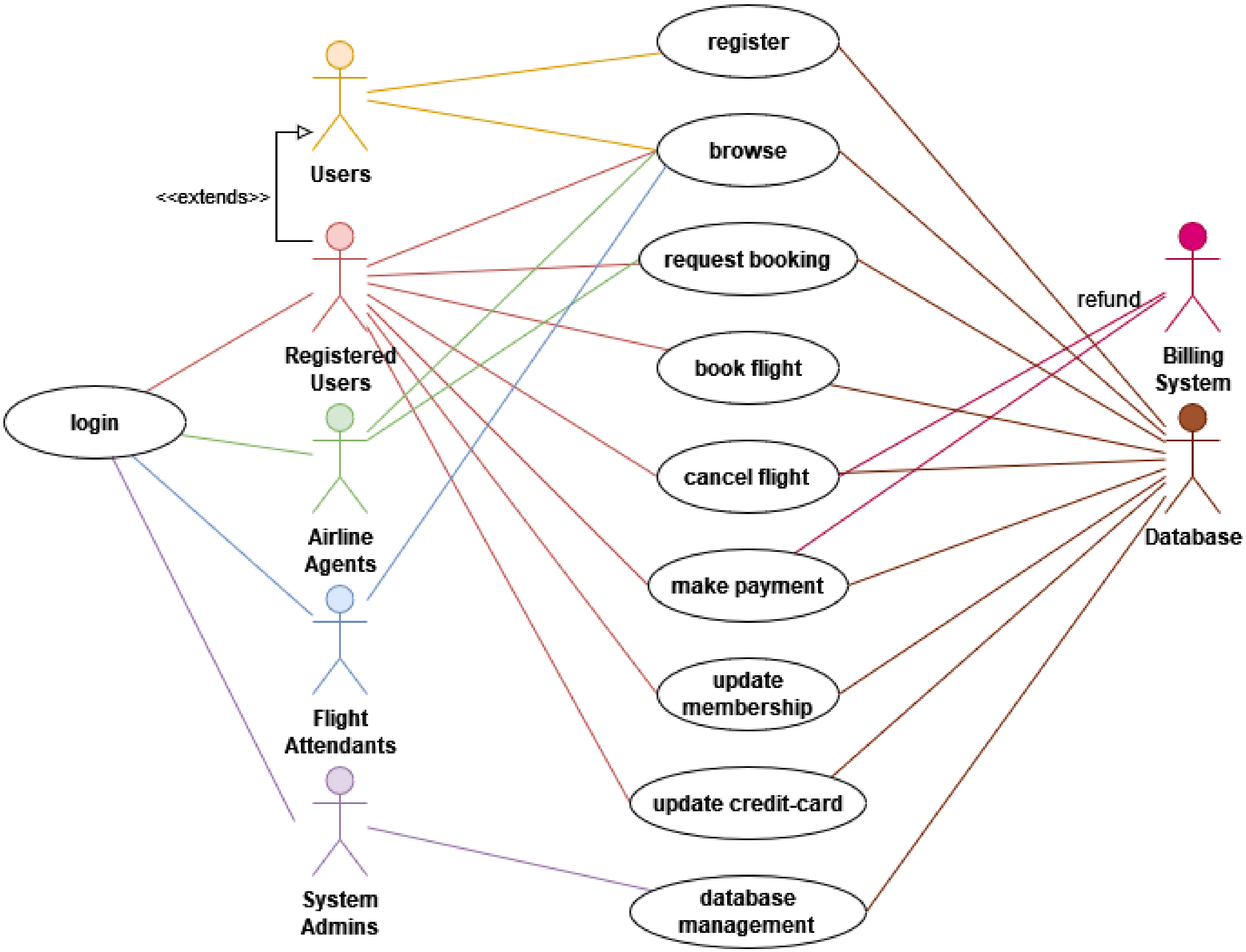
Flight Attendant Interface: Flight attendants, after logging into their dedicated portal, can input necessary details, view their flight schedules, and access passenger manifests for their assigned flights.

Tourism Agent Access: Tourism agents, upon successful login, can perform actions similar to registered users. They can input required details, view flight schedules, and manage bookings on behalf of their clients.

Airline Agent Operations: Airline agents, post-login, can access passenger lists, manage flight bookings for registered users, and perform other customer service-related tasks.

Admin Control: Admin users have privileged access to the system. Upon login, they can manage various system entities including flight schedules, aircraft details, crew assignments, passenger lists, and destination data. They have full control over the database, allowing them to perform CRUD (Create, Read, Update, Delete) operations on all system records.

System's Use-Case Diagram: *(Basic)*



System’s Scenarios:

Use-Case Scenario: Browse Flights

System asks user (Registered User or Airline Agent) to select a destination from a list of available destinations read from the database. User indicates the desired destination. User can now browse all flights going to the desired destination.

Use-Case Scenario: Browse Passenger List

System asks flight attendant to select a flight from a list of flights they are scheduled to work on. Flight attendant indicates the desired flight. Flight attendant can now browse a passenger list of all the passengers on the selected flight.

Use-Case Scenario: Make payment

Once through the flight selection process, the user (Registered User or Airline Agent) will be prompted to enter payment information such as: card information, full name, billing address, etc. If the user to be booked* on the flight has a saved personal card, a saved company card, or both, the user may choose to use one of the saved payment methods to autofill those fields if they wish. Once all the necessary information has been provided the user may proceed to checkout, the system will book the user to be booked *on the flight, while the billing system charges the user's card, a ticket will be generated.

*Airline Agent books on behalf of a Registered User, a Registered User books for themselves.

Use-Case scenario: Booking Flight

Upon logging into their account, the user selects "Book a flight". They browse destinations and select their desired flight. The system displays a seat map for the selected flight, including regular, comfort, or business-class seats. The user selects a desired seat, and checks its availability. The database marks the chosen seat as unavailable for future bookings. The user is prompted to select optional ticket cancellation insurance. The user makes the payment using a credit card or chooses a prepaid company card if they are a member. Additional costs for comfort (40% more) or business-class seats (double the amount) will apply. The system sends payment details to the database for processing and validation, and confirmation is sent back. If the payment is not valid, the system displays an error message. The flight booking information is stored in the database. The system displays a ticket and receipt to the user.

Use-Case scenario: Request Booking

An airline agent logs into the system, reviews the list of booking requests with flight information from registered users. The agent proceeds to the booking flights view following the standard booking process. When reaching the payment section, the user completes the payment process using their credit card or company card. Upon confirmation from the system, the booking is confirmed, and updated in the database, and the user's request is removed from the agent's list.

Use-Case scenario: Cancel Booking

A registered user logs in and chooses their account and the system presents a list of their booked flights. The user selects the flight for cancellation, and the system prompts them to confirm. The system marks the chosen seat as available for future bookings, and the cancellation information is updated in the database. If the user had opted for optional cancellation insurance, the billing system issues a refund to their card, accompanied by a receipt displayed by the system. If the insurance was not selected, the system displays confirmation for the cancellation.

Use-Case Scenario: Registering Users

The user can browse flights as a “guest”, without logging in; but once ready to book a flight, they’re required to register for an account to become a registered user. User wishes to book a flight. Users are prompted to register in order to become a registered user so they can book a flight. Users' registered information is sent to the database. The database displays all the users that have signed up to an account. Database displays information (name, phone number, email, etc.) to the registered users’ account.

Use-Case Scenario: Updating Membership

After the User logs-in (and becomes a Registered User), they can request to update their Membership status. Registered User selects to edit Membership. Registered user then inputs necessary information to the Membership settings. When a Registered User is done with updating information, they’ll press Submit to send the information to the database. The database receives the request to overwrite any previously put membership information for that singular user, and does so Database then will send proper information to the GUI to display the appropriate changes made

Use-Case Scenario: Updating Credit-Card

After the User logs-in (and becomes a Registered User), they can request to update their Credit-Card (sign-up for a company one). Registered User selects to register for a Credit Card. Registered user then inputs necessary information to the credit-card information.

When a Registered User is done with updating information, they'll press Submit to send the information to the database. Database receives request for a credit-card for the instant of that user, and takes information provided to build the billing information for the credit-card. Database stores all necessary information of the Registered Users' credit-card, and sends information to the GUI when necessary for the Registered User to see the credit-card

Use-Case Scenario: Update Flight Info

This use case begins when the system admin chooses a flight and selects "update info". Users are prompted with a screen to select which information they want to adjust and when they select the one they want they can change it accordingly. The Database receives the request to view the information from the system admin and allows them to commit any changes they choose to, to the stored flight information as long as the type matches.

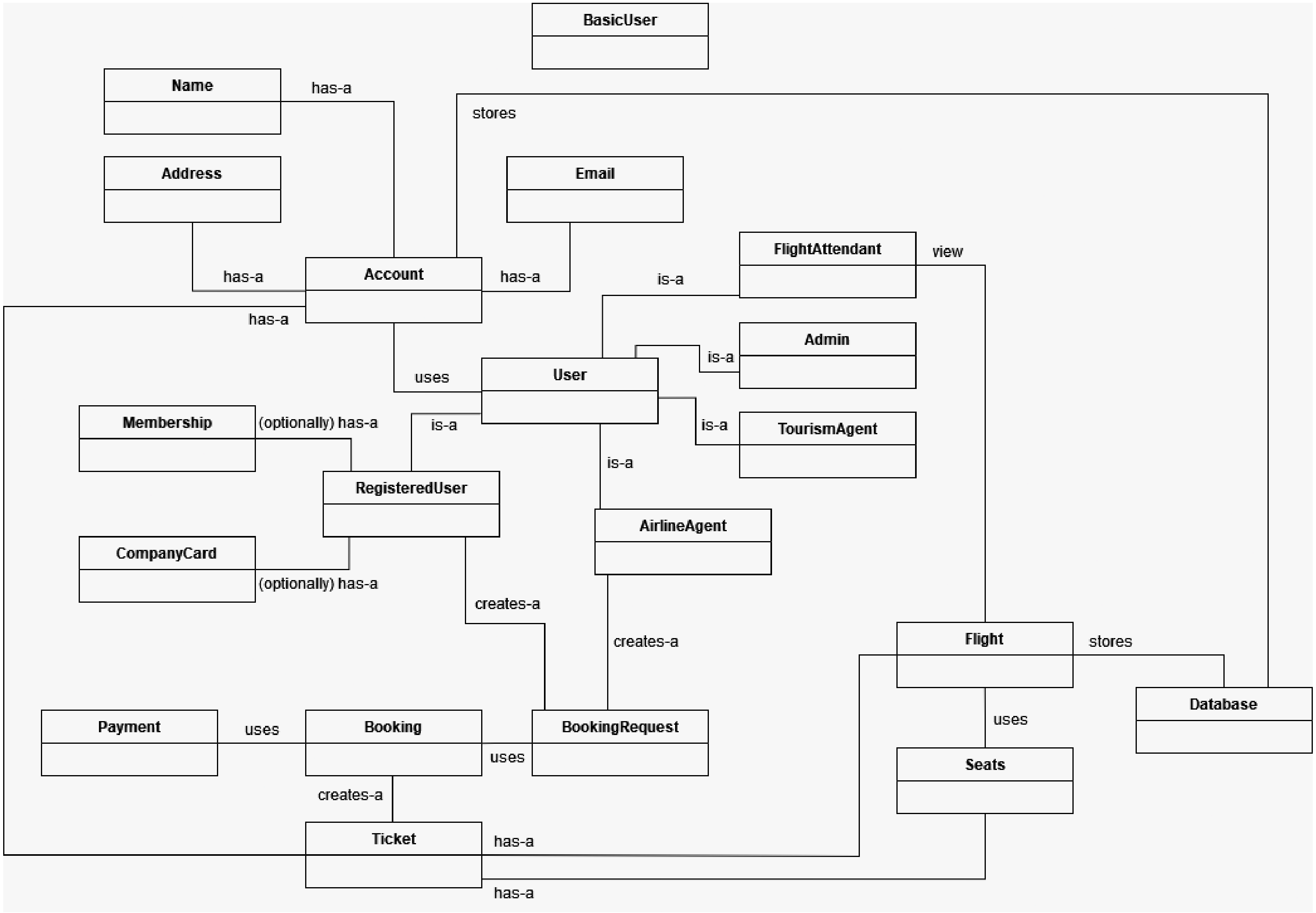
Use-Case Scenario: Browse Database

The database will send all the information to be displayed for the system admin without hiding anything as they have access to everything. The database will send all relevant information when the system admin selects a filter or different viewing option. The system admin is able to view all information and when selecting specific options they are given an updated view that matches what they have selected.

Use-Case Scenario: Update Database

The system admin can select any info they wish to commit any changes to. Once they save the changes it will be sent to the database to update them. The database will receive the changes made by the system admin and reflect them in the stored contents. Any info that is deleted will be changed to a null.

System’s Conceptual Model:



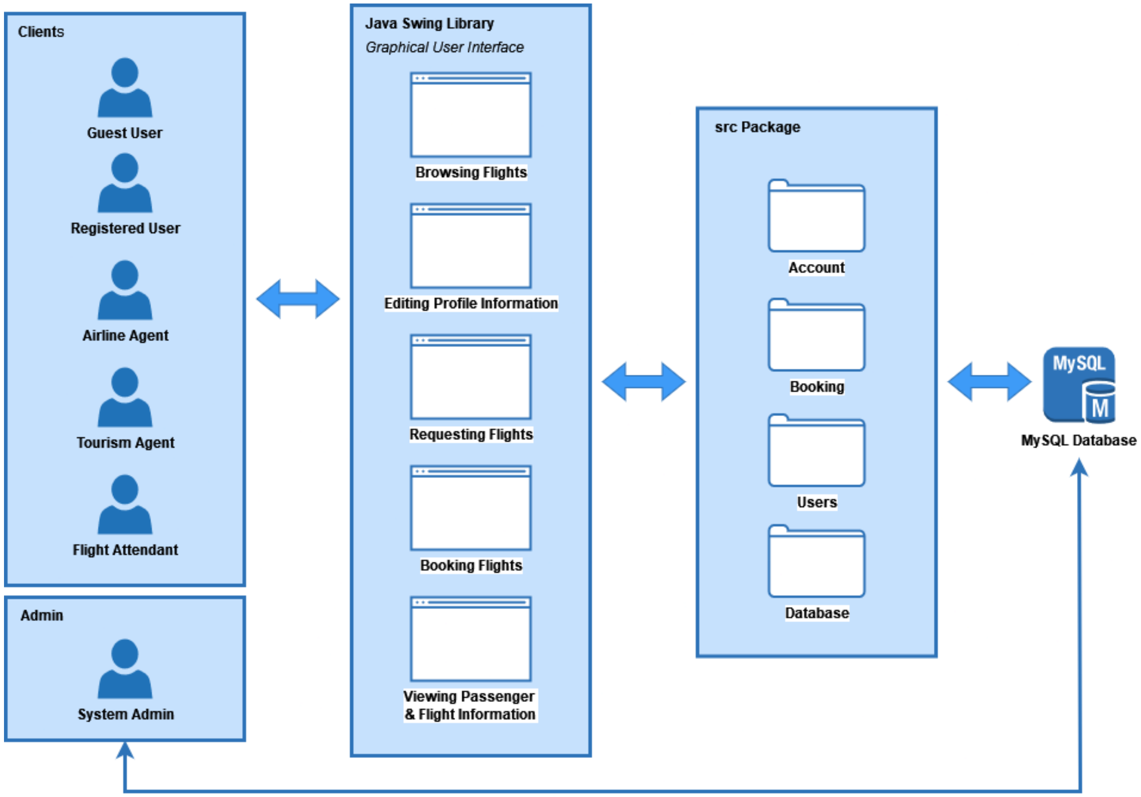
Part B: Domain Diagrams

System's Architecture:

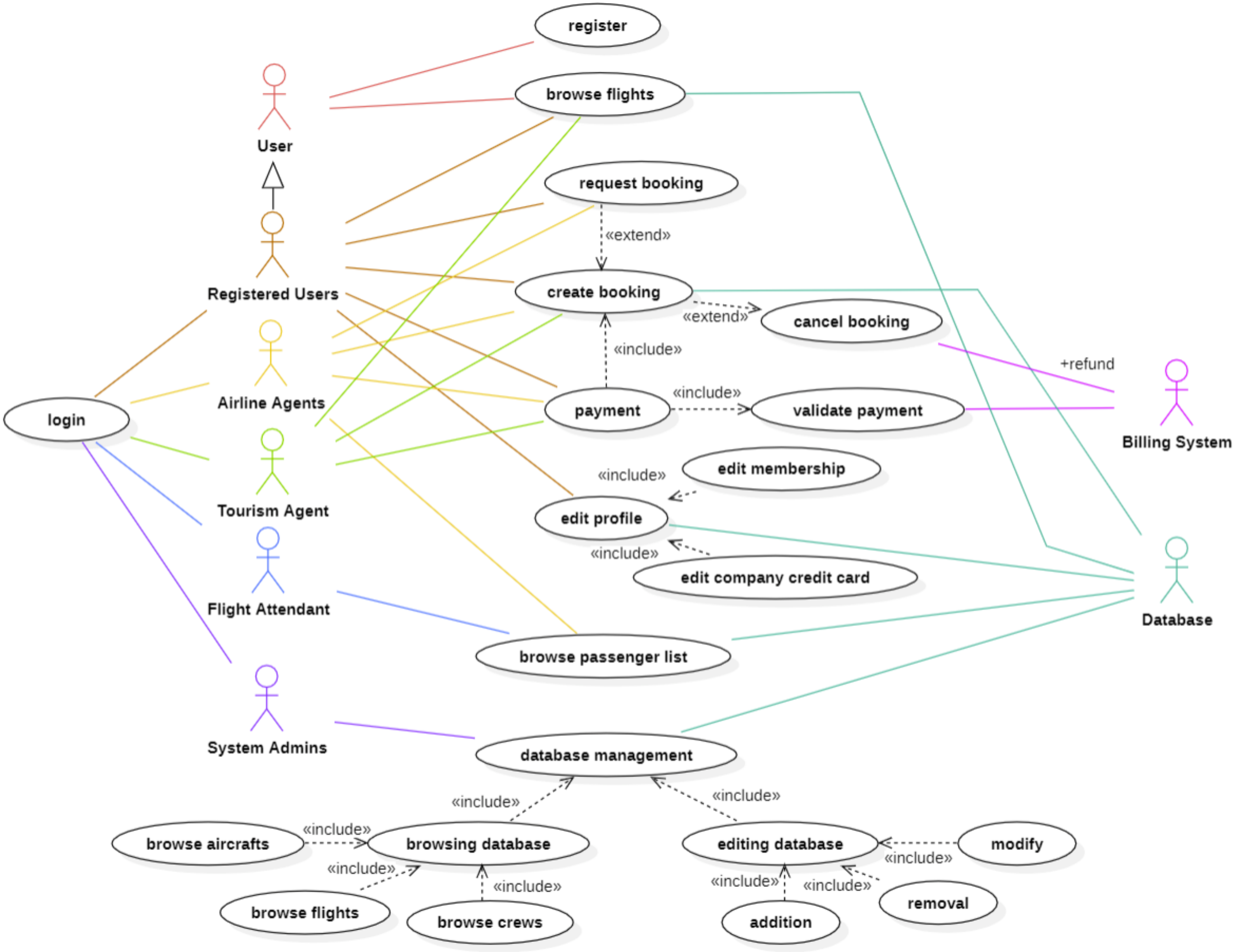
Textual Description:

Our system employs a client-server architecture. The client-side is a GUI, through which users interact with the system. Depending on their user type, they can perform various actions such as booking flights, managing profile information, and viewing passenger lists. The server-side is backed by a MySQL database that stores and manages all data, including user information, flight details, and bookings. Upon launching the program, users can browse flights as guests, with flight data retrieved from the MySQL database. To book flights, users must register and log in. The registration data is stored in the database, and login validation is performed by checking against this stored data. Registered users can book flights, with flight details and seat availability displayed on the GUI. Booking a flight triggers updates to the database. Different user types have different capabilities. Flight attendants can view flight and passenger details, airline agents can book flights based on user requests, tourism agents can browse and book flights, and admins have full control over the database.

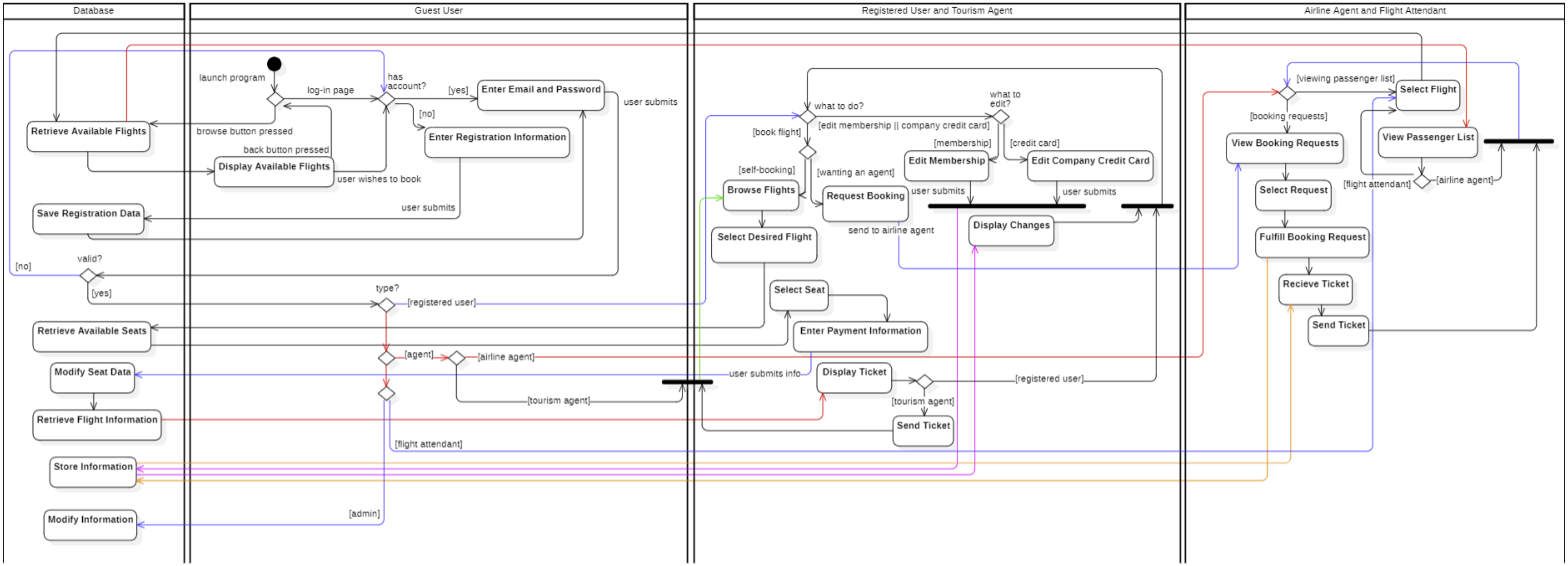
Graphical Interface:



System’s Use-Case Diagram:

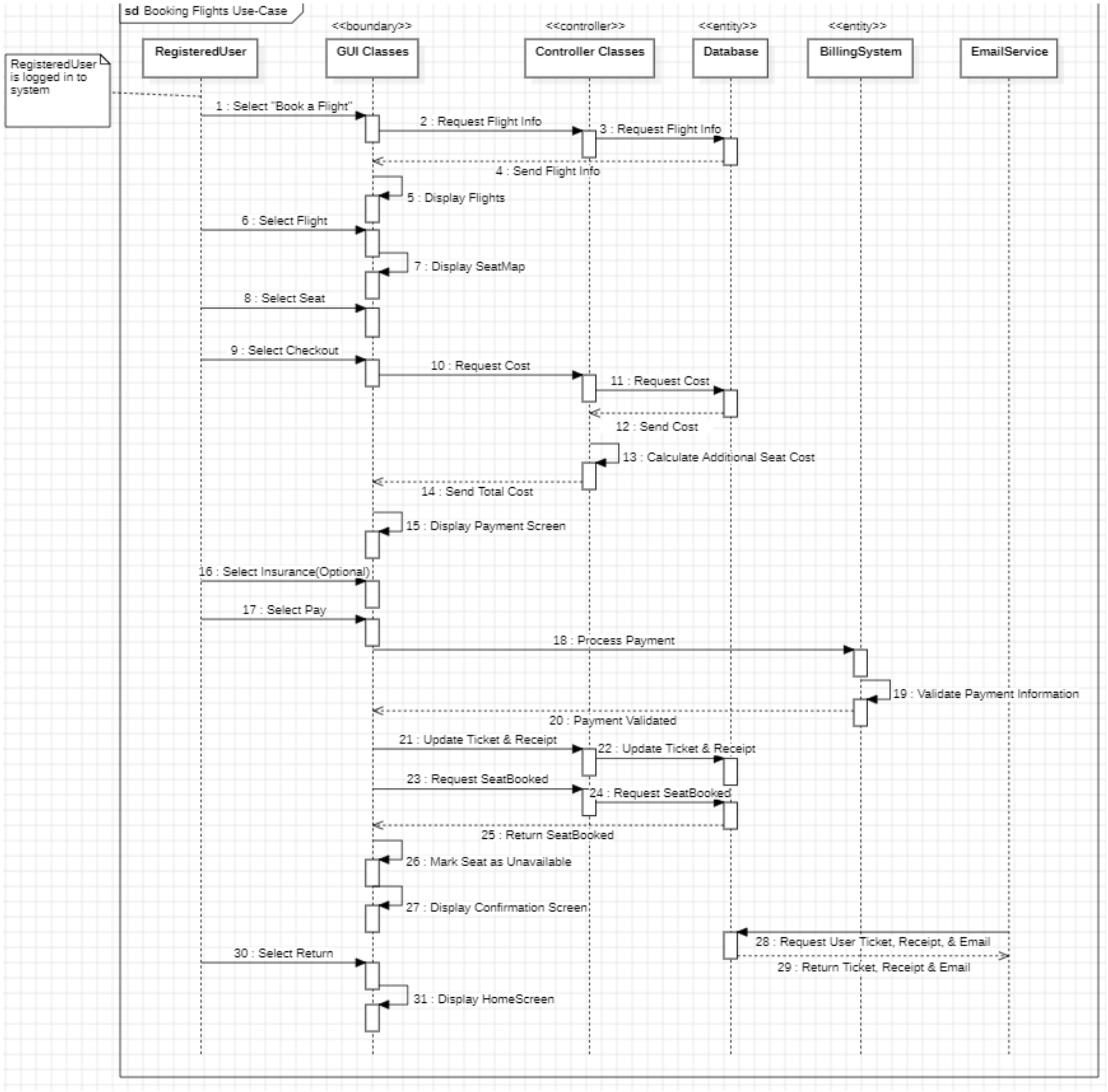


System’s Activity Diagram: *(Lines are coloured for Readability / Traceability)*

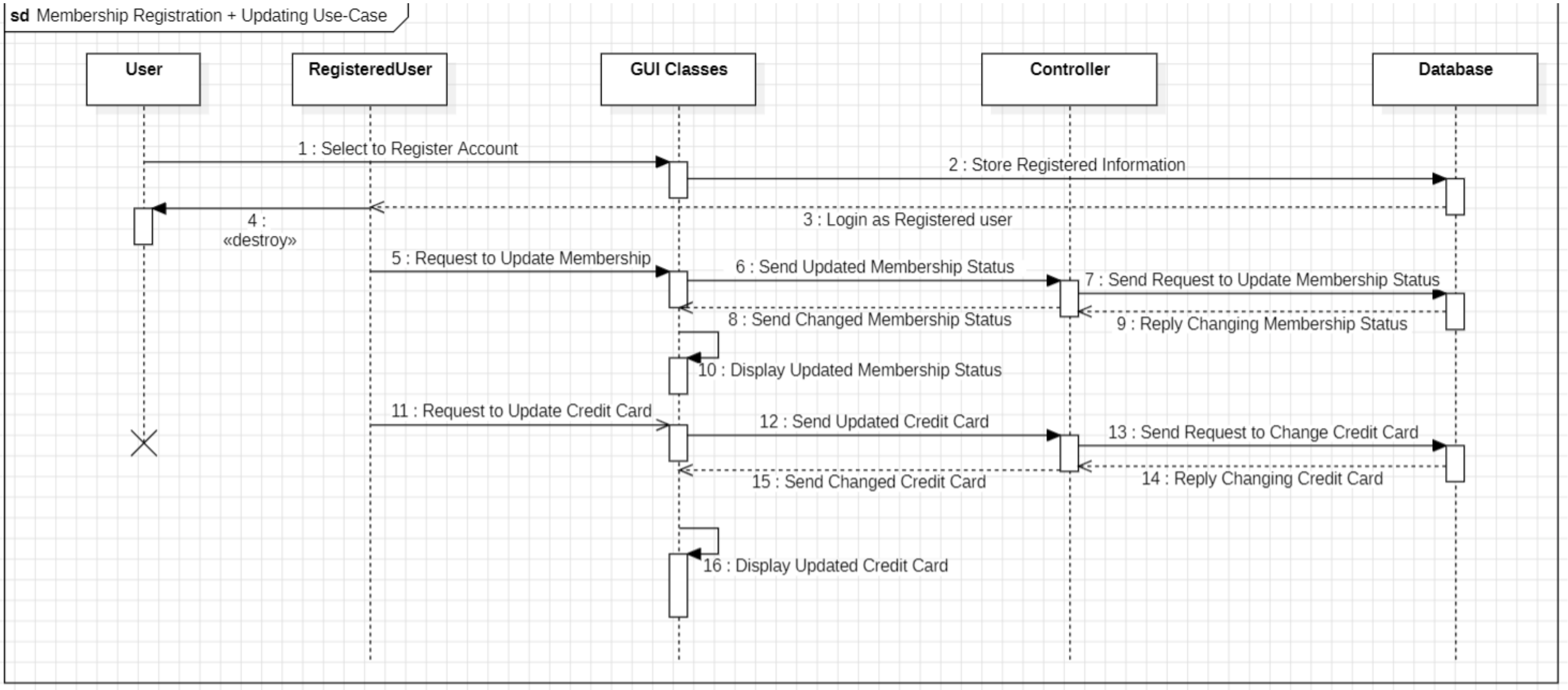


Selected Sequence Diagrams:

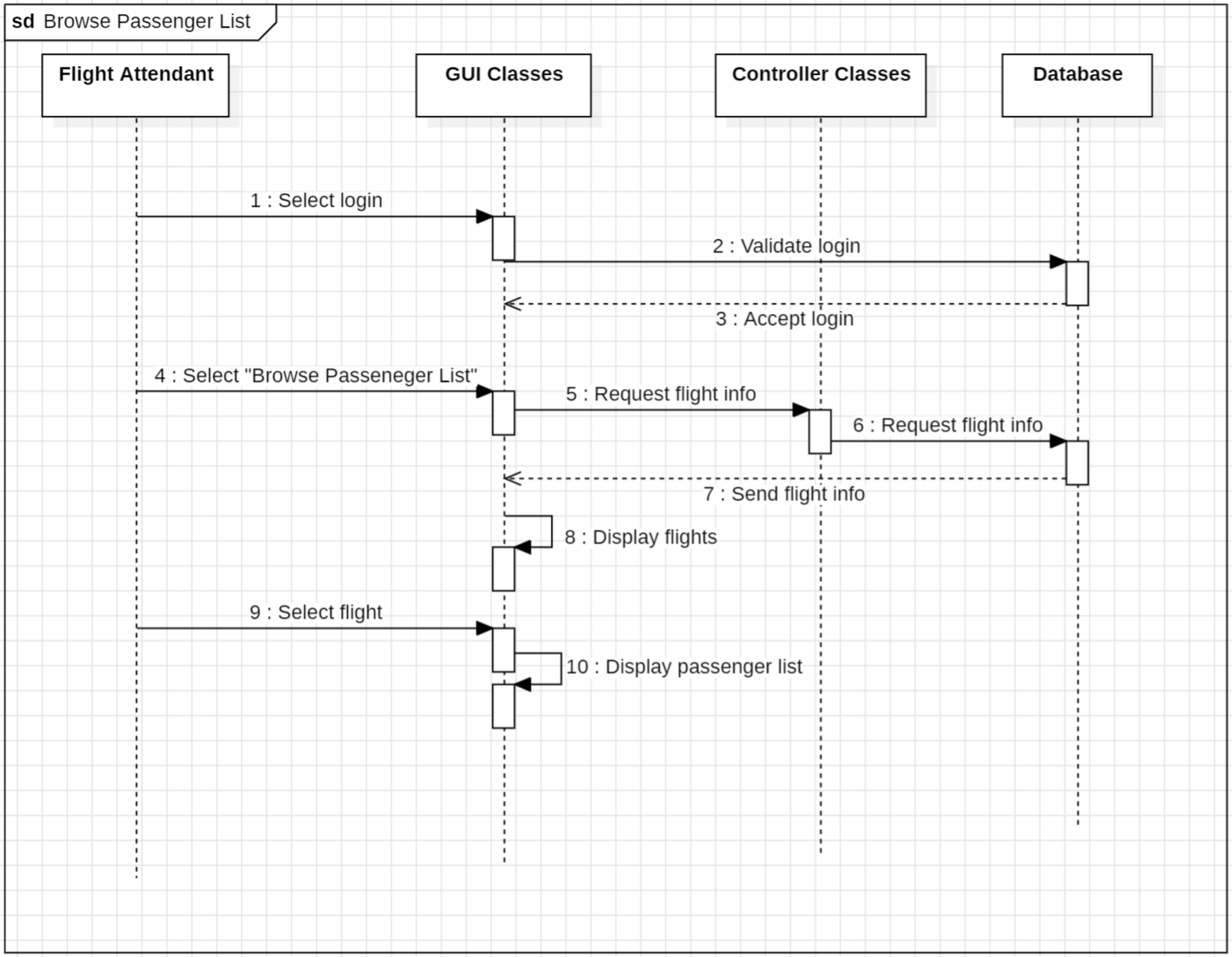
Booking Flights:



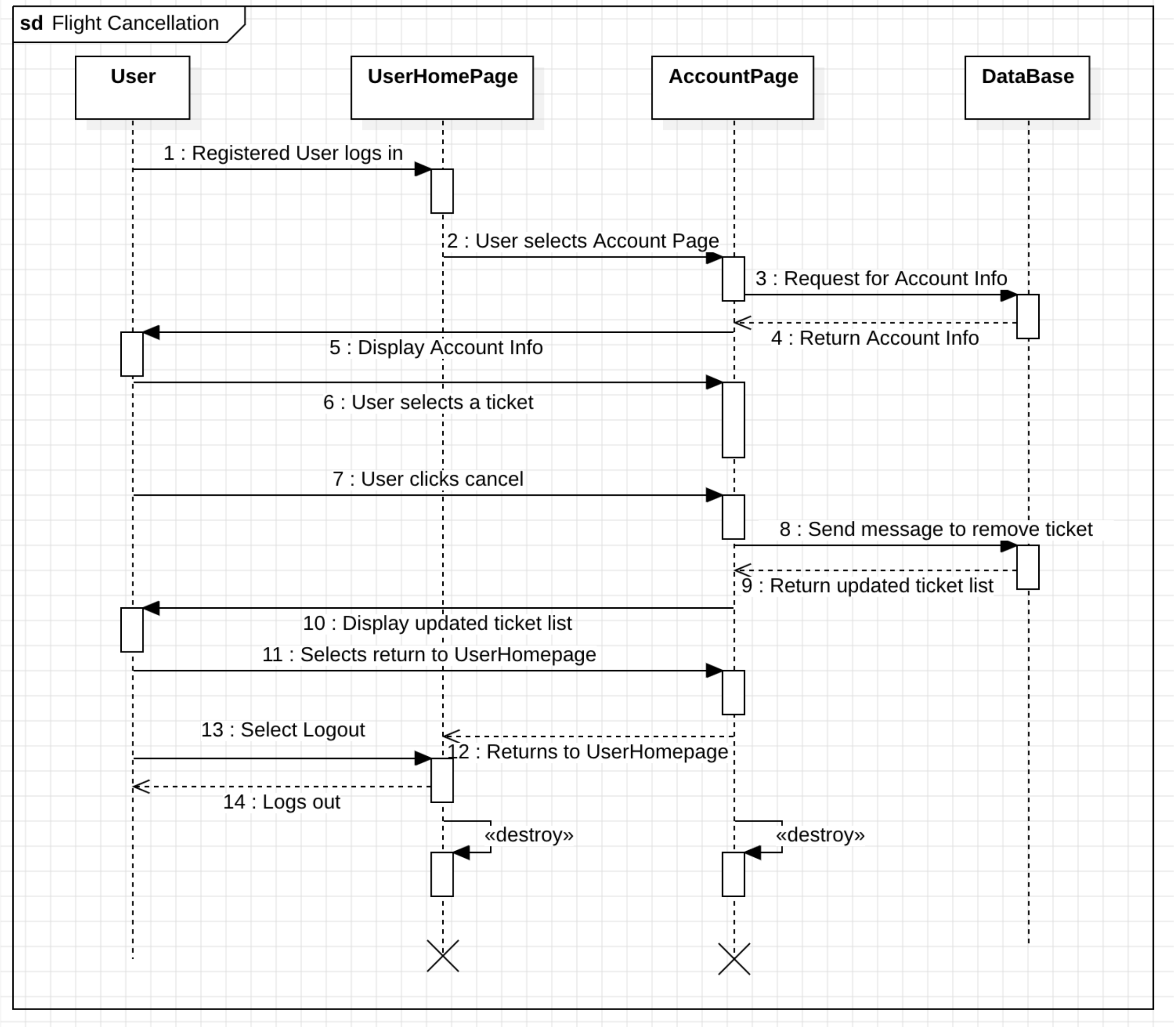
Creating a Registered User + Updating Membership and Credit-Card:


sd Membership Registration + Updating Use-Case
User
RegisteredUser
GUI Classes
Controller
Database
1 : Select to Register Account
2 : Store Registered Information
3 : Login as Registered user
4 :
«destroy»
5 : Request to Update Membership
6 : Send Updated Membership Status
7 : Send Request to Update Membership Status
8 : Send Changed Membership Status
9 : Reply Changing Membership Status
10 : Display Updated Membership Status
11 : Request to Update Credit Card
12 : Send Updated Credit Card
13 : Send Request to Change Credit Card
14 : Reply Changing Credit Card
15 : Send Changed Credit Card
16 : Display Updated Credit Card

Browsing Passenger List:

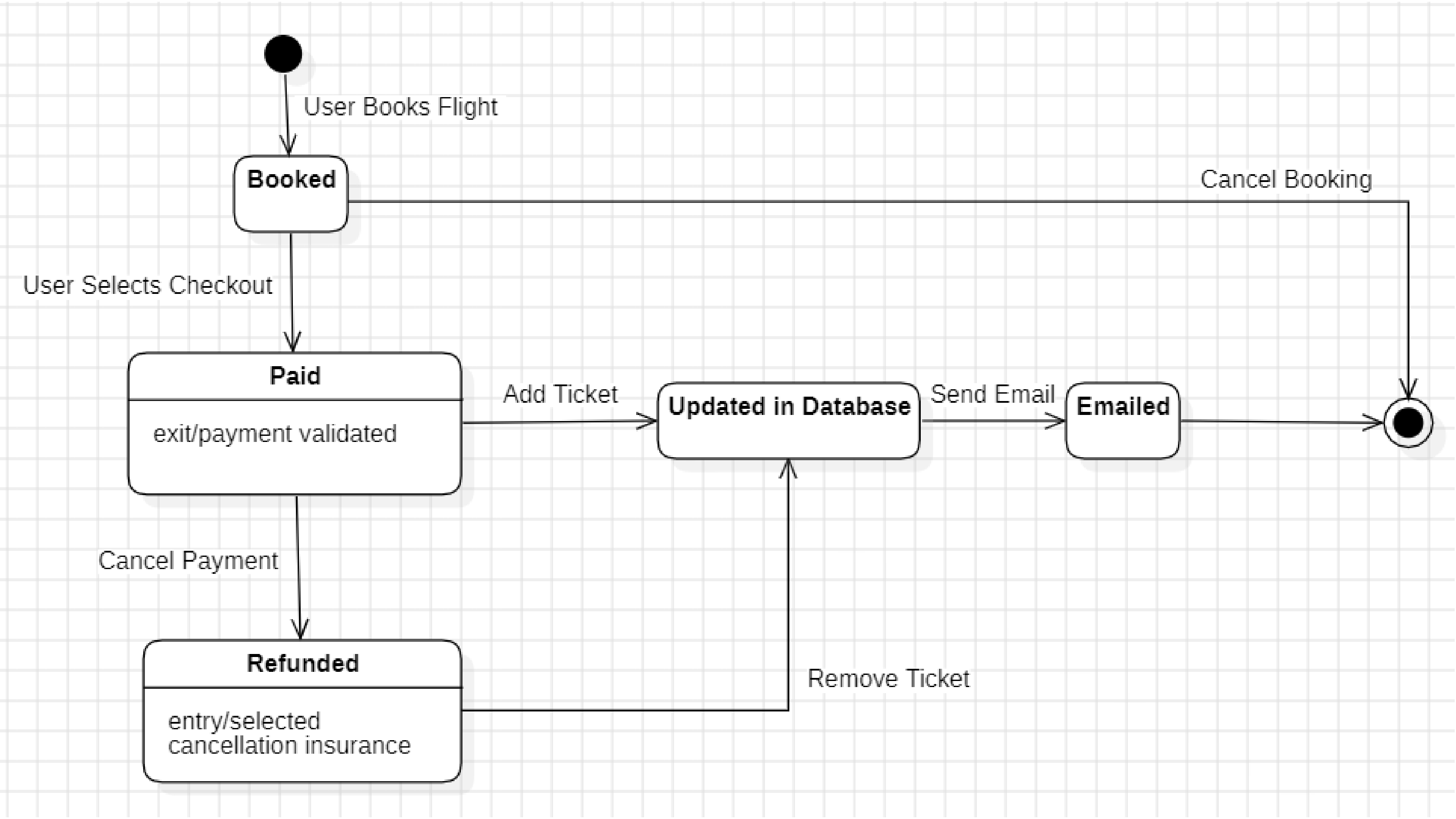


Cancelling a Ticket:

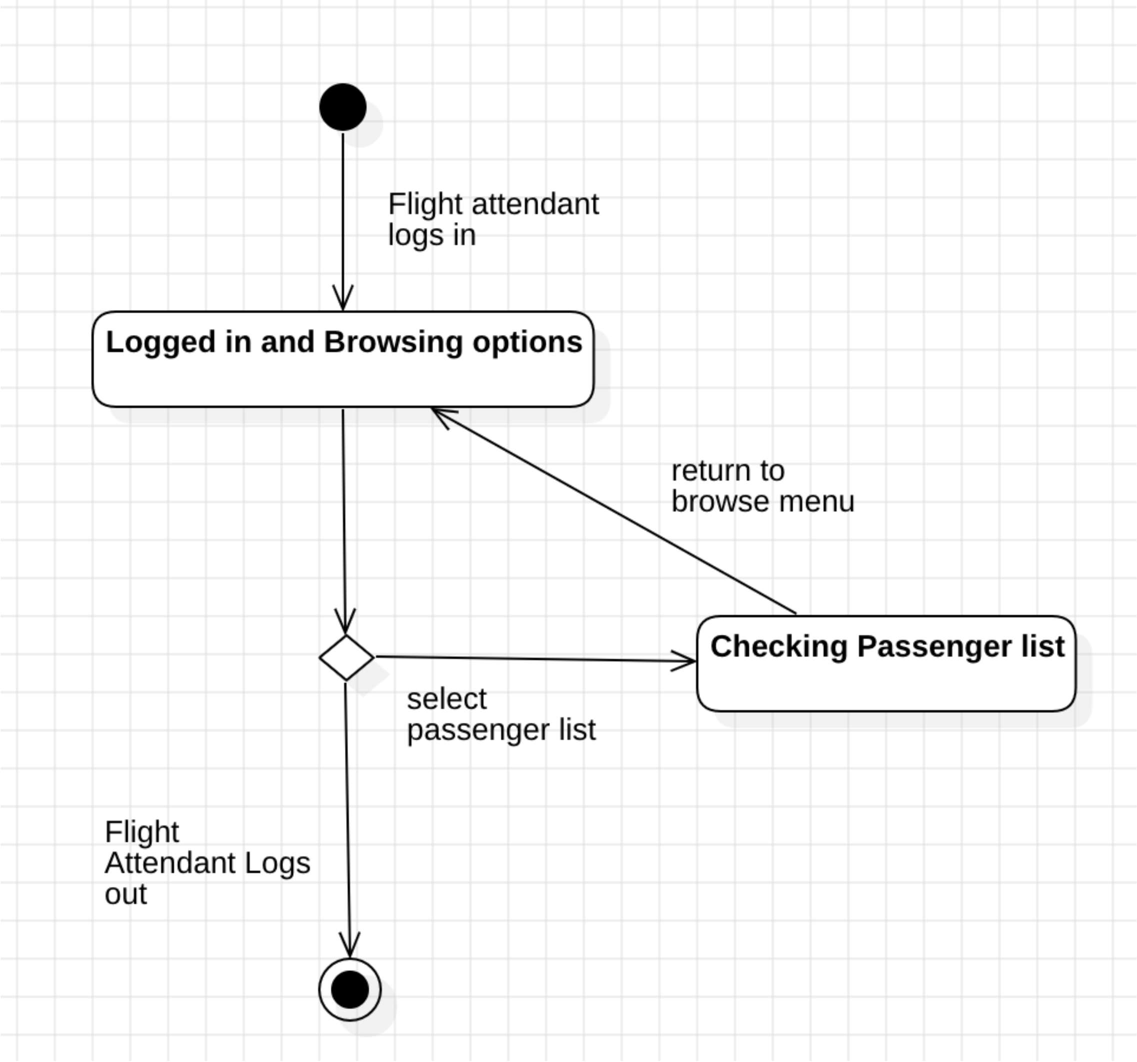

sd Flight Cancellation
User
UserHomePage
AccountPage
DataBase
1 : Registered User logs in
2 : User selects Account Page
3 : Request for Account Info
4 : Return Account Info
5 : Display Account Info
6 : User selects a ticket
7 : User clicks cancel
8 : Send message to remove ticket
9 : Return updated ticket list
10 : Display updated ticket list
11 : Selects return to UserHomepage
12 : Returns to UserHomepage
13 : Select Logout
14 : Logs out
«destroy»
«destroy»

Selected State Transition Diagrams:

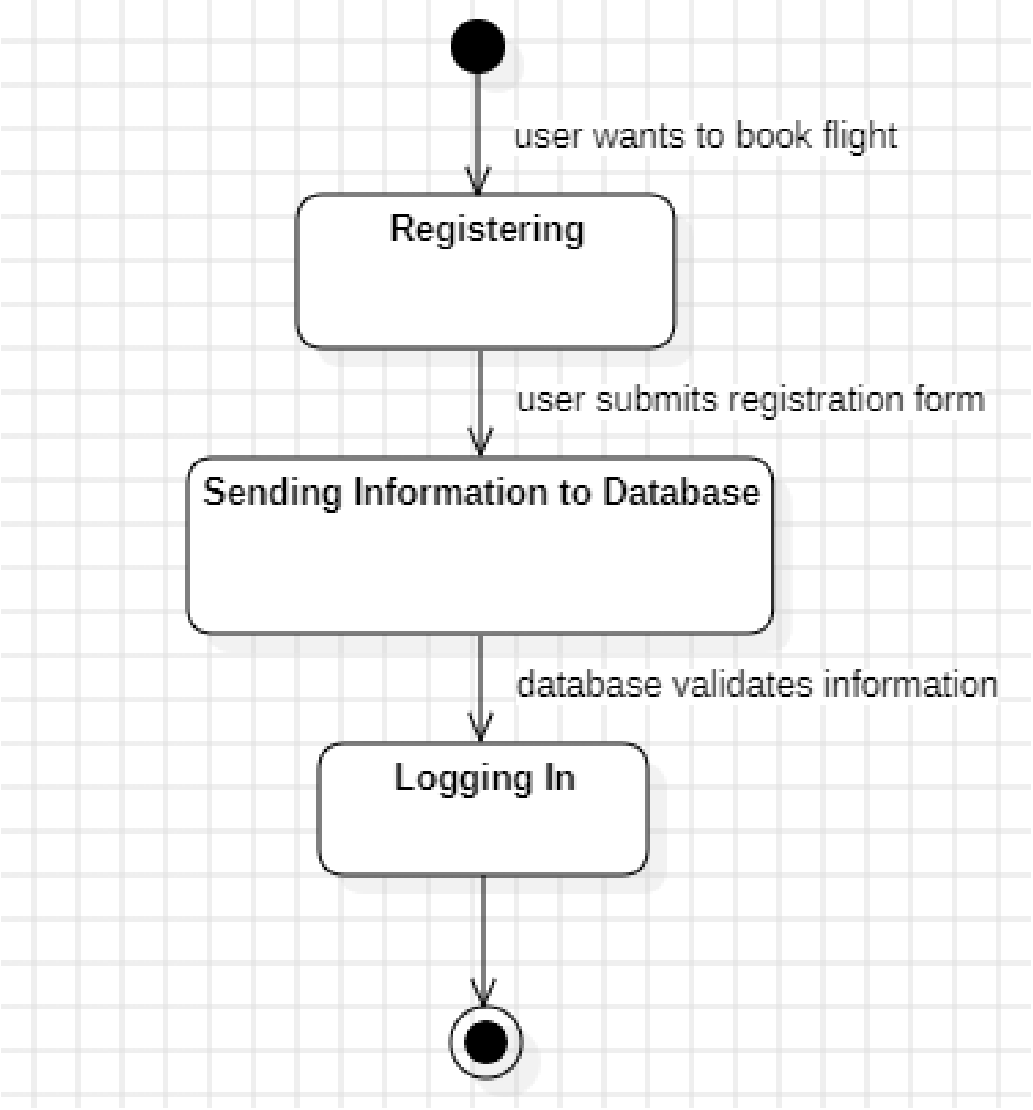
Booking a Flight:



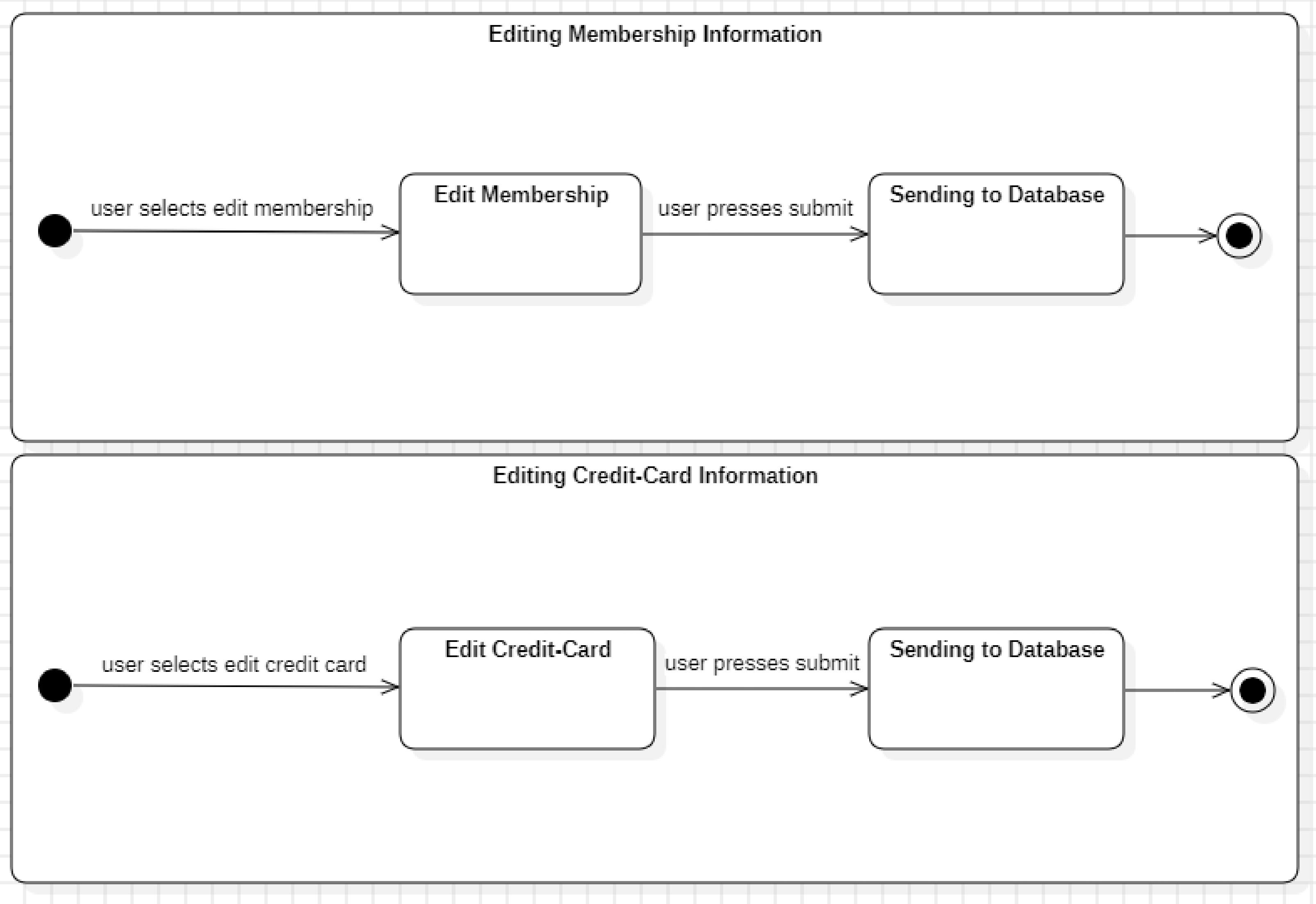
Flight Attendant Profile Page:



Creating a Registered User to Book a Flight:

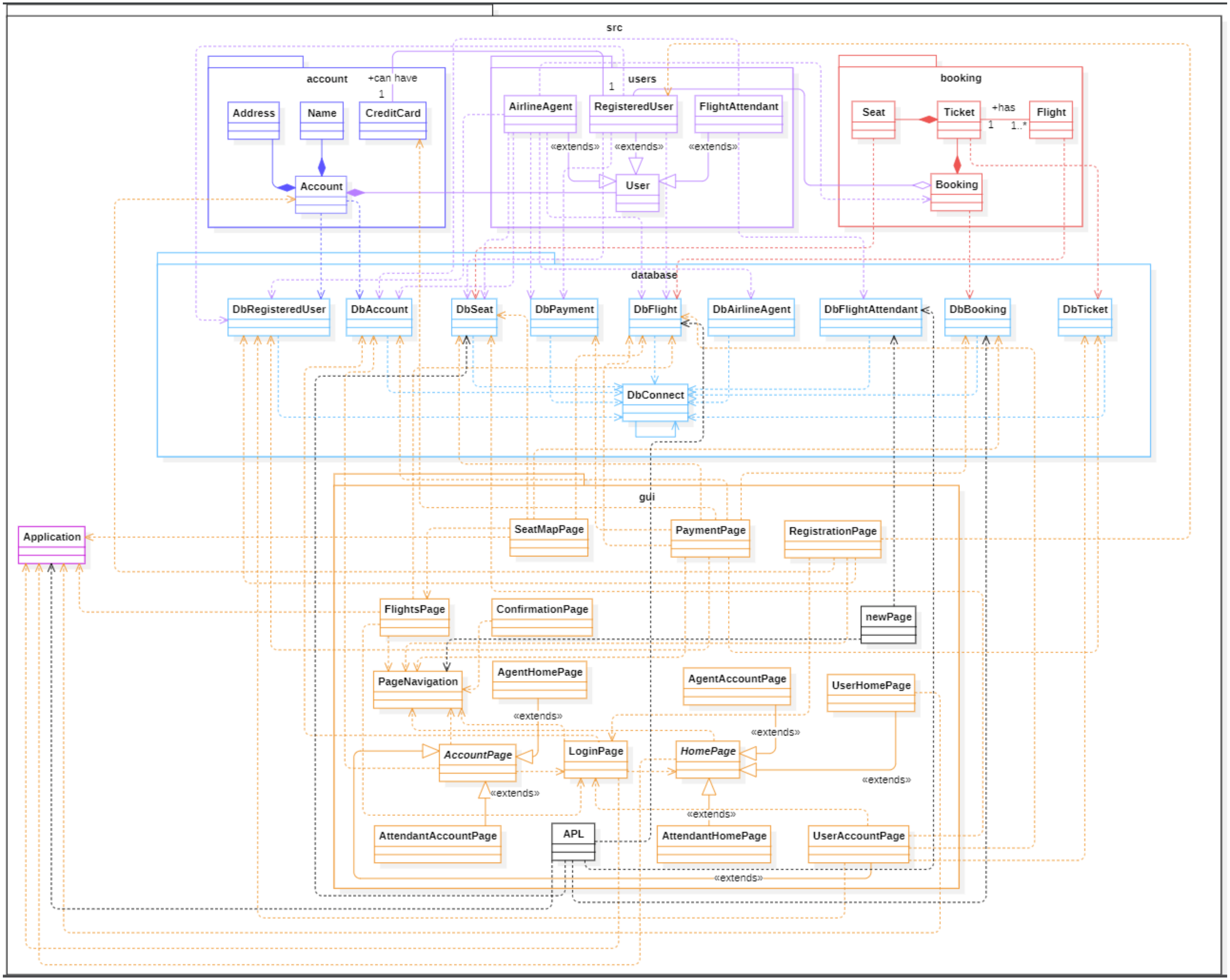

user wants to book flight
Registering
user submits registration form
Sending Information to Database
database validates information
Logging In

Editing Membership and Credit-Card Information:

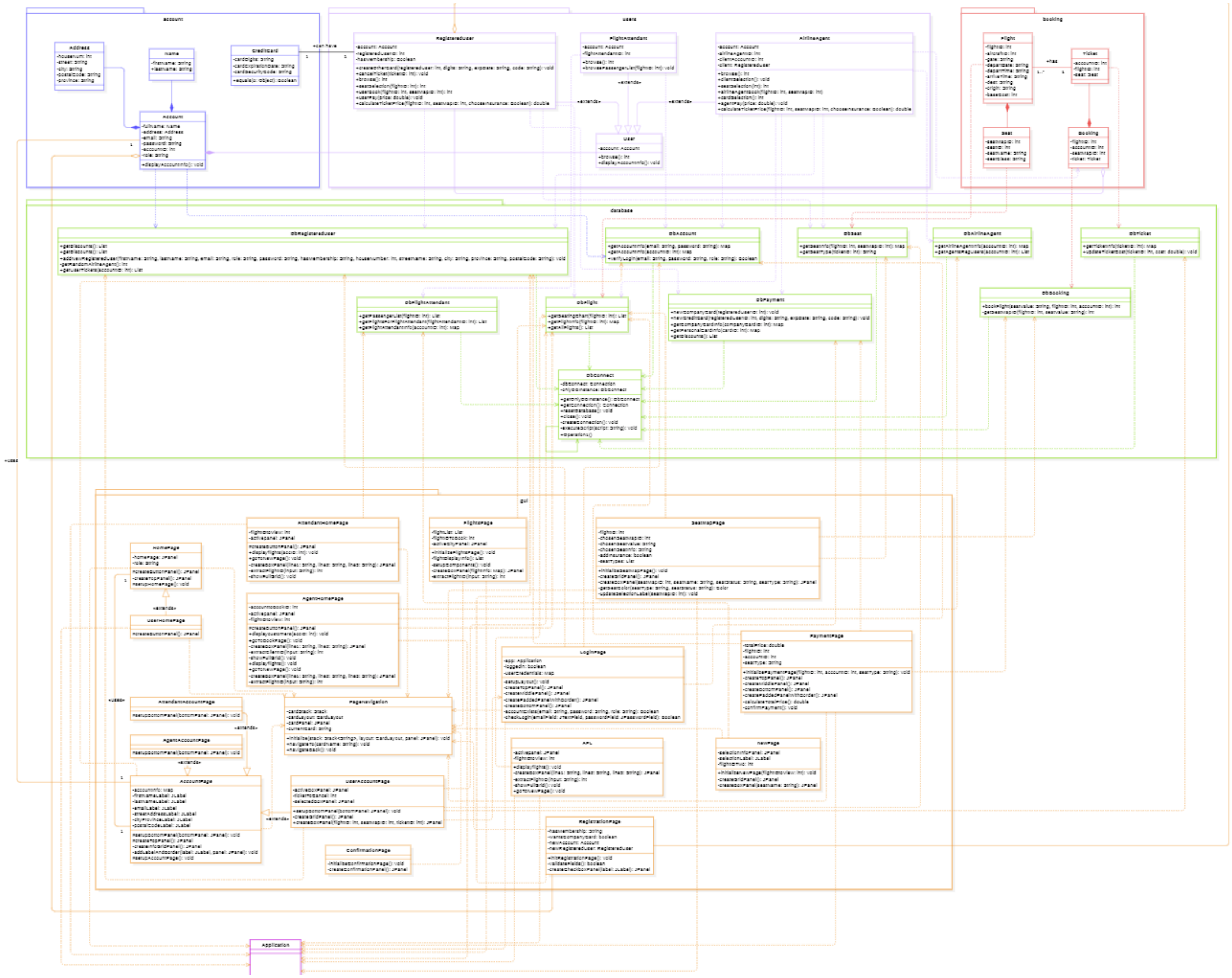


Editing Membership Information
user selects edit membership
Edit Membership
user presses submit
Sending to Database
Editing Credit-Card Information
user selects edit credit card
Edit Credit-Card
user presses submit
Sending to Database

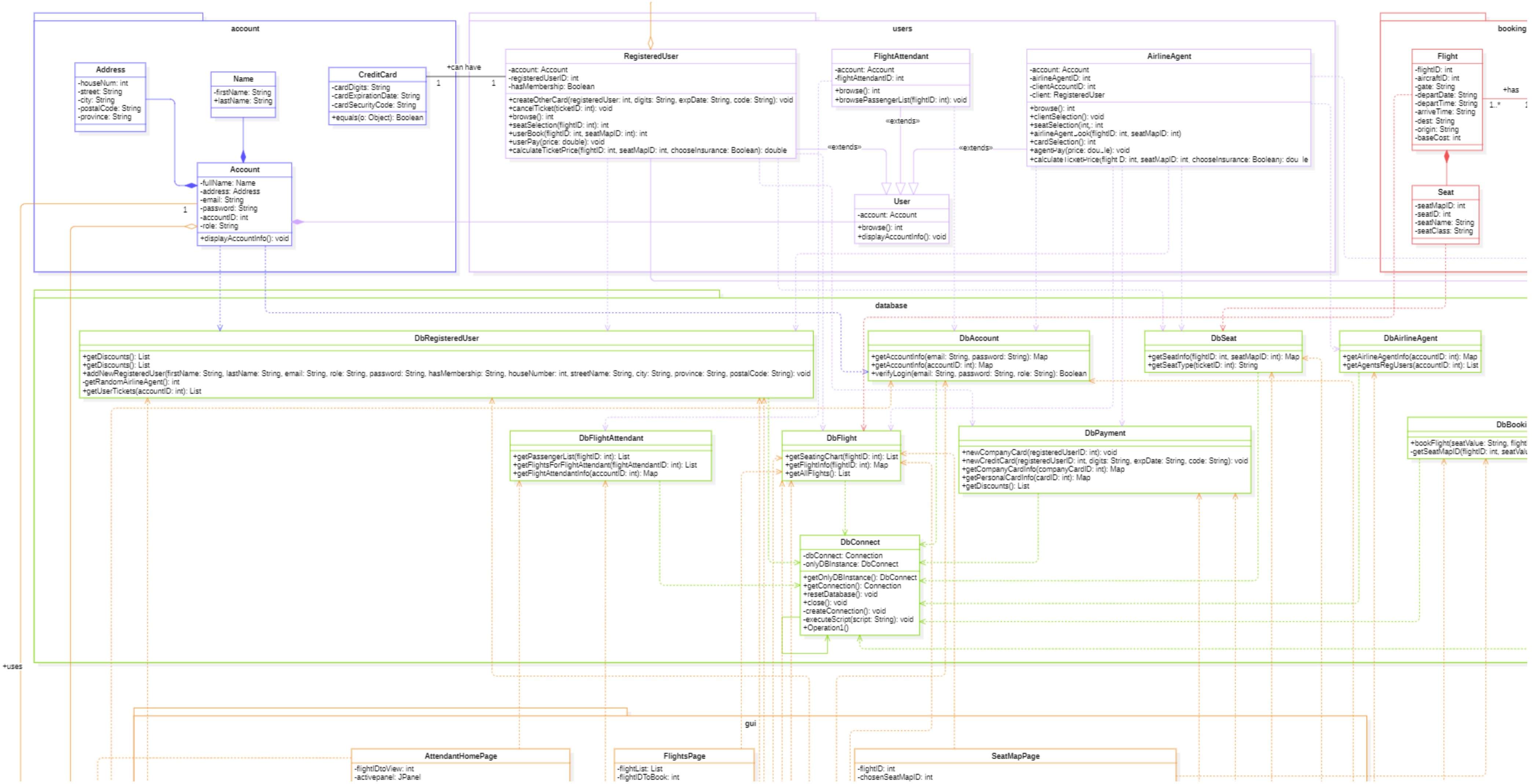
System’s Domain Class Diagram: *(Basic)*

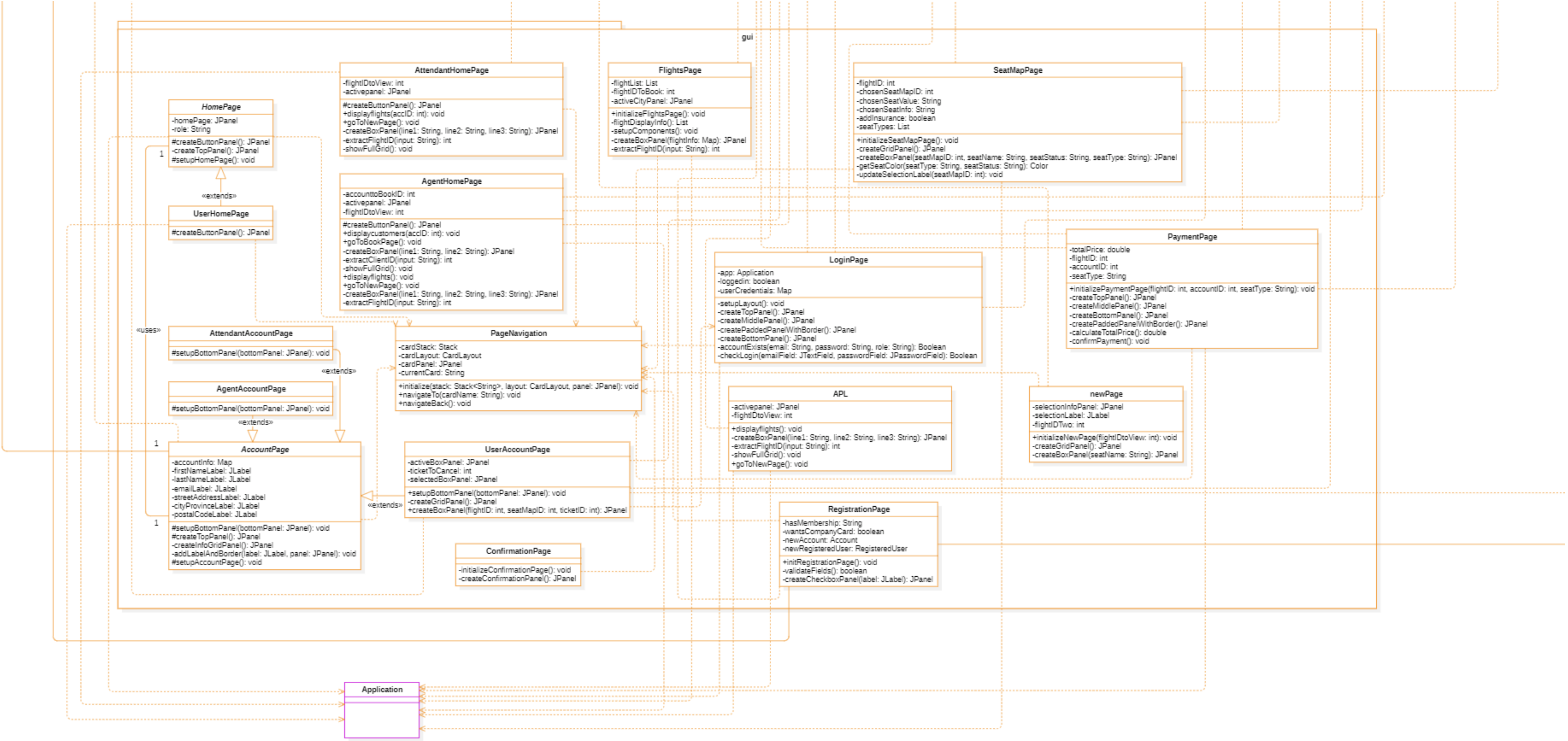


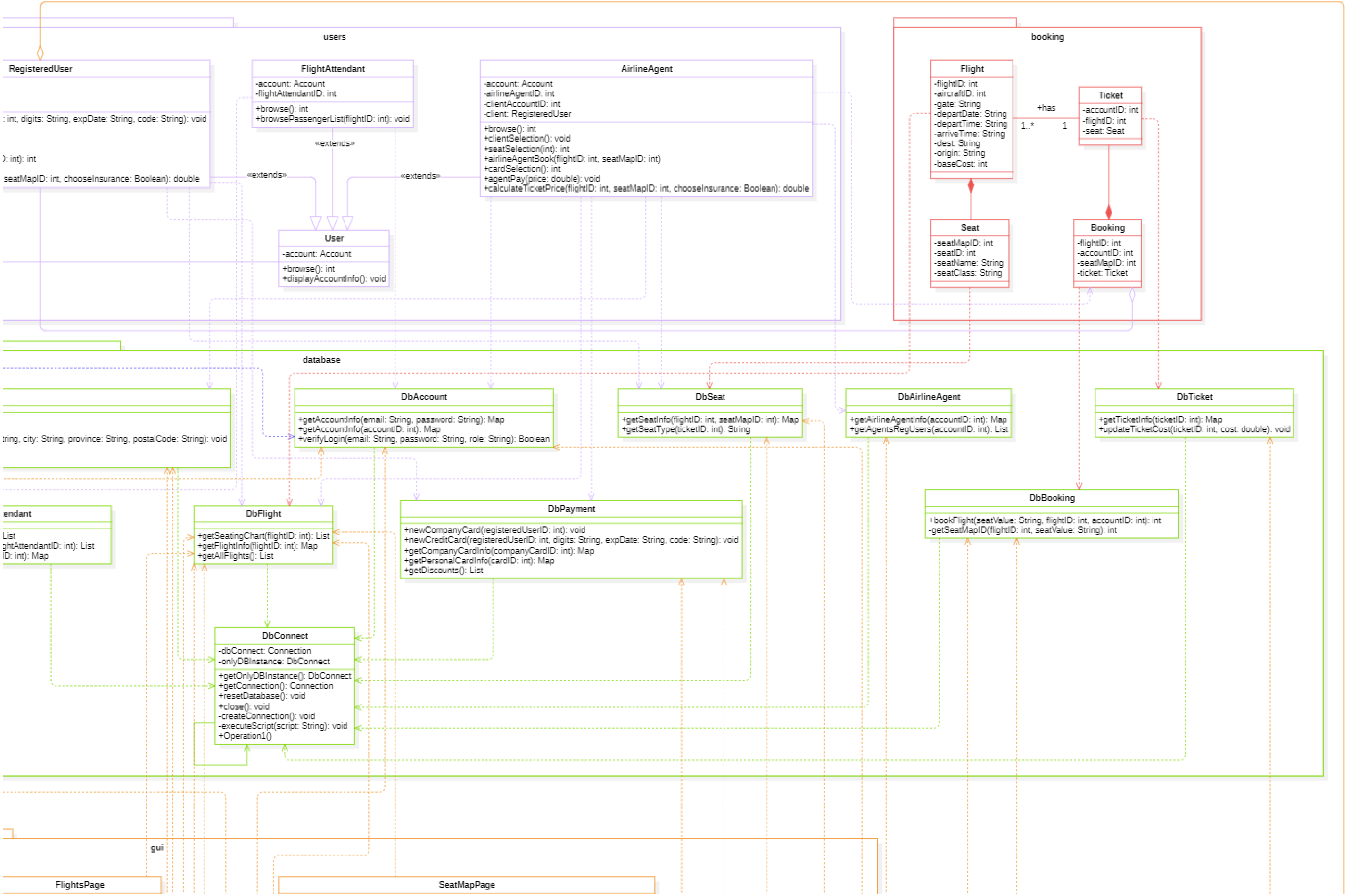
System's Domain Class Diagram: *(With Attributes and Functionalities)*


account
Address
-houseNum: int
-street: String
-city: String
-postalCode: String
-province: String
Name
-firstName: String
+lastName: String
CreditCard
-cardDigits: String
-cardExpirationDate: String
-cardSecurityCode: String
+equals(o: Object): boolean
+can have
1
1
Account
-fullName: Name
-address: Address
-email: String
-password: String
-accountID: int
-role: String
+displayAccountInfo(): void
users
RegisteredUser
-account: Account
-registeredUserID: int
-hasMembership: boolean
+createOtherCard(registeredUser: int, digits: String, expDate: String, code: String): void
+cancelTicket(ticketID: int): void
+browse(): int
+seatSelection(flightID: int): int
+userBook(flightID: int, seatMapID: int): int
+userPay(price: double): void
+calculateTicketPrice(flightID: int, seatMapID: int, chooseInsurance: boolean): double
FlightAttendant
-account: Account
-flightAttendantID: int
+browse(): int
+browsePassengerList(flightID: int): void
AirlineAgent
-account: Account
-airlineAgentID: int
-clientAccountID: int
-client: RegisteredUser
+browse(): int
+clientSelection(): void
+seatSelection(int): int
+airlineAgentBook(flightID: int, seatMapID: int)
+cardSelection(): int
+agentPay(price: double): void
+calculateTicketPrice(flightID: int, seatMapID: int, chooseInsurance: boolean): double
«extends»
«extends»
«extends»
User
-account: Account
+browse(): int
+displayAccountInfo(): void
booking
Flight
-flightID: int
-aircraftID: int
-gate: String
-departDate: String
-departTime: String
-arrivalTime: String
-dest: String
-origin: String
-baseCost: int
+has
1..*
1
Ticket
-accountID: int
-flightID: int
-seat: Seat
Seat
-seatMapID: int
-seatID: int
-seatName: String
-seatClass: String
Booking
-flightID: int
-accountID: int
-seatMapID: int
-ticket: Ticket
database
DBRegisteredUser
+getDiscounts(): List
+getDiscounts(): List
+addNewRegisteredUser(firstName: String, lastName: String, email: String, role: String, password: String, hasMembership: String, houseNumber: int, streetName: String, city: String, province: String, postalCode: String): void
-getRandomAirlineAgent(): int
+getUserTickets(accountID: int): List
DBAccount
+getAccountInfo(email: String, password: String): Map
+getAccountInfo(accountID: int): Map
+verifyLogin(email: String, password: String, role: String): boolean
DBSeat
+getSeatInfo(flightID: int, seatMapID: int): Map
+getSeatType(ticketID: int): String
DBAirlineAgent
+getAirlineAgentInfo(accountID: int): Map
+getAgentsRegUsers(accountID: int): List
DBTicket
+getTicketInfo(ticketID: int): Map
+updateTicketCost(ticketID: int, cost: double): void
DBFlightAttendant
+getPassengerList(flightID: int): List
+getFlightsForFlightAttendant(flightAttendantID: int): List
+getFlightAttendantInfo(accountID: int): Map
DBFlight
+getSeatingChart(flightID: int): List
+getFlightInfo(flightID: int): Map
+getAllFlights(): List
DBPayment
+newCompanyCard(registeredUserID: int): void
+newCreditCard(registeredUserID: int, digits: String, expDate: String, code: String): void
+getCompanyCardInfo(companyCardID: int): Map
+getPersonalCardInfo(cardID: int): Map
+getDiscounts(): List
DBBooking
+bookFlight(seatValue: String, flightID: int, accountID: int): int
-getSeatMapID(flightID: int, seatValue: String): int
DBConnect
-dbConnect: Connection
-onlyDBInstance: DBConnect
+getOnlyDBInstance(): DBConnect
+getConnection(): Connection
+resetDatabase(): void
+close(): void
-createConnection(): void
-executeScript(script: String): void
+Operation1()
+uses
gui
HomePage
-homePage: JPanel
-role: String
#createButtonPanel(): JPanel
-createTopPanel(): JPanel
#setupHomePage(): void
«extends»
UserHomePage
#createButtonPanel(): JPanel
AttendantHomePage
-flightToView: int
-activePanel: JPanel
#createButtonPanel(): JPanel
+displayFlights(accID: int): void
+goToViewPage(): void
-createBoxPanel(line1: String, line2: String, line3: String, line4: String): JPanel
-extractFlightID(input: String): int
-showFlightID(): void
AgentHomePage
-accountToBookID: int
-activePanel: JPanel
-flightToView: int
#createButtonPanel(): JPanel
+displayCustomers(accID: int): void
+goToBookPage(): void
-createBoxPanel(line1: String, line2: String): JPanel
-extractClientID(input: String): int
-showFlightID(): void
+displayFlights(): void
+goToViewPage(): void
-createBoxPanel(line1: String, line2: String, line3: String, line4: String): JPanel
-extractFlightID(input: String): int
FlightsPage
-flightList: List
-flightToBook: int
-activeFlightPanel: JPanel
+initializeFlightsPage(): void
-flightDisplayInfo(): void
-setupComponents(): void
-createBoxPanel(flightInfo: Map): JPanel
-extractFlightID(input: String): int
SeatMapPage
-flightID: int
-chosenSeatMapID: int
-chosenSeatValue: String
-chosenSeatInfo: String
-addInsurance: boolean
-seatTypes: List
+initializeSeatMapPage(): void
-createGridPanel(): JPanel
-createBoxPanel(seatMapID: int, seatName: String, seatStatus: String, seatType: String): JPanel
-getSeatColor(seatType: String, seatStatus: String): Color
-updateSelectionLabel(seatMapID: int): void
PaymentPage
-totalPrice: double
-flightID: int
-accountID: int
-seatType: String
+initializePaymentPage(flightID: int, accountID: int, seatType: String): void
-createTopPanel(): JPanel
-createMiddlePanel(): JPanel
-createBottomPanel(): JPanel
-createPaddedPanelWithBorder(): JPanel
-calculateTotalPrice(): double
-confirmPayment(): void
LoginPage
-app: Application
-loggedIn: boolean
-userCredentials: Map
-setupLayout(): void
-createTopPanel(): JPanel
-createMiddlePanel(): JPanel
-createPaddedPanelWithBorder(): JPanel
-createBottomPanel(): JPanel
-accountCreate(email: String, password: String, role: String): boolean
-checkLogin(emailField: JTextField, passwordField: JPasswordField): boolean
AttendantAccountPage
#setupBottomPanel(bottomPanel: JPanel): void
«uses»
AgentAccountPage
#setupBottomPanel(bottomPanel: JPanel): void
«extends»
«extends»
PageNavigation
-cardStack: Stack
-cardLayout: CardLayout
-cardPanel: JPanel
-currentCard: String
+initialize(stack: Stack<String>, layout: CardLayout, panel: JPanel): void
+navigateTo(cardName: String): void
+navigateBack(): void
AccountPage
-accountInfo: Map
-firstNameLabel: JLabel
-lastNameLabel: JLabel
-emailLabel: JLabel
-streetAddressLabel: JLabel
-cityProvinceLabel: JLabel
-postalCodeLabel: JLabel
#setupBottomPanel(bottomPanel: JPanel): void
#createTopPanel(): JPanel
-createInfoGridPanel(): JPanel
-addLabelAndBorder(label: JLabel, panel: JPanel): void
#setupAccountPage(): void
UserAccountPage
-activeBoxPanel: JPanel
-ticketToCancel: int
-selectedBoxPanel: JPanel
+setupBottomPanel(bottomPanel: JPanel): void
-createGridPanel(): JPanel
+createBoxPanel(flightID: int, seatMapID: int, ticketID: int): JPanel
«extends»
ConfirmationPage
-initializeConfirmationPage(): void
-createConfirmationPanel(): JPanel
APL
-activePanel: JPanel
-flightToView: int
+displayFlights(): void
-createBoxPanel(line1: String, line2: String, line3: String): JPanel
-extractFlightID(input: String): int
-showFlightID(): void
+goToViewPage(): void
NewPage
-selectionInfoPanel: JPanel
-selectionLabel: JLabel
-flightToView: int
+initializeNewPage(flightToView: int): void
-createGridPanel(): JPanel
-createBoxPanel(seatName: String): JPanel
RegistrationPage
-hasMembership: String
-wantsCompanyCard: boolean
-newAccount: Account
-newRegisteredUser: RegisteredUser
+initRegistrationPage(): void
-validateFields(): boolean
-createCheckBoxPanel(label: JLabel): JPanel
1
1
1
1
Application

Close-Up Pictures:

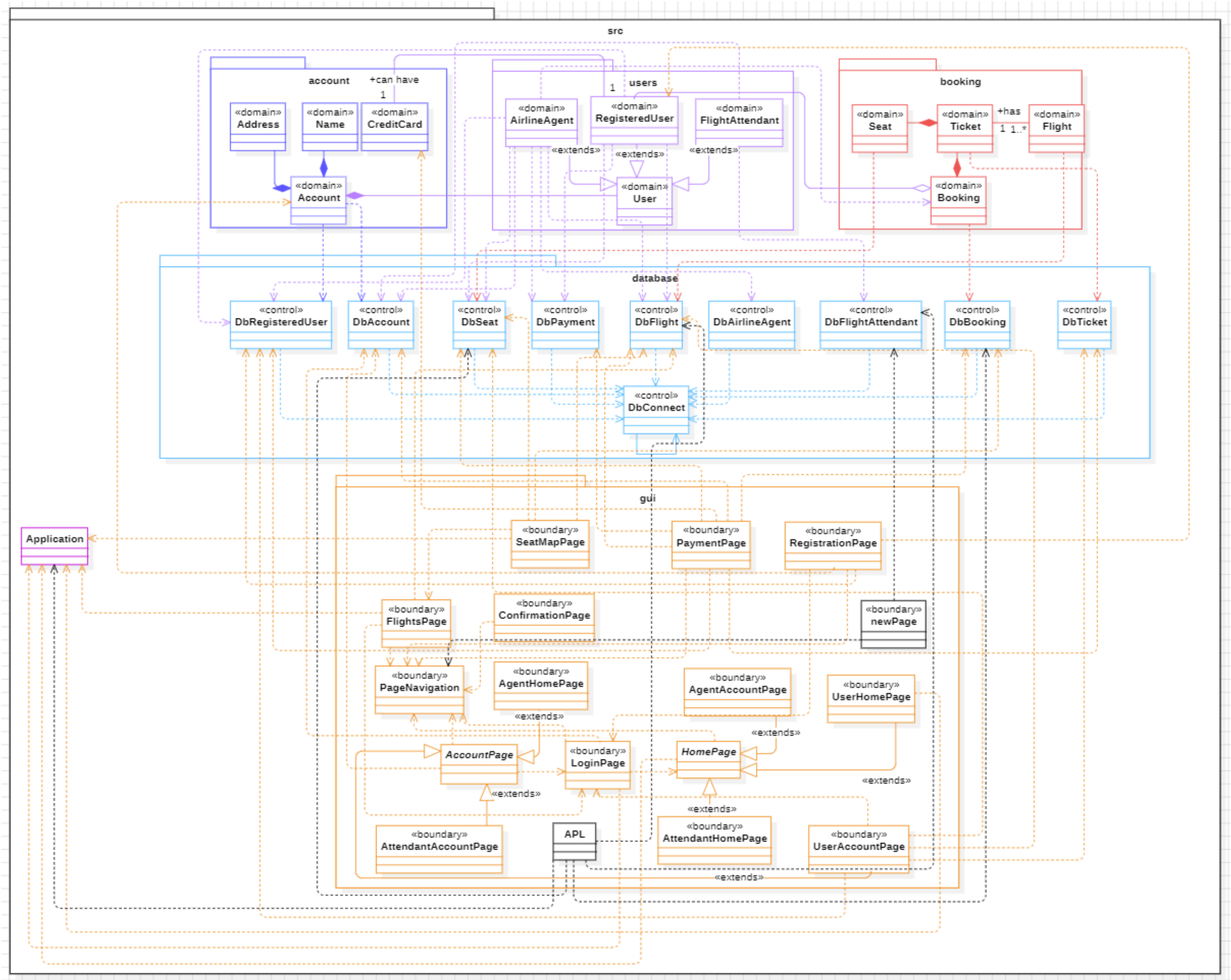

account
Address
-houseNum: int
-street: String
-city: String
-postalCode: String
-province: String
Name
-firstName: String
+lastName: String
CreditCard
-cardDigits: String
-cardExpirationDate: String
-cardSecurityCode: String
+equals(o: Object): Boolean
+can have
1
1
Account
-fullName: Name
-address: Address
-email: String
-password: String
-accountID: int
-role: String
+displayAccountInfo(): void
1
users
RegisteredUser
-account: Account
-registeredUserID: int
-hasMembership: Boolean
+createOtherCard(registeredUser: int, digits: String, expDate: String, code: String): void
+cancelTicket(ticketID: int): void
+browse(): int
+seatSelection(flightID: int): int
+userBook(flightID: int, seatMapID: int): int
+userPay(price: double): void
+calculateTicketPrice(flightID: int, seatMapID: int, chooseInsurance: Boolean): double
FlightAttendant
-account: Account
-flightAttendantID: int
+browse(): int
+browsePassengerList(flightID: int): void
«extends»
«extends»
«extends»
User
-account: Account
+browse(): int
+displayAccountInfo(): void
AirlineAgent
-account: Account
-airlineAgentID: int
-clientAccountID: int
-client: RegisteredUser
+browse(): int
+clientSelection(): void
+seatSelection(int,: int
+airlineAgent⌐ook(flightID: int, seatMapID: int)
+cardSelection(): int
+agentPay(price: dou_le): void
+calculateTicketPrice(flight D: int, seatMapID: int, chooseInsurance: Boolean): dou le
booking
Flight
-flightID: int
-aircraftID: int
-gate: String
-departDate: String
-departTime: String
-arriveTime: String
-dest: String
-origin: String
-baseCost: int
+has
1..*
1
Seat
-seatMapID: int
-seatID: int
-seatName: String
-seatClass: String
database
DbRegisteredUser
+getDiscounts(): List
+getDiscounts(): List
+addNewRegisteredUser(firstName: String, lastName: String, email: String, role: String, password: String, hasMembership: String, houseNumber: int, streetName: String, city: String, province: String, postalCode: String): void
-getRandomAirlineAgent(): int
+getUserTickets(accountID: int): List
DbAccount
+getAccountInfo(email: String, password: String): Map
+getAccountInfo(accountID: int): Map
+verifyLogin(email: String, password: String, role: String): Boolean
DbSeat
+getSeatInfo(flightID: int, seatMapID: int): Map
+getSeatType(ticketID: int): String
DbAirlineAgent
+getAirlineAgentInfo(accountID: int): Map
+getAgentsRegUsers(accountID: int): List
DbFlightAttendant
+getPassengerList(flightID: int): List
+getFlightsForFlightAttendant(flightAttendantID: int): List
+getFlightAttendantInfo(accountID: int): Map
DbFlight
+getSeatingChart(flightID: int): List
+getFlightInfo(flightID: int): Map
+getAllFlights(): List
DbPayment
+newCompanyCard(registeredUserID: int): void
+newCreditCard(registeredUserID: int, digits: String, expDate: String, code: String): void
+getCompanyCardInfo(companyCardID: int): Map
+getPersonalCardInfo(cardID: int): Map
+getDiscounts(): List
DbBooki
+bookFlight(seatValue: String, flight
-getSeatMapID(flightID: int, seatValu
DbConnect
-dbConnect: Connection
-onlyDBInstance: DbConnect
+getOnlyDBInstance(): DbConnect
+getConnection(): Connection
+resetDatabase(): void
+close(): void
-createConnection(): void
-executeScript(script: String): void
+Operation1()
+uses
gui
AttendantHomePage
-flightIDtoView: int
-activepanel: JPanel
FlightsPage
-flightList: List
-flightIDToBook: int
SeatMapPage
-flightID: int
-chosenSeatMapID: int


gui
HomePage
-homePage: JPanel
-role: String
#createButtonPanel(): JPanel
-createTopPanel(): JPanel
#setupHomePage(): void
«extends»
UserHomePage
#createButtonPanel(): JPanel
AttendantHomePage
-flightIDtoView: int
-activepanel: JPanel
#createButtonPanel(): JPanel
+displayflights(accID: int): void
+goToNewPage(): void
-createBoxPanel(line1: String, line2: String, line3: String): JPanel
-extractFlightID(input: String): int
-showFullGrid(): void
AgentHomePage
-accounttoBookID: int
-activepanel: JPanel
-flightIDtoView: int
#createButtonPanel(): JPanel
+displaycustomers(accID: int): void
+goToBookPage(): void
-createBoxPanel(line1: String, line2: String): JPanel
-extractClientID(input: String): int
-showFullGrid(): void
+displayflights(): void
+goToNewPage(): void
-createBoxPanel(line1: String, line2: String, line3: String): JPanel
-extractFlightID(input: String): int
FlightsPage
-flightList: List
-flightIDToBook: int
-activeCityPanel: JPanel
+initializeFlightsPage(): void
-flightDisplayInfo(): List
-setupComponents(): void
-createBoxPanel(flightInfo: Map): JPanel
-extractFlightID(input: String): int
SeatMapPage
-flightID: int
-chosenSeatMapID: int
-chosenSeatValue: String
-chosenSeatInfo: String
-addInsurance: boolean
-seatTypes: List
+initializeSeatMapPage(): void
-createGridPanel(): JPanel
-createBoxPanel(seatMapID: int, seatName: String, seatStatus: String, seatType: String): JPanel
-getSeatColor(seatType: String, seatStatus: String): Color
-updateSelectionLabel(seatMapID: int): void
PaymentPage
-totalPrice: double
-flightID: int
-accountID: int
-seatType: String
+initializePaymentPage(flightID: int, accountID: int, seatType: String): void
-createTopPanel(): JPanel
-createMiddlePanel(): JPanel
-createBottomPanel(): JPanel
-createPaddedPanelWithBorder(): JPanel
-calculateTotalPrice(): double
-confirmPayment(): void
LoginPage
-app: Application
-loggedin: boolean
-userCredentials: Map
-setupLayout(): void
-createTopPanel(): JPanel
-createMiddlePanel(): JPanel
-createPaddedPanelWithBorder(): JPanel
-createBottomPanel(): JPanel
-accountExists(email: String, password: String, role: String): Boolean
-checkLogin(emailField: JTextField, passwordField: JPasswordField): Boolean
«uses»
1
AttendantAccountPage
#setupBottomPanel(bottomPanel: JPanel): void
«extends»
AgentAccountPage
#setupBottomPanel(bottomPanel: JPanel): void
«extends»
PageNavigation
-cardStack: Stack
-cardLayout: CardLayout
-cardPanel: JPanel
-currentCard: String
+initialize(stack: Stack<String>, layout: CardLayout, panel: JPanel): void
+navigateTo(cardName: String): void
+navigateBack(): void
APL
-activepanel: JPanel
-flightIDtoView: int
+displayflights(): void
-createBoxPanel(line1: String, line2: String, line3: String): JPanel
-extractFlightID(input: String): int
-showFullGrid(): void
+goToNewPage(): void
newPage
-selectionInfoPanel: JPanel
-selectionLabel: JLabel
-flightIDTwo: int
+initializeNewPage(flightIDtoView: int): void
-createGridPanel(): JPanel
-createBoxPanel(seatName: String): JPanel
1
AccountPage
-accountInfo: Map
-firstNameLabel: JLabel
-lastNameLabel: JLabel
-emailLabel: JLabel
-streetAddressLabel: JLabel
-cityProvinceLabel: JLabel
-postalCodeLabel: JLabel
1
#setupBottomPanel(bottomPanel: JPanel): void
#createTopPanel(): JPanel
-createInfoGridPanel(): JPanel
-addLabelAndBorder(label: JLabel, panel: JPanel): void
#setupAccountPage(): void
«extends»
UserAccountPage
-activeBoxPanel: JPanel
-ticketToCancel: int
-selectedBoxPanel: JPanel
+setupBottomPanel(bottomPanel: JPanel): void
-createGridPanel(): JPanel
+createBoxPanel(flightID: int, seatMapID: int, ticketID: int): JPanel
ConfirmationPage
-initializeConfirmationPage(): void
-createConfirmationPanel(): JPanel
RegistrationPage
-hasMembership: String
-wantsCompanyCard: boolean
-newAccount: Account
-newRegisteredUser: RegisteredUser
+initRegistrationPage(): void
-validateFields(): boolean
-createCheckboxPanel(label: JLabel): JPanel
Application


users
RegisteredUser
: int, digits: String, expDate: String, code: String): void
): int): int
seatMapID: int, chooseInsurance: Boolean): double
FlightAttendant
-account: Account
-flightAttendantID: int
+browse(): int
+browsePassengerList(flightID: int): void
«extends»
«extends»
«extends»
AirlineAgent
-account: Account
-airlineAgentID: int
-clientAccountID: int
-client: RegisteredUser
+browse(): int
+clientSelection(): void
+seatSelection(int): int
+airlineAgentBook(flightID: int, seatMapID: int)
+cardSelection(): int
+agentPay(price: double): void
+calculateTicketPrice(flightID: int, seatMapID: int, chooseInsurance: Boolean): double
User
-account: Account
+browse(): int
+displayAccountInfo(): void
booking
Flight
-flightID: int
-aircraftID: int
-gate: String
-departDate: String
-departTime: String
-arriveTime: String
-dest: String
-origin: String
-baseCost: int
+has
1..*
1
Ticket
-accountID: int
-flightID: int
-seat: Seat
Seat
-seatMapID: int
-seatID: int
-seatName: String
-seatClass: String
Booking
-flightID: int
-accountID: int
-seatMapID: int
-ticket: Ticket
database
tring, city: String, province: String, postalCode: String): void
DbAccount
+getAccountInfo(email: String, password: String): Map
+getAccountInfo(accountID: int): Map
+verifyLogin(email: String, password: String, role: String): Boolean
DbSeat
+getSeatInfo(flightID: int, seatMapID: int): Map
+getSeatType(ticketID: int): String
DbAirlineAgent
+getAirlineAgentInfo(accountID: int): Map
+getAgentsRegUsers(accountID: int): List
DbTicket
+getTicketInfo(ticketID: int): Map
+updateTicketCost(ticketID: int, cost: double): void
endant
List
ghtAttendantID: int): List
ID: int): Map
DbFlight
+getSeatingChart(flightID: int): List
+getFlightInfo(flightID: int): Map
+getAllFlights(): List
DbPayment
+newCompanyCard(registeredUserID: int): void
+newCreditCard(registeredUserID: int, digits: String, expDate: String, code: String): void
+getCompanyCardInfo(companyCardID: int): Map
+getPersonalCardInfo(cardID: int): Map
+getDiscounts(): List
DbBooking
+bookFlight(seatValue: String, flightID: int, accountID: int): int
-getSeatMapID(flightID: int, seatValue: String): int
DbConnect
-dbConnect: Connection
-onlyDBInstance: DbConnect
+getOnlyDBInstance(): DbConnect
+getConnection(): Connection
+resetDatabase(): void
+close(): void
-createConnection(): void
-executeScript(script: String): void
+Operation1()
gui
FlightsPage
SeatMapPage

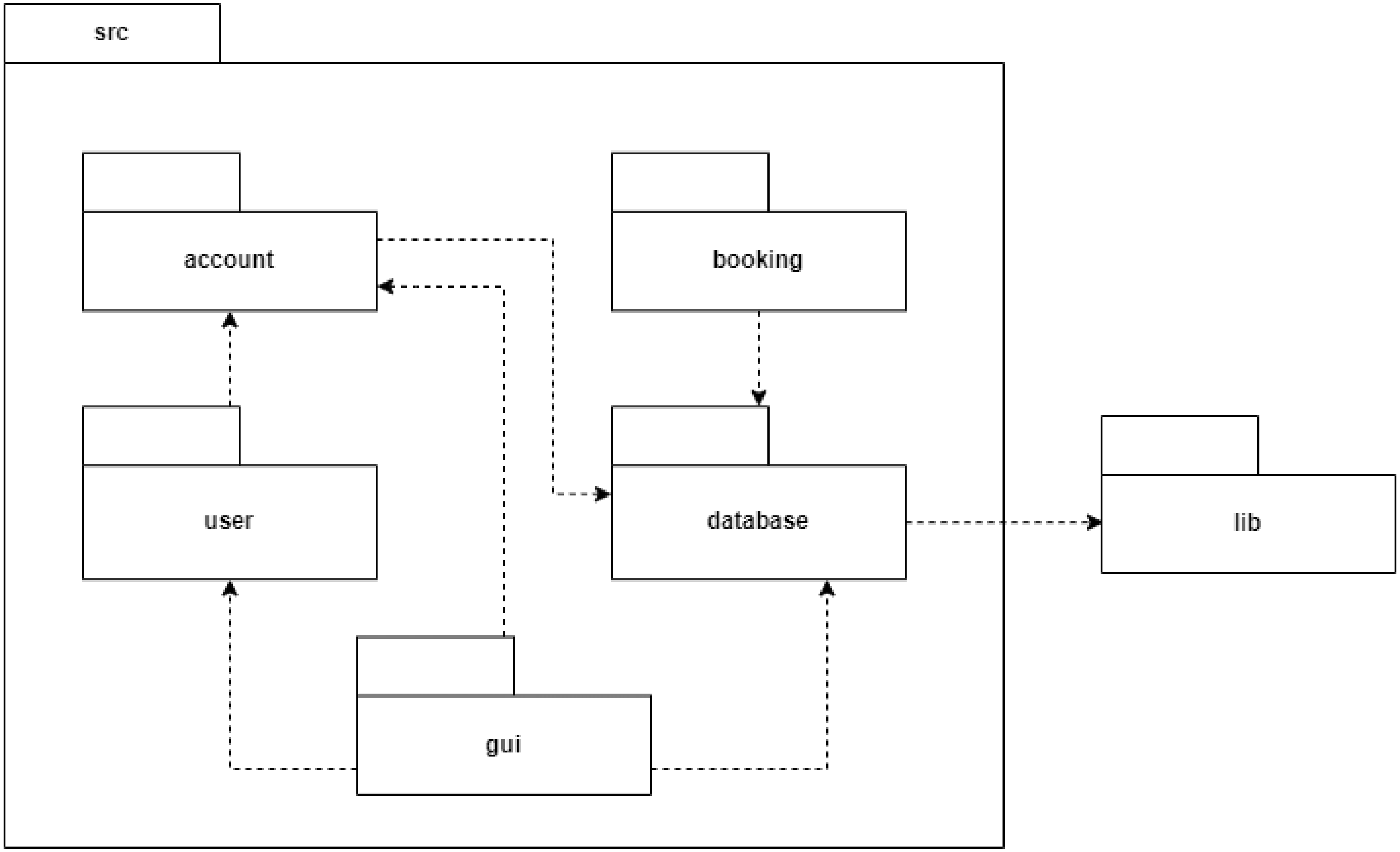
Part C: System’s Detailed Design-Class Diagram

Detailed Design:

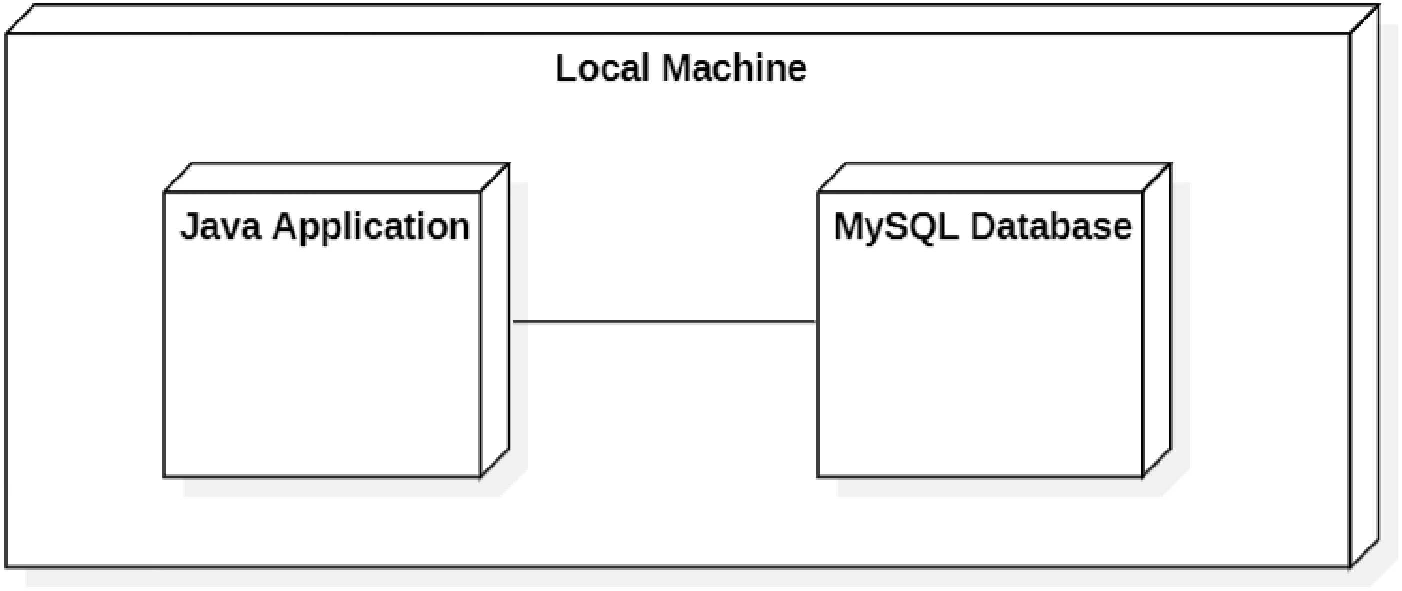


Part D: High-Level System's Architecture

Package Diagram:



Deployment Diagram:


Local Machine
Java Application
MySQL Database