OOAD MINIPROJECT REPORT

AIRLINE RESERVATION SYSTEM

Team Members:

Aniket Acharya (PES1UG20CS052)
Anish Khairnar (PES1UG20CS058)
Anoushka Gupta (PES1UG20CS060)

Synopsis:

As a passenger, it can get tedious to book flights due to the high costs and time associate ticket booking process. There is no transparency regarding subjects like tax before the booking is finalized. Similarly, We there is no flexibility in terms of return dates and cancellation of tickets.

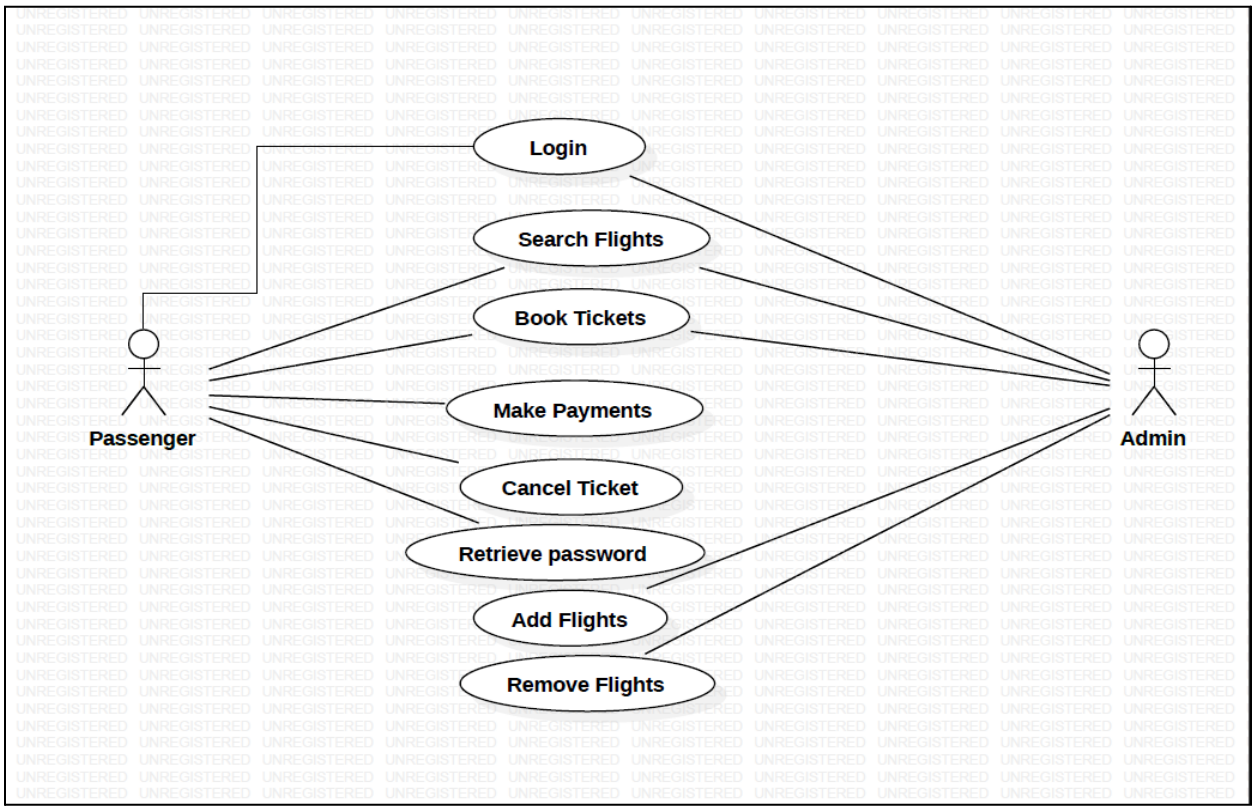
To solve this problem, we've designed a ticket booking system to manage and book flight reservations for a particular airline that allows users to save time and customize their booking as per their choice.

The system automates the process of searching and booking flight tickets, as well as managing customer details and preferences. The airline reservation system includes a payment processing system, which allows customers to pay for their flights via debit card or credit card option.

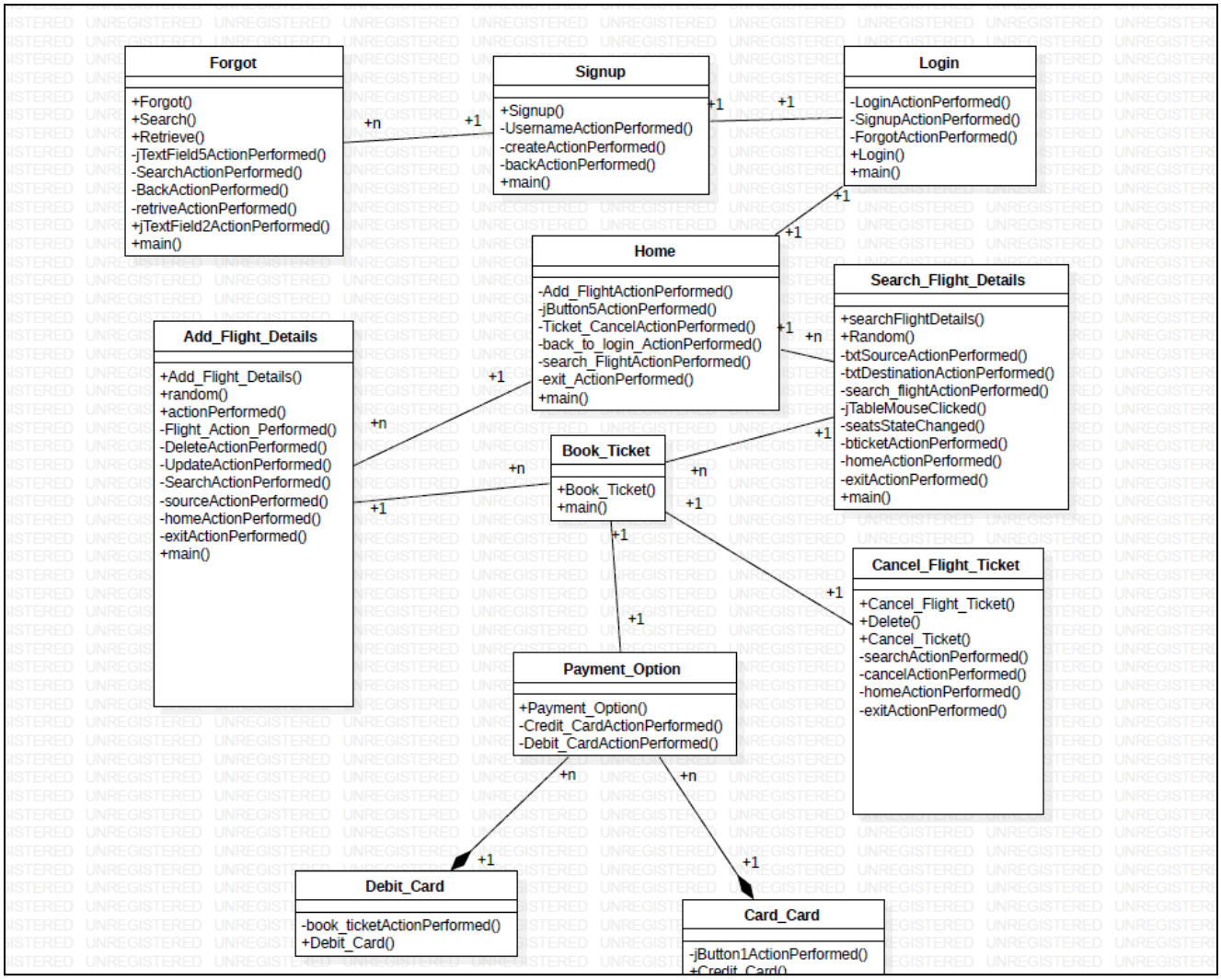
As an add-on, we have provided the option to search for flights and autofill details of the flight so as to save users some time. We also include the tax in the final price to provide complete transparency to the user. As an admin we get to add and update flight details very easily.

Overall, the airline reservation system streamlines the booking process for customers and airlines alike, improving efficiency and reducing overall costs.

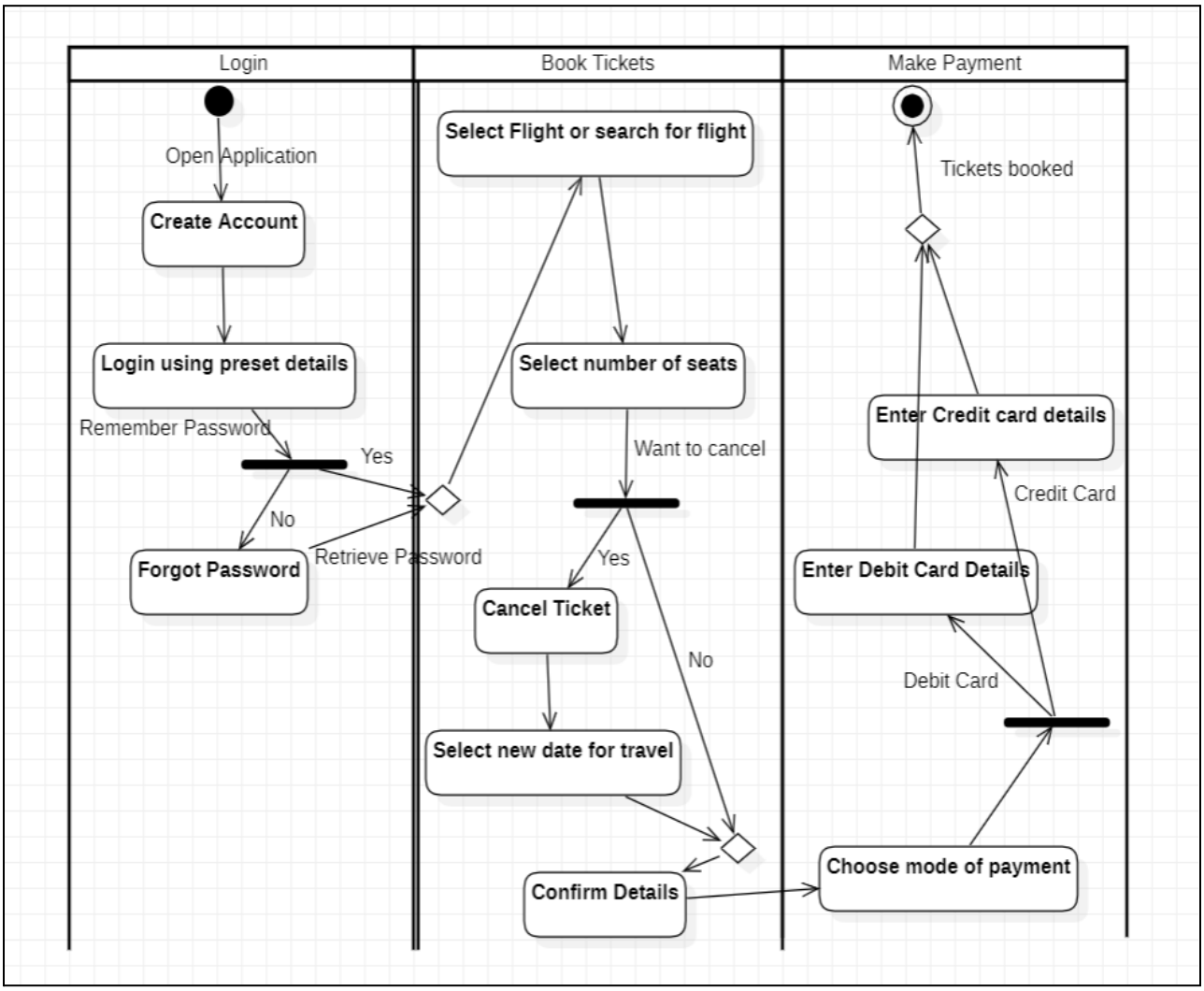
Use case diagram:



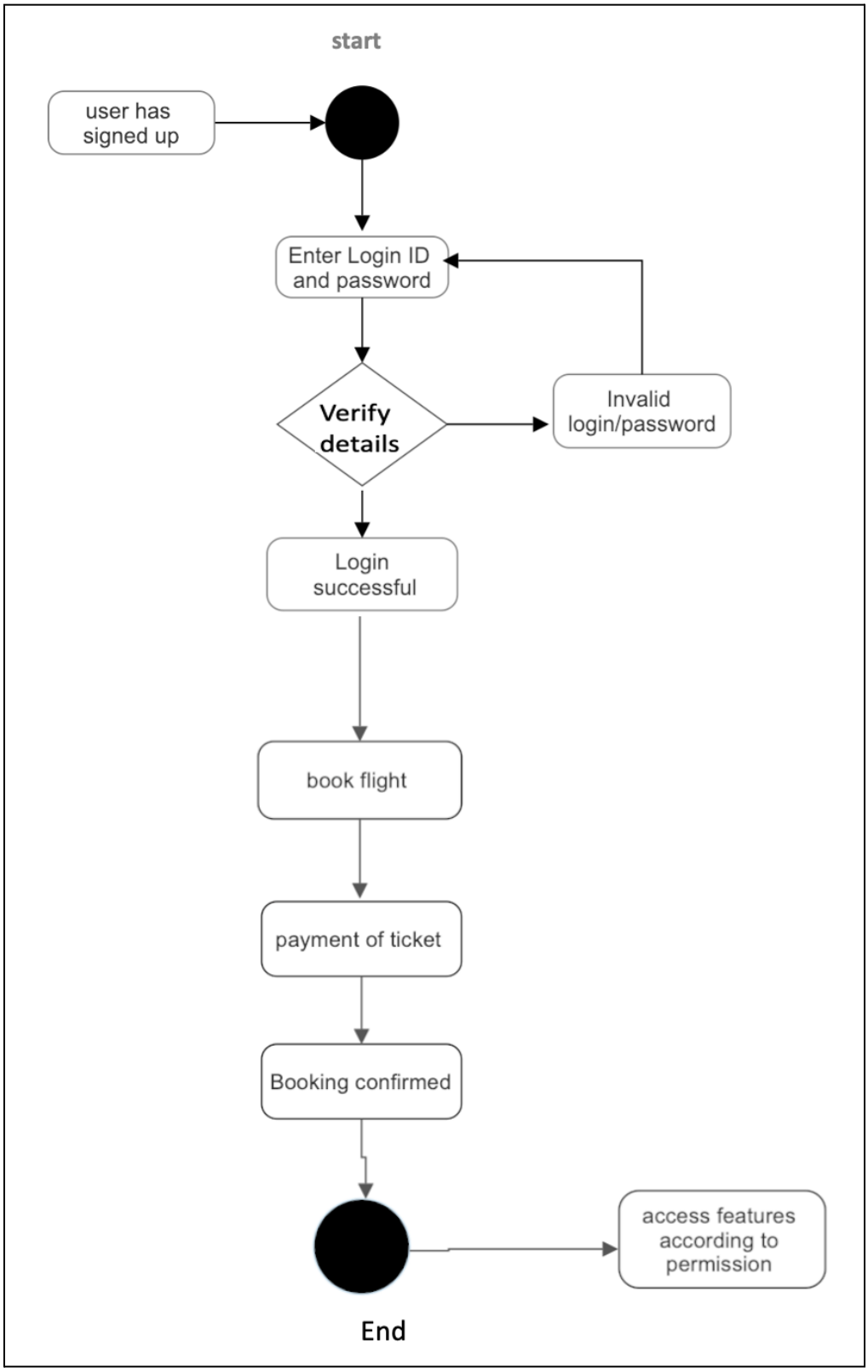
Class diagram:



Activity Diagram:



State Diagram:



Design patterns used:

- **Factory Method Design Pattern:** The code creates new instances of UI objects using a factory method, i.e., the setVisible method. This allows for the creation of different types of UI objects without having to modify the code for each object.

- **Singleton pattern:** The code uses a singleton database connection object (conn) to connect to a database.

Design principles used:

- **Single Responsibility Principle:** Each method in the code has a single responsibility, i.e., to handle a specific event. For example, the Add_FlightActionPerformed method handles the event when the user clicks the "Add Flight" button, while the exitActionPerformed method handles the event when the user clicks the "Exit" button.

- **Separation of Concerns (SoC):** The code separates the user interface (UI) components from the business logic. For example, the event handlers only deal with changing the state of the UI and creating new UI objects, while the actual data processing is done in separate classes.