

*Documentation on*

# **Bus Reservation System**

for JEE Cloud Project Work

By:

Tejaswini Bandaru

Aditya Gautam Mishra

Mayank Raj

## Table of Contents

1. Abstract
2. Scope
3. Diagrams
4. Classes used:-
  - a. Class Description

## 1. Abstract

Bus Reservation System is designed to **automate the online ticket purchasing** through an easy-to-use **online bus booking system**. This project is aimed at developing Bus Reservation System for customers, a web based application that can be accessed throughout the web. This system enables customers to login into BRS Application, search Bus details, book tickets for various routes and destinations, view booked tickets, their booking history and cancel any booking. This is an integrated system that contains both the user and the admin. The admin can login to the system using his/her credentials and view bus details, view user details.

## 2. Scope

Admin:

1. Add Bus
2. View Bus
3. Cancel Bus
4. View User

Customer:

1. Search Running Buses
2. Book a Ticket
3. Add Passengers
4. View Tickets List
5. Cancel Tickets

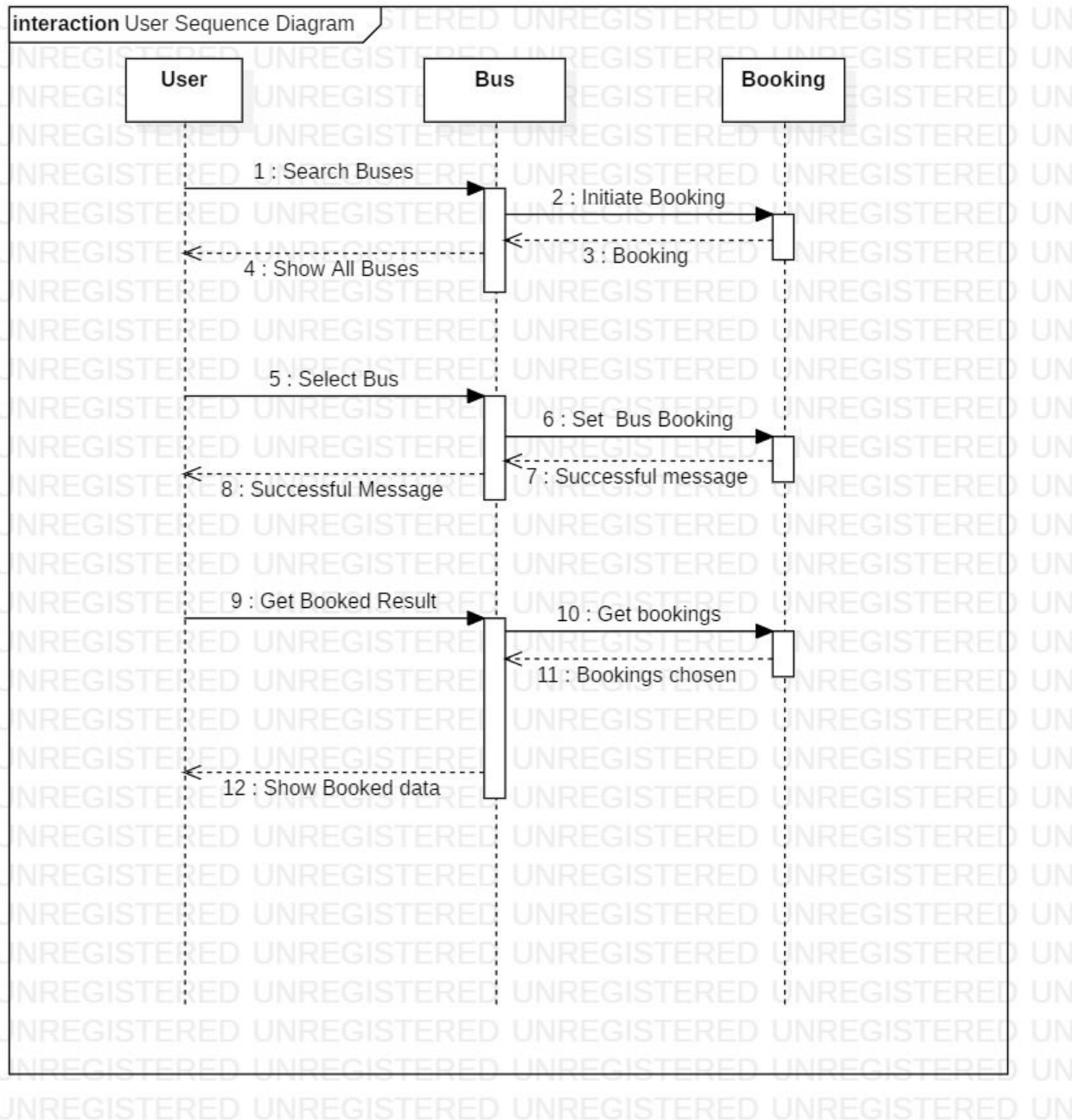
User:

1. Login
2. Register

## Out Of Scope

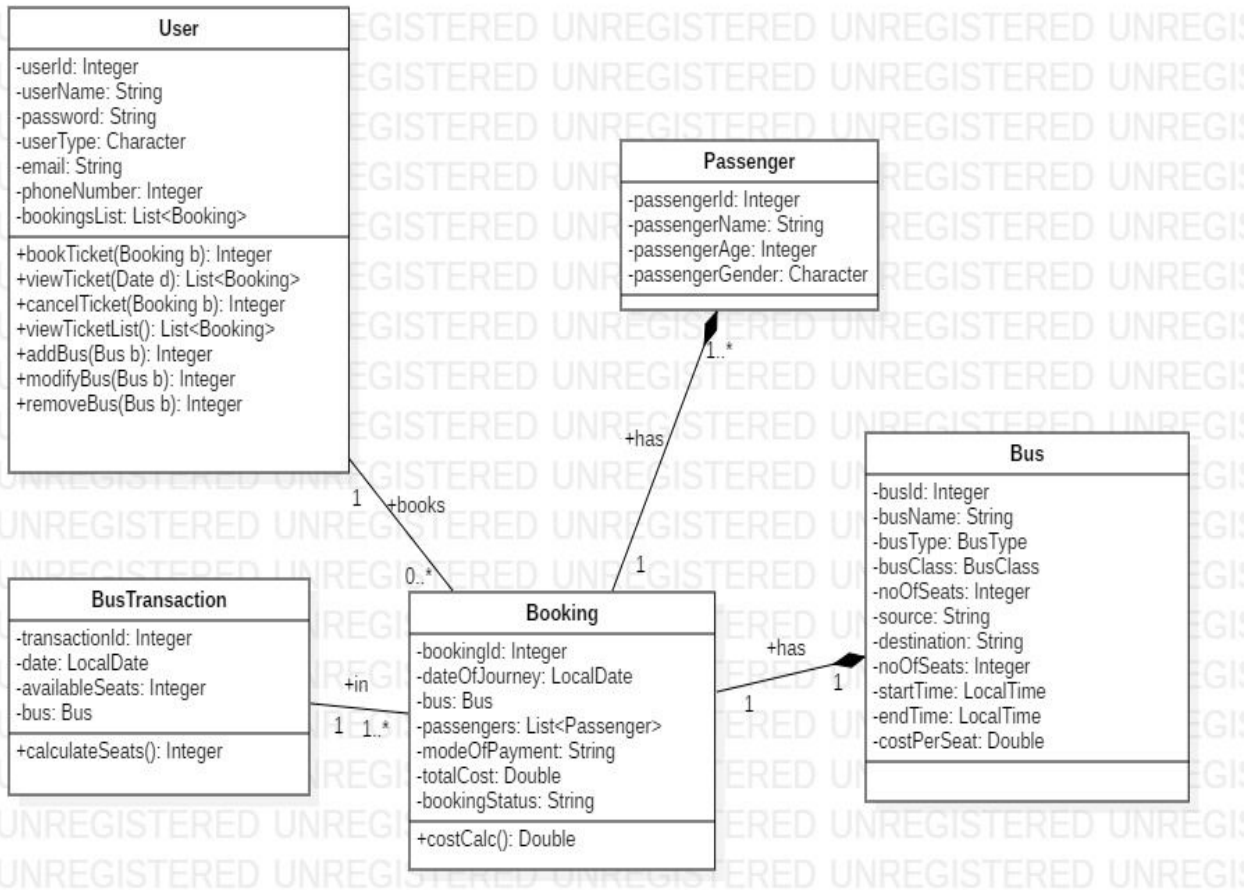
1. User is able to download the data in the same manner from Transactions into a report PDF or Excel file.
2. Payments are not being covered at present.
3. Reservation quota (Women, Handicapped or Senior Citizen) are currently not implemented.
4. Limited routes are being used in this model with no via stations.
5. Buses are assumed to be running everyday and will be deleted entirely not updated.
6. Booking can be cancelled entirely only.

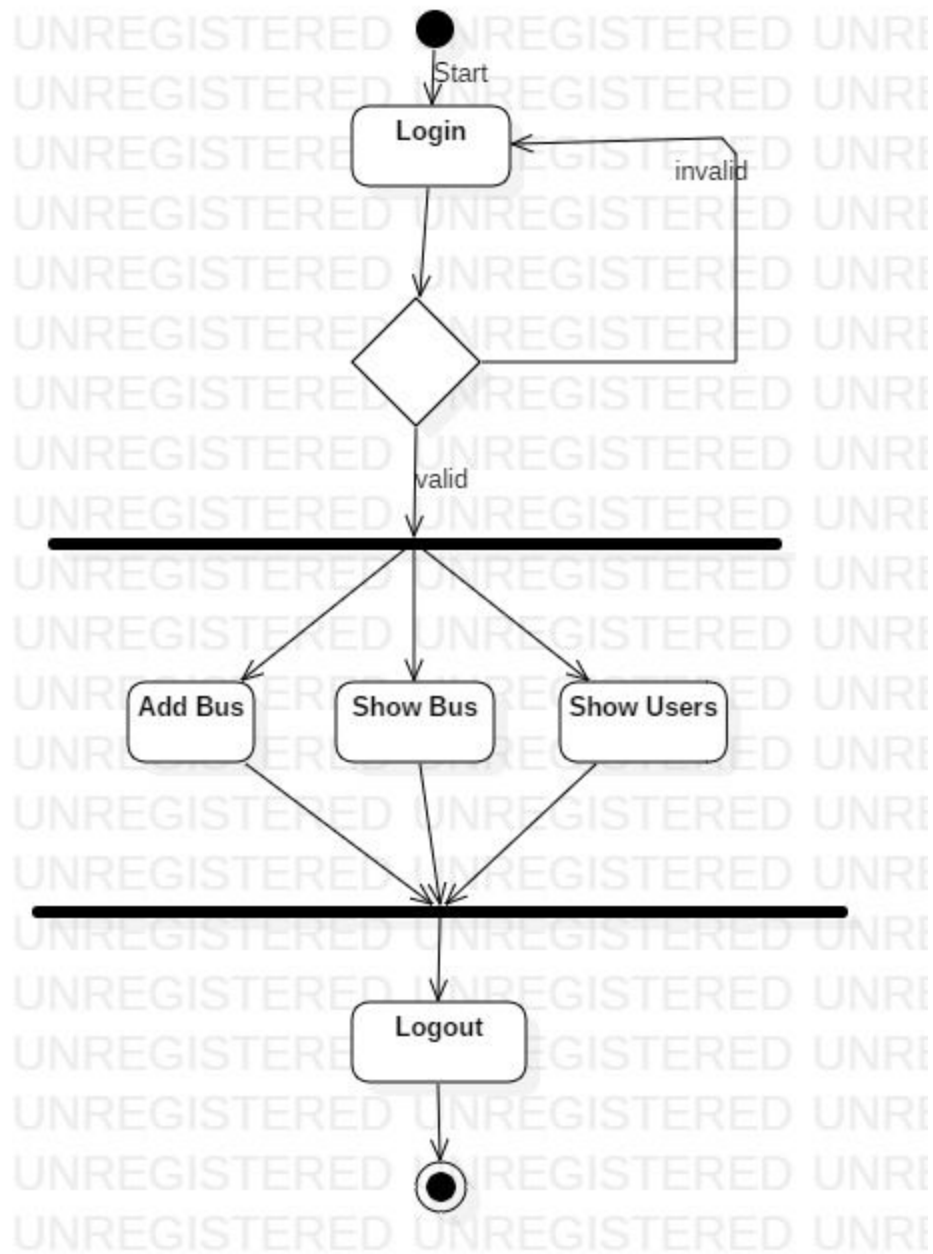
### 3. Diagrams



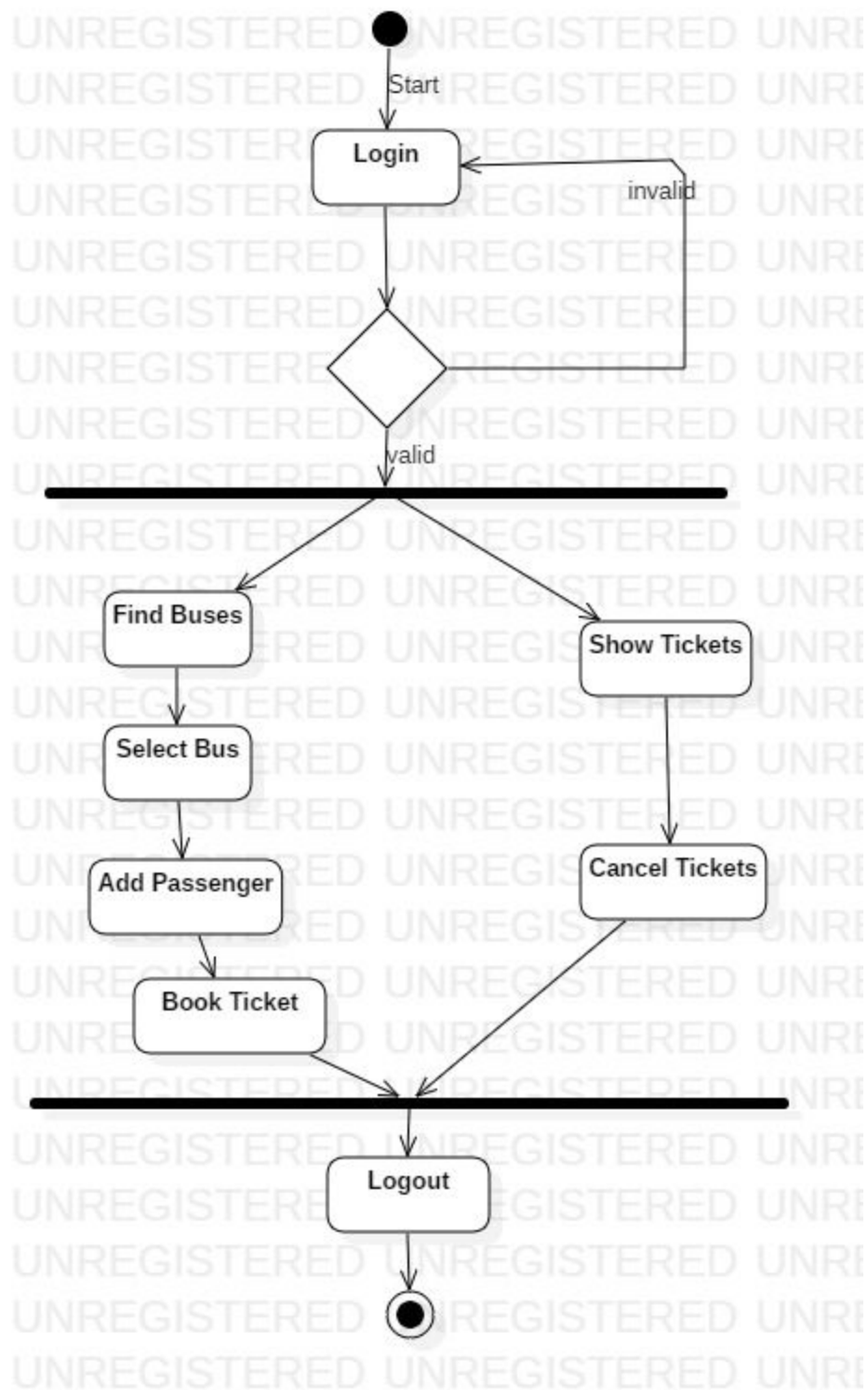
Sequence Diagram

## Class Diagram



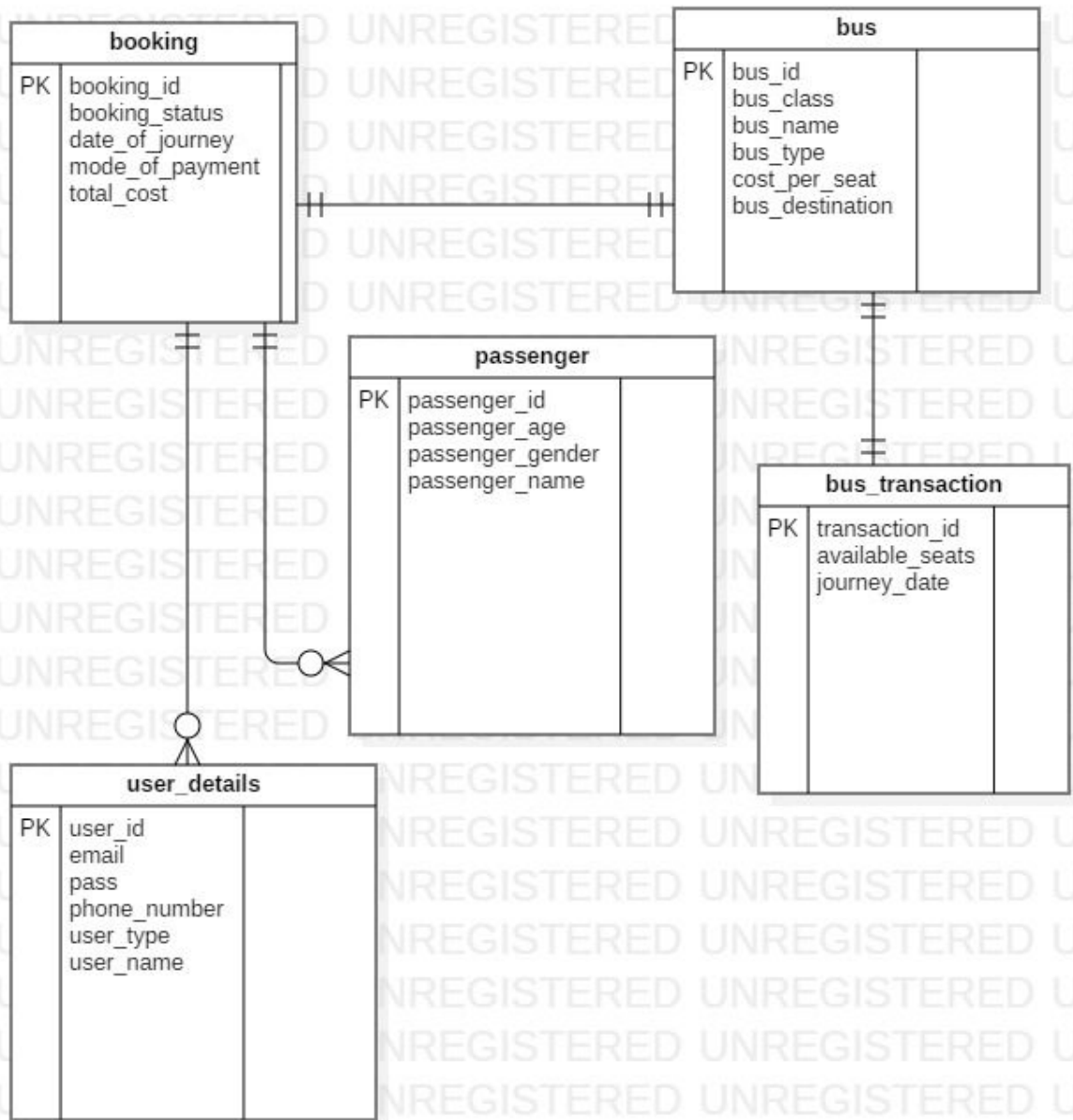


Activity Diagram - Admin

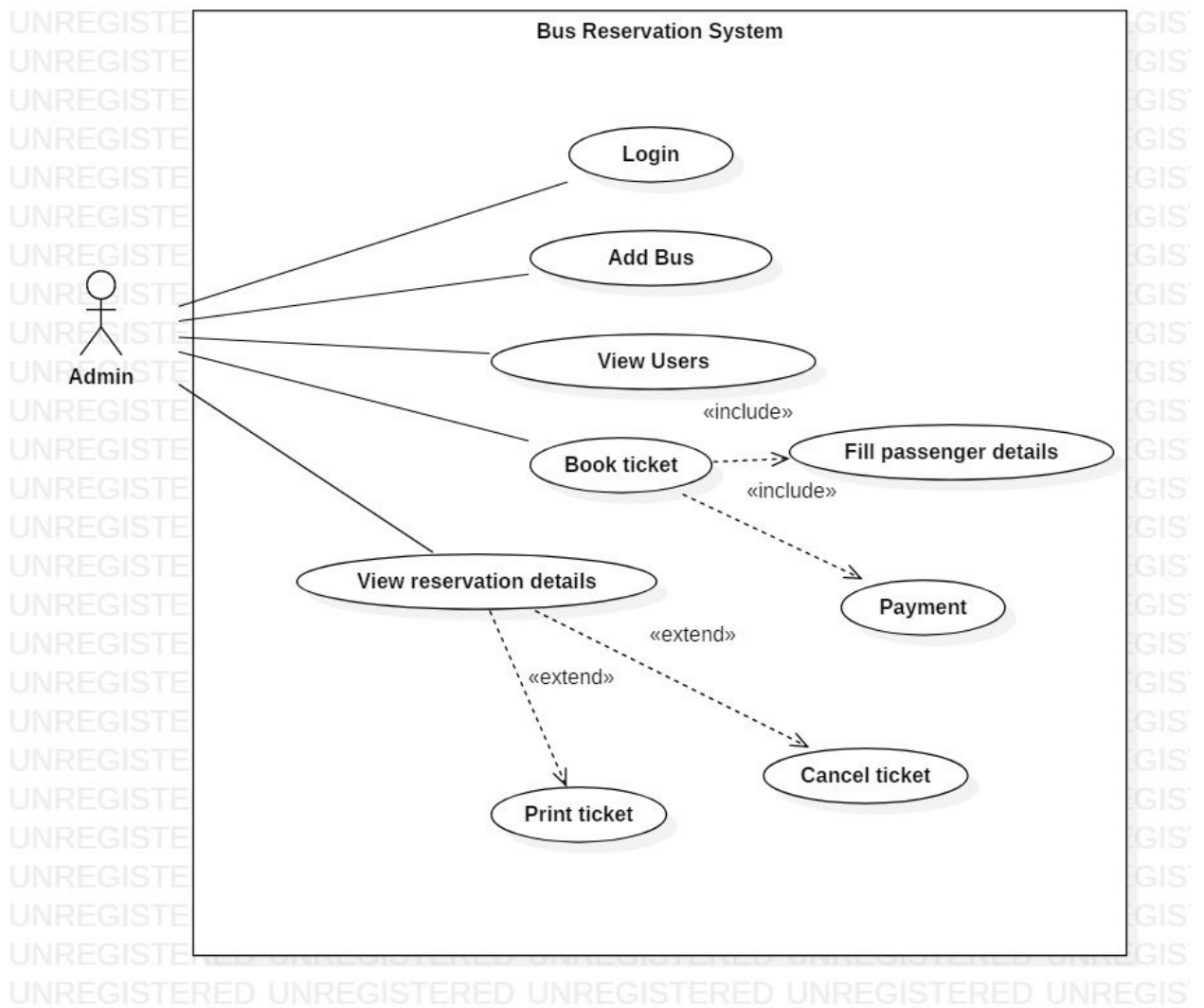


Activity Diagram - Customer

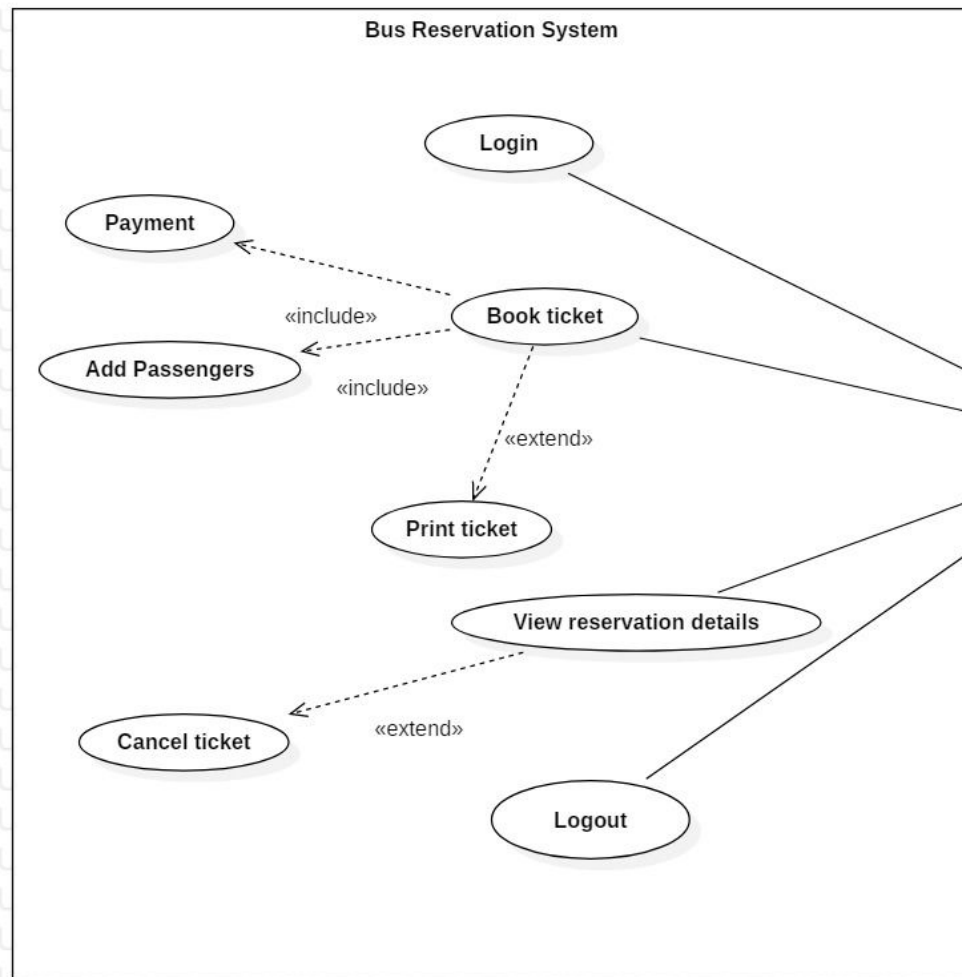




ER Diagram

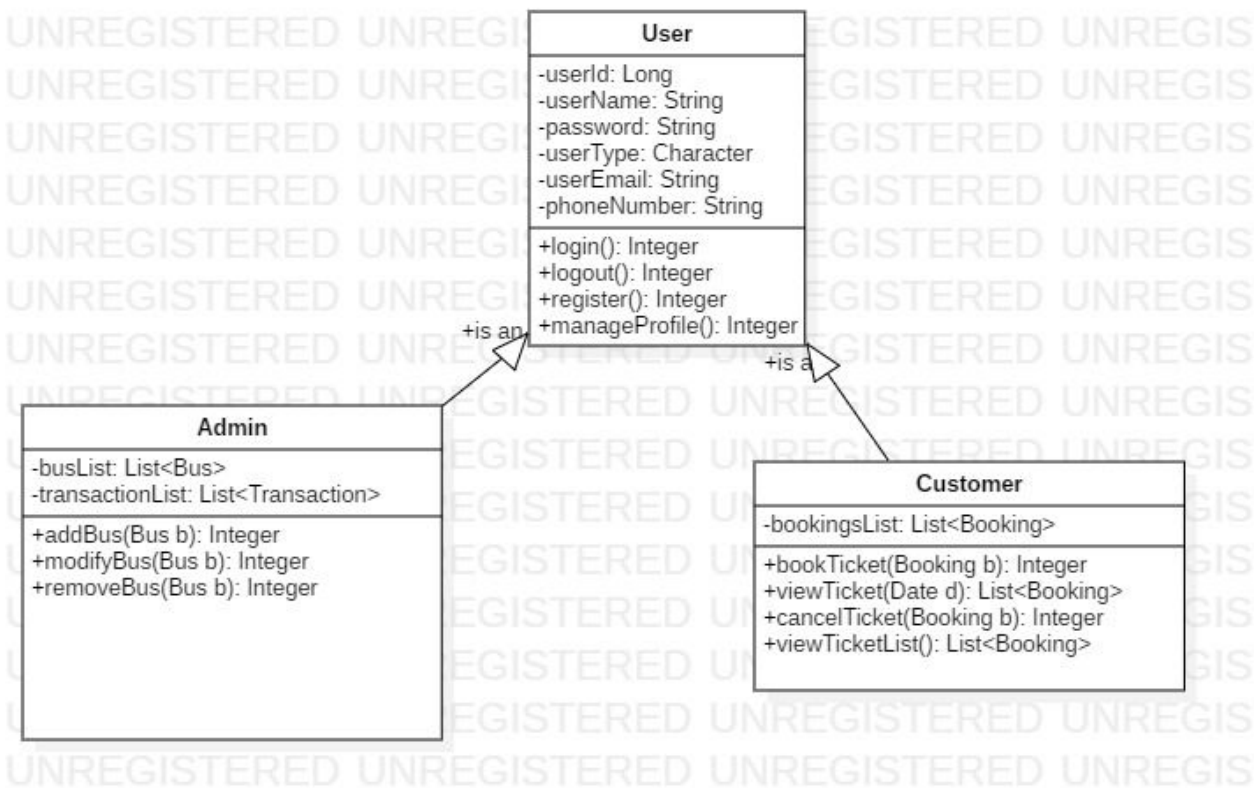


Use Case - Admin

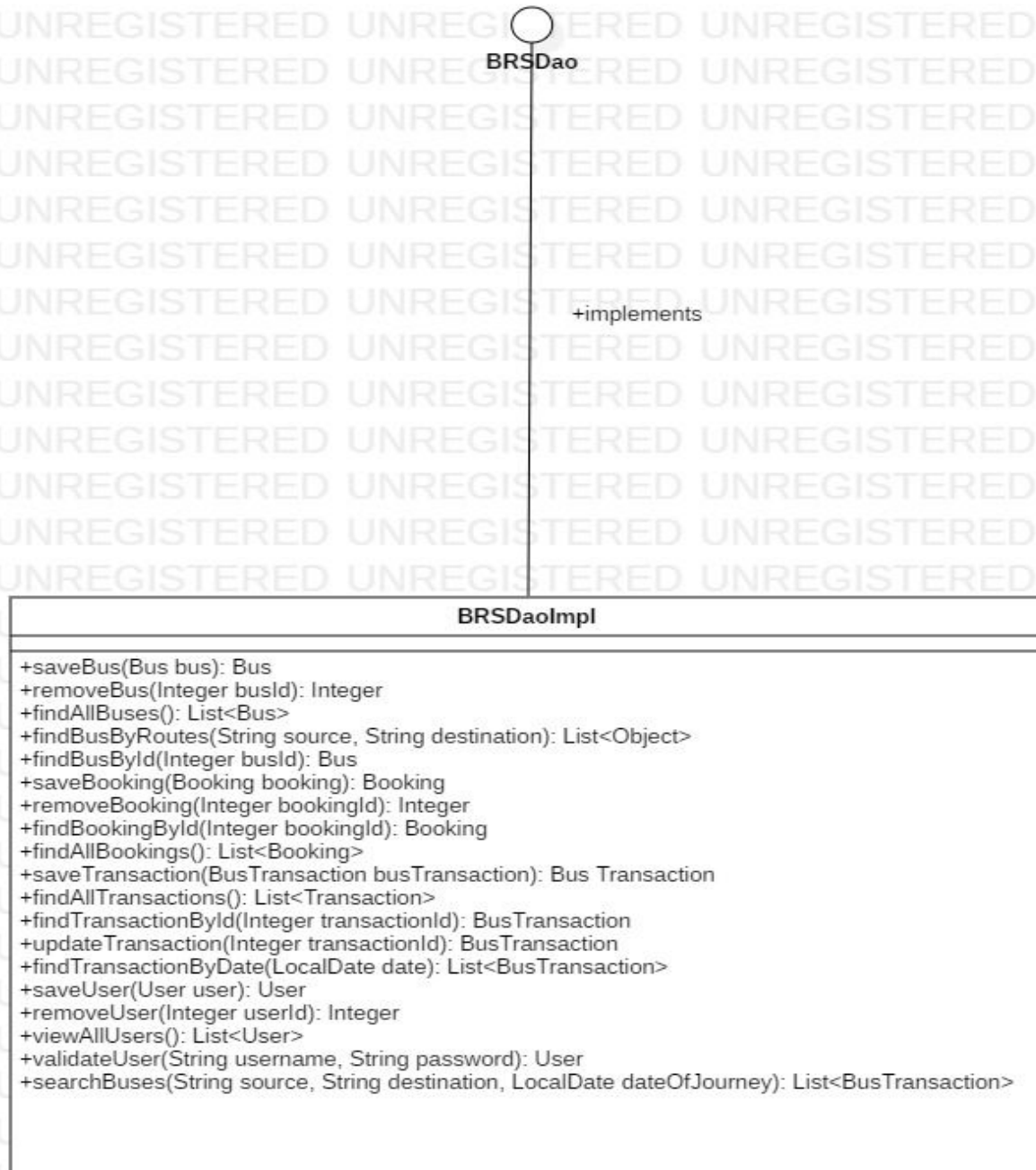


Use Case- Customer

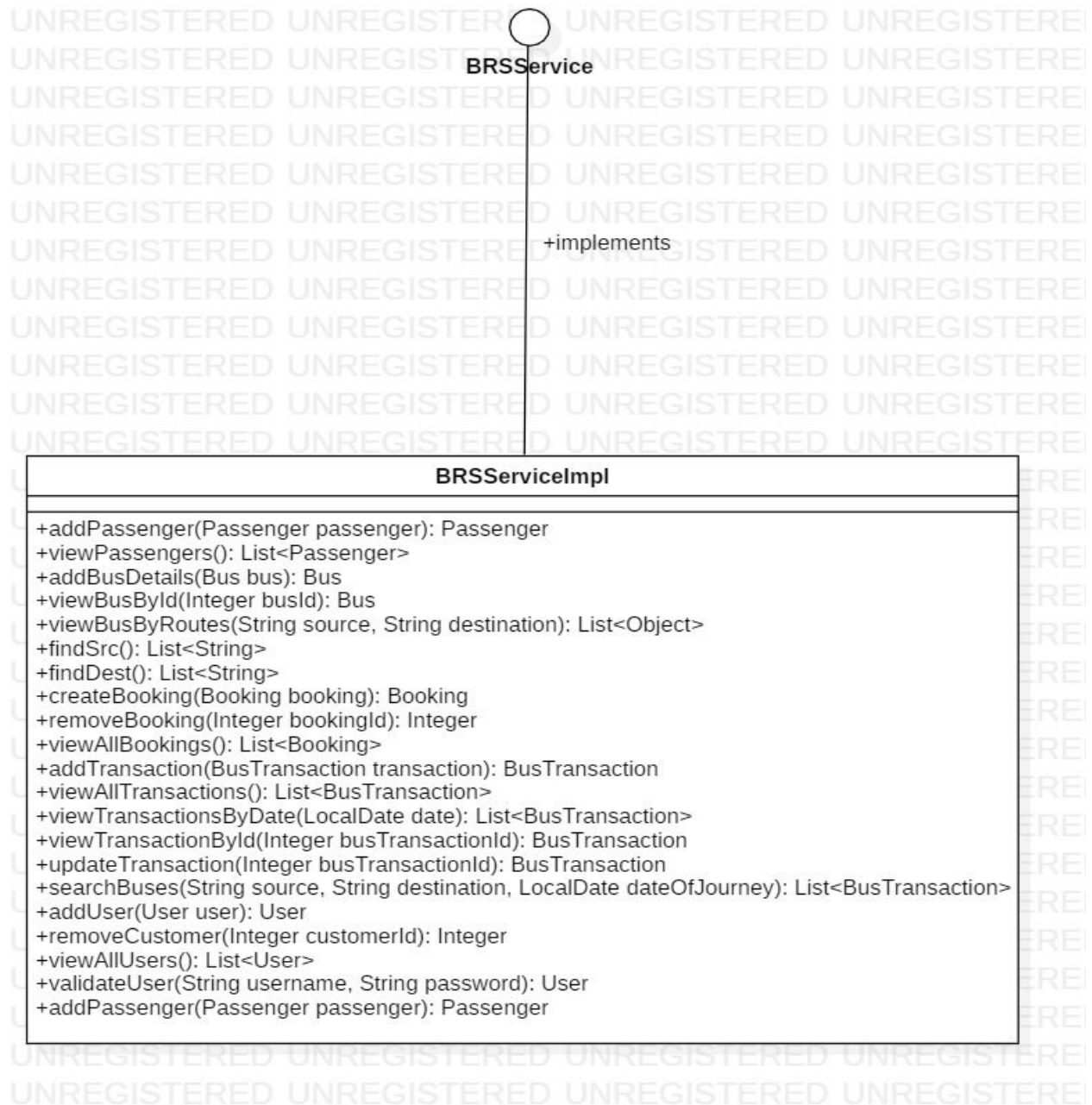
# User generalization diagram



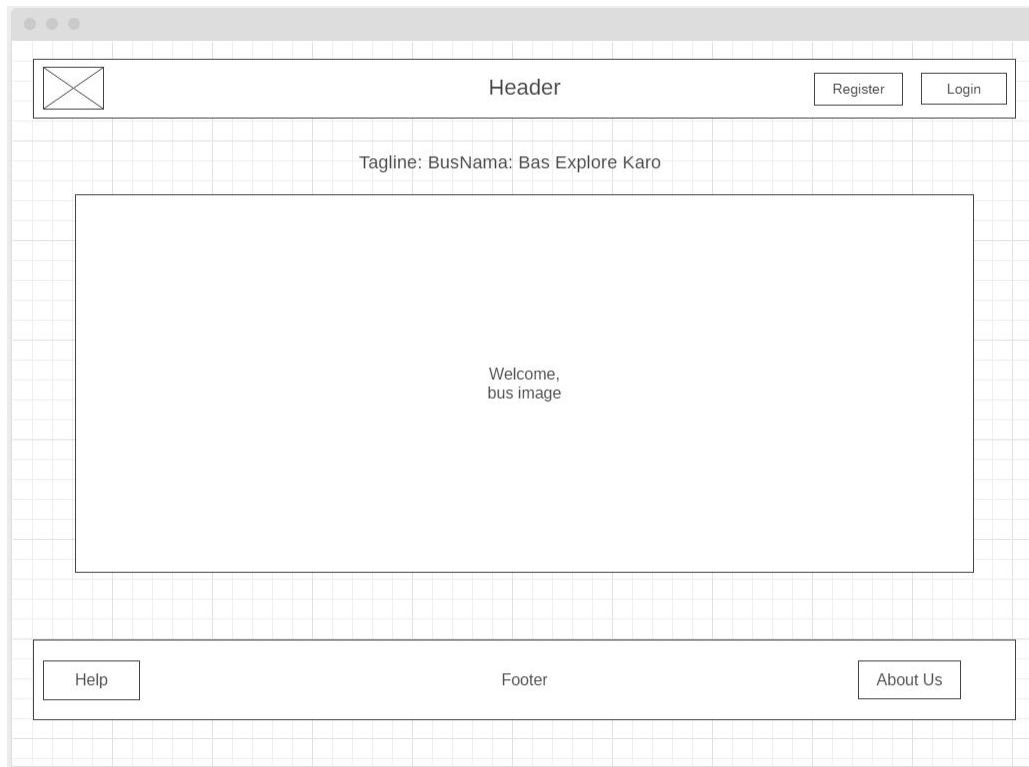
## DAO Diagram



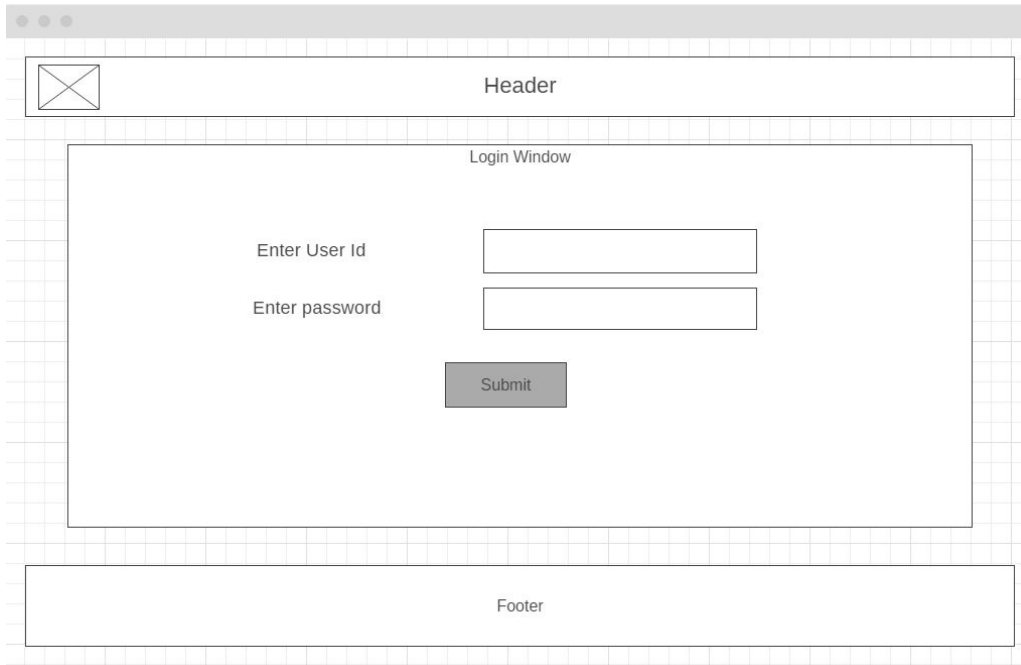
## Service Diagram



# Wireframes

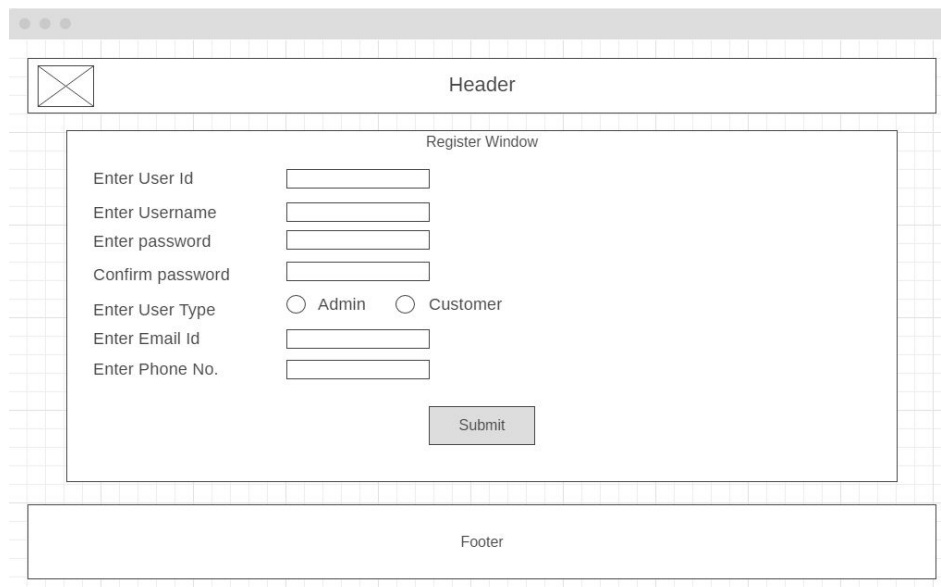


Home Wireframe



A wireframe for a login window. At the top is a header bar with a close button (an 'X' in a square) on the left and the word 'Header' in the center. Below the header is a large rectangular area labeled 'Login Window'. Inside this area, there are two input fields: the first is preceded by the text 'Enter User Id' and the second by 'Enter password'. Below these fields is a gray rectangular button labeled 'Submit'. At the bottom of the entire window is a footer bar labeled 'Footer'.

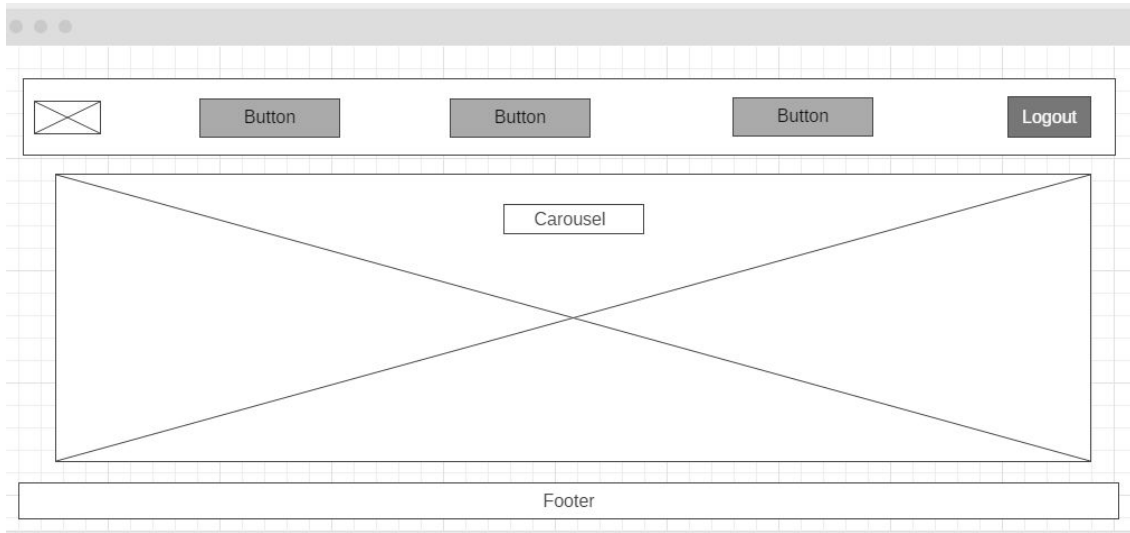
Login Wireframe



A wireframe for a register window. At the top is a header bar with a close button (an 'X' in a square) on the left and the word 'Header' in the center. Below the header is a large rectangular area labeled 'Register Window'. Inside this area, there are several input fields and a radio button group. The fields are preceded by the following labels: 'Enter User Id', 'Enter Username', 'Enter password', 'Confirm password', 'Enter Email Id', and 'Enter Phone No.'. The radio button group is labeled 'Enter User Type' and has two options: 'Admin' and 'Customer'. Below these fields is a gray rectangular button labeled 'Submit'. At the bottom of the entire window is a footer bar labeled 'Footer'.

Register Wireframe



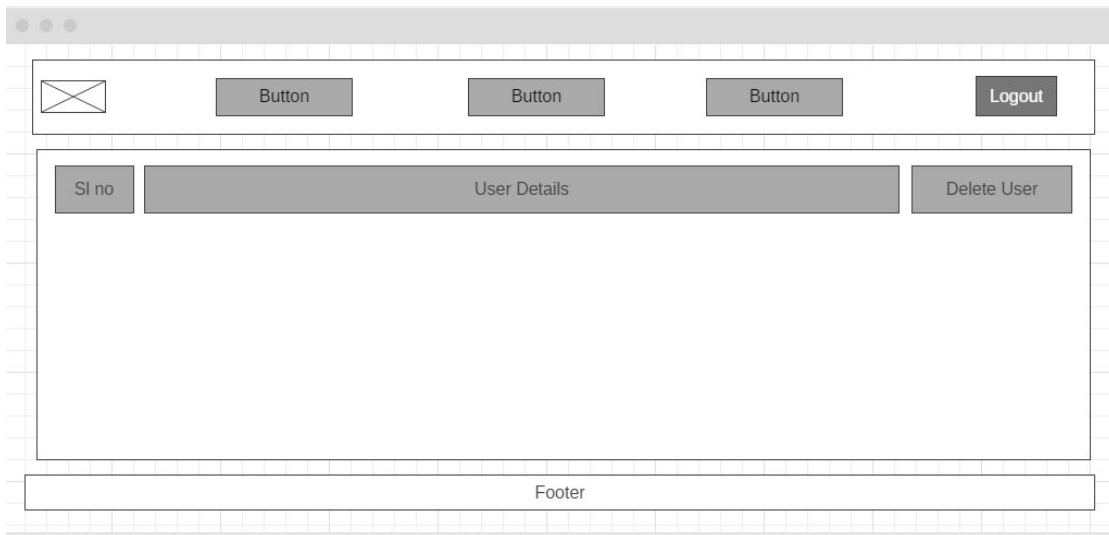


Admin home wireframe



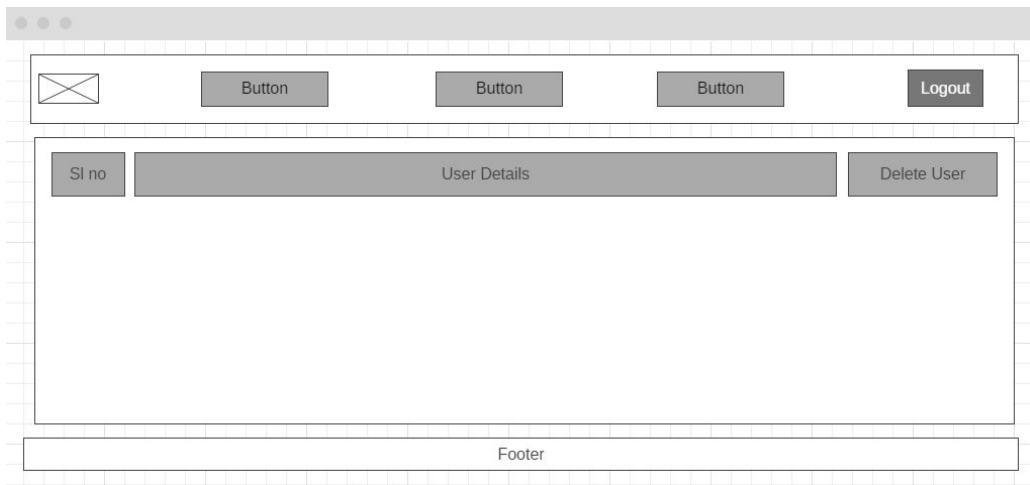
A wireframe for an 'Add Bus' form. At the top is a navigation bar with a home icon, four 'Button' placeholders, and a 'Logout' button. The main content area is a grid with a central form box. The form contains: 'Bus Name' (text input), 'Type' (radio buttons for 'AC' and 'Non AC'), 'Class' (dropdown), 'Source' (dropdown), 'Destination' (dropdown), 'Start Time' (text input), 'End Time' (text input), 'No of seats' (text input), 'Cost per seat' (text input), and an 'Add' button. A 'Footer' bar is at the bottom.

Add Bus Wireframe



A wireframe for a 'View Buses' page. The top navigation bar is identical to the 'Add Bus' wireframe. Below it is a table with three columns: 'SI no', 'User Details', and 'Delete User'. The table body is empty. A 'Footer' bar is at the bottom.

View Buses Wireframe



View Users Wireframe



Customer Home Wireframe

Button

Button

Button

Logout

Source

srcList ▼

Destination

destList ▼

Date

DatePicker

Search Buses

Bus Details Table

Footer

## Show Running Buses Wireframe

Button

Button

Button

Logout

Passenger Name

Passenger Age

Passenger Gender

Male

Female

Add

Payment Methods

UPI

Debit Card

Credit Card

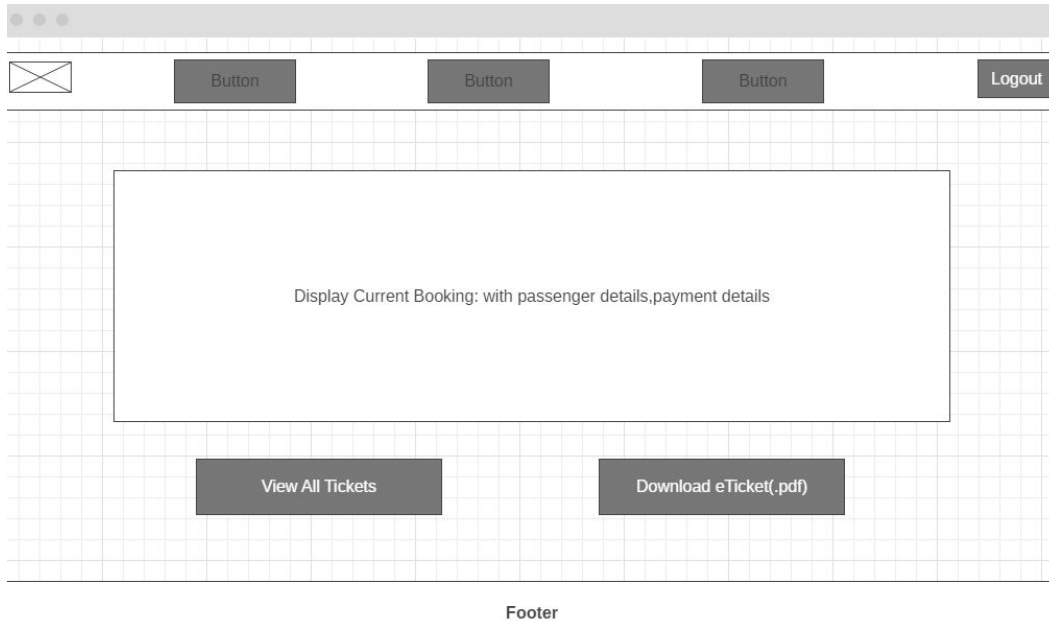
Net Banking

Passengers List

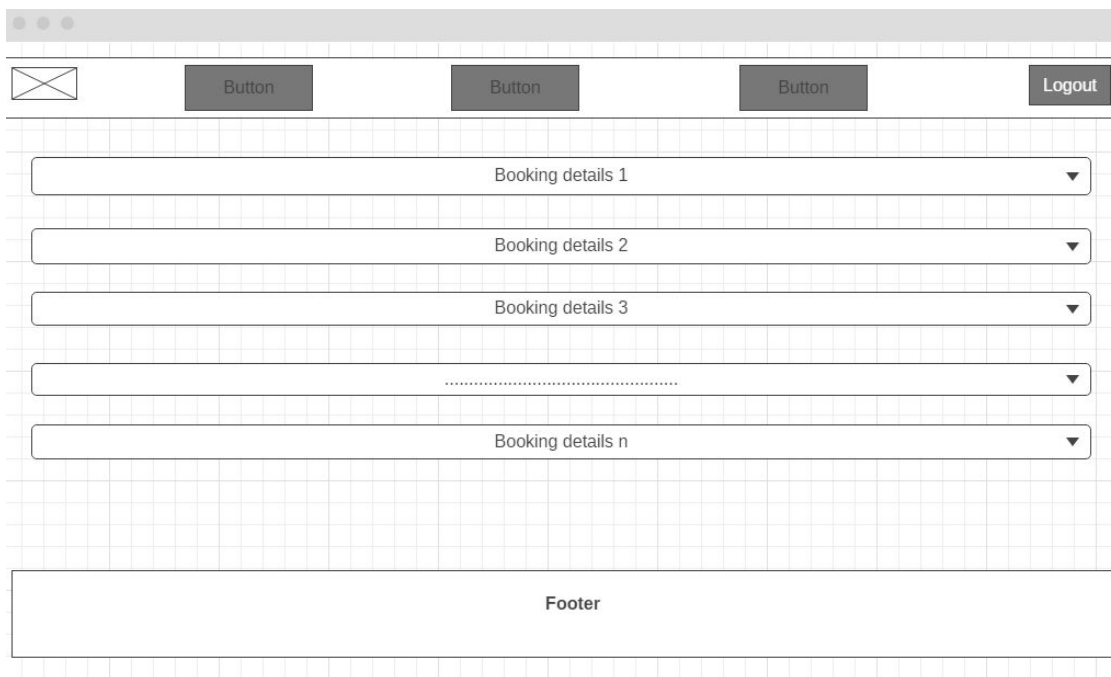
Proceed

Footer

## Add Passenger Wireframe



View Current Booking Wireframe



View All Bookings Wireframe

### 3b. Class Description I

The following classes are being used in our Bus Reservation System :

1. **User**: General class containing basic properties of user

Attributes:

1. *userId*: Integer
2. *userName*: String
3. *password*: String
4. *userType* :Character-'A' for admin and 'C' for 'Customer'.
5. *userEmail*: String
6. *phoneNumber*: Integer
7. *bookingsList*: List<Booking>

Methods:

1. *bookTicket(Booking b)*: Integer - Get the route details, bus details and passenger details from the user. The count of passengers per ticket should not cross more than 5. Date of journey should be either today's date or the date in the future.
2. *viewTicket(Date d)*: List<Booking> - Lists the tickets based on their travel date.
3. *cancelTicket(Booking b)*: Integer - Searches the ticket from the booking history using ticketID and cancels it, and updates the booking history.
4. *viewTicketList()*: List<Booking> - Lists all the bookings made by the customer.
5. *addBus(Bus b)* : Integer :- Adds the details for a new bus. Checks whether all the details have been added properly like busType should be sleeper or non-sleeper, busClass should be AC or non-AC. Returns 1 if all the details have been updated successfully, else 0.
6. *modifyBus( Bus b)*: Integer:-Update details of the bus if required. Returns 1 if details are updated successfully, else 0;
7. *removeBus(Bus b)*: Integer:- Search for the bus using busId, if the busId is present in the list, removes the corresponding bus details. Removes successfully and returns 1. If busId does not exist in the list, it returns 0;

2. **Passenger**: details of individual passenger undertaking the journey

Attributes:

1. *passengerId*: Integer
2. *passengerName*: String

3. *passengerAge*: Integer
4. *passengerGender*: Character-'M' for male, 'F' for female

3. **BusTransaction**: stores all the transaction for that particular date

Attributes:

1. *transactionId*: Integer
2. *date*: LocalDate
3. *availableSeats*: Integer
4. *bus*: Bus

Methods:

1. *calculateSeats()*: Integer-Get the passenger count from Booking and update the available seats on every transaction. When we are about to create a booking, this must be non negative to be able to proceed further.

4. **Booking** : object to store booking made by customers

Attributes:

1. *bookingId*: Integer
2. *dateOfJourney*: LocalDate
3. *bus*: Bus
4. *passengers*: List<Passenger>
5. *modeOfPayment*: String
6. *bookingStatus*: String
7. *totalCost*: Double

Methods :

1. *costCalc(Bus b)*: Double-Updates the count of availableSeats by reducing the noOfSeats with the passenger count and calculates the final cost, including tax.

5. **Bus**: contains attributes of bus type

Attributes:

1. *busId*: Integer
2. *busName*: String
3. *busType*: String
4. *busClass*: String
5. *noOfSeats*: Integer
6. *source*: String
7. *destination*: String
8. *costPerSeat*: Double
9. *startTime*: LocalTime
10. *endTime*: LocalTime

## Class Description II

### 1. **User**: General class containing basic properties of user

#### Attributes:

1. *userId*: Integer
2. *userName*: String
3. *password*: String
4. *userType*: Character-'A' for admin and 'C' for 'Customer'.
5. *userEmail*: String
6. *phoneNumber*: Integer
7. *bookingsList*: List<Booking>
8. *busList*: List<Bus>

#### Methods:

1. *login()*: *Integer* - Logs the user onto the website. Returns 1 if the login credentials of the user are correct, else
2. *logout()*: *Integer* - Logs out of the website. Returns 1 showing that the user has logged out successfully.
3. *register()* : *Integer* - Creates an account for the new user. Asks user to fill in the details. Prompts the user to add password of at least 8 characters, minimum 1 uppercase character, 1 lowercase character, 1 digit and 1 special character; phone number should have 10 digits.
4. *manageProfile()*: *Integer* - Lets the user to update their details such as changing their password, updating their phone number

### 2. **Admin**: Contains admin object, would perform all admin functions -extends User

#### Attributes:

1. *busList*: List<Bus>

#### Methods :

1. *addBus(Bus b)* : *Integer* :- Adds the details for a new bus. Checks whether all the details have been added properly like busType should be sleeper or non-sleeper, busClass should be AC or non-AC. Returns 1 if all the details have been updated successfully, else 0.
2. *modifyBus( Bus b)*: *Integer*:-Update details of the bus if required. Returns 1 if details are updated successfully, else 0;
3. *removeBus(Bus b)*: *Integer*:- Search for the bus using busId, if the busId is present in the list, removes the corresponding bus details. Removes successfully and returns 1. If busId does not exist in the list, it returns 0;



3. **Customer:** client / customer object-extends User

Attributes: *bookingsList: List<Booking>*

Methods :

1. *bookTicket(Booking b): Integer* - Get the route details, bus details and passenger details from the user. The count of passengers per ticket should not cross more than 5. Date of journey should be either today's date or the date in the future.
2. *viewTicket(Date d): List<Booking>* - Lists the tickets based on their travel date.
3. *cancelTicket(Booking b): Integer* - Searches the ticket from the booking history using ticketID and cancels it, and updates the booking history.
4. *viewTicketList(): List<Booking>* - Lists all the bookings made by the customer.