Documentation on

Bus Reservation System

for JEE Cloud Project Work

By:
Mayank Raj
Tejaswini Bandaru
Aditya Gautam Mishra

Table of Contents

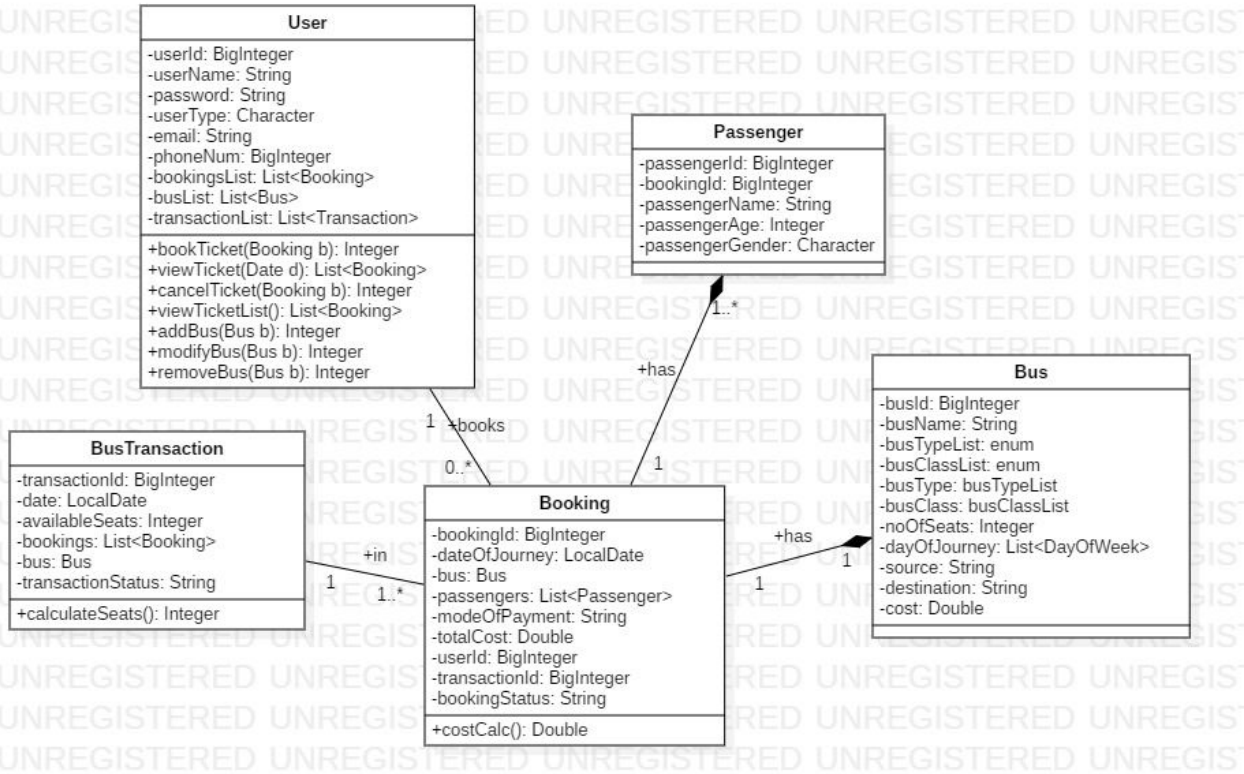
1. Abstract

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, edit and view bus details; edit and view schedule details; edit and view route details.

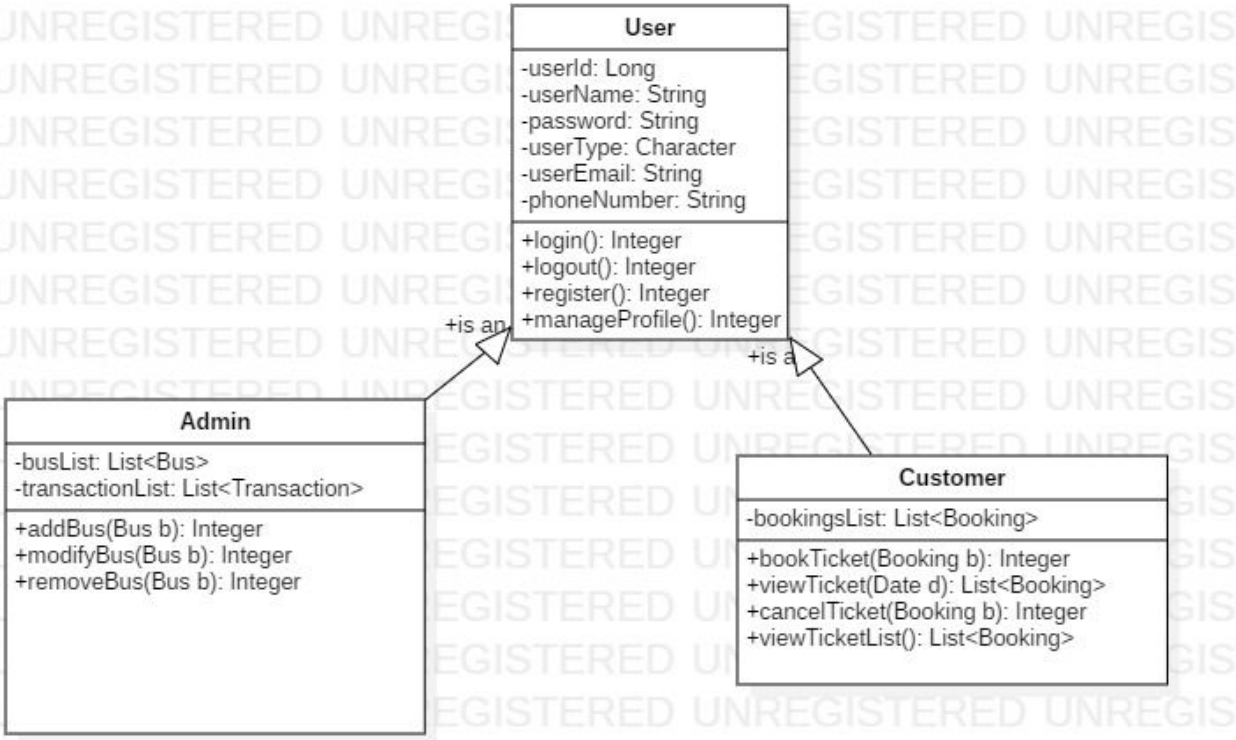
2. Use Case Diagram

3a. Class Diagram

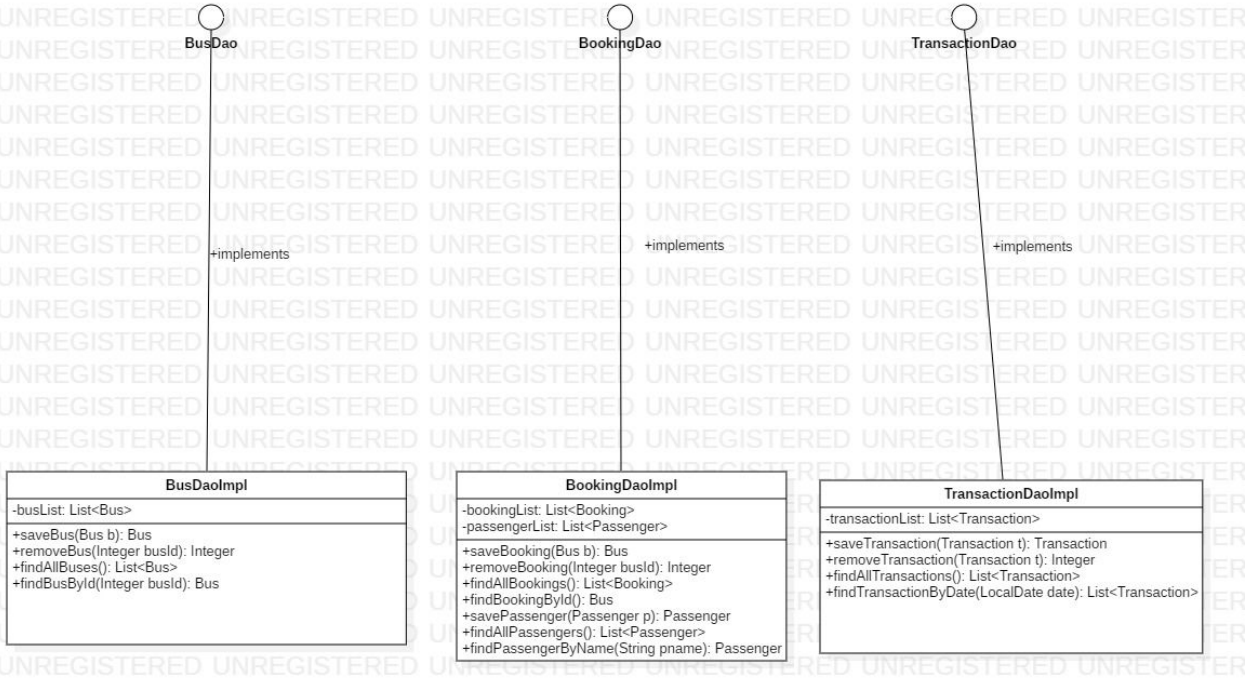
I. Main class diagram:



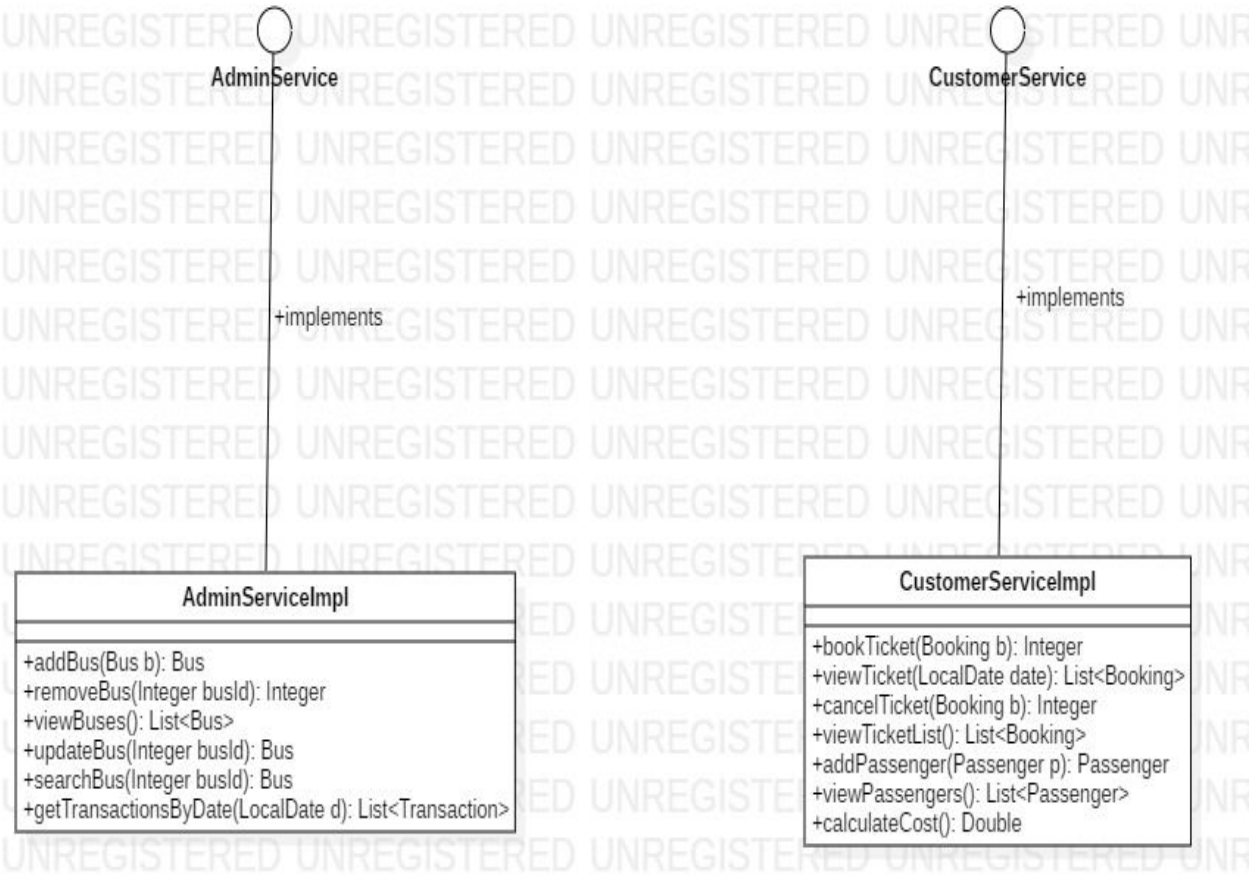
II. User generalization diagram



DAO Diagram


BusDao
+implements
BusDaoImpl
-busList: List<Bus>
+saveBus(Bus b): Bus
+removeBus(Integer busId): Integer
+findAllBuses(): List<Bus>
+findBusById(Integer busId): Bus
BookingDao
+implements
BookingDaoImpl
-bookingList: List<Booking>
-passengerList: List<Passenger>
+saveBooking(Bus b): Bus
+removeBooking(Integer busId): Integer
+findAllBookings(): List<Booking>
+findBookingById(): Bus
+savePassenger(Passenger p): Passenger
+findAllPassengers(): List<Passenger>
+findPassengerByName(String pname): Passenger
TransactionDao
+implements
TransactionDaoImpl
-transactionList: List<Transaction>
+saveTransaction(Transaction t): Transaction
+removeTransaction(Transaction t): Integer
+findAllTransactions(): List<Transaction>
+findTransactionByDate(LocalDate date): List<Transaction>

Service Diagram



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

   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:* BigInteger
   2. *bookingId: BigInteger*
   3. *passengerName*: String
   4. *passengerAge:* Integer
   5. *passengerGender:* Character-'M' for male, 'F' for female

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

Attributes:

1. *transactionId: BigInteger*
2. *date:* LocalDate
3. *availableSeats:* Integer
4. *bookings:* List<Booking>
5. *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.

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

Attributes:

1. *bookingId:* BigInteger
2. *userId: BigInteger*
3. *transactionId: BigInteger*
4. *dateOfJourney:* Date
5. *bus: Bus*
6. *passengers*: List<Passenger>
7. *modeOfPayment:* String
8. *bookingStatus: String*

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.

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

Attributes:

1. *busId*: BigInteger
2. *busName*: String
3. *busTypeList*: String[]-Sleeper or Semi-Sleeper
4. *busClassList*: String[]-AC or non-AC
5. *busType*: String
6. *busClass*: String
7. *noOfSeats*: Integer
8. *dayOfJourney*: Set<DayOfWeek(enum)>- On which days of the week, the bus will run.
9. *source: String*
10. *destination: String*
11. *cost: Double*

Class Description II

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

Attributes:

1. *userId:* BigInteger
2. *userName:* String
3. *password:* String
4. *userType:* Character-'A' for admin and 'C' for 'Customer'.
5. *userEmail:* String
6. *phoneNumber:* BigInteger
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.

4. Out Of Scope

1. Admin is able to upload data from any file e.g. - Excel Sheet and 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.