Dylan Dvorachek
V00863468

SENG 330: Assignment 2

Problem Statement A: “Who put the driver’s seat back like that?”

A car-sharing company is looking to construct a software system in support of their daily operations. Customers “time-share” a car – that is, they pay a specific yearly fee, and that fee gives them access to a car for a maximum number of hours per year. That time may be used all at once, or it may be used in one-hour increments. There are various models of vehicles from sub-compacts up to small trucks. Vehicle depots are located around the city, and users of the service may pick up and drop off a vehicle at the most convenient depot. Individual vehicles must be maintained periodically which keeps them out of service for a short period. Once they have paid their fee, customers will want to both make their own bookings, along with checking their future and past bookings.

TABLE OF CONTENTS

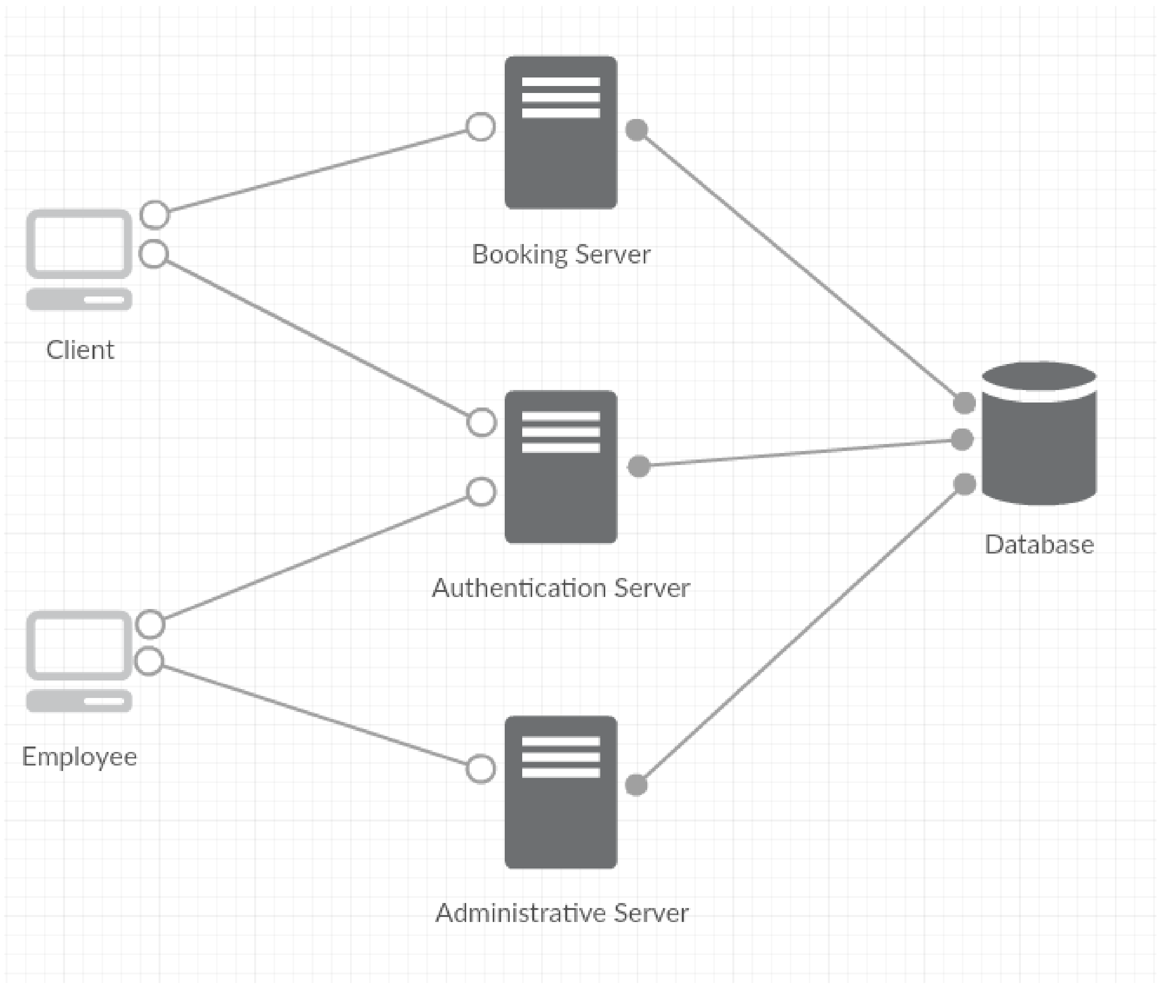
Part A: REVISIONS

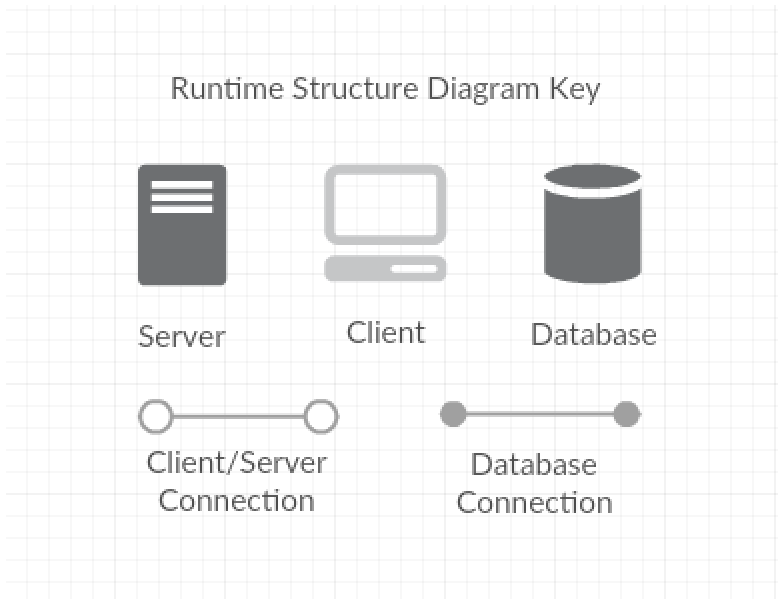
To see revisions look at Assignment1Revised.pdf in repository. All changes made are in red (except the UMD Class Diagram) and at the end of the document is a list noting all changes.

Part B: RUNTIME STRUCTURE

In this section, a runtime structure diagram illustrates how the system operates at runtime. A sequence diagram is used to show how the "cancel a booking" use case is handled.

Runtime Structure Diagram:





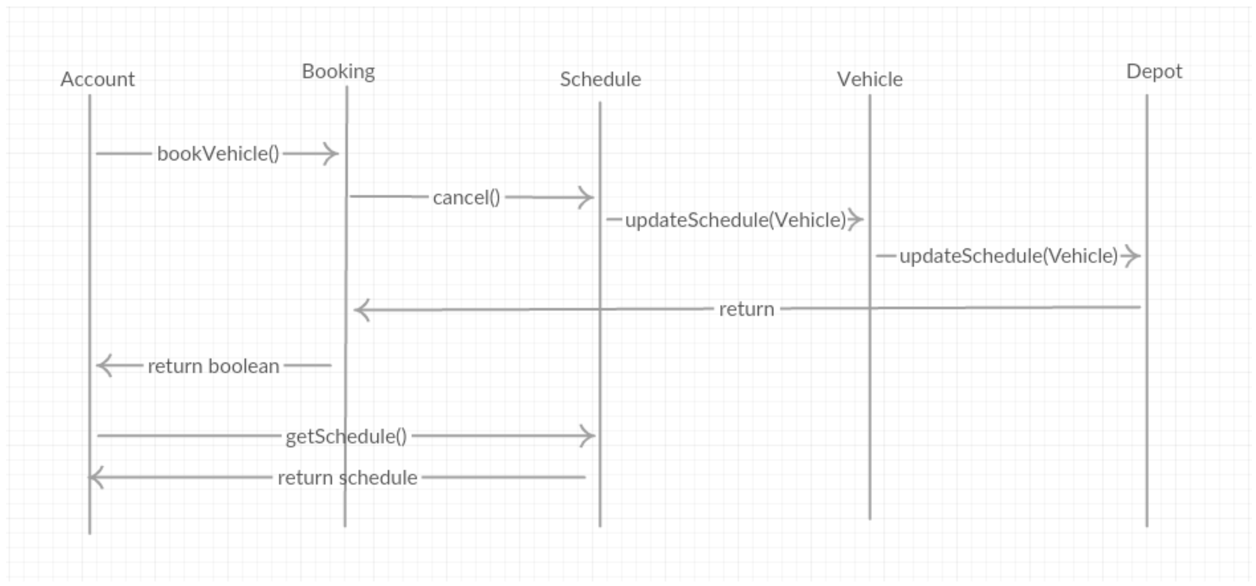
During runtime, the system will recognize two types of clients: employees and customers. When a client logs into the system, the authentication server will hand them a token based on their Role within the system. This token will allow access to either the booking server or the administrative server. The booking server allows customers to create, edit and cancel bookings with vehicles at specified locations. The administrative server allows employee to view and edit the bookings of any customer account, as well as manage vehicle inventory between depots.

There will be two different types of connectors in this system. The hollow circle connection represents connection requests between client and server. The solid circle connection symbolizes a database connector, in which the server may query the database.

For example: once a customer logs into the system, an authentication request to the authentication server is sent. This will query the database for the role in which to place the current session. Since this is a customer account, the appropriate token will be given to allow access to the booking server. From there the customer may post booking requests to the server, which will insert into the database, through a database connection.

Sequence Diagram:

Cancelling a booked vehicle



Part C: OO DESIGN

In this section, three CRC index cards are provided for three classes, along with Java pseudocode, describing properties and behaviours. The section ends with an updated UML class diagram, illustrating the relationships between the classes.

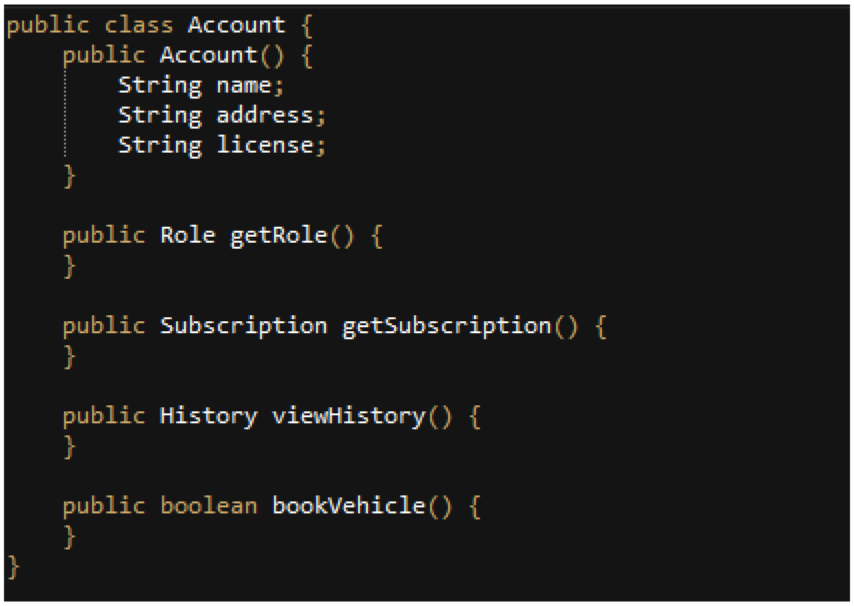
CRC Index Cards:

Account	
• Name • Address • License • Get Role • Get Subscription • View History • Book Vehicle	• Role • Subscription • Schedule • Booking

Booking	
• Account • Vehicle • Datetime • Cancel Booking • Get Vehicle • Get Datetime • Edit Booking	• Account • Vehicle • Schedule

Vehicle	
• Location • Schedule • Send Alert • View Schedule • View History	• Depot • Location • Booking • Locate Vehicle • Schedule

Class Attributes and Operations:



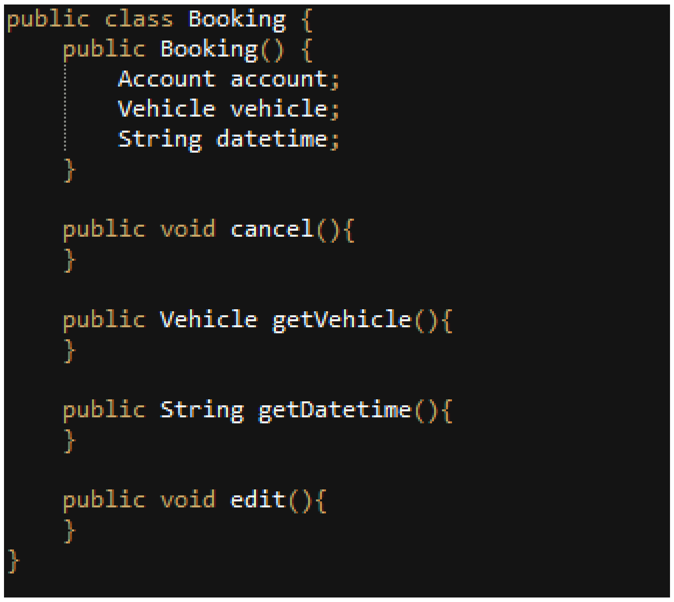
```java
public class Account {
    public Account() {
        String name;
        String address;
        String license;
    }

    public Role getRole() {
    }

    public Subscription getSubscription() {
    }

    public History viewHistory() {
    }

    public boolean bookVehicle() {
    }
}
```



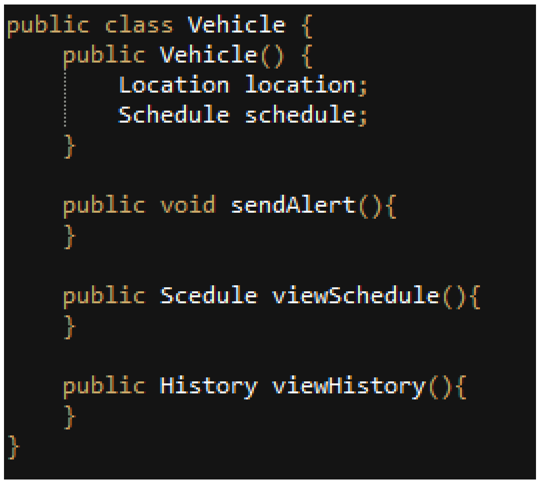
```java
public class Booking {
    public Booking() {
        Account account;
        Vehicle vehicle;
        String datetime;
    }

    public void cancel(){
    }

    public Vehicle getVehicle(){
    }

    public String getDatetime(){
    }

    public void edit(){
    }
}
```



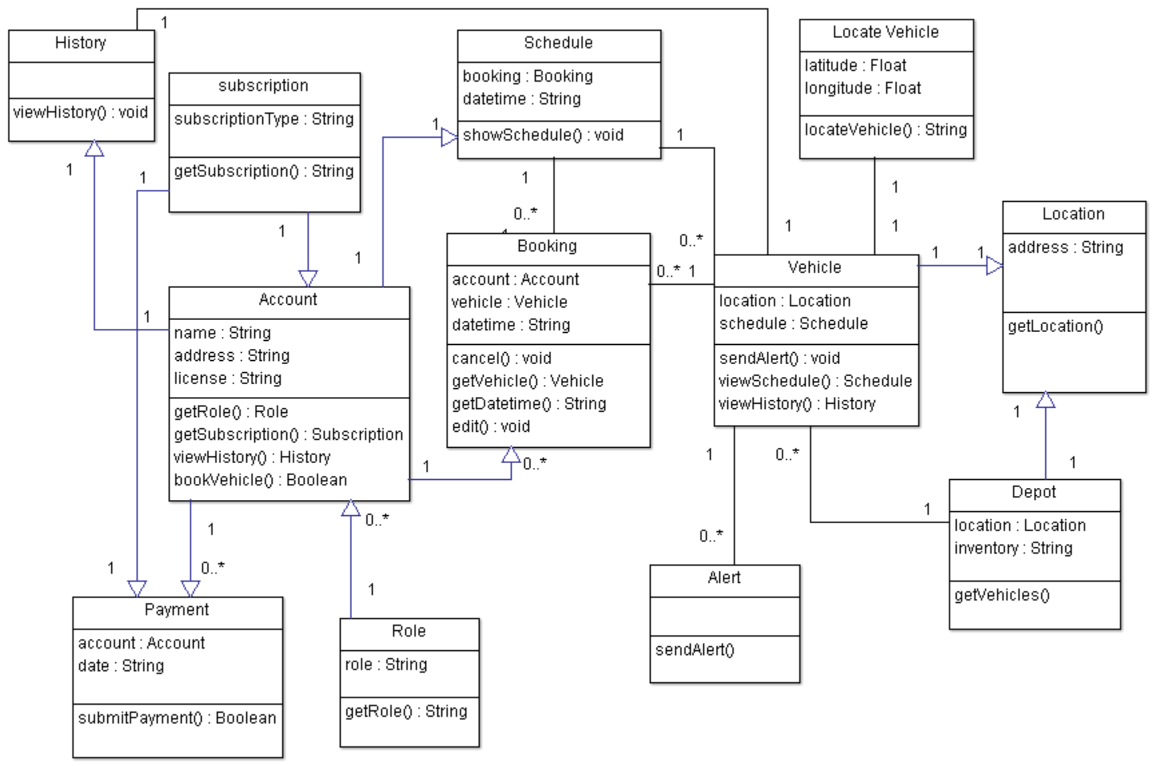
```java
public class Vehicle {
    public Vehicle() {
        Location location;
        Schedule schedule;
    }

    public void sendAlert(){
    }

    public Scedule viewSchedule(){
    }

    public History viewHistory(){
    }
}
```

UML class diagram:


History
viewHistory() : void
subscription
subscriptionType : String
getSubscription() : String
Schedule
booking : Booking
datetime : String
showSchedule() : void
Locate Vehicle
latitude : Float
longitude : Float
locateVehicle() : String
Location
address : String
getLocation()
Account
name : String
address : String
license : String
getRole() : Role
getSubscription() : Subscription
viewHistory() : History
bookVehicle() : Boolean
Booking
account : Account
vehicle : Vehicle
datetime : String
cancel() : void
getVehicle() : Vehicle
getDatetime() : String
edit() : void
Vehicle
location : Location
schedule : Schedule
sendAlert() : void
viewSchedule() : Schedule
viewHistory() : History
Depot
location : Location
inventory : String
getVehicles()
Alert
sendAlert()
Payment
account : Account
date : String
submitPayment() : Boolean
Role
role : String
getRole() : String
1
0..*