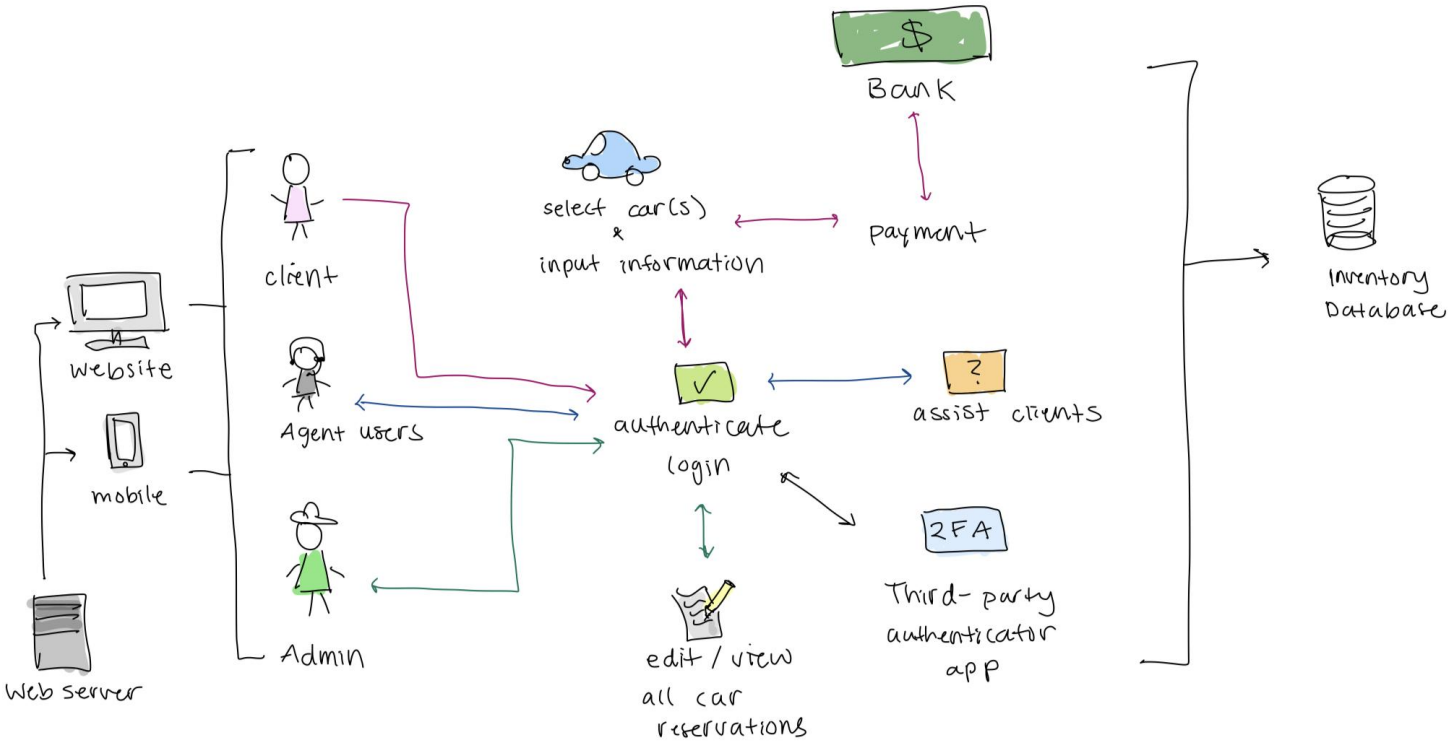
Car Rental System: Software Design Specification

By: Winnie Thong, Khoi Tran, Chloe Kershner
CS250 Group 5

Overview

The purpose of the car rental system is to provide rental car services to customers through a web and mobile application. This is to make renting cars quicker and easier for both the customer and the company. It should replace the paper printing process, reduce lines, and reduce the length of in-person appointments. The application will allow customers to see all car options, make a member account, and pay online. This document will present this system through a UML diagram and software architecture diagram. We will also further explain our illustrations.

Software Architecture System Diagram



The software architecture system diagram for the car rental system first shows that the application can be accessed through a computer or mobile device by three types of users: clients, agents, and administrators. All devices should be connected to the web. After logging into the application, the users can authenticate their login by email, phone, or a third-party authenticator application.

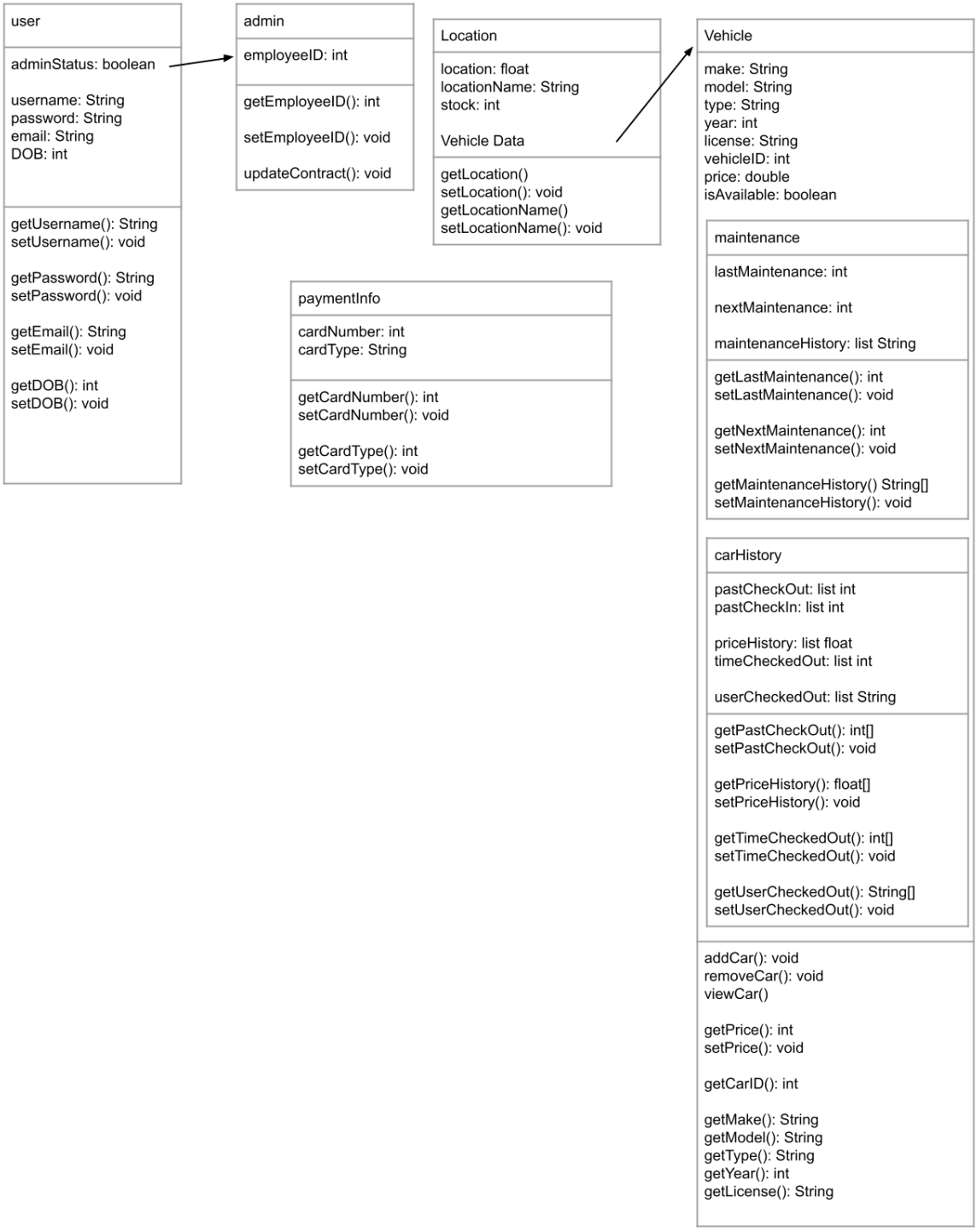
The clients will be able to select their car, fill out and upload their information, and then make a payment. The payment will be connected to their bank account unless they opt to pay in person.

The agent users are there to help clients. They can assist with any technical issues or any questions the clients may have about their car reservations.

The admin users edit the available cars in the inventory or any customer information. They can also view all car reservations, past or present, within their branch.

All information is collected and stored in their inventory database, where security is prioritized.

UML Diagram:



Class Descriptions:

user: this stores the variables and functions pertaining to the individual user of the site. It will have variables such as a username and password, their email address, and their date of birth, as

well as get/set methods for each of these. There is also a boolean that determines whether or not the user is an admin. With this boolean set to true, users can access more information and edit more things within the system.

admin: this class holds employee-specific information such as the employee ID, their contract, and their respective get/set methods.

location: This class represents vehicle location data and stock information. It contains attributes for storing the location's coordinates as a float, the location's name as a string, and the stock level as an integer. The class includes methods for getting and setting the location and location name, as well as retrieving the stock level. It serves as a fundamental component for managing and tracking vehicle locations and stock levels within a system or application.

vehicle: The class is designed to store essential vehicle information. It encompasses variables such as make (String), model (String), type (String), year (int), license (String), vehicleID (int), price (double), and isAvailable (boolean). The class provides various methods for managing vehicles, including addCar(), which adds a vehicle to the system, removeCar() for removing vehicles, and viewCar() for viewing specific vehicle details. In addition to these methods, the class offers getter and setter methods for specific attributes, allowing access and modification of attributes like price, car ID, make, model, type, year, and license. This class is accessible through each individual location class, as different locations will have different vehicles available.

maintenance: This class manages maintenance-related data. It includes attributes such as lastMaintenance (int), representing the timestamp of the last maintenance, nextMaintenance (int), indicating the timestamp for the next scheduled maintenance, and maintenanceHistory (list of strings) to log maintenance history. This class offers methods to interact with this data. You can use getLastMaintenance() to retrieve the timestamp of the last maintenance, setLastMaintenance(lastMaintenance: int) to set the timestamp for the last maintenance, getNextMaintenance() to get the timestamp for the next scheduled maintenance, setNextMaintenance(nextMaintenance: int) to set the timestamp for the next scheduled maintenance, getMaintenanceHistory() to retrieve the maintenance history as an array of strings, and setMaintenanceHistory(maintenanceHistory: String[]) to update the maintenance history by modifying the list of strings. The Maintenance class facilitates the management of maintenance information, allowing you to retrieve and modify maintenance timestamps and maintain a history log.

carHistory: The class manages historical data related to car checkouts and pricing. It includes attributes such as pastCheckOut and pastCheckIn, which are lists of integers used to track checkout and check-in timestamps. Additionally, it has priceHistory, a list of floats for recording pricing history, and timeCheckedOut, a list of integers for storing checkout duration information.

The class also manages userCheckedOut, a list of strings that records users who checked out the car. It provides methods for retrieving and setting this data, such as getPastCheckOut(), setPastCheckOut(), getPriceHistory(), setPriceHistory(), getTimeCheckedOut(), and setUserCheckedOut(). The CarHistory class is designed to facilitate the management of historical data related to car usage, enabling record-keeping and analysis within a system or application.

Development Implementation

Phase 1: Project Initiation
- Define Project Scope and Requirements: 1 week (Collaborative)
- Set Up Development Environment: 1 week (Collaborative)

Phase 2: Design and Planning
- Create Detailed Design Documents: 2 weeks (Collaborative)
- User Account Management: 1 week (Collaborative)

Phase 3: Database Design
- Database Schema Design: 2 weeks (Collaborative)
- Data Integration: 1 week (Collaborative)

Phase 4: Frontend Development
- Develop Web and Mobile Interfaces: 3 weeks (Divided)
  - Winnie (Web): 3 weeks
  - Khoi (Mobile): 3 weeks
- User Registration and Login: 3 weeks (Chloe)

Phase 5: Backend Development
- Implement Classes and System Structures: 4 weeks (Divided)
  - Winnie (Vehicle and Location): 4 weeks
  - Khoi (User and and Payment): 4 weeks
- Implement Business Logic: 4 weeks (Divided)
  - Winnie (Client Logic): 4 weeks
  - Khoi (Agent and Admin Logic): 4 weeks
  - Chloe (Payment Processing Logic): 4 weeks

Phase 6: Testing and Quality Assurance
- Unit Testing: 2 weeks (Collaborative)
- Integration Testing: 2 weeks (Collaborative)
- User Acceptance Testing (UAT): 2 weeks (Collaborative)

Phase 7: Deployment and Rollout

Deployment to Production: 2 weeks (Collaborative)

Training and Documentation: 1 week (Collaborative)

Phase 8: Post-Deployment

Maintenance and Support: Ongoing (Collaborative)

Scaling: As needed (Collaborative)