Software Design Specification

Giovanna Ehrig, SarahAnne Espinoza, Jorge Campillo

System Description - Car Rental System

The car rental system is a software solution designed to replace the pen-and-paper rental process for the car rental company, BeAvis. The system is intended to manage and conduct day-to-day operations of the company and is accessible through a mobile application and a website.

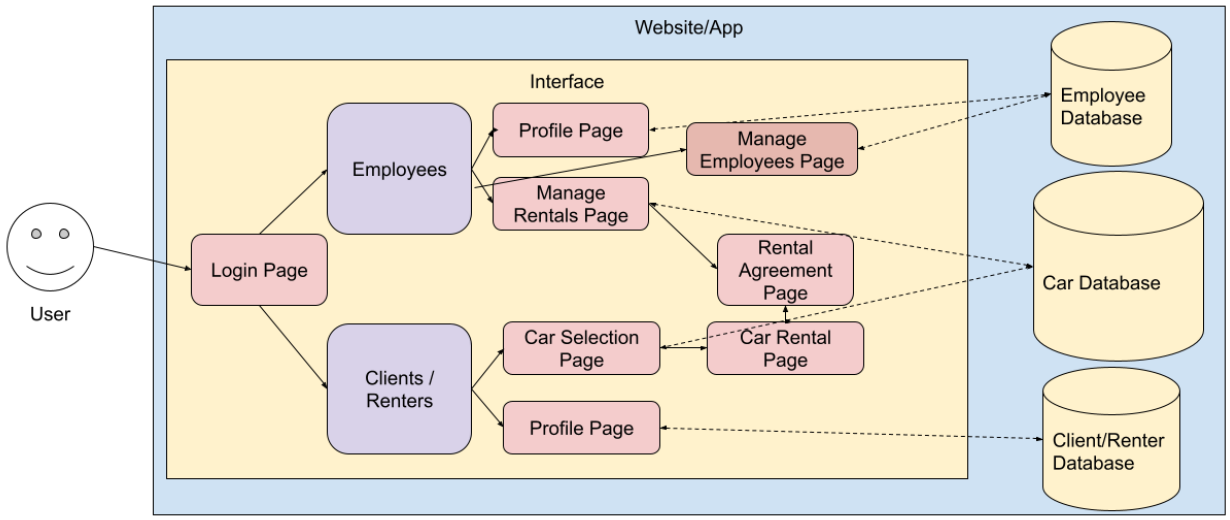
Users will need to create an account to use the system, and they must enter valid information into each data field to complete their account creation. A verification code will be sent to their email to finalize the process. Two-step authentication is not enabled by default but should be allowable through the app's settings, which can link with an authenticator app. Users can link a payment method to their account for purchases, and the payment information should be stored securely.

The system must facilitate the distribution and signing of rental agreement contracts. These contracts will be kept under secure conditions, as they will include customer information like their signature.

Employees of the company can log into the system to review customer rental status and contracts, check the fleet of cars available for rental, and update the status of cars that need maintenance.

Overall, the car rental system for BeAvis aims to streamline the rental process for customers and simplify day-to-day operations for employees, while ensuring the security and privacy of sensitive information.

Software Architecture Overview



This Architecture Diagram is a surface level view of the interactions between different parts of the system, where the smiley-face represents the user interacting with the website or app and the databases will be lists of the classes created when someone creates an account. Upon opening of the app, the user will be prompted with a login page, then depending on whether they are a client or employee, they will have access to different parts of the website or app. These pages will allow the user to have access to corresponding functions.

Employees

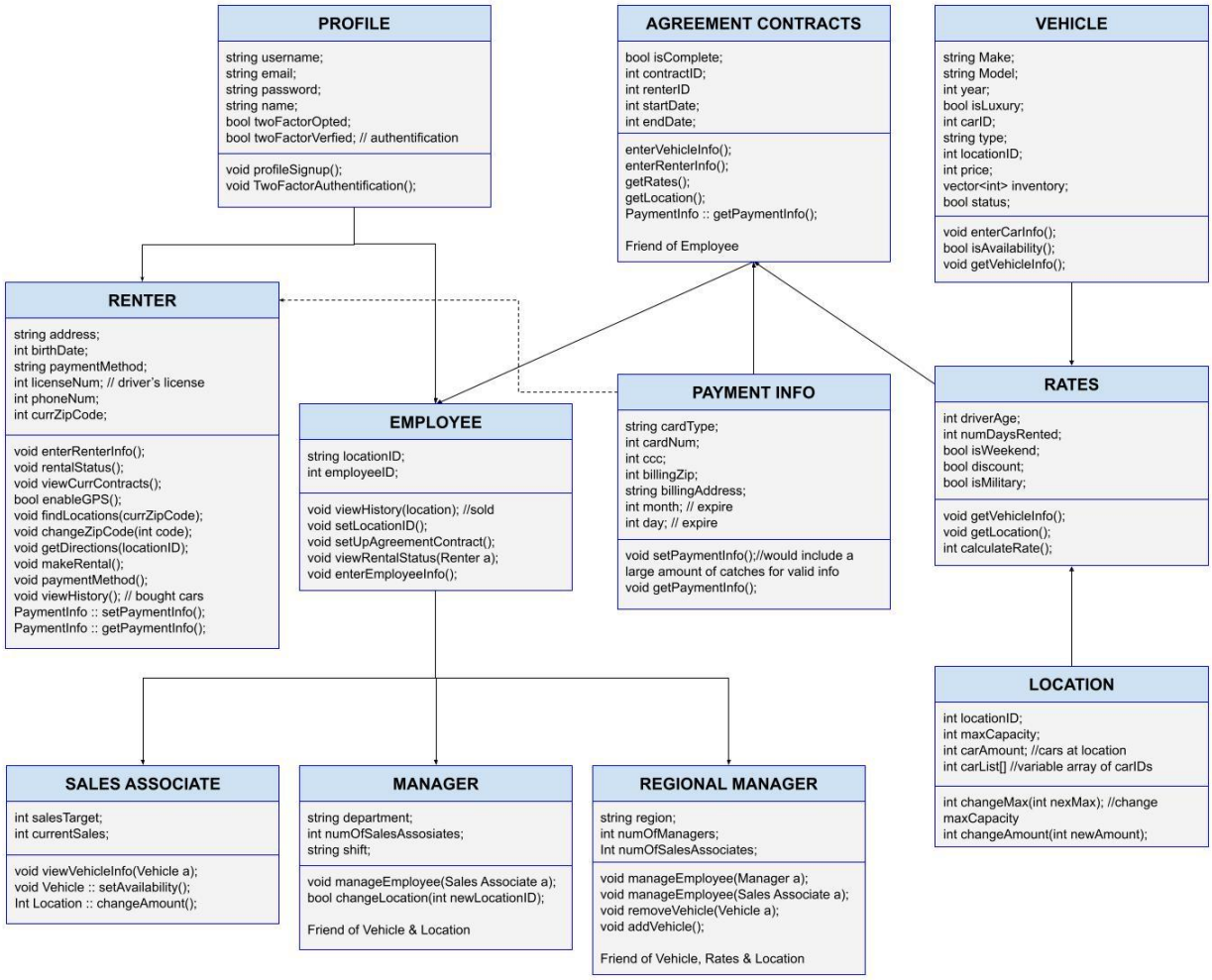
An employee on the profile page will have the ability to view and update their profile information which will access and update the employee database. An employee in the manage rentals page will be able to view and update information on the cars they are currently renting at a certain location, this will access and update the car database. The “Manage Employees Page” is darker than the others because it is a locked page only visible to those in the manager subclass of employees and is slightly different for those who are regional managers because they also have access to other lower level managers.

Clients / Renters

A client, a user given the “renter class”, on the profile page will have similar functions as the employee. However, they will have access to the client/renter database separate from the

employee database. A client on the car selection page will be able to choose a location and view all of the cars available to rent at that location. They will be able to view the car database but the arrow is a one way because clients are not allowed to update the car database. The car selection page will lead to a car rental page if they decide on which car they want to rent and then to a rental agreement page where both employees and clients can access the same page and have the car reserved for when the client meets the employee in person.

UML Class Diagram Overview



The program will have eleven classes within it, dealing with the requirements of the system. One of the master classes will be the Profile class. The Profile Class will have two subclasses, Renter and Employee. The Employee class will have three subclasses of its own: Regional Manager, Manager, and Sales Associate. The other six classes will not have any subclasses, but will rather interact with one another and the Profile master and sub-classes.

Profile Class

Each 'Profile' class will have a username, email, name and password associated with it, as well as the option to enable two-factor authentication. The client requests the option to do so through a third party software. twoFactorOpted will be true if the user has chosen to be opted into two-factor authentication and twoFactorVerified will be true if the profile has been verified. Both of the boolean functions in this class will default to false.

The function *profileSignup()* will be the function called for when the client signs up, prompting them to enter their email address, select a username and password and put in their preferred name. The function *twoFactorAuthentification()* will take the third party site and make changes to twoFactorVerified as needed if twoFactorOpted is true, otherwise the program will log the user in regardless of twoFactorVerified.

Renter Class

The 'Renter' class is the type of profile given to the users who will be searching for cars to rent. This class will hold personal information about the renter such as address, birthdate, phone number, drivers license number, payment information, and their current zip code while using the app for convenience purposes, upon request, as well as a bool to mark if they are currently renting a car.

The *enterRenterInfo()* function will prompt the user to enter information for all of the variables, where address is their home address, birthDate is their date of birth, paymentMethod is between in-person and online with a credit or debit card, licenseNum is their driver's license number, phoneNum is their mobile phone number. currZipCode will be added separately as part of the enableGPS function. The *enableGPS()* function will return true if the user says they want to allow it as well as automatically set currZipCode to the zip code of the user's current location.

Otherwise, it will ask the user to enter a zip code for currZipCode to be updated manually. The zip code can also be changed later with the function *changeZipCode(int code)*. The function *rentalStatus()* will give the user the ability to view whether or not the car is available and rented under their name. If rentalStatus returns false, the *makeRental()* function is called so users can make a rental. In this function, users will be able to browse through available cars at a given location and select a car to reserve for rental, actual rental will be finalized at the location where the employee can grant the renter permission to rent the car.

The function *viewCurrContracts()* will be called so renters can review what contracts are attached to their account and are active. The *viewHistory()* function is called for renters to review contracts attached to their account and no longer active, otherwise called their rental history. To find the closest locations, the *findLocation(currZipCode)* uses the zip code to locate them and the *getDirections(locationID)* provides the user directions to the selected location.

In order to set the renter's payment method, the *paymentMethod()* function is called, along with a call to two functions from the Payment Info class, the *setPaymentInfo()* function and the *getPaymentInfo()*, to set and retrieve the payment information respectively.

Employee Class

The ‘Employee’ class in the car rental service system represents the employees who work for the rental service company. The class contains two member variables, the locationID and the employeeID, which identify the specific location and employee within the company and is set up using the function *enterEmployeeInfo()*.

The class also includes several other member functions such as *viewHistory()* to view the rental and sales history of a particular location, *setLocationID()* to set the location for the employee, *setUpAgreementContract()* to create and set up rental agreements, and *viewRentalStatus(Renter a)* to view the status of a particular renter's rental. Overall, the Employee class plays an important role in managing the rental service company's day-to-day operations and providing excellent customer service to renters.

Sales Associate Class

The ‘Sales Associate’ class represents the employees who work directly with the renters. This class includes two member variables, the sales target and the current sales of the sales associate. The class includes a *viewVehicleInfo()* function to view the details of a specific vehicle, a *setAvailability()* function to update the availability of the vehicle for rental and a *changeAmount()* function to change how many cars are stored on location. Sales Associates also work with the Managers to ensure excellent customer service and to meet sales targets.

Manager Class

The ‘Manager’ class represents a lower-level manager who oversees the operations of a single location. This class includes three member variables, the department, number of sales associates, and the shift of the manager. The class also includes a *manageEmployee(Sales Associate a)* function to manage the sales associates at their location and a *changeLocation(int newLocationID)* to change the location of a sales associate. Managers also work with the Regional Manager to ensure smooth operations of the rental service company in their region.

Regional Manager Class

The ‘Regional Manager’ class represents a high-level manager who oversees multiple locations within a specific region. This class includes two member variables, the region and the number of managers and sales associates within that region. The class also includes several member functions such as *manageEmployee(Manager a)* and *manageEmployee(Sales Associate a)* to manage the employees under them and *addVehicle()* and *removeVehicle(Vehicle a)* to add and remove vehicles from the database of available cars. The Regional Manager class acts as a bridge between the lower-level managers and the upper management of the rental service company.

Agreement Contracts Class

The ‘Agreement Contracts’ class is responsible for managing the rental agreements between renters and the car rental service. Each instance of the ‘AgreementContracts’ class represents a single rental agreement, which may involve one or more vehicles and one or more rental transactions. The class contains attributes such as isComplete, which indicates whether the rental agreement has been fully executed, and startDate and endDate, which define the start and end dates of the rental period.

The class also has functions for entering and retrieving information about the rented vehicles and renters, *enterVehicleInfo()*, *enterRenterInfo()*, and a call to the associated Payment Info class’s *getPaymentInfo()* function, as well as functions for obtaining the rental rates and location information, *getRates()* and *getLocation()*. By managing the rental agreements through the AgreementContracts class, the car rental service can easily track and manage all aspects of the rental process, from initial bookings to final transactions.

Payment Info Class

The ‘Payment Info’ class is a crucial part of the rental car application, responsible for storing and processing users' payment information. This class contains several variables to represent different fields of payment information, including card type, card number, CVV code, billing zip code, billing address, and expiration date.

The *setPaymentInfo()* function is used to set the payment information for a user, and it includes a variety of checks to ensure that the payment data is valid and complete. This function checks that the credit card number is in a valid format and matches the card type, that the billing address and zip code match the card's billing information, and that the expiration date is in the future.

The *getPaymentInfo()* function is used to retrieve the payment information stored in the Payment Info object. This function may be used to display the payment information to the user or to pass it the agreement contract in order to receive payment from the renter.

Vehicle Class

The ‘Vehicle’ class represents a rental car in the rental car application. It contains variables for make, model, year, car ID, type, location ID, price, inventory, and status. The class provides functions for entering car information, checking availability, and retrieving vehicle information.

The *enterCarInfo()* function allows the employee to enter the car information, and the *isAvailability()* function checks if the car is available for rental. The *getVehicleInfo()* function retrieves the car information, which includes make, model, year, and price. This class ensures proper tracking and management of rental cars, allowing for a smooth rental process for users and employees.

Rates Class

The ‘Rates’ class is to quickly calculate the cost of renting a vehicle, using the driver’s age, the number of days rented, and whether or not that includes any weekends. It also checks if the renter has a discount or is a member of the military. The functions for the class are *getVehicleInfo()* and *getLocation()*, which get the vehicle info and where it is stored, and *calculateRate()* to work out the rate.

Location Class

The ‘Location’ class is the class that deals with information about the rental locations. The attributes in the class are locationID, a unique location code, maxCapacity, the maximum number of cars that can be parked at location, carAmount, how many cars are actually parked at that location, and carList, a list of all the carIDs for the cars parked on location. The functions for the class are *changeMax(int newMax)*, which is called to change maxCapacity, and *changeAmount(int newAmount)*, for changing carAmount.

Development Plan and Timeline

Development will be done in two phases. Phase One will focus on the development of the classes individually, Phase Two will focus on the interaction between the classes and Phase Three will be focused on developing the UI and testing for bugs and errors.

During Phase One, the classes will be divided into three groups based on the similarities between them. Group A is focused on the Profile class and all of its subclasses, due to the requirements of the classes all falling under the Profile database. Group B is focused on the Vehicle, Location and Rates class, as these are the classes that deal with other two databases of the system. Group C is focused on the last two classes, the Agreement Contracts class and the Payment Info class, due to the nature of the information that is stored within the classes.

Phase One is completed once all three Groups report that they have completed their classes satisfactorily, after which Phase Two will begin. Phase Two will continue to have the established Groups, but will require cooperation between them, as it is when the intended interactions are coded.

Upon completion of Phase Two, Phase Three will begin, focused on the construction of the user interface. The user interface will include ensuring that the website and app are user friendly, testing that the classes function as intended, and fixing any bugs and errors not caught in the prior phases.

Completion of Phase Three will mark the end of development and the rolling out of the system to general users. All phases will ideally be completed in a timeframe of eighteen to twenty-four months, six to eight months for each phase, depending on how many setbacks there may end up being. Reconsideration of this timeline will be done if any phase takes longer than eight months or if it is completed before six months.