BeAvis Car Rental System Software Design Specification

Prepared by: Brandon Ho, Sumeya Sayd, Nitin Chatlani

April 2, 2023

Overview

BeAvis's car rental system seeks to replace the traditional pen-and-paper rental process with a new, modern mobile application to make the process as smooth and efficient as possible. Past rental processes have proved to be time costly and vulnerable to many mistakes and complications. The main purpose of the software system is to conduct and manage a car rental company, BeAvis. It will contain the various functions essential for the day-to-day management of BeAvis. Employees of BeAvis would be able to log into the system to access the management data and assist with customer service. This includes pulling up customers to review their rental status and contracts, checking on the amount and types of cars both currently available for rental and on the road, and updating the status of any cars whenever they are removed or needed for maintenance. On the customer side, users will have the ability to locate BeAvis establishments near them as well as directions to their choice of establishments if they intend to request a rental. There will also be an additional option regarding the rental process, including making a rental, viewing rental history, and looking up locations. The user will be able to make purchases on their side of the software system. During the acquisition of the rental car, the system facilitates the distribution and signing of rental agreement contracts. This software system will change how managers, employees, and customers interact with the rental process. However, this will not completely remove the pen-and-paper of the rental process, as if a customer decides to show up at a BeAvis establishment and chooses to fill out the necessary rental paperwork, they will be

allowed to. This document will showcase the user and system requirements of the BeAvis Car Rental System and demonstrate the specifics of how users will interact with the system and the technological aspects behind the system.

Software Architecture Overview

CarRentalStore: Class that would include the location of the store the customer would rent the vehicle from and the cars available to be rented at the specific location.

- *Attributes:*
    - Location: a String variable that contains the name of the location of the Rental Store
    - CarsAvailiable: string array of models are cars available

RentalSystem: Components of the vehicle that would be rented, its availability, and its history.

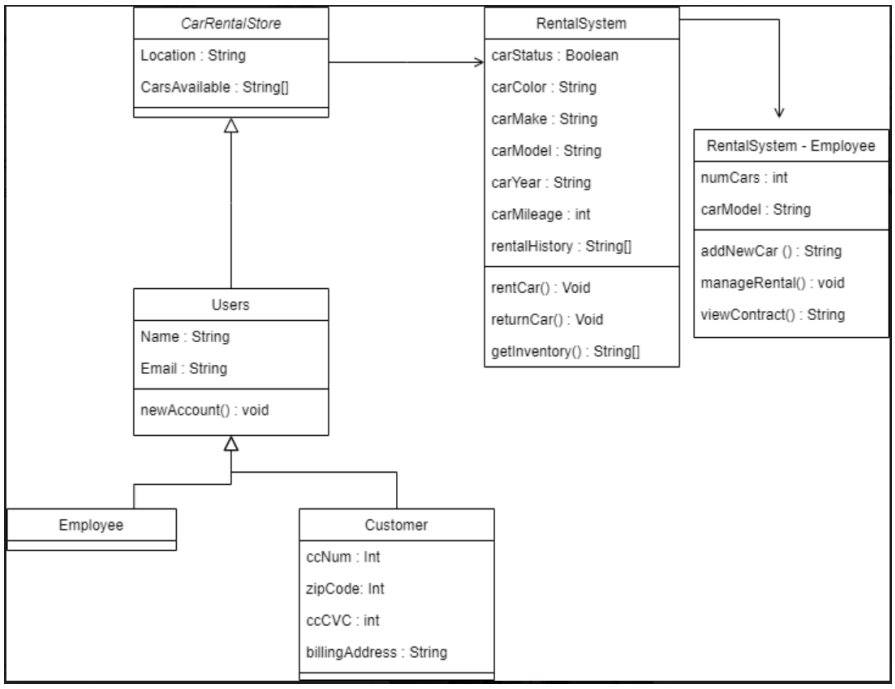
- *Attributes:*
    - carStatus: a boolean that is true when the car is available to rent, false when otherwise
    - carColor: a string that contains the color of the car
    - carMake: a string that contains brand of the car
    - carModel: a string that contains the model of the car
    - carYear: a string that contains the year which the model of the car is from
    - carMileage: an integer that represents the mileage of the rental car
    - rentalHistory[]: a string array that contains the ID of previous owners of the car
- *Functions:*
    - rentCar(): changes the carStatus of a specific car to true, called when the car is being rented
    - returnCall(): changes the carStatus of a specific car to false, called when the car is being returned
    - getInventory(): returns an array of strings that contains the carModel of the available cars in the rental system
- Employee: Comprises additional information from the RentalSystem class that only employees are able to access.
    - *Attributes:*
        - numCars: an integer that represents the number of cars that are currently available at the rental location
        - carModel: a string that contains the model of the car
    - *Functions:*
        - addNewCar(): takes in a string parameter that represents the car model to add to the car inventory
        - manageRental(): allow access to the rental system to add, remove, or change rental status

- viewContract(): takes in a string parameter that represents the customer ID and outputs the contract of the customer

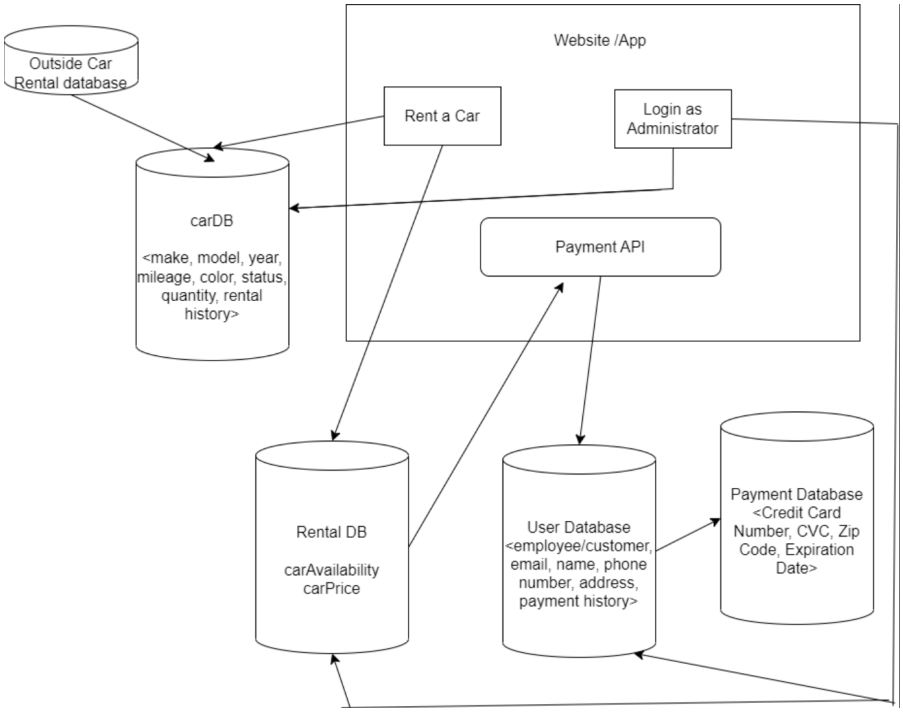
Users: These classes would contain information about users utilizing the system.

- *Attributes:*
  - name: string that represents the name of the user account
  - email: string that represents the email associated with the account
- *Functions:*
  - newAccount(): creates a new account for new users
- Employees: Connects directly to the users' class and consists of employees' personal information.
- Customers: Stems from user class and holds information about customers' payment information.
  - *Attributes:*
    - ccNum: integer that represents the credit card number of user account
    - zipCode: integer that represents the zip code of user account
    - ccCVC: integer that represents the CVC code of credit card of user account
    - billingAddress: string that represents the address of the customer's billing address

UML Diagram:

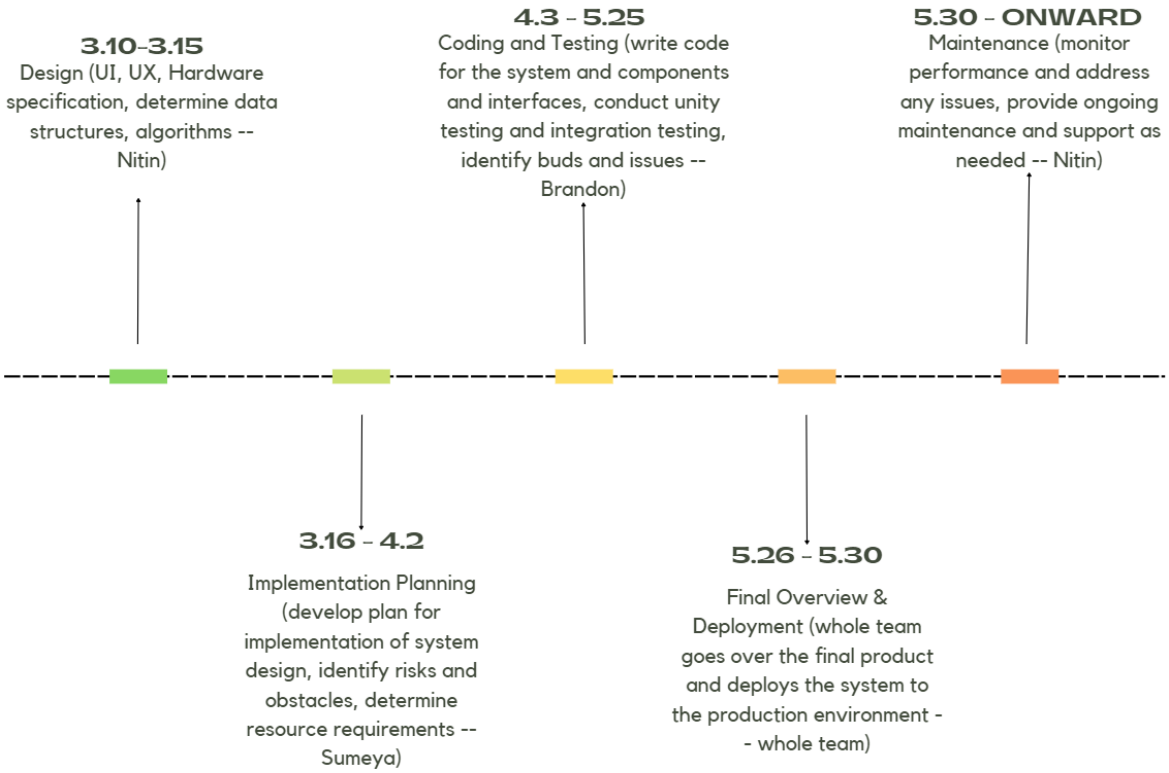


Software Architecture Diagram:



Description: This diagram depicts the car rental system website and app that will be utilized by the user to act as an interface between the user and the software system. Within this interface, users will be given the option to log in, rent a vehicle, and utilize the payment API. The user database, which stems directly from the payment API and the login, contains information about the users of the system, such as their email, name, phone number, address, and payment history. The payment database, which branches from the user database, contains the customer's payment information, such as their credit card number, the CVC, their zip code, and the expiration date of their card. The rental database, which is connected to all three components of the interface, comprises all of the current vehicle’s prices and availability. The car database, which is connected to the outside car rental database, includes the make, model, color, year, mileage, status, rental history, and the number of cars.

Development Plan & Timeline:



Unit Testing

First Test

We are testing the functions of getInventory() and viewContract().

RentalSystem.getInventory("San Diego") → should return a array where each index is a string of the car models available to rent at the specific location

Result: ["Tesla1", "Tesla2", "Tesla3", "Honda1", "Honda2", "Jeep1", "Jeep3", "Jeep5"]

- If the string parameter is not a valid location or not a string variable, should return an error, prompting the user to input a valid string and location

Employee.RentalSystem.viewContract("user382394") → should return a string containing the rental contract assigned to the userID

Result: "I agree to return the car on March 25, 2023 at 1:00 pm without any damages and with the gas tank filled -

- If the string parameter is not a valid user ID or not a string variable, should return an error, prompting the employee to input a valid user ID

Integration testing

First Test

Scenario - Checking to see if a list of cars with specific models are available for rent

Test Case - RentalSystem.getInventory(carModel)

Feature tested - Retrieving inventory

Result - lists all honda model cars within the system that are available for rent

Second Test

Scenario - Types of cars that are available

Test Case - CarRentalStore.CarsAvailable(carColor)

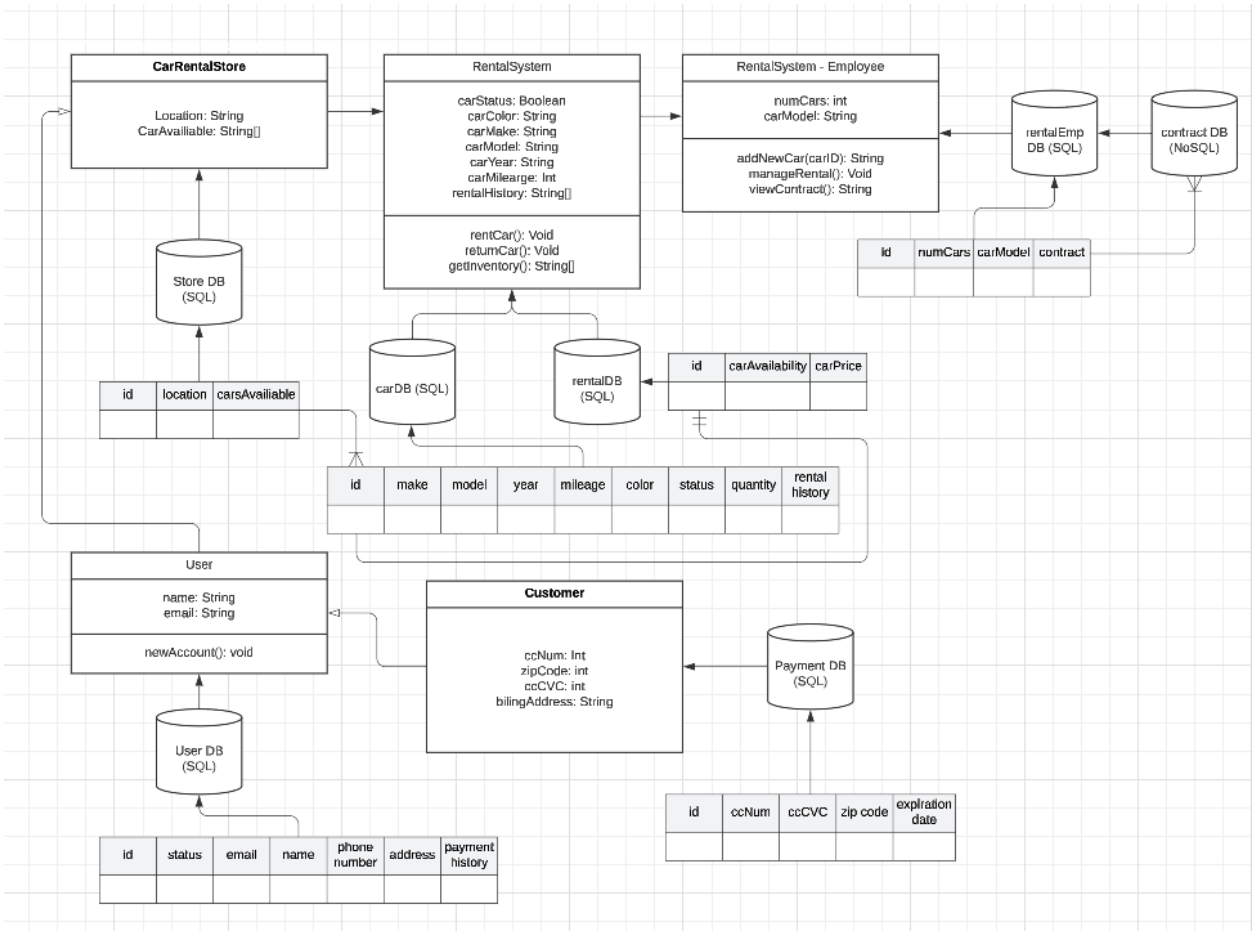
Feature tested - Cars available

Result - lists the colors of the cars that are available for rent

System Testing

1. rentCar()
    - Verify that customers can easily select and rent a car from the available inventory, and that the car's status is updated to "rented".
    - Confirm that customers are prompted to provide their contact and payment information before renting a car, and that the rental price is accurate and includes any additional options selected.
2. returnCar()
    - Ensure that the return process is straightforward and efficient for customers, and that they receive a confirmation of their return.
    - Check that the car's status is updated to "available" when it is returned to the designated location.
3. getInventory()
    - Check that the system provides an accurate and up-to-date inventory of available cars, including the model, rental price, and any additional options.
    - Verify that the inventory matches the actual inventory at the rental shop.
4. addNewCar()
    - Confirm that the Employee class can successfully add a new car to the inventory, and that the car model and number of cars are updated accordingly.
    - Check that the newly added car appears in the inventory list and matches the actual car added.
5. manageRental()
    - Verify that the Employee class can change the rental status of a car as needed, such as marking it as "unavailable" for maintenance or repairs.
6. viewContract() (in the Employee class)
    - Check that the viewContract() function displays the rental contract of a specific customer, given their customer ID.
    - Verify that the rentalHistory attribute of the RentalSystem class is updated to reflect the rental history of the specific car.
7. newAccount() (in the Users class)
    - Ensure that new users can easily create an account with accurate and up-to-date information.
    - Verify that the account information is saved correctly and can be used for future rentals.
8. viewContract() (in the Customers class)
    - Check that the viewContract() function displays the rental contract of the customer who is currently logged in.
    - Verify that the rentalHistory attribute of the RentalSystem class is updated to reflect the rental history of the specific car.

SQL Database Design



We will have multiple databases for different classes. The StoreDB will be a SQL database containing information about the location and if there are cars available, presented in a boolean. This database is in a one to many relationship with the carDB (SQL). In the car database, each car will be represented by an ID and will list out its make, model, year, mileage, color, status, quantity, and rental history. The carDB will be under the RentalSystem class, alongside the rentalDB (SQL). The rentalDB also has a one to one relationship with carDB and will show the car's availability and the car price. We chose to have two separate databases for the car because we decided that we want to show the specific information at different times. The rentalDB will be shown to the customer at a first glance, while the more specific details of the car will be given to the customer if they want it from the carDB. The employee rental class also will contain a rentalEmp DB (SQL) that will provide the employees a database of the car id, number of cars, car model, and rental contracts of the car. Because the rental contracts will take a large amount of space, we decided to store the contracts in another NoSQL database. Under the user class, there will be a userDB (SQL), that will have information of the user, such as their status, email, name, phone number, address, and payment history. The customer class will hold a payment database in SQL, that will provide the credit card number, CVC number, zip code, and expiration date. The

benefit of having multiple databases for different class is that it will be easier to access and read and it is more organized. We could store some large data files in other NoSQL databases.