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.

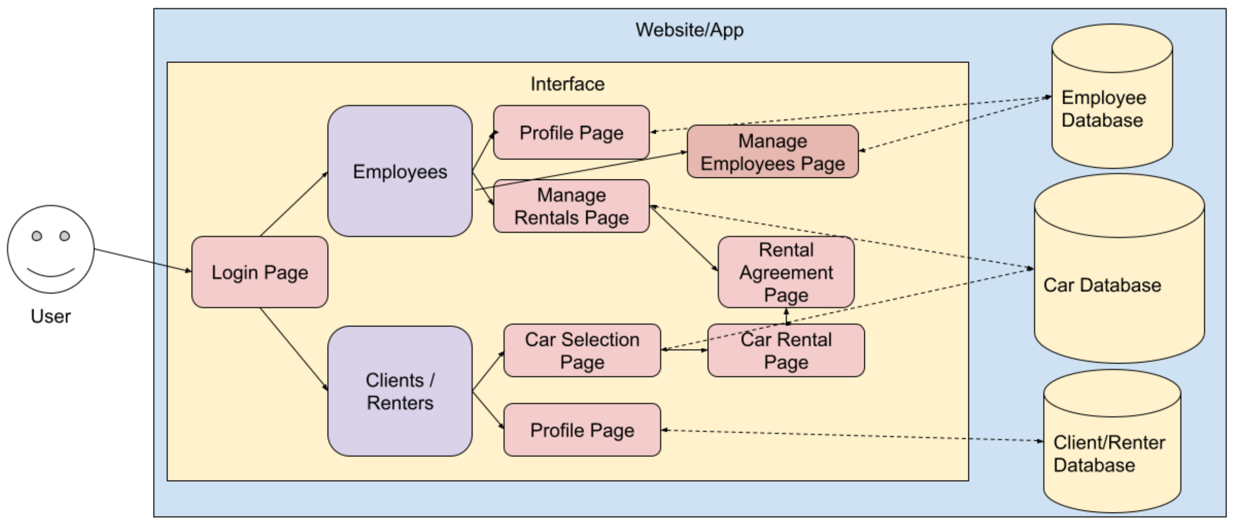
Users will need to create an account to use the system, and they must enter valid information into each data field to complete their account creation. A verification code will be sent to their email to finalize the process. Two-step authentication is not enabled by default but should be allowable through 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.

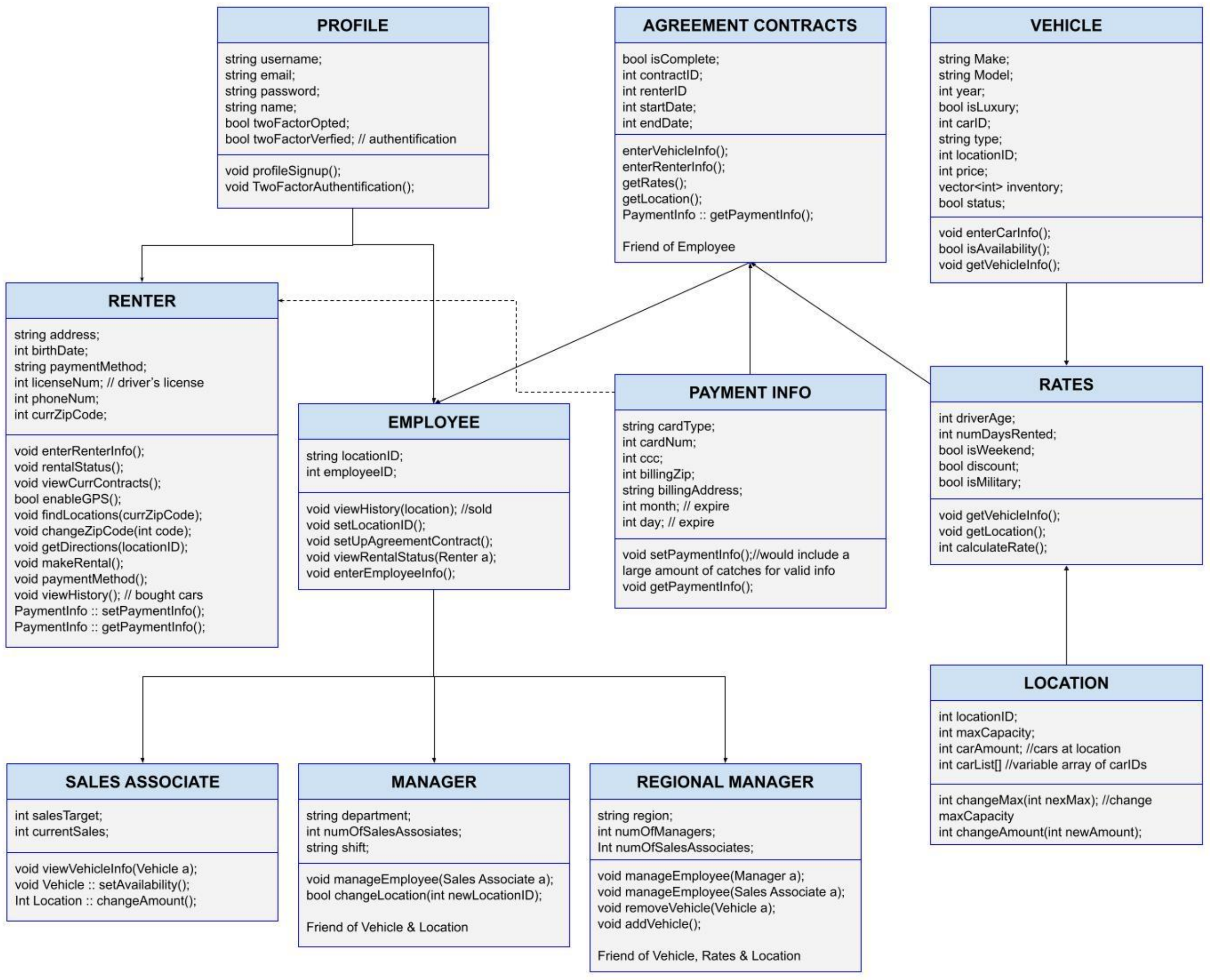
Employees

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.

SDS Test Plan

The car rental system is a comprehensive application designed to facilitate the process of renting vehicles to customers, managing fleet inventory, and streamlining reservation and billing procedures. The primary objectives of the test plan for this system are to verify its functionality and usability, ensuring a seamless and efficient user experience for both customers and employees. By executing a thorough testing process, we aim to identify and resolve potential issues and ensure that the car rental system meets the highest quality standards before deployment.

The test plan is broken down into three sections, each focusing on a different level of complexity for testing. Each section has three different test sets, using different parts of the system and testing them in different ways. Unit tests were used to test specific methods, functional tests tested communication between classes and system tests tested the actions expected to be performed on the system daily.

Unit Test Sets

Unit Test Case: Class *'Vehicle*,' Method isAvailability()

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
1	bool isAvailability();	none	Vehicle is available	true	true	PASS
2	bool isAvailability();	none	Vehicle is available	true	false	FAIL
3	bool isAvailability();	none	Vehicle is not available	false	true	FAIL
4	bool isAvailability();	none	Vehicle is not available	false	false	PASS

The function *isAvailability()* is a boolean under the vehicle class. Its purpose is to allow the user to know whether or not the vehicle is available to be rented out. If the vehicle is not available to be rented, the function should return false. If the vehicle is available to be rented, the function should return true. The test set validates whether or not the function holds true when the vehicle is available and when the vehicle is not available. Since the function is assigned to a vehicle class, it will not be usable if the vehicle does not exist, therefore there need not be a check that the vehicle is valid.

Tests:

1. Positive
2. False Negative
3. Negative
4. False Positive

Unit Test Case: Class *'Payment Info*,' Method getPaymentInfo()

Test #	Being Tested: getPaymentInfo()	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
1	Valid Payment Info	N/A	A user with valid payment information saved in their account	Retrieved valid payment information (card type, card number, CVV, billing zip code, billing address, and expiration date)	Valid payment information retrieved for the user	Pass: This test checks if the getPaymentInfo() function retrieves the correct and valid payment information for the user.

Test #	Being Tested: getPaymentInfo()	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
2	Invalid Payment Info	N/A	A user with invalid or incomplete payment information saved in their account	An error message or exception indicating that the payment information is invalid or incomplete	Error message received for invalid payment information	Pass: This test checks if the getPaymentInfo() function handles invalid or incomplete payment information appropriately.
3	No Payment Info	N/A	A user with no payment information saved in their account	An error message or exception indicating that there is no payment information available for the user	Error message received for missing payment information	Pass: This test checks if the getPaymentInfo() function handles cases where there is no payment information available for the user.
4	Expired Payment Info	N/A	A user with expired payment information saved in their account (e.g., credit card expiration date has passed)	An error message or exception indicating that the payment information is expired	Error message received for expired payment information	Pass: This test checks if the getPaymentInfo() function handles cases where the payment information is expired.

The unit tests for the getPaymentInfo() function cover a range of scenarios to ensure the function performs as expected. These tests assess the function's ability to handle valid payment information, invalid or incomplete payment information, missing payment information, and expired payment information. By simulating these different situations, the tests evaluate the function's correctness and its ability to handle various types of input data.

The getPaymentInfo() function is being tested by simulating user accounts with different payment information states. In each test, the function is expected to either retrieve valid payment information or return an appropriate error message or exception based on the given precondition. These tests help ensure that the function can handle different error cases gracefully, providing useful feedback to the user while maintaining the system's overall stability and reliability.

Unit Test Case: Class *'Renter*,' Method enterRenterInfo()

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
1	Test With Completely Correct Input	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 E9305744 (619) 501-9505	Renter Profile should be created	As Expected	PASS

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
			68506			
2	Test With All Incorrect Input	Basic Profile Account Already Created	Use only spaces in address bar 15/32/2063 E@)^7>4 (GlH) 5oT-95Q5 6e5O6	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
3	Test With No Input	Basic Profile Account Already Created	Leave address blank Leave birthday blank Leave license number blank Leave phone number blank Leave zip code blank	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
4	Test With No Address Input	Basic Profile Account Already Created	Leave address blank 03/14/1985 E9305744 (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
5	Test Address With Only Spaces	Basic Profile Account Already Created	Use only spaces in address bar 03/14/1985 E9305744 (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
6	Test With No Birthday	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE Leave birthday blank E9305744 (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
7	Test Birthday With Letters	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE Iy/QO/Ytfg E9305744 (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
8	Test Birthday With Special Characters	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE \$/%@/!*** E9305744 (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
9	Test Birthday With Year Out Of Range	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 07/13/2052 E9305744 (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
10	Test Birthday With Month Out Of Range	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 18/16/2000 E9305744 (619) 501-9505	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
			68506			
11	Test Birthday With Day Out Of Range	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 06/56/2001 E9305744 (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
12	Test With No License Number	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 Leave license number blank (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
13	Test License Number With Special Characters	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 5124#(&* (619) 501-9505 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
14	Test With No Phone Number	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 E9305744 Leave phone number blank 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
15	Test Phone Number With Letters	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 E9305744 (987) 76g-Ruew 68506	Renter Profile should not be created, incorrect fields highlighted	Incorrect field was not highlighted and a renter profile was made	FAIL
16	Test Phone Number With Letters	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 E9305744 (JIF) 76g-9uew 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
17	Test Phone Number With Special Characters	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 E9305744 (&84) 45*-^8$% 68506	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
18	Test With No Zip Code	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 E9305744 (619) 501-9505 Leave zip code blank	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS
19	Test Zip Code	Basic Profile	739 North Leatherwood Street	Renter Profile	As Expected	PASS

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
	With Letters	Account Already Created	Lincoln, NE 03/14/1985 E9305744 (619) 501-9505 tgH67	should not be created, incorrect fields highlighted		
20	Test Zip Code With Special Characters	Basic Profile Account Already Created	739 North Leatherwood Street Lincoln, NE 03/14/1985 E9305744 (619) 501-9505 $&87#	Renter Profile should not be created, incorrect fields highlighted	As Expected	PASS

Testing Steps:

1. Confirm email address
2. Click on the ‘Next’ button
3. Fill out the five required fields
4. Click the ‘Finish’ button
5. Be sent to the Profile Home page

This unit test set tested the function enterRentalInfo(), which requests for five fields to be filled out by the user, each with its own restrictions. The address has the least, only requiring it to be filled out. The license number accepts alphanumeric characters, throwing an error if special characters are inputted. The last three fields only accept numeric characters, throwing errors if any other characters are inputted into the field.

Twenty tests were made, one being the control test with entirely correct input and output and the other nineteen tested the restrictions of the input fields. Of those nineteen tests, only one gave an unexpected result. The one failed test was testing the phone number with letters, using five numbers followed by five letters. The input was accepted and a profile was made instead of the field for inputting the phone number highlighted. The next test was done after the bug was patched and performed as expected.

Functional Test Sets

Functional Test Case: View Vehicle Information for a Specific Vehicle (Employee Function)

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
1	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with invalid license plate format (e.g., "AA1234X")	Error message indicating that the license plate format is invalid	Display of vehicle information with incorrect license plate format	Fail: Fails to validate license plate format, displaying incorrect vehicle
2	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with empty or uninitialized data attributes	Error message or exception indicating that the vehicle data is invalid or incomplete	Display of vehicle information with empty or uninitialized values	Fail: Fails to validate the completeness of the vehicle data and displays incorrect vehicle information.
3	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with incomplete attributes (e.g., missing make or model)	Error message or exception indicating that some vehicle information is missing or incomplete.	Error message or exception indicating that some vehicle information is missing or incomplete.	Pass: Handles incomplete vehicle information and displays an error message or exception.
4	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with invalid attributes (e.g., negative year value)	Error message or exception indicating that some vehicle information is invalid.	Error message or exception indicating that some vehicle information is invalid.	Pass: Handles invalid vehicle information and displays an error message or exception
5	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with special characters in the make or model	Error message or exception indicating that some vehicle information contains invalid characters.	Error message or exception indicating that some vehicle information contains invalid characters.	Pass: Handles invalid characters in vehicle information and displays an error message or exception.
6	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with a negative value for the 'pricePerDay' attribute	Error message or exception indicating that the price per day value is invalid	Display of vehicle information with negative price per day value	Fail: Fails to validate price per day value and displays it without error.

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
7	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with all attributes set to empty strings	Error message or exception indicating that some vehicle information is empty.	Error message or exception indicating that some vehicle information is empty	Pass: Handles empty vehicle information and displays an error message or exception.
8	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with past end-of-life (e.g., decommissioned)	Display of vehicle information with a warning message about the vehicle's decommissioned status.	Display of vehicle information with a warning message about the vehicle's decommissioned status.	Pass:
9	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with exceptionally high mileage (e.g., 600,000 mi)	Display of vehicle information with a warning message about the vehicle's high mileage.	Display of vehicle information with a warning message about the vehicle's high mileage.	Pass: Displays vehicle information for a high-mileage vehicle along with a warning message.
10	viewVehicleInfo(Vehicle a)	Vehicle a	Vehicle object with an invalid 'year' attribute (e.g., a future year or a year earlier than the invention of cars)	Error message or exception indicating that the year value is invalid	Display of vehicle information with incorrect year value	Fail: Fails to validate the year value and displays incorrect vehicle information

Testing Steps:

1. Open the Car Rental Application.
2. Navigate to the 'View Vehicle Info' section.
3. Enter the Vehicle ID in the input box.
4. Click the 'Search' button to retrieve the vehicle information.
5. Review the displayed vehicle information to verify if it matches the expected output.

The tests for the 'viewVehicleInfo()' function aim to assess its ability to handle various scenarios, ensuring accurate vehicle information is displayed while managing user errors and incomplete data. The primary objective is to verify that the function can correctly process valid vehicle information and display the corresponding details, as well as handle invalid Vehicle IDs, vehicles with incomplete data, and vehicles with no data.

These tests not only evaluate the successful execution of the function but also the system's ability to manage errors gracefully. The tests include scenarios where the user inputs incorrect or non-existent Vehicle IDs, as well as situations where the vehicle object itself is missing data or is empty. Through these tests, the car rental system demonstrates its resilience and ability to handle a range of user inputs and data conditions, ensuring a seamless and user-friendly experience.

Functional Test Case: View The Current Rental Status of a Renter (Employee function)

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass /Fail)
1	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter can rent and is renting	Renter can rent and is renting	PASS
2	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter can rent and is renting	Renter can rent and is not renting	FAIL
3	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter can rent and is renting	Renter cannot rent	FAIL
4	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter can rent and is not renting	Renter can rent and is not renting	PASS
5	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter can rent and is not renting	Renter can rent and is renting	FAIL
6	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter can rent and is not renting	Renter cannot rent	FAIL

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass /Fail)
7	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter cannot rent	Renter cannot rent	PASS
8	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter cannot rent	Renter can rent and is renting	FAIL
9	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account (We assume we are viewing the correct renter)	Renter cannot rent	Renter can rent and is not renting	FAIL
10	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account	Correct renter and correct status	Correct renter and correct status	PASS
11	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account	Correct renter and correct status	Correct renter and incorrect status	FAIL
12	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account	Correct renter and correct status	Incorrect renter and correct status	FAIL
13	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter must have an account	Correct renter and correct status	Incorrect renter and incorrect status	FAIL
14	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter doesn't have an account	Renter not found	Renter not found	PASS
15	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter doesn't have an account	Renter not found	Renter and status are given	FAIL
16	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter has an account	Renter and status are given	Renter and status are given	PASS
17	void viewRenterStatus(Renter a)	A specific renter (r1)	The renter has an account	Renter and status are given	Renter not found	FAIL

The function *viewRenterStatus(Renter a)* allows an employee to view whether a renter can rent and also if they are currently renting a car. The first 9 tests, in darker blue, address whether or not the correct information is being displayed. Given a certain renter, we should be able to determine their status. For this we have 9 different outcomes, so we counter with 9 different tests.

Because this is a functional test, it is important to check input functionality as well. The next 8 tests, in light blue, make sure that regardless of the input the function works as intended.

Functional Test Case: View The Currently Rented Cars At Location

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
1	Test With Completely Correct Input	User Is Logged Into A Employee Account	000111	Displays rented cars at location	As Expected	PASS
2	Test With No Input	User Is Logged Into A Employee Account	Leave field blank	Displays a request for input	Displayed all rented cars in the system	FAIL
3	Test With No Input	User Is Logged Into A Employee Account	Leave field blank	Displays a request for input	As Expected	PASS
4	Test With Letters	User Is Logged Into A Employee Account	iuAewQ	Displays a message for invalid input	Input was accepted and a random location was displayed	FAIL
5	Test With Letters	User Is Logged Into A Employee Account	opuJkL	Displays a message for invalid input	As Expected	PASS
6	Test with Special Characters	User Is Logged Into A Employee Account	%$*@^#	Displays a message for invalid input	As Expected	PASS
7	Test With Invalid Code	User Is Logged Into A Employee Account	852436	Displays a message for invalid location ID	The location with the largest location ID was displayed	FAIL
8	Test With Invalid Code	User Is Logged Into A Employee Account	754862	Displays a message for invalid location ID	A random location was displayed	FAIL
9	Test With Invalid Code	User Is Logged Into A Employee Account	259876	Displays a message for invalid location ID	The first location ID in the system was displayed	FAIL
10	Test With Invalid Code	User Is Logged Into A Employee Account	999999	Displays a message for invalid location ID	As Expected	PASS

Testing Steps:

1. Select ‘View Rented Cars’ Button
2. Enter Location ID in box
3. Click ‘Search’ Button

This function test set tested the employee’s ability to view all currently rented cars at a specific location. The user inputs a six-digit code in a field that only accepts numeric characters. The system uses that code to search the locations database and display all cars linked to that location and marked as rented.

Ten tests were made, one being the control test with entirely correct input and output and the other nine tested the restrictions of the input fields. Of those nine tests, five gave unexpected results. One failed with testing for no input, one for testing with letters as the input and the last three failed when testing invalid location codes.

The test with no input displayed all cars in the system insteading of requesting input, the test for inputting letters accepted the input and displayed a random location, and the three for invalid location codes had three different results. One showed the location with the largest ID, one showed a random location and one showed the first location in the database.

System Test Sets

System Test Case: Making a renter profile.

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
1	Test with complete and valid profile information	User is on the registration page	Fill in all required fields with valid data and submit	Profile is successfully created	Profile is successfully created	PASS
2	Test with missing required fields	User is on the registration page	Leave the "First Name" field empty, fill in all other required fields with valid data, and submit	Profile creation fails with an error message, such as "First Name is a required field."	Profile creation fails with an error message, such as "First Name is a required field."	FAIL.
3	Test with invalid email format	User is on the registration page	Fill in all required fields with valid data, but use an invalid email format (e.g., "john.doe@domain"), and submit	Profile creation fails with an error message, such as "Invalid email format."	Profile creation fails with an error message, such as "Invalid email format."	FAIL
4	Test with duplicate email	User is on the registration page and a profile with the same email already exists	Fill in all required fields with valid data, but use a duplicate email and submit	Profile creation fails with an error message	Profile creation fails with an error message	PASS
5	Test with valid input and optional fields filled	User is on the registration page	Fill in all required fields with valid data, provide optional data such as "Middle Name" and "Phone Number," and submit	Profile creation is successful and all provided data is saved in the user's profile	Profile creation is successful and all provided data is saved in the user's profile	PASS
6	Test with special characters in the password	User is on the registration page	Fill in all required fields with valid data, use a password containing special characters (e.g., "P@ssw0rd!"), and submit	Profile creation is successful with the provided password	Profile creation is successful with the provided password	PASS
7	Test with a long email address	User is on the registration page	Fill in all required fields with valid data, use a long email address (e.g., "johndoe.longemailaddress@example.com"), and submit	Profile creation is successful with the provided email address	Profile creation is successful with the provided email address	PASS
8	Test with mixed case in the email address	User is on the registration page	Fill in all required fields with valid data, use a mixed-case email address (e.g., "JohnDoe@example.com"), and submit	Profile creation is successful with the provided email address	Profile creation is successful with the provided email address	PASS

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
9	Test with invalid password format	User is on the registration page	Fill in all required fields with valid data, but use an invalid password format (e.g., "password" without numbers or special characters), and submit	Profile creation fails with an error message, such as "Password must contain at least one number and one special character."	Profile creation fails with an error message, such as "Password must contain at least one number and one special character."	FAIL
10	Test with Unicode characters in the first and last name	User is on the registration page	Fill in all required fields with valid data, use Unicode characters in the first and last name (e.g., "Élise Müller"), and submit	Profile creation is successful with the provided first and last name	Profile creation is successful with the provided first and last name	PASS

Testing Steps:

1. Navigate to the profile creation page.
2. Fill in the required fields with appropriate data.
3. Optionally, fill in any additional fields.
4. Click the "Submit" button.
5. Observe the results of the profile creation process.

This set of system tests assesses the functionality of the profile creation process in a car rental application. The primary objective is to verify that users can successfully create their profiles by providing valid information in the required fields while ensuring that the system handles errors gracefully. The tests cover a variety of scenarios, including valid inputs, optional fields, special characters in passwords, long email addresses, mixed-case email addresses, and Unicode characters in the first and last names.

The tests aim to confirm that the system can process valid inputs, resulting in successful profile creation. In case of errors, the system should provide informative error messages to guide the user in correcting their input. Moreover, the system should be able to handle different types of inputs, such as special characters, Unicode characters, and varying email formats. By thoroughly examining the profile creation process through these system tests, the application's overall reliability and user experience can be ensured.

System Test Case: Renting A Car

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
1	Test With Completely Correct Input	User Is Logged Into A Renter Account	58426 Click search Select car Request rental Confirm rental	Successfully make a rental	Rental was successfully made, however car was not marked as rented	FAIL

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
2	Test With Completely Correct Input	User Is Logged Into A Renter Account	58426 Click search Select car Request rental Confirm rental	Successfully make a rental	As Expected	PASS
3	Test With A Unavailable Car	User Is Logged Into A Renter Account	58426 Click search Select car Request rental	Inform that car is already rented	User was able to go to the confirm rental screen and rent the car	FAIL
4	Test With A Unavailable Car	User Is Logged Into A Renter Account	58426 Click search Select car Request rental	Inform that car is already rented	User was correctly informed, but was still able to continue on to make the rental	FAIL
5	Test With A Unavailable Car	User Is Logged Into A Renter Account	58426 Click search Select car Request rental	Inform that car is already rented	As Expected	PASS
6	Test With The Renter Already Marked Renting	User Is Logged Into A Renter Account	58426 Click search Select car Request rental	Inform that the renter is already renting	User successfully made a rental	FAIL
7	Test With The Renter Already Marked Renting	User Is Logged Into A Renter Account	58426 Click search Select car Request rental	Inform that the renter is already renting	User made a rental, then was informed and account crashed	FAIL
8	Test With The Renter Already Marked Renting	User Is Logged Into A Renter Account	58426 Click search Select car Request rental	Inform that the renter is already renting	As Expected	PASS
9	Test With No Search Requests	User Is Logged Into A Renter Account	Click search	Request for minimum one search parameter	Search screen displayed a random location	FAIL
10	Test With No Search Requests	User Is Logged Into A Renter Account	Click search	Request for minimum one search parameter	As Expected	PASS
11	Test With Zero Results Search	User Is Logged Into A Renter Account	58474 Click search	Display zero search results screen	Search screen displayed all cars of a single model	FAIL
12	Test With Zero Results Search	User Is Logged Into A Renter Account	58474 Click search	Display zero search results screen	Search screen displayed all available cars in the system	FAIL
13	Test With Zero Results Search	User Is Logged Into A Renter Account	58474 Click search	Display zero search results screen	As Expected	PASS

Test #	Condition Being Tested	Precondition	Input	Expected Output	Actual Output	Pass/Fail
14	Test Zip Code With Special Characters	User Is Logged Into A Renter Account	%8#&* Click search	Request valid zip code	As Expected	PASS
15	Test Zip Code With Letters	User Is Logged Into A Renter Account	uyN4P Click search	Request valid zip code	As Expected	PASS
16	Test Zip Code With Unavailable Make	User Is Logged Into A Renter Account	58426 Select make Click search	Inform that that make is unavailable at that location	As Expected	PASS
17	Test Zip Code With Unavailable Model	User Is Logged Into A Renter Account	58426 Select model Click search	Inform that that model is unavailable at that location	As Expected	PASS
18	Test Zip Code With Unavailable Year	User Is Logged Into A Renter Account	58426 Enter year Click search	Inform that that year is unavailable at that location	Search screen displayed all cars of the year 2000	FAIL
19	Test Zip Code With Unavailable Year	User Is Logged Into A Renter Account	58426 Enter year Click search	Inform that that year is unavailable at that location	Search screen displayed all cars at location of the year 2000	FAIL
20	Test Zip Code With Unavailable Year	User Is Logged Into A Renter Account	58426 Enter year Click search	Inform that that year is unavailable at that location	As Expected	PASS

Testing Steps:

1. Open the website or application
2. Sign in using username and password
3. Click on the ‘Search for Rental’ button
4. Input criteria for search
5. From the displayed results, select the first one
6. Click on ‘Rent Car’ button
7. Confirm information on ‘Confirm Information’ page
8. Check that car status has changed to rented

This system test set tested the renter’s ability to make an online car rental using the search feature. The user can search by zip code, year, make or model, or any combination of the four. The zip code and year only accept numeric values and the make and model can be selected from a dropdown menu.

Twenty tests were made, two tests using completely correct input but only one passing and the other eighteen tests testing the system. Of those eighteen tests, eight gave unexpected results. Three failures were for requesting a car rental, two for an already rented car and one for a renter that was already renting.

The remaining tests failed on the search function. One tested a no input search and displayed a random location. Two tested a zero results search, one displaying all cars of a specific make and the other displaying all cars in the system. The final two tests tested searching a location for a specific year that it did not have, both displaying all cars of the year 2000, one just for that location and one for the whole system.

System Test Case: Searching for a Rental Location

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
1	Searching a Location	Zip code (92115)	Zip code is within 50 miles of a valid location	Nearby locations that rent cars	Nearby locations that rent cars	PASS
2	Searching a Location	Zip code (92115)	Zip code is within 50 miles of a valid location	Nearby locations that rent cars	No nearby locations that rent cars	FAIL
3	Searching a Location	Zip code (92115)	Zip code is within 50 miles of a valid location	Nearby locations that rent cars	Locations that rent cars but are far	FAIL
4	Searching a location	Zip code (92115)	Zip code is within 50 miles of a valid location	Nearby locations that rent cars	Wrong locations	FAIL
5	Searching a Location	Zip code (12345)	Zip code is more than 50 miles from any location	No nearby locations that rent cars, suggest others	No nearby locations that rent cars, suggest others	PASS
6	Searching a Location	Zip code (12345)	Zip code is more than 50 miles from any location	No nearby locations that rent cars, suggest others	No nearby locations that rent cars, no suggestions	FAIL
7	Searching a Location	Zip code (12345)	Zip code is more than 50 miles from any location	No nearby locations that rent cars, suggest others	Gives random locations anyway	FAIL
8	Searching a Location	none	No location is specified	Gives locations by rating	Gives locations by rating	PASS
9	Searching a Location	none	No location is specified	Gives locations by rating	Gives no locations	FAIL
10	Searching a Location	none	No location is specified	Gives locations by rating	Gives random locations	FAIL

Test #	Being Tested:	Parameters	Input (precondition)	Expected Output (post-condition)	Actual Output	Description (Pass/Fail)
11	Searching a Location	none	No location is specified	Gives locations by rating	Does nothing	FAIL

Testing Steps:

1. Open the website or application
2. Sign in using username and password
3. Click on the ‘Search for Rental’ button
4. Input zip code or choose to opt not to
    a. If a zip code is entered, the software will give a list of the closest locations
    b. If the user does not want to enter a zip code, the software will give a list of locations in order of rating.

This system test set targets the feature of searching for a location to rent a vehicle. The tests address different outcomes for when the user enters a zip code and different outcomes for when the user chooses not to enter a zip code.

Data Management Strategy

For our data management strategy, we chose to use five different databases in an SQL format to keep everything organized and easily accessible. Our original database plans only used three databases, an employee database, a vehicle database and a client database, but we realized that there were still important items not covered by these three databases.

Due to that, we added another two databases to the plan, a location database and a payment database. We considered splitting the employee database into three more, but decided that that would repeat too much information and simply chose to use another field in the database to indicate the level of the employee.

We chose to do a purely SQL database because of how our data is structured. There is very little that would require large changes, so the rigid setup of SQL is much better suited to our needs. The only one of our current databases that would likely find a non-SQL structure as effective would likely be the employee database, especially if the three subclasses of the Employee class were included in the database.

The below tables cover how our databases are structured, including descriptions of the fields and of the database itself.

Name: Vehicle Database
Type: SQL Database
Description: Contains information about vehicles

Name	carID	*Name*	make
Type	int	*Type*	string
Description	ID number of the car	*Description*	The make of the car

Name	model	*Name*	year
Type	string	*Type*	int
Description	The model of the car	*Description*	The year of the car

Name	isLuxury	*Name*	status
Type	bool	*Type*	bool
Description	Marks if the car is a luxury car	*Description*	Marks if the car is currently rented

Name	price	*Name*	locationID
Type	int	*Type*	int
Description	The base price of the car	*Description*	ID number of the location the car is held at

Name: Employee Database
Type: SQL Database
Description: Contains information about employees

Name	employeeID	*Name*	username
Type	int	*Type*	string
Description	ID number of the employee	*Description*	The username of the account

Name	password	*Name*	email
Type	string	*Type*	string
Description	The password of the account	*Description*	The email associated with the account

Name	twoFactor	*Name*	name
Type	bool	*Type*	string
Description	Indicates if the account uses two factor authorization	*Description*	The name of the employee

Name	locationID	*Name*	employeeLevel
Type	int	*Type*	string
Description	ID number of the location the employee currently works at	*Description*	Indicates what level of access the employee has in the system

Name: Client Database
Type: SQL Database
Description: Contains information about clients

Name	clientID	*Name*	username
Type	int	*Type*	string
Description	ID number of the client	*Description*	The username of the account

Name	password	*Name*	email
Type	string	*Type*	string
Description	The password of the account	*Description*	The email associated with the account

Name	twoFactor	*Name*	name
Type	bool	*Type*	string
Description	Indicates if the account uses two factor authorization	*Description*	The name of the client

Name	address	*Name*	birthdate
Type	string	*Type*	string
Description	The provided client address	*Description*	The birthdate of the client

Name	phoneNumber	*Name*	zipcode
Type	int	*Type*	int
Description	The provided phone number of the client	*Description*	The current zip code of the client

Name	licenseNumber	*Name*	paymentID
Type	string	*Type*	int
Description	The license number of the client	*Description*	ID number of the client's payment method

Name: Location Database
Type: SQL Database
Description: Contains information about locations

Name	locationId	*Name*	name
Type	int	*Type*	string
Description	ID number of the location	*Description*	The name of the location

Name	maxCapacity	*Name*	carAmount
Type	int	*Type*	int
Description	The maximum amount of cars that can be held at the location	*Description*	How many cars are currently held at the location

Name: Payment Database
Type: SQL Database
Description: Contains information about client payment information

Name	paymentID	*Name*	cardType
Type	int	*Type*	string
Description	ID number of payment	*Description*	The type of card
Name	cardNum	*Name*	cvv
Type	int	*Type*	int
Description	The card number	*Description*	The CVV of the card
Name	billingZip	*Name*	billingAddress
Type	string	*Type*	string
Description	The provided billing zip code	*Description*	The provided billing address
Name	month	*Name*	day
Type	int	*Type*	int
Description	Expiration month	*Description*	Expiration day