Car Rental System Design Specification

Prepared by: Andressa Wu (824360167), Sadia Abdirahim (826998101), Trevor Thayer (826194012)

1. **SYSTEM DESCRIPTION:** The purpose of this document is to specify the requirements for a simple software application for a Rental Car System. This system is to be used by both employees and customers to keep inventory, pricing, and payments up to date.

    1.1. **Customer/Employee Management:** The system should allow all users (both employees and customers) to create and manage personal accounts. For employees, it should consider different information authorization levels based on employee level. Customers should be able to view their rental history and update personal/payment information.

    1.2. **Car Management:** The system should allow employees with proper authorization to add and remove car models. Each model should have specifications of year, make, model, color, and gas mileage. Customers should be able to search for cars based on filters relevant to car specifications.

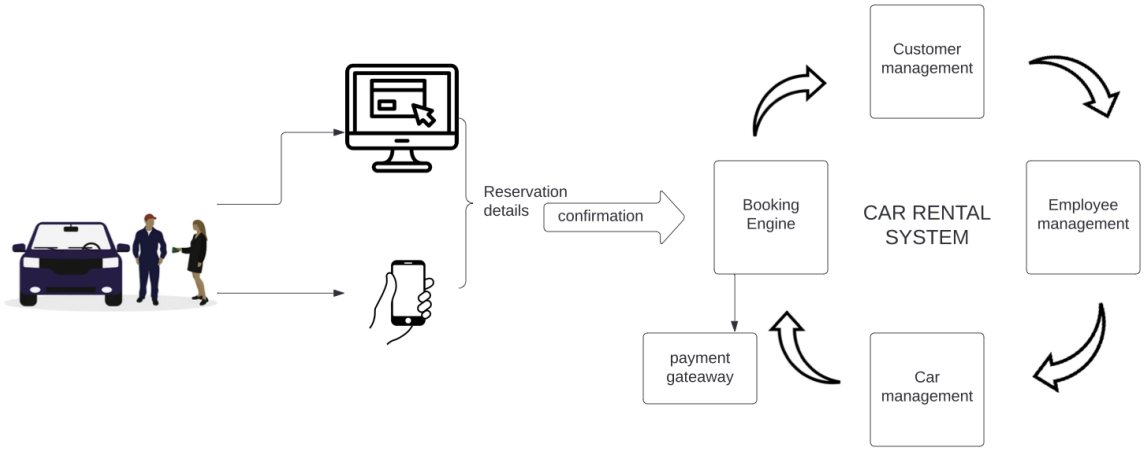
    1.3. **Booking Management:** The system should allow for reservations to be made in advance with specific dates, as well as in person. It should compare these dates with availability for the specific vehicle requested.

    1.4. **Rental Management:** Each employee should be able to check out vehicles to customers. They should also be able to take vehicles out of circulation based on damages. During each check in and check out, the system should require an update of date and time, gas level and any damages. The system should then calculate any additional fees based on the time the car was utilized and any damages.

    1.5. **Payment Management:** The system should allow for payments in advance, as well as in person payments. Cash and card should be options for in person payments. Processing refunds should be handled as well. Each payment should be added to the account of the customer, as well as a record of all payments received.

2. SOFTWARE ARCHITECTURE OVERVIEW

2.1. Architectural Diagram of All Major Components:

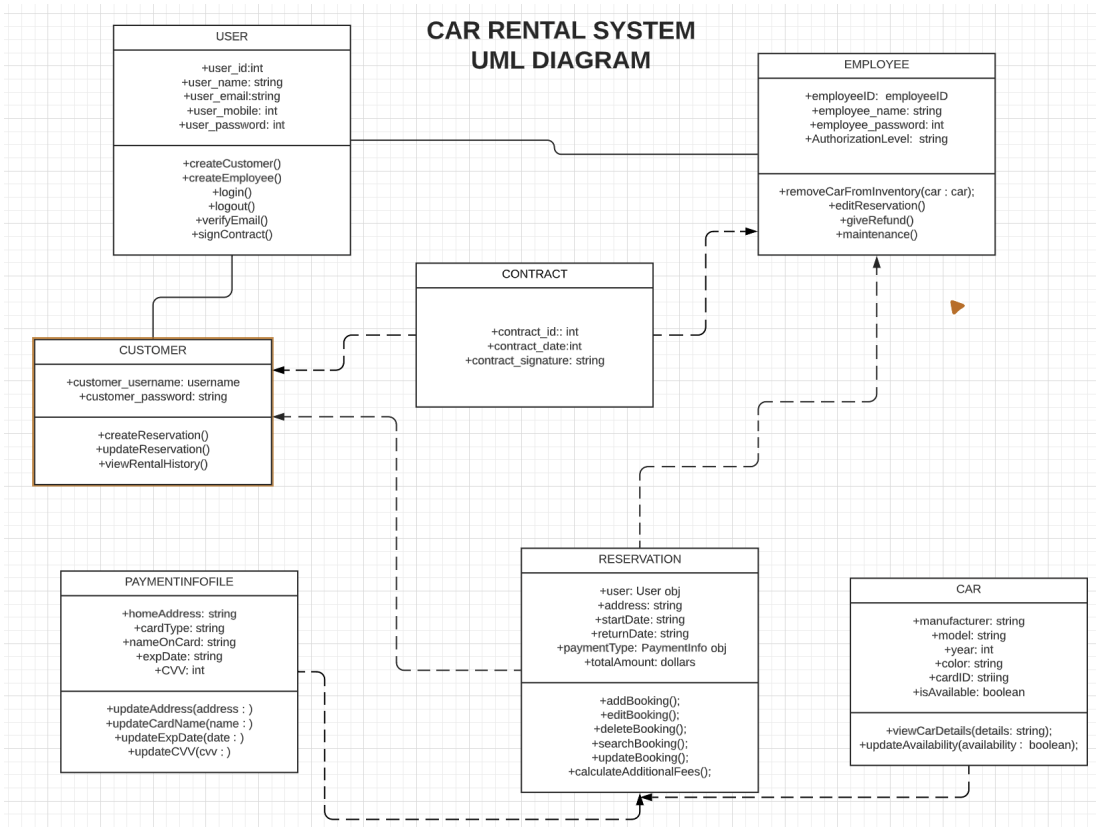


2.2. Description of Architecture Diagram:

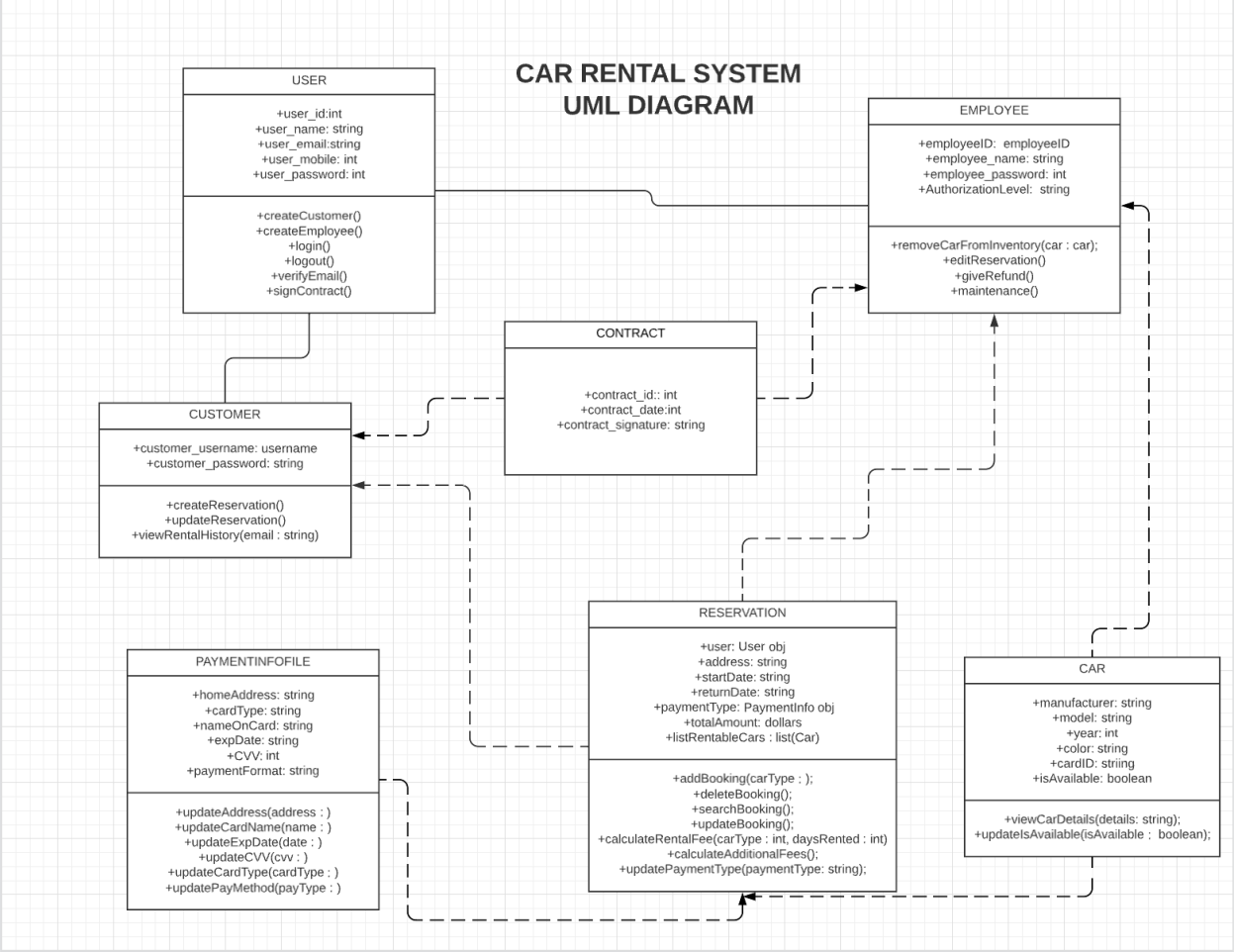
The diagram above represents the architecture diagram of a car rental system. The system is to be used by the customers and the employees of the company. In order to access the system, they can either use it through a mobile application or the website version. The reservation details are linked to the booking engine which is at the forefront of the system. Through the booking engine, customers and employees can access the cars, the costs, availability, maintenance, payments and among other things.

2.3. UML Class Diagram:

OLD:



NEW:



2.4. Description of Classes/Operations/Attributes:

2.4.1. USER

The User class consists of both the Employee and Customer class. Its attributes contain a user ID, a username, an email, their mobile number, and their password. These attributes specify information about the customer and/or the employee. Operations representing the User class include createCustomer, createEmployee, login, logout, verifyEmail, and signContract.

2.4.2. CUSTOMER

The Customer class is associated with the User class because the User can either be a customer or an employee. The attributes of this class are customer_username that returns a username type and customer_password that return a string type. These two attributes let the Customer log in to the website or the app. The operations of this class are createReservation,

updateReservation and viewRentalHistory. These operations allow the user to modify the reservation and also look for rental history.

2.4.3. EMPLOYEE

The Employee class is included in this system as another form of the User class - to be implemented separately from the Customer class. This is important because they need employeeID's and authorizationLevels for the system to determine what actions they can do. Employees should be able to request maintenance or remove cars from the system in case of damages or product recalls. They should also be able to edit reservations and give refunds upon request by customers.

2.4.4. CONTRACT

The Contract class is simply utilized as an agreement between the rental company and the customer to ensure liability is properly handled. This requires an identification number for the contract, as well as a date, and a signature from both parties. The User class ensures that customers and employees both have the ability to sign the contract.

2.4.5. RESERVATION

The Reservation class enables the process of booking an appointment for the vehicle. This class is utilized by both the employee and customer, and depends on the PaymentInfoFile class. This ensures that the payment information is provided prior to booking a reservation. The attributes of the reservation class include an object of the user class, the address of either the customer or the employee, the startDate of when the vehicle will be purchased and its counterpart, the returnDate of the car rental. Other attributes entail a paymentType (object of the PaymentInfoFile class) and the totalAmount paid in dollars. The operations of this class include addBooking, editBooking, deleteBooking, searchBooking, updateBooking, and calculateAdditionalFees.

2.4.6. PAYMENTINFOFILE

The PaymentInfoFile class signifies the information provided by the card holder. Its attributes include the home address, type of card, the name provided on the card, its expiration date, and the CVV (3 digit code on the back). The operations of this class allows the user to update four entities which include their address, card name, expiration date, and their CVV.

2.4.7. CAR

The Car class depends on the Reservation class which also depends on the Employee class and the Customer class. In order to access the Car Class the customer must first make a reservation so they can choose the car and the employee can look at the reservation and the car details. The attributes of this class are manufacture, model, year, color, carID, isAvailable. The

return types are string, int and boolean. These attributes give the customer/employee some information about the car. The operations are viewCarDetails and updateAvailability. These operations are behavioral features that allow the User to view or update some information about the car.

2.5. EXPLANATION OF THE DIAGRAM CHANGE

Some changes were made to the new UML Diagram. For the PaymentProfile class, new operations were added such as updateCardType and updatePayMethod. These operations are necessary so the user can have the option to change the card type or the payment method. In the Reservation class, calculateRentalFee and updatePaymentType operations were also added. The calculateRentalFee operations let the user see the total price of their rental and the updatePaymentType allows the user to change their payment type in case they need it.

3. DEVELOPMENT PLAN AND TIMELINE

3.1. Partitioning of Tasks/Team Member Responsibilities:

- **3.1.1. Trevor -** System Description, Description of Classes, UML diagram.
- **3.1.2. Andressa -** Architectural Diagram, Description of Car and Customer class and UML Diagram.
- **3.1.3. Sadia -** Description of User Class, Reservation class, PaymentInfoFile class, and UML diagram

VERIFICATION TEST PLAN

Unit - specific to that class
Integration - uses multiple classes
System - must interact with the user (use case diagram)

UNIT TESTS

1. **PAYMENTINFOFILE Feature: updateAddress(String : address)**
   - 1.1. **Test 1:** updateAddress("806 Main Lane, Ponte Vedra Beach, FL 32082");
     - 1.1.1. **Returned:** "Address has been successfully updated."
     - 1.1.2. **Explanation:** This test tracks the method's ability to be able to handle a full address. The reason that the address was successfully updated is because it contains all parts of a valid address: street, city, state, and zip code. Because it was passed a valid address, the method then returns a message stating that the address was updated.
   - 1.2. **Test 2:** updateAddress("806 Main Lane, Ponte Vedra Beach");
     - 1.2.1. **Returned:** "Error updating billing address: incomplete or invalid address given."
     - 1.2.2. **Explanation:** This test tracks the method's ability to be able to handle an incomplete address. The reason that this address returned an error message is because it is missing some parts of a valid address. The inputted address is missing the state and zip code. In order to ensure that the address is correct, the method will not update the address unless it is given a valid one.
   - 1.3. **Test 3:** updateAddress("");
     - 1.3.1. **Returned:** "Please enter an address."
     - 1.3.2. **Explanation:** This test tracks the method's ability to be able to handle an empty address. In this case, there was no inputted address, so the method simply asks for an address.

2. **CAR Feature: updateIsAvailable(boolean : isAvailable)**
   - 2.1. **Test 1:** updateIsAvailable(true);
     - 2.1.1. **Returned:** "Car is now available for rent."
     - 2.1.2. **Explanation:** This test tracks the method's ability to be able to handle a valid 'true' boolean. Since this is a valid parameter, the method will then update the isAvailable member variable in the Car Class to true. This means that it is now possible to rent the Car to a customer. Hence, the message returned states that the Car can now be rented.
   - 2.2. **Test 2:** updateIsAvailable(false);
     - 2.2.1. **Returned:** "Car is now unavailable for rent."

2.2.2. **Explanation:** This test tracks the method's ability to be able to handle a valid 'false' boolean. Since this is a valid parameter, the method will then update the isAvailable member variable in the Car Class to false. This means that it is now not possible to rent the Car to a customer. Hence, the message returned states that the Car can now be rented.

2.3. **Test 3:** updateIsAvailable("true");

2.3.1. **Returned:** "Please enter a valid condition."

2.3.2. **Explanation:** This test tracks the method's ability to handle the case where an invalid condition is given as a parameter. Because the given parameter is not a boolean, the method returns a message asking for a valid condition and will not update the availability of the Car.

INTEGRATION TESTS

3. RESERVATION Feature: updatePaymentType(String : paymentType)

3.1. **Test 1:** updatePaymentType("cash");

3.1.1. **Returned:** "Payment type is now cash. To book your car, you must pay on site."

3.1.2. **Explanation:** This test ensures that the method is able to handle updating the payment method. For each Reservation class, there is also a PaymentInfoFile. Updating the payment method through the Reservation class requires the PaymentInfoFile to be updated as well. This test ensures that the Reservation class correctly accesses and updates the 'paymentFormat' member variable in the PaymentInfoFile. Cash is a valid form of payment, hence the payment type is updated and the customer is reminded that they must pay in person.

3.2. **Test 2:** updatePaymentType("check");

3.2.1. **Returned:** "Payment type is now checked. To book your car, you must pay on site."

3.2.2. **Explanation:** This test also ensures that the method is able to handle updating the payment method. For each Reservation class, there is also a PaymentInfoFile. Updating the payment method through the Reservation class requires the PaymentInfoFile to be updated as well. This test ensures that the Reservation class correctly accesses and updates the 'paymentFormat' member variable in the PaymentInfoFile. Check is a valid form of payment, hence the payment type is updated and the customer is reminded that they must pay in person.

3.3. **Test 3:** updatePaymentType("card");

3.3.1. **Returned:** "Payment type is now card. To book your car, pay either online or on site. If you're paying online, please enter your card information below."

3.3.2. **Explanation:** This test also ensures that the method is able to handle updating the payment method. This test ensures that the Reservation class correctly accesses and updates the 'paymentFormat' member variable in the PaymentInfoFile. Card is a valid form of payment, hence the payment type is updated and the customer is reminded that they are able to pay either online or in person. Since it is a card payment, the PaymentInfoFile must now be able to store the card number, expiration date, and CVV. This is demonstrated in the returned message since it asks the user for the card information.

3.4. **Test 4:** updatePaymentType("bitcoin");

3.4.1. **Returned:** "I'm sorry, but we do not accept that form of payment. Please enter a valid payment type."

3.4.2. **Explanation:** This test ensures that the method can properly handle a payment type that is invalid. This car rental system is unable to use bitcoin, so the system does not update the payment method. Instead, it simply tells the user that the entered payment type is invalid and asks for a valid type.

4.

RESERVATION Feature: addBooking(Car: carType)

4.1. **Test 1:** Car BMW = new BMW ("BMW", "2023 BMW 3 Series", 2023, "Silver", True)

4.1.1. **Returned:** "We have received your request to book a silver 2023 BMW 3 Series. This car is currently in stock and is available for booking during your requested time period."

4.1.2. **Explanation:** This test utilizes both the Reservation and Car Class in order to book a car from the system's inventory. The addBooking (carType:) feature accesses information from the Car class members in order to expedite the process of booking a car. It employs members of the Car Class including the manufacturer, the car model, the year, its color, and returns the boolean value "true" since the requested car is currently available.

4.2. **Test 2:** Car Honda = new Honda ("Honda", "2023 Honda Accord", 2023, "Red", False)

4.2.1. **Returned:** "We have received your request to book a Red 2023 Honda Accord. Unfortunately, this car is currently unavailable for booking and we can not move forward with your request."

4.2.2. **Explanation:** This test utilizes both the Reservation and Car Class in order to book a car from the system's inventory. The addBooking (carType:) feature accesses information from the Car class members in

order to expedite the process of booking a car. It employs members of the Car Class including the manufacturer, the car model, the year, its color, and returns the boolean value "false" since the requested car is currently not in stock.

4.3. **Test 3:** Car Toyota = new Car (2023 Toyota Camry)

4.3.1. **Returned:** "We're sorry, the requested car doesn't exist in our inventory"

4.3.2. **Explanation:** This test ensures that a correct output would be showcased when the user input is invalid and the car requested does not exist in the system's inventory.

SYSTEM TESTS

5. RESERVATION Feature: calculateRentalFee(Car : carType, int : daysRequested)

5.1. **Test 1:** Car jeep = new Car("Jeep", "Wrangler", 2010, "Black");
calculateRentalFee(jeep, 2);

5.1.1. **Returned:** "The requested Jeep Wrangler costs $89 per day, and a total of $178 for 2 days. It is available for rent during your requested time period.

5.1.2. **Explanation:** This feature calculates the rental fees based on the car that the user enters and the number of rental days requested. In this case, the user entered a Car object called Jeep and requested 2 days for rent. The Reservation class accesses the startDate for the Reservation and the isAvailable member variable of the Jeep object to check if it is available for rent for both days. In this case, the Jeep is available for the entire time. Hence, the system returns a message to the user that the Jeep costs $89 a day and that it is available for rent.

5.2. **Test 2:** Car mustang = new Car("Ford", "Mustang", 2020, "Red");
calculateRentalFee(mustang, 3);

5.2.1. **Returned:** "We're sorry, but the requested car is unavailable."

5.2.2. **Explanation:** This feature calculates the rental fees based on the car that the user enters and the number of rental days requested. In this case, the user entered a Car object called Mustang and requested 3 days for rent. The Reservation class accesses the startDate for the Reservation and the isAvailable member variable of the Mustang object to check if it is available for rent for all 3 days. In this case, the Mustang is not available for the entire time. Hence, the system returns a message to the user that the car they requested to see the price on is not available for rent.

5.3. **Test 3:** Car lamborghini = new Car("Lamborghini", "Huracan", 2018, "Red");
calculateRentalFee(lamborghini, 1);

5.3.1. **Returned:** "We're sorry, but the requested car does not exist."

5.3.2. **Explanation:** This test ensures that the feature can handle when the user enters cars that do not exist in the system. In this case, the method checks

if the entered car exists in the Reservation's listRentableCars list. If it does not, then the car does not exist in the rental system. Because the lamborghini is not in the list, the system returns a message telling the user that it does not exist.

6. CUSTOMER Feature: viewRentalHistory(email : String)

6.1. **Test 1:** viewRentalHistory("alexSmith@gmail.com");

6.1.1. **Returned:** "You have no past rental history on this email."

6.1.2. **Explanation:** This feature is to provide the customer their rental history. The customer class is associated with the User class and the reservation class depends on the Customer class in order to exist. In this case, the user enters the email string "alexSmith@gmail.com", which is the account created by the customer, to see if he has a rental history. The customer class accesses the viewRentalHistory operation to check if he does but returns a message saying that the customer has no rental history registered in this email.

6.2. **Test 2:** viewRentalHistory("chefTom@aol.com");

6.2.1. **Returned:** "You have 2 past rentals under this email:

1. 6/10/2021: Jeep Wrangler, $89 a day
2. 4/11/2022: Honda Odyssey, $96 a day

6.2.2. **Explanation:** This feature is to provide the customer their rental history. The customer class is associated with the User class and the reservation class depends on the Customer class in order to exist. In this case, the user enters the email string "chefTom@aol.com" which is the account created by the customer. The customer class accesses the viewRentalHistory to see if there is a rental history. It then returns a message to the customer saying that he does have 2 past rentals under his email. Hence, the operation successfully works.

6.3. **Test 3:** viewRentalHistory("trevorThayer@gmail.com");

6.3.1. **Returned:** "Please enter a valid email address."

6.3.2. **Explanation:** This test ensures that the feature can allocate a valid email address in their system. In order to do that, the Customer class has to access the User class because it is where the customer or the employee creates their account. Because there was no account created under this email the system returns a message telling the user to put a valid email address.