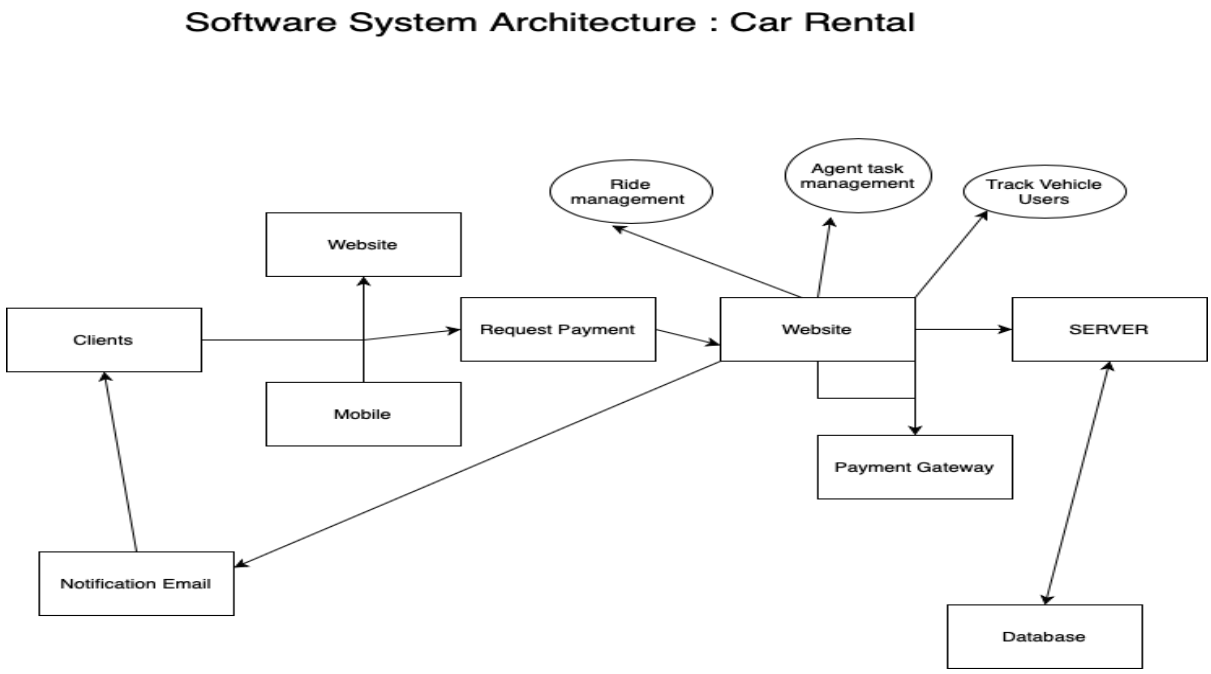
Software Design Specification

Software Title: CarRentalPro

Team Members:
1. Jesus Serna
2. Matthew Sprague

System Description:
Brief Overview of System: CarRentalPro is a comprehensive car rental management system designed to streamline the process of renting vehicles. It provides an intuitive user interface for customers to book and manage reservations while offering robust backend functionality for administrators.

Software Architecture Overview:
Architectural Diagram:

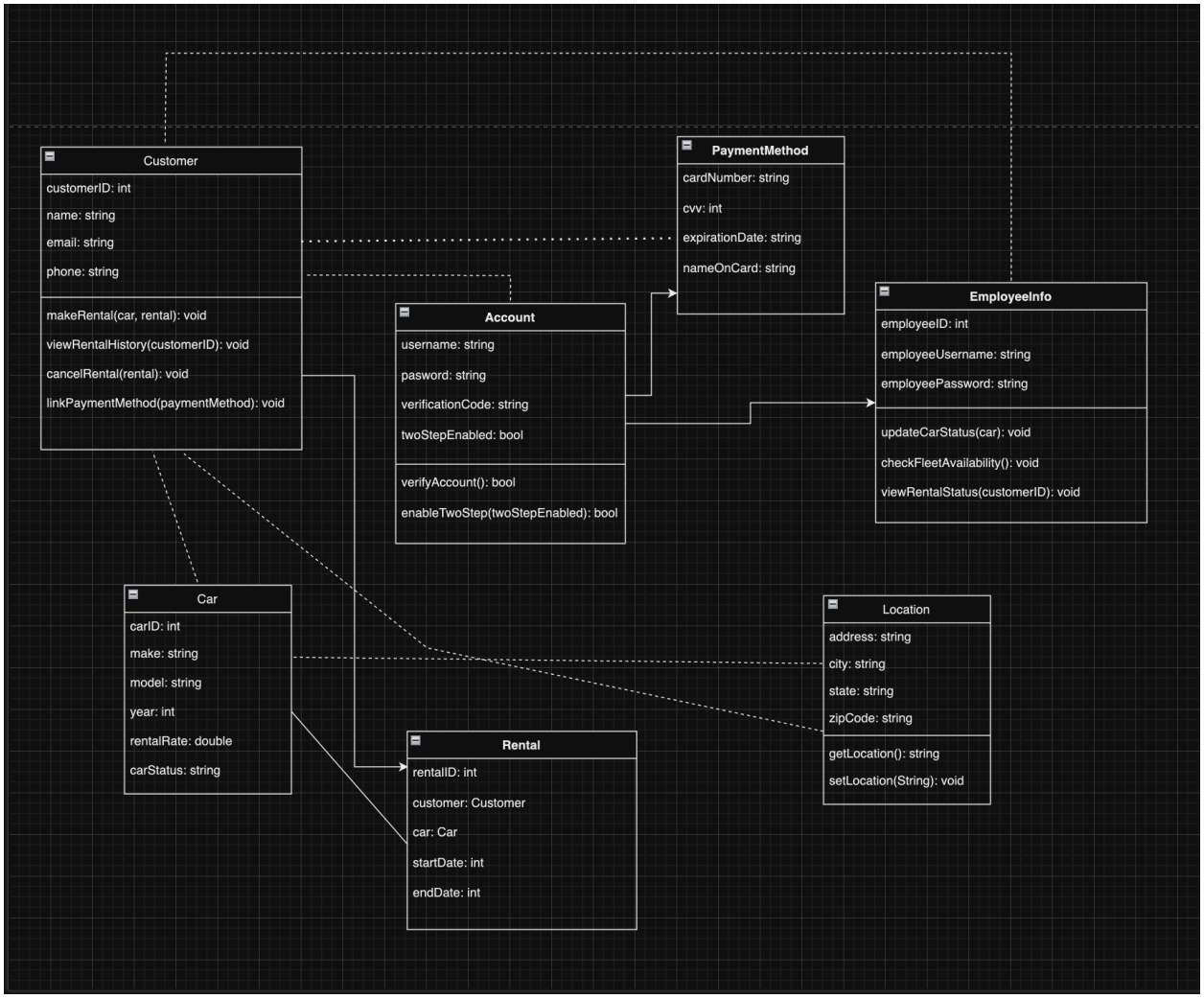


Software System Overview:

1) Mobile and Web Users: represent the interaction through the browser or mobile app which includes registration, vehicle booking and management.
2) Server: The functionality is to act as the central processing unit which handles business communication with a database
3) Database: The functionality is to store and manage data that is relevant related to users.

4) The Website system communicates via accessing specific URL and communicates with the server request, it utilizes HTTP/HTTPS protocols for secure data transfer and uses APIS for data exchange. To store data SQL is use so that it becomes efficient.

UML Class Diagram:



Description of Classes:

Customer: represents a customer of the car rental company
Attributes:

- customerID: int - unique ID the customer receives when account is created so info can be looked up
- name: string - the customer's name
- email: string - the customer's email address
- phone: string - the customer's phone number

Operations:

- makeRental(car, rental): void - initiates a rental for the customer by utilizing an object from the car class and an object of the rental class as parameters
- viewRentalHistory(customerID): void - using the customer's unique ID, past rentals by that customer are displayed
- cancelRental(rentalID): void - cancels a current rental by using the unique rental ID
- linkPaymentMethod(paymentMethod): void - links a payment method with the customer's account

Car: represents a car in the fleet

Attributes:

- carID: int - the unique ID number for the car so info can be looked up
- make: string - the brand/manufacturer of the car
- model: string - the model of the car
- year: string - the year the car was made
- rentalRate: double - the cost of renting the car per day
- carStatus: string - whether the car is available for rent, currently being rented, or under maintenance

Account: represents the account that is created for the customer

Attributes:

- username: string - the customer's created username
- password: string - the customer's created password
- verificationCode: string - a code sent to the customer when creating an account so the customer can verify their email address
- twoStepEnabled: bool - indicates if customer has enabled two step authentication for when they sign in to their account

Operations:

- verifyAccount(): bool - checks if the customer's account is verified
- enableTwoStep(twoStepEnabled): void - enables two step authentication if the customer indicated they wanted it enabled

Rental: represents the rental info for the car the customer rented

Attributes:

- rentalID: int - the unique ID number for the customer's rental so info can be looked up
- customer: Customer - uses an object of the customer class to indicate who is renting the car

- car: Car - uses an object of the Car class to indicate what car is being rented
- startDate: int - the start date of the rental period
- endDate: int - when the car must be returned

PaymentMethod: represents the customer's payment method for making rentals
Attributes:
- cardNumber: string - the customer's credit card number
- cvv: int - the card verification value of the customer's card
- expirationDate: int - the expiration date of the customer's card
- nameOnCard: string - the name on the card

Location: represents the physical location of where the car can be picked up for rental
Attributes:
- address: string - the address of where the car is located
- city: string - the city of the where the car is located
- state: string - the state of where the car is located
- zipCode: string - the zip code of where the car is located

Operations:
- getLocation(): string - returns the location of the car
- setLocation(String) - identifies the location of the car

EmployeeInfo: represents the employee account so they can access the system to view information or make changes
Attributes:
- employeeID: int - the ID number of the employee
- employeeUsername: string - unique employee username for their account
- employeePassword: string - unique employee password for their account

Operations:
- updateCarStatus(Car): void - takes an object of the car class so the status of the car can be updated to show if it is being rented, available, or under maintenance
- checkFleetAvaibility(): void - shows the cars available for rent
- viewRentalStatus(customerID): void - indicates the status of a customer's rental

Development plan and timeline

- Partitioning of tasks:
    - Define Project roles and responsibilities
    - Gather requirements by conducting interviews and documentation
    - System Design finalization including Software architecture, UML diagrams and database schema

- Development by implementing the logic, setting up the database and developing APIs, for frontend implement customer and admin interfaces.
- Test by doing unit testing, integration testing, UAT and fix bugs for optimization and document.
- Finalize product and deploy.

- Team member responsibilities: Team members will share all software system duties equality Jesus initiated the draw.io diagram and once it was initialized both members added the corresponding methods, functions and declarations that were previously discuss during class. each team member work on one test case set each Jesus worked on test case number 1 and Matthew worked on test case 2

TEST CASE SET 1: LOCATION

Unit Test:
Test if the system correctly identifies the current location of a rental car using GPS.
Purpose: Ensure accurate determination of a rental car's current location using GPS.

```
// Unit Test
// Test Data
RentalCar car = new RentalCar();
Location currentLocation = car.getCurrentLocation();

// Execution
// Simulate GPS signal and update car's current location

// Expected Output
if currentLocation matches expectedLocation:
   Unit test passes
else:
   Unit test fails
```

Detailed Steps:
- Initialize a rental car object.
- Retrieve the current location of the car using GPS.
- Verify if the obtained location matches the expected location.

The unit test evaluates the system's ability to correctly identify the current location of a rental car through GPS. It begins by initializing a rental car object and retrieving its current location using GPS. Simulated GPS signals are utilized to update the car's current location. The expected output is compared with the obtained location to determine if the unit test passes or fails. If the obtained location matches the expected location, the test is successful; otherwise, it fails

Integration Tests:

Test if the rental car's location is correctly displayed on a map in the customer app.
Purpose: Validate accurate display of a rental car's location on a map in the customer app.

```
// Integration Test
// Test Data
RentalCar car = new RentalCar();
Location currentLocation = car.getCurrentLocation();
CustomerApp app = new CustomerApp();

// Execution
app.displayCarLocation(car);

// Expected Output
if car's location is correctly displayed on the map:
   Integration test passes
else:
   Integration test fails
```

Detailed Steps:
- Instantiate a rental car object and retrieve its current location.
- Open the customer app and display the location of the rental car on the map.
- Verify if the displayed location matches the actual location of the rental car.

System Tests:

Test if the system correctly handles the situation when multiple rental cars are in the same location.
Purpose: Ensure proper management of multiple rental cars present at the same location.

```
// System Test
// Test Data
List<RentalCar> cars = new ArrayList<>();

// Execution
// Simulate multiple cars at the same location

// Expected Output
if system handles multiple cars at the same location correctly:
```

System test passes
else:
System test fails

Detailed Steps:
- Simulate a scenario where multiple rental cars are at the same location.
- Verify if the system accurately tracks and manages the presence of multiple cars at the same location.

The system test aims to ensure the accurate tracking and management of multiple rental cars concurrently present at the same location within the system. By simulating the presence of multiple rental cars at identical coordinates, the system's capability to distinguish, update status, and provide accurate reporting on each car's presence is evaluated. The test's success hinges on the system's ability to effectively manage this scenario, accurately reporting the number of rental cars at the location, and appropriately handling the situation.

Test Plan 2: Employee

Unit Test

This unit test is designed to test the functionality associated with the EmployeeInfo class, as well as its ability to create information that is associated with it. The functionality that is being tested is the method that allows the employee to check the fleet of cars at the physical location that they work for to check the availability of cars for rent. An employee should be shown a list of cars available or unavailable for rent if this method executes properly.

Testing: EmployeeInfo checkFleetAvailability()

Employee Steve

Steve.employeeID = 860173

Steve.employeeUsername = “cool_guy_steve”

Steve.employeePassword = “steveisnotcool**”

if (checkFleetAvailability() == list of cars that are available for rent)

PASS

else

FAIL

Integration Test

This integration test will again test the functionality of the EmployeeInfo class but integrating the car interface to test if an employee can correctly update the status of a car that is in the fleet. The updateCarStatus() method of the EmployeeInfo class is being utilized, as it takes in a newly created object of the car class in order to have the information present to be updated. A car that is currently undergoing maintenance should be shown as now available for rent if this method executes properly.

Testing: EmployeeInfo updateCarStatus(Car) = string

Employee Steve

Steve.employeeID = 860173

Steve.employeeUsername = “cool_guy_steve”

Steve.employeePassword = “steveisnotcool**”

Car myCar

myCar.CarID = 63917

myCar.make = “BMW”

myCar.model = “335i”

myCar.year = 2012

myCar.rentalRate = 104.75

myCar.carStatus = “under maintenance”

If (Steve.updateCarStatus(myCar) == “available”)

PASS

Else

FAIL

System Test

Testing the system's ability to handle all employee functions as well as account functionality as employees log in and out of the system to view info about cars.

On a computer at a physical location of the rental dealership, an employee should be able to select an employee login option. They can enter their unique employee ID, username, and password when prompted to log into the system. Employees should not be prompted for two step authentication for that is not required with employees. While logged into the system, an employee will have multiple icons that they can click on. One of these icons indicates the fleet of cars that is present at the location. If they click on that icon, a detailed list of the cars in the fleet are available, including photos of the cars with their ID, Make, Model, Year, Rental Rate, and the status of the car, whether it is available, being rented currently, or if it's undergoing maintenance. They can click on a specific car, and there will be an edit section where they can make necessary changes including updating the status of the car. When a customer creates an account they are given a unique customer ID number, which an employee can enter into the system to look up the status of a customer's rental. Employees can log out of the system at any time.