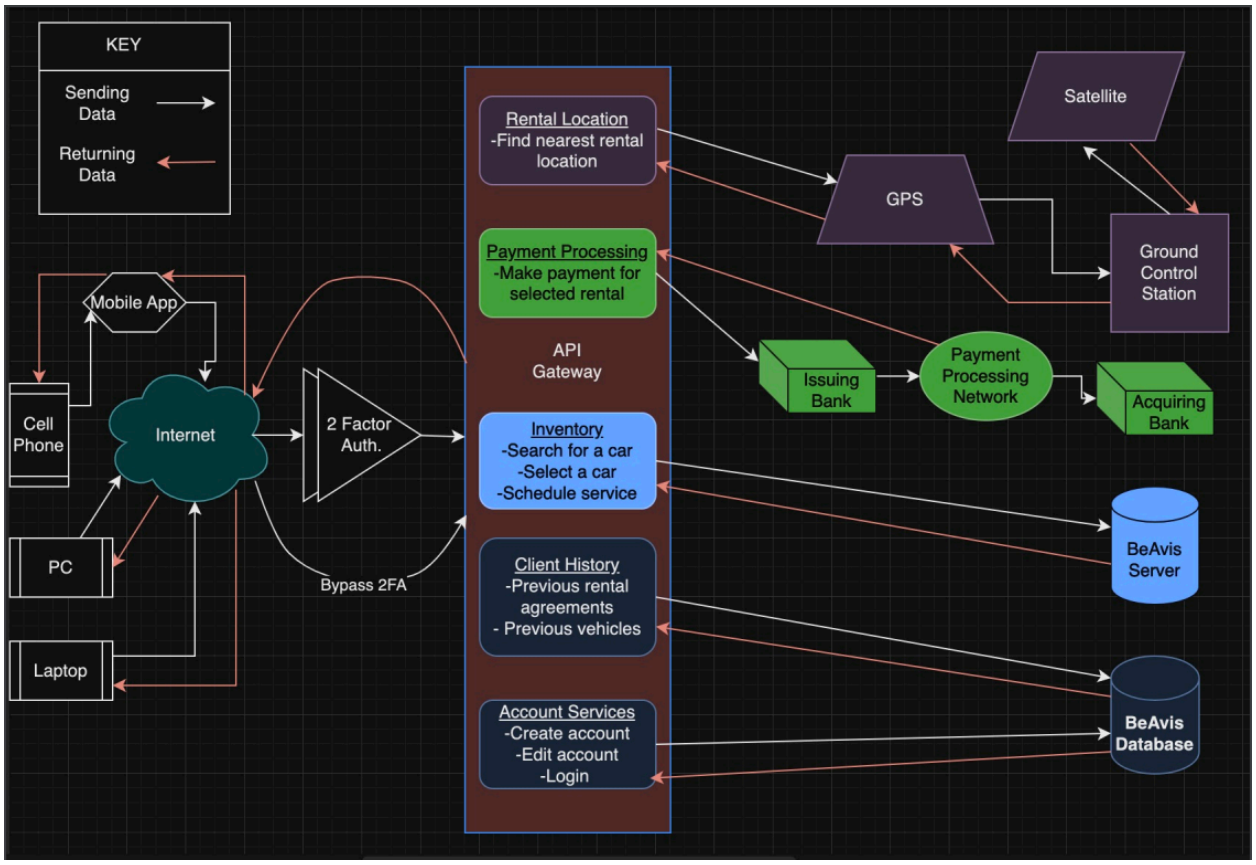
BeAvis Car Rental Software Design Specification

Prepared by: Adriel Guerrero, Owen Morales, Patrick Dowell

System Overview:

This overview explores the BeAvis Car Rental System, a contactless car rental agency designed to be completely operable via the internet. This system contains features that will grant clients, with a valid account, to perform and/or allow business transactions, GPS location, rental history access, contract e-signatures, and explore rental options. On the other end, BeAvis employees with valid user accounts will be able to access client account information, vehicle inventory and schedule routine maintenance for vehicles available.

Architectural Diagram:



Architectural Overview:

The software architecture diagram for BeAvis' Car Rental starts on the left-hand side with the user devices (client or employee), which may be a cellular device, laptop, or a personal computer. Computer devices connect directly to the internet via preferred browser, while cell phones connect via an installed BeAvis application. Users have the option to utilize two-factor authentication or to connect directly to their account bypassing 2FA. Once the user has gained access to the BeAvis system API Gateway, the system can connect to different features (GPS Service, Payment Processing, Inventory, Client History, and Account Services) and then return data to the user.

The GPS feature connects to GPS, which then communicates with the Ground Control Station, which has control over satellites, and then returns the data from the satellite back to the BeAvis system, and then back to the client.

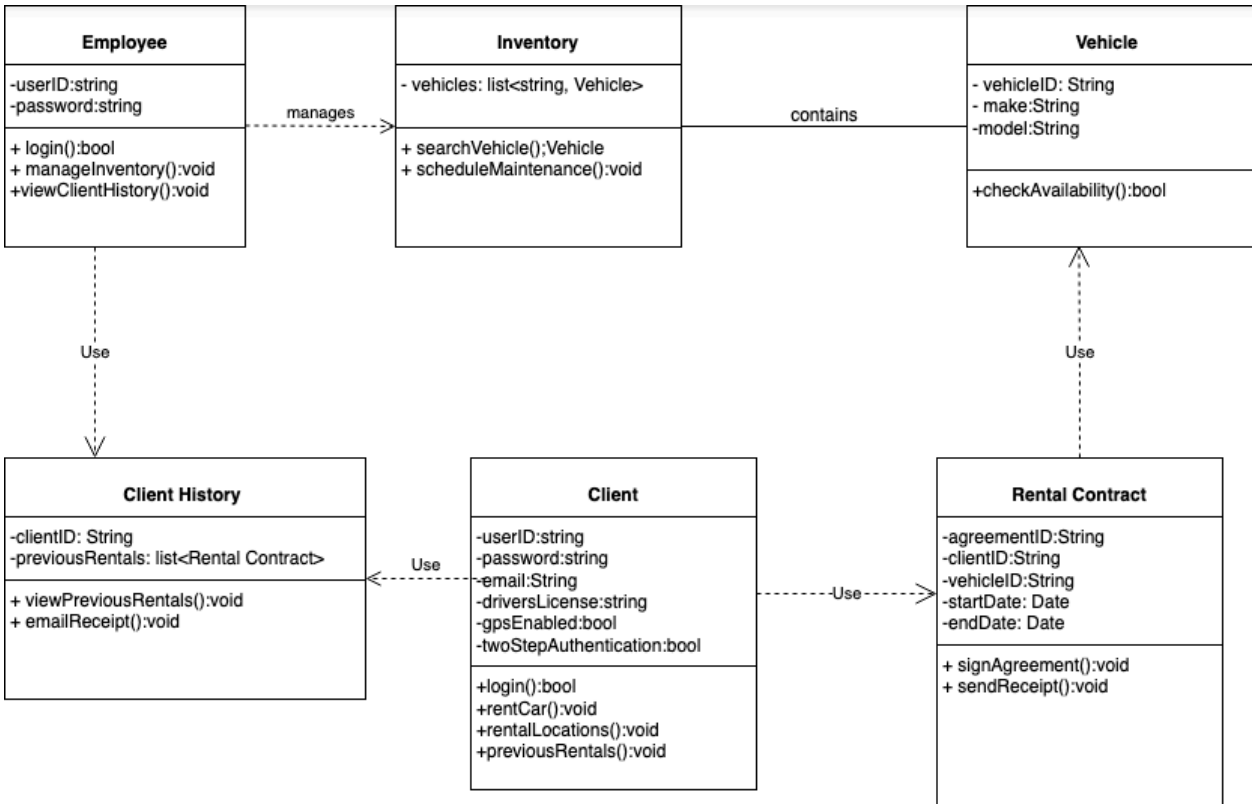
The Payment Processing feature initializes payment by connecting to the user's bank, submitting the payment from the user's bank to a Payment Processing Network, which in turn distributes the payment to BeAvis' bank as well as submitting proof of payment back to the BeAvis system.

The Inventory feature connects directly to the BeAvis Server which contains all of the inventoried vehicles, allowing users to browse and select available vehicles or employees to view inventory and schedule maintenance on vehicles.

The Client History feature connects directly to the BeAvis Database which contains all of the user's previous rental contracts and which vehicles were rented out, allowing both user and employees to navigate prior agreements.

The Account Services feature connects directly to the BeAvis Database which contains all of the user and employee account information and log-in credentials. This feature allows users or employees to make an account, view their account, or make changes to their account, such as enabling GPS or 2 Factor-Authentication.

UML Car Rental Diagram:



Classes**:**

Employee – The Employee class is responsible for handling tasks that involve viewing customer purchase history and the ability to manage vehicle inventory.

Attributes -

- userID:string - Requires the employees unique userID.

- Password:string - Requires a password to access employee account.

Operations -

- login(): bool - Allows access to the account if employees credentials are valid, if credentials are invalid, account access will be denied.
- manageInventory():void - Allows employees to navigate vehicle inventory.
- viewClientHistory():void - Allows employees to search client rental purchase history.

Inventory - Contains vehicles that are available for customers to rent.

Attributes -

- vehicles: list<string, Vehicle> - Contains a list of vehicles.

Operations -

- searchVehicle(): Vehicle - Allows employee to search vehicle.
- ScheduleMaintenence():void - Lets employee schedule maintenance for vehicle.

Vehicle - Contains identifying info on vehicles in the system.

Attributes -

- vechicleID: String - Contains info on the vehicle's registered ID.
- make: String - Contains info on the make of a vehicle.
- model:String - Contains info on the model of a vehicle.

Operations -

- checkAvailablility(): bool - Allows employees to see if the vehicle is available.

Rental Contract - Contains information on rental contracts from clients.

Attributes -

- agreementID: String - The ID associated with the rental contract.
- clientID: String - The ID of the client who signed the contract..
- vehicleID: String - The vehicle string associated with the contract
- startDate: Date - Information on when the contract starts.
- endDate: Date - Information on when the contract ends.

Operations -

- signAgreement(): void - Allows the signage of a contract.
- sendReceipt(): void - Allows the receipt to be sent to the client after signage.

Client - Contains all information from each client.

Attributes -

- userID: String - The client's user ID for their account.
- password: String - The clients's password for their account.
- email: String - The client's email for their account
- driversLicense: String - The client's driver's license ID.
- gpsEnabled: bool - Shows if the client as gps enabled.
- twoStepAuthentication: bool - Checks if the account has properly authenticated.

Operations -

- login(): bool - Allows the Client to log in.
- rentCar(): void - Allows the client to rent a car.
- rentalLocations(): void - Allows the client to find rental locations in their area.
- previousRentals(): void - Shows the user a list of their previous rentals.

Client History - The Client History class is responsible for viewing previous rentals and emailing receipts. It does this by viewing the Client ID.

Attributes -

- clientID: String - Requires the client-specific ID in order to find their history

- perviousRentals: Array<Rental Contract> - This checks the Array to see what the previous rental contracts are.

Operations -

- viewPreviousRentals():void - This allows the class to access the previous rentals inorder to gather the information it needs.
- emailRecepit():void - Once the information has been gathered this is used inorder to email the receipt to the client.

Development Plan and Timeline:

Partitioning of tasks -

Backend Developers - The Bulk of the work will be divided among the backend developers. Programmers will be responsible for the coding of the main functions of the system. Server architectures are responsible for making sure the clients get fast responses. A security team will be tasked with securing and encrypting user info as well as two-factor authentication. The database team will ensure that the data is stored properly.

Frontend Developers - The user interface for the web and app, as well as usability, will be handled by the frontend team. The UI will ensure that the app is visually pleasing and fits the company's themes. Additionally, they will make the app as accessible as possible in order to reach the maximum amount of potential clients.

Timeline -

With scale in mind, the expected delivery this app should be around 6 - 9 months based. Based on the amount of time that is required to be invested to make it both web and app accessible. The complexity of having to make an app with user accounts, contacts, stored car information, and GPS will cause much more effort and time tobe invested. Easy beta tests and implementations can be expected as early as 5 months.