Automotive Management Cloud System

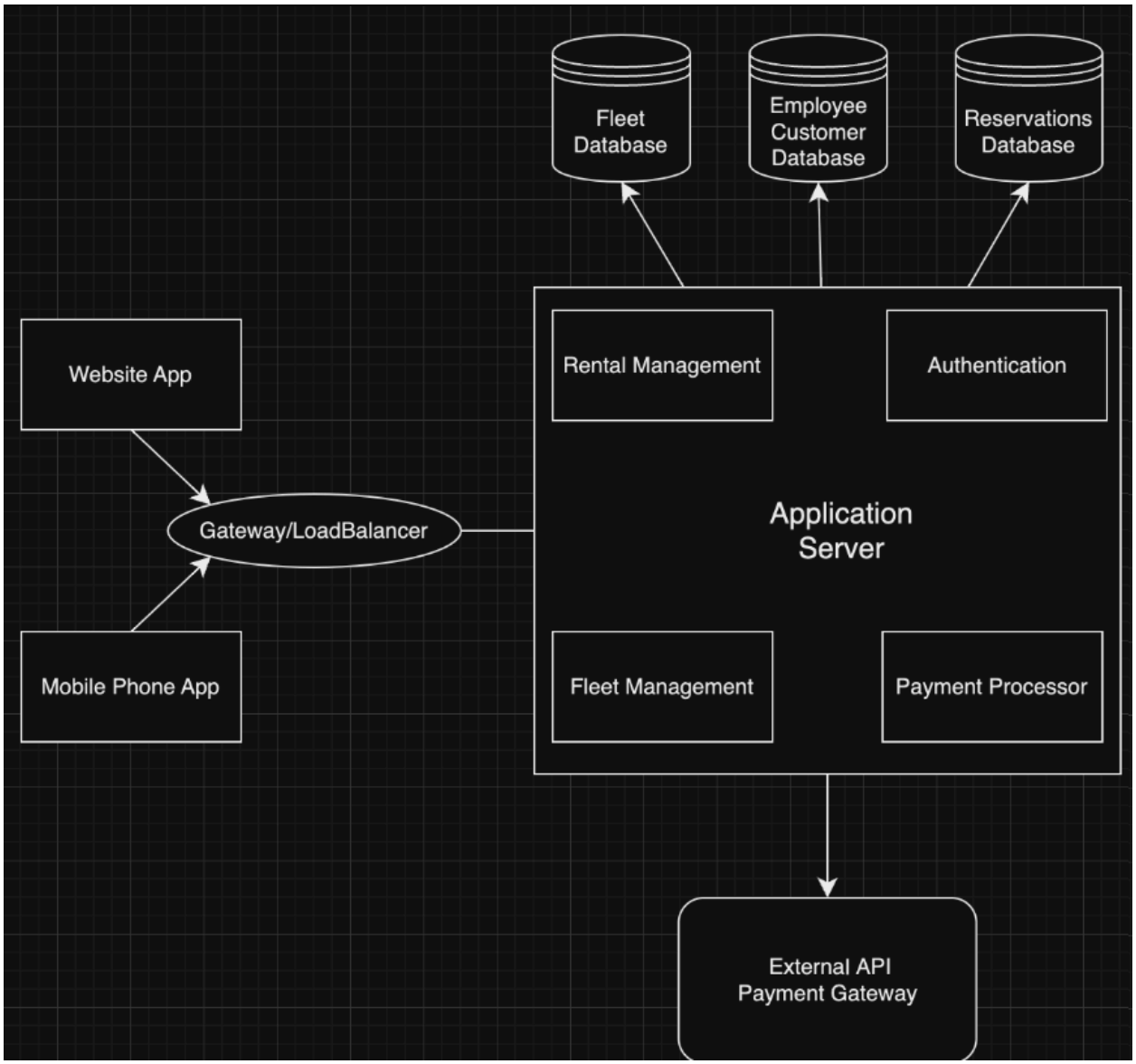
By Max Carrillo, Jeremiah Cho, and Ramil Carino

Introduction and Overview

This document outlines the functional and nonfunctional requirements for the Automotive Management Cloud System (AMCS) Software for BeAvis Car Rental Company, to modernize the car rental process and operations. Integrating the AMCS expands the capabilities of BeAvis and transitions the current manual, pen-and-paper, method to a fully digital, online platform. The AMC system aims to handle the car rental business for customer applications and reservations, maintaining records and fleet data, and business control for BeAvis Car Rental Company. Integration of the AMC system reduces operating costs, minimizes record errors and inaccuracies, and modernizes the company for customers and employees to be accessed by mobile and website applications.

Automotive Management Cloud System (AMCS) software provides a suite of online services to replace traditional paperwork car rental processes with an efficient and user-friendly application, accessible across iOS and Android devices, as well as a website.

2) Software Architecture Diagram Overview



Architecture Diagram Description

The AMCS Software Architecture diagram for BeAvis Car Rental. There are two access points for the AMCS application server. Through the website application and mobile application (across iOS and Android Devices) the user is able to access the application server to initiate rental process tasks including vehicle browsing and reservation services, user account management, and inventory and reservation management, as well as make payments. These functions are displayed as arrows or connectors to the Gateway and communicate each function between the application and application server. The application server has four different processors to manage the data and databases that allow it to continuously be up to date. The application server handles payments in particular that utilizes an external API Gateway for

payments.

Gateway / Load Balancer API - This component ensures that computationally intensive workloads are equally distributed across multiple servers to avoid potential outages or slowdowns.

Application Server - This component handles business logic for each portion of the application. It contains microservices that handle each separate purpose of the application. For example, the Authentication service would handle user account creation, email verification, and logging in/out of the apps.

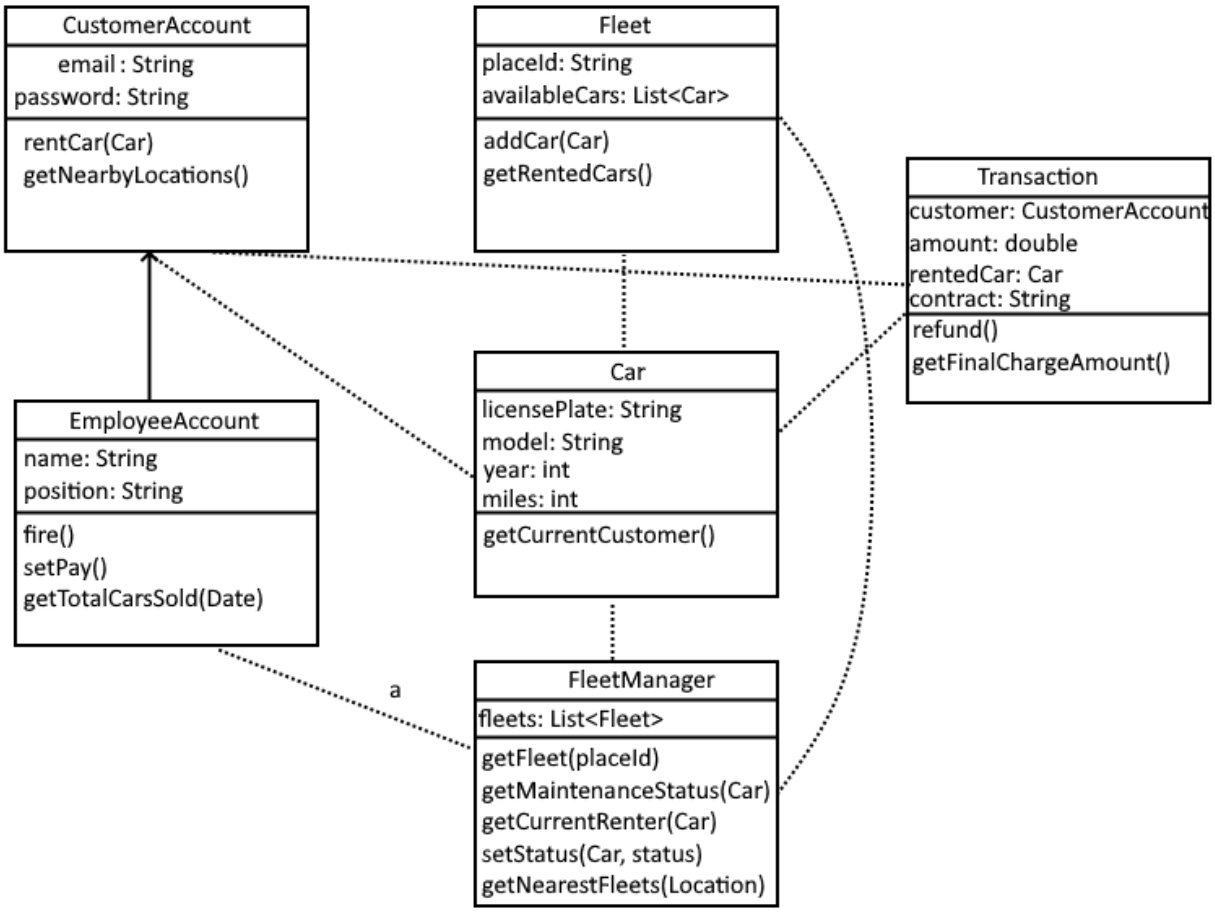
External API Payment Gateway - Interactions between the application server and a third-party service that securely manages all payments. This is necessary to ensure our application uses a reputable payment handler that users can trust with sensitive payment information.

Fleet Database - Stores records about each fleet of cars owned by BeAvis. Access to this information is guarded by the Server Application.

Employee Customer Database - Stores records about employees and customers, including all rental, contract, and payment histories.

Reservations Database - Stores current information about which cars are currently being rented and which cars are available for rent.

UML Class Diagram



CustomerAccount - Stores information about a customer that can rent cars.

attributes: email and password used to sign into the account. Email must be verified through the application server.

operations:

- rentCar(Car): Rents a car. Creates a new Transaction capturing information such as who is renting a car, the amount the rental costs before local taxes, payment processing, and other unpredictable fees, the car that was rented, and a String containing the location of a Contract PDF on the Application server.
- getNearbyLocations(): Returns a list of nearby Fleets in the vicinity of the user based on their GPS location.

EmployeeAccount - Stores information about an employee. Is a subclass of CustomerAccount, as employees also have an email and password.
attributes:

- name: the first and last name of the employee
- position: the specific job position or role the employee has within BeAvis.

operations:

- fire(): Fires the employee from BeeAvis. Can only be performed by employees with management roles.
- setPay(): Sets the hourly pay rate for this employee. Can only be performed by employees with management roles.
- getTotalCarsSold(Date): Gets the number of cars an employee sold on a given day so that commissions may be calculated.

Car - Stores information about a car owned by the BeAvis company.
attributes:

- licensePlate: The license plate of the given car.
- model: The model of the given. For example, Honda Civic.
- year: The year of the car's model. For example, 2005.
- miles: The total miles driven in this car.

operations:

- getCurrentCustomer(): Returns the CustomerAccount of the customer who is currently renting this car, or none if this car is not being rented.

Fleet - Represents a fleet of cars, and is associated with a physical location where rentals would usually be made on pen and paper before introduction of this service.
attributes:

- placeId: The ID of the physical location this fleet is kept at. May correspond to a place ID from Google Places API, for example.
- availableCars: A list of all cars that may or may not be rented from this fleet.

operations:

- addCar: Adds a car to this fleet. Used when a new fleet is created or a car is purchased by a certain location that wants to expand this fleet.
- getRentedCars: Returns a list of cars that are already being rented by a customer at the current moment.

FleetManager - Class that stores information about all BeAvis fleets across all physical locations. Can be used by management to control fleets of cars or manage cars across different fleets.
attributes:

- fleets: The list of all fleets owned by BeAvis.

operations:

- getFleet(placeId): Gets all fleets associated with the physical location referenced by the given placeId.
- getMaintenanceStatus(Car): Gets the maintenance status of the given car, such as if the car is in working condition or not, and how long a car is estimated to be out of service before being safe to reintroduce into the fleets.
- getCurrentRender(Car): Determines the current customer renting the given car, or returns none if nobody is renting the car.
- setStatus(Car): Setter that changes the maintenance status of a given car.
- getNearestFleets(Location): Given a GPS location, returns a list of fleets in order of increasing physical distance from the location.

Transaction - Stores information about a specific time when a car was rented.
attributes:

- customer: The account of the customer who was involved in this transaction.
- amount: The cost of the transaction to the customer.
- rentedCar: The Car that was rented by the customer in this transaction.
- contract: A String pointing to a PDF for the contract of this transaction on the Application Server's file storage.

operations:

- refund(): Returns all the money paid by the customer to the customer. Can only be performed by management in the case where they would like to reimburse the customer for a faulty rental.
- getFinalChargeAmount(): Calculates the actual cost to the customer of this transaction based on local fees and taxes as well as payment processing fees, if credit card was used.

Development Plan

The development of the AMCS Software involved the collaborative minds of three talented individuals, each contributing to every part of the project. From the Introduction and Overview to the Software Architecture and UML Diagrams, everyone’s ideas, opinions and work were valued. This method ensured each team member stayed

engaged throughout the project. Of course, there were days where priorities did not align, however, that was easily mediated through the trust and mutual understanding our contract was founded on. Each day, the team tackled every segment of the project from the Introduction and Overview to the Software Architecture design and descriptions, and lastly the UML design and description to give each person a responsibility and a voice.

Five Key Phases

1. Requirements Gathering (3 Weeks)

   This first phase involves the members of BeAvis Car Rental Company to create a guideline for functional and functional requirements for the system.

2. System Design (3 Weeks)

   This second phase is to create a detailed system architecture diagram and UML diagram that include key components and their interactions,

3. Development (5 Weeks)

   The third phase focuses on building the software for AMCS with the thought of the car rental industry at the core for front end mobile app and website design, as well as the controls of management for the fleet to customer information.

4. Testing (3 Weeks)

   The fourth phase is the testing phase for the key components and ensures the communication and result meets the clients requirements and the expected outputs. By using structural and functional testing, the software undergoes rigorous testing for the validation and verification this system strives for.

5. Launching and Maintaining (1 Week)

   The final phase is a confident live launch of the AMCS as well as training for BeAvis staff and management, as well as establishing maintenance for stability and software updates.

The Automotive Management Cloud System (AMCS) software is the transformative gateway designed to modernize the car rental industry for BeAvis Car Rental Company. By transforming from a traditional manual paperwork system to a fully digital platform, AMCS provides a suite of online services focused on a seamless and efficient experience for both customers and employees aimed at enhancing business capabilities, reducing operational costs, and minimizing errors. AMCS offers customers a user-friendly interface across mobile and web platforms for browsing vehicles, creating reservations, and managing accounts, while giving comprehensive control to BeAvis for inventory management, reservation handling, and payment processing. AMCS is capable of growing with BeAvis business with the flexibility to add and alter

any features, added security to reduce the risk of fraudulent transactions, scalability for large-scale reporting and system loads, and many more.