# BeAvis Car Rental Software Design Specification

Authors: Spencer Schmitt, Alex Zucker, Sean Tran

Delivered 10/11/2024

## Section 1: Introduction

The client, BeAvis, has ordered the creation of the software system that will be used to replace the pen-to-paper car rental process, making it electronic-based and useful for managing the inventory of rental cars. The client sees physical documents and records as impractical and wants to keep up with an evolving electronic world. The client aims to digitize their day-to-day operation. We expect that overhauling the process will increase customer satisfaction and employees' workload efficiency.

The client's users, customers, and employees will interact extensively with the system. The users, managers, and administrators will have access to the backend of the software system for their respective tasks.

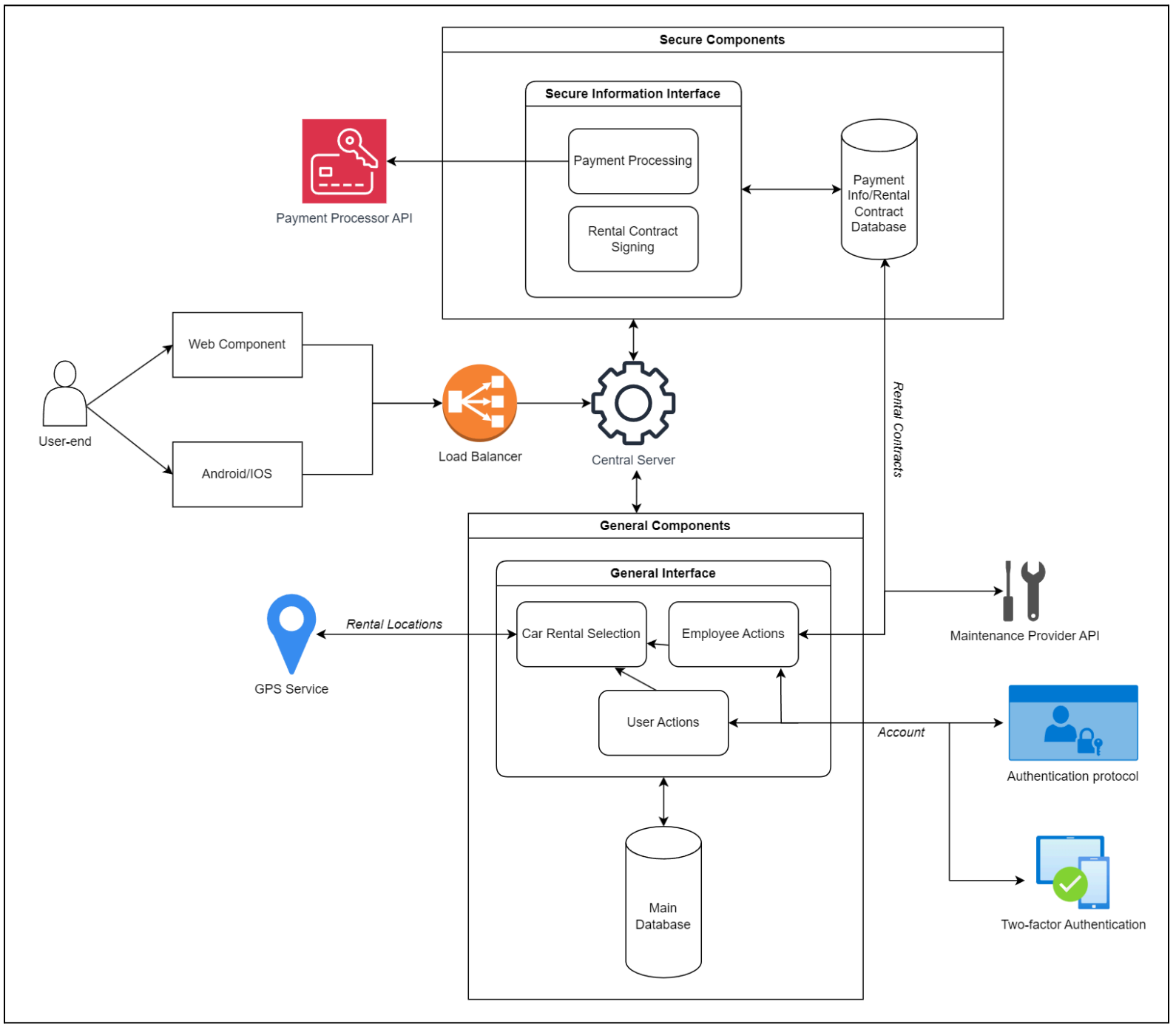
Successful design and implementation of the software system will improve the ease of customer experience, making it comfortable for them. The system will also increase employee productivity and decrease the time for workload by several hours per week.

## Section 2: Software Architecture Overview

### Section 2.1: Architecture Diagram of all major components

This software architecture diagram (see next page) covers all the major components necessary for the BeAvis Car Rental software

# BeAvis Software Architecture Diagram



# Description:

Proceeding this diagram is a description of all the components and connections within it, starting from the very left.

User End:

- This component represents the point of interaction between the user and BeAvis Car Rental Software. The user-end has two arrows to "Web Component" and "Android/IOS", representing the different interfaces a user can interact with.
    - Web Component
        - The user should be able to interact with the software via a web-app interface.
    - Android/IOS
        - The user should be able to interact with the software via a mobile interface, accounting for both IOS and Android.

Load Balancer:

- Arrows pointing from the Web Component and Android/IOS to the load balancer represent how traffic from both interfaces will need to be throttled and distributed between BeAvis servers (which will be done by the Load Balancer).

Central Server:

- Traffic and inputs passed through the load balancer will then be directed by a central server based on if the input requires access to secure or general components. It also handles communication and processes between the two major types of components.

General Components

- General Interface
    - The general interface is an organizational distinction that represents user actions (Including both customers and employees) that interact with the main database or general 3rd party APIs including:
        - GPS Service:
            - Necessary for finding nearby BeAvis Car Rental Locations using GPS.
        - Authentication Protocol
            - Necessary for authenticating login attempts.

        - Two-Factor Authentication
            - Necessary for allowing the user to set up two-factor authentication for their account to better secure it.
        - Maintenance Provider API
            - This API allows an employee to efficiently submit maintenance requests through the BeAvis software system.
- Main Database:
    - This database contains all data deemed "low-priority" for security purposes. This includes data like user car rental history and car status.

Secure Components

- Secure Interface
    - The secure interface is an organization distinction that represents user actions related to sensitive information like payment processing and rental contract signing.
    - Payment Processor API
        - The secure interface interacts with the payment processor API to handle payments.

- Payment Info/Rental Contract Database
    - This database is designated as a high-priority database that holds sensitive information like payment information and rental contracts. It may also hold other sensitive information like login credentials.

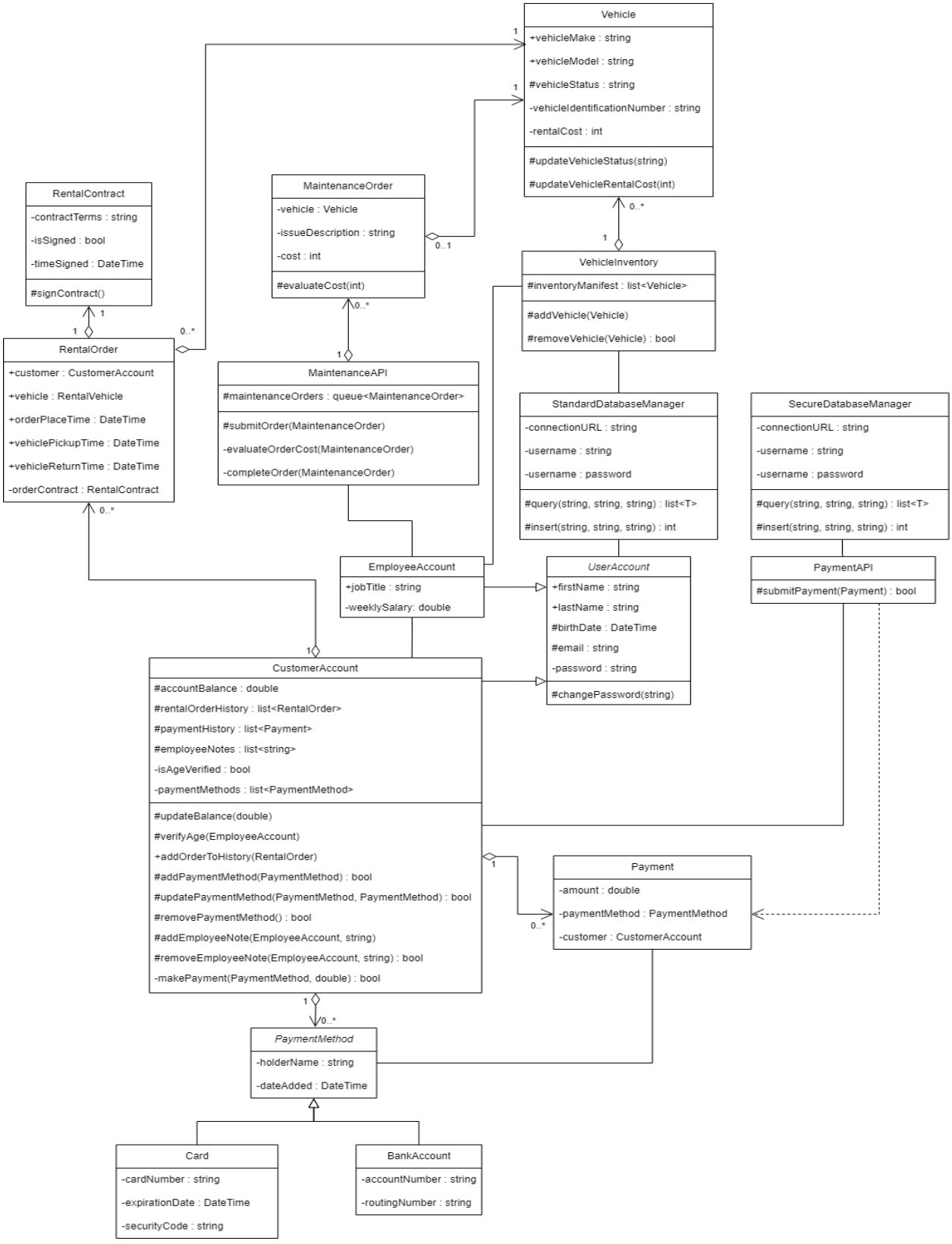
Miscellaneous Connections

- The majority of connections with 3rd party systems represent necessary implementations that the BeAvis system must interact with to properly function.

**Section 2.2: UML Class Diagram**

This UML class diagram (see next page) illustrates the classes representing the backend of the BeAvis Car Rental software, as well as their member variables, functions, and relationships to one another.

# BeAvis UML Class Diagram



# Description:

This section will provide a textual summary of the functionality of each depicted class, as well as its relationships to other classes.

Nonvolatile storage classes

- StandardDatabaseManager
    - Acts as an interface between the software system's code and its permanent data storage solution. This class connects to the database used to store all information not related to customer payments. Runtime data-heavy classes, such as those that store lists of objects, query this class to obtain and update their respective member variables.
- SecureDatabaseManager
    - Functions nearly identically to StandardDatabaseManager, but connects to the secure database that is solely responsible for storing and retrieving information relating to customer payments and payment methods. As a result, this class is used by PaymentAPI for read/write requests.

API classes

- PaymentAPI
    - Acts as the interface between the software system's code and the third party service used to process customer payments. This class is dependent on the Payment class to function, and simply reads data from parameterized Payment objects in order to submit payment requests for processing.
- MaintenanceAPI
    - Acts as the interface between the software system's code and the third party service used to repair rental vehicles. Currently outstanding orders are represented via a queue member variable of type MaintenanceOrder, which can be added to by EmployeeAccount instances and removed from by the maintenance service.

User classes

- UserAccount
    - An abstract class which serves as the superclass for all instantiable account class objects. Contains member variables and functions that are required for both customers and employees alike. UserAccount

instances derive their initialized data values by querying StandardDatabaseManager.

- EmployeeAccount
    - A subclass of UserAccount which represents employee users of the software system. Compared to their customer counterparts, EmployeeAccount instances have elevated data read/write permissions within the software system. These elevated permissions are illustrated by association relationships with CustomerAccount, MaintenanceAPI, and VehicleInventory.
- CustomerAccount
    - A subclass of UserAccount which represents customer users of the software system. This class contains a large number of member variables and functions which are used to store, access, and modify data pertaining to individual customers. These member variables include lists of RentalOrder, PaymentMethod, and Payment objects, all of which are illustrated via aggregate relationships to the respective classes.

Payment information classes

- Payment
    - A simple class which serves as the representation of a single payment made by a customer user of the software system. Instances of this class are used to represent past payments that have been made, as well as payments that are currently being processed by the third party payment processor service.
- PaymentMethod
    - An abstract class which serves as the superclass for all instantiable payment method objects. Contains member variables which are relevant to all payment methods.
- Card
    - A subclass of PaymentMethod which represents a debit or credit card that has been registered to a CustomerAccount instance.
- BankAccount
    - A subclass of PaymentMethod which represents a bank account that has been registered to a CustomerAccount instance.

Rental order classes

- RentalOrder
    - A class representing an individual order for a rental vehicle placed by a CustomerAccount instance. Among other member variables,

contains a RentalContract and Vehicle, which are visualized by aggregate relationships with 1:1 multiplicities.

- RentalContract
    - Represents the legally binding contract between vehicle renter and rentee. A RentalContract instance is always related to 1 and only 1 RentalOrder instance.

Vehicle inventory classes

- VehicleInventory
    - Serves as the digital inventory manifest of the software system. This inventory is directly represented by a member variable list of type Vehicle, which is initialized via queries to StandardDatabaseManager.
- Vehicle
    - A class representing an individual rental vehicle registered to the VehicleInventory digital manifest. Corresponds to any number of RentalOrder instances, only one of which can be a currently active order. May also correspond to 1 currently outstanding MaintenanceOrder.
- MaintenanceOrder
    - Represents a single maintenance request that has been sent to the third party maintenance service. Contains member variables representing the vehicle to be serviced, as well as a single member function allowing the maintenance provider to specify the quoted cost for fulfilling the service request.

## Section 3: Development plan and timeline

### Section 3.1: Partitioning of tasks

***Development plan -*** For the current phase, the software requirements should have already been finished and given to the development and implementation team to start the process of designing the software system via the software architecture diagram and the use case diagram. With the completed diagrams, the team should be able to start implementing the software requirements in accordance.

**First week -** Dedicated to gathering user and system requirements

**Second week -** Dedicated to designing the software architecture diagram and the UML diagram following the user and system requirements of the software system.

**Third to sixth week -** Dedicated to the implementation of the software system following the user requirements, system requirements, functional and non-functional requirements, the SWA diagram, and the UML diagram

**Seventh to Eleventh week -** The most important phase where we have to perform quality checks on the system, making sure that the system has performed the necessary actions that the client wants without fail and bugs.

**Eleventh to Fourteenth week -** Maintaining the software system.

### Section 3.2: Team member responsibilities

#### Alex Zucker

*Team roles*

Design
Implementation

*Responsibilities*

Responsible for the UML Diagram of the software system
Collaborated with Spencer Schmitt and Sean Tran to implement the software system following the software requirements.

Collaborated with Spencer Schmitt and Sean Tran to design and format the current software design specification document.

#### Spencer Schmitt

*Team roles*

Design
Implementation

*Responsibilities*

Responsible for the Software Architecture Diagram of the software system

Collaborated with Alex Zucker and Sean Tran to implement the software system following the software requirements.

Collaborated with Alex Zucker and Sean Tran to design and format the current software design specification document.

**Sean Tran**

*Team roles*

Design
Implementation

*Responsibilities*

Collaborated with Spencer Schmitt and Alex Zucker to implement the software system following the software requirements.

Collaborated with Spencer Schmitt and Alex Zucker to design and format the current software design specification document.