Software Design Specification

BeAvis Car Rental System

Connor Pham, Devin Widmer

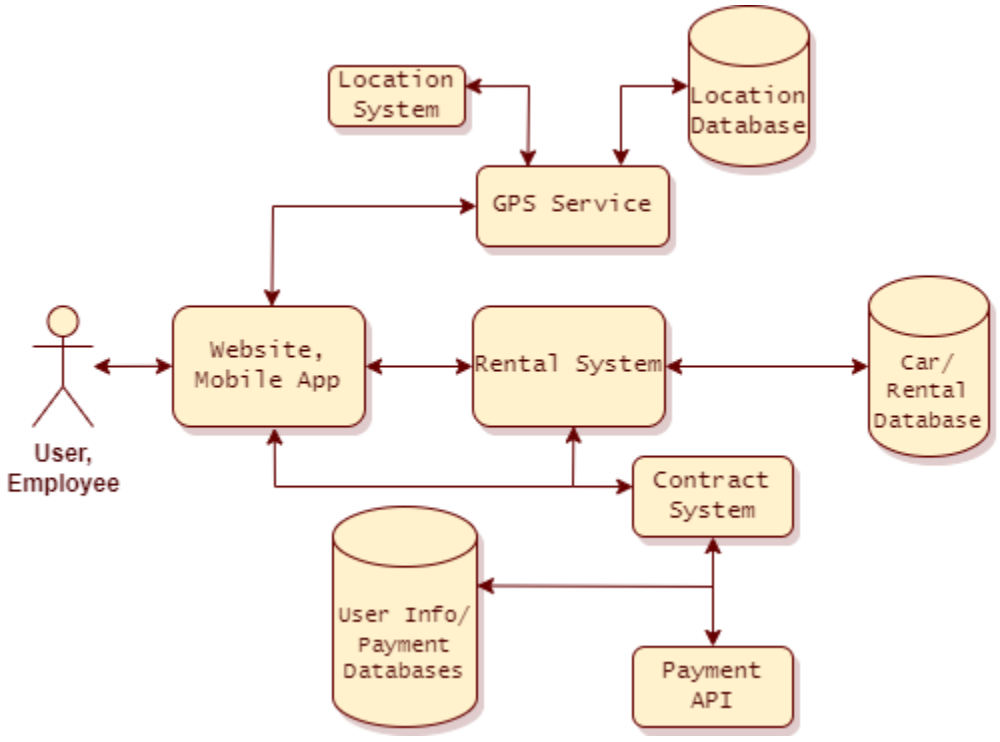
March 8, 2024

System Description:

The car rental system will enable users to create accounts, verify their identity, link payment methods, electronically sign rental agreements, and manage their rentals through an Android/iPhone mobile app as well as a dedicated website. Additionally, it will feature an administrative interface for employees and administrators to manage customer accounts, monitor car statuses, and review contracts. This document aims to provide a comprehensive specification for the development of a car rental system designed to digitize the traditional pen-and-paper process. Accessible via a mobile application and a website, the system offers an efficient, user-friendly, and secure way to rent cars. It is intended for customers seeking rental services, employees who manage rental operations, and administrators in charge of overseeing secure information.

Software Architecture Overview:

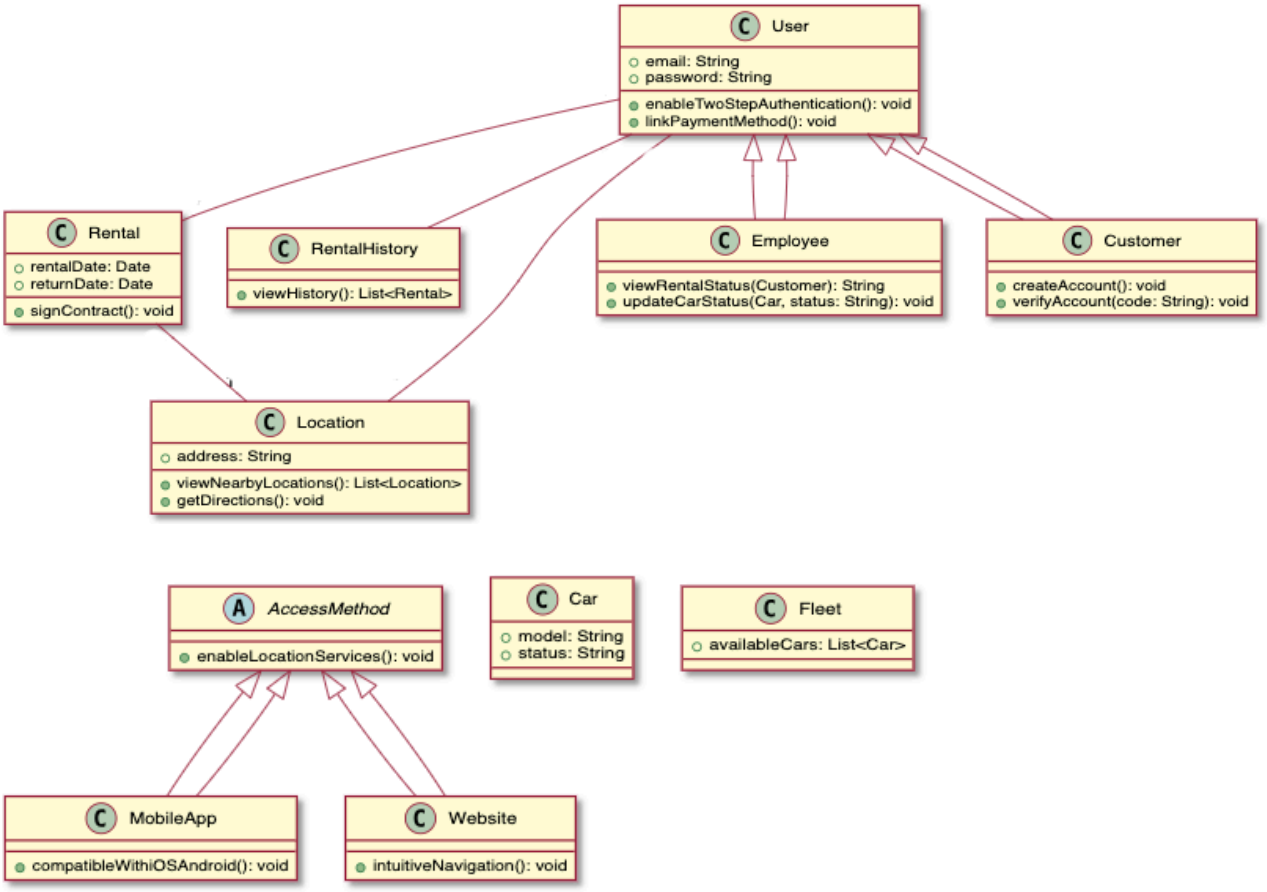
Software Architecture Diagram:



Software Architecture Description:

The software architecture diagram depicts the overall architecture of the BeAvis Car Rental System. The components, starting with the User and Employee are the systems users. They interact with the system through a website or mobile app, which serves as the primary interface for the car rental services. The Website and Mobile App component acts as the way for users to access the rental system. It provides functionality for users to browse available cars, make car rentals, payments, sign contracts, and manage their accounts by connecting to the rental system. The central rental system component manages the core functionality of the car rental service. It processes rental requests, retrieves information from the databases, and coordinates and connects with other APIs and systems like the location and contract systems. It is connected to the Car/Rental Database, which stores information about the available cars and their rental status. The GPS service is responsible for managing geographical information, possibly used to track the cars' locations or to assist users in finding rental locations. It is connected to the location database which stores location-related data and the location system which provides tracking tools and information. The Car/Rental database contains all data related to the cars available for rent, including their models, availability, status, or rental history. The contract system manages rental agreements between the service and the users. It records terms, conditions, and duration of car rentals. This system interacts with the user info and payment databases which store personal and payment information of users, ensuring that the service has the necessary data to process transactions and maintain records for administrators, users, or employees to access. It is also connected to the payment API, that handles payment transactions. This would be used to process user payments securely when renting a car.

UML Diagram:



UML Class Description:

In this UML diagram the 'User' class holds most of the functionality. In the 'User' class there are 2 string variables known as 'email' and 'password.' It has two methods to enable two-step authentication and links payment method, both of these are of void return type that take in no parameters. The 'Employee' class inherits from 'User' and the operations include 'viewRentalStatus(Customer)' which returns a string and requires a parameter 'Customer'. 'UpdateCarStatus(Car, status: String): void' is a method that updates the current status of the car while taking in two strings as parameters. Similarly, the 'Customer' class inherits from the 'User' class and has two operations one to 'createAccount():' which returns void requiring no parameters and another to 'verifyAccount(code: String): void.' This takes in a string as input and

does not return anything to the user. The input is a code whether it is a full string or a string concatenated with numbers to compose its ID.

Moreover, in the ‘Rental’ class, it has two attributes: ‘rentalData: Date’ and ‘returnDate: Date.’ These two variables are of type Date. They do as their name describes, one tells one the rental begins and when the rental is over. The only operation it does is ‘signContract(): void,’ which only updates the software system and does not return anything. In the ‘RentalHistory’ class, the operation ‘viewHistory(): List<Rental>’ looks up if the user has previously rented any vehicles. It returns a list type whether it is a linked list, arraylist, or other. The ‘Location’ class has a string called address which stores the address of the car rental place and it has two methods ‘viewNearbyLocations(): List<Location>’ and ‘getDirections():’. Where one returns a list of ‘Location’ type wherever there may be rentals nearby, both require no parameters.

Furthermore, in the Car class some of the attributes include two String variables called ‘model’ and ‘status’ where one tells what the model of the car is and the other concerns the car’s availability, carfax reports, etc. While the ‘Fleet’ class has a list variable that contains a list of available cars within the database of the car rental establishment and can be rented by the customer.

Throughout this entire UML there is an interface called ‘AccessMethod’ which has the operation ‘enableLocationServices(): void’ that allows the user to have access to location services. This only applies if this is implemented. Likewise, the ‘MobileApp’ implements the previous Interface which has the operation ‘compatibleWithIOSAndroid(): void’ that allows for compatibility with iOS and Android devices with the car rental’s system. Lastly, the ‘Website’ class is implemented from the ‘AccessMethod’ Interface and it has a method that allows for users to surf the website, it is called ‘intuitiveNavigation(): void’ and it serves as a more general function.

Development Plan and Timeline:

2 Weeks: Implementation Planning
Begin formulating a plan for development working with team to identify requirements, risk assessments, and resources provided. Follow software architecture and UML planning.
(Connor & Devin)

1.5 Months: Web and Mobile Application Development
Begin development of UI design, and application development to satisfy all requirements of overall application structure. Develop according to software architecture plan. Develop based off of specifications in software and hardware requirements.
(Devin)

2 Months: Development, Database, API
Begin development of Car/Rental and User Info/Payment databases and develop system components and interfaces. Build based off of software and hardware requirements and resources given to create required software systems.
(Connor)

1.5 Months: System component implementation and testing
Integrate developed components and interfaces into application. Include concurrent testing and test cases to identify risks, failures, and to streamline implementations.
(Connor & Devin)

1 Month: Overview and Deploy
Team overviews product after identifying issues, risks, and failures to start deployment of the overall system.
(Connor & Devin)

After release Indefinietely: Maintenance and assistance
Perform needed maintenance to fix errors, failures, risks, issues, or new implementations required by the customer. Providing ongoing support and system updates.
(Connor & Devin)