VIVU

Software Architecture Document

Version 1.8

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 23/11/2023 | 1.0 | Fill out the content of GUI: MyTrip and Profile | Hoàng Lê Cát Thanh |
| 24/11/2023 | 1.1 | Fill out the content of Architectural Goals and Constraints, GUI: Login, Sign up. | Đoàn Thị Yến Nhi |
| 25/11/2023 | 1.2 | Fill out the content of GUI: Homepage and Booking screen.  Fill out the Use Case Model | Lê Ngọc Thảo |
| 27/11/2023 | 1.3 | Fill out the content of Logical view | Đoàn Thị Yến Nhi |
| 27/11/2023 | 1.4 | Fill out the content of Component Controller Back-end | Trần Trung Hiếu |
| 28/11/2023 | 1.5 | Fill out the content of Component Controller Front-end | Trần Trung Hiếu |
| 28/11/2023 | 1.6 | Fill out the content of Component  Model and Class diagram | Võ Cao Trí |
| 20/12/2023 | 1.7 | Fill out the content of Folder’s structure | Trần Trung Hiếu |
| 24/12/2023 | 1.8 | Review doc and fix UC model, logical view, section 5 and 6 | Lê Ngọc Thảo |

Table of Contents

[**1. Introduction 4**](#_heading=h.gjdgxs)

[**2. Architectural Goals and Constraints 4**](#_heading=h.30j0zll)

[**3. Use-Case Model 5**](#_heading=h.1fob9te)

[**4. Logical View 6**](#_heading=h.2et92p0)

[4.1 Component: GUI 7](#_heading=h.tyjcwt)

[4.1.1 LogIn 7](#_heading=h.uw20n09jvrcf)

[4.1.2 Homepage: 8](#_heading=h.80t279893pxj)

[4.1.3 BookingCarScreen: 9](#_heading=h.vw1qnhdhjlj4)

[4.1.4 MyTrip: 9](#_heading=h.rjb3vsd7w63u)

[4.1.5 Profile: 10](#_heading=h.z1bb26seuafx)

[4.2 Component: Controller(Front-end) 10](#_heading=h.b2i8i39ki5be)

[4.3 Component: Controllers (Back-end) 11](#_heading=h.i1nfdeisld3x)

[4.3.1. Account controller 11](#_heading=h.92y8fy7xyf9e)

[4.3.2. Car controller 12](#_heading=h.ihmdouopfo5k)

[4.3.1. Reservation controller 12](#_heading=h.dpzd5y3q4vk4)

[4.3.2. Review controller 13](#_heading=h.wymc9ykbnjv)

[4.3.1. Payment controller 13](#_heading=h.5j2egt14dpkg)

[4.4 Component: Models 14](#_heading=h.15qzhsvbmtfs)

[4.4.1. Account Model: 14](#_heading=h.902lp2wwgpky)

[4.4.2. Car Model: 15](#_heading=h.mfuvpz33dtkc)

[4.4.3. Car Rental Deal Model: 15](#_heading=h.9mj3p1fa23c5)

[4.4.5. Payment Model: 16](#_heading=h.wm0ixs3xu9bc)

[4.5. Component Navigator: 17](#_heading=h.ejko61jmcgrk)

[4.6. Interaction between components in the server:](#_heading=h.7h83jml9365) 18

[4.7. Class diagram: 19](#_heading=h.8g4p1f9ndia3)

[4.8. Entities Relationship Diagram: 20](#_heading=h.ydbqtfivipz4)

[**5. Deployment Diagram: 20**](#_heading=h.3dy6vkm)

[**6. Folder’s structure: 22**](#_heading=h.1t3h5sf)

Software Architecture Document

# Introduction

The introduction of the Software Architecture Document provides an overview of the entire Software Architecture Document. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Architecture Document.

1.1 Purpose

Provides an overview of the architecture of the entire system and from many different perspectives. Besides, functional descriptions and use cases are converted into the system architecture.

1.2 Scope

The document will focus on decisions when choosing a software architecture for a system.

1.3 Definitions, Acronyms and Abbreviations

None.

1.4 References

* Vision document
* Use-case specification document

1.5 Overview

* This software architecture document includes:
* Architectural Goals and Constraints: The goals and constraints of software architecture.
* Use-case Model: Contains the use-case model.
* Logical View: Provides a list of components in the system and the relationships between them.
* Includes package diagram and class diagram.
* Deployment: Method of deploying application components.
* Implementation View: Directory structure containing code and how to install components.

# Architectural Goals and Constraints

Programming Language:

* For the front end:
  + Dart: used to build web applications through Flutter, a framework for web and mobile app development. Flutter leverages Dart to construct apps featuring responsive and high-performance user interfaces (UI).
* For the back end:
  + Python: It utilizes the Flask framework to establish an API connection, acting as a communication bridge between the front end and back end, and is employed for storing user information from the app in the database.

Programming Environment:

* Visual Studio Code: is a free and open-source code editor developed by Microsoft.
* Android Studio: allows testing the application on various devices and Android API levels without needing to have each physical device.

Application Environment: Mobile

System Requirements: The user’s device must be connected to the internet.

Performance Requirements:

* The app should be able to handle 200 concurrent users.
* The application will restart no more than 10 minutes after a crash.
* Operations must be processed within 5 seconds after the customer performs the function.

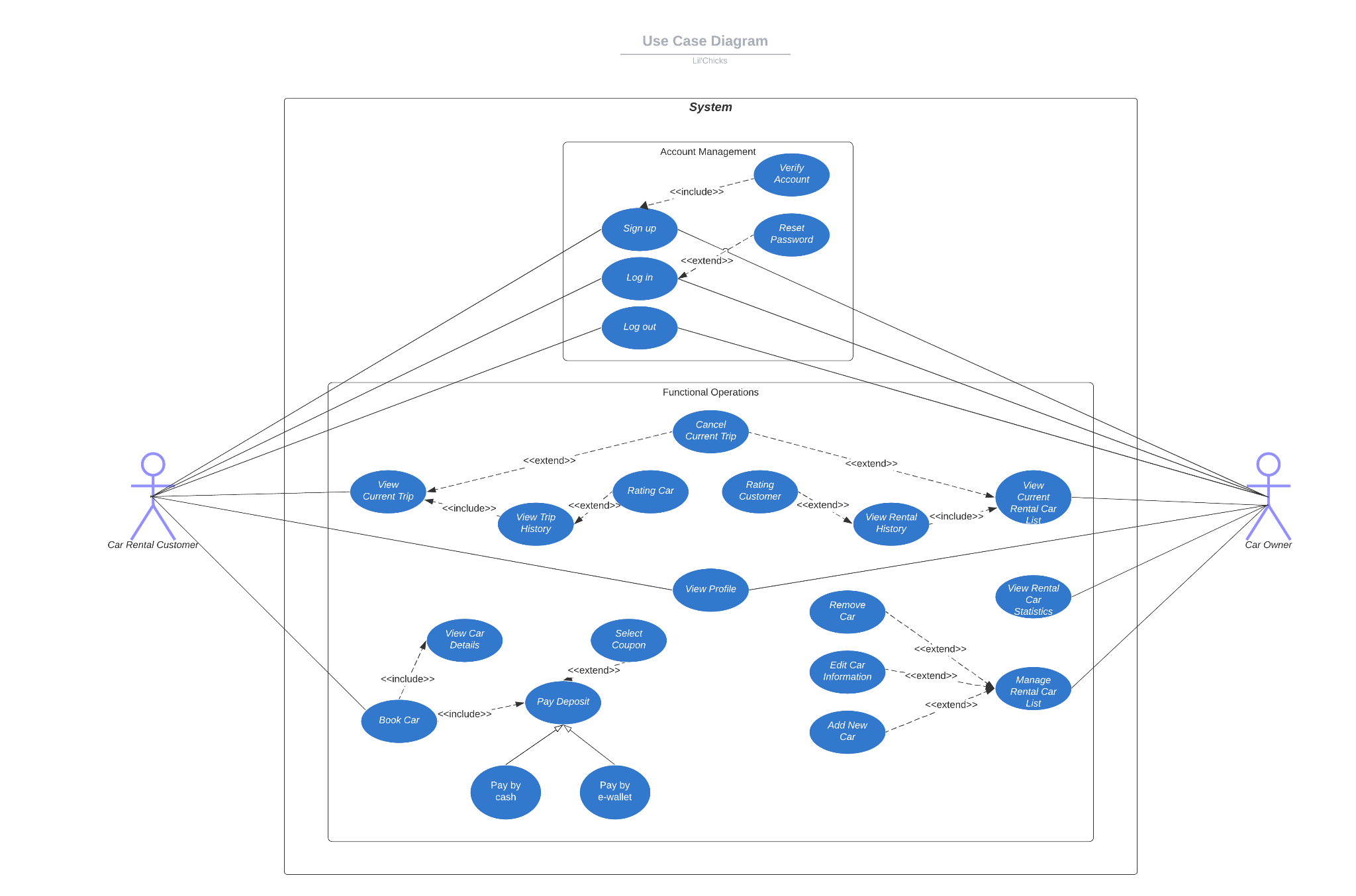
Security Constraints:

* User information in the database is continuously replicated to the backup database.
* Ensure the security of personal information and data.
* User data is applied with security techniques to avoid data overflow or theft.

Database Requirements:

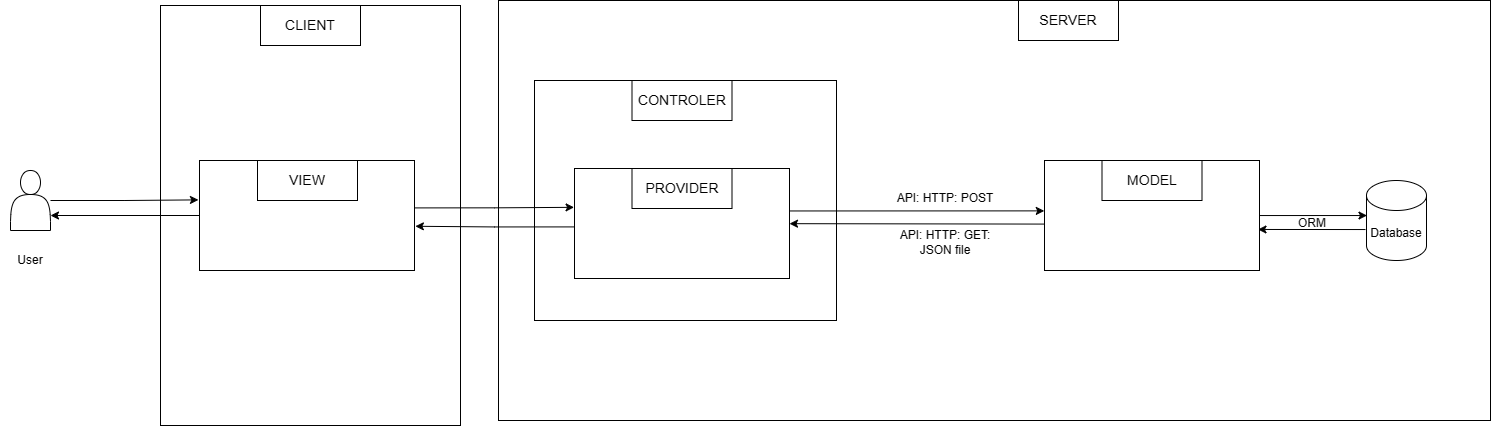
* Each account must be designated as either a car renter or a car owner to prevent conflicts and ensure data integrity. This avoids overwriting and potential data misunderstandings between car owners and car rental information.
* When registering an account, do not allow duplicate usernames to prevent errors and ensure account security for users.
* Avoid car rental conflicts: 1 car at a time can only be rented by 1 person, avoiding the situation where 2 people can rent a car at the same time.

# Use-Case Model



# Logical View

The logical view of the VIVU app is described through the Model-View-Controller (MVC) design pattern. The main components and their operations are outlined as follows:



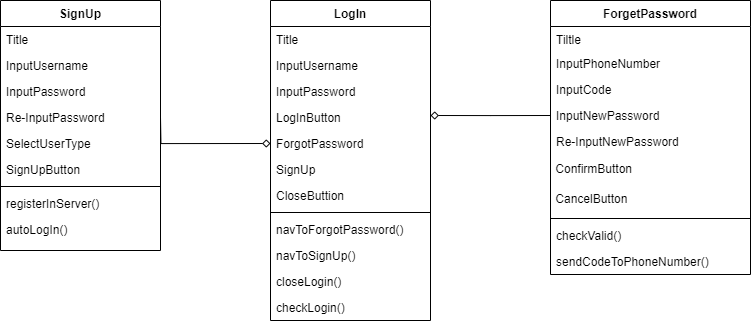
* View: First, serves as the user interface that presents data to the user and captures user commands or interactions. The data displayed within the View is obtained through the Controller, which acts as an intermediary between the View and the Model. Second, the View makes API requests to the server to retrieve or post data as needed. These API requests are processed on the server-side and the results are then rendered on the user interface, providing an interactive experience for the user.

* Controller: is the central hub for processing all business logic and handling incoming requests from the user interface. It manipulates data using the Model, which represents the application's data and business rules. After processing data, the Controller selects an appropriate View to render that data back to the user. Additionally, the Controller communicates with a component referred to as the "Provider," which serves as a service layer. This Provider facilitates interaction with server-side APIs, typically over HTTP.

* Model: is responsible for managing the application's data and enforcing business rules. It interacts with a MySQL database through an Object-Relational Mapping (ORM) layer. This ORM layer abstracts the complexities of database operations, making it easier to perform data manipulation and querying.

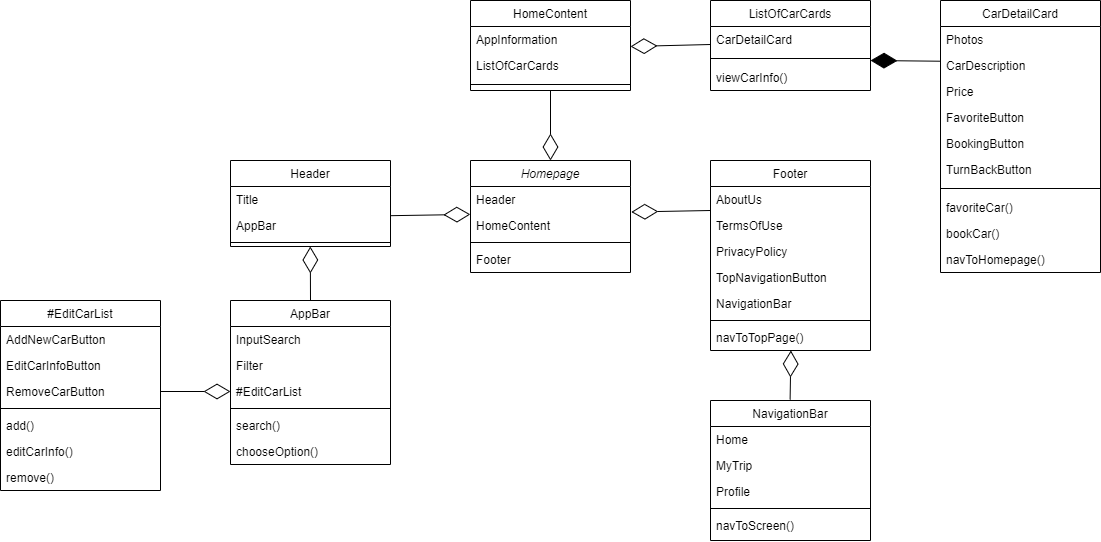
## Component: GUI

### LogIn

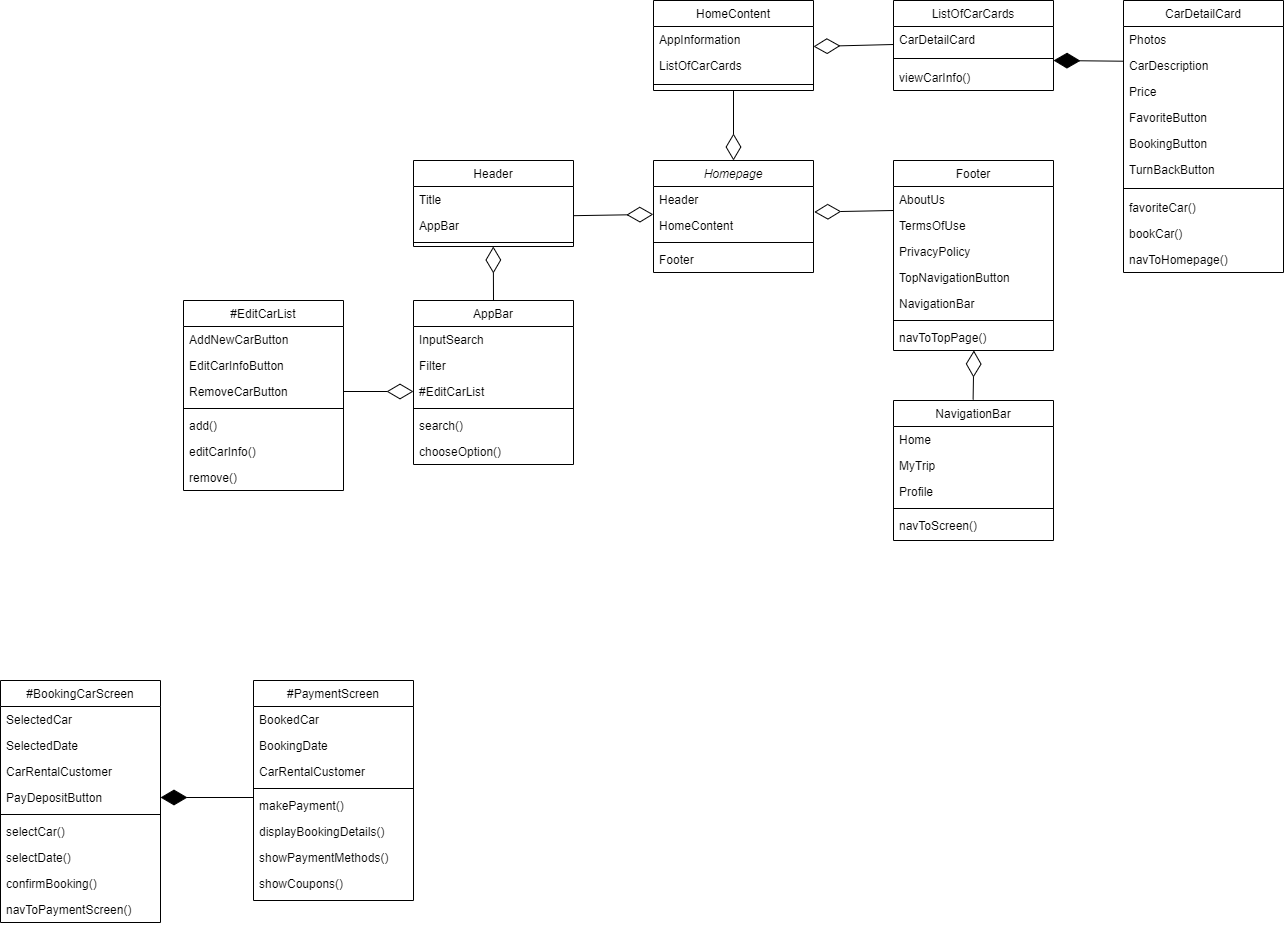


* SignUp: Component to register an account.
* LogIn: Component to log in to account.
* ForgetPassword: Component to change password when forgotten password.

### Homepage:

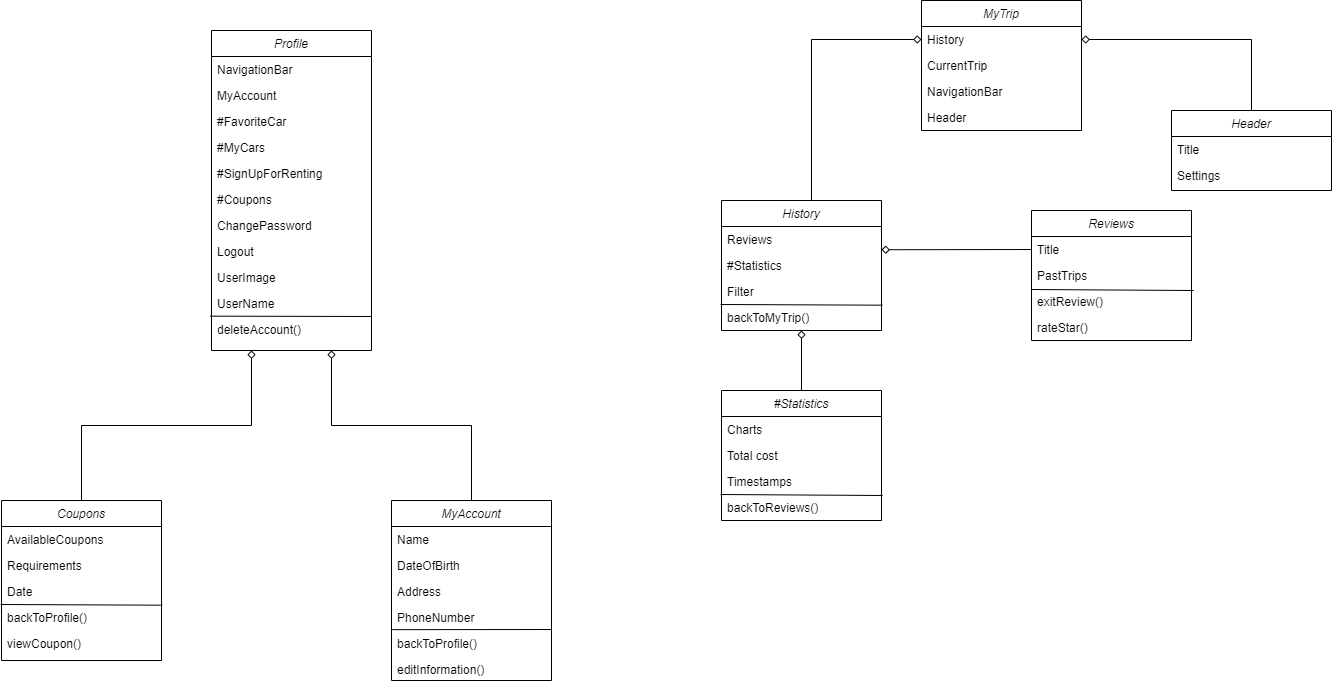
* Homepage: the main screen of the application.
* Home Content: displaying a list of cars available for rent.
* List of Car Cards: a list of cars available for rent, with each car card containing information such as car name, rental price, photo, etc.
* Car Detail Card: a detailed car card, displaying full information about a car, including car name, rental price, photo, description, etc.
* Header: the top part of the application, containing the title and App bar.
* App Bar: containing search bar and edit car button.
* Filter: allowing users to filter cars by car type, price, etc.
* Footer: the bottom part of the application, containing links about the application information.
* #Edit Car List: allowing car owners to add, edit, and delete cars.

### BookingCarScreen:



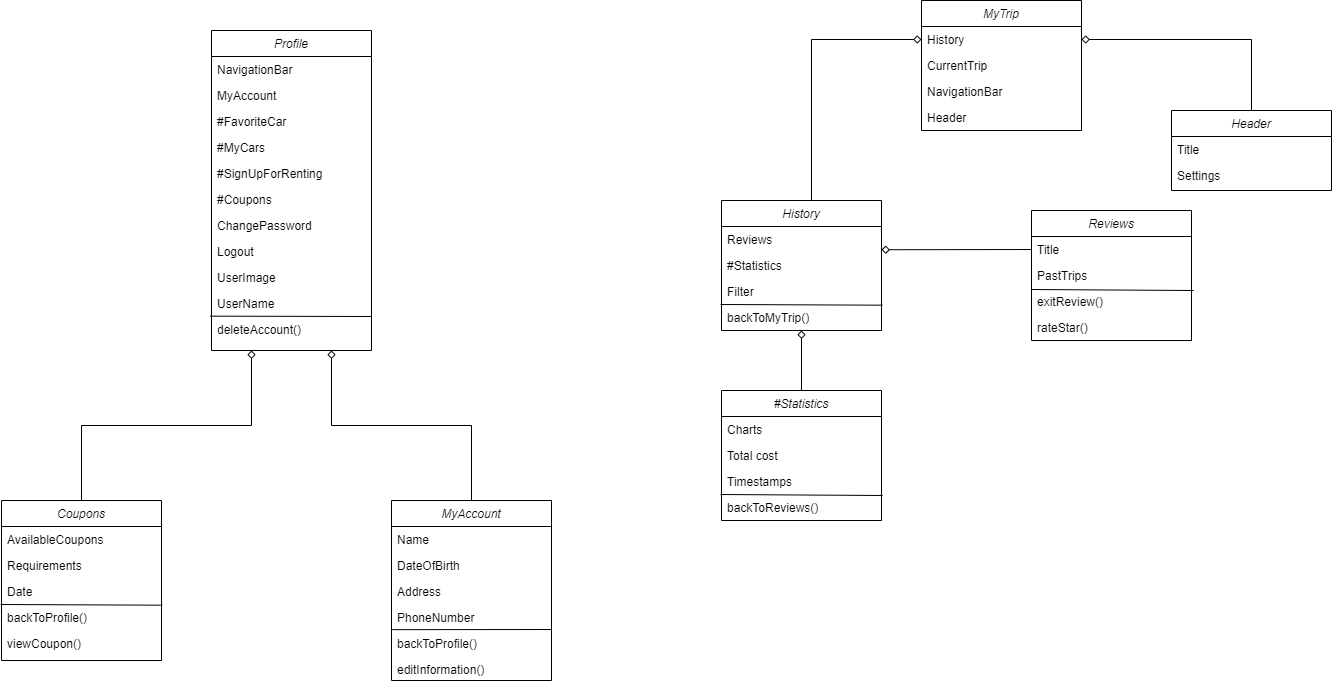
* BookingCarScreen: Screen where users click BookingButton.
* PaymentScreen: Screen where users can make payments for car rentals.

### MyTrip:



* MyTrip: component displays the current trip.
* History: component displays previous trips of the account.
* Reviews: component to rate past customer/ car’ owner.
* Statistics: component visualizes the data of all previous trips by charts for cars’ owners only.

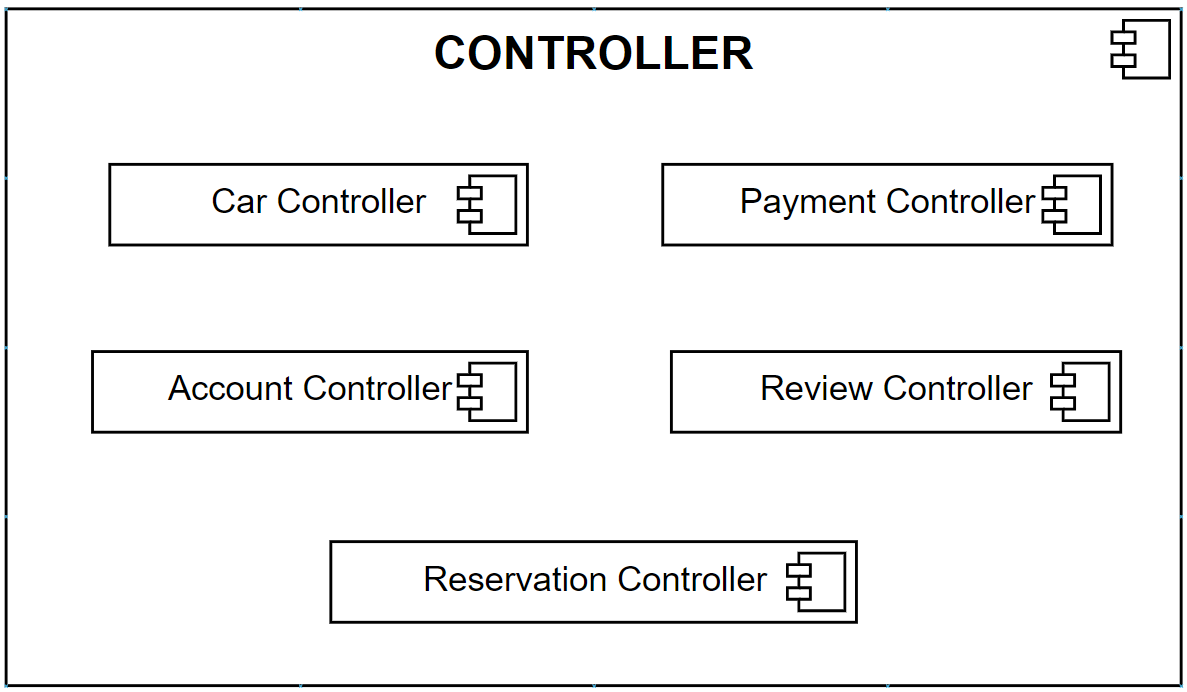
### Profile:

* Profile: component displays user’s information.
* #Coupons: component displays renters’ available coupons.
* MyAccount: component displays user’s detailed information.
* #FavoriteCar: component for renters’ favorite cars display only.
* #MyCars: component display cars for owners only.
* #SignUpForRenting: component for renters that want to put cars on rent.
* ChangePassword: component for users to change their current password.
* LogOut: component to log out of the account.

## Component: Controller(Front-end)

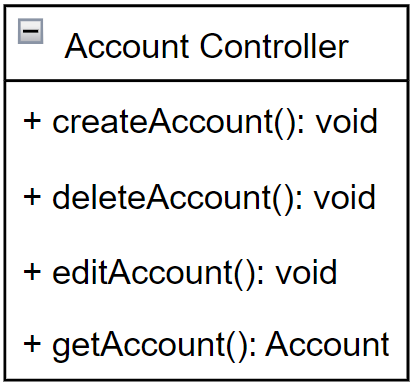
* BE Interaction: Services provide methods to make requests to the BE:
  + Car Controller
  + Payment Controller
  + Account Controller
  + Review Controller
* Application State Management: Application state is data stored in the application and used to change the user interface. Provider is a state management solution for Flutter apps. Provider uses a mechanism called InheritedWidget to share data between widgets.
  + Car state Model
  + Reservation state Model
  + Payment state Model
  + Account state Model
  + Review state Model
* Routing (distributing pages): Routing is a mechanism that allows users to switch between pages in the application. In the MVC model, routing is done through routes. Routes determine which pages will be displayed when users switch between pages. Navigator is the class used to convert Widgets in Flutter. The Navigator widget displays screens as a stack using the correct transition animations for the target platform. To navigate to a new screen, access the Navigator through the route’s BuildContext and call imperative methods such as push() or pop().

## Component: Controllers (Back-end)



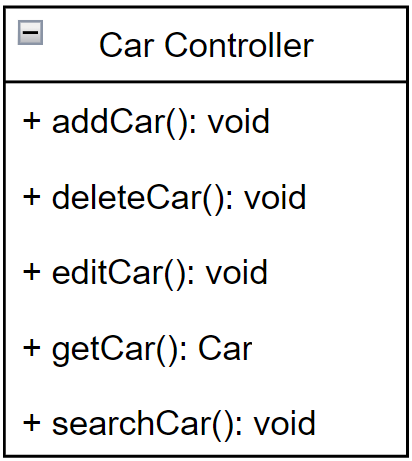
* **Describe**: Controllers get data from models to handle data and post to client when client requests and get data from client when they respond and handle data so post to models.

### 4.3.1. Account controller



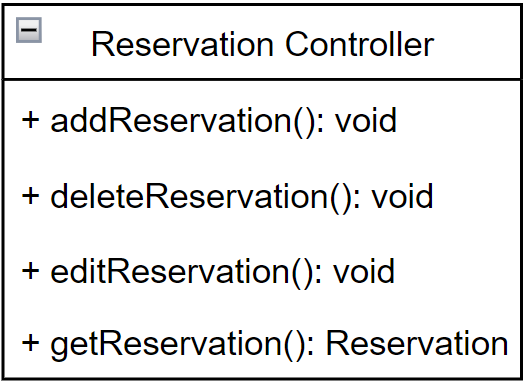
* **Describe**: The Account Controller has the effect of managing accounts in the system.
  + createAccount(): Create an account for new users.
  + deleteAccount(): Delete the user's account if the user no longer wants to use the app.
  + editAccount(): Edit account information including password and user information.
  + getAccount(): Get the registered user's account information.

### 4.3.2. Car controller



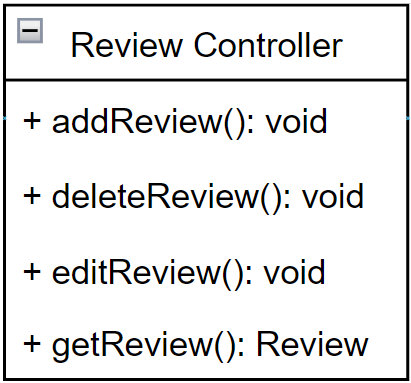
* **Describe**: The Car Controller has the effect of managing rented cars in the system.
  + addCar(): The car owner adds a new car to the list of cars he wants to rent.
  + deleteCar(): The car owner removes the existing car from the list of available rental cars.
  + editCar(): The car owner edits the vehicle's detailed information in the rental car list.
  + getCar(): get detailed information on the car so the user can decide to rent the car.
  + searchCar(): Search car information according to customer requests.

### 4.3.1. Reservation controller



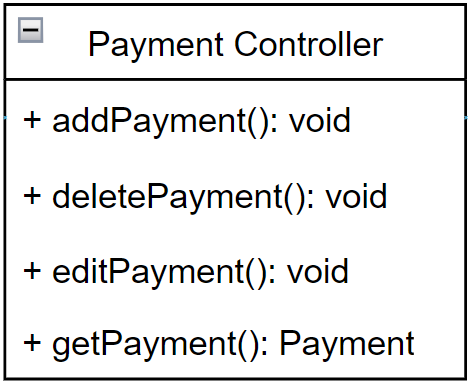
* **Describe**: The Reservation Controller has the effect of managing rental cars in the system.
  + addReservation(): add a new car rental trip after the user pays the deposits
  + deleteReservation(): delete a car rental trip if the user no longer wants to rent a car
  + editReservation(): Edit car rental trip information if the user returns the car
  + getReservation(): get car rental trip information to the user look at the reservation.

### 4.3.2. Review controller



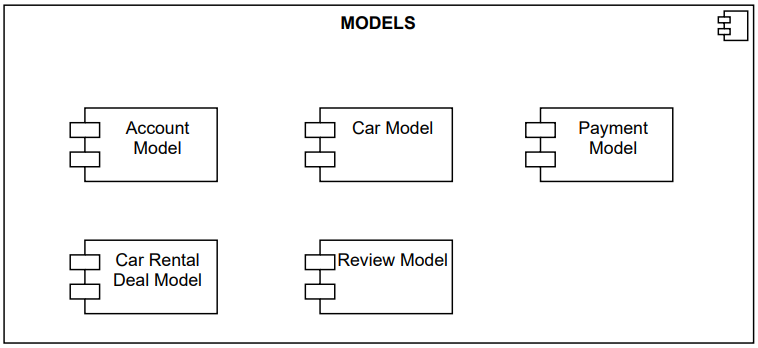
* **Description**: The Review Controller has the effect of managing feedback from car rental customers in the system
  + addReview():Add new user reviews after renting a car>
  + deleteReview(): delete reviews if users no longer want reviews.
  + editReview(): edit review if the user needs to edit the review content.
  + getReview():Get detailed information about the vehicle's review content or the user's review

### 4.3.1. Payment controller



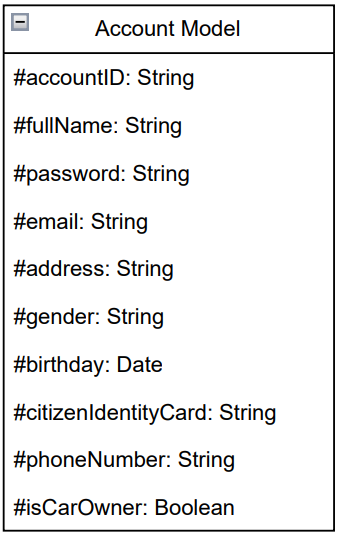
* **Description**: The Payment Controller has the effect of managing car rental payments of customers in the system.
  + addPayment(): Add new user payment invoice after confirming car rental.
  + deletePayment(); Delete the payment invoice if the user wants to cancel the car rental.
  + editPayment(): Edit invoice details if the user needs to edit content about the car rental.
  + getPayment(): Get detailed invoice information that the user has paid.

## Component: Models



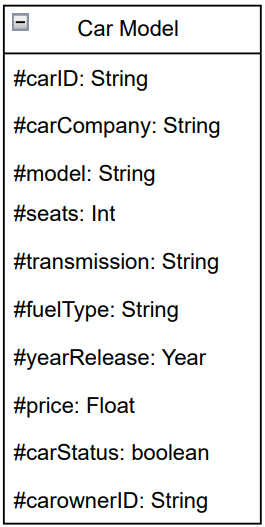
* **Description**: Models contain data for controllers to access and process. There are 5 main models, including Car model, Account model, Reservation model, Evaluation model, Payment model. Each model has its own features such as follows:

### Account Model:



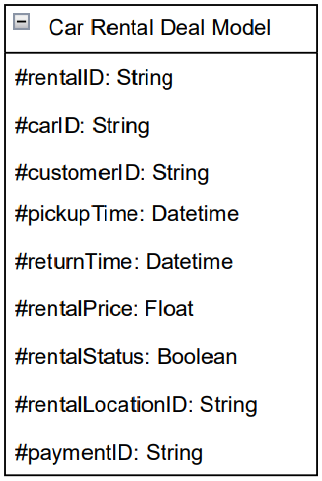
* **Description:** Account model contains account data such as accountID, fullName(full name), password, email, address, gender, birthday, citizenIdentifyCard, isCarOwner(if the account belongs to the car owner)

### Car Model:

****

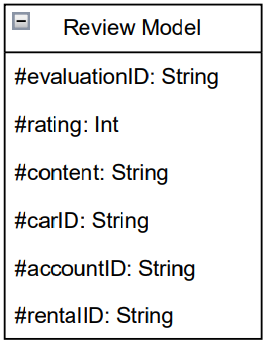
* **Description:** Car model contains car data such as carID, carCompany (brand of car), model, seats(the number of seats), transmission(manual transmission or automatic transmission), fuelType(type of fuel), yearRelease, price(rental price per day), carStatus(the status of the car has been rented or not), carOwnerID

### Car Rental Deal Model:

****

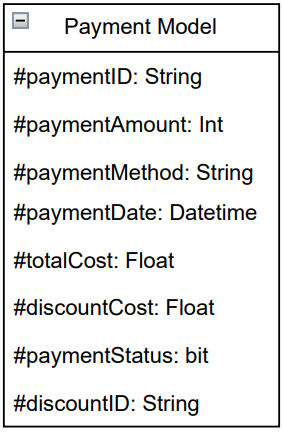
* **Description:** Car Rental Deal model contains reservation data such as rentalID, carID, customerID, pickupTime, returnTime, rental Price, rental Status(ongoing, completed or canceled), rentalLocation(), paymentID.

1. Review Model:

****

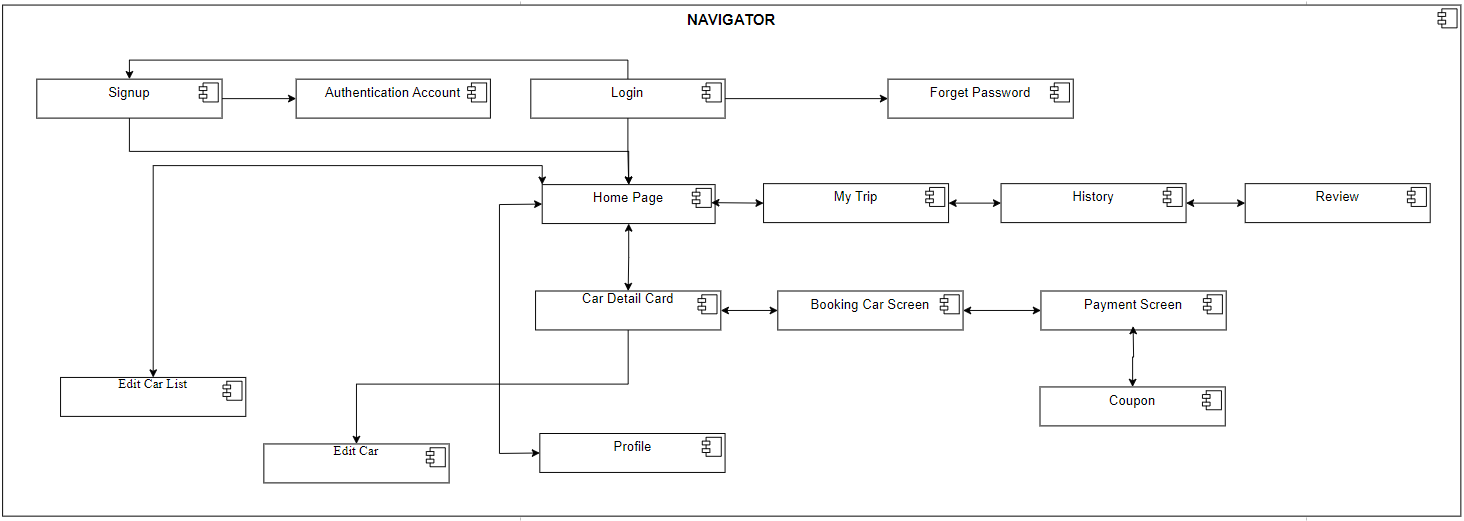
* **Description:** Review model contains evaluation data such as evaluationID, rating, content, carID, accountID, rentalID.

### Payment Model:

****

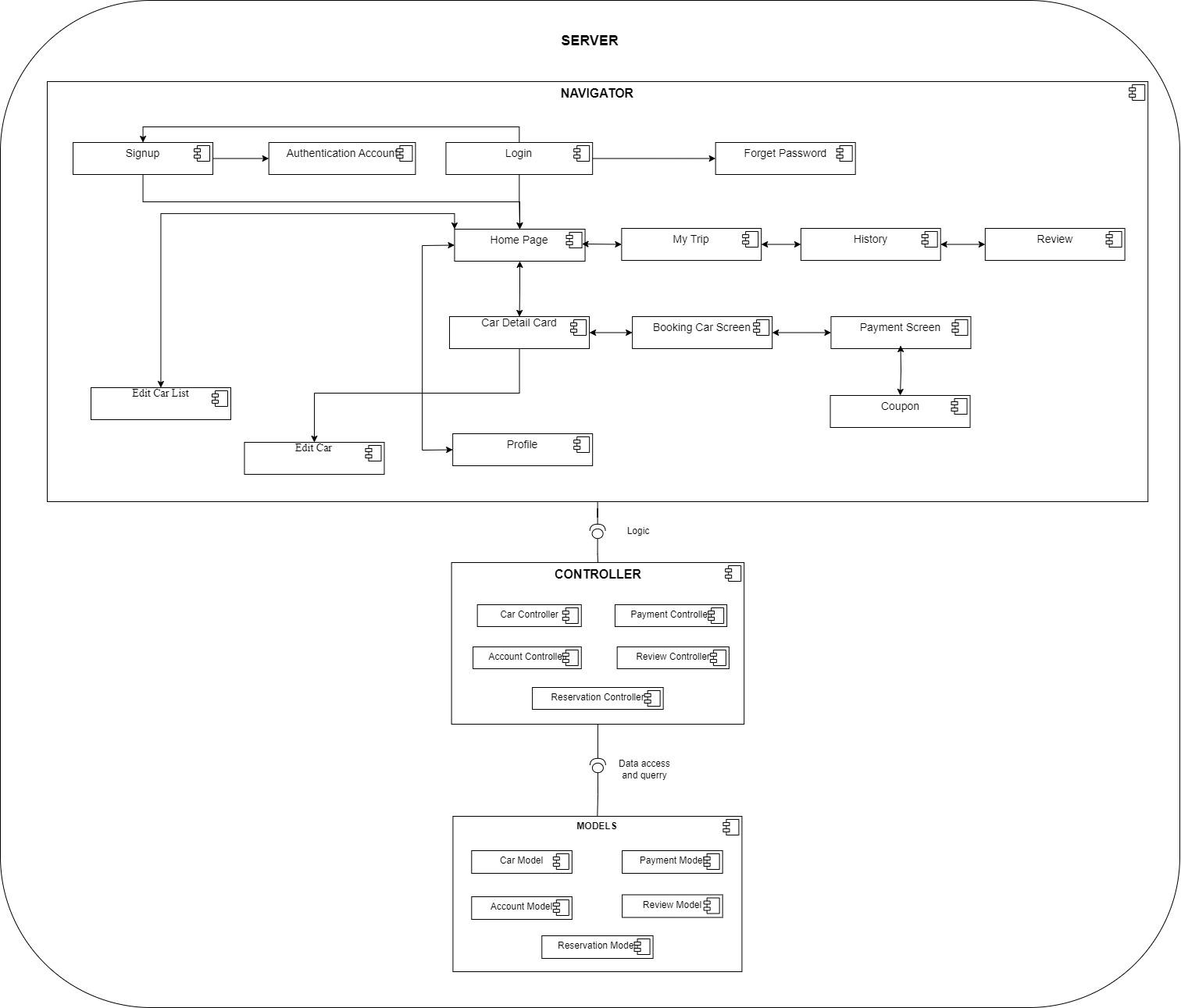
* **Description:** Payment model contains payment data such as paymentID, paymentAmount(the amount of money involved in the payment), paymentMethod, paymentDate, totalCost, discountCost(any discount applied to the total cost), paymentStatus(‘pending’, ’completed’ or ‘failed’), discountID(reference to the discount applied, if any).

## Component Navigator:



* **Describe**: Navigator helps support transitions on the UI

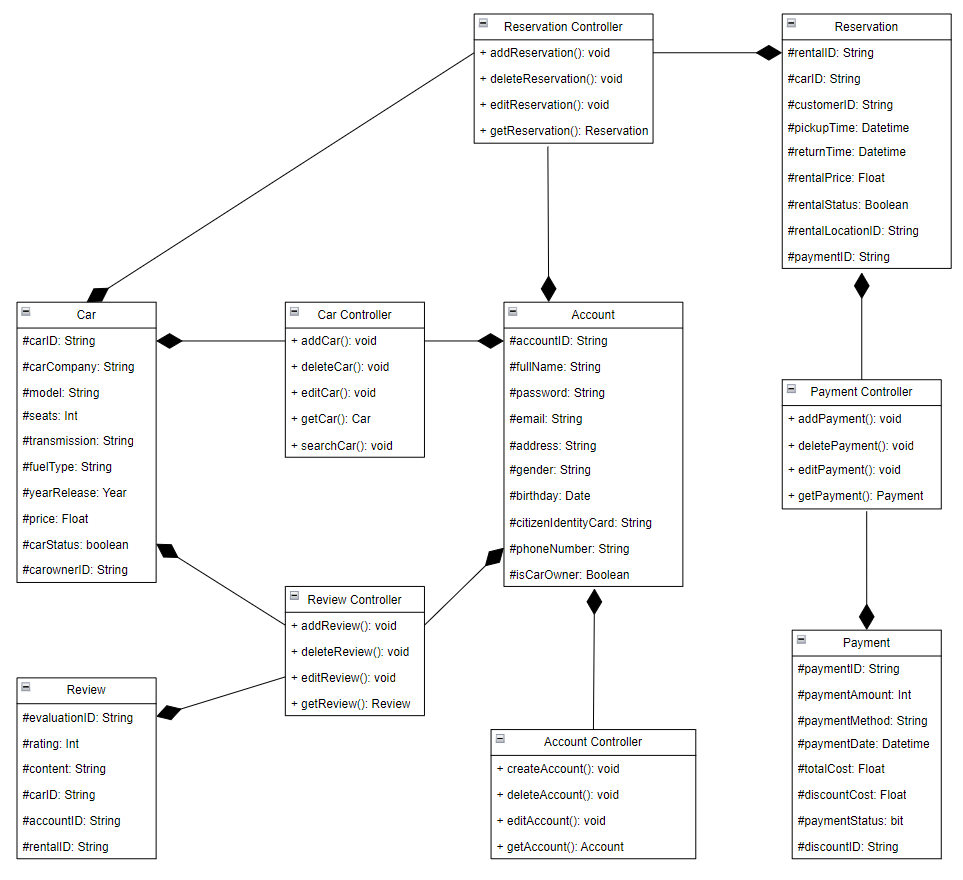
## Interaction between components in the server:



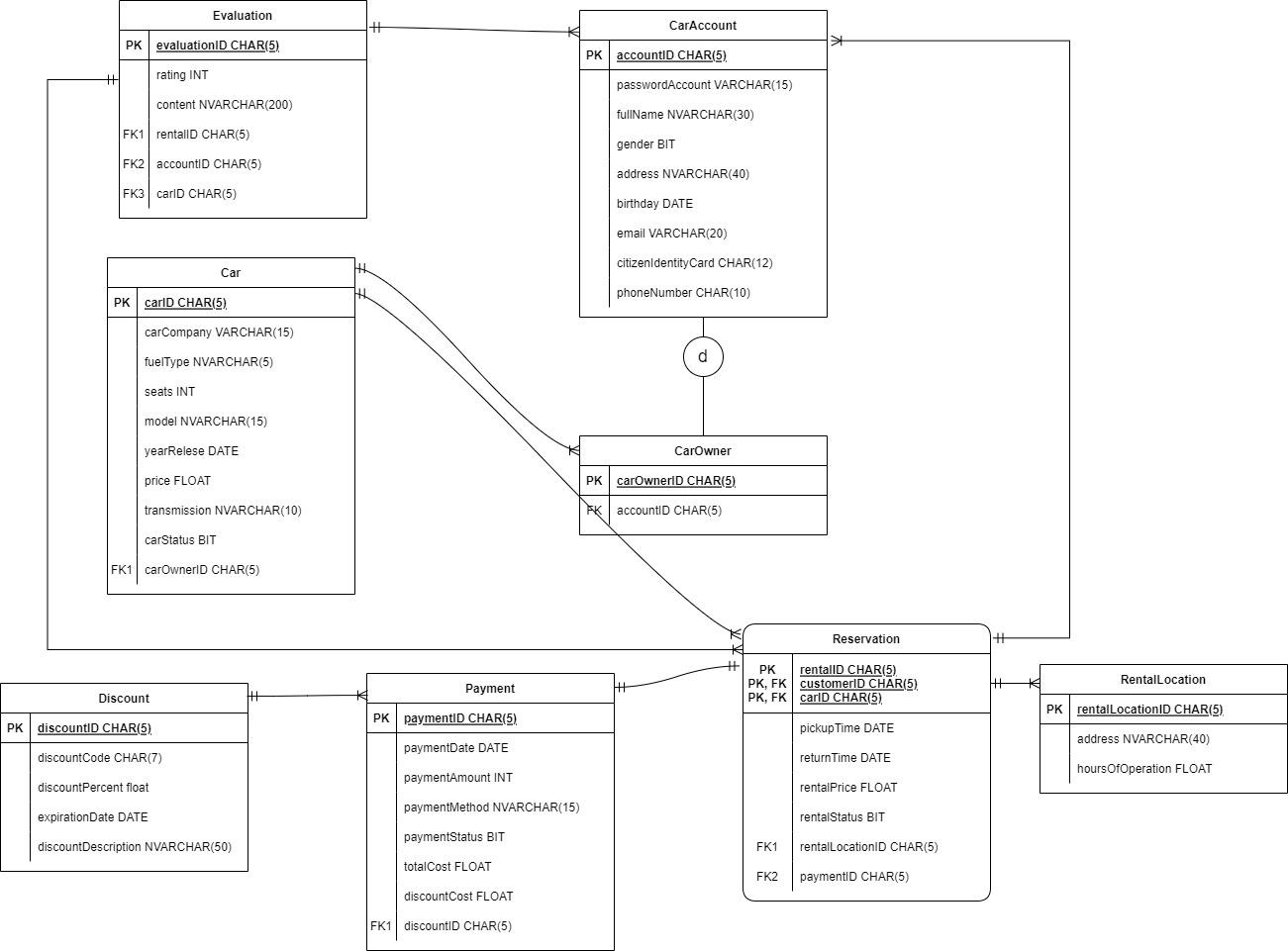
* **Describe**: The way to handle the server side is explained as follows: When a request is made from the client side, the provider will have a function to listen to the interaction to get data and assign it to a variable to the controller, it will go through the controller as an intermediate step. to process data requests. The corresponding controller will access and query its corresponding models to process data as requested by the user. After the controller finishes processing, it sends it back to the provider through the corresponding API for the client to render.

## Class diagram:

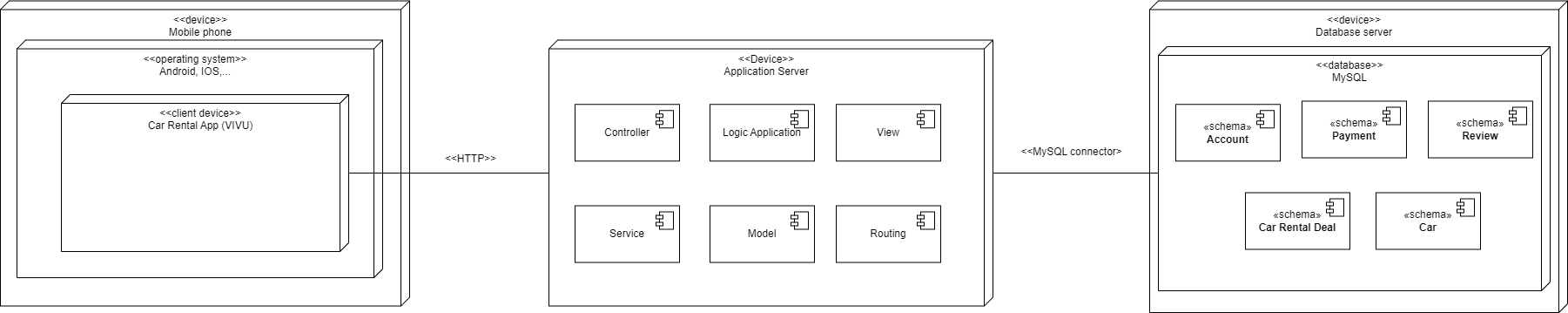
Class diagram shows relationships between modes and controllers.



## 4.8. Entities Relationship Diagram:



# Deployment Diagram:



Detailed description:

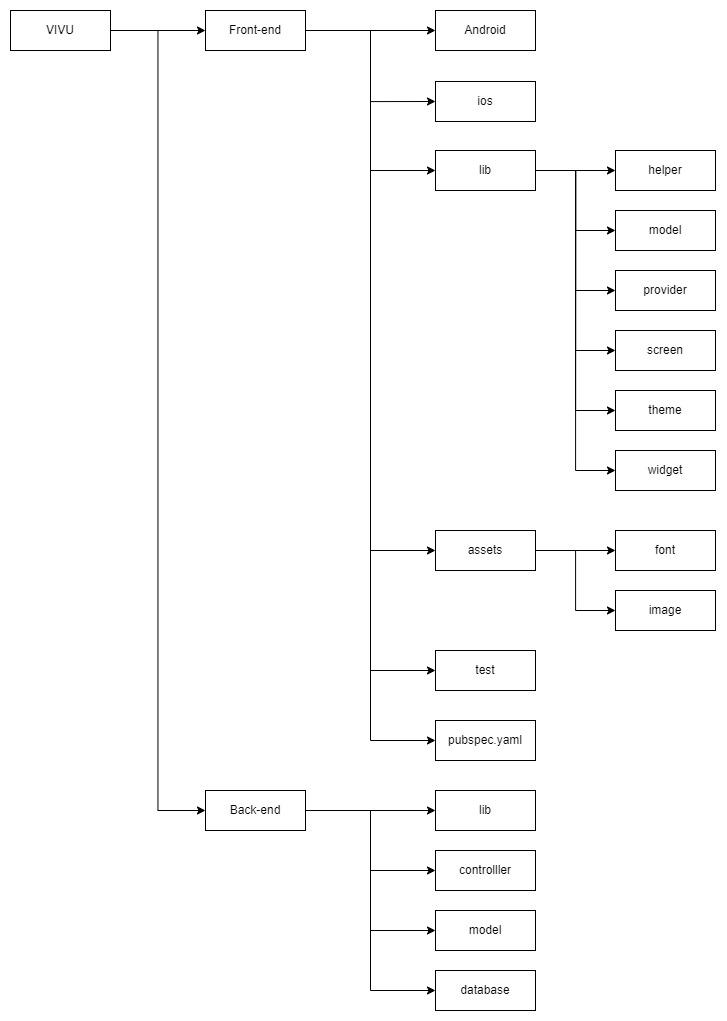
On the client side:

* Users can access the app on their mobile phones or tablets.
* Supported operating systems including Android and iOS.
* The app sends requests to the server-side components to: retrieve car data, process reservations, and user information,...
* The app receives responses from the server, displaying the requested information or confirming actions.
* The client-side components provide the user interface and handle user interactions.

On the server side:

* The VIVU app employs the Model-View-Controller architecture to structure its backend logic.
* Utilizes the Flask web framework, a lightweight and flexible framework for Python, to handle routing, request processing, and response generation.
* The web application has a Model that represents the data structure and relationships between data, often interacting with a database. Route redirecting the request to reach the desired resources.
* Views Render the data received from the controllers into consumable formats for the client-side app and determine the presentation of the data.
* Controller acts as the intermediary between the mode and view. It receives HTTP requests from the mobile app, retrieves necessary data from the model, processes the data according to business logic, prepares a response in a suitable format, and sends the response back to the mobile app for display to the user.
* Moreover, the web application communicates with the database through a MySQL connector to store, retrieve, and update data.
* For this project, we have chosen MySQL as our database management system due to its scalability and flexibility. Within our MySQL implementation, we've incorporated the following schema types:
  + Account schema: stores account information including id, username, email, password, …
  + Payment Schema: stores financial transactions, storing details like paymentID, paymentAmount(the amount of money involved in the payment), paymentMethod, paymentDate, totalCost, discountCost(any discount applied to the total cost), paymentStatus(‘pending’, ’completed’ or ‘failed’), discountID(reference to the discount applied, if any).
  + Car Schema: store details about cars, such as carID, carCompany (brand of car), model, seats(the number of seats), transmission(manual transmission or automatic transmission), fuelType(type of fuel), yearRelease, price(rental price per day), carStatus(the status of the car has been rented or not), carOwnerID
  + Car Rental Deal Schema: store reservations made by customers, including rentalID, carID, customerID, pickupTime, returnTime, rental Price, rental Status(ongoing, completed or canceled), rentalLocation(), paymentID.
  + Review Schema: store user review and ratings for the reservation, including evaluationID, rating, content, carID, accountID, rentalID.

# Folder’s structure:



* Description:
* android: crucial for building, running, and customizing your Flutter app specifically for Android devices.
* ios: critical for building, running, and customizing your Flutter app specifically for iOS platforms.
* helper: contains files that support connection to third-party services.
* model: Files in Flutter's model are often defined as classes. This class will contain properties and methods to represent the application's data and logic.
* provider: contains files that define providers. Provider is a state management tool for Flutter. It allows the sharing of data and states between widgets in the app efficiently.
* screen: contains files that define the user interface (UI) for the application. These files use Flutter widgets to create the user interface for the application
* Theme: contains files that define themes - a set of values used to style the user interface (UI) of your application.
* Widget: contains the widget definition files are stored. Widgets are a basic building block of the user interface (UI) in Flutter.
* font: contains font resources that the application uses to display to the user.
* image: contains image resources that the application uses to display to the user.
* test: placed for unit and widget tests.
* pubspec.yaml: correctly placed for dependency management.
* lib: content file to import libraries to use for backend controllers and models and main file that API uses.
* controller: contains files that define controllers which are a component of the Flask application responsible for handling requests from users and returning responses.
* model: define database model classes here, representing database tables.
* database: Stores user data, car information, car rental trip information, and user reviews as well as user payment when booking a car.