
The University of Hong Kong

Faculty of Engineering

Department of Computer Science

COMP7704
Dissertation Title
# A Smart Phone Application for Valet Parking

Submitted in partial fulfillment of the requirements for the admission to the degree of Master of Science in Computer Science

By
LUO Xianyang
3035237420

Dr. T.W. Chim
Date of submission: 30 / 11 / 2016

Abstract

Nowadays lots of people drive to hotels, clubs or restaurants after working. One of big issues for these people is to find a parking lot around the destination. And one of concerns for these customers is saving time and convenience. Although some places offer valet parking service, it is not that convenient in a big city like Hong Kong. Cause customers need to find the drop-off point, keep the parking ticket and can not recall their vehicles in advance.

To solve these problems, this project develops a smart phone application on iOS to help customers to use valet parking service. Users can create their cars in the application and place an order to park their cars. The application can act as a parking ticket for customers to redeem their cars, which is rather convenient for car owners. Using this application can save both customers and valets a lot of valued time.


**Declaration of Candidate**

I, the undersigned, hereby declare that the work contained in this thesis is my own original work, and has not previously in its entirely or in part been submitted at any university for a degree.

Only the source cited in the document has been used in this draft. Parts that are direct quotes or paraphrases are identified as such.

I agree that my work is published, in particular that the word is presented to third parties for inspection or copies of the work are made to pass on to third parties.

The University of Hong Kong,

LUO Xianyang

# Table of Contents

# Chapter 1. INTRODUCTION

## 1.1. Background Information

Nowadays a lot of people will go to clubs for dinners, after-work drinks or having fun with friends. Since Hong Kong is an International and fast-tempo city, it is rather common for citizens to go to places like Lan Kwai Fong after a day's work or at weekend. It is convenient for customers to drive their own cars to the hotels, clubs or bars. But as a matter of fact that Hong Kong is one of the most crowded cites in the world, it is not easy for drivers to find a parking lot quickly.

Valet parking service can help customers park their cars. It is offered by some restaurants, shopping malls, clubs and so on. A person called valet will drive a customer's car to parking lot when the customer arrives at the gate of the hotel and return the car when the customer leaves. The main advantage of valet parking is convenience. On one point, customers do not need to find a parking lot by themselves. On the other point, they do not have to walk a long way from the parking

lot to the hotel, which saves lots of time. All they need to do is just dropping their cars at drop-off point.

However, in such a fast-tempo city and such a high-tech era, some problems of traditional valet parking for a customer are:

1) must use a valet parking ticket to redeem their cars. If the ticket is lost, customer need to prove that the car belongs to him or her by showing driver license or identity card
2) may do not know where is the drop-off point for a certain hotel, restaurant or clubs and may take time to find it
3) do not know current status of the car

To solve these problems, I would like to develop a mobile application for people in Hong Kong to use valet parking service easier and more efficient.

## 1.2.Project Description

This project mainly focuses on helping drivers enjoy better valet parking service. A customer can register our service via its phone number. After adding a new car by inputting plate, brand and color, he or she can choose a parking lot and generate a QR code. When a valet

scans the QR code successfully, the order will be generated and the customer can use the phone as a valet ticket. Whenever the customer wants to get the car back, he or she can just click the “recall” button. Then our valets will return the car back to the customer.

This project has three parts and functions are as follows

1) iOS version application for customer
    a. allows a user to register, login and reset password via phone number
    b. allows a user to add a new car, edit an existing car and delete a car. Information like plate, brand and color are requested
    c. allows a user to generate order by choosing parking lot and car
    d. allows a user to get the car back by just clicking a button
    e. allows a user to view all the historical orders
2) iOS version application for valet
    a. allows a valet to login and reset password via phone number
    b. allows a valet to add an order by scanning the QR code generating by a user

c. allows a valet to view all the opening orders

d. allows a valet to end an order by just clicking a button

3) server and database

a. processes all the http request and sends a proper response back to phone

b. stores data safely of all users, valets, cars and orders sends notification to a user then an order has been generated and ended successfully

## 1.3.Project Objectives

Since the traditional valet parking service has matured, so the app need to be more attractive to gain users. We have to follow the current trend of design, follow the guideline of user interface design and take some of them into consideration to fulfill the goal, which is Shneiderman's Golden Rules of Interface Design[1], Jun Gong's Guideline for Mobile Application[2] and Nurul's Threes Layers Design Guideline for Mobile Application[3].

Apart from user interface, the app should be rather easy to use. Users do not to do lots of setting or follow a complicate guideline to generate an order. There are some mobile applications in the market with

similar purpose like Meibo, Youbo and so on. After analyzing those applications, I found out that they were not that easy to use. Since there are a new technology in iOS called 3D touch which brings a new powerful dimension to Multi-Touch interface, users can enjoy the best convenience while parking their cars. Also, it is rather innovative if there is an Apple Watch application cooperating with application on the iPhone.

The final goal of this project is to change the traditional valet parking service by attractive and innovative features so our objectives can be conducted in three aspects:

1) attractability: to attract users, the application should have friendly and beautiful user interface, reasonable layout and overall clean look.
2) innovation: users are more willing to use the application by using new technology like 3D touch. This project allows user to enter the “parking now” and “current orders” views from the icon which makes it rather convenient and enjoyable.
3) connectivity: a user may enjoy more convenience if he or she has an Apple Watch. The all the parking and recall request can be

done through Apple Watch. A user do not even need to take his or her phone out the pocket.

## 1.4.Summary of Chapters

Chapter 1 firstly introduces the background and motivation of this project and then gives brief introduction and objectives of this project

Chapter 2 firstly introduces two similar mobile applications in the market and analyze the advantages and disadvantages. The first one is Luxe operating in the USA and the second one is Youbo operating in mainland China.

Chapter 3 talks about the background knowledge of this project including third party frameworks used in mobile clients and server.

Chapter 4 introduces the detail design for this project. It will firstly give the system requirements then introduces the whole design for this project including the system architecture, the system work flows, use case design, UI design and database design.

Chapter 5 talks about the implementation for this project including server side and two mobile clients. For mobile clients of customers, it will introduce some major modules and then comes with the implementation of user interface.

Chapter 6 concludes the whole project and discusses about the future work.

# Chapter 2. RELATED WORKS

To develop a successful application, we need to refer to the existing applications and see if how we can do it better.

In this part, I will introduce two mobile applications offering valet parking service in the market. The first one is Luxe, which is available in American and then Youbo in China. I will firstly give a brief introduction and analyze the advantages and disadvantages of both applications respectively.

## 2.1. Luxe

http://www.luxe.com/

Luxe is a valet parking app available on both iOS and android. It is currently available in San Francisco, New York, Chicago, Seattle, Austin and Log Angeles. Figure 2-1 shows demo pictures for Luxe.

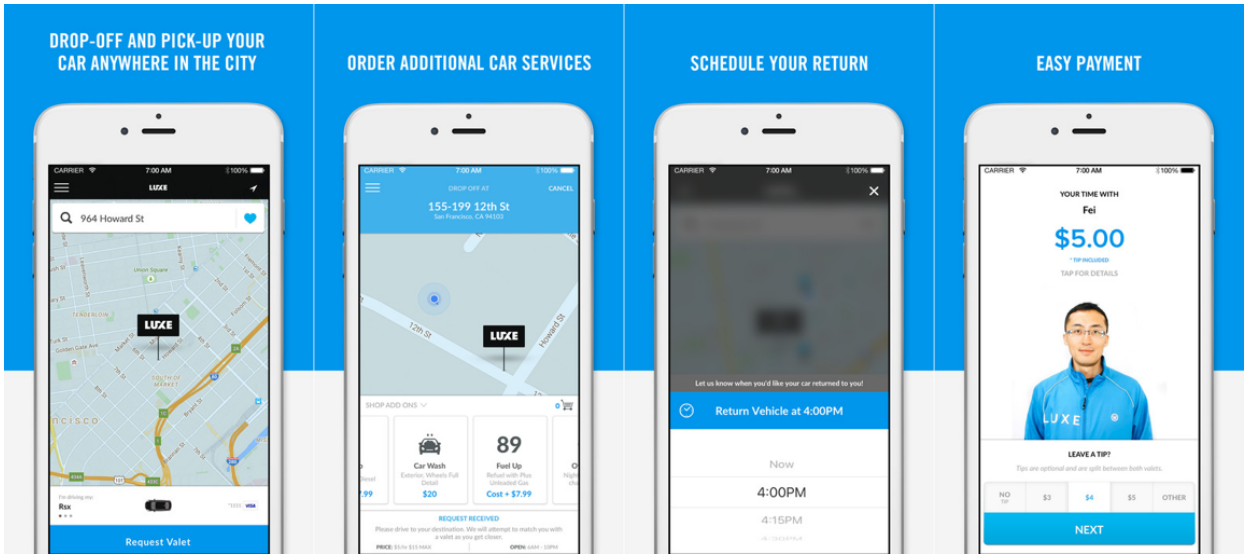


Figure 2-1 iPhone Screenshots for Luxe

After trying Luxe a few days, I found the advantages and disadvantages as below:

Advanages:

1) the user can drop off and gets returned anywhere and a valet will wait at the drop point. This is the best experience for a valet parking service. The user saves time since he or she does not need to find a certain drop-off point
2) the user can pay for the service using its phone, which makes it a Uber-like application. All the process can be done by the application.
3) the user interface is attractive and modern

Disadvantages:

1) the application does not use new technology like 3D touch and does not offer Apple Watch application
2) the register process is rather complicated, which requires both email account and phone number
3) all those service are only available in the USA

## 2.2. Youbo

http://www.uboche.com/

Youbo is a valet parking app available on both iOS and android. It is currently available in Beijing, Shanghai, Chengdu and other big cities in mainland China. Figure 2-2 shows demo pictures for Youbo


Figure 2-2 iPhone Screenshots for Youbo

Youbo offers similar services as for Luxe. After taking a real experience, I found the advantages and disadvantages below:

Advantages:

1) a valet can drive the customer to the destination and then park the car. And the customer can get the car returned at wherever he or she wants. This is rather similar to Luxe
2) a valet could help wash and refuel a customer's car
3) a user can register and login just using its phone number

Disadvantages

1) the application does not use new technology like 3D touch and does not offer Apple Watch application
2) the user interface does not look good.

# Chapter 3. BACKGROUND KNOWLEDGE

There are a lot of third party frameworks to enhance the performance or to optimize the user interface of this project. This chapter mainly introduces some useful frameworks in the development. The first part talks about tools used in front-end which is the mobile application. And the second part talks about tools in back-end which is the server and database.

## 3.1. Tools used in Front-End

### 3.1.1. AFNetworking

AFNetworking is networking library used in iOS and Mac OS X development. It is built on top of the Foundation URL Loading System, extending the powerful high-level networking abstractions built into Cocoa. It is a high efficient networking module along with feature rich API which is rather easy to use. It powers some of most popular applications on iPhone and iPad.

The usage of AFNetworking is simple, which differs from the origin method offered in iOS. After initializing a session manager, the user just needs to configure some parameters, sends the request and then waits for the response. A simple post request can be implemented below:

```
AFURLSessionManager *manager = [AFURLSessionManager manager];
NSDictionary *parameters = @{@"foo": @"bar"};
[manager POST:@"http://example.com/resources.json"
success:^(AFHTTPRequestOperation *operation, id responseObject) {
parameters:parameters
    NSLog(@"JSON: %@", responseObject);
} failure:^(AFHTTPRequestOperation *operation, NSError *error) {
    NSLog(@"Error: %@", error);
}];
```

This code block sends an asynchronous request containing a parameter dictionary to the server. If the request is successful, the manager will get a *responseObject* containing request information or an *error* if the request is failed.

### 3.1.2. MBProgressHUD

MBProgressHUD is an iOS drop-in class that displays a translucent HUD with an indicator and/or labels while work is being done in a background thread. I use this framework in almost every view in this

project. It's easy to add a text indicator or progress indicator as shown in Figure 3-1

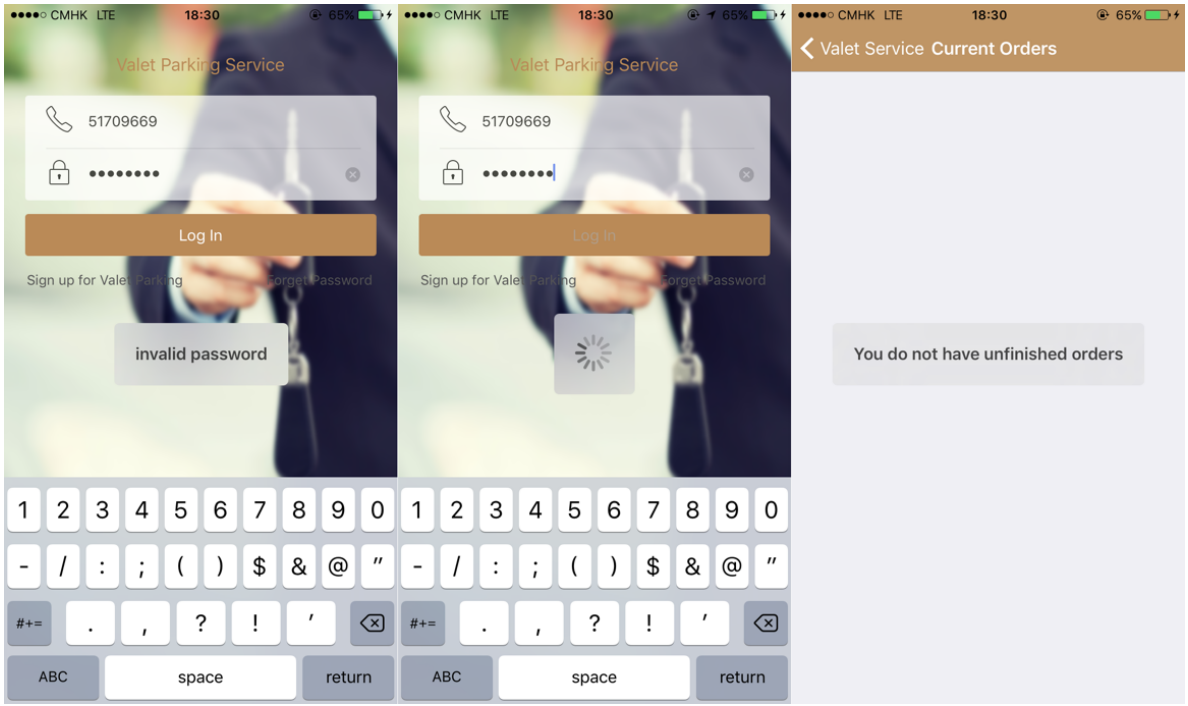


Figure 3-1 Screenshots for usage of MBProgressHUD

The usage MBProgressHUD is very simple by initializing an MBProgressHUD instance and show it in current view. By choosing the mode of a hud, it can display to show text or process indicator or both. The code block below shows how to add a hud to current view and hide until a certain process is finished.

```
[MBProgressHUD showHUDAddedTo:self.view animated:YES];
dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, 0.01 *
NSEC_PER_SEC);
dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
   // Do something...
   [MBProgressHUD hideHUDForView:self.view animated:YES];
});
```

### 3.1.3. JSONModel

MVC (Model, View, Controller) is widely used in object-oriented developing. A model is used to store and manage data. And the format of most data downloaded from server is JSON. In order to use this data, front-end needs to transfer it to model. JSONModel allows rapid creation of smart data models. By creating an model and inheriting it from JSONModel, the front-end can use the *initWithDictionary* method to generate instance of a model rapidly without reading each key-value from response data. The code block below shows an statement of class named *CountryModel* which is inherited from JSONModel. It has three attributes. As long as the JSON data has the same key, the application can generate a model using one line of code.

```
#import "JSONModel.h"
@interface CountryModel : JSONModel
@property (assign, nonatomic) int id;
@property (strong, nonatomic) NSString* country;
@property (assign, nonatomic) BOOL isInEurope;
@end
```

### 3.1.4. SMS_SDK

SMS_SDK is a framework used to send verification code to users' mobile phones. It's totally free and easy to be integrated in the application. Since our valets host customers' cars, it is important for valets to verify the identity of the user. The code block below shows how to use SMS_SDK to verify a user's phone number.

```
[SMSSDK getVerificationCodeByMethod:SMSGetCodeMethodSMS
                        phoneNumber:@"51709669"
                               zone:@"852"
                   customIdentifier:nil
                             result:nil];
```

### 3.1.5. GoogleMaps

GoogleMaps is a map API offered by google. It is very easy to use and offers a lot of functions. It can be installed to a project using Cocoapods. To use this service, I need to get an App key from its official website and add the key to my application.

The code block shows how to use GoogleMaps to add a map view and a marker to application using Objective-C.

```
GMSCameraPosition *camera = [GMSCameraPosition
cameraWithLatitude:_latitude longitude:_longitude zoom:14];
_mapView = [GMSMapView mapWithFrame:CGRectZero camera:camera];
GMSMarker *marker = [[GMSMarker alloc] init];
marker.position = camera.target;
marker.snippet = _title;
marker.appearAnimation = kGMSMarkerAnimationPop;
marker.map = _mapView;
self.view = _mapView;
```

## 3.2. Tools used in Back-End

### 3.2.1. MongoDB

MongoDB is a free and open-source cross-platform document-oriented database program. Being different from database like MySQL, MongoDB is classified as NoSQL database program and uses JSON-like documents with schemas. It works well with Node.js and can implement common database method like create, read, update and delete rather easily and efficiently.

Compared to MySQL, MongoDB has some main advantages as follows:

1) rich data model, dynamic schema, typed data and data locality
2) rich query language which is easier to code and read
3) development is simplified as MongoDB documents map naturally to modern, object-oriented programming languages

### 3.2.2. Express

Express is a fast, unopinionated and minimalist web framework that provides a robust set of features for web and mobile applications. It is the most popular framework in Node.js as for now. Because it provides lots of powerful and useful features including organizing application's routing and providing template solutions, the user can focus on

developing the functions of applications. The main advantages of express are as follows:

1) event driven: this feature enables registering various profound functionalities by connecting them to other events that are being executed once the event is triggered.
2) javascript closures: this feature allow the user to use variables that are defined in the outer calling function inside the callback body, which is extremely useful to solve the conflict while working with various applications.

# Chapter 4. REQUIREMENTS AND DESIGN

In this part, I will introduce the requirements and design for this project. Firstly, I will talk about the requirements analysis including product perspective, user perspective and functional requirements. Then comes with design for this project including system architecture, system workflow, object oriented design, user interface design and database design.

## 4.1. Requirements Analysis

### 4.1.1. Product Perspective

This project has front-end and back-end. The front-end is built on Xcode 8 using Objecive-C and the back-end is built with Node.js and MongoDB. And there are two applications in the front-end. One is for customers and the other one is for valets. Both customers and valets use their phone number as login account.

### 4.1.2. User perspective

There are two types of user – customer and valet. After a customer logging in, he or she can add a car, choose a place to park, check orders and request his or her car. After a valet logging in, he or she can access to all the opening orders, generate an order by scan the QR code on a customer's phone and end an order after returning a customer's car.

### 4.1.3. Functional Requirements

Using this product, a customer will be able to choose a place and choose a car to park. Here is the list of high-level functional requirements that this project is focus on. The two apps and the back-end must:

1) Authenticate and authorize a customer or a valet
2) Store a number of data records describing cars. Each record will have attributes as below:
    - Car plate
    - Car brand
    - Car color
3) Store a number of data records describing orders. Each record will have attributes below:
    - Order date and time (create, update, end)

- Order place
- Customer name
- Customer phone
- Car plate

4) Allow customers to add, edit a car by typing in plate, brand and color. Also allow customers to delete a car.
5) Allow customers to choose a place and a car to park
6) Allow customers to check all current orders and historical orders
7) Allow customers to request his or her car back
8) Allow valets to generate an order by scanning QR code
9) Allow valets to access all the opening orders
10) Allow valets to end an order

Next I will introduce the details of design.

## 4.2. Design

### 4.2.1. System Architecture

The entire system can be divided into two parts: front-end and back-end. Back-end is a server based on Node.js with MongoDB as database. All the *get* and *post* requests including registering an account, logging in, adding a car, editing a car, deleting a car, creating an order, updating an order are handled by this server. It will check every request for correctness and validity and send corresponding response back. And front-end are two mobile applications built using Xcode and Objective-C. The database used in front-end is CoreData which is native in iOS and easy to use. Figure 4-1 shows the architecture of the system

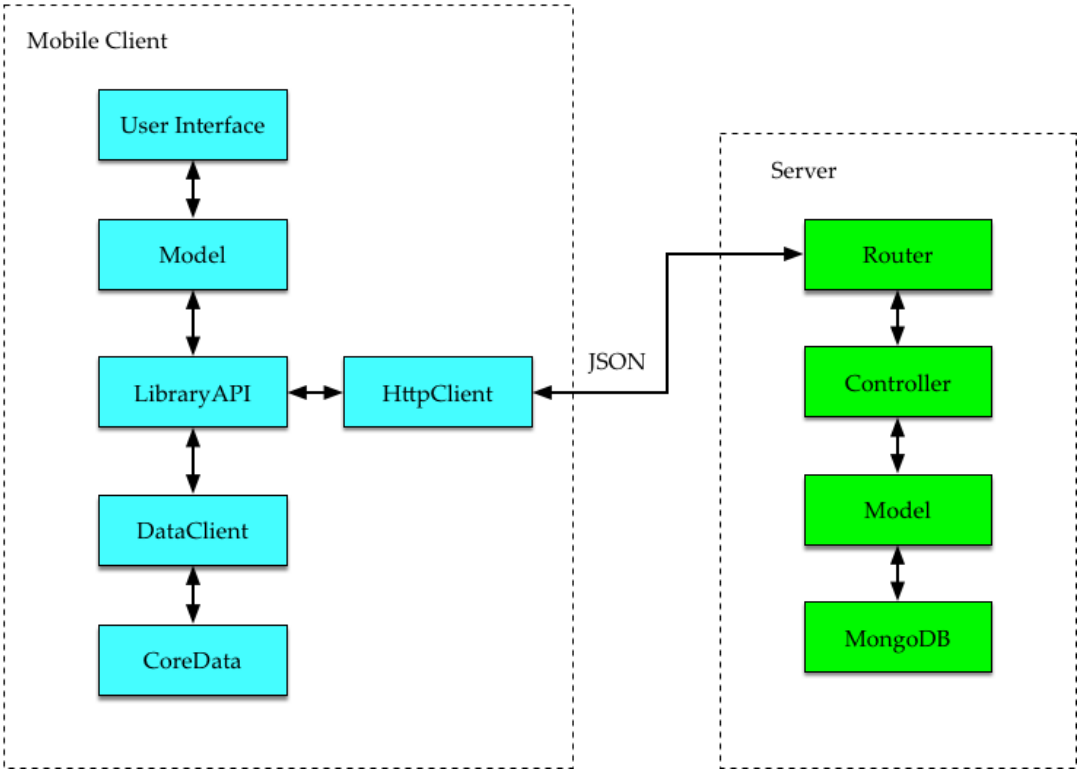


Figure 4-1 Architecture of System

### 4.2.2. System Workflow

The system workflow consists of two parts. The first part is the work flow in the server and the second part is the work flow in the mobile client. These two parts will be introduced separately.

#### 4.2.2.1. Workflows in Server

Server works as back-end in this project and will handle all the http request sent from front-end. There are three major management in server: account management, car management and order management.

1) Flow chart for account management

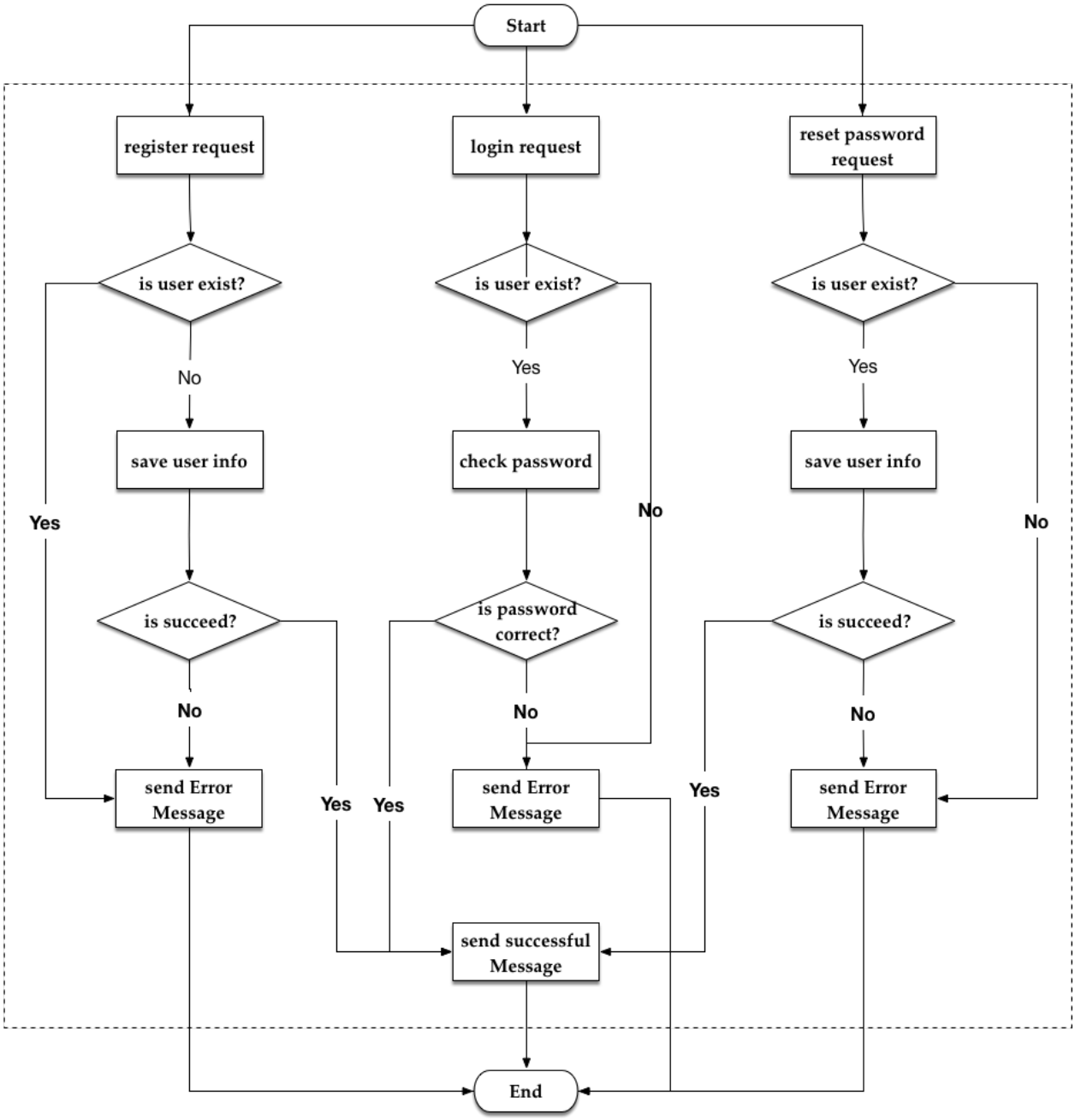


Figure 4-2 flow chart for account management

As shown in figure 4-2, server needs to deal with three kinds of account management request: register request, login request and reset password request. For register request, if there exist an account with the same phone number or fail to save user file, the

server will send error message. For login request, if it does not find the user with the phone number or the password is incorrect, the server will send error message. For reset password request, if it does not find the user with the phone number or fail to save new password, the server will send error message. For other circumstance, the server will send successful message with the user model.

2) Flow chart for car management

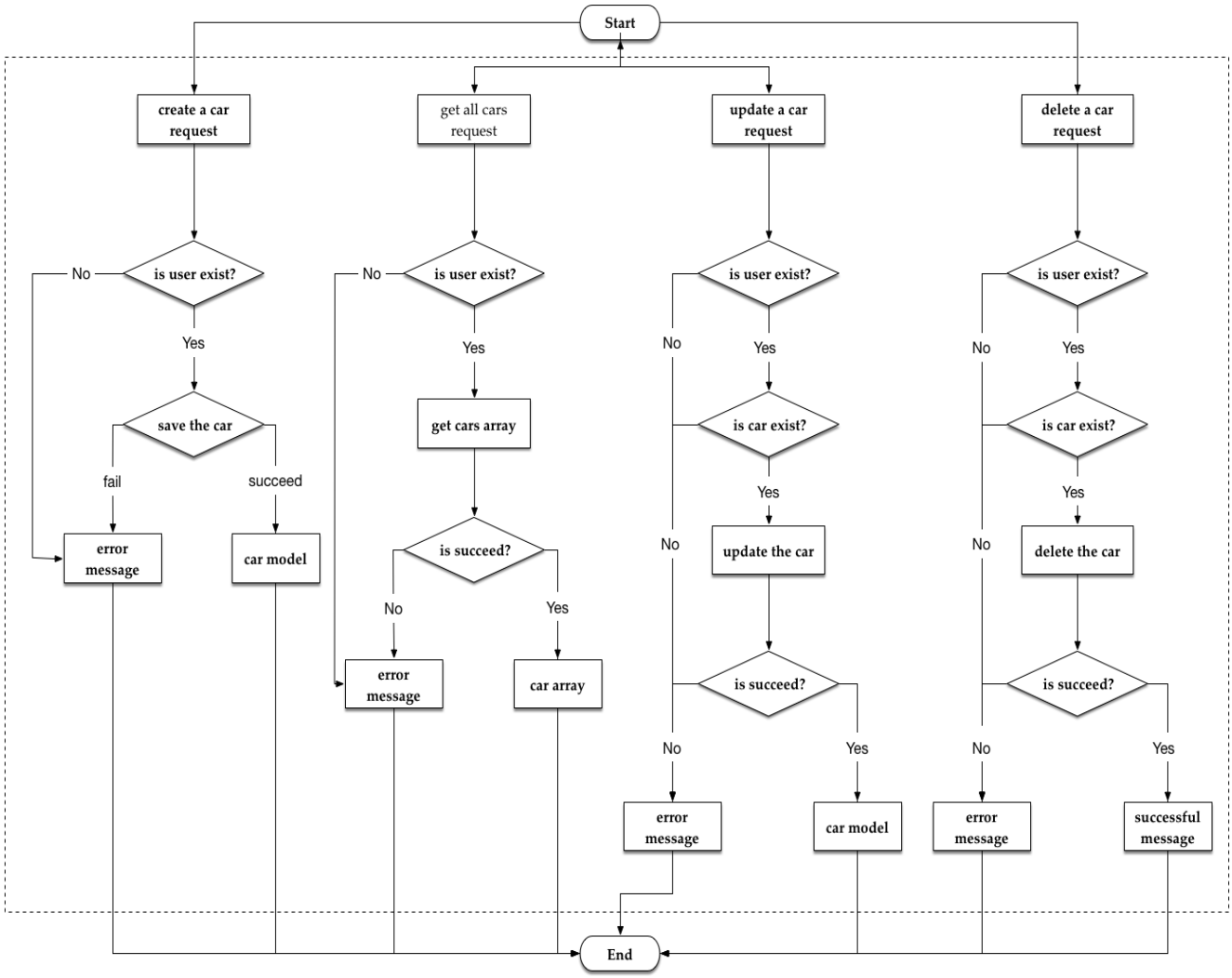


Figure 4-3 flow chart for car management

As shown in figure 4-3, the server needs to deal with four kinds of car management request: create, update, delete a car and get all cars for a user. For all these four requests, server will firstly check if the user exists. If user's account can not be found, an error message will be sent to front-end. If the user's account is valid, for "create a car request, server will save this car to database. For "get all cars

request", server will get all the car models with corresponding user account from database. For "update a car request" and "delete a car request", server will check if the car exists. If the car can not be found, an error message will be sent to front-end. If the car is valid, then corresponding database method will be called. If there is any error when processing the data in the database, an error message will be sent to the front-end.

3) Flow chart for order management

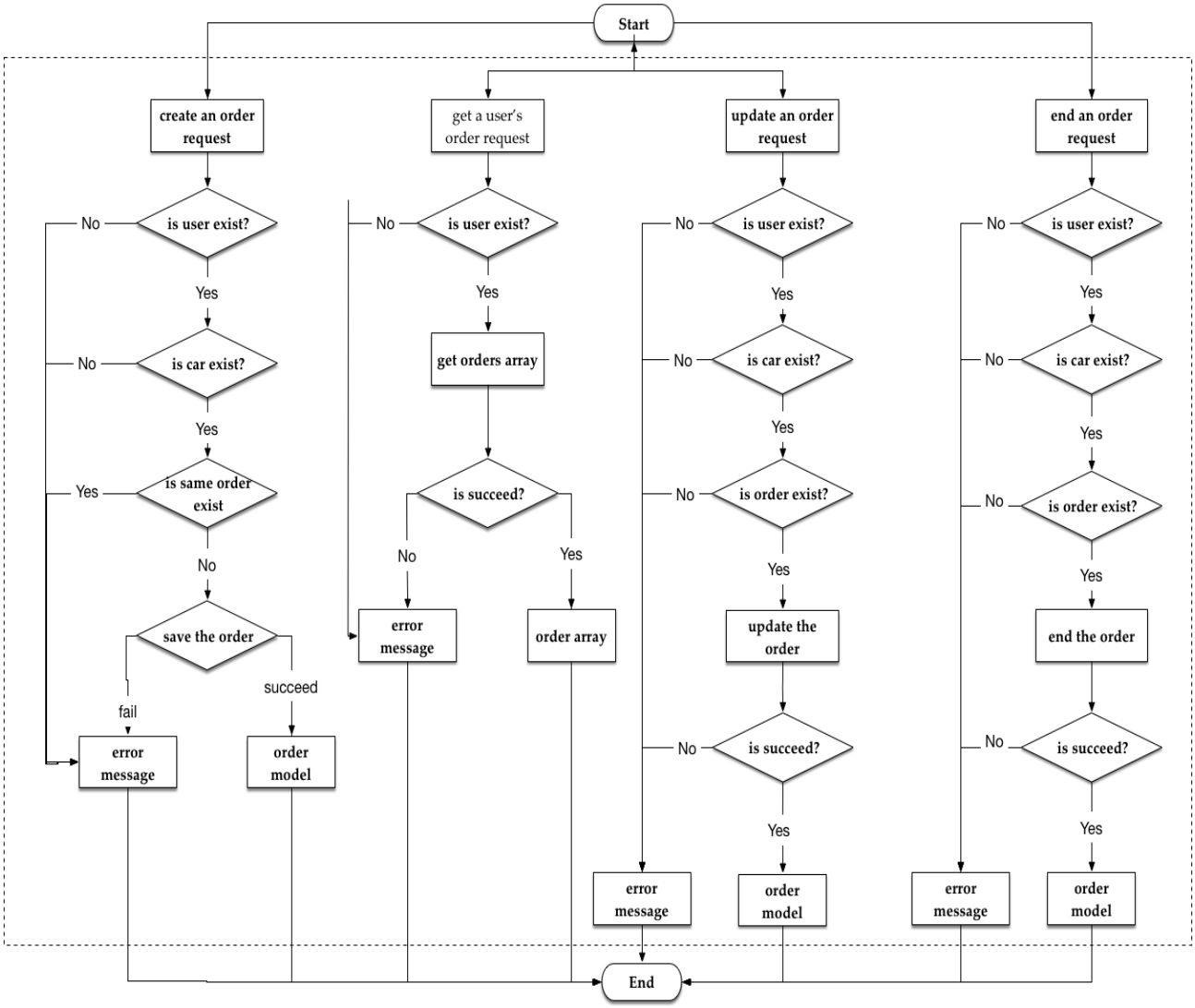


Figure 4-4 flow chart for order management

As shown in figure 4-4, the server needs to deal with three kinds of order management request: create, update, read and end. For all these four requests, server will firstly check if the user exists. If user's account can not be found, an error message will be sent to front-end. If the user's account is valid, for "create an order request", server will check if the car exist and if there is an order

with the same car unfinished. For “get a user’s order request”, server will get all the orders belong to the user and send them back. For “update an order request” and “end an order request”, server will check if the car and the order exist. If not, an error message will be sent back. If the order is valid, server will update or end that order and send the order model back.

#### 4.2.2.2. Workflows in Mobile Client

There are two major managements in front-end. One is account management and the other one is car and order management. They are less complex than that in server. Figure 4-5 shows the flow chart for mobile client.

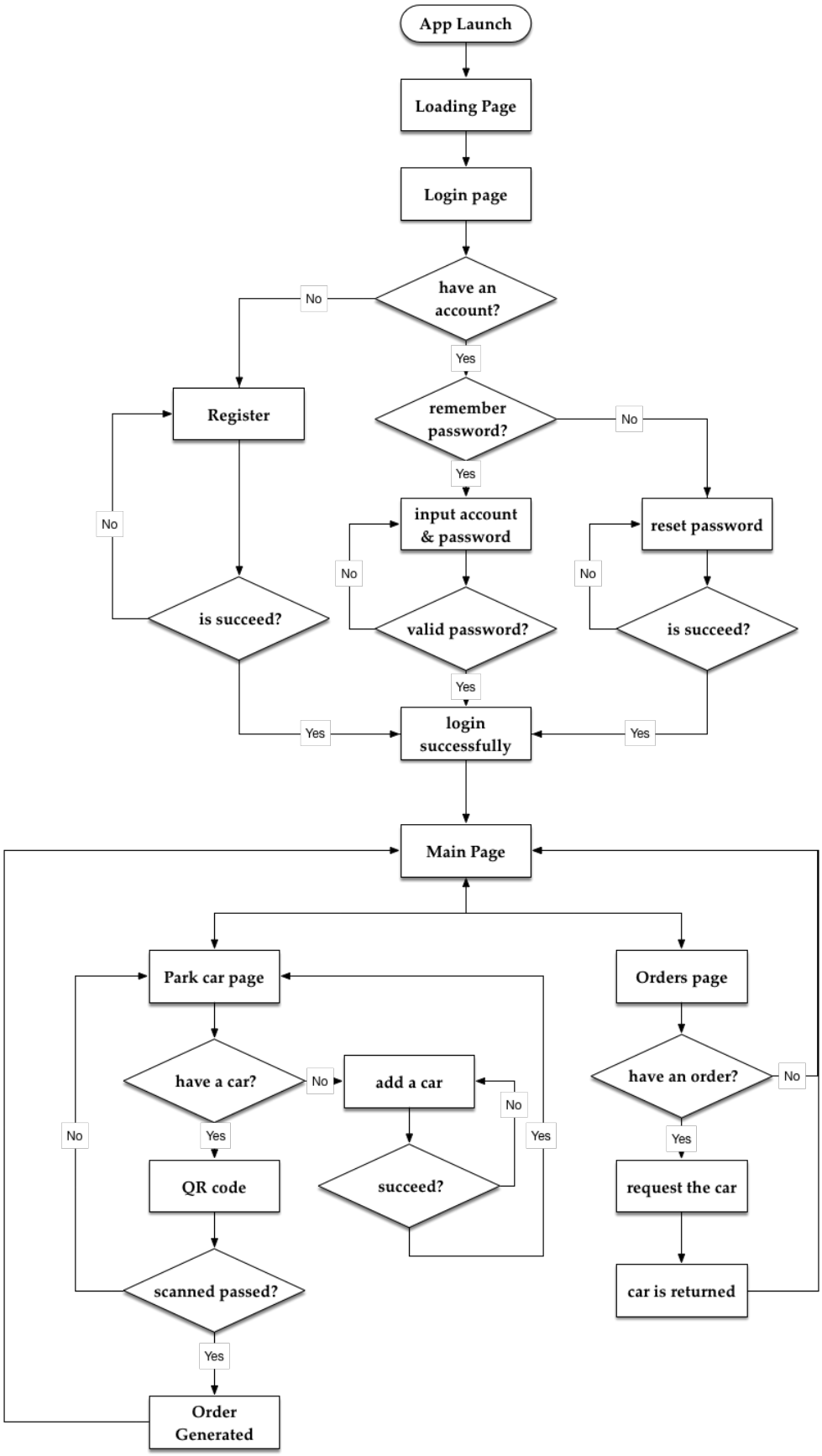


Figure 4-5 flow chart for mobile client

1) Account management

When the application is launched at the first time, it will ask the user to input account and password to login. If the user does not have an account, he or she needs to register using its phone to get a verification code. If the user forgets the password, he or she needs to set a new password using its phone to get verification code. The server will do all the verify works including check if the password is valid, if the account exists and so on.

If the account and password are valid, the *login* view will dismiss and the main page will show. If not, it will ask the user to input account and password again.

2) Car and order management

In the main view, a user can park its car. But if the user does not create a car in the application, he or she needs to create a car by inputting information including plate, brand and color. The server will check if a car with the same plate exist. Then the user can choose a place to park its car and a QR code will show on the screen for our valet to scan. After some checking on the

server, if all the information is valid, a new order will be generated. The application will go back to the main page.

Whenever the user wants its car back, he or she can enter the order page, choose an order and request that car. Out valet will start to drive its car back.

### 4.2.3. Object Oriented Design

Below I will introduce the object oriented design in this project.

#### 4.2.3.1. Identify actors and use cases

Table 4-1 shows the actors and use cases for this project

Table 4-1 table for actors and use cases

| Actor | Use Case | Use Case Description |
|---|---|---|
| **Customer** | Login | Customer logs in |
| **Customer** | Register | Customer registers the service |
| **Customer** | Reset password | Customer resets its password |
| **Customer** | Modify a car | Customer can add, update or delete a car |
| **Customer, valet** | Park a car | Customer parks its car |

#### 4.2.3.2. Use case diagrams

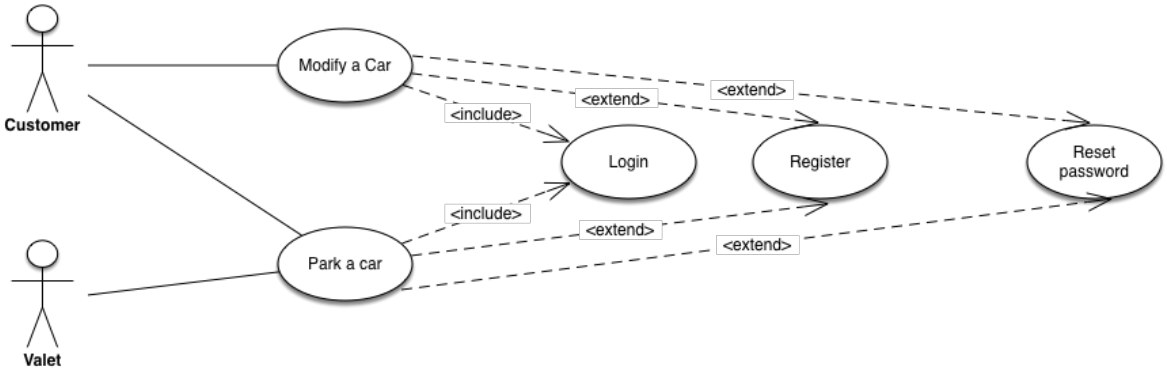


Figure 4-6 use case diagram

Figure 4-6 shows the use case diagram. As it shows in the picture, there are three use cases.

1) For the “modify a car” use case, a customer can modify its cars including create, update and delete a car.
2) For the “park a car” use case, a customer can park its car if the order is generated successfully by a valet. And the user can request its car if the car is in a parking lot. Then a valet will return the car.
3) Both the use cases above need to include “login” use case.
4) If the user does not have an account, the “register” use case will be invoked.

#### 4.2.3.3. Sequence diagrams

1) Figure 3-7 shows the sequence diagram for register and login. As shown in the figure, a customer need to register first to use

our service. In this sequence diagram, all the message are asynchronous.

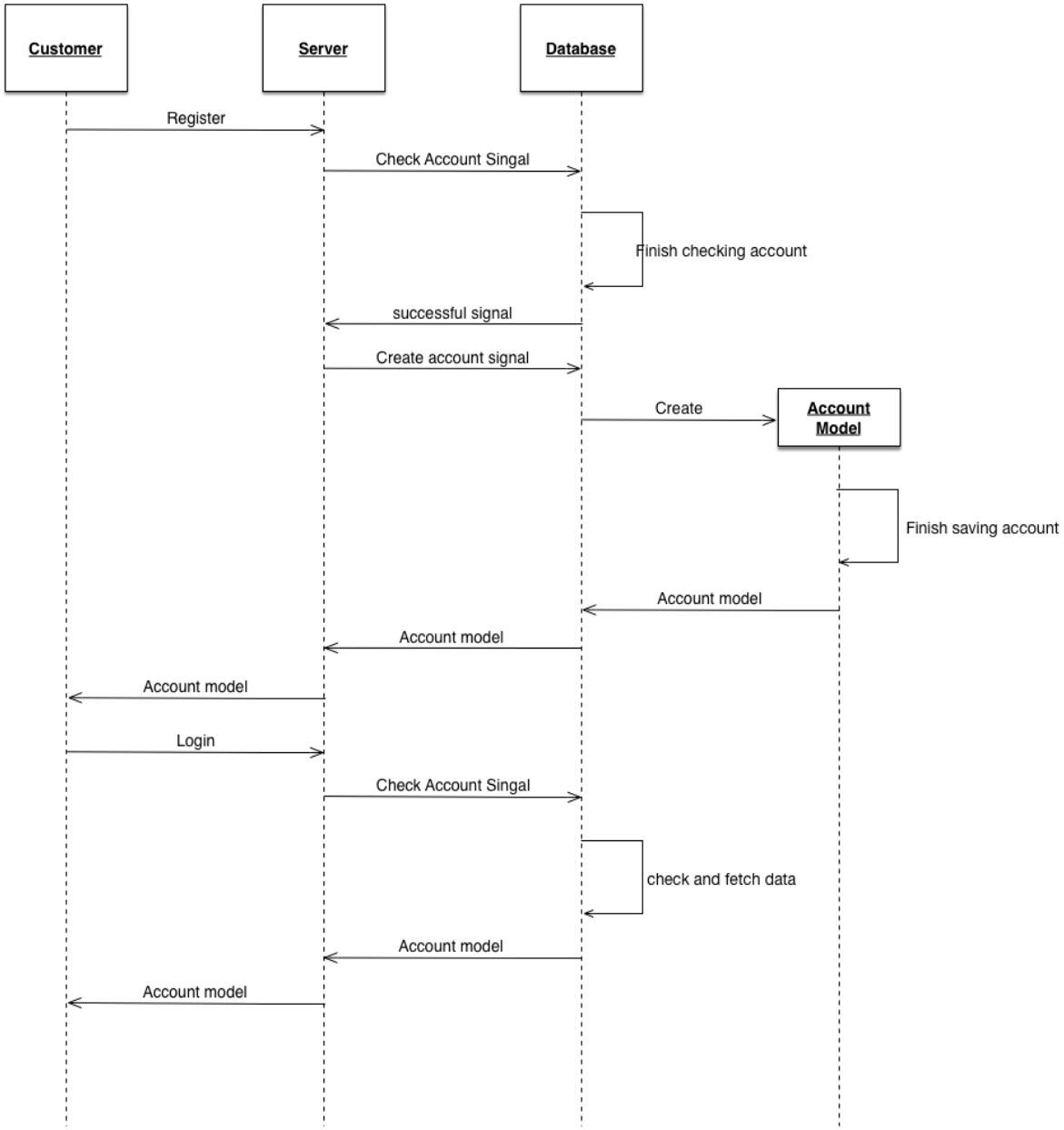


Figure 4-7 sequence diagram for registration and logging in.

2) Figure 4-8 shows the sequence diagram for parking a car. As shown in the figure, a customer needs to add a car before using our valet parking service. And it is a valet who sends a an order

request to server after scanning the QR code generated by a customer. The server will check each request including “add a car” and “create an order” for correctness and validity. Then the server will communicate with database and send back the corresponding model object.

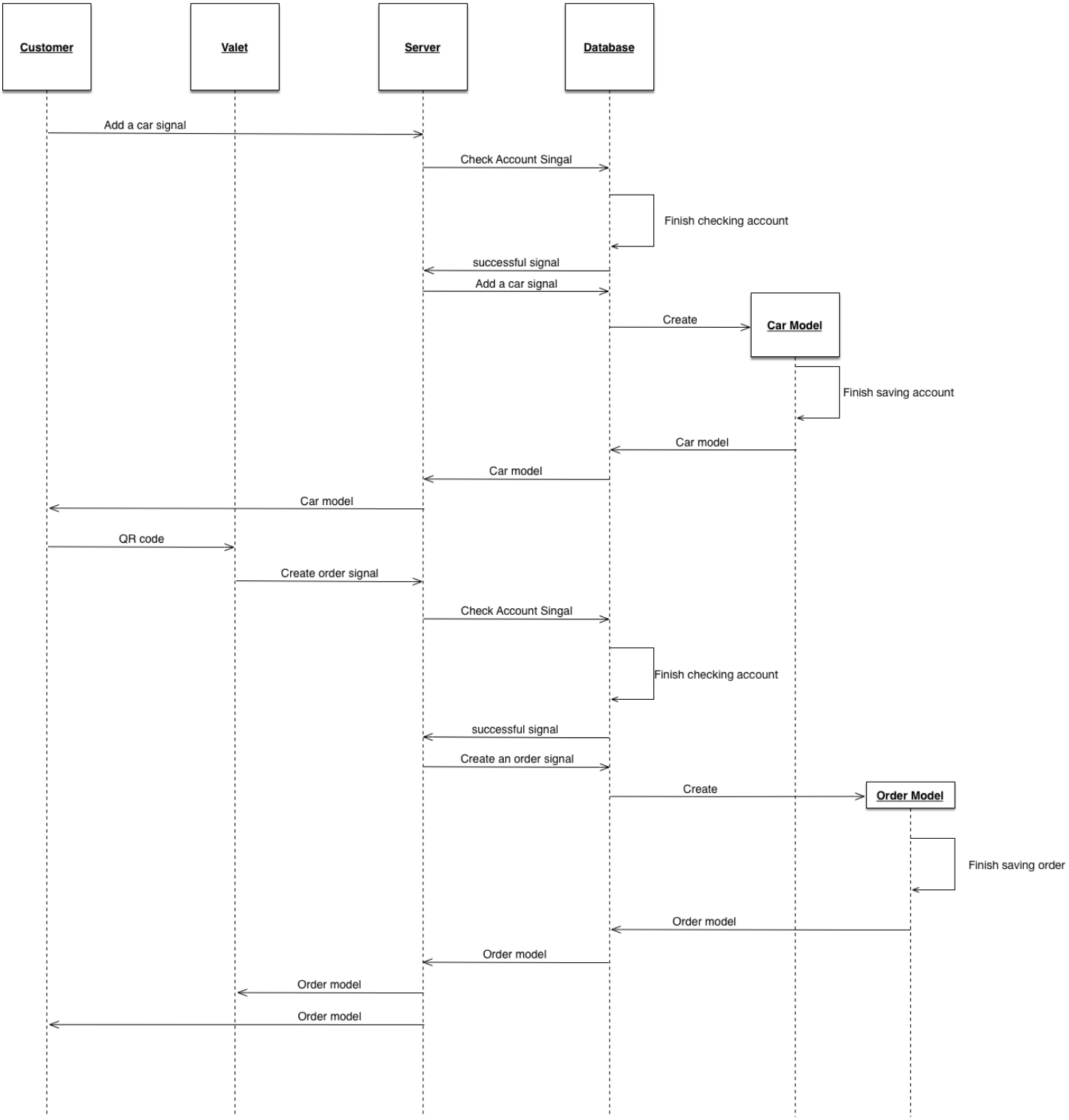


Figure 4-8 sequence diagram for parking a car

#### 4.2.3.4. Documents for design use cases

Below introduce the documents for the four use cases.

1) "Register" use case

| **Use Case Name:** | Register | |
|---|---|---|
| **Actor(s):** | Customer | |
| **Description:** | This use case describes the membership registration process. | |
| **Reference:** | FS-1 | |
| **Typical Courses of Events:** | **Actor Action** | **System Response** |
| | The welcome window is currently displayed on the screen waiting for the customer to select an option. | |
| | **Step1**: Initiate this use case if a customer clicks the [Sign up for Valet Parking] button. | |
| | | **Step2**: The system pops up a multi-lined window requesting the following information to be input: First Name, Last Name, Phone Number, Verification Code and Password. The window has one buttons at the bottom: [Sign up], one button at the left top corner: [Cancel] and a [Get verification code] button besides the text field for verification code. And the customer can read Terms by clicking the [Terms] button. |
| | **Step3**: The customer inputs First Name, Last Name and Phone Number then clicks | |

| | | |
|---|---|---|
| | the [Get Verification Code] button. | |
| | | Step4: The system sends the phone number to Mob to get a verification code. If the format of phone number is correct, the customer will get an message with verification code |
| | Step 5: The customer inputs the verification code and a password then clicks the [Sign up] button. | |
| | | Step 6: A progress indicator will display on the screen. The system sends all the information to the server to verify if the phone number is valid, if so, the server will create a user model and save all the information in the server. If the saving process is successful, the server will send the user model back to mobile client. |
| | | Step 7: this use case concludes when the mobile client receives a successful message and the user model. The registration view will dismiss and the main page will show. |
| **Alternative(s):** | Step3: if the customer clicks [Cancel] button on the left top corner, the registration view will dismiss and the welcome page will show. This use case concludes. | |

| | |
|---|---|
| | Step4: if the phone number or verification code is not correct, customer needs to correct the number or get verification code again. This use case goes back to Step3.<br>Step6: if the account with same phone number already exist, the server will send back an error message and the mobile client will display a MBProgressHUD showing "This account already exists". Then this use case goes back to step3. If server can not save the information, it will send back an error message and the mobile client will display a MBProgressHUD showing "Fail to Register". Then this use case goes back to step3. |
| **Precondition:** | None |
| **Postcondition:** | User successfully registers to use our service |
| **Assumptions:** | None |

2) "Login" use case

| | | |
|---|---|---|
| **Use Case Name:** | Login | |
| **Actor(s):** | Customer | |
| **Description:** | This use case describes the login process for a user | |
| **Reference:** | FS-1 | |
| **Typical Courses of Events:** | **Actor Action** | **System Response** |
| | **Step1**: Initiate this use case when a customer opens the application for the first time or if the user has logged out. | |
| | | **Step2**: The system shows a welcome view with two text field requesting the user to input: Phone Number (account number) and Password. This view has three buttons: [Log in], [Sign up for Valet Parking] and [Forget Password] |

| | | |
|---|---|---|
| | **Step3**: The customer inputs Phone Number and Password then clicks the [Log in] button. | |
| | | Step4: A progress indicator will display on the screen. The mobile client sends customer's phone number and password to the server for verification. |
| | | Step 5: the server checks if the phone number and password are valid. Is so, the corresponding user model will be fetched from database and sent back to mobile client. |
| | | Step 6: this use case concludes when the mobile client gets the response and dismisses the welcome view to show the main page. |
| **Alternative(s):** | Step3: if the customer clicks [Sign up for Valet Parking] button, a registration view will pop up. Then it goes to use case "Register". If the customer clicks [Forget Password] button, a view for resetting password will pop up. Then it goes to use case "Reset password".<br>Step4: if the phone number and password are invalid. Server will send back an error message indicating invalid password. And the mobile client will display a MBProgressHUD showing "invalid password". Then the customer needs to correct its information. This use case goes to Step 3. | |
| **Precondition:** | This customer already has an account | |
| **Postcondition:** | None | |
| **Assumptions:** | None | |

3) "Reset password" use case

| **Use Case Name:** | Reset password | |
|---|---|---|
| **Actor(s):** | Customer | |
| **Description:** | This use case describes the reset password process for a user | |
| **Reference:** | FS-1 | |
| **Typical Courses of Events:** | **Actor Action** | **System Response** |
| | The welcome window is currently displayed on the screen waiting for the customer to select an option. | |
| | **Step1**: Initiate this use case if a customer clicks the [Forget Password] button. | |
| | | **Step2**: The system pops up a multi-lined window requesting the following information to be input: Phone Number, Verification Code and a new Password. The window has one buttons at the bottom: [OK], one button at the left top corner: [Cancel] and a [Get verification code] button besides the text field for verification code. |
| | **Step3**: The customer inputs Phone Number then clicks the [Get Verification Code] button. | |
| | | Step4: The system sends the phone number to Mob to get a verification code. If the format of phone number is correct, the customer will get |

| | | |
|---|---|---|
| | | an message with verification code |
| | Step 5: The customer inputs the verification code and a password then clicks the [OK] button. | |
| | | Step 6: A progress indicator will display on the screen. The system sends all the information to the server to verify if the phone number is valid, if so, the server will fetch the corresponding user model and update the password information in the server. If the saving process is successful, the server will send the user model back to mobile client. |
| | | Step 7: this use case concludes when the mobile client receives a successful message and the user model. The reset password view will dismiss and the main page will show. |
| **Alternative(s):** | Step3: if the customer clicks [Cancel] button on the left top corner, the registration view will dismiss and the welcome page will show. This use case concludes.<br>Step4: if the phone number or verification code is incorrect, customer needs to correct the number or get verification code again. This use case goes back to Step3.<br>Step6: if the account is not found, the server will send back an error message and the mobile client will display a MBProgressHUD showing “Account not found”. Then this use case goes back to step3. If server can not save the | |

| | |
|---|---|
| | information, it will send back an error message and the mobile client will display a MBProgressHUD showing “Fail to reset password”. Then this use case goes back to step3. |
| **Precondition:** | None |
| **Postcondition:** | User successfully resets its password |
| **Assumptions:** | User already has registered for this service. |

4) “Modify a car” use case

| | | |
|---|---|---|
| **Use Case Name:** | Modify a car | |
| **Actor(s):** | Customer | |
| **Description:** | This use case describes the process of modifying a car | |
| **Reference:** | FS-1 | |
| **Typical Courses of Events: Alternative(s):** | **Actor Action** | **System Response** |
| | The user’s cars collection view is currently displayed on the screen waiting for the customer to select an option. | |
| | **Step1**: Initiate this use case if a customer clicks a cell or the [add] button on the right top of the view. | |
| | | Step 2: if the customer wants to add a new car. A [Add a car] view will pop up with three information for customer to input: Car Plate, Car Brand and Car Color. If the customer wants to edit an existing car, a [Edit a car] view will pop up with plate, brand, color already filled. There are two buttons on this page, on is [Cancel] on the top left corner and the other |

| | | |
|---|---|---|
| | | one in [Done] on the top right corner. |
| | **Step 3:** the customer needs to type in the plate, choose brand and color from the system. And then click the [Done] button. | |
| | | Step 4: the mobile client will firstly check if the user has a car with same plate locally, if not, it will display a progress indicator and send the car model to server. |
| | | Step 5: the server will firstly check if the account is valid, if so, it will save the car model to the database. If the saving process is successful, it will send a successful message with the car model to the mobile client. |
| | | Step 6: the mobile client gets the message and save the car model locally. This use case concludes when the saving process is finished. |
| | Step3: if the customer clicks [Cancel] button on the left top corner, this view will dismiss and cars collection view will show. This use case concludes.<br>Step4: if the mobile client finds out that the user adds a duplicate car, a MBProgressHUD will display showing “You already has this car”. Then the customer needs to change the plate. This use case goes back to step 3.<br>Step6: if the account is not found, the server will send back an error message and the mobile client will display a MBProgressHUD showing “Account not found”. Then this | |

| | |
|---|---|
| | use case goes back to step3. If server can not save the information, it will send back an error message and the mobile client will display a MBProgressHUD showing “Fail to save this car”. Then this use case goes back to step3. |
| **Precondition:** | None |
| **Postcondition:** | User successfully update a car |
| **Assumptions:** | User already has an account |

5) “Park a car” use case

| | | |
|---|---|---|
| **Use Case Name:** | Par a car | |
| **Actor(s):** | Customer, Valet | |
| **Description:** | This use case describes the process of parking and returning a car | |
| **Reference:** | FS-1 | |
| **Typical Courses of Events:** | **Actor Action** | **System Response** |
| | The main page is currently displayed on the screen waiting for the customer to select an option. This view is a table view with two options: [Park Now] and [Current Orders] | |
| | **Step1**: Initiate this use case if a customer clicks [Park Now]. | |
| | | Step 2: A [Park] view will show. This is a table view with two sections. The first section has four rows: Place, Name, Phone and Car. If the customer already adds at least one car, the lasted added car will be displayed. The section section is a [Confirm] button. |

| | | |
|---|---|---|
| | | And there is a [Back] button on the top left corner. |
| | **Step 3:** the customer chooses which car to park and clicks [Confirm]. | |
| | | Step 4: a [Placing an Order] view shows with a label and a QR code. The label tells the customer to show the QR code to a valet. |
| | **Step 5:** a valet scans the QR code using its application. | |
| | | Step 6: the mobile client of valet sends the order information to server. The server will firstly check if the account and the car are valid. If so, it will check if the user has an unfinished order using the same plate. If not, the server will create a new order model and save it to database. If the saving process is successful, the server will send a successful message with the order model to mobile client. |
| | | Step 7: the mobile client of valet goes back to main page and refresh the table to show the latest order. |
| | **Step 8:** the customer wants to get its car back. He or she clicks the [Current Orders] | |

| | | |
|---|---|---|
| | | Step 9: a [Current orders] view shows displaying all unfinished orders of this user. There are kinds of unfinished order: the one in parking lot and the one with a valet. |
| | Step 10: the customer clicks an order with status “in parking lot” | |
| | | Step 11: a [Status] view shows displaying a table view showing order information. There are two sections, the first section show plate, brand and color. The second section is a [Recall] button. |
| | Step 12: the customer clicks the [Recall] button. | |
| | | Step 13: the mobile client of customer sends the request to server and server will update the order status. Then server will send the new order object to customer and notify all valets that a user requests it car. The [Status] view will pop out and [Current Orders] view will show again with new car status. For mobile client of valets, the main page will refresh with new car status. |
| | Step 14: after the car returning the customer, a valet clicks the finished order cell | |

| | | |
|---|---|---|
| | | Step 15: a [Order] view shows with order information: customer's name, car plate. And there are a [End] button. |
| | Step 16: a valet clicks the [End] button | |
| | | Step 17: the mobile client sends an end order request to server. The server will check if the customer, car and order are valid. If so, it will update the order send a new order model to all valets. This use case concludes. |
| **Alternative(s):** | Step2: if the customer has not added a car, an [Add a car] view will pop up. The system goes to use case [Modify a car].<br>Step3: if the customer clicks [Back] button on the left top corner, this view will dismiss and main view will show. This use case concludes.<br>Step6: if the server finds that the account or car or order information is wrong, it will send an error message to valet telling the reason. The customer needs to correct its information. This use case goes back to step 3. | |
| **Precondition:** | None | |
| **Postcondition:** | User gets its car back | |
| **Assumptions:** | User already has an account | |

4.2.3.5. Class diagrams

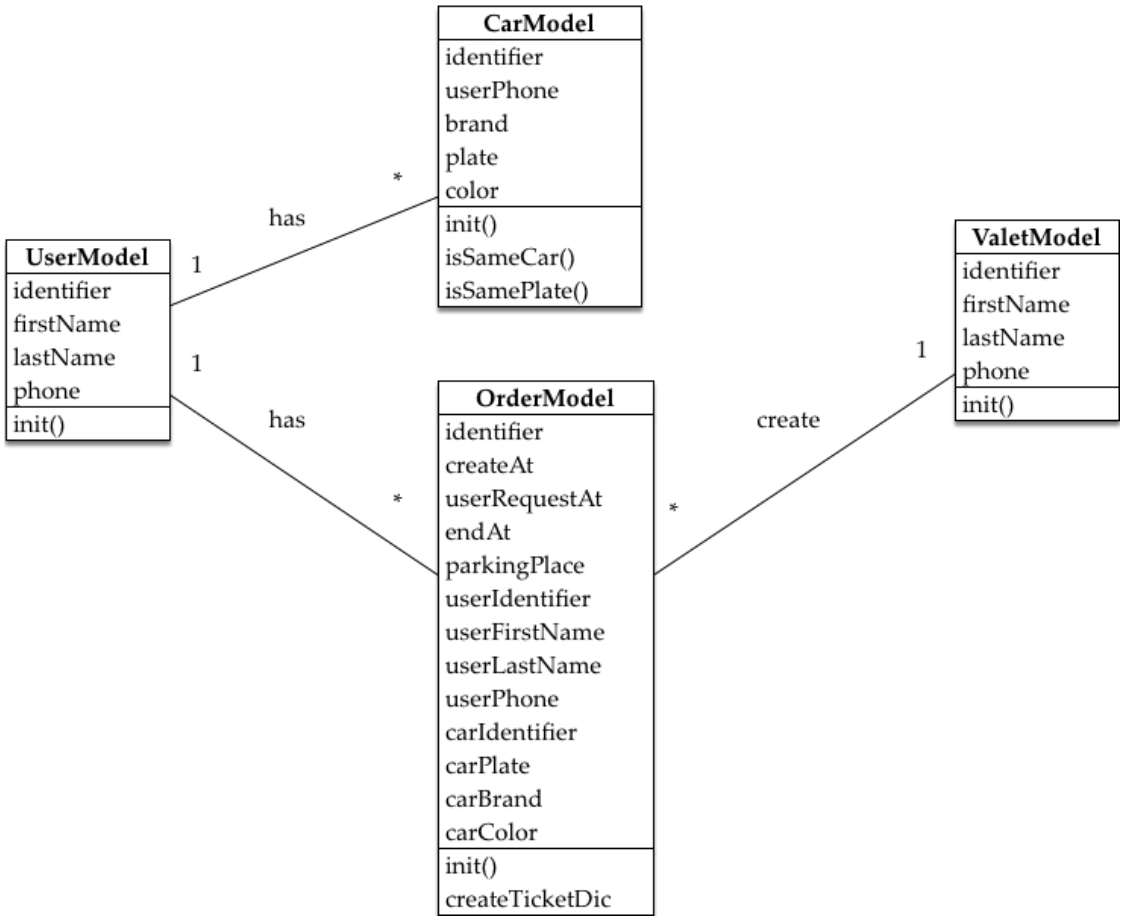


Figure 4-9 class diagram

As shown in figure 4-9, the class diagram shows the relationship between different class. The *UserModel* indicates the customer. It stores identifier, first name, last name and phone of a customer. The *CarModel* stores identifier, customer's phone, brand, plate and color for a car. The *OrderModel* stores identifier, creating time, requesting time, ending time, user's identifier, user's first name, user's last name, user's phone, car's identifier, car's plate, car's brand and car's color. A *UserModel* may have multiple *CarModel* and *OrderModel*.

The *ValetModel* indicates the valet. It stores identifier, first name, last name and phone of a valet. A *ValetModel* may create multiple *OrdreModel*.

### 4.2.4. User Interface Design

User interface (UI) is important because a good UI will attract a user at first glimpse and keep that user. In this part, I will introduce the UI design of this project. There are mobile applications and the UI design will be introduced separately.

#### 4.2.4.1. UI design for Customers

1) *Login/ Register/ Reset Password Pages*

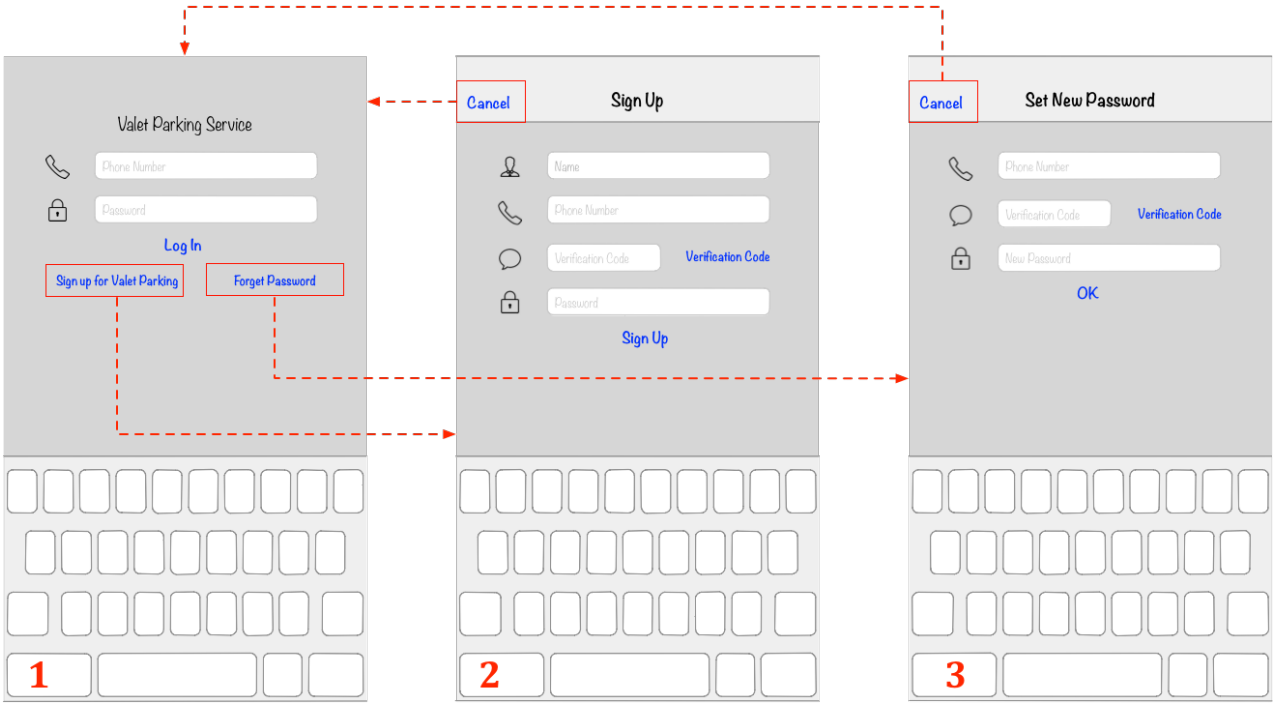


Figure 4-10 UI design of *Login/ Register/ Reset Password Pages*

As shown in figure 4-10, there are three pages to display before a customer has logged in: *Login Page, Register Page* and *Reset Password Page*.

- *Login Page*: this page shows when the application if opened for the first time. Customers can login on this page by inputting phone number and password the clicking [Log In] button.
- *Register Page*: this page shows when a customer clicks [Sign up for Valet Parking] on the *Login Page.* Customers can

register on this page by inputting all the information including a verification code.

- *Reset Password Page*: this page shows when a customer clicks [Forget Password] button on the *Login Page*. Customers can reset their password on this page by inputting all the information including a verification code.

2) Main function pages

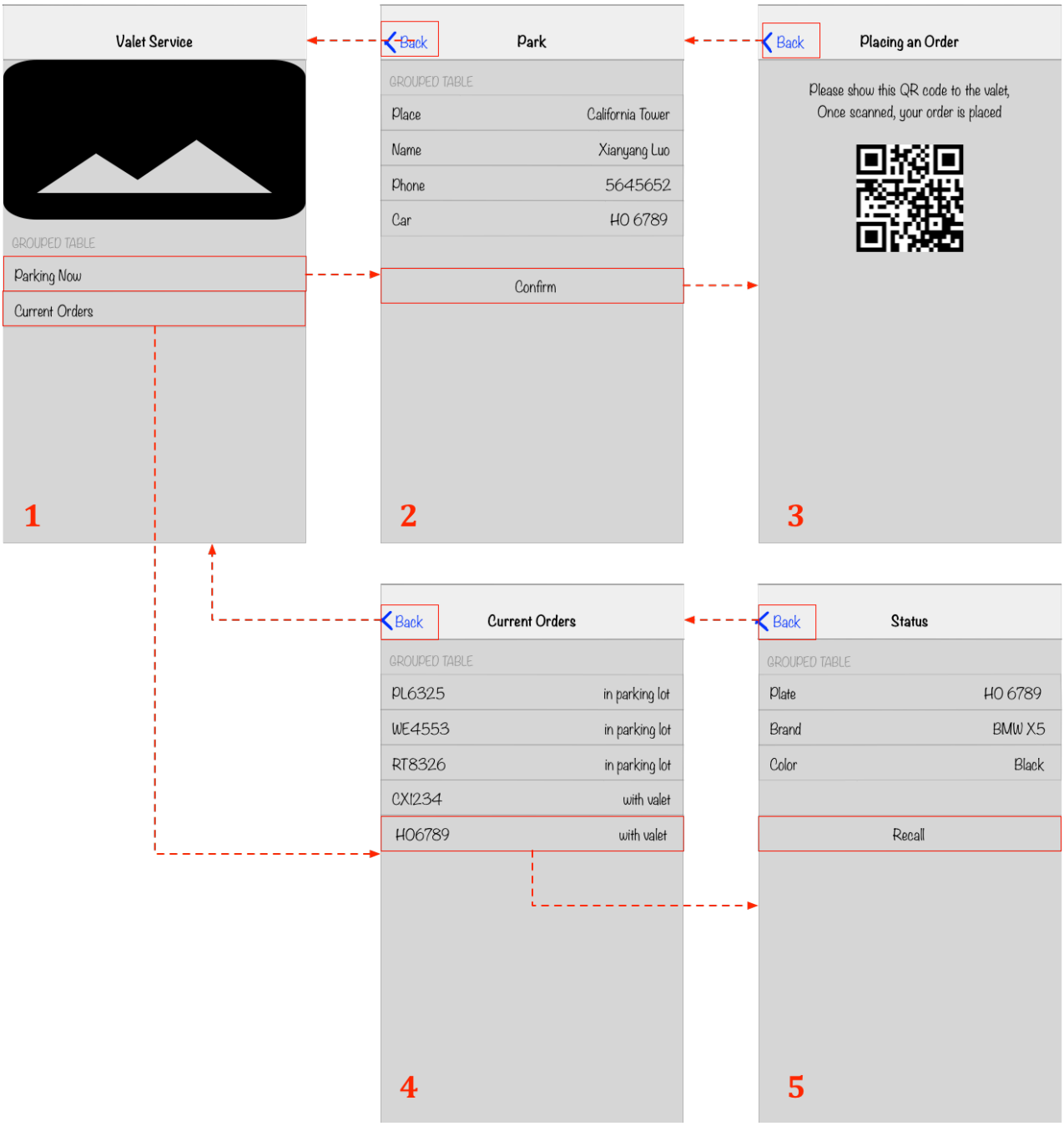


Figure 4- 11 UI design of Main Functions Pages for Customers

As shown in figure 4-11, there are five pages containing main functions of this application. They are *Main Page, Park Park Page, Placing an order Page, Current Orders Page* and *Order Status Page.*

- *Main Page*: this page shows when a customer logs in. There are two parts in this page. The first part is a scrollable bar displaying images. The second part is a table view with two cells: *"Park Now"* and *"Current Orders"*.
- *Park Page*: this page shows when a customer clicks "Park Now" cell on the *Main Page*. There is a table view with two sections in this page. The first section is four rows for customer to type in information. The second section is a *"Confirm"* button.
- *Placing an order Page:* this page shows when a customer clicks *"Confirm"* button on the *"Park Page"*. This page has a label telling customer to show the QR code to a valid. The QR code below the label contains basic information for this order.
- *Current Orders Page:* this page shows when a customer clicks *"Current Orders"* in the *"Main Page"*. There is a table view in this page displaying all the unfinished orders for the customer.
- *Order Status Page:* this page shows when a customer clicks an order in *"Current Orders"* page. There is a table view with two sections in this page. The first section displays the basic

information including car plate, car brand and car color. If order status is “in parking lot”, the second section is a button showing “Recall”. Customer can click this button to recall its car. If the order status is “returning with valet”, the second section is a label displaying “returning with valet”.

3) Car Collection Pages

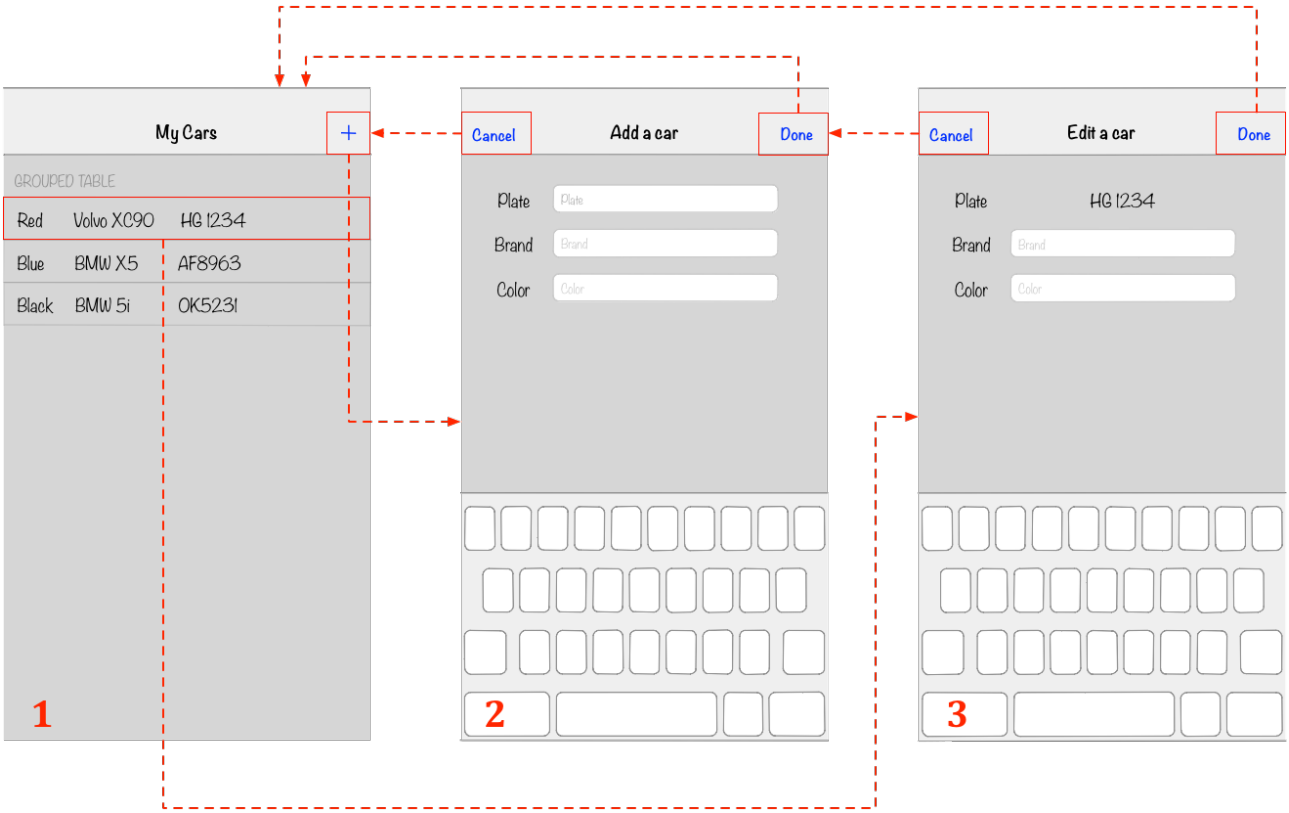


Figure 4-12 UI design of Car Collection Page

As shown in figure 4-12, there are three pages for customers to view, add or update a car. They are *My Cars page, Add a car page* and *Edit a car page.*

- *My cars page*: this page shows when a customer clicks "*My Cars"* in the *Me* tab. There is a table view in this page displaying all the cars belong to the customer. Each car has three information: plate, brand and color. There is an [Add] button on the top right corner of this page.
- *Add a car page*: this page shows when a customer clicks the [Add] button on the *My cars Page*. There are three information that customer needs to input or choose and two buttons on the top. After filling all the information, the customer can click the [Done] button on the top right corner. Then the information will be sent to server to check and save. If the customer wants to cancel the process, he or she can click the [Cancel] button on the top left corner then the page will dismiss.
- *Edit a car page*: this page shows when a customer clicks a row on the *My cars Page*. The customer can change the car's brand and color. Then he or she can click the [Done] button and the information will be saved. If the customer wants to cancel the process, he or she can click the [Cancel] button on the top left corner then the page will dismiss.

#### 4.2.4.2. UI design for Valets

The mobile clients for valets is rather simple. The *login page* and *reset password page* is almost the same with the customers'. So I will just introduce the *Main functions page* of this mobile client. The design for *Main functions page* is shown in figure 4-13.

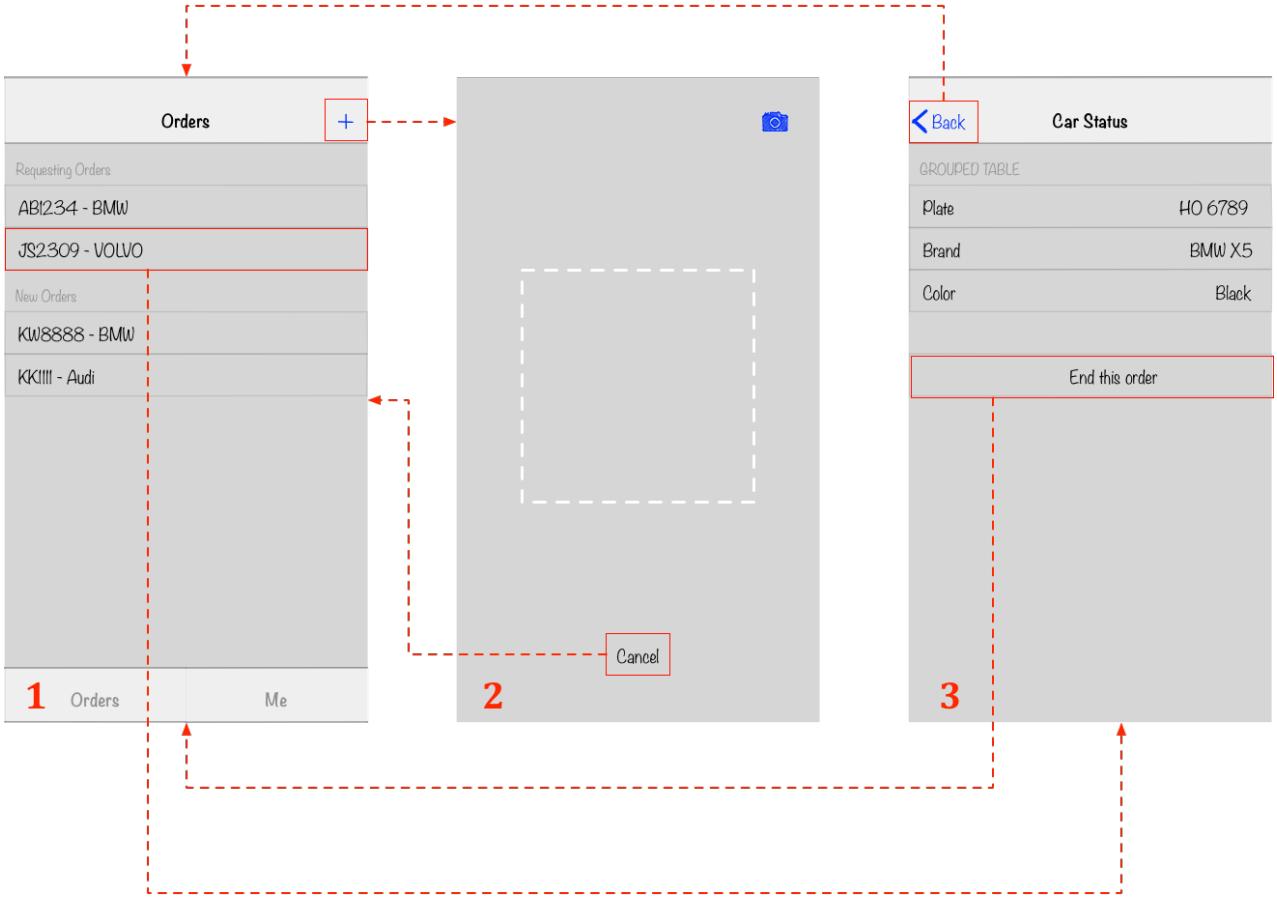


Figure 4-13 UI design of Main Functions Pages for Valets

As shown in figure 4-13, there are two main functions for valets. The first one is creating an order and the second one is ending an order. The first page is the main page in this application. There is a table with two sections displaying "new orders" and "requesting orders". "New orders" are newly created by valets and "requesting orders" are orders

that are requested by customers. This main page can automatically refresh every 30 seconds or pull to refresh to keep update with the server. The two functions are introduced below.

1) Create an order

Any valet can use his or her mobile applications to scan a QR code of a user. A valet should open the "Orders" tab and press the "+" button on the top right corner. Then the application will push to second view: scanner view. This view is for scanning QR code. There is a rectangle in the middle of the page. The valet should put a QR code in the rectangle to scan it. If the order is valid, the order should be created automatically on the server.

2) End an order

Any valet can use his or her application to end an order by clicking "End this order" on the third page after returning a car to its owner. In the *Car Status* page, there is a table view with two sections. The first section indicating the information including car plate, car brand and car color of this order. The second section is a button for the valet to end this order. Once the button is clicked, the mobile application will send an end order request to server. The server will check the validity and sends response back. Once the information is

valid, the mobile application in both customers and valets will be refreshed with new data.

### 4.2.5. Database Design

This project uses MongoDB as database. The basic data transfer is shown in figure 4-14

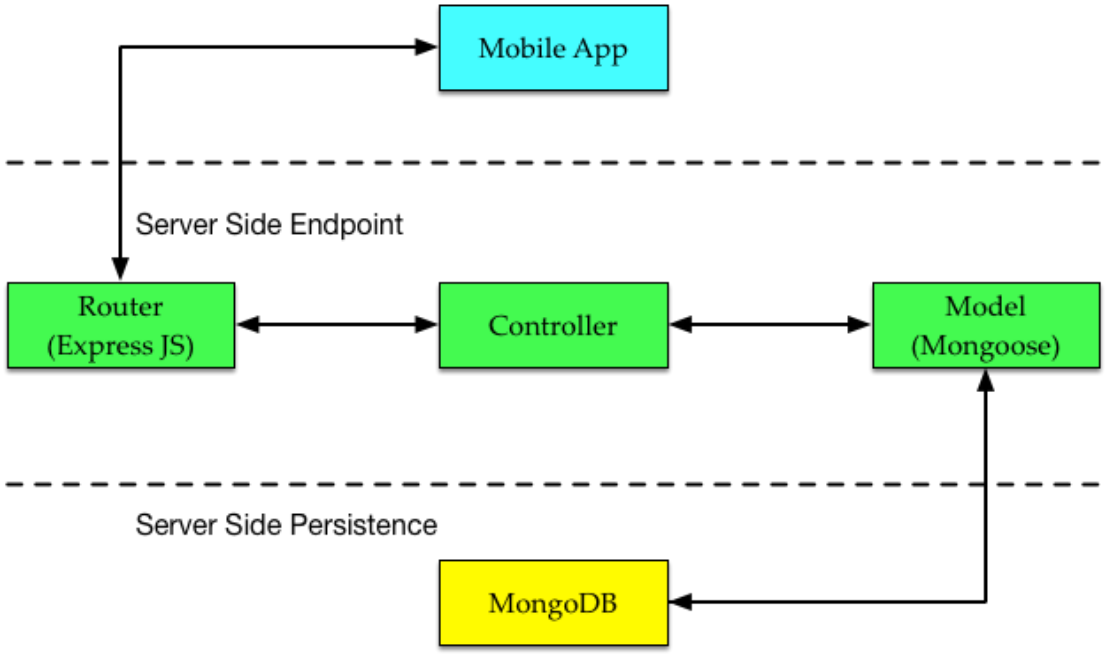


Figure 4-14 path for data transfer

As shown in figure 4-14, Mongoose acts as the upper layer of MongoDB. All the four models in server: *user, valet, car* and *order* inherit from Mongoose. They can create, read, update and delete documents in database easily. The high level database structure is shown in figure 4-15

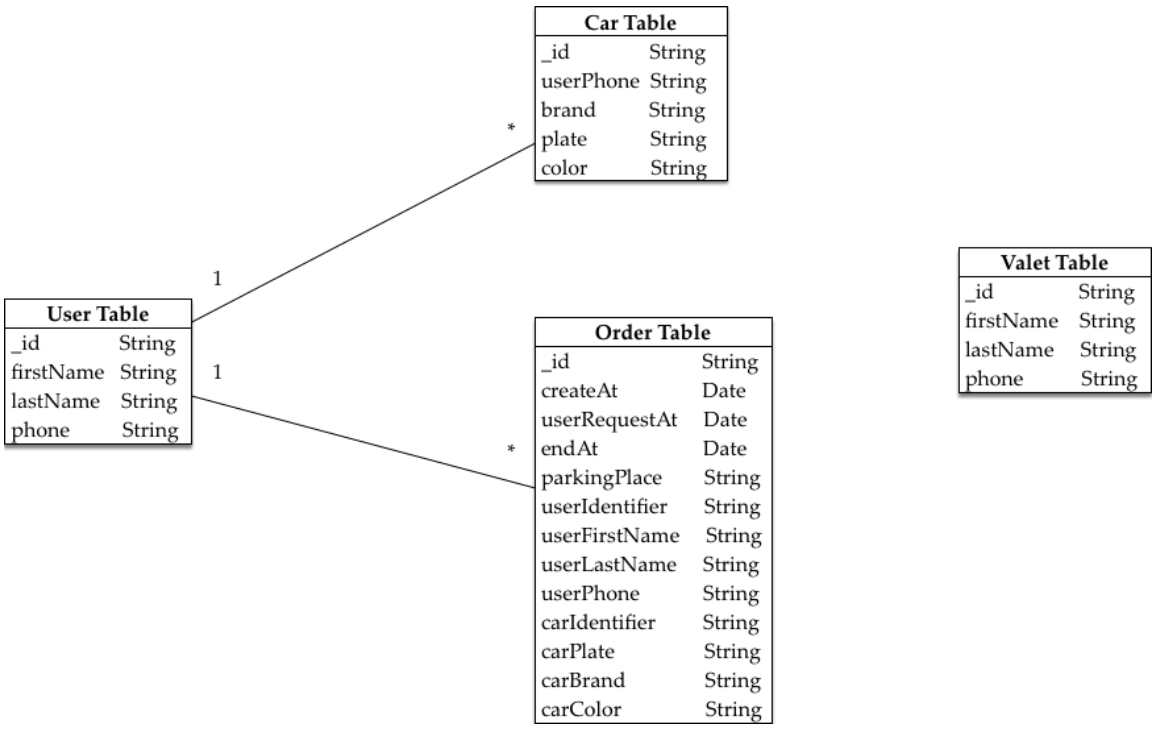


Figure 4-14 Database design

As shown in figure 4-14, there are four tables in this database. The *user* holds all the information for customers and valet is for valets. The *car* holds information for cars including plate, brand and color. The *order* holds information for orders including date, user's name, car's plate and so on. Each user may have multiple cars and orders. So the relationship between user and car are one-to-many, so as to user and order.

# Chapter 5. IMPLEMENTATION AND TEST

In this chapter, I will introduce how this project is implemented. Firstly I will give some brief introduction for the development environment. And then I will introduce implementation of server and implementation of mobile clients respectively.

## 5.1. Development environment

### 5.1.1. Hardware configurations

The hardware I use to develop the server and mobile clients of this project is MacBook Pro with mac OS Sierra version 10.12.1 as shown in figure 5-1.


Figure 5-1 Hardware configurations

And the hardware I use to test this project is iPhone 6s and iPhone 7 plus with iOS 10.1.1.

### 5.1.2. Software configurations

The software I use to implement the mobile clients is Xcode version 8.1 as shown in figure 5-2. Xcode is developed by Apple and is the official IDE in iOS application developing.


Figure 5-2 Xcode with version number

The software I use the implement the server is Axiom which is a text editor. And the application I use the test the server is Postman.

## 5.2. Implementation of server

As shown in Figure 4-14, there are three layers in the server: router, controller and model. Router is used to invoke corresponding method

based on different request. Controller is used to control the whole process and model is used to interact with database. I will introduce the implementations of these three parts respectively.

### 5.2.1.Implementation of model

**I use Mongoose to generate model of customer, valet, order and car. Figure 5-3 shows the implementation of customer model.**

```
3  var mongoose = require('mongoose');
4  var Schema = mongoose.Schema;
5
6  var UserSchema = new Schema({
7      phone: String,
8      firstName: String,
9      lastName: String,
10     passwordHash: String,
11     passwordSalt: String
12 });
13
14 module.exports = mongoose.model('User', UserSchema);
```

**Figure 5-3 Implementation of customer model**

**The implementation of valet, order and car model are similar. Controller can do database action easily using 'User' key in the controller.**

### 5.2.2.Implementation of controllers

**A controller is used to fulfill requests received from mobile clients. There are four controllers in this project: account controller, valet controller, car controller and order controller. Taking account controller as an example, it needs to respond to the following requests: 1) register a customer 2) log on**

**a customer 3) set new password for a customer. The declaration of *AccountController* is shown in figure 5-4**

```
var AccountController = function (userModel) {
    this.crypto = require('crypto');
    this.uuid = require('node-uuid');
    this.ApiResponse = require('../models/api-response.js');
    this.ApiMessages = require('../models/api-messages.js');
    this.UserProfile = require('../models/user/user-profile.js');
    this.User = require('../models/user/user.js');
};
```

**Figure 5-4 declaration of *AccountController***

**The *userModel* argument is an instance of User Mongoose Class. The *crypto* and *uuid* argument are used to generate hash password for user. And *ApiResponse* and *ApiMessages* include the error messages that may happen. For handling request in a controller, an example of *logon* method in *AccountController* is shown in figure 5-5.**

```
AccountController.prototype.logon = function(phone, password, callback) {
    var me = this;
    me.User.findOne({ phone: phone }, function (err, user) {
        if (err) {
            return callback(err, new me.ApiResponse({ success: false, extras: { msg: me.ApiMessages.DB_ERROR } }));
        }
        if (user) {
            me.hashPassword(password, user.passwordSalt, function (err, passwordHash) {
                if (passwordHash == user.passwordHash) {
                    var userProfileModel = new me.UserProfile({
                        identifier: user.id,
                        phone: user.phone,
                        firstName: user.firstName,
                        lastName: user.lastName
                    });
                    return callback(err, new me.ApiResponse({
                        success: true, extras: {userProfileModel:userProfileModel}
                    }));
                } else {
                    return callback(err, new me.ApiResponse({ success: false, extras: { msg: me.ApiMessages.INVALID_PWD } }));
                }
            });
        } else {
            return callback(err, new me.ApiResponse({ success: false, extras: { msg: me.ApiMessages.ACCOUNT_NOT_FOUND } }));
        }
    });
};
```

**Figure 5-5 implementation of *logon* method for customer**

**As shown in figure 5-5, the *logon* method takes *phone, password* and *callback* as variables. It will check if the account exists and then check if the password is valid using *hashPassword*. If both results are true, the controller will return an instance of *UserProfile* which contains basic information for a user including *identifier, phone, first name* and *last name*. If these is an error with the database or the password, the controller will return the corresponding error message to router.**

### 5.2.3.Implementation of routers

**A router receives request from mobile clients and forwards them to controller. I use *Router()* in express to implement router in this project. The implementation of logon in account router is shown in figure 5-6**

```
38  router.route('/account/user/logon').post(function (req, res) {
39      console.log('----------get a user logon post----------');
40      var userLogon = new UserLogon(req.body);
41
42      res.set("Access-Control-Allow-Origin", "http://localhost:42550");
43
44      accountController.logon(userLogon.phone, userLogon.password, function (err, response) {
45          return res.send(response);
46      });
47  });
```

**Figure 5-6 implementation of *logon* request in account router**

**As shown in figure 5-6, if a mobile client sends a post request to "account/user/logon", router will firstly convert the request body to an instance of *UserLogon* which contains user's account and user's password. Then the router will forward the request to *AccountController* together with the phone and password. Then the controller will invoke the *logon* method**

to handle this request and return corresponding response. In this example, the *logon* request should use the format: {phone: 51709669, password: 123} in json format.

### 5.2.4.Implementation of express application

The express application is used to start the server. It uses *express()* to build the server and do some basic configurations as shown in figure 5-7. The database used in this project is "valetParkingDB". And the server and parser file in json and urlencoded format. Then the application adds four routers including account router, valet router, car router and order router. At last, the server will start to listen at port *address().port*.

```
16  var dbName = 'valetParkingDB';
17  var connectionString = 'mongodb://localhost:27017/' + dbName;
18  mongoose.connect(connectionString);
19
20  app.use(expressSession({
21      secret: '128013A7-5B9F-4CC0-BD9E-4480B2D3EFE9',
22      resave: true,
23      saveUninitialized: true,
24      store: new MongoStore({
25          url: 'mongodb://localhost/test-app',
26          ttl: 20 * 24 * 60 * 60 // = 20 days.
27      })
28  }));
29  app.use(bodyParser.json());
30  app.use(bodyParser.urlencoded({ extended: true }));
31  app.get('/', function (req, res) {
32      res.send('<html><body><h1>Valet Parking</h1></body></html>');
33  });
34  app.use('/api', [accountRoute, carRoute, valetRoute, orderRoute]);
35  var server = app.listen(port, function () {
36      console.log('Express server listening on port ' + server.address().port);
37  });
```

**Figure 5-7 implementation of express application**

## 5.3.Implementation of mobile clients

There two mobile clients in this project: customer client and valet client. They will be introduced separately.

### 5.3.1. Customer Client

#### 5.3.1.1. Modules

1) LibraryAPI

LibraryAPI inherits from NSObject and is the most important module in this project. I use singleton pattern to implement this class. It handles all the data request in this project. It has a class method naming *sharedInstance*. The implementation of this method is shown in figure 5-8.

```
25 + (LibraryAPI *)sharedInstance {
26     static LibraryAPI *_sharedInstance = nil;
27     static dispatch_once_t oncePredicate;
28     dispatch_once(&oncePredicate, ^{
29         _sharedInstance = [[LibraryAPI alloc] init];
30     });
31
32     return _sharedInstance;
33 }
34
35 - (id)init {
36     if (self = [super init]) {
37         self.httpClient = [[HttpClient alloc] init];
38         self.dataClient = [[DataClient alloc] init];
39     }
40
41     return self;
42 }
```

Figure 5-8 implementation of *sharedInstance()*

Within the *sharedInstance()* method, it declares a static variable to hold this class and ensure the *init()* method will only be invoked

once using *dispatch_once_t*. And in the *init()* method, two clients will be created: *httpClient* and *dataClient. HttpClient* handles all the request with server and *dataClient* handles all the requests with local database. So the basic work flow using LibraryAPI is shown in figure 5-9.

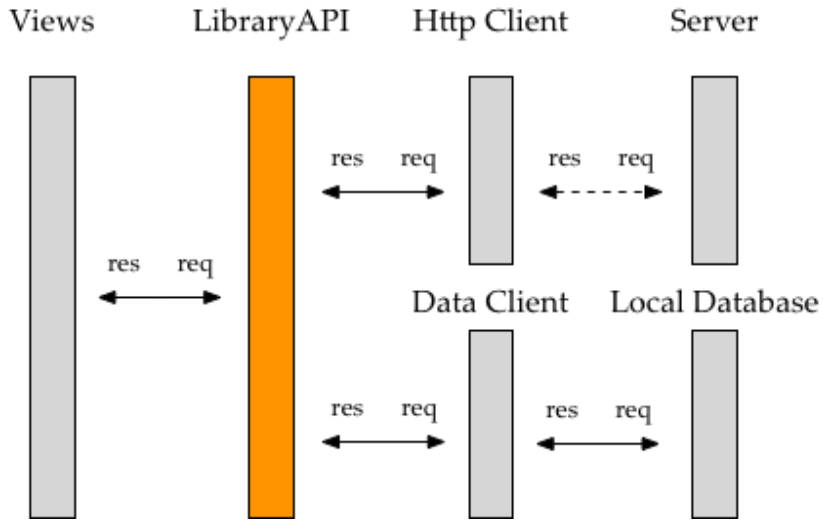


Figure 5-9 Data transfer using LibraryAPI

The method in LibraryAPI is shown in figure 5-10. The variables of these methods are removed for simplicity. It handles requests including *account, car, order, QR image* and some *transformation* request.

```objc
// Account
- (BOOL)isUserLogin;
- (void)tryLoginWithLocalAccount;
- (void)loginWithPhone;
- (void)registerWithPhone;
- (void)resetPasswordWithPhone;
- (void)logout;
- (UserModel *)getCurrentUserModel;
// cars
- (void)getCarsForUser;
- (void)addACar;
- (void)updateACar;
- (void)deleteCarWithCarModel;
- (void)deleteAllCarsInCoreData;
- (NSArray *)getAllCarModelsInCoreData;
// Orders
- (void)getCurrentOrdersForUser;
- (void)recallACar;
- (void)checkOrderWithParkingPlace;
// qr
- (UIImage *)qrImageForString;
- (UIColor *)themeColor;
// check data & transformation
- (BOOL)isPhoneNumberValid;
- (NSString *)transferOrderDateToYYYYMMDDAndTime;
- (NSString *)transferOrderDateToMMDDAndTime;
- (NSTimeInterval)calculateTimeIntervalSinceTime;
```

Figure 5-10 methods in LibraryAPI

Most methods interacting with *httpClient* uses block as return message. An example is shown in figure 5-11 which is *login()* method in LibraryAPI.

```objc
- (void)loginWithPhone:(NSString *)phone
              password:(NSString *)password
               success:(void(^)(UserModel *userModel))successBlock
                  fail:(void(^)(NSError *error))failBlock
{
    [self.httpClient loginWithPhone:phone
                           password:password
                            success:^(UserModel *userModel) {
                                // save login seesion
                                [self.dataClient setLoginInShareApplication];

                                // save the user's account and password
                                [self.dataClient saveAccountToKeychain:phone password:password];

                                // save the user's profile
                                if ([self.dataClient saveUserModelToCoreData:userModel]) {
                                    successBlock(userModel);
                                } else {
                                    failBlock(nil);
                                }
                            }
                               fail:^(NSError *error) {
                                   failBlock(error);
                               }];
}
```

Figure 5-11 *Login()* method in LibraryAPI

This method takes user's phone and password as variable. It invokes *httpClient* to verify the account in server. If the *httpClient* gets the *userModel* successfully. LibraryAPI will begin to save user's information to local database using *dataClient*. If all the information is saved successfully, it will invoke the *successBlock* to refresh the view.

2) HttpClient

*HttpClient* inherits from NSObject and handles all the http request in the mobile client. It uses *AFNetworking* to post a request and handle response. An example is shown in figure 5-12 which is *login()* method in *HttpClient.*

```
121     [manager POST:[url absoluteString]
122        parameters:parameters
123          progress:nil
124           success:^(NSURLSessionDataTask * _Nonnull task, id  _Nullable responseObject) {
125               NSError *error = nil;
126               // get response from the server successfully
127               if ([responseObject isKindOfClass:[NSDictionary class]]) {
128                   BOOL isSuccess = [responseObject[@"success"] boolValue];
129                   if (isSuccess) {
130                       NSError *error;
131                       UserModel *userModel = [[UserModel alloc] initWithDictionary:responseObject[@"extras"][@"userProfileModel"]
132                                                                              error:&error];
133                       if (error) {
134                           failBlock(error);
135                       } else {
136                           successBlock(userModel);
137                       }
138                   } else {
139                       // login failed
140                       ServerErrorMessage failMessage = (ServerErrorMessage)[responseObject[@"extras"][@"msg"] integerValue];
141                       error = [NSError errorWithDomain:NetworkErrorDomain
142                                                   code:failMessage
143                                               userInfo:nil];
144                       failBlock(error);
145                   }
146               } else {
147                   failBlock(nil);
148               }
149           }
150           failure:^(NSURLSessionDataTask * _Nullable task, NSError * _Nonnull error) {
151               failBlock(error);
152           }];
153   }
```

Figure 5-12 *Login()* method in *HttpClient*

As shown in figure 5-12, the manager which is a AFHTTPSessionManager, sends a post request to the url. The parameters is a NSDictionary containing user's account and password. If the server sends back a successful response with user model, *httpClient* will invoke successBlock() with the user model. If not, it will invoke failBlock() with the corresponding error message.

3) DataClient

*DataClient* inherits from NSObject and handles all the local data request. It cooperates with CoreData and KeychainItemWrapper to create, read, update and delete data. KeychainItemWrapper is used to save user's account and password. CoreData is the native iOS database and is used to handle database actions relative to user model and car model locally. Figure 5-13 shows two methods in *DataClient*. The first one is saving login information to KeychainItemWrapper and the second one is saving a car model to local database.

```objc
- (void)saveAccountToKeychain:(NSString *)userAccount password:(NSString *)userPassword {
    KeychainItemWrapper *keychain = [[KeychainItemWrapper alloc] initWithIdentifier:AccountNameInKeychain
                                                                        accessGroup:nil];
    [keychain setObject:userAccount forKey:(__bridge id)(kSecAttrAccount)];
    [keychain setObject:userPassword forKey:(__bridge id)(kSecValueData)];
}

// create a car locally
- (BOOL)saveCarToCoreData:(CarModel *)carModel{
    NSManagedObject *newCar = [NSEntityDescription insertNewObjectForEntityForName:@"Car"
                                                             inManagedObjectContext:self.managedObjectContext];

    [newCar setValue:carModel._id forKey:@"identifier"];
    [newCar setValue:carModel.userPhone forKey:@"userPhone"];
    [newCar setValue:carModel.plate forKey:@"plate"];
    [newCar setValue:carModel.brand forKey:@"brand"];
    [newCar setValue:carModel.color forKey:@"color"];

    return [self saveContext];
}
```

Figure 5-13 example methods in *DataClient*

#### 5.3.1.2. User Interface

1) Welcome Views

The welcome views includes login view, register view and reset password view shown in figure 5-14

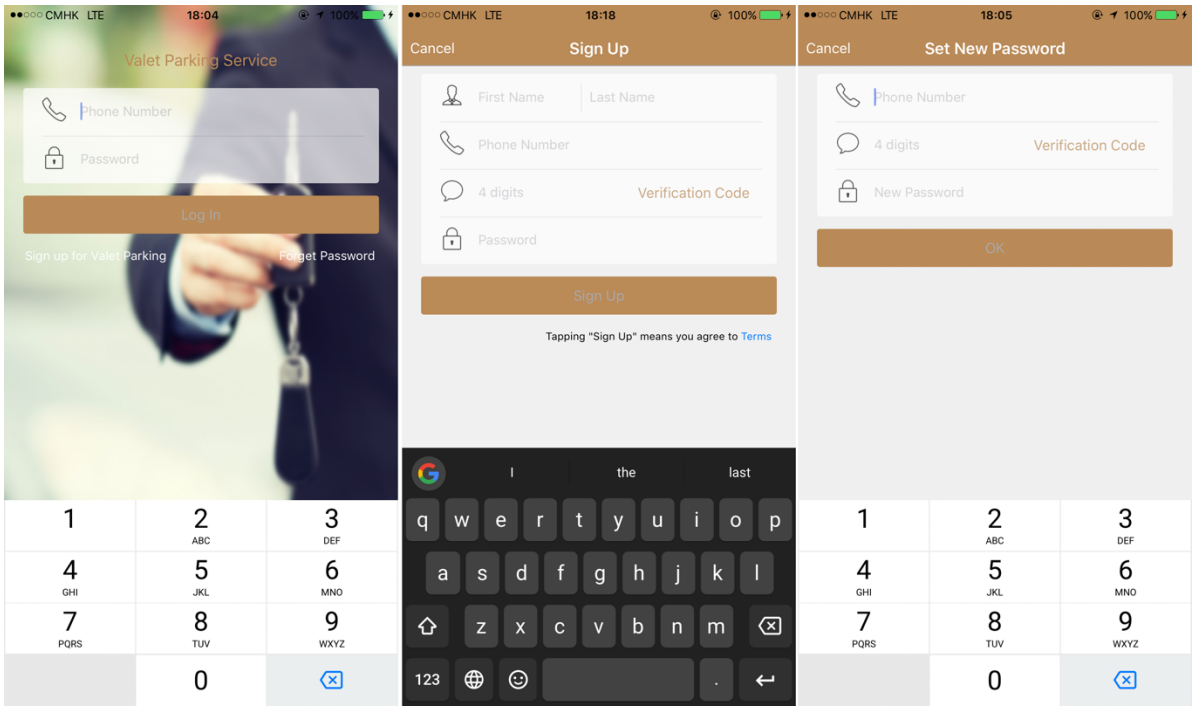


Figure 5-14 welcome views

As shown in figure 5-14. The first view is login view. Customers can login in this view by inputting correct account and password.

The second view is register view. Customers can register a new account in this view by fulfill the information including name, phone number, verification code and password. Customers can get a 4-digits verification code after inputting the phone number.

The third view is reset password view. Customers can set a new password in this view by fulfill phone number, verification code and the new password.

2) Main Page

Main page is the first tab of this application as shown in figure 5-15. It has a scroll bar at the top and two cells for customers to choose. Clicking on the scroll bar will push to corresponding web page and clicking on a cell will push to the corresponding service.

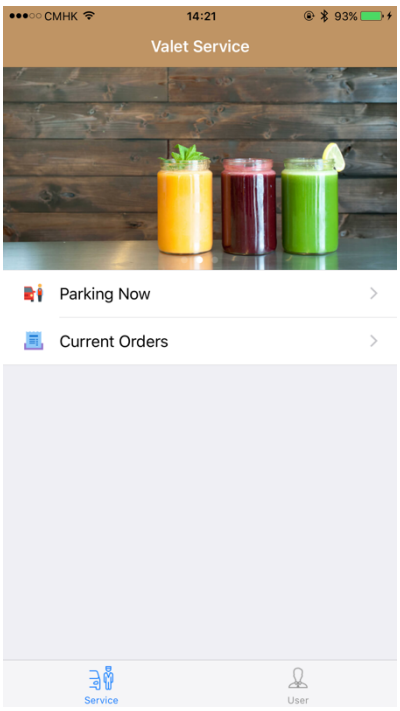


Figure 5-15 main page

3) Park Views

These views are for customers to park their cars. A QR code will be generated at the end.

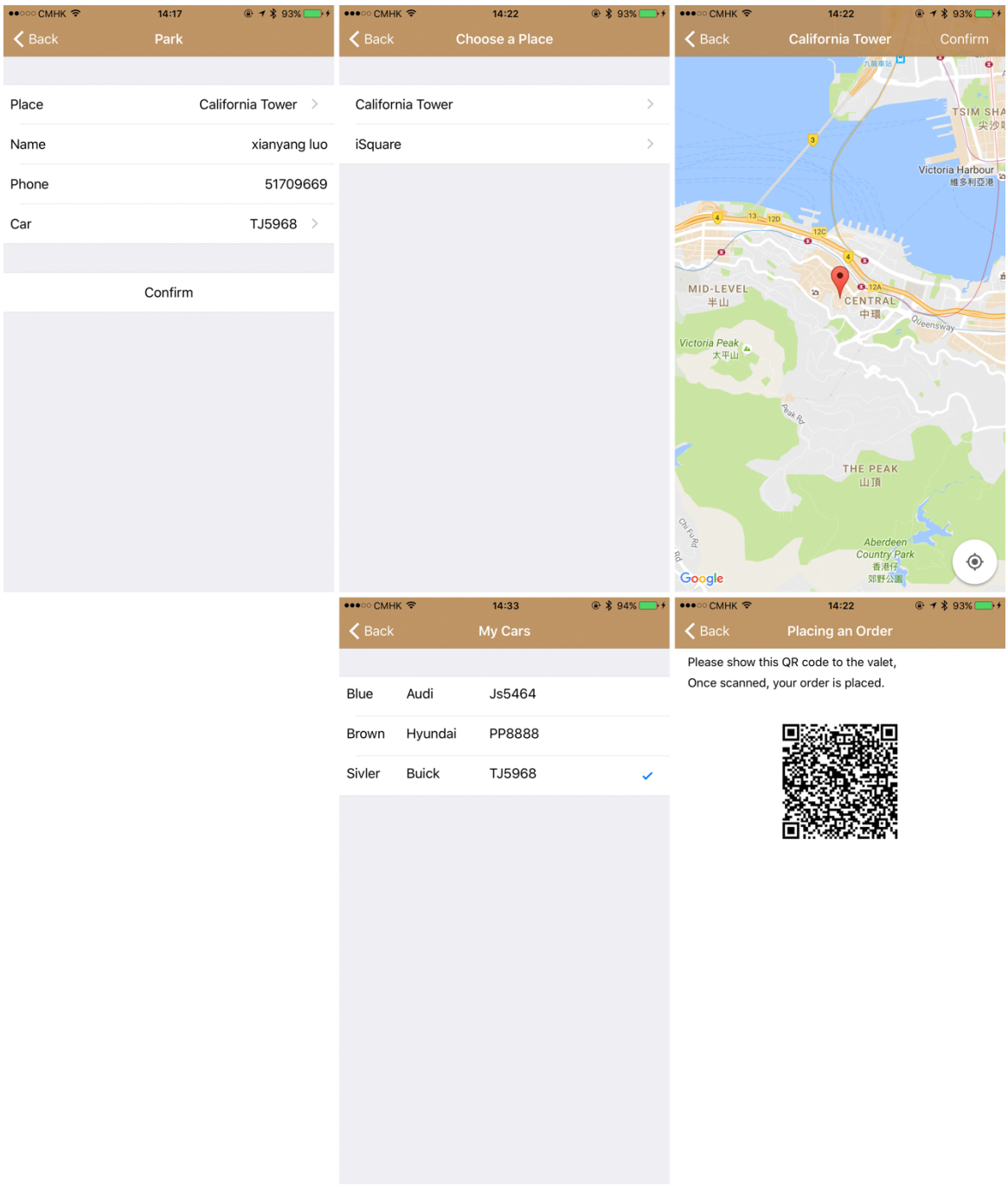


Figure 5-16 park pages

Customers can choose place and choose cars before parking. Once the QR code is scanned by a valet, the order is placed.

4) Order Views

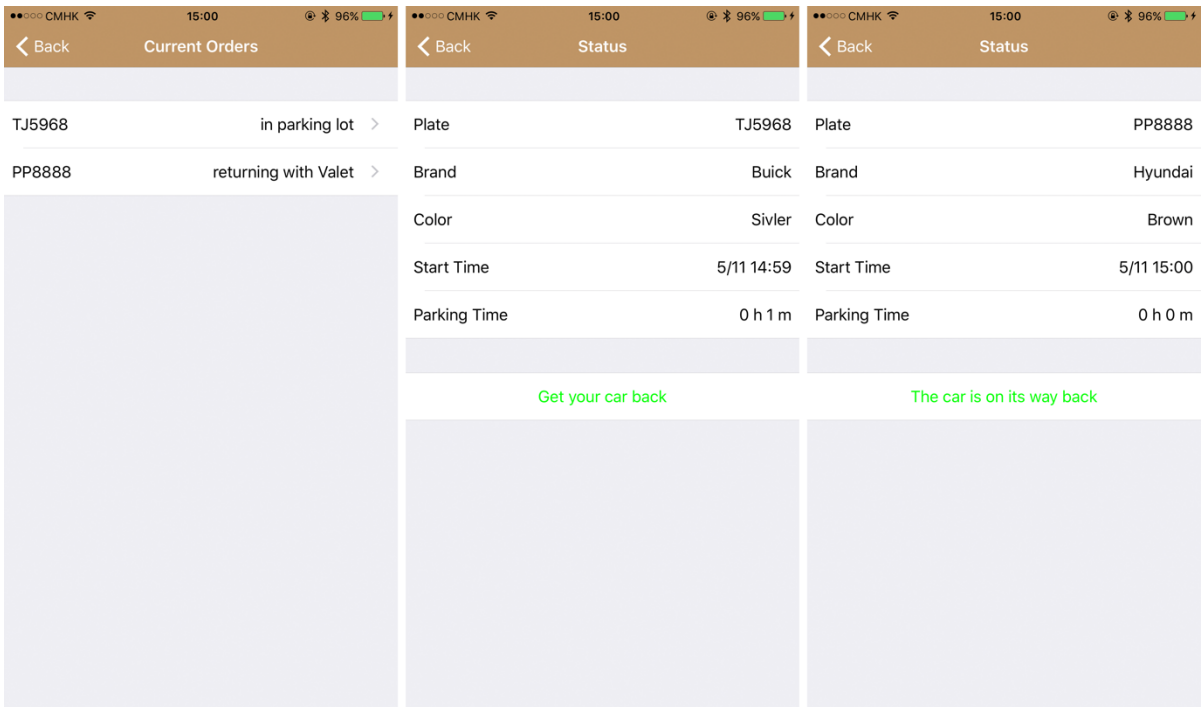


Figure 5-17 order views

In these three views, a customer can check its order and gets its car back by tapping “Get your car back”.

5) Car Views

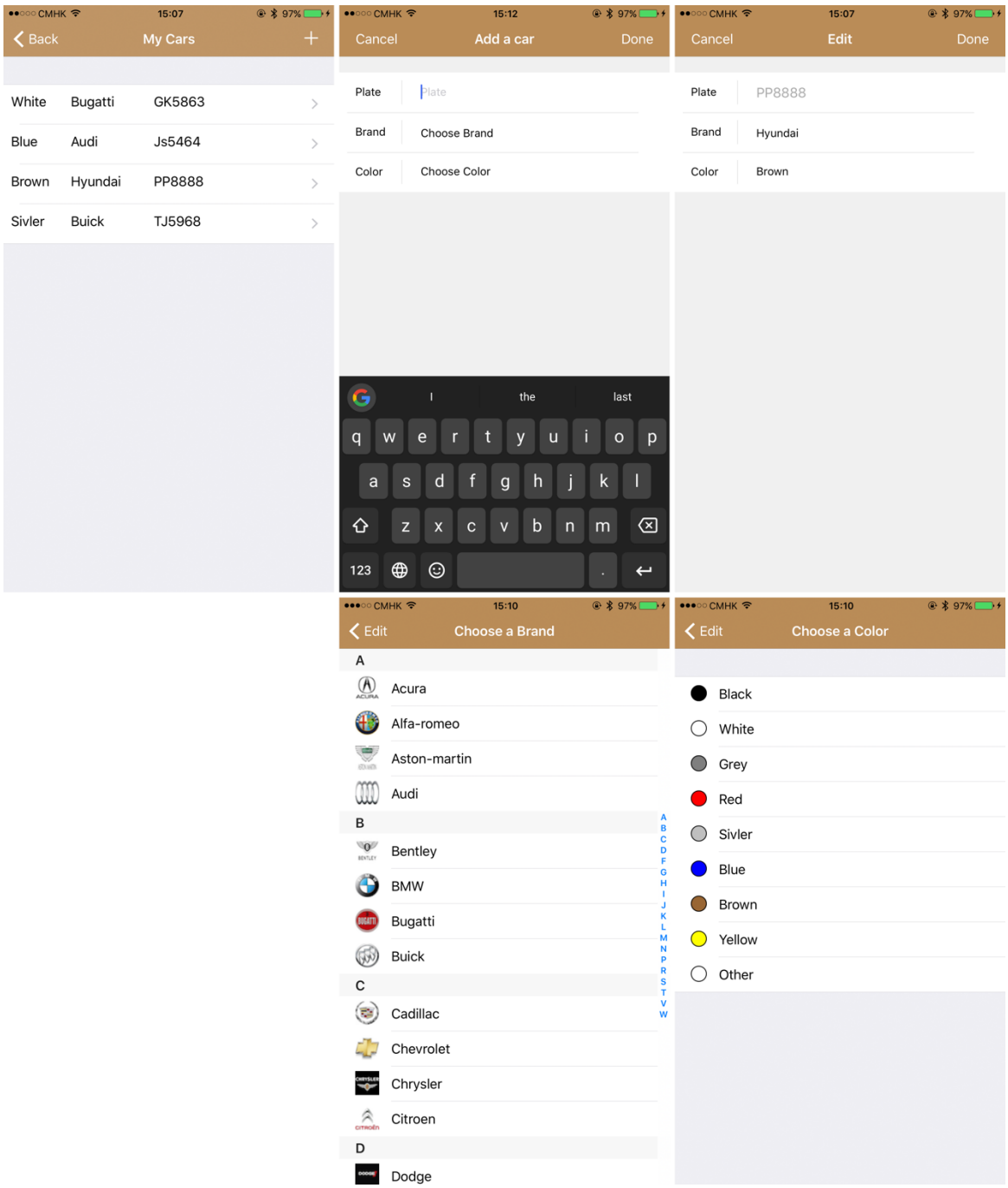


Figure 5-18 car views

In these views, a customer can view, add and update a car.

Customer needs to input car plate and choose brand and color from a list.

6) 3D Touch View

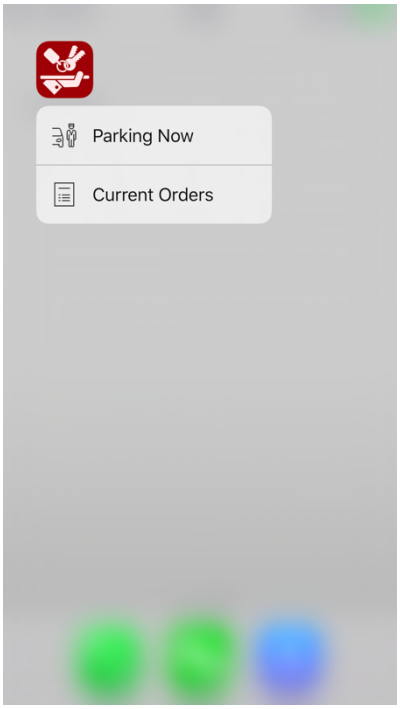


Figure 5-20 3D touch view

Customers can easily access park view and current order view by just pressing the icon of this application from the Home screen. Customers need to login firstly to jump to these two pages. If not, the welcome view will pop up asking for logging in.

The code to launch park view is shown in figure 5-21. As shown in the figure, this code block is written in AppDelegate.m. It firstly get the *tabBarController* and *navigationController*. Then an instance of *ParkingNowViewController* will be initialized. At last the *navigationController* pushes the view to *ParkingNowViewController*.

```
- (void)launchParkingNow {
    // grab our storyboard
    UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];

    // instantiate our tabbar controller
    UITabBarController *tabBarController = [storyboard instantiateViewControllerWithIdentifier:@"MainTab"];

    // instantiate our navigation controller
    UINavigationController *controller = tabBarController.viewControllers[0];

    // instantiate second view controller
    UIViewController *vc = [storyboard instantiateViewControllerWithIdentifier:@"ParkNowViewController"];

    // now push both controllers onto the stack
    [controller pushViewController:vc animated:NO];

    // make the nav controller visible
    self.window.rootViewController = tabBarController;
    [self.window makeKeyAndVisible];
}
```

Figure 5-21 implementation fo 3D touch

### 5.3.2. Valet Client

The valet client is used to generate and manage customers' orders. Because the welcome pages are similar to customer client. So only the main pages are introduced in this part. Figure 5-22 shows the main pages for this client.

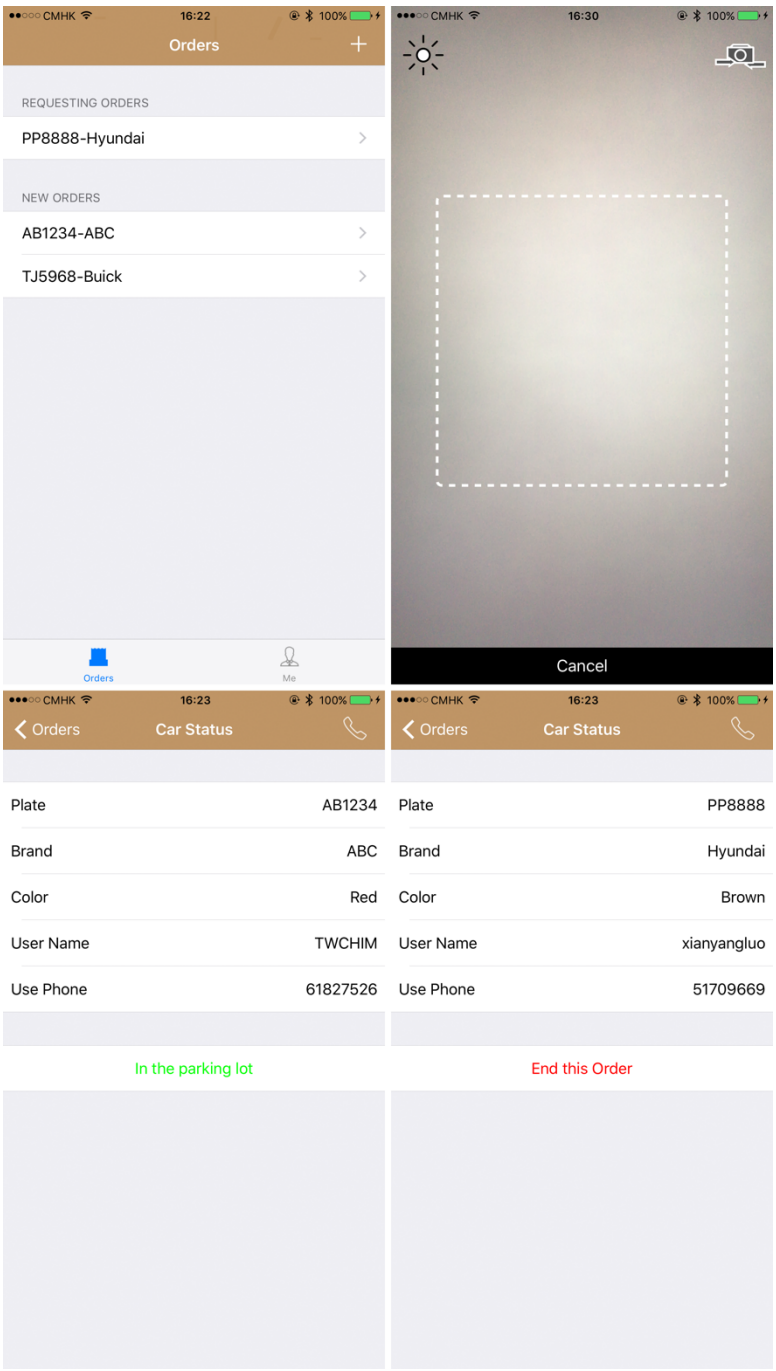


Figure 5-22 main pages for valets

As shown in figure 5-22, there are three types of view for valets to use.

1) Orders View: in this page, valets can view all the unfinished orders. This view is implemented using UITableView.

2) Scanner View: in this page, valets can generate new order by scanning the QR code of a customer. This camera is implemented using QRCodeReader.
3) Car Status View: in this page, valets can view the status of an order and end an order if the car has been returned to its owner. A valet can call a customer using the button on the right top of the view. This page is implemented using UITableView.

# Chapter 6. REVIEW AND FUTURE WORKS

After well-design and implementation, this application can provide convenient valet parking service for drivers in Hong Kong. However, in order to provide better user experience and enhance the performance of the system, there are still some aspects can be improved. I will introduce

## 6.1. Mobile Payment

Since Apple Pay and Android Pay both launched in Hong Kong, more and more people start to enjoy convenient payment method during their daily life. However, most valet parking service in Hong Kong use cash or octopus, which needs to be done immediately when a customer gets its car back. If this application can use mobile payment, customers can pay for the service anytime within a time limit. And they do not need to worry about cash or insufficient balance in octopus.

## 6.2. Easier Process

Although this application offers 3D touch to help a customer place an order quickly, the customer still needs to choose place to park and car to park each time if he or she goes to different place. If this application can use location service, search for the nearest drop-off point and generate the QR code automatically, it is more convenient and time-saving for customers to use this service.

## 6.3. Apple Watch

Apple introduces apple watch in 2015. Now a lot of mobile applications offer their apps in apple watch version. Users can easily use some basic function by raising their hand. I think in the future customers can use apple watch to enjoy this service. They can show QR code to a valet or recall their cars using apple watch, which makes it more convenient.