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: 28 / 4 / 2017

Abstract

Nowadays lots of people drive to hotels, clubs or restaurants after work. One of the big issues for these people is to find a parking lot around the destination. And these customers want to save time and enjoy 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 cannot recall their vehicles in advance.

To solve these problems, this project develops a smart phone application on iOS platform to help customers to use valet parking service. Customers can create a parking order by choosing a location and tapping a button. Customers can monitor real-time locations of valet and vehicle. The application can act as a parking ticket for customers to redeem their vehicles, which is rather convenient for vehicle 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 report.

Parts that are direct quotes or paraphrases are identified as such.

The University of Hong Kong,

LUO Xianyang

Acknowledgements

First of all, I would like to thank my supervisor Dr. T.W.Chim. Dr. Chim gave me advice and support during the whole dissertation. And he could always answer my questions fast and helpful, either through email or whatsapp. Without his great help, I cannot finish this project.

Besides my supervisor, I would like to thank my second examiner Dr. Ronald H.Y. Chung. He gave me creative questions, insightful advice and great guide.

Also I would like to thank my partner XU Xiaodong. I discuss the idea and design with him and I improve the project with his creative ideas.

Last but not least, I must thank my family who gave my great support when I encountered difficulties.

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 [1].

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) must drop off or return at certain point
4) do not know where his or her car is parked and do not know how long it has parked

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 to use our service via its mobile phone number. After signing in, he or she can request dropping its vehicle anywhere in the service area. If there is an available valet nearby, our system will generate an order for user. Customer will get valet's basic information like profile image, name and mobile phone number. And the valet will also get customer's information, the drop off point and a route to that position. When our valet meets the customer, he will show his name card to the user to verify identity. Then the valet will park the vehicle and the customer can do its own business. Whenever the customer wants to get the vehicle back, he or she can just pick a address in the service area and our valets will return the vehicle back to the customer.

This project has three major parts as below:

1) iOS version application for customer
    a. allows a customer to sign up, sign in and reset password via mobile phone number
    b. allows a customer to pick a place in the service area to drop off its vehicle or get its vehicle back

c. allows a customer to check the status of its vehicle

d. shows a route from customer's position to meet point

e. allows the user to view all the historical orders

2) iOS version application for valet

a. allows a valet to login and reset password via mobile phone number

b. allows a valet to see all the current drop off orders and return orders

c. shows customer's basic information

d. shows a route from valet's position to meet point

e. allows a valet to update status of an order

3) server and database

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

b. stores data safely of all users, valets, cars and orders

c. sends notification to a user when the status of an order is changed

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[2], Jun Gong's Guideline for Mobile Application[3] and Nurul's Threes Layers Design Guideline for Mobile Application[4].

Apart from user interface, the app should be rather easy to use. Users do not need to follow a complicate guideline to generate an order[5]. 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 is 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 customers, the application should have friendly and comprehensive user interface, reasonable layout and overall clean look.
2) innovation: customers are more willing to use the application by using new technology like 3D touch. This project allows customer to request service rather fast from the icon which makes it rather convenient and enjoyable.
3) connectivity: a customer may enjoy more convenience if he or she has an Apple Watch[6]. If all the parking and recall request can be done through Apple Watch. A customer does not even need to take his or her phone out the pocket.

1.4. Division of Work

I and XU Xiaodong work as a team. I develop the iOS version of this application alone and XU Xiaodong develops the android version of this application. We develop the back-end including server and database together.

1.5. 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 and the second one is Youbo

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 including requirements, system architecture, workflows, 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 makes a conclusion and discusses about future works.

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 mainland China. I will firstly give a brief introduction and analyze the advantages and disadvantages of both applications respectively.

1.1. 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.

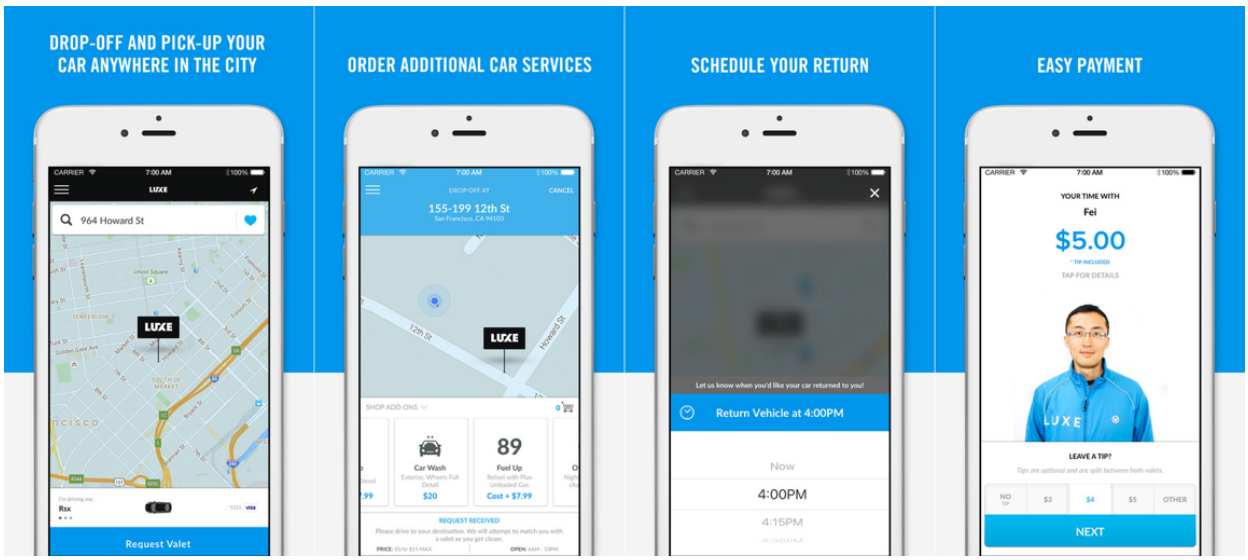


Figure 2-1 iPhone Screenshots for Luxe

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

Advantages:

1) the customer 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 customer saves time since he or she does not need to find a certain drop-off point
2) the customer can pay for the service using its phone, which makes it an 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

1.2. 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) a customer can only drop off and get its vehicle return at certain position but not everywhere around the service area
2) the application does not use new technology like 3D touch and does not offer Apple Watch application
3) the user interface does not look good.

Chapter 3. BACKGROUND KNOWLEDGE

There are a lot of third party frameworks which enhance the performance or to optimize the user interface of this project. This chapter mainly introduces some useful frameworks in the development.

3.1. Google Maps SDK for iOS

https://developers.google.com/maps/documentation/ios-sdk/

Google map is one of the most powerful maps and google map sdk offers map service which is fast to integrate[7]. It powers lots of great application on both iOS and android platform such as UBER, Didi and Runtastic. The user can add map to its application after obtaining an API key. A simple demo of google map with a certain location is shown below:

```
self.mapView = [[GMSMapView alloc] init];
GMSCameraPosition * camera = [GMSCameraPosition
cameraWithLatitude:22.2860547 longitude:114.13799 zoom:16];
self.mapView = [GMSMapView mapWithFrame:CGRectZero
camera:camera];
```

```
self.mapView.delegate = self;
self.view = self.mapView;
```

This code block creates an instance of *GMSMapView*. Then it creates a camera with latitude, longitude and scale size. After setting the camera property of map, the view will show the location according to the coordinate.

In this project, the main view is built on google map. Customer can see its position, vehicle's position, a route from its location to drop off point and so on.

3.2. Google Places API for iOS

https://developers.google.com/places/ios-api/

Google places API offers data from the same database used by Google Maps and Google+ Local. It features over 100 million businesses and points of interest that are updated frequently through owner verified listings and user-moderated contributions[8]. A user can get information of places with place ID or coordinate. A simple demo of google places API is shown below:

```
self.geocoder = [[GMSGeocoder alloc] init];
[self.geocoder reverseGeocodeCoordinate:position.target
  completionHandler:^(GMSReverseGeocodeResponse * response, NSError *
error)
  [self.mapSearchPlaceView setParkAddress:response.results[0].lines[0]];}];
```

The code block creates an instance of *GMSGeocoder*. The *position* variable is an instance of *CLLocation*. The geocoder can use the coordinate of the location to get the address of it. The format of the address is "1-2 Queen Victoria Street".

In this project, I use this API to get address of a location and create a view allowing customers to search an address in Hong Kong.

3.3. AFNetworking

http://afnetworking.com/

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[9]. 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 *para = @{@"foo": @"bar"};
[manager POST: aURL
parameters: para
 success:^(AFHTTPRequestOperation *operation, id response) {
    NSLog(@"JSON: %@", response);
} 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.

3.4. MBProgressHUD

https://github.com/jdg/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

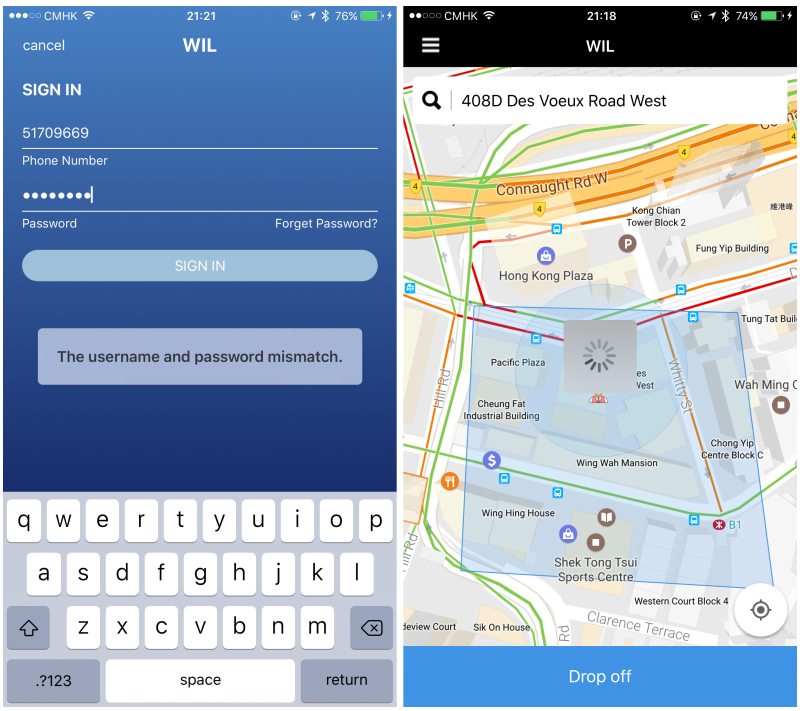


Figure 3-1 Screenshots for usage of MBProgressHUD

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];
});
```

3.5. RESideMenu

https://github.com/romaonthego/RESideMenu

A sliding side menu GUI interface for computerized devices that shows indicia of updated content on at least some of its various menu items[10]. Lots of mobile applications start to hide their menus in the side bar such us Uber, Instapaper, Spark and so on.

RESideMenu is an iOS style side menu with parallax effect. Nowadays many iOS applications in the market has a side menu. Users can open the menu by tapping a button or drag the screen from the edge. The advantage of this design is saving space of the screen since the space is very valuable. In this project, I use RESideMenu to create the menu options for user. User can choose function views like payment, orders and contact from the menu.

3.6. OpenALPR

http://www.openalpr.com/

Automatic identification of vehicles become more and more practical in many applications during the past two decades[11-12]. Using a license detection can help valet record information of a vehicle more

efficiently[13]. What's more, we can also add features like model detection and color detection.

OpenALPR is free service running on cloud for vehicle detecting[14]. The service use image data as input and responds with license plate information, vehicle color, vehicle model and so on. The advantage of this service is light and easy to configure. In this project, valet needs to take a picture of the vehicle after parking. And this service will detect basic information of the vehicle including plate, model and color.

Chapter 4. REQUIREMENTS AND DESIGN

In this part, I will introduce the requirements and design for this project. At first, 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

I develop the front-end and back-end of this project. The front-end is built on Xcode 8 using Objective-C and the back-end is built with LeanCloud. And the front-end has two different applications on iOS platform. One is for customers and the other one is for valets. Customers can use their mobile phone number to sign up, sign in and reset password. And valets can use their mobile phone number to sign in and reset password.

4.1.2. User perspective

There are two types of user in this project: customer and valet. Customers are who uses our valet parking service and valets are who park and return vehicle for customers.

After a customer signing in, he or she can see available valets around, choose anywhere in the service area to park, check its order status, check valet's information if he or she has an unfinished order, request its vehicle back anywhere in the service area and other basic settings. After a valet signing in, he or she can access to all opening orders related to him or her, check customer's information and location, get an route to the drop off or return point, update the status of an order and other basic settings.

4.1.3. Functional Requirements

Hers is a list of high-level functional requirements that this project is focus on. The back-end must:

1) Authenticate and authorize a customer or a valet
2) Store a number of data records describing customers and valet. Each record will have attributes below:

- First name
- Last name
- Mobile phone number
- Profile image URL

3) Store a number of data records describing customers' locations and valets' locations. Each record will have attribute below:
    - User's identifier
    - User's coordinate
    - User's status
4) Store a number of data records describing orders. Each record will have attribute below:
    - Customer's identifier
    - Drop off valet's identifier
    - Return valet's identifier
    - Create time
    - End time
    - Drop off location
    - Return location
    - Order status
5) Store photos and URLs of photos
6) Return corresponding response to front-end

The front-end must:

1) Allow customers to view its location and valets nearby
2) Allow customers to choose a place in the service area to drop off or return its vehicle
3) Show the route from customer's location to meet point
4) Allow customers to check order status
5) Allow customers to view valet's basic information and call valet
6) Allow valets to view all the unfinished orders related to him or her
7) Allow valets to check customer's basic information
8) Allow valets to update order status
9) Show the route from valet's location to meet point

Next I will introduce the details of design.

4.2. Design

4.2.1. System Architecture

The entire system can by divided into two parts: front-end and back-end. Back-end is a server and database based on LeanCloud. It handles all the http request including signing up, signing in, fetching valets' locations, uploading customers' locations, creating an order, updating an order, sending notifications to user and so on. 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 entire system of this project.

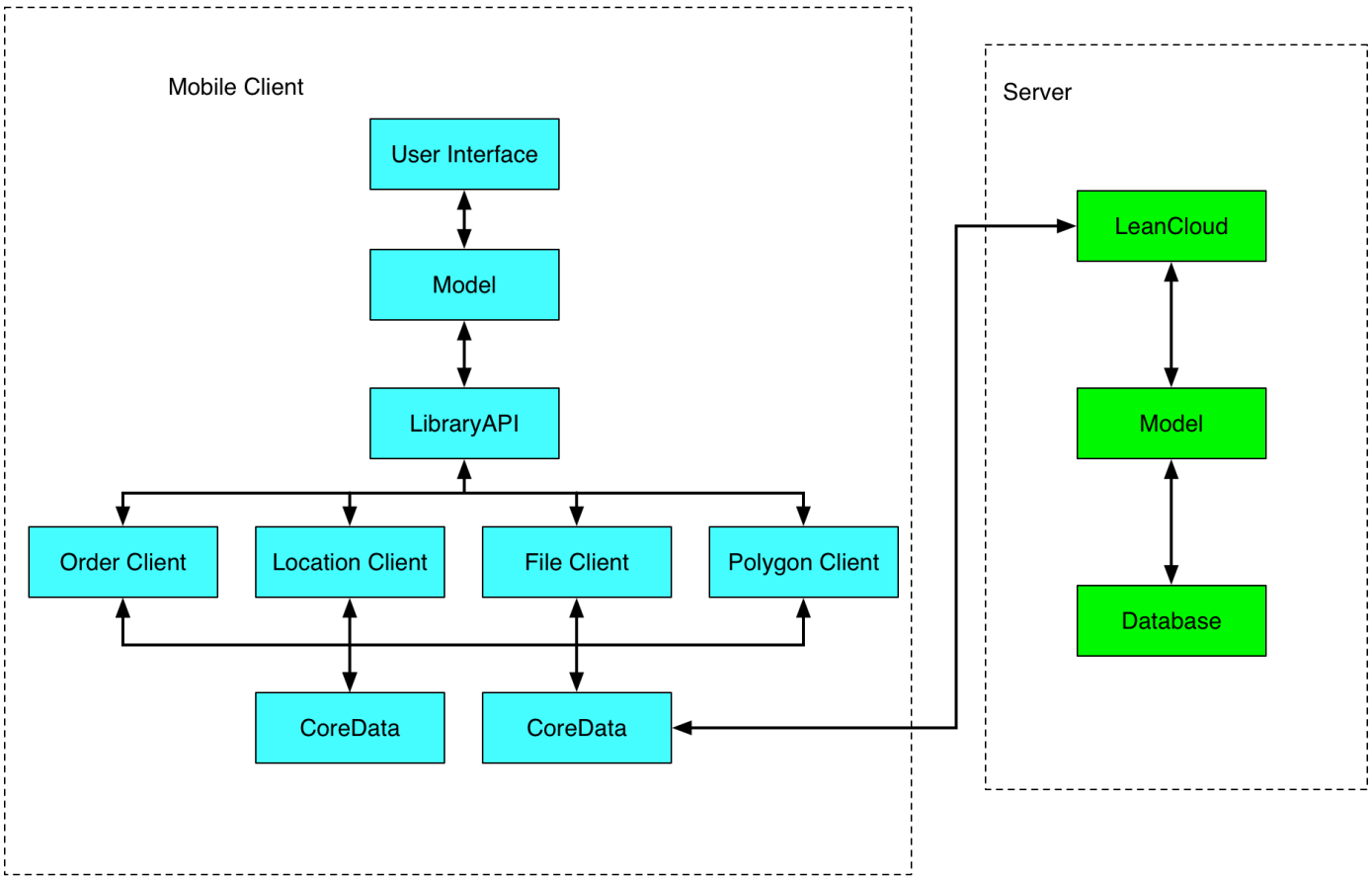


Figure 4-1 Architecture of System

4.2.2. System Workflow

The system workflows consist of two parts. The first part is the workflow in the back-end and the second part is the work flow in the front-end. These two parts will be discussed separately.

4.2.2.1. Workflows in Back-End

Server works as back-end in this project and will handle all the http request sent from front-end. There are three major parts of

management in server: account management, location management and order management.

1) Flow chart for account management

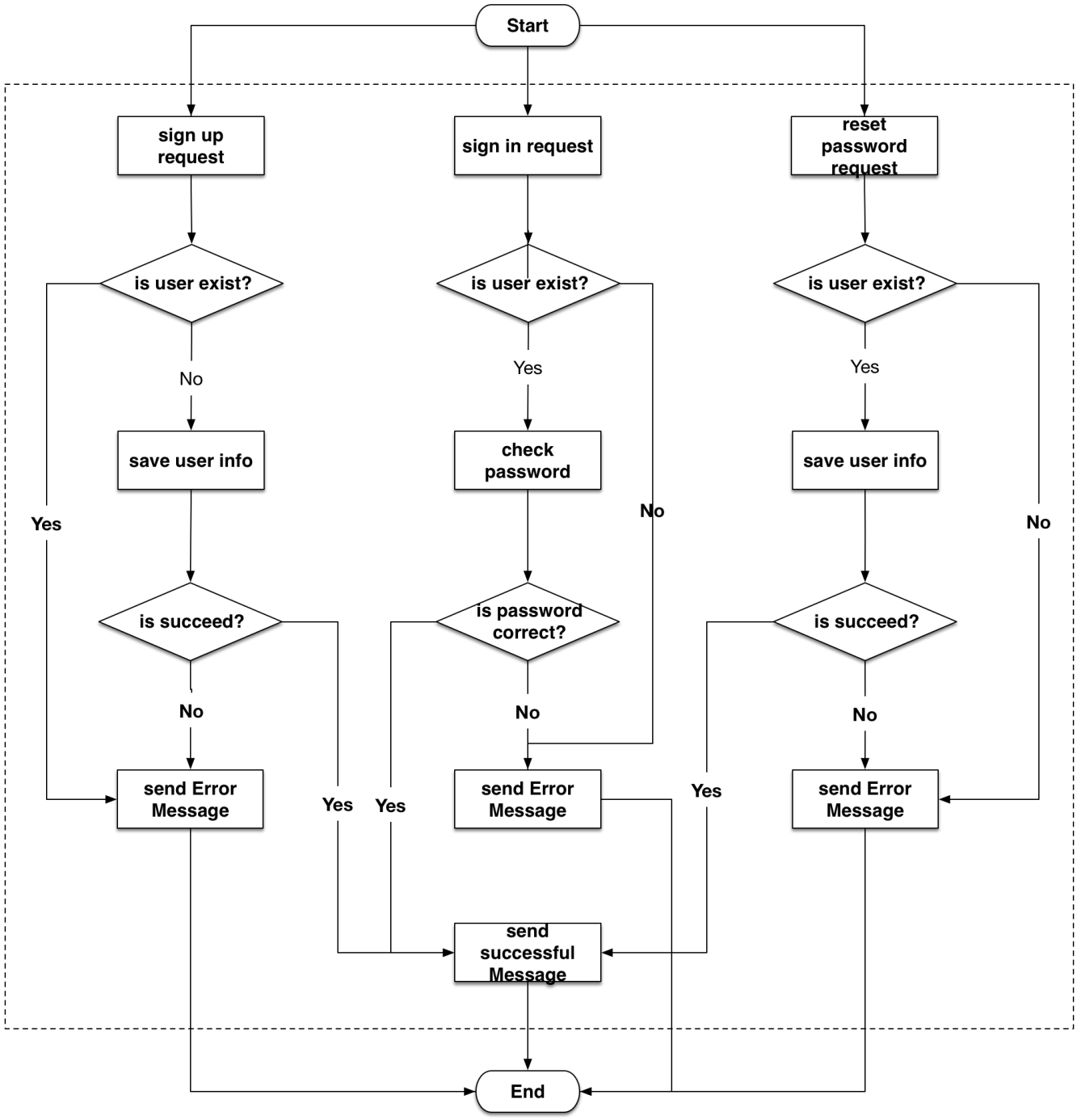


Figure 4-2 Flow chart for account management

As shown in figure 4-2, server needs handle three different request related to accounts management: sign up request, sign in request and reset password request.

For sign up request, if there exist an account with the same mobile phone number or user name, the server will send error message back to front-end. If sign in process is success, the server will send an *User* object back to front-end.

For sign in request, if the user enter the correct account and password, the server will send a successful message back to front-end. If the account and password mismatch or cannot find the user, the server will send the corresponding error message back to front-end with the *User* model. Then user needs to sign in again.

For reset password request, if the user enter the correct verification code and a new password, then the server will change renew user's password and return a successful message along with the *User* model. If the server cannot find the user or the verification code is wrong, then it will send back a corresponding error message.

2) Flow chart for order management

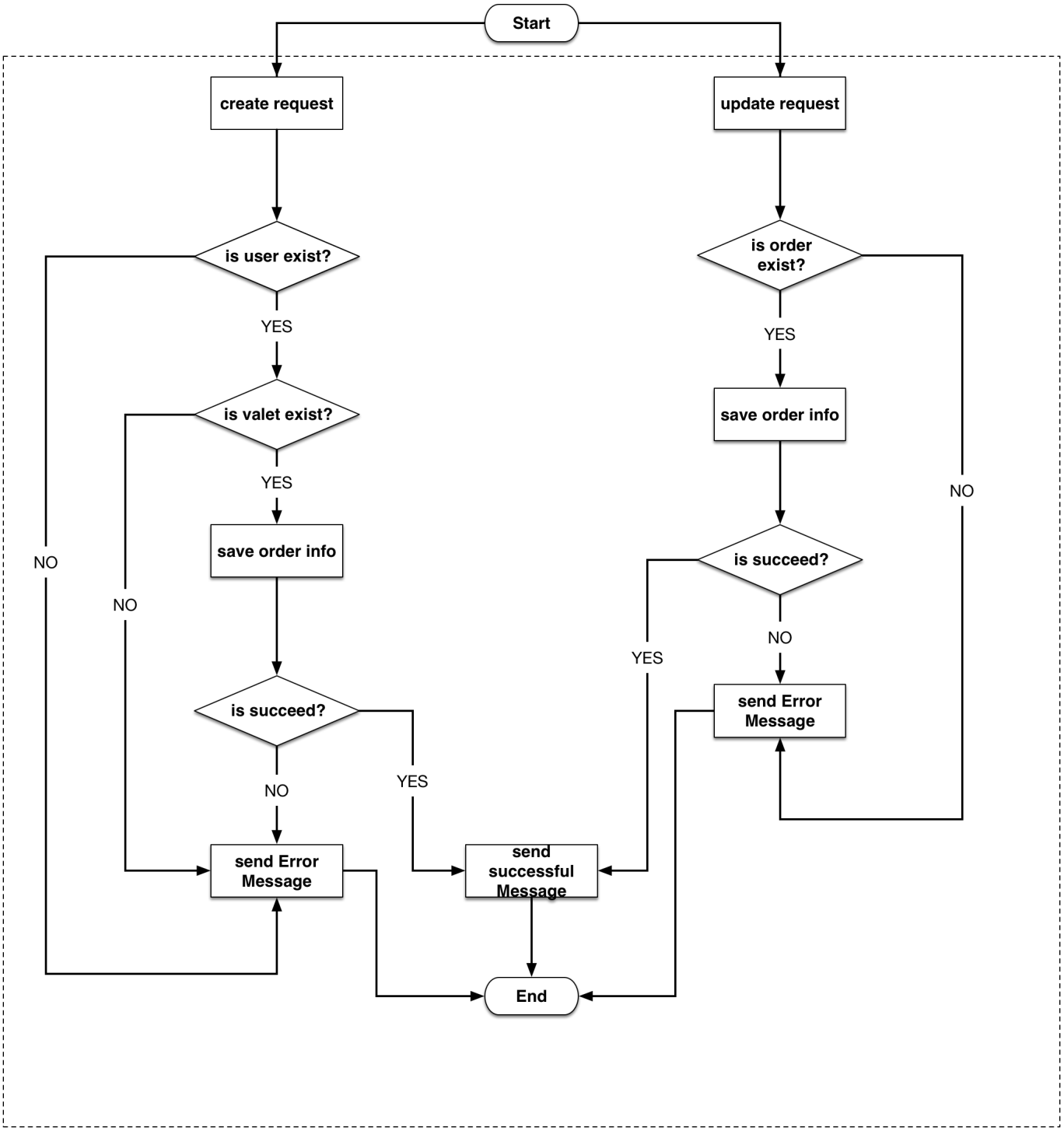


Figure 4-3 Flow chart for order management

As shown in figure 4-3, the server can handle two kinds of order management request: create and update.

The creating order quest has user's information, valet's information, drop off point coordinate. So the server will firstly check if the customer exists, if so, it will check if the valet exists. Final step is save this order in the database. If saving process is successful, server will send the order object back to front-end. If any step occurs error, server will send corresponding error message back to front-end.

After getting the order object, the front-end will save this object locally for later use. If a customer or a valet wants to update an order such as cancel or end an order, the front-end will send an update order request along with the order identifier and next order status. After getting the request, the server will firstly check if the order exists, if so, the server will update the status of the order. If saving process is successful, server will send the order object back to front-end. If cannot find the order cannot save the order, the server will send an error message back to front-end.

3) Flow chart for location management

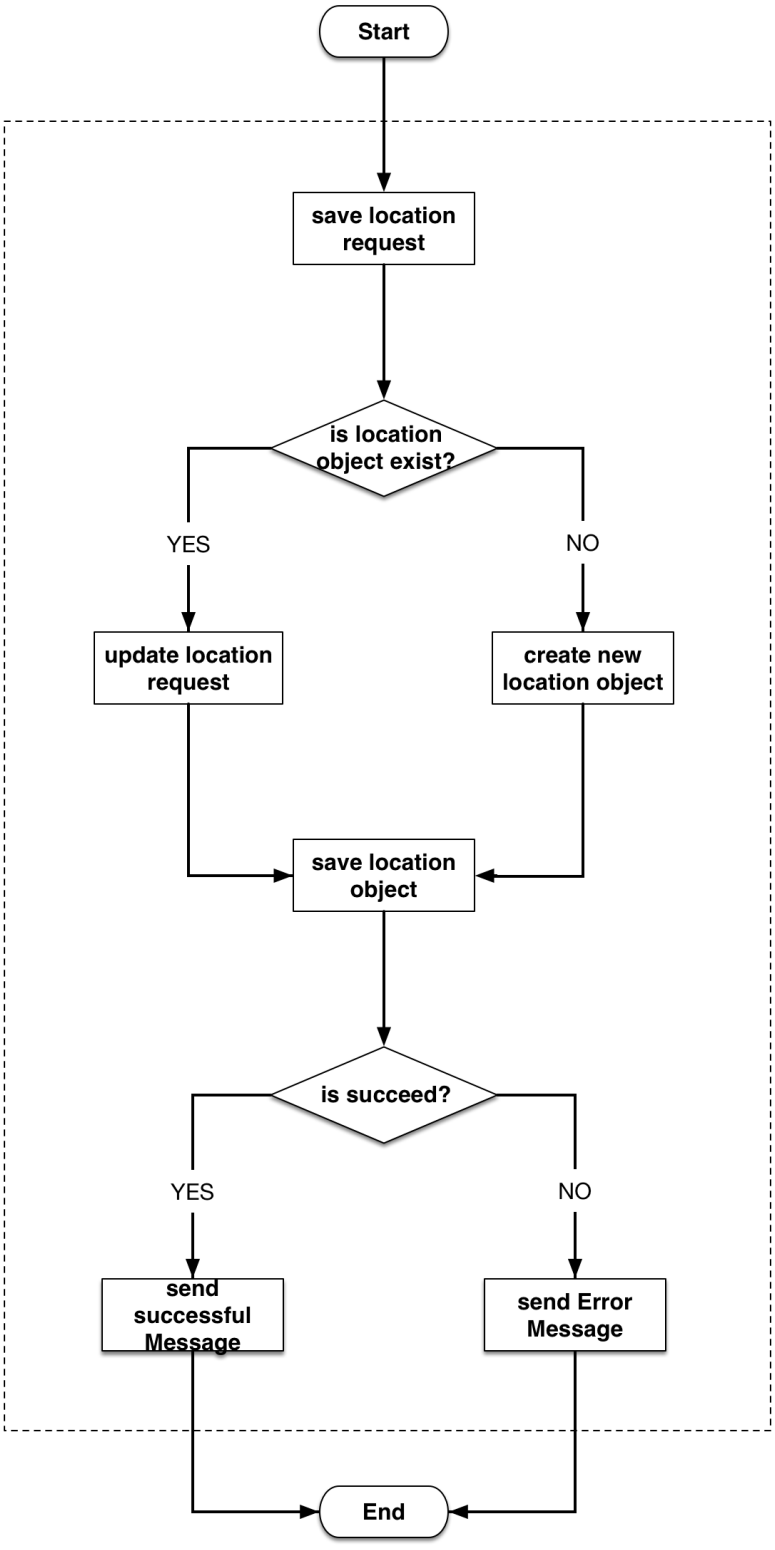


Figure 4-4 Flow chart for location management

When a customer or a valet opens the application, their locations are uploaded to server every 10 seconds. Each user has only one location object on the server and the location object is updated in real time. As shown in figure 4-4, when the server gets a request of saving location object, if will firstly check if the location object exists in the database. If so, it will update the location object with the new coordinate. If not, it will create a new location object and save all the information in the database. If there is no error, the server will send a response to front-end with the location object. If not, the server will send a corresponding error message to front-end.

4.2.2.2. Workflows in Front-End

The two mobile applications work as front-end in this project. One is for customers and the other one is for valets. There are three major managements in the front-end: account management, order management and location management. Figure 4-5 shows the flow chart for front-end.

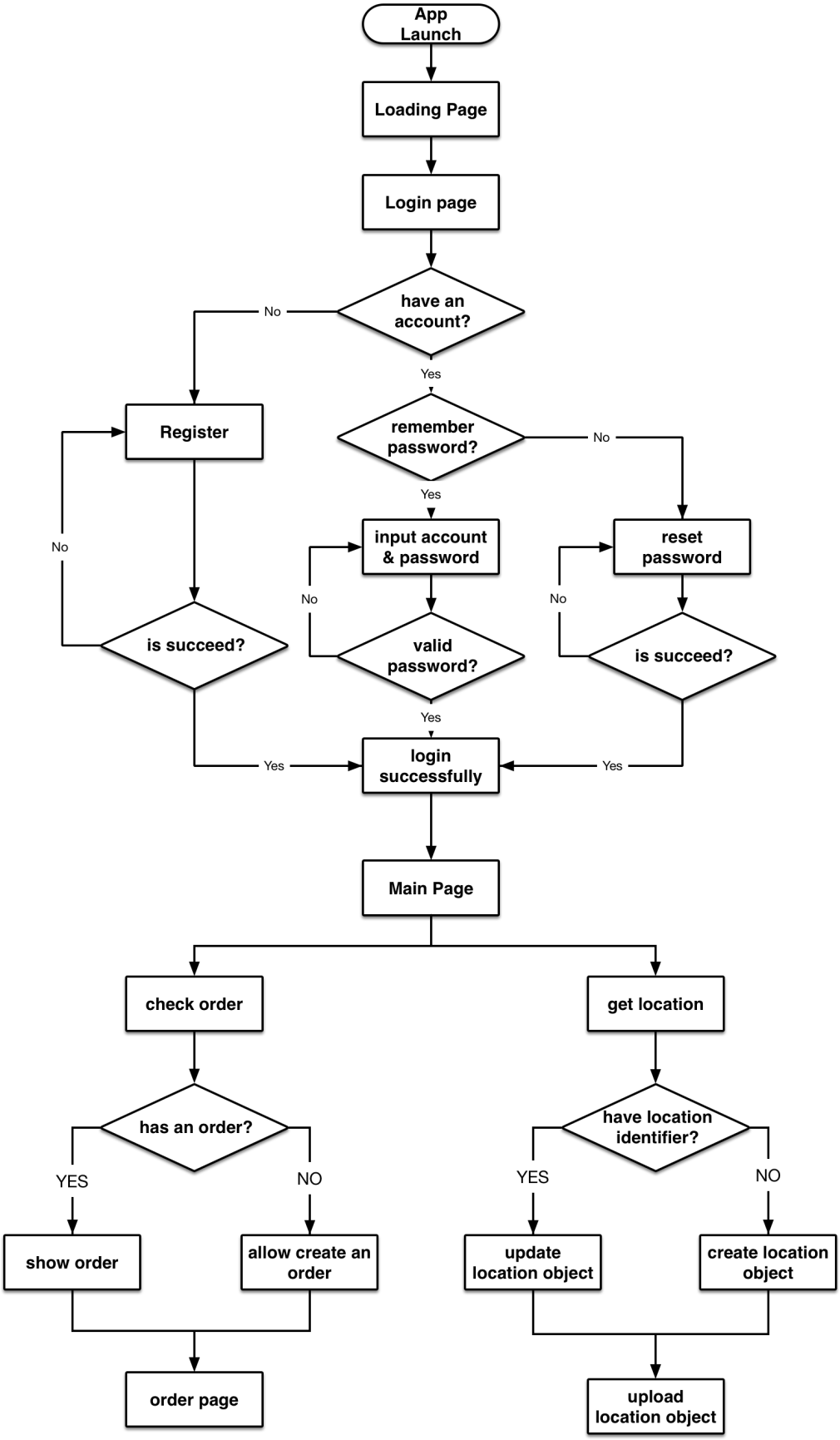


Figure 4-5 Flow chart for front-end

I. Account Management

When the application is launched for the first time, it will show an introduction page and ask user to sign up or sign in. If the user does not have an account, he or she needs to sign up using its mobile phone number. Our system will send a verification code to that phone and user needs to fill in basic information. After successfully signing up, the instruction view will disappear and main view will show. If the user has an account, he or she can sign in using its mobile phone number and password. After successfully signing in, the main view will show up. If the mobile phone number and password mismatch, the application will ask the user to input mobile phone number and password again. If the user forgets its password, he or she can use its mobile phone number to get a verification code to reset password.

II. Order Management

In the main view, a customer can park its vehicle. An order has 10 statuses:

- Undefined: the application does not know if the user has an unfinished order

- None: the user does not have an unfinished order
- Dropping off: the user is dropping off it vehicle
- Parking: a valet is parking user's vehicle
- Parked: the vehicle is parked
- Requesting back: the user wants its vehicle back
- Returning back: a valet is returning user's vehicle back
- Waiting for payment: service is finished and waits for user's payment
- Finished: user pays the bill and the order is finished
- Cancel: the user cancels the order when dropping off

When the application for customer is launched, it will ask the server if the user has an unfinished order. During this time, the status of order is *undefined*. If the user does not have an unfinished order, the order's status will change to *none.* And the user can request our service now. The flow chart for order status is shown in figure 4-6.

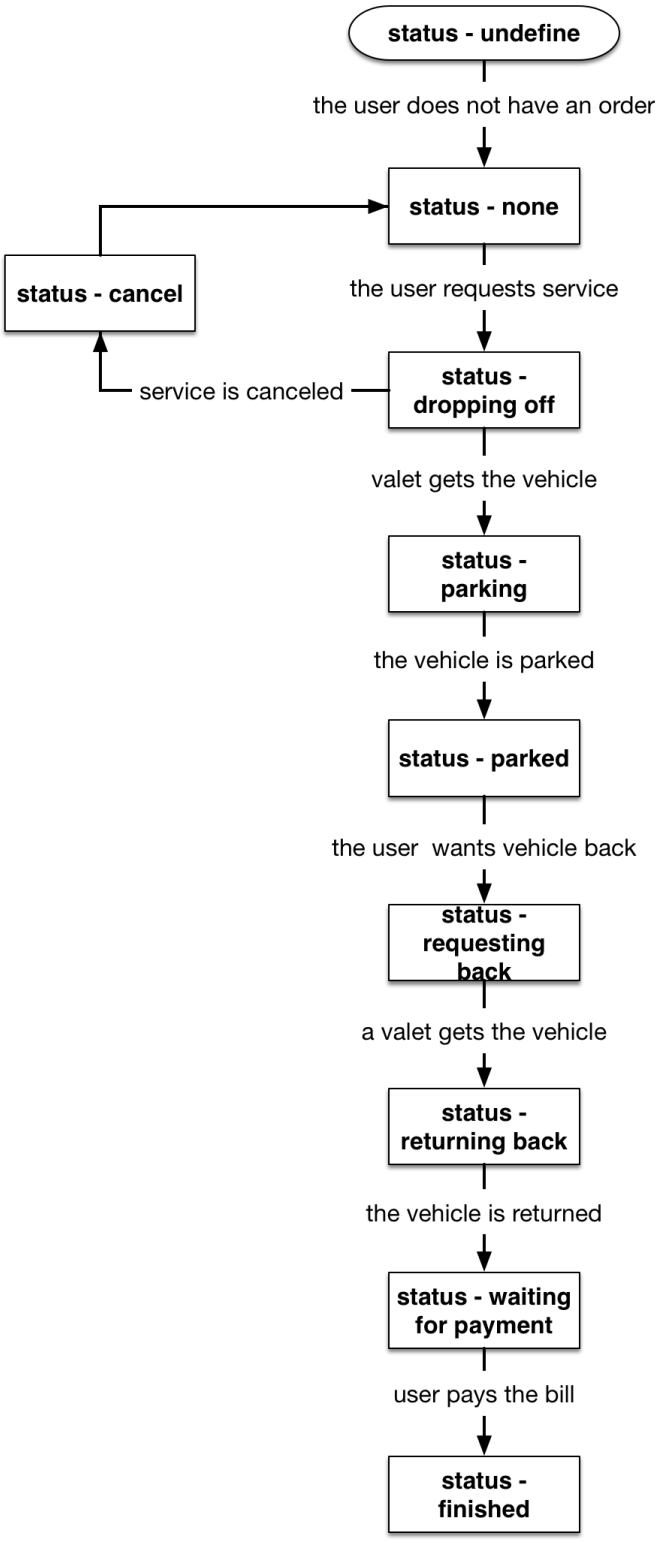


Figure 4-6 Flow chart for order status

III. Location Management

The location object in server saves users' and valets' locations. Attributes for each location object are:

- User's name
- User's mobile phone number
- User's identifier
- User's coordinate
- User's status (only for valet)

When the applications for customers or valets are launched, they will start to get user's location. And applications will check if the user has a location object on the server, if so, they will get the identifier of that object. When the iPhone pass user's location to our applications, they will update the location object if the user already has an location object or create a new location object and upload it to server if not.

The location object for valet has an attribute *status*. This attribute indicates the status of the valet. It can be *busy* or *available*. *Busy* means the valet is parking to returning a vehicle. And *available* means the valet is ready for service.

4.2.3. Object Oriented Design

Below I will introduce the object oriented design of 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	Sign in	Customer signs in
Customer	Sign up	Customer signs up the service
Customer	Reset password	Customer resets its password
Customer, valet	Park a vehicle	Customer parks its vehicle
Customer, valet	Return a vehicle	Valet returns customer's vehicle

4.2.3.2. Use case diagrams

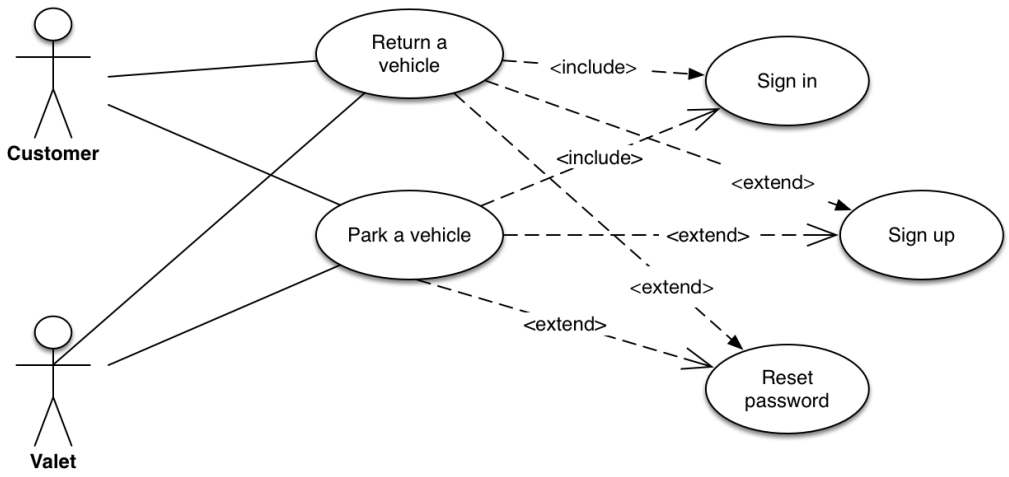


Figure 4-7 Use Case Diagram

Figure 4-7 shows the use case diagram. As it shows in the picture, there are four use cases as below:

1) For the “park a vehicle” use case, a customer can use our service anywhere in the service area if he or she does not have an unfinished order and there are valets available around.
2) For the “return a vehicle” use case, a customer can request its vehicle back anywhere in the service area. Our system can choose a nearest valet to serve the customer.
3) For “sign in” use case, it is included in the “park a vehicle” use case
4) If the user does not have an account, the “sign up” use case will be invoked
5) If the user forgets its password, the “reset password” use case will be invoked. User can use its mobile phone number to reset password.

4.2.3.3. Sequence diagrams

1) Figure 4-8 shows the sequence diagram for “sign up”. All the messages in this sequence diagram are asynchronous.

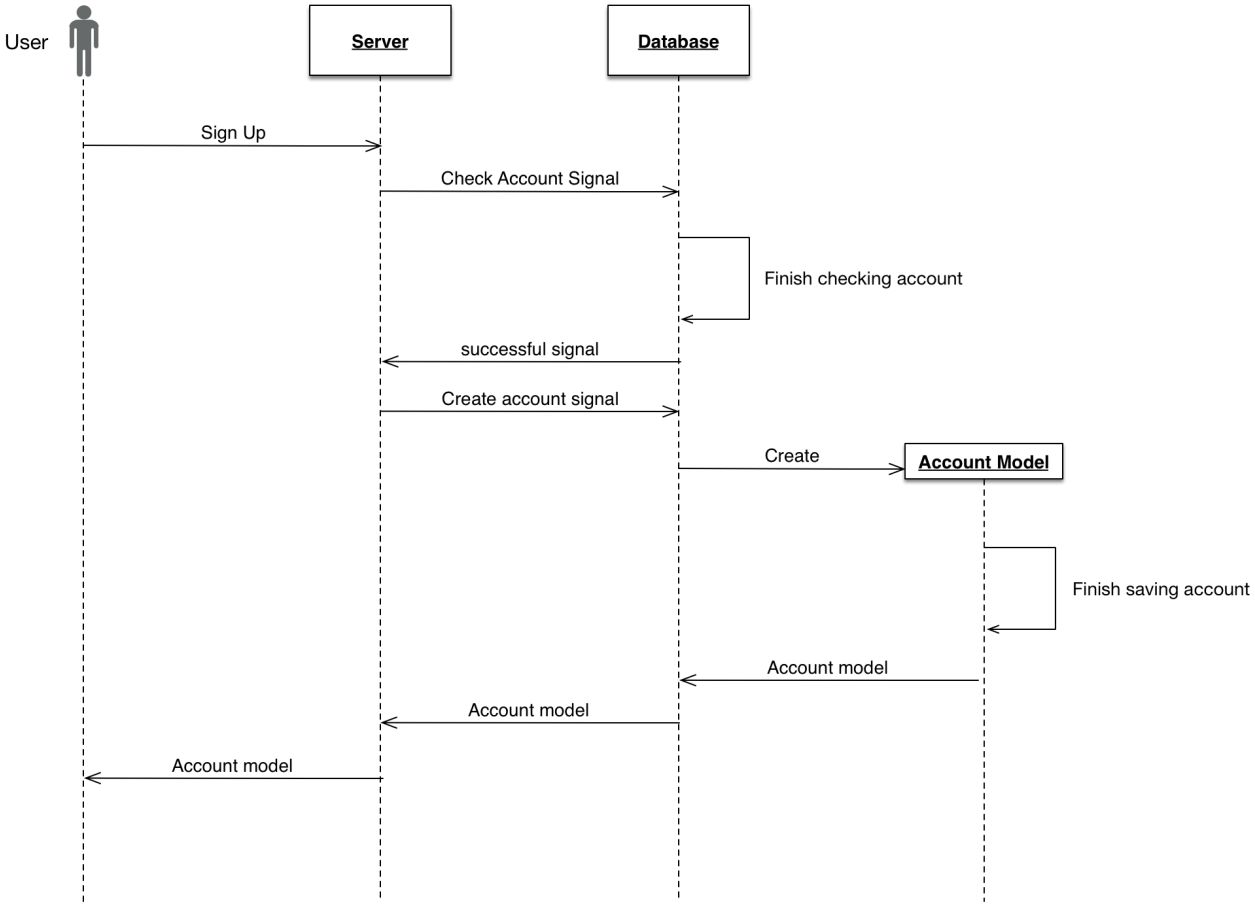


Figure 4-8 Sequence diagram for “Sign Up”

As shown in the figure 4-8, customer will firstly send a sign up request to server with its information including name, mobile phone number, password and verification code. And the server will check with the database if the information is valid. If so, database will create an account and server will send this account object back to front-end.

2) Figure 4-9 shows the sequence diagram for “sign in”. All the messages in this sequence diagram are asynchronous.

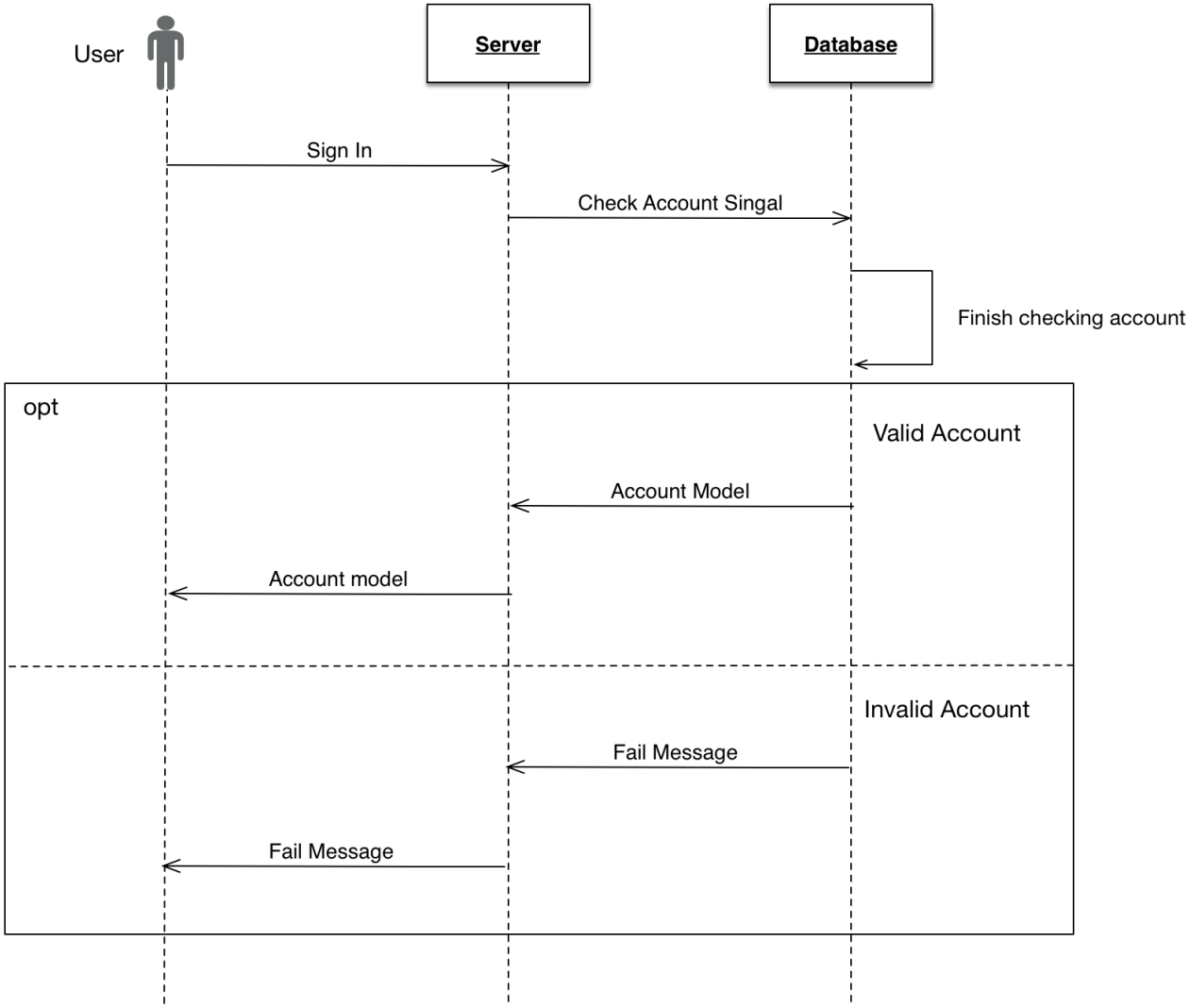


Figure 4-9 Sequence diagram for “Sign In”

As shown in the figure 4-9, a user can sign in using its mobile phone number and password. Server will check the validity of the account with the database. If the account is valid, server will send back the account object. If not, server will send back a corresponding error message.

3) Figure 4-10 shows the sequence diagram for “reset password”.

All the messages in this sequence diagram are asynchronous.

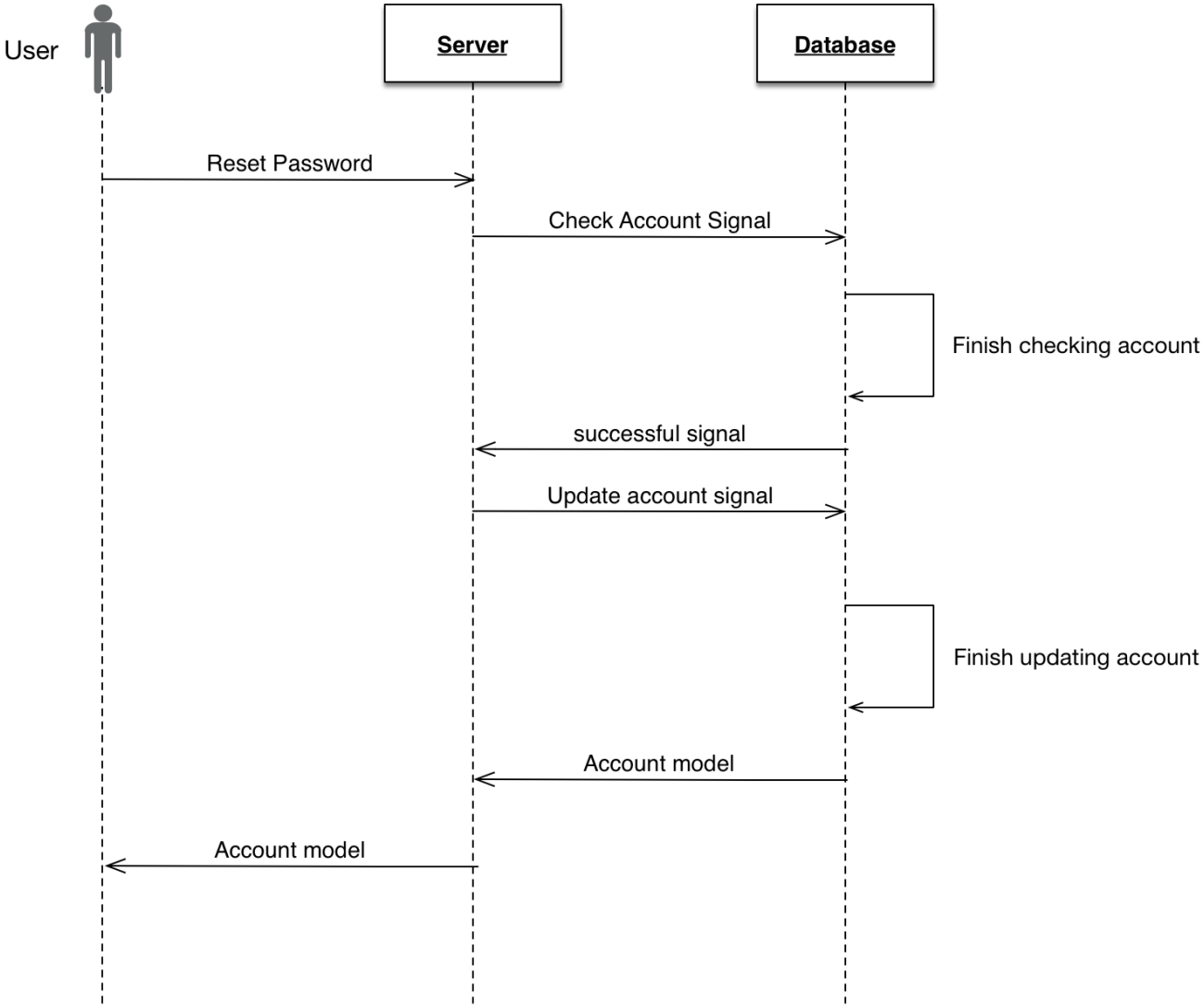


Figure 4-10 Sequence Diagram For “Reset Password”

As shown in figure 4-10, a user can reset password using its mobile phone number and a verification code. If the account and information is valid, the server will update user’s password and send a new account object back to front-end.

4) Figure 4-11 show the sequence diagram for “park a vehicle”. All the messages in this sequence diagram are asynchronous

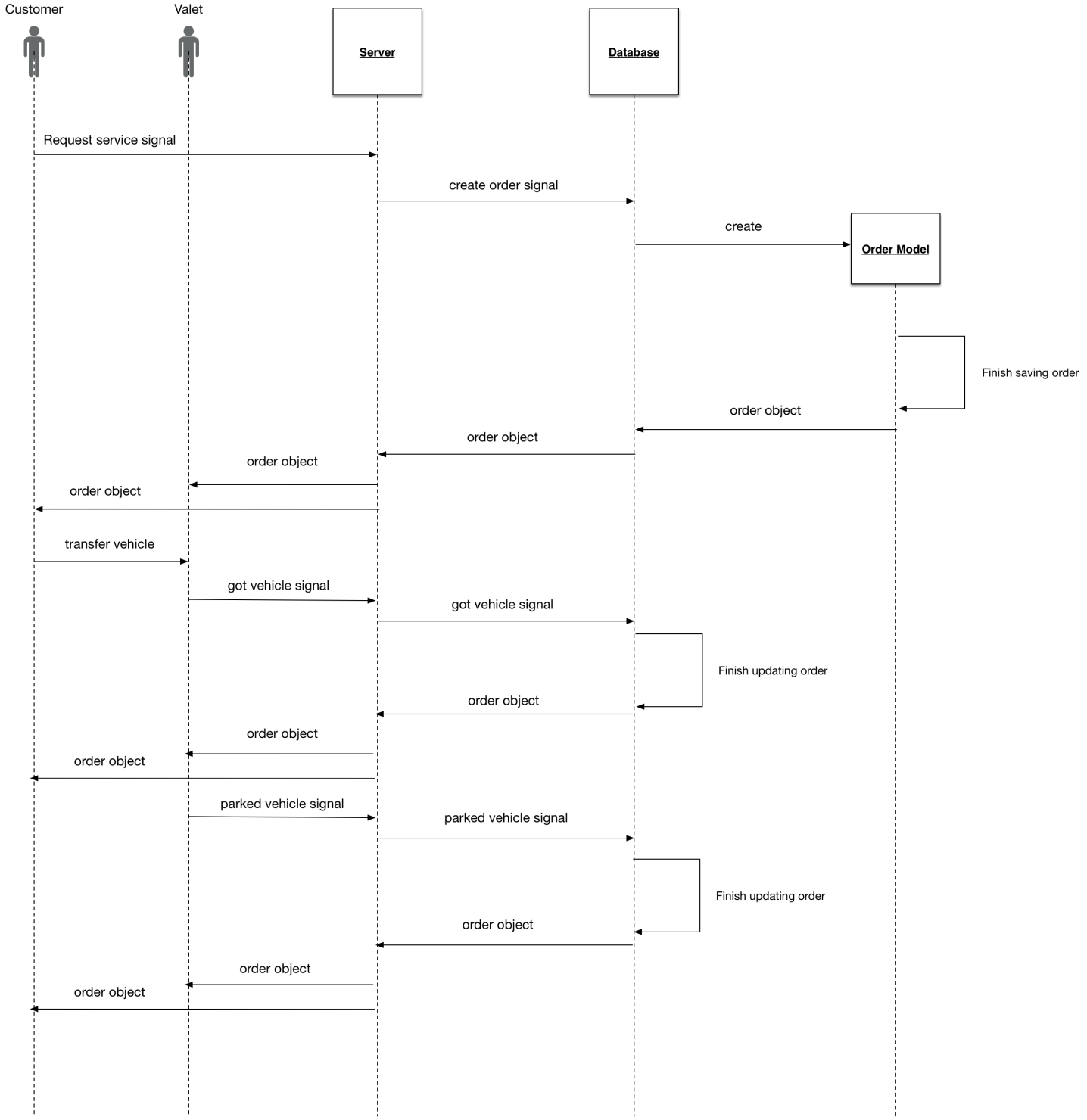


Figure 4-11 Sequence diagram for “Park a vehicle”

As shown in the figure 4-11, the use case “park a vehicle” contains steps below

I. A customer requests service and send a request to server.

II. Server checks the account and creates an order for this customer. If the order is created successfully, the order object will be sent to both customer and valet

III. Customer transfers its vehicle to valet

IV. Valet gets the vehicle and sends a request to server to update order status

V. Server checks and updates the order. Then it will send the updated order to both customer and valet

VI. Valet parks the vehicle and sends a request to server to update order status

VII. Server checks and updates the order. Then it will send the updated order to both customer and valet

5) Figure 4-12 show the sequence diagram for “return a vehicle”.

All the messages in this sequence diagram are asynchronous

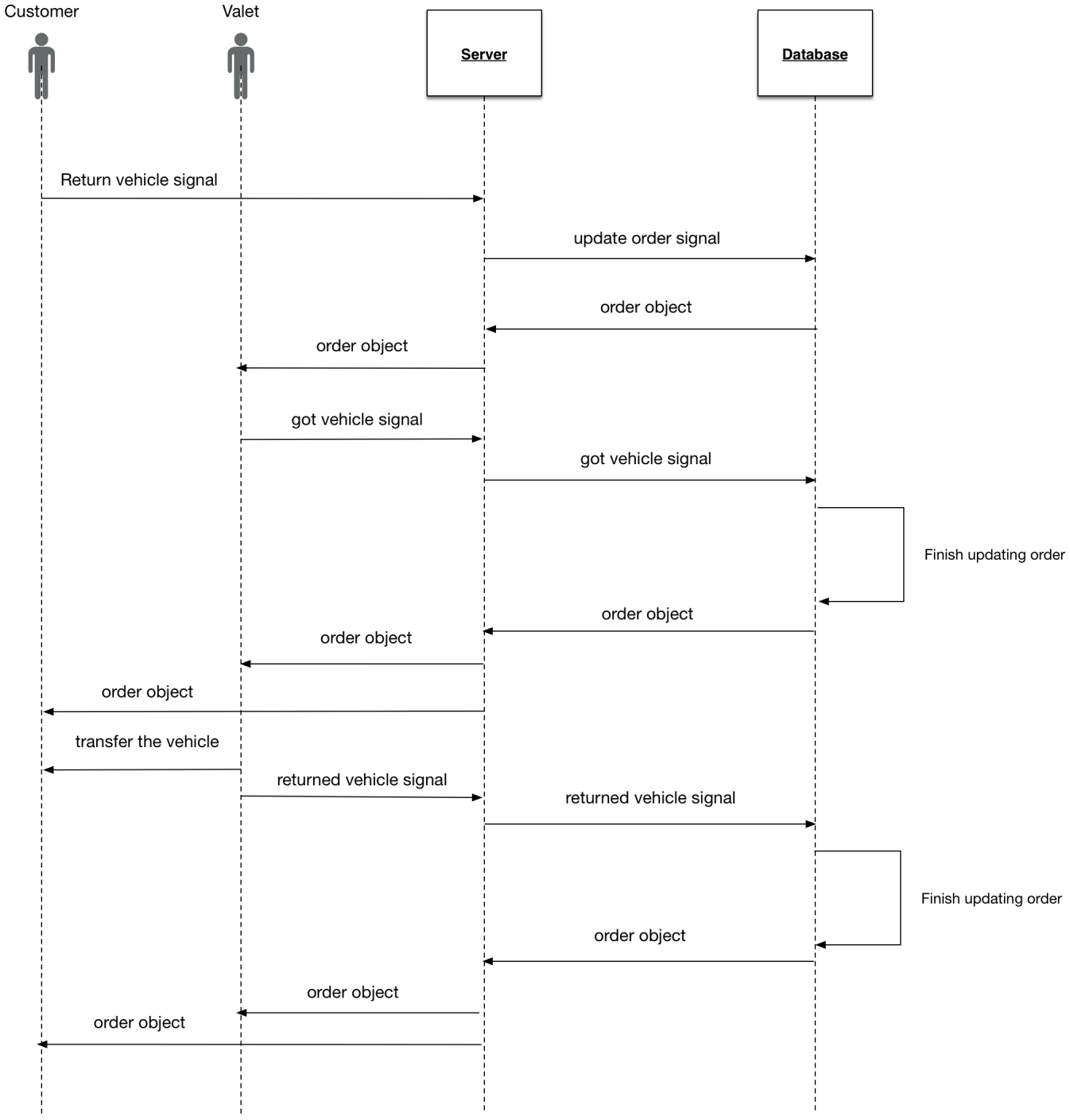


Figure 4-12 Sequence Diagram for “Return A Vehicle”

As shown in the figure 4-12, the use case “return a vehicle” contains steps below

I. A customer wants its vehicle back and sends an request to server
II. Server gets the request, checks the order and assigns the nearest valet for this customer. If the order is updated successfully, the updated order object will be sent to both customer and valet
III. Valet gets the order and he or she will go to parking lot. When the valet gets the vehicle, he or she will send a request to server to update the order status
IV. Server checks and updates the order. Then the updated order will be sent to both customer and valet
V. Valets transfers the vehicle to customer. Then the valet will send a request to server to update the order
VI. Server checks and updates the order. Then the updated order will be sent to both customer and valet
VII. Customer pays the fee using Apple Pay then the app will send a request to server to update the order is payment is successful

4.2.3.4. Documents for design use cases

Below introduce the documents for the five use cases.

1) "Sign up" use case

<table>
<tr><td>Use Case Name:</td><td colspan="2">Sign up</td></tr>
<tr><td>Actor(s):</td><td colspan="2">Customer</td></tr>
<tr><td>Description:</td><td colspan="2">This use case describes the membership registration process.</td></tr>
<tr><td rowspan="4">Typical Courses of Events:</td><td>Actor Action</td><td>System Response</td></tr>
<tr><td>The welcome window is currently displayed on the screen waiting for the customer to select an option.</td><td></td></tr>
<tr><td>Step1: Initiate this use case if a customer clicks the [Continue with Phone] button on the bottom.</td><td></td></tr>
<tr><td></td><td>Step2: The system pops up a multi-lined window requesting the following information to be input: First Name, Last Name, Phone Number and Password. The window has one buttons at the bottom: [Next], one button at the left top corner: [Cancel]. And the customer can read</td></tr>
</table>

		Terms by clicking the [Terms] button.
	Step3: The customer inputs First Name, Last Name, Mobile Phone Number and password then clicks the [NEXT] button.	
		Step4: The application sends the customer's information to server. The server checks the information. If the information is valid, server will create a new customer object and send this object to application along with a verification code by text message.
		Step5: the application gets the successful message and push into a new view. This view request customer to input verification code. And there is one button at the bottom: [NEXT] and one button on the right top corner: [SKIP]
	Step 6: The customer inputs the verification	

	code and then clicks the [NEXT] button.	
		Step 7: A progress indicator will display on the screen. The application sends the verification code to server to check. If the code is correct, server will send a successful message back to front-end.
		Step 8: this use case concludes when the mobile client receives a successful message. 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 information is invalid, server will send an error message back to front-end. Customer needs to correct the number or get verification code again. This use case goes back to Step3. Step6: if the customer clicks the [SKIP] button. The view asking for verification code will dismiss and the main view will show. Customer's mobile phone number is not verified but he or she can use our service. This use case concludes. Step7: if the code is incorrect. Server will send an error message asking customer to input code again. The front-	

	end will display a MBProgressHUD showing “invalid verification code”. This use case goes back to step 6.
Precondition:	None
Postcondition:	User successfully registers to use our service
Assumptions:	None

2) “Sign in” use case

Use Case Name:	Sign in	
Actor(s):	Customer / Valet	
Description:	This use case describes the sign in process for a user	
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 when a customer clicks the [SIGN IN] button	
		Step2: The system shows a view with two text field requesting the user to input: Phone Number (account number) and Password. This view has three buttons: [SIGN IN], [Cancel] and [Forget Password?]

	Step3: The customer inputs Phone Number and Password then clicks the [SIGN 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 [Cancel] button, the application will go back to welcome view and this use case concludes. Step3: if the customer clicks [Forget Password?] button, the application will go to "Reset Password" view. This use cases concludes. 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 "account and password mismatch".. 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 / Valet	
Description:	This use case describes the reset password process for a user	
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 user to input Phone Number. The view has two buttons: [SEND CODE] button on the

		bottom and [Cancel] button on the left top.
	Step3: The customer inputs Phone Number then clicks the [[SEND CODE] button.	
		Step4: The application sends the phone number to server to get a verification code. If the phone number is valid, the customer will get an message with verification code
		Step5: the application gets the successful message and goes to another view. This view is multi-lined window requesting user to input verification code and new password. There are two buttons on this view: [SUBMIT] button on the bottom and [Cancel] button on the left top corner.
	Step 5: The customer inputs the verification code and a password then clicks the [SUBMIT] 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. 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. 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) "Park a vehicle" use case

Use Case Name:	Par a vehicle	
Actor(s):	Customer and Valet	
Description:	This use case describes the process of parking a vehicle	
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 shows a map with customer's current location and two buttons: [Drop off] and [Search].	
	Step1: Initiate this use case if a customer is in the service area and clicks [Drop off]	
		Step 2: A progress indicator will display on the view. The front-end sends customer's

		information to server to request to create an order
		Step 3: the server gets customer's information then uses its location to find the nearest valet. If there is a valet nearby. Server will create this order and send order information to both customer and valet.
		Step 4: the front-end of customer gets the order object and update the view. The [Drop off] and [Search] button disappear. A [cancel] button show on the right top of the view. The basic information of the valet shows on the top of the view.
		Step 5: the front-end of valet gets the order object and update the view. The system show a map with valet's own location, customer's location and drop off point. The basic information of the customer shows on the top

		of the view with a [Got the vehicle] button
	Step 6: customer meets the valet and transfers its vehicle to valet	
	Step 7: valet gets the vehicle and press the [Got the Vehicle] button	
		Step 8: the front-end of valet send a request to server to update the order status to "parking". The server gets the request and update the order. Then it sends the updated order object to both customer and valet.
		Step 9: the front-end of customer gets the response and update the view. The [cancel] button disappears and tells the customer that a valet is parking its vehicle
		Step 10: the front-end of valet gets the response and update the view. The [Got the vehicle] button changes to [Park finished] button

	Step 11: the valet park the vehicle and press the button [Park finished]	
		Step 11: the front-end of the valet send a request to server to update the order status to "Parked". The server gets the request and update the order. Then it sends the updated order object to both customer and valet
		Step 12: the front-end of the customer gets the response and update the view. It hides the valet information and shows two buttons: [Search] and [Return your vehicle]. This use case concludes here.
Alternative(s):	Step1: if the customer is not in the service area. A button [Go to service area] will show on the map. If the customer clicks this button, the map will move to the preset point in the service area. This use case stays in step 1 Step3: if there is no valet available nearby. A MBProgressHUD will show on the view telling customer "no valet available, try later". This use case goes back to step 1. Step4: if the customer clicks the [cancel] button. Then the order will be canceled and this use case concludes.	

Precondition:	None
Post condition:	The vehicle of customer is parked in the parking lot.
Assumptions:	User already has an account

5) "Return a vehicle" use case

Use Case Name:	Return a vehicle	
Actor(s):	Customer and Valet	
Description:	This use case describes the process of returning a vehicle	
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 shows a map with customer's current location and two buttons: [Return your vehicle] and [Search].	
	Step1: Initiate this use case if a customer is in the service area and clicks [Return your vehicle]	
		Step 2: A progress indicator will display on the view. The front-end sends an request to server to update to order status to "Requesting back"

		Step 3: the server gets the request and finds the nearest valet for this customer. If there is a valet available, server will update the order and sends the updated order to both customer and valet.
		Step 4: the front-end of customer gets the order object and update the view. The [Drop off] and [Search] button disappear. The basic information of the valet shows on the top of the view.
		Step 5: the front-end of valet gets the order object and update the view. The system show a map with valet's own location, customer's location and return point. The basic information of the customer shows on the top of the view with a [Got the vehicle] button
	Step 6: valet gets the vehicle and press the [Got the Vehicle] button	

		Step 7: the front-end of valet send a request to server to update the order status to “returning back”. The server gets the request and update the order. Then it sends the updated order object to both customer and valet.
		Step 8: the front-end of customer gets the response and update the view.
	Step 9: customer meets the valet and transfers its vehicle to valet	
	Step 10: the valet press the button [Return finished]	
		Step 11: the front-end of the valet send a request to server to update the order status to “Finished”. The server gets the request and update the order. Then it sends the updated order object to both customer and valet
		Step 12: the front-end of the customer gets the response and updates the view. It hides the valet

		information and shows a button: [pay for your bill] with amount
	Step 13: customer pay the bill using Apple Pay and authenticate the payment using its fingerprint.	
		Step 14: if the payment is successful. The application hides the payment button and shows two buttons: [Search] and [Drop off]. This use case concludes here.
Alternative(s):	Step1: if the customer is not in the service area. A button [Go to service area] will show on the map. If the customer clicks this button, the map will move to the preset point in the service area. This use case stays in step 1 Step3: if there is no valet available nearby. A MBProgressHUD will show on the view telling customer "no valet available, try later". This use case goes back to step 1. Step 14: if the payment is not successful. A MBProgressHUD will show on the view telling customer "try again". This use case goes back to step 12	
Precondition:	The vehicle of customer is parked in the parking lot.	
Post condition:	Customer gets its vehicle back.	
Assumptions:	User already has an account	

4.2.3.5. Class diagrams

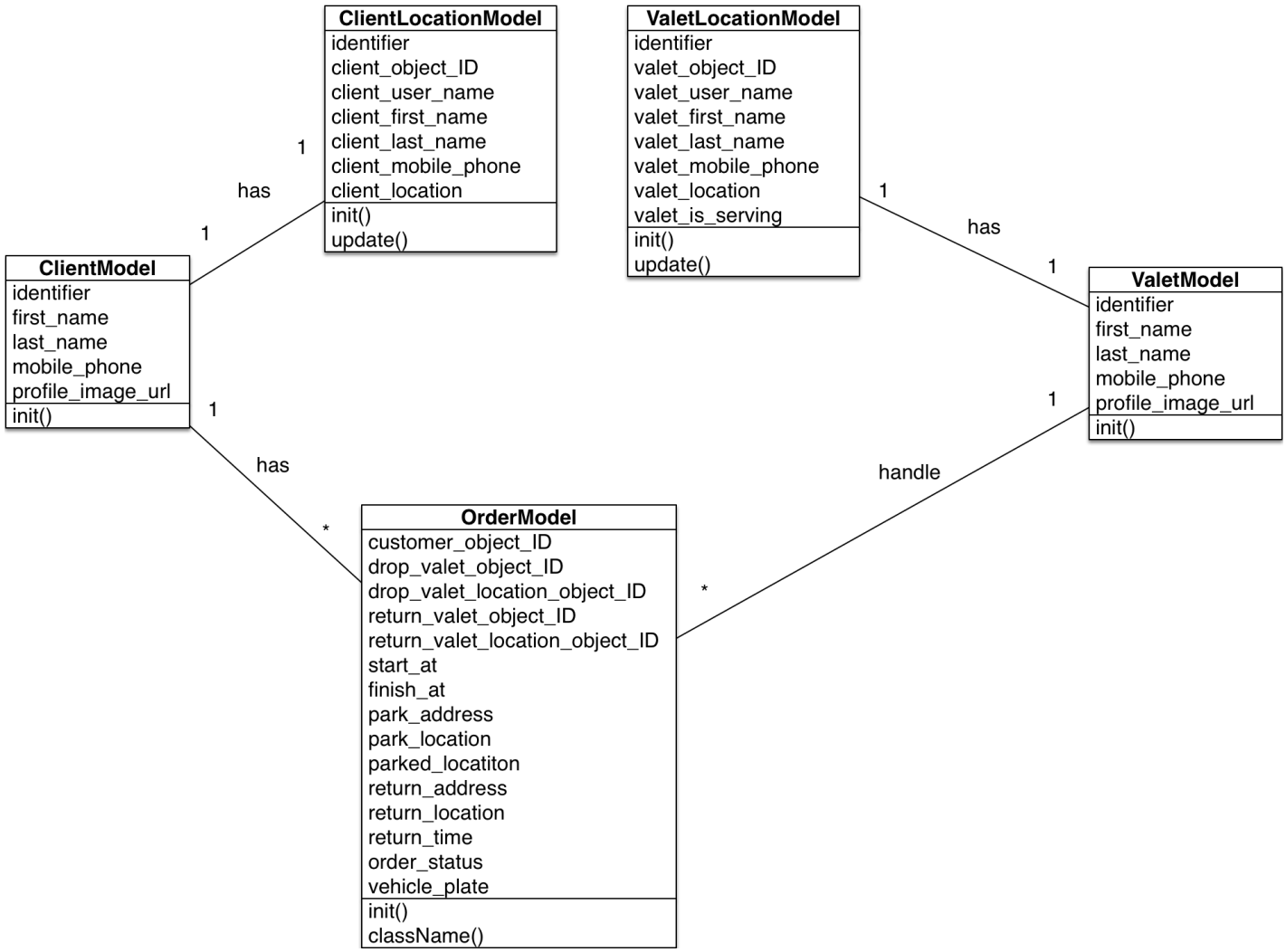


Figure 4-13 Class Diagram

As shown in figure 4-13, the class diagram shows the relationship between different class. The *ClientModel* indicates the customer. It stores identifier, first name, last name, mobile phone number and an image URL of a customer. The *ValetModel* indicates the valet. It stores identifier, first name, last name, mobile phone number and an image URL of a valet. The *LocationModel* indicates customers' and valets' locations. Each customer or valet has only one location model. The

OrderModel indicates an order created by customer. Each customer can have multiple orders. And each valet can handle multiple orders. Aside from the basic attribute in the *OrderModel*, the most important attribute is *order_status*. It indicates the status of the order. Both customer and valet can update *order_status*.

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. The user interface design for the two front-ends will be introduce separately.

4.2.4.1. UI design for customer side

1) Sign up pages

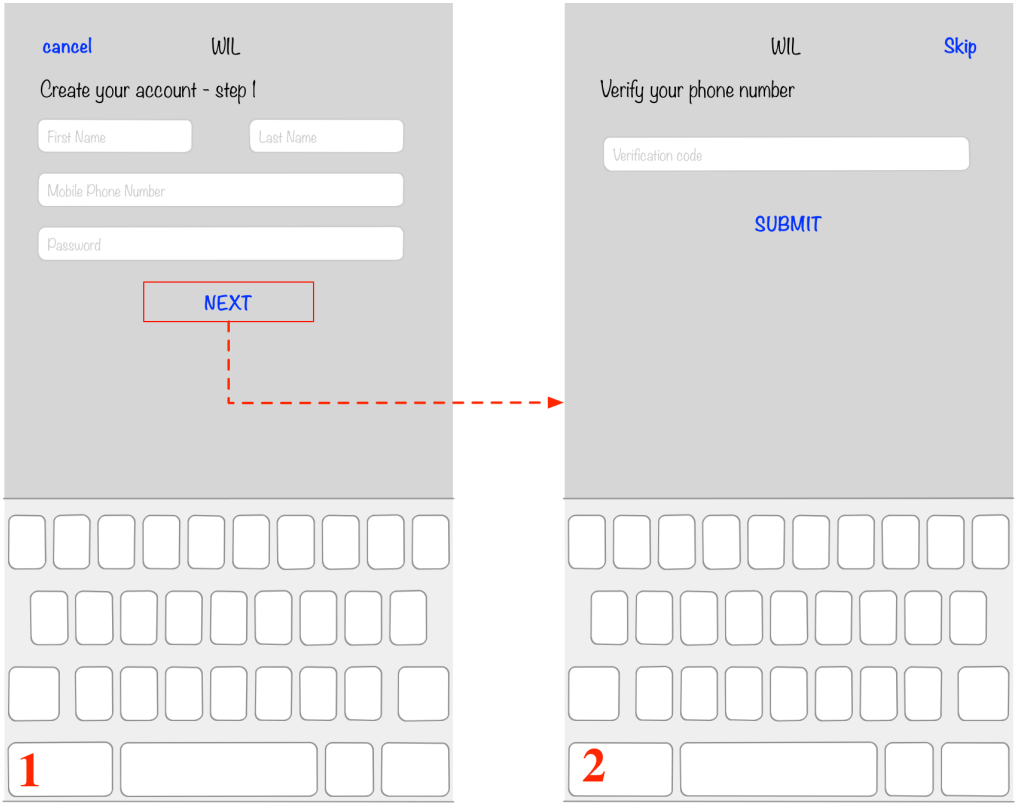


Figure 4-14 UI design of sign up pages

As shown in the figure 4-14, there are two pages for a customer to sign up our service

I. basic information page: on this page, customer needs to input basic information: first name, last name, mobile phone number and password. If the customer clicks the [NEXT] button, the application will go to next page.

II. verification code page: on this page, customer needs to input the verification code sent by text message. If the user clicks the [SUBMIT] button, the application will go to next page.

2) Sign in page & Reset password page

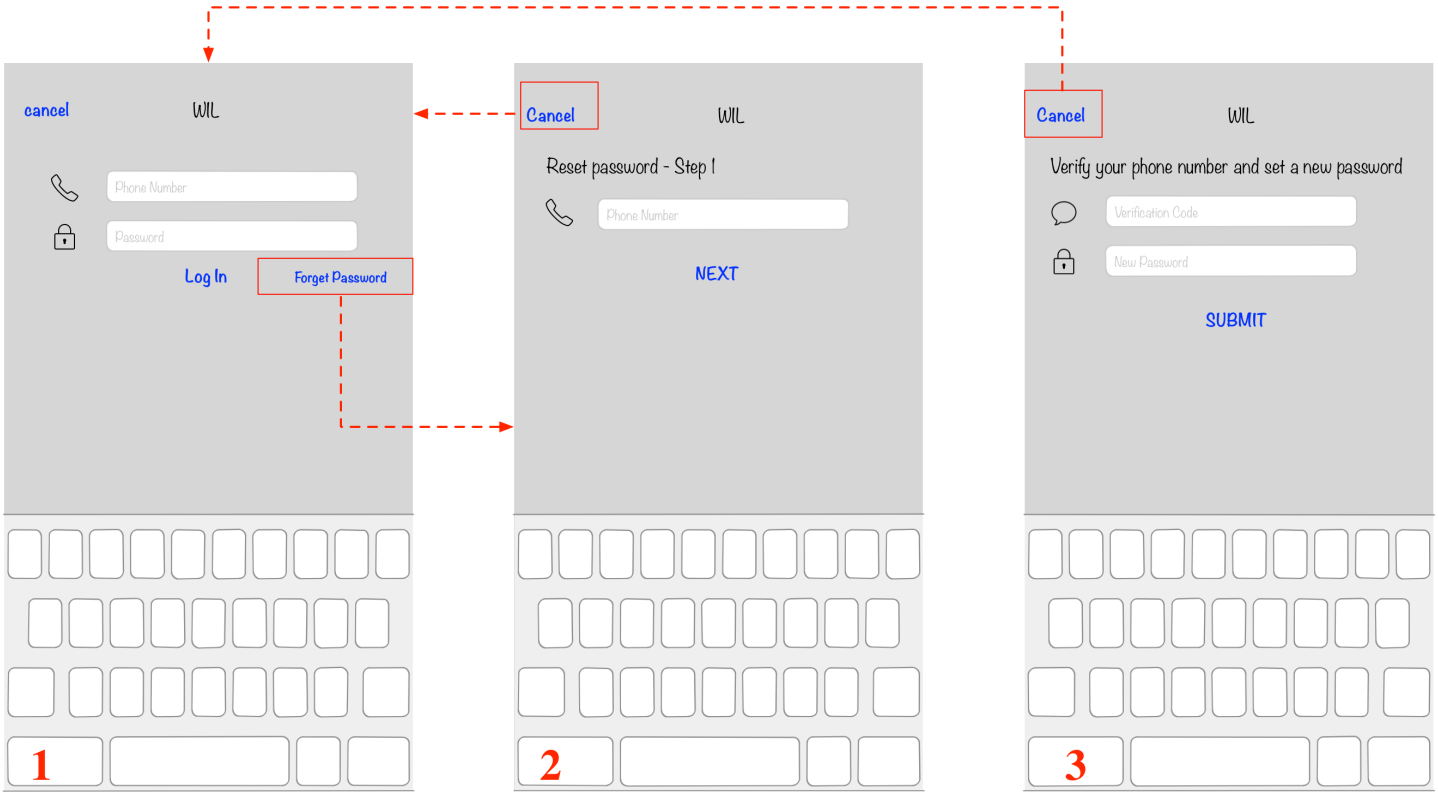


Figure 4-15 UI design of sign in page and reset password pages

As shown in figure 4-15, user can use the first page to sign in our service and use the other two pages to reset password

I. Sign in page: user needs to input mobile phone number and password to sign in. If the phone number and the password is valid, the front-end will go to main service view.

II. Reset password page 1: if a user wants to reset password, he or she need to input its mobile phone number at first. Then the user needs to clicks the [NEXT] button and our server will send a text message to the mobile phone number

III. Reset password page 2: user needs to input verification code and a new password then clicks the [SUBMIT] button. If the verification code is valid then our system will update user's password.

3) Main functions pages

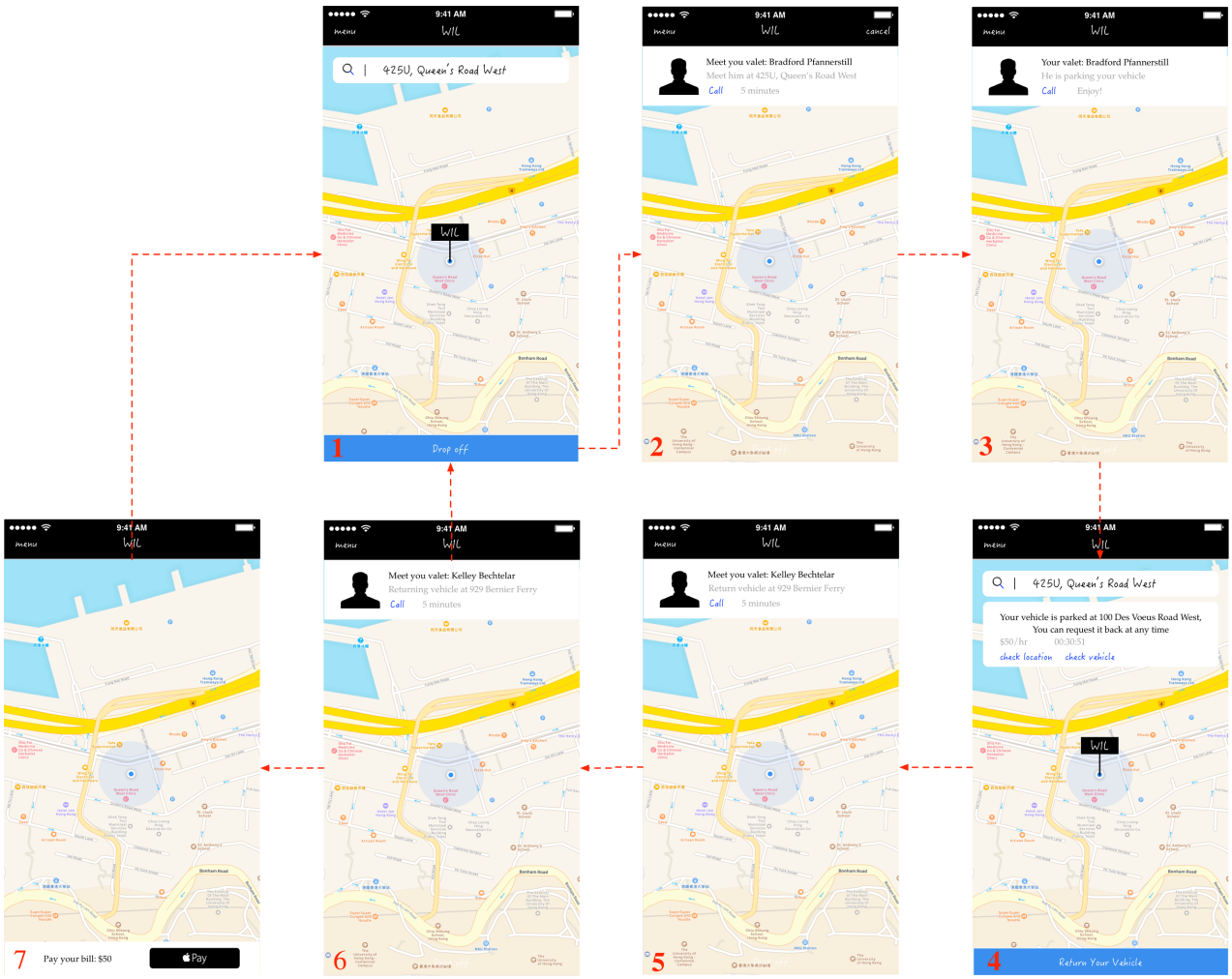


Figure 4-16 UI design of main function pages for customers

As shown in figure 4-16, there are 6 pages in the main function pages. These six pages contain the main process in this project including requesting parking a vehicle and requesting returning a vehicle.

I. Main page: this page shows when a customer opens the application. Our system will firstly check if the customer has an unfinished order. If so, it will jump to corresponding page. If not, it will show a map with two buttons: [search] button on the top and [Drop off] button on the bottom. On this page, a customer can view valets nearby and service area. If the customer wants to request valet service, he or she can just tap the [Drop off] button. If there are valets available, the application will jump to page 2.

II. Requesting service page: this page shows when a customer has an order with status “requesting service”. It means the customer has requested valet service but still on the way to drop off point. There is a [cancel] button on the right top corner for customer to cancel the service. And on the of the map shows the basic information of this order including valet’s name, drop off address and estimated time for valet to go to the meet location.

Customer can call valet by tapping the [call] button. On the map, there is a marker showing current location of the valet.

III. Parking page: this page shows when a customer has an order with status “parking”. It means that the customer has requested valet service and the valet has already got the vehicle. Customer cannot cancel the order now so the [cancel] button has disappeared. The label showing the drop off address changes to “valet is parking your vehicle”. On the map, there is a marker showing current location of customer’s vehicle.

IV. Parked page: this page shows when a customer has an order with status “parked”. It means that the customer has requested valet service and its vehicle is parked in our parking lot. There is a marker on the map showing current location of customer’s vehicle. And a window on top of the map shows the time duration and additional options. And there are two buttons on this page: [search] button for search a location and [return your vehicle] button on the bottom of the page for customer to request to get its vehicle back.

V. Requesting back page: this page shows when a customer has an order with status “requesting back”. It means that the customer has requested to get its vehicle back and our valet is on way to parking lot. There is a marker on the map showing the current location of its vehicle and another marker showing the real-time location of the valet. The window on top of the map offers basic information including valet’s name, return address and estimated time. Customer can also call our valet by tapping the [call] button.

VI. Returning back page: this page shows when a customer has an order with status “returning back”. It means that the customer has requested to get its vehicle back and our valet has already got the vehicle and is on its way to meet point. There is a marker on the map showing the real-time location of valet and vehicle.

VII. Payment page: this page shows when a customer has an order with status “payment pending”. It means that the customer needs to finish the payment to continue use our service. This is a button [Apple Pay] for customer to use.

4.2.4.2. UI design for valet side

The *sign in* page and *reset password* page in valet side is similar to that in customer side. So I will only introduce the *main function pages* of this front-end. The design for *main function pages* is shown in figure 4-17

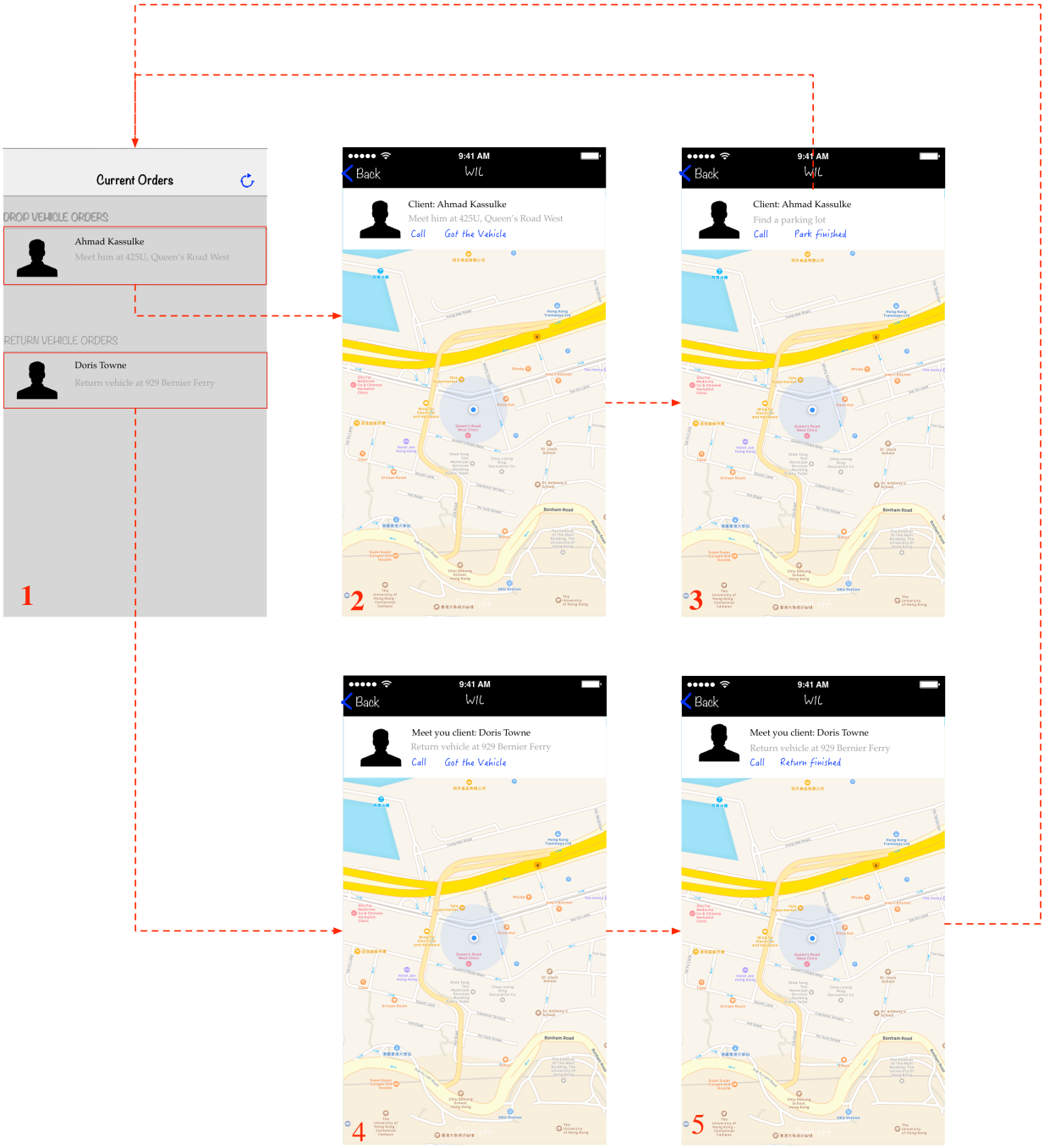


Figure 4-17 UI design of main function pages for valets

As shown in figure 4-17, there are 5 pages in *main function pages*.

I. Main page: this page shows a table view with two sections. The first section shows all the order requesting parking and the second section shows all the order requesting returning. And there is a refresh button on the right top corner for valet to refresh orders. If a valet clicks the cell in the first section, the application will push to page 2. If a valet clicks the cell in the second section, the application will push to page 4.

II. Park page1: this page shows when the valet is going to meet the customer. There is a map with customer's basic information on top of the map. Valet can get customer's name and drop off address. There is a marker on the map indicating customer's real-time location and drop off location. If the valet gets the vehicle for the customer, he or she can click the [got the vehicle] button to go to next page

III. Park page2: this page shows when the valet is parking customer's vehicle. There is a map with customer's basic information on top of the map. If the valet has already parked

the vehicle, he or she needs to clicks the [park finished] button to update the order status.

IV. Return page 1: this page shows when the valet is going to return customer's vehicle and on the way to parking lot. There is map showing customer's location, return location and vehicle's location. And customer's information is on top of the map. If the valet has already got the vehicle from the parking lot, he or she needs to click the [got the vehicle] button to update order status.

V. Return page 2: this page shows when the valet has already got the vehicle from the parking lot and is going to meet customer at return location. There is map showing customer's location, return location and a route to return location. If the valet has already returned the vehicle back to customer, he or she needs to click the [return finished] button to update status.

Chapter 5. IMPLEMENTATION

In this chapter, I will introduce how this project is implemented. At first, I will give some brief introduction for the development environment. And then I will introduce the implementation of front-ends and back-end respectively.

5.1. Development Environment

5.1.1. Hardware Configurations

The hardware I use to develop the back-end and front-ends of this project is a MacBook Pro with mac OS Sierra version 10.12.3 as shown in figure 5-1.

The hardware I use to test this project are iPhone 6s and iPhone 7 Plus with iOS 10.2.

Figure 5-1 Hardware configuration

5.1.2. Software Configurations

The software I use to implement the front-ends is Xcode version 8.2.1 as shown in figure 5-2. . Xcode is developed by Apple and is the official IDE in iOS application develop[15].

Figure 5-2 Xcode with version number

5.2. Implementation of front-ends

There are two mobile clients in this project. One for customers and another one for valets. They will be introduced separately.

5.2.1. Front-end for customers

The implementation of front-end for customers is based on Model-View-Controller (MVC). The relationship and modules are shown in figure 5-3.

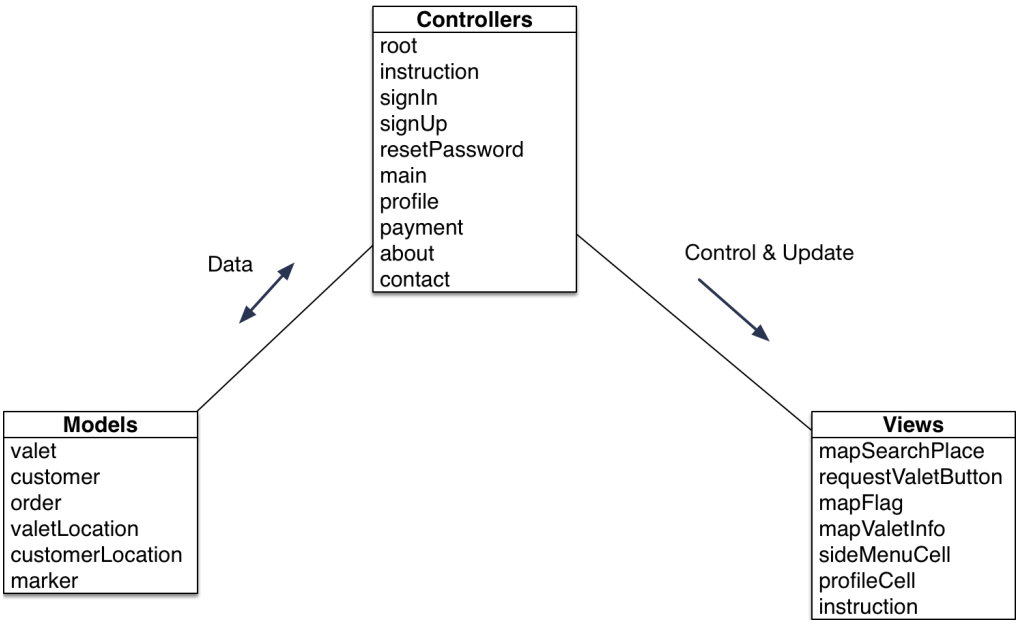


Figure 5-3 Model, View, Controllers

I will introduce the implementation of modules and views respectively.

5.2.1.1. Modules

1) LibraryAPI

LibraryAPI inherits from NSObject and is the most import module in this project. I use the singleton pattern to implement this class. It handles all the data and file request in this project. It has a class method naming *sharedInstance*. The implementation of this method is shown in figure 5-4

```
+ (LibraryAPI *)sharedInstance
{
    static LibraryAPI *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[LibraryAPI alloc] init];
    });

    return _sharedInstance;
}

- (id)init
{
    if (self = [super init])
    {
        self.polygonClient = [[PolygonClient alloc] init];
        self.clientLocationClient = [[ClientLocationClient alloc] init];
        self.valetLocationClient = [[ValetLocationClient alloc] init];
        self.orderClient = [[OrderClient alloc] init];
        self.fileClient = [[FileClient alloc] init];
    }

    return self;
}
```

Figure 5-4 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, five clients will be created. These five clients handle the network request, file request and basic settings in this project. The basic workflow using *LibraryAPI* is shown in figure 5-5.

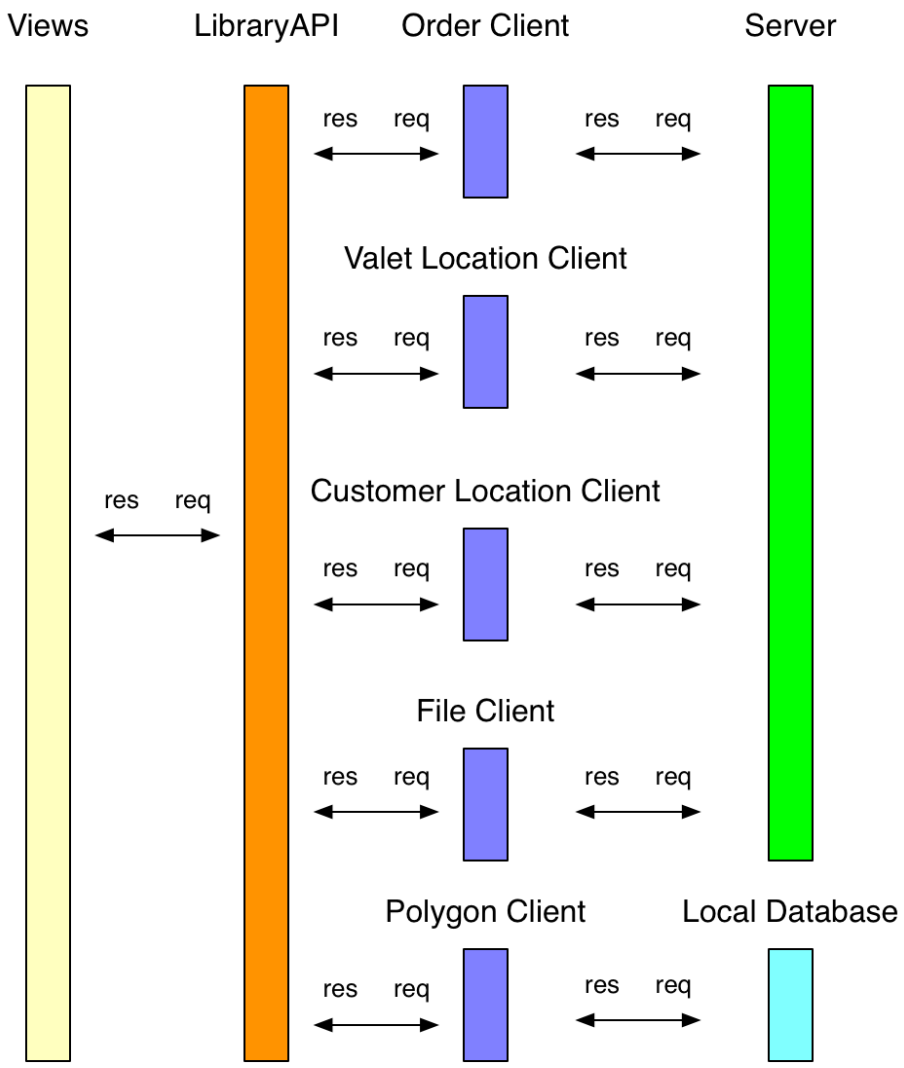


Figure 5-5 Data transfer using *LibrayAPI*

As shown in figure 5-5, all the request from *Views* is transferred to *LibraryAPI*. Then *LibraryAPI* will send these requests to corresponding client. Order client, valet location client, customer location client and file client will communicate with server and polygon client will communicate with local database.

Methods in *LibraryAPI* are shown in figure 5-6. The variables of these methods are hidden for simplicity.

```
// polygon
- (NSArray *)polygons;
- (GMSPolygon *)polygonForHKIsland;
- (GMSPolygon *)polygonForKowloon;
- (CLLocationCoordinate2D)serviceLocation;

// client's location
- (void)uploadClientLocation;
- (void)saveClientLocationObjectIDLocally;
- (void)getRouteWithMyLocation;

// valets' locations
- (NSMutableArray *)onlineValetLocations;
- (NSMutableArray *)availableValetLocations;
- (NSMutableArray *)busyValetLocations;
- (ValetLocation *)dropValetLocation;
- (ValetLocation *)returnValetLocation;

- (void)fetchValetsLocationsWithStatus;
- (ValetLocation *)nearestValetLocation;

// order
- (void)createAnOrderWithValetObjectID;
- (void)fetchOrderStatusWithObjectID;
- (void)cancelAnOrderWithOrderObject;
- (void)requestingVehicleBackWithOrderObject;
- (void)checkIfUserHasUnfinishedOrder;

// file
- (void)uploadFile;
- (void)getPhotoWithURL;
```

Figure 5-6 Methods in *LibraryAPI*

Most methods interacting with clients use block as callback. An example which handles order creation is shown in figure 5-7.

```
- (void)createAnOrderWithValetObjectID:(NSString *)valetObjectID
                  valetLocationObjectID:(NSString *)valetLocationObjectID
                            parkAddress:(NSString *)parkAddress
                           parkLocation:(AVGeoPoint *)parkLocation
                                success:(void (^)(OrderObject *orderObject))successBlock
                                   fail:(void (^)(NSError *error))failBlock {
    [self.orderClient createAnOrderWithValetObjectID:valetObjectID
                               valetLocationObjectID:valetLocationObjectID
                                         parkAddress:parkAddress
                                        parkLocation:parkLocation
                                             success:^(OrderObject *orderObject) {
                                                 successBlock(orderObject);
                                             }
                                                fail:^(NSError *error) {
                                                    failBlock(error);
                                                }];
}
```

Figure 5-7 Order creation method in *LibraryAPI*

This method takes valet's identifier, valet's location identifier, park address and park location as variables. It invokes *OrderClient* to get current customer's information then create an order object with customer's information, valet's information and park address. Then *OrderClient* will send this order object to server. If all the information is valid and saved successfully, the *successBlock* will be invoked to refresh the main view.

2) Order Client

OrderClient inherits from NSObject and handles all the request related to order. It can create an order, check order status and update order status. The main functions in *OrderClient* are as below

- *createAnOrder()*: this function creates a new order using customer's information, valet's information and park address. If the order is created successfully, it will return an order object to main view. This function as well as other methods in client use block to invoke callback. The implementation of block is shown in figure 5-8.

```
    [orderObject saveInBackgroundWithBlock:^(BOOL succeeded, NSError * _Nullable error) {
        if (succeeded) {
            // set valet status to busy
            ValetLocation *valetLocation = [ValetLocation objectWithObjectId:valetLocationObjectID];
            [valetLocation fetchInBackgroundWithBlock:^(AVObject * _Nullable object, NSError * _Nullable error) {
                if (object) {
                    if (valetLocation.valet_is_serving == nil || [valetLocation.valet_is_serving boolValue] == NO) {
                        valetLocation.valet_is_serving = @(YES);
                        [valetLocation saveInBackgroundWithBlock:^(BOOL succeeded, NSError * _Nullable error) {
                            if (succeeded) {
                                successBlock(orderObject);
                            } else {
                                [orderObject deleteInBackground];
                                failBlock(nil);
                            }
                        }];
                    } else {
                        [orderObject deleteInBackground];
                        failBlock(nil);
                    }
                } else {
                    [orderObject deleteInBackground];
                    failBlock(nil);
                }
            }];
        } else {
            [orderObject deleteInBackground];
            failBlock(nil);
        }
    }];
}
```

Figure 5-8 Implementation of saving order object using block

- *checkIfUserHasUnfinishedOrder()*: this function is invoked when a customer enters the main view. It checks if the customer has an unfinished order. If so, it will return an order object to main view.
- *fetchOrderStatusWithObjectID()*: this function uses the identifier of order object to fetch the order object from server.

- *updateOrderWithObjectID()*: this function updates order status using the identifier of the object and status that needs to update.

3) Valet Location Client

ValetLocationClient inherits from *NSObject*. It handles all the request related to valets' locations. The main view uses this client to get valets' locations to tell user that there are valets nearby. The main functions in *ValetLocationClient* are as below.

- *fetchValetsLocationsWithStatus():* this method can fetch valets' locations based on the order status. If the order status is *None* or *Parked*, it will return all the available valets nearby. If the customer is requesting our service, it will only return the valet serving the customer.
- *nearestValetLocation()*: this function can get nearest valet for customer. When a customer wants to request valet service or wants to get its vehicle return, this method will be invoked and return a valet location object.

4) Customer Location Client

CustomerLocationClient inherits from *NSObject*. It handles all the request related to customers' locations. When the application is launched for the first time, the front-end will ask customer for location service permission. If the customer allows the application to access location, the front-end will start upload customer's location to server. The main function in this client is as below:

- *uploadClientLocation()*: this function takes user's location, which is a *AVGeoPoint* object, as variable. It will firstly check if the user has a location object on the server. It not, it will create a new location object an upload it to server. Otherwise it will update that location object.

5) File Client

FileClient inherits from *NSObject*. It handles all the files transfer including uploading to server and downloading from server. In this project, this client is mainly used to upload and download photos since both customers and valets have profile image. The main functions in this client are as below

- *uploadFile()*: this function takes an *AVFile* as input. It calls *saveInBackgroundWithBlock* to upload this file to server. If the

uploading process is successful, the *file* object will have a new URL attribute indicating the address of the photo on server.

- *getPhotoWithURL()*: this function takes an URL as input. It creates a file with the URL and call *getDataInBackgroundWithBlock()* to download the image. If the image is downloaded successfully, this method will call *successBlock* and return the image to main view.

6) Polygon Client

PolygonClient inherits from *NSObject*. It saves the polygon information locally. The polygon indicates the service area. Customer must drop off or get vehicle return in the service area. The main function in this client is as below

- *polygons()*: this method returns a *NSArray* containing the polygon information. Each object in this client is an instance of *GMSPolygon*.

5.2.1.2. User Interface Implementation

In this part, I will introduce the implementation of user interface in front-end for customers.

1) Sign up pages

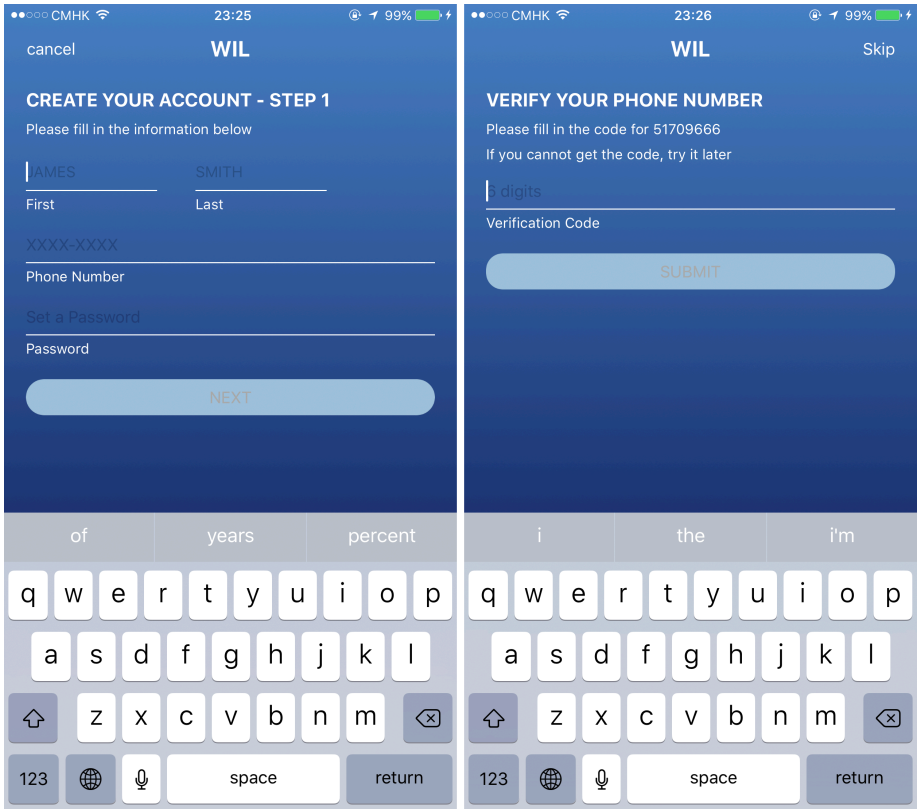


Figure 5-9 Sign up pages

As shown in figure 5-9, the sign up pages consist of two pages. Customer can use these two pages to create an account.

On the first page, a customer needs to fulfill the information including first name, last name, phone number and a password. Then he or she needs to click the [NEXT] button to go to next stage.

On the second page, the customer need to input the verification code sent by text message. Then he or she needs to click the [SUBMIT] button to verify the code. The customer can also choose to skip this stage but its mobile phone number is not verified.

2) Sign in page and reset password pages

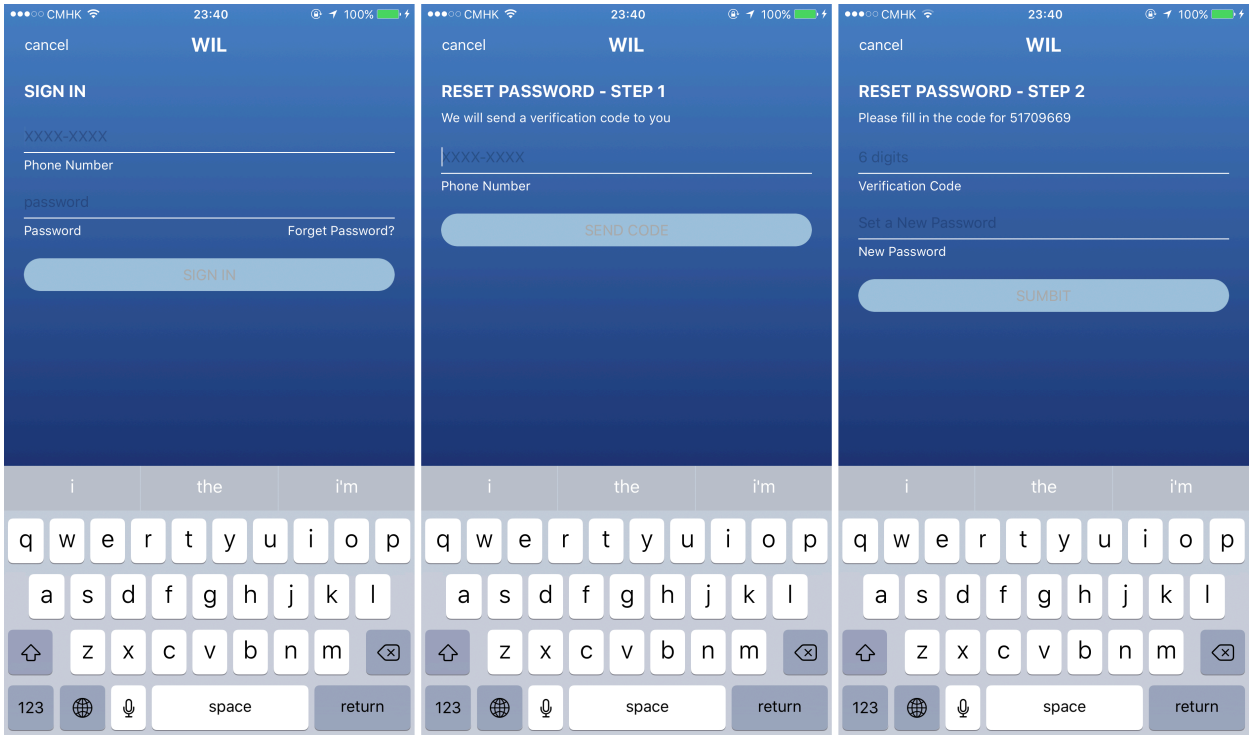


Figure 5-10 Sign in and reset password pages

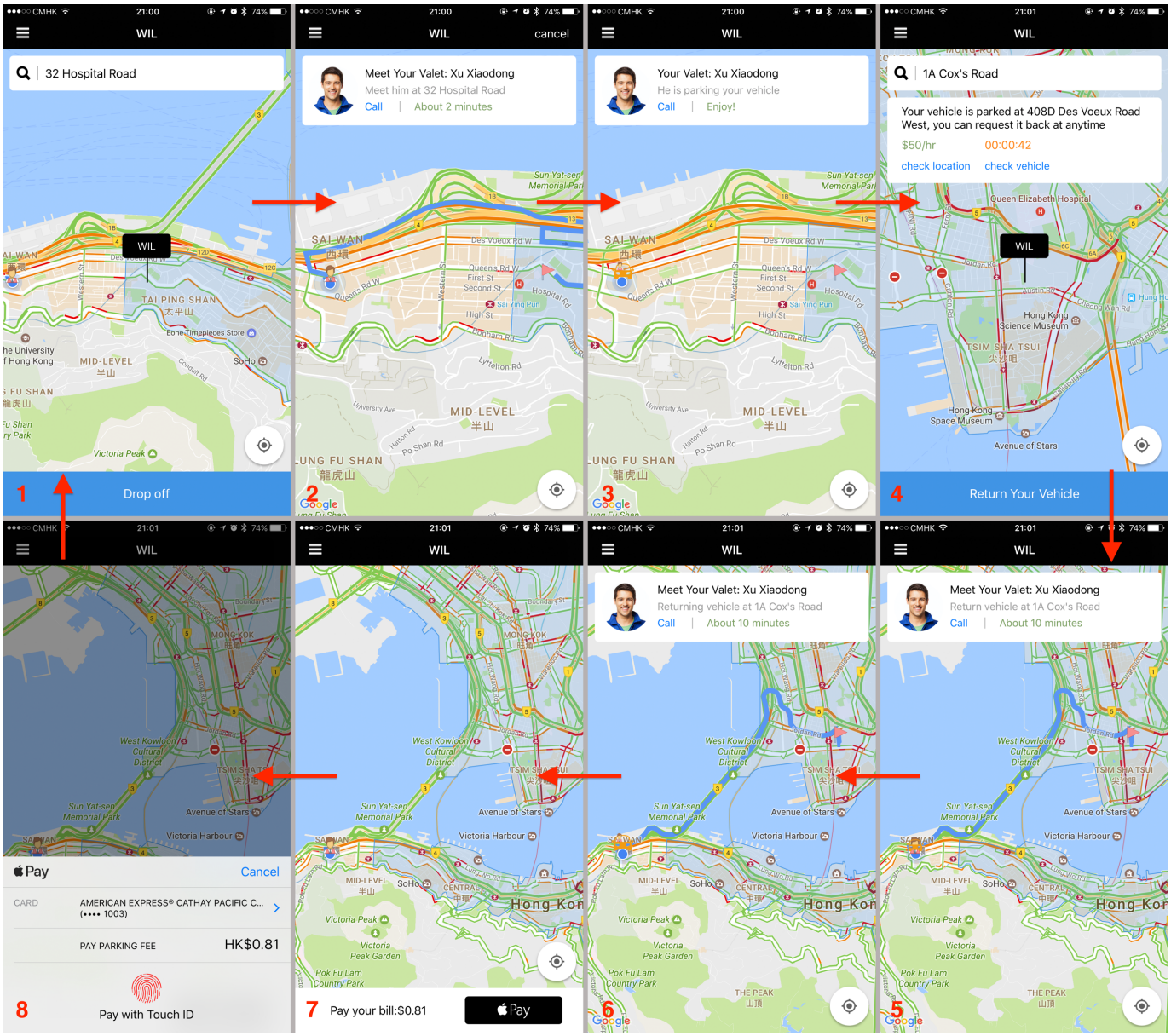
As shown in figure 5-10, a customer can use the first page to sign in and use the other two pages to reset password.

On the first page, a customer needs to input its phone number and password to sign in. If the account and password are valid, this view will dismiss and main view will show. If the account is not found or the password is wrong, a hud will show on this page telling customer to input phone number or password again.

On the second page, a customer can start to reset password by inputting phone number. He or she needs to click [NEXT] button after inputting its number. Then our system will send a verification code by text message to the customer. If the phone number is valid, the front-end will go to the third page. If the phone number is invalid, a hud will show on this page telling customer to input phone number again.

On the third page, a customer needs to input the verification code to finish resetting password. If the verification code is correct, this page will dismiss and main view will show. If the verification code is incorrect, a hud will show on the page telling customer to input the code again.

3) Main function pages



5-11 Main Function Pages

As shown in figure 5-11, there are 6 pages to show in a park and return process.

On the first page, a customer can view its location and request our service. There is a map with some components as below

- Map: an instance of *GMSMapView*. The *myLocationButton* and *trafficEnabaled* are set to *YES*.
- Valet Markers: an instance of *GMSMarker*. They show positions of valets
- Search View: an instance of *MapSearchPlaceView* and inherits from *UIView.* By tapping this button, a customer can search its destination
- Drop off button: an instance of *RequestValetButton* and inherits from *UIButton.*
- Flag: an instance of *MapFlag* and inherits from *UIView.* The flag indicates the customer's location and if he or she is in the service area
- Service Area: an instance of *GMSPolygon*

On the second page, a customer has an order with status "dropping off". There are some new components on this page

- Route: an instance of *GMSPath*. It shows the direction from customer's location to drop off point.
- Cancel button: an instance of *UIButton*. The customer can cancel the order by tapping this button
- Valet information view: an instance of *MapValetInfoView*, which inherits from *UIView*. This view shows basic information of valet and order status.

On the third page, a customer has an order with status "parking". This view is similar to the second view. The information on the *MapValetInfoView* changes to "parking". And the image of the valet marker changes to a valet on a vehicle.

On the fourth page, a customer has an order with status "parked". This view is similar to the first view. The text on the request service button changes to "Return your vehicle". And there is a marker on the map shows current location of customer's vehicle.

On the fifth page, a customer has an order with status "requesting back". This view is similar with the second view. The text on the

MapValetInfoView changes to “return vehicle at an address”. And there is a valet marker and a vehicle marker on the map.

On the sixth page, a customer has an order with status “returning back”. It means the valet has already got the vehicle from parking lot. This view is similar to the third view. The text on the *MapValetInfoView* changes to “returning vehicle at an address”.

On the seventh page, a customer has an order with status “payment pending”. It means the customer has already got its vehicle back and need to pay for the bill. The amount of the bill is shown on left bottom corner and apple pay button is shown on right bottom corner.

On the eighth page, the apple pay view pops up and the customer can use its fingerprint to authenticate the payment. He or she can also change the credit card if he or she wants.

4) Menu page, profile page and orders page

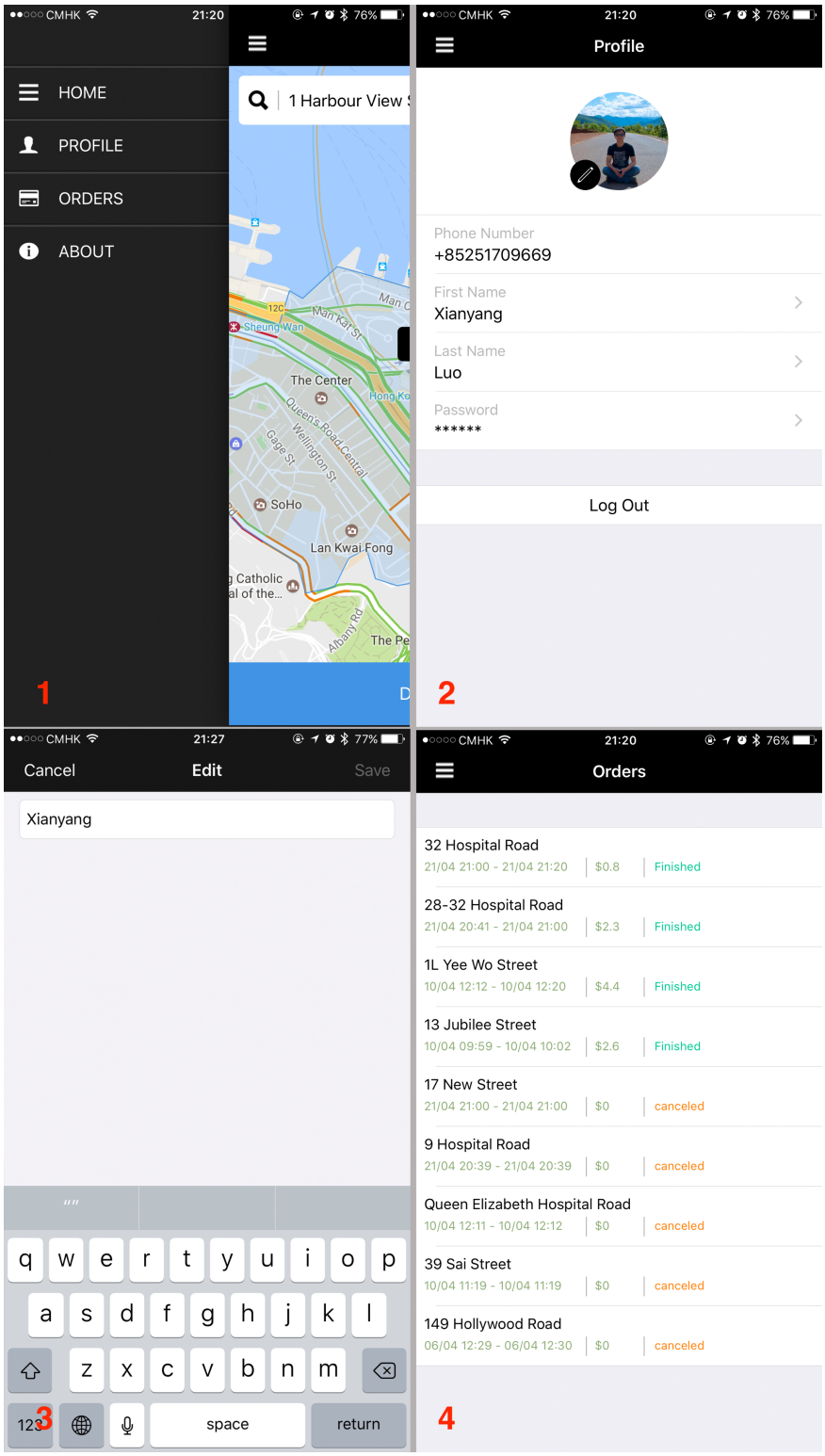


Figure 5-12 Menu page, profile page and orders page

As shown in figure 5-12, the first page is a menu view. A customer can open the menu hidden in the left side by tapping the menu button on the left top corner. The customer can enter different views from the menu.

The second page is profile page. It shows the basic information of the customer including profile image, mobile phone number and name. The customer can also change this information by tapping the cell and the third page will show.

The third page is used for editing information. A customer can edit its name or change its password here. By tapping the [Done] button on the right top corner, the front-end will upload the new data to server.

The fourth page shows all the historical orders of the customer. Each order shows with basic information including drop off address, start and end time, fee and last status. The list is implemented using *UITableView*. I create a class naming *OrderCell* to configure each order.

5.2.2. Front-end for valets

The architecture and modules of front-end for valets is similar to that for customers. So I will only introduce the difference of views.

The main view of front-end for valets is a *UITableView* showing all the orders related to the valet. I create a class naming *OrderCell* to configure each order. The implementation of main view is shown in figure 5-13.

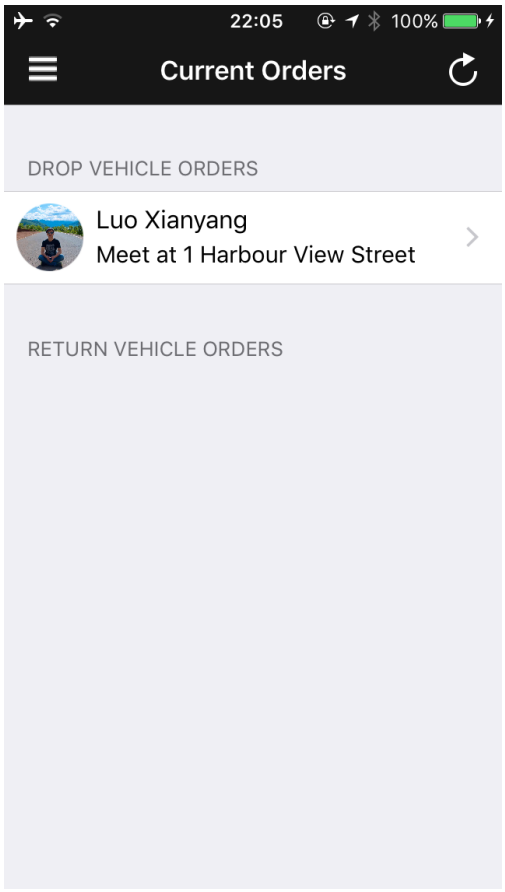


Figure 5-13 Main view of valet's front-end

The table view on this page has two sections. Orders in the first section have status "dropping off" or "parking" and orders in the second

section have status “requesting back” or “returning back”. Valets can click the refresh button on the right top corner to refresh this page.

A valet can click a cell to view the detail of that order as shown in figure 5-14.

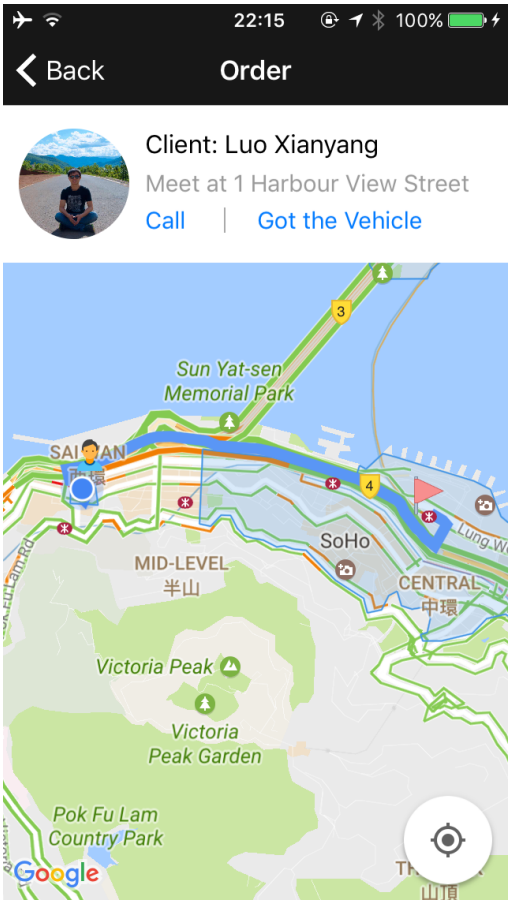


Figure 5-14 Detail of an order

A valet can get customer’s name, customer’s phone number to contact, customer’s location in blue marker, drop off point or return point and a route to meet point. The map is implemented using *GoogleMapService*. When the valet taps the [Park finished] button, our system will ask for a picture as shown in figure 5-15.

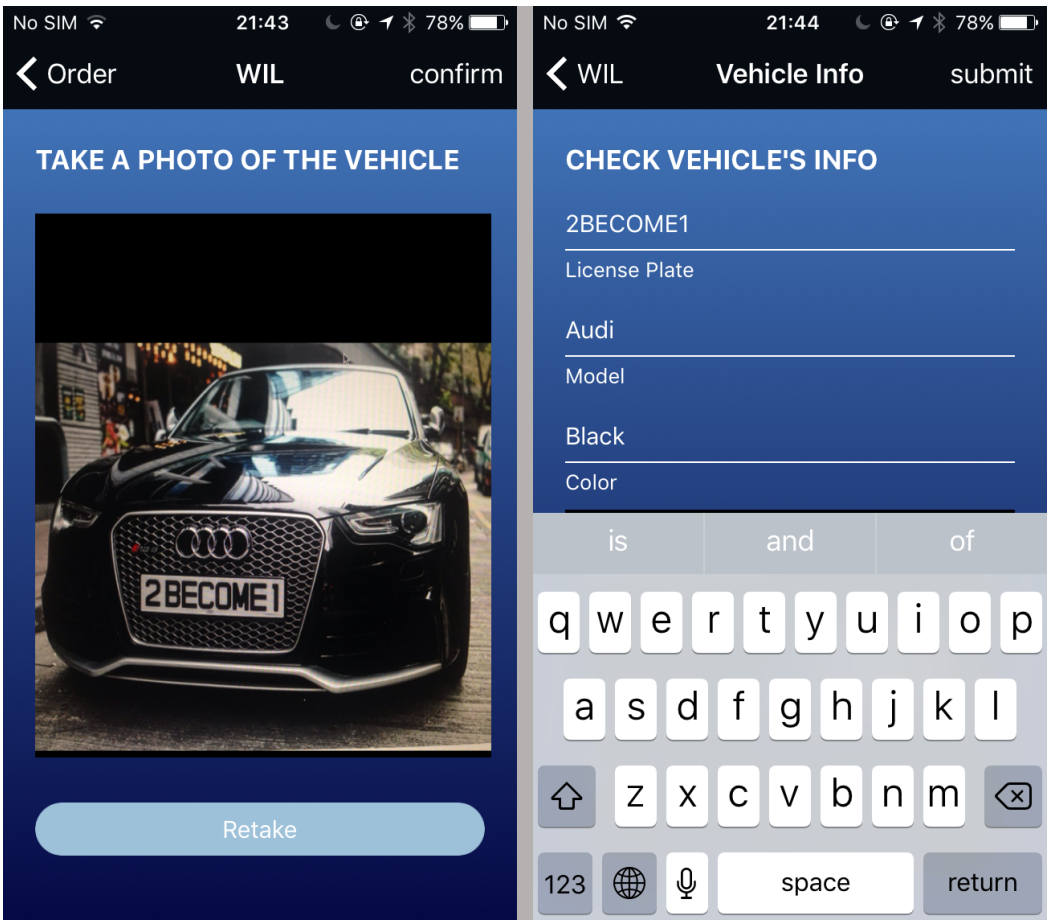


Figure 5-15 Upload an image of the vehicle

As shown in figure 5-15, the valet can take a picture of the vehicle on the first page. Camera will pop up when the valet enters this page. The code to invoke camera is shown figure 5-16.

```
- (void)takePhotoBtnClicked {
    UIImagePickerControllerSourceType sourceType = UIImagePickerControllerSourceTypeCamera;
    if ([UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
        UIImagePickerController *picker = [[UIImagePickerController alloc] init];
        picker.allowsEditing = YES;
        picker.sourceType = sourceType;

        [self presentViewController:picker animated:YES completion:nil];
    } else {
        NSLog(@"Can not find camera");
    }
}
```

Figure 5-16 Method to invoke camera

When the valet takes a picture and taps the [submit] button, the application will send this picture to OpenALPR to detect plate, model and color of the vehicle. The method for detecting is shown in figure 5-17

```
[apiInstance recognizeBytesWithImageBytes:imageBytes
                                secretKey:secretKey
                                  country:country
                          recognizeVehicle:recognizeVehicle
                                    state:state
                              returnImage:returnImage
                                     topn:topn
                                  prewarp:prewarp
                        completionHandler: ^(SWGInlineResponse200* output, NSError* error) {}
 ];
```

Figure 5-17 Method for detecting vehicle

The *apiInstance* in figure 5-17 is an instance of *SWGDefaultApi*. By calling the *recognizeBytesWithImageBytes()* method, it sends the image to server of OpenALPR. If the detection is successful, our system will save the image to LeanCloud and the application goes to page 2.

On the second page, the valet need to check if the detection is correct. If so, he or she can tap the [confirm] button to finish the order. If not, he or she can edit the information manually then tap the [confirm] button.

5.3. Implementation of back-end

The server of this application is based on LeanCloud which offers server and database service. I need to create two applications on the website and obtain application ID and key. The application ID and key need to be added to front-end as below

```
[AVOSCloud setApplicationId:@"vayB5LU7DOkUMkam7pUvMVDl-gzGzoHsz" clientKey:@"hn5bbWyOSTK9RJOwkTPLWYX2"];
```

I need to create a database on the server. By using LeanCloud to create database, I must set the class name and attribute on the website. Classes in the database of front-end for customers are shown in figure 5-18

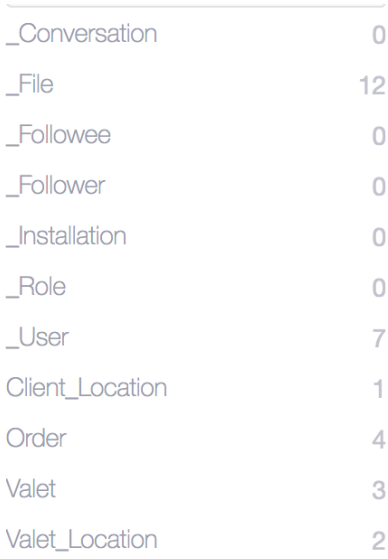


Figure 5-18 Classes in database of front-end for customers

To create, read, update and delete an object, the front-end needs to integrate SDK offers by LeanCloud.

- Create

```
AVObject *order = [AVObject objectWithClassName:@"Order"];
[order setObject:@"8 harbour street" forKey:@"park_address"];
[order saveInBackground];
```

The code about create an order object and set the park_address to 8 harbour street. Then call *saveInBackground* to save it in database.

- Read

```
AVObject *order =[AVObject objectWithClassName:@"Order"
objectId:@"558e20cbe4b060308e3eb36c"];
[todo fetchInBackgroundWithBlock:^(AVObject *avObject, NSError
*error) {
    NSString *title = avObject[@"park_address"];
}];
```

The code above use object ID and class name to create an object. Then it calls *fetchInBackgroundWithBlock* to get the data from database. *avObject* in the block is the saved object in the database.

- Update

```
AVObject *order =[AVObject objectWithClassName:@"Order"
objectId:@"558e20cbe4b060308e3eb36c"];
```

```
[order setObject:@"8 queen's road west" forKey:@"park_address"];
[order saveInBackground];
```

The code about use object ID and class name to create an object. Since this object has an object ID, the server has already saved this object in the database. It call *setObject()* to update an attribute then call *saveInBackgroud()* to save it to database.

- Delete

```
AVObject *order =[AVObject objectWithClassName:@"Order"
objectId:@"558e20cbe4b060308e3eb36c"];
[order deleteInBackground];
```

The code above use object ID and class name to create an object then call *deleteInBackground()* to delete this object in database.

Chapter 6. REVIEW AND FUTURE WORKS

6.1. Conclusion

After well-design and implementation, this application can provide convenient valet parking service for drivers in Hong Kong. It is so easy to register the service because all the information that needed is name and phone number. A customer can view service area and valets around in the main view. And it's so convenient to create an order by choosing a location and tapping a button. The customer can always view the location of its vehicle, time duration and estimated charge. If the customer wants its vehicle back, all he or she needs to do is just tapping a button. Our valets offer door to door service. I am sure drivers will love our service.

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 them below.

6.2. Future Works

6.2.1. 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 request to park to return by tapping a button on the watch and check the real-time location of valet. They do not need to look at the phone while driving, which makes it safer.

6.2.2. Navigation

In the front-end for valets and customers, there is a route shows the direction from user's location to meet point. But the route is static and cannot update and notify user. I think the front-end can integrate a navigation system to guide user to meet point.

6.2.3. Additional Service

Addition services can be added to your system. If a customer uses our application, we can offer additional service like washing the vehicle and refueling the vehicle. And customer can also pay for those services using mobile payment.

Reference

[1] Schwesinger, Ulrich, et al. "Automated valet parking and charging for e-mobility." Intelligent Vehicles Symposium (IV), 2016 IEEE. IEEE, 2016.

[2] Shneiderman, Ben, et al. "Designing the User Interface: Strategies for Effective Human-Computer Interaction." (2009).

[3] Gong J, Tarasewich P. Guidelines for handheld mobile device interface design[C]//Proceedings of DSI 2004 Annual Meeting. 2004: 3751-3756.

[4] Ayob N, Hussin A R C, Dahlan H M. Three layers design guideline for mobile application[C]//Information Management and Engineering, 2009. ICIME'09. International Conference on. IEEE, 2009: 427-431.

[5] Donahue, Megan, Chad Michael Roberts, and Rhoniel Villano Manlapaz. "Display screen with graphical user interface." U.S. Patent No. D722,608. 17 Feb. 2015

[6] Nystrom, Ryan, et al. "WatchKit by Tutorials: Updated for Swift 1.2 Making Apple Watch Apps with Swift." (2015)

[7] Gottipati H: Hacking Maps with the Google Maps API. (O'Reilly XML.com – 10 Aug 2005)

[8] Dalton, Craig M. "For fun and profit: the limits and possibilities of Google-Maps-based geoweb applications." Environment and Planning A 47.5 (2015): 1029-1046.

[9] Mortillaro, Diego, Simone Offredo, and Francesco Scrufari. "Method and apparatus for animating digital pictures." U.S. Patent Application No. 14/995,931.

[10] Rossmann, Alain. "Sliding side menu gui with menu items displaying indicia of updated content." U.S. Patent Application No. 13/856,765.

[11] Hetal N. Patel, Khushboo Desai, Tejendra Panchal, "An algorithm for automatic license plate detection from video using corner features", Information Processing (ICIP) 2015 International Conference on, pp. 301-306, 2015.

[12] Aura Conci, John Carvalho, Thomas Rauber, "A complete system for vehicle plate localization segmentation and recognition in real life scene.", Latin America Transactions IEEE (Revista IEEE America Latina), vol. 7, 2009, ISSN 1548-0992.

[13] Rodriguez J F, Diaz E J, Cleaver D A. Interactive valet parking management system: U.S. Patent Application 13/415,750[P]. 2012-3-8.

[14] Buhus E R, Timis D, Apatean A. AUTOMATIC PARKING ACCESS USING OPENALPR ON RASPBERRY PI3[J]. Acta Technica Napocensis, 2016, 57(3): 10.

[15] Allan, Alasdair. Learning iOS Programming: From Xcode to App Store. " O'Reilly Media, Inc.", 2013.