Smart Parking Solution

By Jason Jiang, Mercury Ding, Bo Peng, Chi Zhang

## 

## 

## 

## 

## 

## Abstract

Searching of a vacant parking spot can be painful, especially during busy hours and on weekends. Our team is looking for natural user experience that allows people to find parking spots with real time data and navigation system. In this paper, we are going to propose a solution with the concept of Internet of Things (IoT), which means that our application is connected to other services, for example, Google Calendar and Exchange Calendar. We also create a prototype with fake data to show that the architecture we designed can be adapted and implemented easily with a few modifications.

## Introduction

Anyone who has a car in Rose-Hulman should have spent a lot of time finding a parking lot. It is even more annoying and disappointing when you are in hurry. You could be late for your class or behind your schedule because of wasting fifteen minutes to find somewhere to park your car. We designed a parking management system combined with an application to help you find the unoccupied parking spot.

Imagine a situation like this. You need to go to Sports and Recreational Center to attend your commencement in 20 minutes. It usually takes about 15 minutes’ drive. On Graduation Day, you could imagine there must be a lot of students’ parents. It’s impossible to find a parking lot near the SRC. Even worse, find a parking lot on campus could be a challenge. However, if there is an application that could automatically find you a parking just by monitoring real-time parking situation and you only have to open it. You don’t need to enter any words because it connects your calendar to sync all the events on your schedule, and it knows where you should be. It finds the closest available parking spot from SRC for you and uses google map to navigate you directly to the available parking spot. There’s no need to drive around campus to look for one. There’s no need to type in the place you want to go to. Just a single click and you are all set.

That is the ideal user story that we try to accomplish. In the following paper, we will explain more details about our design and equipment setup, etc.

## Application Setup

Sensor

Ideally, we intent to use sensors to detect entry and exit of vehicles and the vacancy of parking spots. Also, we plan to use single board computer to collect data and send to server. However, due to time and funding constraints, we didn’t get to purchase these equipments. Instead, we simulated status data of parking lots and used real GPS coordinates of Rose-Hulman Parking lots in our application.

Application

The front-end part is a iOS based application totally finished using Swift. Client could log in by using its google account or microsoft exchange account in design. After the client is signed in, the application should automatically grab his or her calendar using the corresponding API. Then, the application should try to connect to parking server and send parking request with user’s identification and destination to request a parking lot. After the server responses the closest available parking lot, the user could open the google map by clicking the “map” button to see the location of the parking lot and do a direct navigation.

Server

Server is setup via Amazon ec2 service, in which we created an windows server 2012 r2 virtual machine instance. We created our server response application based on Java. Client could talk to the server based on via socket connection. Server will respond to client request by giving plain text string by its outputstream. Responses are formed based on client request type and database information. We use Mongodb as our database to store different types of information. Right now, we have four documents including parking lots, user types, rules and parking types. Client will receive the string by the server after the stream connection is closed. The server is a multithreading java socket application which supports multiple connections at the same time.

API

API has a series of unambiguous formats that can clearly send the client's operation information to server. There are two main parts: send to server and server replies. There are detailed description about each format.

Send to server:

1. Request a parking lot

Format: “Request\_parkinglot| id: XXX@XXX.com| curposition: (x1, y1)| destination: (x2, y2)|”

Id: the person’s id who makes the request

Curposition: current position of the person

Destination: destination

\*\*\*The server can figure out the sender type (such as students).The database needs the sender type, curposition and destination to determine the qualified parking lot.

2. Cancel a parkinglot (cannot be active when there is no request)

Format: “Cancel\_parkinglot| requestID:XXX|”

requestID: the request id that needs to be canceled.

\*\*\*The server needs to requestID from the previous request in order to cancel it. This ID is generated by the server when the request of parking lot is generated.

3. Change information

a) Change the user type

Format: “Change\_userType| id: XXX@XXX.com| newType: XXX|”

newType: new type, such as visitors, students

\*\*\*When the user needs to change his or her “type”, the server needs to have the new type that he or she wants to change to and his or her user id. The server will determine whether the requirement can be accepted.

b) Change the destination

Format: “Change\_destination| requestID:XXX| destination: XXX|”

requestID: the request id that needs to be change.

destination: the new destination

\*\*\*When the user wants to change the destination, the server needs to have the request id and new destination, so that the server can find out the request from the data and change the information.

c) Other change

Format: “Change\_username| id: [XXX@XXX.com](mailto:XXX@XXX.com)|username:XXX|”

New\_username: the new username that the user wants to change to.

\*\*\*The username is not the userID, so it can be changed when the user wants to change it (nickname).

Format: “Change\_password| id: [XXX@XXX.com](mailto:XXX@XXX.com)| old\_password:XXX| username:XXX|”

\*\*\*This one might need to be done in a more secure environment and other verification.

Server reply:

1. Request a parkinglot

Accept:

Format: “parkingpos: (x, y) |”

parkingpos: the parking position

\*\*\*The server will generate the request id.

Denied:

Format: “Denied| Reason: XXXX|”

Reason: the reason that the request is denied

\*\*\*The server will send a reason of denying the request back to the user.

2. Cancel a parkinglot

Accept:

Format: “ parkingpos: (x, y) |”

new\_des: the new destination

\*\*\*Cancel a parking lot cannot be denied since the user cannot select which reservation needs to be canceled.

3. Change information

Accept:

Format: “ Accept|”

Denied:

Format: Denied| Reason: XXXX|

## External APIs:

Google Map API

We use google map API by using the GPS position received from server to locate the potential parking lot and do further navigation. In order to use Google Map API, a valid special web address is needed to use.

Google Calendar API

Application certificate and validated tokens are required in order to use Google Calendar API in the application. It is required to grab calendar information from Google.

Socket IO

Socket IO is an api which is pretty convenient to use to create a point to point socket connection. No need to consider the streaming and error handle. It will do most of things for you.

Outlook API

Outlook API is pretty hard to use in iOS side development. Application authentication, request authentication, and permission settings are required. Pretty complex and not recommend to use in further projects.

## Reference

Google Map API for iOS: <https://developers.google.com/maps/documentation/urls/ios-urlscheme>

Google Calendar API for iOS:

<https://developers.google.com/google-apps/calendar/quickstart/ios>

Microsoft Exchange REST API for iOS:

<https://docs.microsoft.com/en-us/outlook/rest/ios-tutorial>

Socket io For iOS:

<https://github.com/socketio/socket.io-client-swift>