

# ***Park-N-Go: Proof of Concept of an Automated Parking Solution***

Computer Science Honours Project (COMP4905)

Mohammed Abushawish

100857775

Carleton University, Ottawa, Ontario

Dr. Dwight Deugo

April 13, 2016

# Acknowledgements

I would like to take the opportunity to thank Dr. Dwight Deugo not only for agreeing to supervise me for this honours project, but also for all that he has taught me throughout his mobile computing courses. Without the courses that he taught, I highly doubt that I would have such a passion for mobile development or even pursued an Honour's Project such as this.

I would also like to thank the various developers whose open source libraries I had the pleasure to use - CreditCardEntry, Android Asynchronous Http Client, Butterknife, and node-sqlite3.

## Table of Contents

1. Introduction	4
1.1 Problem	4
1.2 Motivation	5
1.3 Goal	7
1.4 Objectives	8
2. Background	10
3. Approach	13
3.1 High Level Overview	13
3.2 Client	16
3.3 Server and Database	21
4. Results and Validation	26
5. Conclusion	28
5.1 Future Work	29
References	30

# 1. Introduction

Automation has introduced a lot of conveniences in day-to-day life to many people; things that required interaction can now be done in the background without us thinking about them saving us energy and time. In this day and age, people want things to be done quickly, simply, and, if possible, without our interaction. From smart-thermostats to autonomous robotic vacuum cleaners, society is moving to interaction-less technologies that takes advantage of 21st century technology.

However, one action that is done by millions of people daily, yet seems awkwardly stuck in the past, is parking a car. Specifically, paying for parking. This report will examine the current problems with parking payments, why I decided to pursue developing something to fix these problems, what I aimed to accomplish with my concept, how I developed the solution, and what was finally produced. In the process I will also outline what I learned about multiple new technologies, both physical and software, challenges that I faced, and solution that I pursued.

## 1.1 Problem

Currently, paying for parking is simply inconvenient; too much interaction is needed by the user and not enough automation is in place. The individual who is parking faces multiple inconveniences, questions, and problems that arise from manually having to pay for parking:

- a. Having to make multiple trips - walk to the payment machine, pay, return to place stub on the dashboard, and then finally walk away. Especially when considering individuals with disabilities or the aggravation during inclement weather

- b. Not having a set-and-forget option, that allows the user to simply “forget” and not worry about paying for the parking, thus avoiding worries and tickets
- c. Needing to hold cash and coins to feed a parking meter. Not only inconvenient but also dangerous as this could be a target for thieves
- d. Incorrectly plan for how long to prepay for, thus either overpaying or underpaying
- e. Be unclear about the parking payment rules and prices, therefore incorrectly paying and being surprised at the accrued expense
- f. Not having a convenient automated solution to track and view parking payments, thus relying on manually logging these payments

## 1.2 Motivation

My initial motivation came about from my own annoyance and inconvenience that arises from having to pay for parking. Then it became clear that I am not the only one who faces such inconveniences, every aforementioned problem was not just something that I imagined, but they are true problems that affect people. During my research, it became obvious that not only is paying for parking inconvenient, but that it could be down-right difficult for some individuals as reported by *New Westminster Record*, a town newspaper that criticized the introduction of parking pay stations by stating, “customers don't always have their licence plate numbers memorized, and more walking to reach a pay station is a deterrent, especially during bad weather or for people with mobility issues” [1]. Utilizing automation to automate parking payment will not only make it more convenient to park, but much more possible for some individuals. Such as those with disabilities, especially when considering the need of number of trips needed at some pay stations,

or forgetful persons, who may forget their license plate number or simply forget to pay for the parking.

With all the current automation technologies being introduced for a variety of things, from home automation to work automation, I noticed a general absence of automated parking payment solutions. There were some solutions to simplify the parking payment parking process. For example, to alleviate the need of having coins, change, or a credit cards on hand or having to constantly travel back to the vehicle to buy additional time, *iParked.ca* was developed. Created at Carleton University by this Honour's Project's very own supervisor, Dwight Deugo, *iParked.ca* aimed to make it safer, easier, and simpler to pay for parking [2]. And while it did indeed fulfil those goals, it did not automate the parking payment process, thus allowing room to introduce even more convenience to make the process completely set-and-forget. However, the reasoning behind the need for interaction by the user, not matter how limited, as is the case for *iParked.ca*, to pay for parking stems from the fact that, until only a few years ago, cars themselves have progressed very little in the high-tech aspect. There is little within the car that allows wireless communication between the car and parking payment system thus needing a human to bridge that gap. Which brings me to my next motivation - new technologies which can alleviate this downside have been created.

Within only the last few years automobiles have quickly been attaining high-tech features within them, such as Google's Android Auto, meaning they are behaving more and more like computers [3]. These tech-packed vehicles, called "connected cars" [4], are able to communicate and connect through wireless technology, such as Bluetooth, which when combined with other new technologies such as Bluetooth beacons make solutions such as the project detailed in this paper not just imaginable but very practical and implementable. These technologies combined

can be used to provide the driver a truly automated, hands-off, and convenient approach to paying for parking with little to no interaction.

Furthermore, it was very motivating to work on this project from a technical standpoint. First, I do not own nor have access to a connected car, thus challenging me to create novel solutions using technologies that I can affordably and easily access without the need to buy a new car, thus why this project is more of “proof of concept” rather than a production product. I would develop and combine multiple technologies, such as an On-Board Diagnostics (OBD) adapter, to replicate a “connected car” in my personal not-so-connected car. Second, I would have the opportunity to develop and learn about products and technologies that are completely new and foreign to me, yet excite me, such as Bluetooth beacons, financial APIs, Node.js server, OBD, and so on. Third, even with all the new technologies that I will be using, I would also be able to develop on Android, an operating system that I am an immense fan of and familiar with, thus insuring this project does not turn so challenging that it is a chore but remains personally fun.

The real-world inconvenience, and even outright challenge to some individuals, of paying for parking, the lack of fully interaction-free alternative solutions, and the opportunity to experiment with, develop on, and combine new technologies, were the motivating factor to pursue and work on *Park-N-Go*.

## 1.3 Goals

The concept proposed throughout this project aims to fulfil several goals. It aims to make the manual interaction of paying for parking unnecessary, thus making parking more convenient, safer, and simpler. It provides the driver an automated solution to pay for parking using a

“connected car” and Bluetooth beacons. In the process, the project also aims to replicate some of the technological functions of a “connected car” by combining several affordable technologies, this allows non-connected cars to utilize this solution. While automated, the solution will still provide visual feedback to the driver; allowing them not to only see what is happening but override the automation and cancel the parking action if it is not intended.

## 1.4 Objectives

The solution’s objective is to remove the inconveniences and difficulties that are associated with an action that is performed so frequently by drivers - parking payment. This will be done by developing the solution that is outlined within this report, *Park-N-Go*. This solution will involve several hardware components, both inside and outside the vehicle, as well as software that will tie all these components together.

Bluetooth parking beacons are located on the outside of the vehicle, transmitting and indicating a parking location. On the inside of the vehicle, the solution will combine an On-Board Diagnostics (OBD) adapter and an Android device to replicate *Android Auto*, or an equivalent “connected car” dashboard system. The OBD will communicate with the device that the car has been put in “park” while the device will communicate with the Bluetooth beacons. When the vehicle arrives at a parking spot, which hosts a Bluetooth beacon, and is turned off, the vehicle will begin to track the duration of the parking. The end of the parking duration will be indicated by leaving the Bluetooth parking beacon’s region, and so the system will charge the user as necessary using a financial application program interface (API).



## 1.5 Outline

This report has so far examined the problem that the solution aims to eliminate, the motivation behind implementing the solution, the main goals of it, and the objectives behind implementing these goals. Throughout the rest of the report, the history of this problem will be explored, as well as related and alternative works, including why they do not solve the problem.

Then, the approach behind designing, developing, and implementing my project will be detailed, including difficulties, problems faced, and the solutions implemented. This will lead to exploring the validation, use, and results of my final product. The report will conclude by giving a closing a statement, including what future work can be done to improve my product.

## 2. Background

As described earlier, paying for parking is as an action that is best described as inconvenient and a hassle, especially being something that is performed sometimes multiple times a day by a driver. This is a surprise when in the modern world repetitive actions are usually the first to be automated, especially when they are inconvenient. The reasoning behind this however is that, as discussed before, little exists in an automobile to allow it to communicate and transmit information between itself and a parking spot. Until very recently, cars have had very simple computers onboard, capable of basic functions with the goal of worrying only about what is on the inside of the car rather than what is also on the outside, they are not context aware. Cars are not able to communicate payment processing information to be charged, they do not have the basic ability know that they arrived to a particular spot, specifically a parking spot in this case, and by extension do not know that they left that particular location. This reason causes creative constraints on solutions that could be developed to create automated payment solutions.

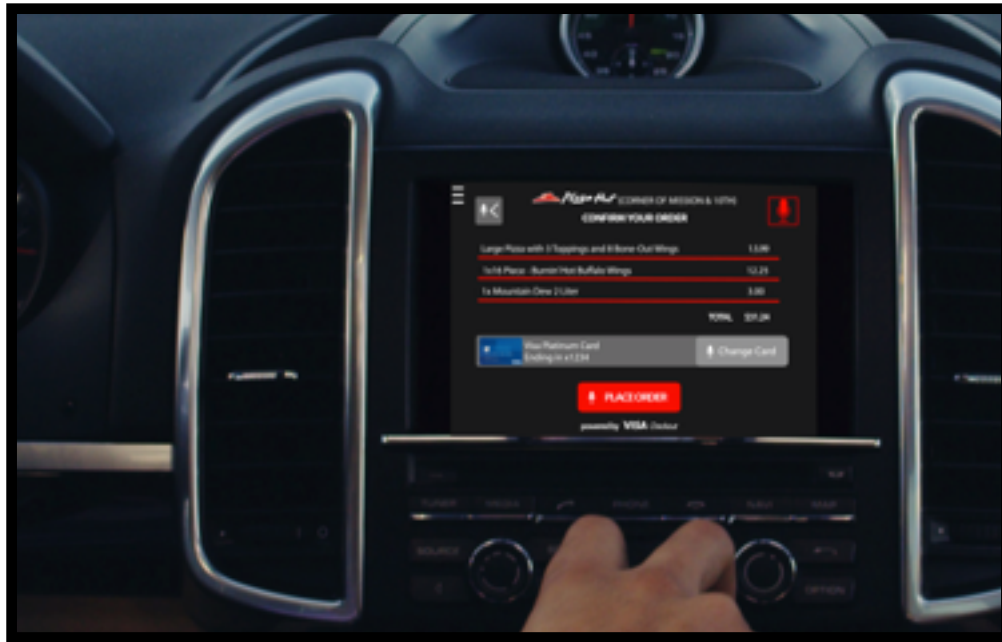
Therefore, solutions to the inconveniences of parking payment have mostly revolved around how to minimize the interaction and make the interaction easier, rather than eliminating it completely. For example, *Parkmobile*, a parking payment system which was introduced in several cities in Australia, allows users to pay for their parking time by using scanning a QR code located at the parking location using a downloaded mobile application. While convenient because it does remove the need to carry money, this solution removes little from the other inconveniences; users must still walk to the QR code itself, they must still return to scan the QR code again if more time is needed, they must know how long in advanced to pay for, and so on. Little is done by a solution such as this to remove the parking payment hassle.

Other solutions include *iParked.ca* and *PayByPhone*. As discussed earlier, *iParked.ca*, removes the need to interact with a parking payment machine by hand, instead allowing a particular user to pay by texting through their phone. Similar in nature is *PayByPhone*, which allows users to pay using a mobile application at participating parkings by entering the parking location code, the duration period, and then paying through their application [5]. This allows the user the convenience of not worrying about the expiration of their ticket, due to the ability of paying remotely, and removes any difficulties and dangers of paying the parking payment machine by hand. While eliminating the inconveniences of paying by hand, users must still interact to some degree with the payment system. They have to send the text message or use the mobile application, they have to make sure they have cellphone reception, they must know the parking location identifier, and no on-screen information is provided to the user regarding the parking rates and rules for the given parking location. These options are not fully “set-and-forget”.

However, to the defence of all the previously mentioned system, the reason that some interaction is still required is simply because, as mentioned earlier, until recently no vehicles with connectivity or “smart” abilities were being developed. And while slow, this fact is changing. Internet-of-Things is becoming reality not only in the home but in vehicles as well - introducing connectivity and networking through electronics, software and sensors, thus gaining the ability to send and receive data. By 2020, more than 250 million vehicles worldwide will be connected with Bluetooth and internet [6]. Combined with technologies such as Bluetooth beacons that have become very affordable and simple, a whole range of new and creative solutions to problems that previously had been limited by the lack of connectivity has been introduced.

For example, Visa, partnering with car manufacturer BMW and restaurant chain Pizza Hut, has already demonstrated the creative abilities introduced by a “connected car”. In the 2015

Mobile World Congress, Visa demonstrated a proof-of-concept solution that allowed a user to order a pizza on the go using an in-dashboard system of the vehicle, which then automatically charged the user's credit card, and lastly, using Bluetooth beacon technology, notified the restaurant's staff when the Bluetooth-enabled connected car has arrived [7].



*Figure 1: Visa's Demonstrated the Ability to Pay and Pickup a Pizza Order through a Connected Car and Bluetooth Beacons*

Just as Visa introduced the pizza ordering ability as a proof-of-concept of their vision to purchasing stuff on the go, my solution also aims to be a proof-of-concept. *Park-N-Go* is a proof-of-concept of paying for parking differently, removing the current inconveniences of paying for parking, and envisioning the possible future.

### 3. Approach

As described previously, an Internet-of-Things enabled connected car is what would be able to utilize the solution proposed throughout this project. To be able to park, communicate with a parking spot, and charge the user's credit card when finished. However, I do not have access to a connected car and yet I wanted to have my solution work inside my own vehicle.

#### 3.1 High-Level Overview

I began by sketching a high-level overview of the components that were needed and how they would interact with one another.

First, observing the dashboards of connected cars, they appeared to function and look very similar to a regular mobile operating system, hosting applications, touchscreens, Bluetooth, and so on. Thus, I decided to use an Android smartphone as the main hub that would tie all the required components together and be the communication portal, replicating the dashboard computer of a connected car. I decided to develop with Android, rather the iOS, especially



*Figure 2: An Android Auto Dashboard is Very Similar to the Regular Mobile Android*

because *Android Auto* is an Android operating system, making my concept closer to real-world connected car implementation. Running on this Android device would be an actual mobile

application that I would develop. The purpose of this application is to, programming wise, tie all of the succeeding components together so they can work seamlessly.

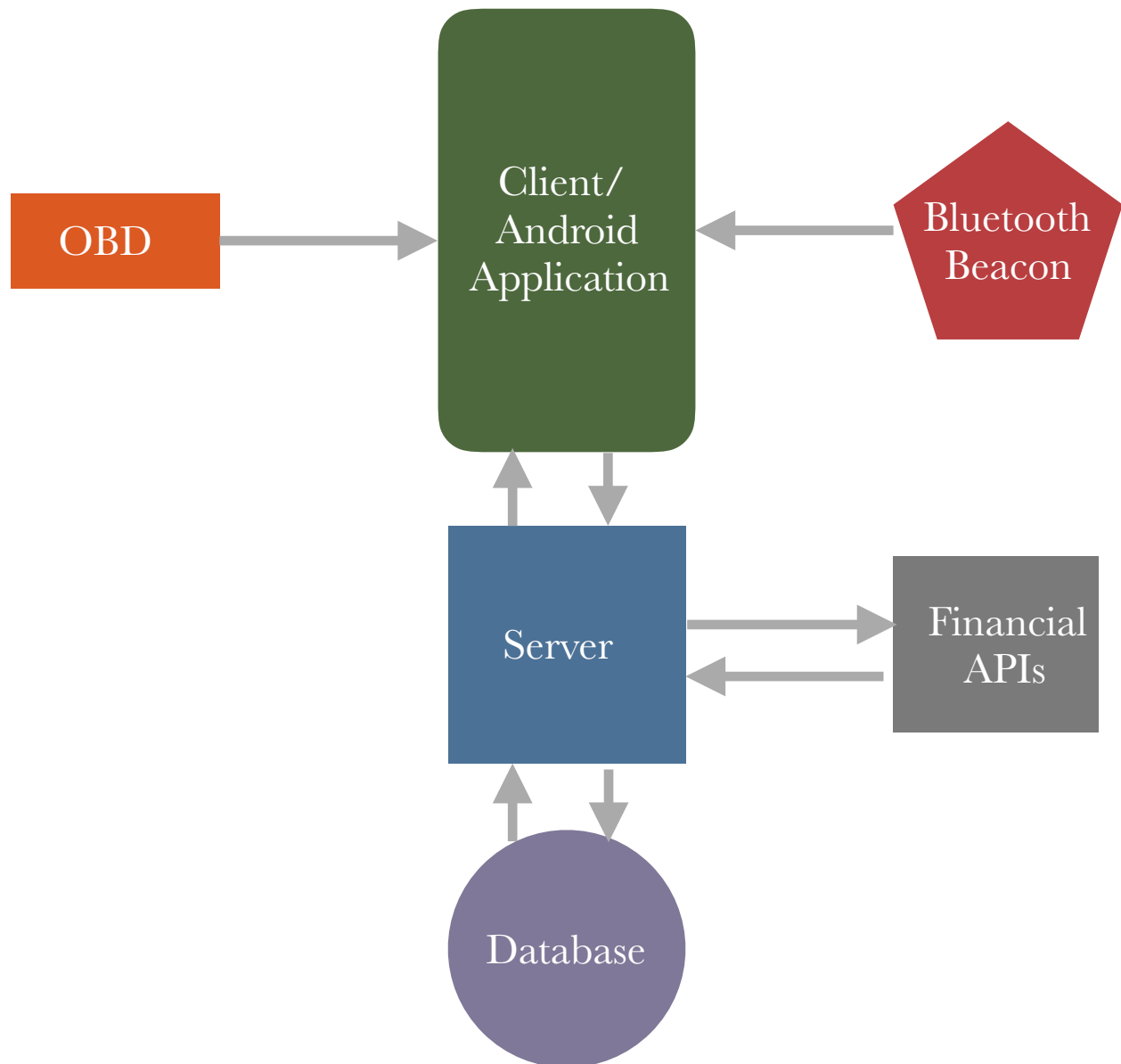
Second, unlike a connected car's computer, my Android smartphone does not have access to basic real time data about the car on its own, thus I needed to bridge the car and the phone using an on-board diagnostic (OBD) adapter. By plugging into the car, the OBD adapter allows access to a vehicle's various system's status and reports, and could be programmed to know if the car is put in park, something which I thought would be essential and interesting to attempt.

Third, on the outside of the vehicle would be where the Bluetooth beacons would be found. These Bluetooth beacons represent the parking location and would transmit their location and presence, allowing the Android device to know when it has arrived to a parking location and when it has left it. I decided to utilize Bluetooth beacons for parking detection because they fill the void between very close-proximity, such as NFC, and larger proximity, such as GPS, which can imprecise.

Fourth, I would require a server and a database for the *Park-N-Go* application to communicate to. The server and database would receive and store the user's personal information, when a user has arrived, when the user has departed, calculate the total amount due, and using financial APIs charge the user's credit card for the total due. I needed a server because of security concerns of trusting the client to charge itself the proper amount and a server would give the ability for the parking attendants to keep track and view parking history.

When a driver with an Android device with the *Park-N-Go* application installed on it arrives within range of a Bluetooth beacon enabled parking spot, the application is triggered. The application waits for the user to shut off the car, indicating he has indeed parked and not simply passing through, which when done triggers the application to inform the server that a user

has arrived. Through the server, the user and his arrival time are stored on the database. When the user departs a parking spot, which is triggered by leaving the Bluetooth beacon's range, the application informs the server, which then calculates the total time and amount due and charges the customer as needed.

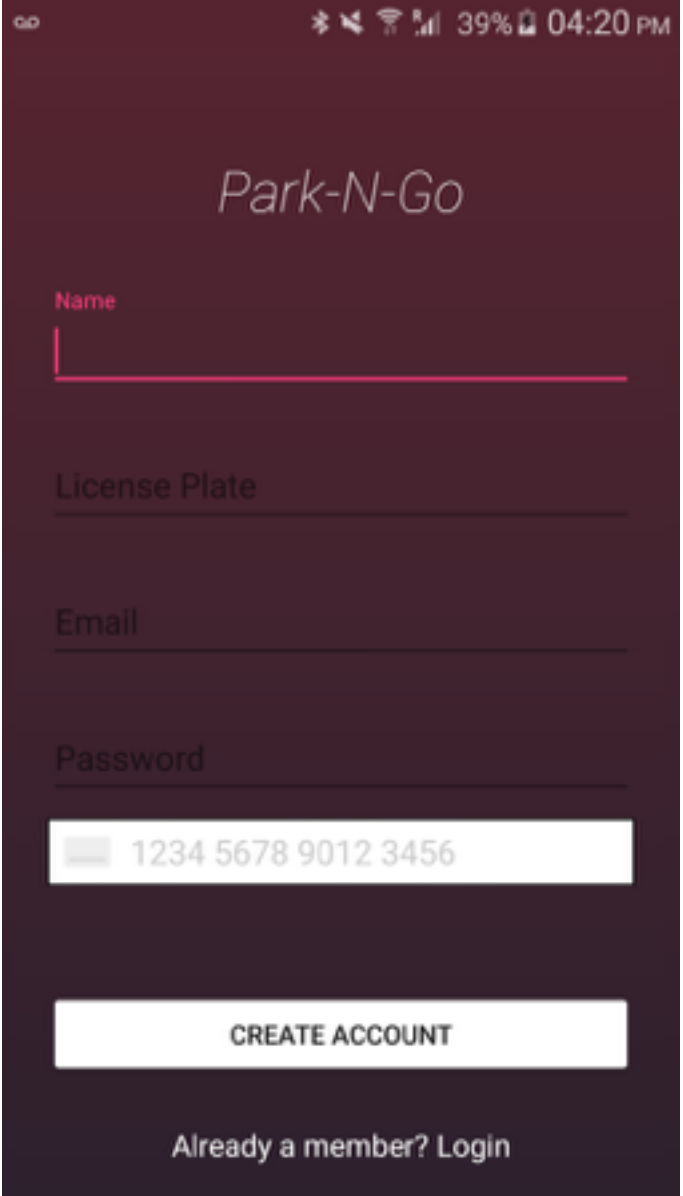


*Figure 3: A High-Level OverView of the Components of Park-N-Go*

## 3.2 Client

Developing the client side began by creating an Android application through Android Studio. The application, as described earlier, aims to connect all the various components together. Together, they will automate the process of departing and arriving from a parking location and informing the server, while also giving the user an easy way to look into their parking payment history.

First, I needed to implement a screen that allows the user to configure and register their information. Exactly only once, unless the user logs out, the application will begin off by asking the user to login or register. To register, an Android Activity class was developed for the user to enter their vehicle license plate (this is equivalent to a username), full name, email address, and valid credit card information. I implement CreditCardEntry, an open source library, that conveniently allows a user to enter their credit card information and proceeds to validate the credit card using a Luhn algorithm, a checksum formula to validate a given card number [8].



*Figure 4: The Initial Registration Screen. Notice the CreditCardEntry Library Field at the Bottom*



Next, when all the fields are entered and registration submitted, it was planned to have all the information, including the credit card information, be sent to the server and stored on the database. However, in my research to make this secure, it became clearly that storing credit card details on my server in the real world would be next to impossible due to the strict PCI-DSS standards, an information security standard by the major credit cards. With Deugo and Matic stating, “SAQ [the attestation of compliance] requires that servers are locked down almost as tightly as Fort Knox” [9]. Thus it became clear that I would need to use a third party payment processing gateway, especially when I would eventually need to bill the customer’s credit card.

I opted to use *Stripe*, a payment processing application program interface (API), that makes it incredibly simple to accept credit card payment through a mobile application. By implementing Stripe’s Java library into my Android project, I was very easily and securely able to generate a secure unique token from the customer’s given credit card information. Simply put, this token can then be used throughout Stripe’s API to charge the customer as needed on that credit card. The customer’s details, alongside Stripe’s token in hand, were sent to be stored on my server (the server will be discussed in more detail in the next section).

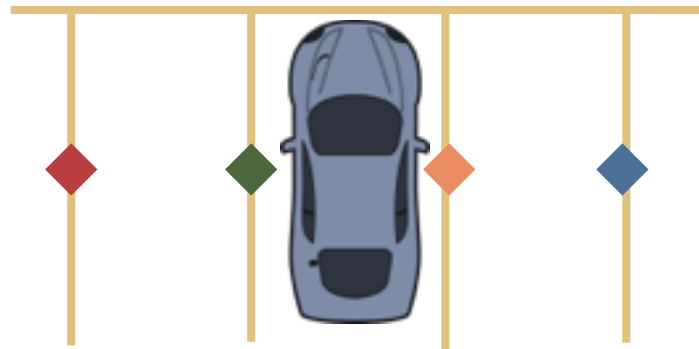
With the user successfully registered, their credentials get stored in the application’s SharedPreferences for later use. Following this initial setup, if the application is open again it will display the user’s previous parking history details. Next, I integrated “Android Asynchronous Http Client”, an open source library that immensely simplifies HTTP requests, into the *Park-N-Go* application. The library allows for asynchronous, background-thread network calls while providing callbacks in the main thread, all this without needing to worry about making and handling background threads and async calls myself.

With the prerequisites complete, the objective of the application is to automatically detecting that it has arrived or departed from a parking location without the user's interaction can now be examined. To complete this, at first I thought about using the phone's GPS. However, the GPS has a few issue; first, horrendous use of energy when always polling, something that would always be needed to detect if the user arrived to a parking spot; second, it is not very precise, sometimes it can be up to 8 meters off the actual location; and third, users, including myself, would not be appreciative of an always polling GPS, one that could hypothetically know where they always are. Thus, I opted to utilize Bluetooth beacons, specifically *Estimate Beacons*.

Bluetooth beacons are devices that broadcast their presence by transmitting their Universally Unique Identifier (UUID) through Bluetooth Low Energy (BLE), which then can trigger certain applications on compatible devices. What this means, in my project's case, is that every beacon can do considered a unique parking spot that only triggers when the device is within a certain distance of it. Unlike GPS, due to being BLE, these Bluetooth beacons are less-power intrusive, have more precise distance approximation, and do not receive any data, only transmit it much like a lighthouse.

Using the Estimote SDK, it was quite simple to implement beacon monitoring into my application, even when the application is not running in the foreground. The application will scan for a specific given UUID, indicating a parking spot, at 30-second intervals, and when a beacon is discovered its distance will be calculated. If the calculated distance between the device and the Bluetooth beacon is less than a certain amount of meters, the car is considered to be in the parking spot. However, during experimentation an important obstacle quickly appeared: the car itself. Due to the metal frame, the car behaves to a "faraday cage", blocking most, if not all, of the beacon's transmitted signal from reaching the Android device [10].

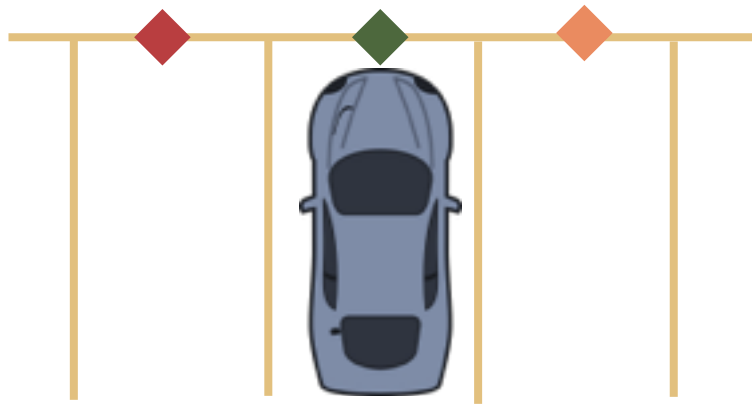
Originally, I intended to have a two Bluetooth beacons to every parking spot, a beacon to the left and one to the right on the spot. The intention was to make the ranging detection more accurate, guaranteeing that the car was indeed in a given parking spot if the two opposing beacons were equal distance apart from the Android device. However, using this method it was impossible to even detect the Bluetooth beacons - the metal doors of the vehicle, which would face the beacons, were simply too thick.



*Figure 5: The Initial Beacon Placement. The Diamonds Represent Bluetooth Beacons*

To circumvent this issue some changes had to be made. First, the Estimote beacons were relocated to the front of the vehicle. The front windshield of a vehicle has a much larger surface area of glass compared to the side door windows, and glass is much more forgiving on the broadcast signals of the Bluetooth beacons, with Estimote stating that glass has “low interference potential” versus the metal’s “very high interference potential” [11]. Second, after being relocated to the front, the beacons were raised on a pedestal to the same horizon as the glass windshield of the vehicle. Lastly, the Broadcasting Power of the beacon was increased. As the term suggests, the “Broadcasting Power is the power with which the beacon broadcasts it’s signal”, meaning the signal can travel farther, and more importantly, has greater potential to penetrate through interference obstacles, but with a negative consequence on the beacon’s

battery life [12]. These changes resulted in the Android device successfully detecting the Bluetooth beacons when the vehicle arrived to the parking spot, and gave a fairly accurate distance reading. With the Android device, which represents the vehicle, successfully within range of the beacon to be considered in the parking spot, the application awaits confirmation that the user has indeed intended to park and is not simply passing by. This confirmation is done by using an on-board diagnostic (OBD) Bluetooth adapter found in the car.



*Figure 6: The Redesigned Beacon Placement. The Diamonds Represent Bluetooth Beacons*

While originally intended to diagnose engine problems, a vehicle's OBD has dramatically increased in capability, offering access to a number of parameters about the status of the vehicle. Using the *OBD-II Java* API, an open source library to interact with an OBD adapter, I was able to interact with my installed ELM327 OBD Bluetooth adapter to determine whether the vehicle was indeed physically stopped and parked. Within the application, a Bluetooth socket is created and connected to my ELM327's MAC address and UUID. Once the socket is created and connected, the OBD Bluetooth adapter can be interacted with.

Due to the OBD lacking a parameter identifying that the vehicle's gearbox was placed in "park", I combined two alternatives to signify a park. First is when the user shuts off the vehicle. Programmatically, this is signified by the application failing to connect to the Bluetooth OBD

adapter due to the adapter losing power, with the serendipity being that a user can therefore utilize *Park-N-Go* without a Bluetooth OBD adapter. Second is when the user puts the ignition in accessory mode. In this case, the OBD should return a status code indicating the revolutions-per-minute (RPM) of the engine to be 0, when the RPM is requested. This request is performed by the library by sending a specific hex Parameter ID to the OBD, which then replies with a hex which when parsed indicates the RPM of the engine. This request is performed in the application as often and alongside the scans for the Bluetooth beacons, once every 30 seconds.

The Android application, the Bluetooth beacons, and the on-board diagnostic (OBD) combined allow the automation of the client to be context aware when a user arrives and departs from a parking spot, and completes the client portion of the *Park-N-Go*.

When a user arrives to a parking spot and parks, the *Park-N-Go* application appears on the Android device informing the user that he has arrived parked at a particular parking spot, a countdown begins

### 3.3 Server, Database, and Implementation

While the client had the necessary components to be context aware of when it had arrived and departed from a parking spot, it needs a server to compliment it so that data can be exchanged as a result of the context based information. For example, it is not enough for the client to simply know that it has arrived to a parking spot, something must be done with that knowledge. Also, by having administrative access to the database, the parking authorities can also utilize it to easily confirm that a particular vehicle has properly been recorded as parked (therefore be charged on exit), track various parking history, and so on.

To develop the server and database, I opted for a Node.js, Express Framework, and SQLite3 combination. I had only developed on Express once previously, but I found it extremely

easy and quick to use, and was really excited to have the opportunity to gain some more knowledge in it. I was also interested in utilizing JavaScript, the programming language of Node.js, because I have used it so seldom that I want to be challenged. Like the Android application on the client side, the server also makes use of the Stripe API libraries.

After creating a new Express project and importing SQLite3, two tables were created. The first being a table to store users as they sign up and the second being a table to store parking sessions that are performed by the users. Next, I had to create several routes that would be utilized by my project. Routes is the way that a server responds to a client's request to a given endpoint and HTTP request method [13]. In chronological order, I would need a routes for registration, login, entering a parking, leaving a parking, and listing parking history. By examining these routes, a better understanding of the server's and database's components can be achieved.

#### **‘/newAccount’**

As described earlier, when a user opens the mobile application for the very first time, they are greeted with a registration screen which, alongside accepting their information, also tokenizes their credit card details using Stripe's API. After pressing the register button on the *Park-N-Go* application, the client submits an HTTP POST request to the appropriate route on the *Park-N-Go* server. Within that POST request are the parameter's for the user's license plate, full name, email address, password, and their credit card token generated by Stripe. When the POST request is received by the server, two security related operations occur. First, using Stripe's API, a customer object is created using the given credit card token. Unlike a Stripe credit card token, which can only be used exactly once due to it being generated from the client, a customer object, which is created by the server, can be charged as often as needed. Second, the user's password is encrypted using a "bcrypt" algorithm which hashes the password in slow speeds, making it

impossibly time-consuming to unhash the password, and thus extremely safe and effective. Then, the Stripe customer ID and encrypted password, alongside the other aforementioned parameters, are stored into the “users” table. With the server returning a success message to the client, the user is automatically logged in into the mobile application.

### **‘/login’**

Login, whether automatically following registration, or by user sign in, functions much like the previous “/newAccount” route. The HTTP POST request provides the license plate and password. The route confirms that the license plate exists within the “user” table, and if so, compares the given password to the hashed password. A success or error is returned to the client, thus having the application login or not.

### **‘/parked’**

When a user arrives to a parking spot and parks, the client produces an HTTP POST request to the “/parked” route including the parameters indicating the license plate, which identifies the user that has parked, and the parking spot number, which is retrieved from the Bluetooth beacon by the client. Once requested, the route confirms that the license plate is a registered user and creates an entry into the “parking” table on the database. The entry contains the license plate relating to the user, the parking spot number, and the arrival time. When a row is inserted, it is missing some column entries: the time of departure, total minutes parked, and total charged, these missing columns indicate an active parking session.

The significance of the active parking session is twofold; first, when the user departs the total amount due will be calculated using the time difference between their initial arrival time and current departure time; and second, if a non-registered driver, a driver without *Park-N-Go*, or a driver without any of the required client components dishonestly parks at the parking spot,

then they will be ticketed by the parking attendant because their vehicle's license plate would not exist within the "parking" table.

### **'/leftparking'**

When departing a parking spot, the client, as before, produces an HTTP POST request to the "/leftparking" route with one parameter indicating the license plate. The route on the server looks into the database's "parking" table by searching for a row with the given license plate and an empty entry for the departing time column, indicating an active parking session. An arbitrary amount is then multiplied by the arrival and departure time difference, which produces a total charge amount. Through the parking row's foreign key, the license plate, the Stripe customer ID can be located. If the charge is greater than 50 cents, Stripe's API, which as mentioned previously is integrated into the server, can then be used to charge the customer's credit card using their customer ID. Once the customer is successfully charged, the parking row is then updated to indicate the departure time, the total time parked, and the total amount charged.

The importance of the timestamp for the arrival and departure being generated by the server is important to note. Initially, I foolishly had wanted the client to personally track when it has arrived, when it has departed the parking spot, and calculate the amount due through the time difference. However, it quickly occurred to me how much of a security risk that would be as the client could very simply change the device time, making the amount due incredibly low or even negative. This spoke volumes to the need for the server, even further beyond it being convenient by keeping track of parking sessions and users.

### **'/listofparking'**

By manually opening the *Park-N-Go* application on the client, the user is able to see their previous parking history. This is done by producing an HTTP POST request to the "/"



leftparking” route with one parameter for the license plate. When the request is received server side, a query is performed on all the rows for the given license plate in the “parking” table. The rows are then sent to the calling client as a JSONArray of JSONObject, which, when parsed client side, are then displayed to the user. This can be used to not only see past historical parking transactions by a particular user, but to also see a current pending parking session.

The final component operating on the server is “ngrok”. Allowing a local webserver to be exposed to the internet using a secure a tunnel [14], ngrok easily allows a localhost server to be accessed from anywhere in the world. This makes it incredibly simple for a client to communicate with a server without being on the same network and without needing to pay hosting fees for a cloud provider.

## 4. Results and Validation

The grand problem for which *Park-N-Go* set out to eliminate was the lack of full automation in the parking payment industry. By introducing automation, *Park-N-Go* aimed to correct several problems with the current parking payment options that can be grouped into a few categories: timing, ease of use, and convenience.

The other alternatives require time to use and interact with the payment system, no matter how little, *Park-N-Go* requires almost none. Outside of the setup process, which thanks to the likes of Stripe, is seamless and take seconds to setup, the user may choose to never physically interact with the device again. *Park-N-Go* is truly time saving; setup takes only several seconds, payments succeed in less than a second, and a user no longer has to do any actions beyond parking the car.

While most of the parking-payment alternatives are simple and straightforward, nothing matches the ease of use of *Park-N-Go*. No longer does a user need to worry about how to properly use the payment machine, or worry about understanding the cost of parking for parking location, *Park-N-Go* automation allows the system to automatically take care of payment, and provides visual feedback to explain the parking costs to the user.

No longer does a user need to carry cash or change around, not even a credit card is needed any longer. Once the user registers himself on the *Park-N-Go* application, he no longer needs to worry about taking the wallet out to feed the machine. Another convenience as a result of *Park-N-Go*, is the ability to view historic parking logs, without the need of logging and storing these expenses.

On a personal validation level of project, using my personal vehicle and the various components required by *Park-N-Go*, I was able to pull to a parking spot, which I marked with a Bluetooth beacon, park then leave and was correctly charged as needed using a test credit card number provided by *Stripe*.

While it is hard to validate my concept on a larger scale because all the needed components and need of a vehicle make it difficult for my classmates and friends to beta test my product, similar ventures that utilize a combination of Bluetooth beacons, vehicles, and financial technology can be examined. As mentioned earlier, for example, Visa in partnership with Pizza Hut developed a system that orders and pays for a pizza directly from the vehicle then informs the employees as the vehicle arrives to the restaurant using Bluetooth beacons proximity. While that was venture was simply a proof-of-concept prototype, it excited Visa about other possible ventures in the future using the same technology, with Visa's executive vice president, Jim McCarthy, stating, "as the number of connected vehicles on the road increases, so does our ability to bring this secure, frictionless option of online commerce to consumers everywhere" [15]. This is validation that while my project may be proof-of-concept, as was Visa's pizza ordering project, it opens the doors to many possibilities, including perfecting my concept.

## 5. Conclusion

There has always been an inconvenience revolving around paying for parking; whether the hassle of needing cash, needing to make multiple trips to for a pay stub, forgetting to buy more time, accidentally paying overtime, or one of the other dozen problems. This is without even mentioning the safety concerns or complete difficulty that arises from parking payment. In such a high-tech world such as ours, these kinds of difficulties and inconveniences should no longer be reality. This is even further true when considering all the automation that we see around us for minuscule repetitive tasks, something that paying for parking definitely is. Yet, this is reality because of lack of connectivity-technology within vehicles. Thus a creativity limit was imposed on battling this issue. A handful of application attempted to circumvent the inconvenience of parking payment even with the creative limit imposed. Yet, none were fully automated solution and left something desired. However, the reality is changing. “Connected cars” are the future and allow for remarkable things to happen. And so, *Park-N-Go* was motivated.

With no access to a connected car, *Park-N-Go* utilized and tied several various technologies together to operate and be proof of concept. Being the central hub, the mobile application running on the Android device replicated the dashboard of a connected car and has financial processing access. When the vehicle arrives to a Bluetooth beacon enabled parking and is parked, *Park-n-Go* is aware. It is able to begin timing the parking duration, and, as the vehicle departs, charge the user’s credit card automatically for the duration.

*Park-N-Go* truly represents the future in paying for parking. No longer does a user have to interact with a machine or even a phone. Using context-aware technologies, such as Bluetooth

beacons and an OBD, paying for parking will truly be an automated process and be completely “set and forget”, or should I say completely “park ’n’ go”

## 5.1 Future Work

As discussed several time in this paper, I do not have access to a connected car, and thus was required to use an Android phone and an OBD to represent the dashboard of a connected car such as *Android Auto*. However, in the future I would definitely prefer developing my system on a connected car. These cars are the future, and would take *Park-N-Go* from a proof of concept to reality.

Also, for the future, I would find more uses for the application’s main activity. Even with the parking history list, it feels quite bare and useless. Possibly allow the user to research into the parking’s prices, hours of operation, and so on.

## References

- [1] <http://www.newwestrecord.ca/news/parking-pay-stations-deter-customers-1.552017>
- [2] <http://worldcomp-proceedings.com/proc/p2012/SER4402.pdf> page 6
- [3] [https://www.android.com/intl/en\\_ca/auto/](https://www.android.com/intl/en_ca/auto/)
- [4] <http://mashable.com/2011/02/26/connected-car/#HYiWg.Lr3uqD>
- [5] <https://www.paybyphone.com/how-it-works/parking>
- [6] <http://dupress.com/articles/internet-of-things-iot-in-automotive-industry/>
- [7] <http://www.businesswire.com/news/home/20150302006046/en/Visa-Pizza-Hut-Accenture-Develop-Connected-Car>
- [8] <https://github.com/dbachelder/CreditCardEntry>
- [9] <http://worldcomp-proceedings.com/proc/p2012/SER4402.pdf> page 3
- [10] <https://forums.estimote.com/t/beacon-outdoor-and-passing-cars/1254>
- [11] <https://community.estimote.com/hc/en-us/articles/202041266-Best-practices-for-installing-Estimote-Beacons>
- [12] <https://community.estimote.com/hc/en-us/articles/201636913-What-are-Broadcasting-Power-RSSI-and-other-characteristics-of-beacon-s-signal->
- [13] <http://expressjs.com/en/starter/basic-routing.html>
- [14] <https://ngrok.com/>
- [15] <https://www.accenture.com/us-en/success-visa-connected-commerce-car.aspx>