

Cloud Based Smart Parking System using IoT Technology

Zubydur Rahman

Ahmed Tanzeer Hossain

Nibras Abdullah Karim

Mohammad Jayeed

Supervisor:

Dr Amitabha Chakrabarty

Department of Computer Science and Engineering

August 2017



Inspiring Excellence

BRAC University, Dhaka, Bangladesh

Declaration

This is to certify that this final thesis report '*Cloud Based Smart Parking System using IoT Technology*' is submitted by the authors for the purpose of obtaining a degree in Bachelors of Science in Computer Science. We hereby declare all the work presented in this thesis paper are authentic and any inspiration of the work have been accredited with proper referencing.

Signature of Supervisor

Dr. Amitabha Chakrabarty

Signature of Authors

Ahmed Tanzeer Hossain

Mohammad Jayeed

Nibras Abdullah Karim

Zubdur Rahman

Acknowledgement

First and foremost, all praises to Almighty God, the most high and merciful.

Our most sincere gratitude to our thesis supervisor Dr. Amitabha Chakrabarty. Without his assistance and dedicated involvement in every step throughout the process, completing this thesis might have proved to be an impossible task.

Also, we want to thank our family, friends and well-wishers who inspired, encouraged and fully supported us through every trial that came our way, who helped us not only financially, but morally and spiritually.

Finally, we are very thankful to BRAC University, Bangladesh for giving us a chance to complete our B.Sc. degree in Computer Science.

Abstract

The number of vehicles on Bangladeshi roads are expected to increase by 360 000 by 2035, reducing average speed from an already miserly 6.4 kph to 4.7 kph – walking speed for most people. [18] While major infrastructural improvements are scheduled to take place to reduce the number of cars on the road, we proposed a solution directed at traffic space management accelerate the process.

Studies have shown illegal encroachments by motor vehicles parking on roads have a massive effect of the flow in traffic [19]. The purpose of this paper is to remove the all too familiar sight of cars parked on the roads by introducing smart parking lots.

Table of Contents

| | |
|--|----|
| Chapter 1 | 9 |
| 1.1 Introduction..... | 9 |
| 1.2 Motivation..... | 11 |
| 1.3 Problem Statement | 11 |
| 1.4 Solution..... | 12 |
| 1.5 Thesis Outline | 12 |
| Chapter 2..... | 13 |
| 2.1 Literature Review..... | 13 |
| Chapter 3..... | 17 |
| 3.1 System Flow..... | 17 |
| 3.2 Algorithm | 18 |
| 3.1 System Architecture..... | 20 |
| Cloud-Based Server | 20 |
| Local Unit | 20 |
| Software Client | 21 |
| Chapter 4..... | 22 |
| 4.1 Hardware Components..... | 22 |
| Arduino Uno vs. Raspberry Pi:..... | 22 |
| Sharp gp2y0a21yk_e Infrared Reader and sensor: | 23 |
| Keypad | 25 |
| LED and Resistors | 25 |
| 4.2 Hardware Prototype | 26 |
| Reading Data from Sensors | 26 |
| Check condition of all parking spots | 27 |

| | |
|---|----|
| Book a Parking Spot | 28 |
| Generate and send a random password..... | 29 |
| Match input of password to open gate in front of parking..... | 29 |
| Chapter 5..... | 31 |
| 5.1 Software Analysis | 31 |
| Feasibility..... | 31 |
| Requirement..... | 32 |
| 5.2 Software Design..... | 32 |
| Platforms and Tools | 32 |
| Third party Libraries and APIs: | 35 |
| 5.3 User Interface and Design..... | 40 |
| Main Activity | 40 |
| Registration Activity..... | 41 |
| Login Activity..... | 42 |
| Home Activity..... | 43 |
| History Activity | 44 |
| My Spots Activity | 45 |
| Profile Activity..... | 46 |
| 5.4 Application's Features | 47 |
| 5.5 Client's View of the whole process | 48 |
| For First Time Users | 48 |
| Time out pop up | 51 |
| 5.6 Database Design..... | 52 |
| Hierarchical Structure of Database | 52 |
| 5.6 System Implementation | 57 |

| | |
|--------------------------------------|----|
| Our Manifest File | 57 |
| App Dependencies | 59 |
| Taking User's Current Location | 60 |
| Chapter 7 | 62 |
| 7.1 Limitations | 62 |
| 7.2 Future Work | 63 |
| 7.3 Conclusion | 65 |

Table of Figures

| | |
|---|----|
| Figure 1: System Flow Chart | 17 |
| Figure 2: Arduino Uno Front | 23 |
| Figure 3: Block Diagram of IR sensor | 24 |
| Figure 4: Output voltage vs. distance graph of IR sensor..... | 24 |
| Figure 5: Matrix styled hex keypad | 25 |
| Figure 6: RGB LED and Resistors..... | 25 |
| Figure 7: Main Activity | 40 |
| Figure 8: Registration Activity | 41 |
| Figure 9: Login Activity | 42 |
| Figure 10: Home and Navigation Activity..... | 43 |
| Figure 11: History Activity..... | 44 |
| Figure 12: My Spots Activity | 45 |
| Figure 13: Profile Activity | 46 |
| Figure 14: Time out pop up | 51 |
| Figure 15: Root DB..... | 53 |
| Figure 16: History DB | 54 |
| Figure 17: Parking Place DB | 55 |
| Figure 18: Passcode DB..... | 55 |
| Figure 19: Users DB | 56 |

Chapter 1

1.1 Introduction

Everyone who has ever been frustrated driving around urban areas in search of parking has wished for a solution that could quickly lead them to that elusive spot. In a recent research it has been found that a driver takes nearly 8 minutes to park his vehicle because he spend more time in searching the parking lot. This searching leads to 30 to 40% of traffic congestion. This concern attracted strategic investments from dedicated industry sectors to boost parking revenues through technology-enabled solutions. Parking industry is being revolutionized by new technologies that enable cities to reduce traffic congestion and carbon emission. The Internet of Things (IoT) permeates with the world of parking to streamline processes that deliver intelligent parking solutions, which extend and manage parking inventories. In this context, IoT uses embedded wireless sensor networks to connect physical parking space infrastructures with information and communication technologies, where cloud-based smart management services are provided. This interconnectivity shift is also driving socio-economical changes, where data unleashed from physical infrastructures is leading to productivity gains through new applications and new business models.

Currently, the common method of finding a parking space is manual where the driver usually finds a space in the street through luck and experience. This process takes time and effort and may lead to the worst case of failing to find any parking space if the driver is driving in a city with high vehicle density. The alternative is to find a predefined car park with high capacity. However, this is not an optimal solution because the car park could usually be far away from the user destination. In recent years, research has used vehicle-to-vehicle and vehicle-to-infrastructure interaction with the support of various wireless network technologies such as radio frequency identification (RFID), Zigbee, wireless mesh network, and the Internet. This study aimed to provide information about nearby parking spaces for the driver and to make a reservation minutes earlier using supported devices such as smartphones or tablet PCs. Furthermore, the services use the ID of each vehicle in booking a parking space. However, the current intelligent parking system does not provide an overall optimal solution in finding an available parking

space, does not solve the problem of load balancing, does not provide economic benefit, and does not plan for vehicle-refusal service.

To resolve the aforementioned problems and take advantage of the significant development in technology, the Internet-of-Things technology (IoT) has created a revolution in many fields in life as well as in smart-parking system (SPS) technology.

The present study proposes and develops an effective cloud-based SPS solution based on the Internet of Things. Our system constructs each car park as an IoT network, and the data that include the vehicle GPS location, distance between car parking areas and number of free slots in car park areas will be transferred to the cloud data center. We have used a Geolocation Service that calculates the cost of a parking request in terms of distance and time, and this cost is frequently updated and are accessible at any time by the vehicles in the network.

The SPS is based on several innovative technologies and can automatically monitor and manage car parks. Furthermore, in the proposed system, each car park can function independently as a traditional car park. This research also implements a system prototype with wireless access in an open-source physical computing platform based on Arduino using a smartphone that provides the communication and user interface for both the control system and the vehicles to verify the feasibility of the proposed system.

Modeling smart parking with a reservation policy provides many advantages for both the users (i.e. drivers) and parking areas controllers (managers). A parking reservation system allows users to book their parking spots prior to their arrival through the use of advanced communication technologies, such as Web interfaces and Push Notifications. A parking reservation system presents major advantages compared to the traditional Parking Guidance and Information (PGI) architectures. They continuously inform drivers regarding the parking space availability. Also provides optimum utilization of infrastructures for the parking owners In this paper, we propose a new intelligent parking reservation management architecture which is based on an optimization strategy that exploits interval scheduling principles. In interval scheduling, a list of parking requests is given as a set of requested time intervals (expressed as start and finish time) and the scheduler (parking management architecture) decides if it can accept a task and assign it to some resource or reject it.

1.2 Motivation

Navigating the contested streets of Dhaka on a daily basis is a tiring experience and drains all energy from one's body and is a topic that is very common among those that fall victim to it. However, our true motivation came from a discussion where we were talking about the future prospects of IoT and cloud computing in Bangladesh.

Technological advances have seen no implementation in parking management systems till date. With the vision of 'Digital Bangladesh' in mind and all the resources that will be available to us, we plan to create an affective smart parking system that significantly reduces traffic congestion.

1.3 Problem Statement

The parking management problem can be viewed from several angles. Limited number of parking lots, drivers not knowing where parking lots are, drivers not sure if parking lots have enough space and a tendency to park illegally on the roads.

1.4 Solution

To achieve this goal, each parking lot has to be equipped with a control system that enables monitoring of the number of free and occupied parking places and informing potential parking lot users about the parking lot status (open with/without free available parking spaces or closed) locally and in a wider area. Additionally, it is preferable that the systems contains driver navigation to a parking lot with free parking spaces in an urban area and driver navigation to a free parking space in a parking lot, tracking of parking lot occupancy during parking lot working hours for further analysis, parking service payment according to parking time duration, and security monitoring of the parking lot.

1.5 Thesis Outline

Chapter 1: Introduction, Motivation, Problem Statement and Solution of the proposed system.

Chapter 2: Literature Review and Background Study.

Chapter 3: System Overview – System Flow, Algorithm and System Architecture.

Chapter 4: Hardware Prototyping and Implementation.

Chapter 5: Software Analysis, Design and Implementation.

Chapter 6: Database Design.

Chapter 7: Limitations, Future Work and Conclusion.

Chapter 2

2.1 Literature Review

In some studies [1]–[3], the authors proposed a new algorithm for treatment planning in real-time parking. First, they used an algorithm to schedule the online problem of a parking system into an offline problem. Second, they set up a mathematical model describing the offline problem as a linear problem. Third, they designed an algorithm to solve this linear problem. Finally, they evaluated the proposed algorithm using experimental simulations of the system. The experimental results indicated timely and efficient performance. However, these papers do not mention the resource reservation mechanism (all parking requirements are derived immediately and are placed in the queue), the mechanism for assessing the resources system, the mechanism to guide vehicles to the parking space, the mechanism for handling situations when the request for service is denied and do not calculate the average waiting time and average total time that each vehicle spends on the system.

In another study Mainetti et al.[4], the authors propose an SPS based on the integration of UHF frequency, RFID and IEEE 802.15.4 Wireless Sensor Network technologies. This system can collect information about the state of occupancy of the car parks, and can direct drivers to the nearest vacant parking spot by using a software application. However, in this work, the authors have no mathematical equations for the system architecture and do not create a large-scale parking system. The results of this paper only implement the proposed architecture; they do not mention the performance of the parking system.

Hsu et al. [5] proposed an innovative system including the parking guidance service. A parking space can be reserved by a smartphone via Internet access. Upon entering the car park, the reserved parking space will be displayed on a small map using wireless transmission for vehicles under the dedicated short-range communication protocol DSRC. An inertial navigation system (INS) is implemented to guide the vehicle to the reserved space. The system will periodically update the status of the parking space in real time to help ensure system accuracy. System performance is measured through the accuracy of the inertial navigation systems run in an indoor environment, and the system implementation is evaluated by considering the accuracy of the

GPS. In this paper, the authors have not evaluated the performance of the parking services, they do not provide any mathematical model of the system, and do not consider the waiting time of each vehicle for service.

Other researchers have designed architecture for parking management in smart cities Barone et al.[6]. They proposed intelligent parking assistant (IPA) architecture aimed at overcoming current public parking management solutions. This architecture provides drivers with information about on-street parking stall availability and allow drivers to reserve the most convenient parking stall at their destination before their departure. They use RFID technology in this system. When a car parks or leaves the IPA parking spot, the RFID reader and the magnetic loop detect the action and send this information to the unit controller to update the information on the car park status. This study uses only some simple mathematical equations for the system architecture and does not create a large-scale parking system.

In other works, authors have designed and implemented an SPS whose bottom part is composed of ZigBee network which sent pressure information to PC through a coordinator and then update database Shiyao et al.[7] to solve the parking problem. A part of this system is implemented in the Zigbee network which sends urgent information to a PC through a coordinator and then updates the database. The application layer can quickly pass the parking information over the Internet, and use the advantages of a web service to gather all the scattered parking information for the convenience of those who want to find a parking space. This paper simply reports the design and implementation of an SPS and does not evaluate the system performance.

Bonde et al. [8] aimed to automate the car and the car parking. The paper discusses a project which presents a miniature model of an automated car parking system that can regulate and manage the number of cars that can be parked in a given area at any given time based on the availability of parking spaces. The automated parking method allows the parking and exiting of cars using sensing devices. Entry to or exit from the car park is commanded by an Android based application. The difference between the Bonde system and the other existing systems is that the authors were aiming to make the system as little human dependent as possible by automating the cars as well as the entire car park; on the other hand, most existing systems require human intervention (the car owner or other) to park the car.

To this extent a number of parking guidance systems aiming to minimize driver inconvenience have been reported in the literature over the last decade [11–13]. The aforementioned works describe architectures for real-time parking space availability which take into account information obtained from parking meters or sensors installed across the parking lot. However, the main drawback of such approaches is the cost of sensor deployment, especially in large parking areas.

To avoid the cost of using parking sensors, an alternative solution Lu et.al[14] performs parking reservation by exploiting the capabilities of vehicular ad hoc networks (VANETS). VANETs utilize wireless communication enabling cars to communicate with each other and with the roadside infrastructure. The main drawback of such technology is that it requires special equipment to be installed in cars and the roadside; it is anticipated that such deployment will take some time and therefore the aforementioned approach is not realistically implementable at present.

Another concept in the area of intelligent transportation systems is the development of Parking Reservation Systems (PRs) that aim to enable drivers to reserve a parking spot prior to their trip Mouskos et al.[15]. These schemes, instead of introducing dynamic message signs that continuously update the number of available parking spaces, propose parking reservation based on the use of an optimization strategy. This paper formulates parking reservation as a binary assignment integer linear program, which can be solved through the use of any linear programming software. Linear programming optimization is rather simple; it does not provide an efficient max-min optimization strategy-minimization of parking overlapping (satisfaction of user's requirements) and simultaneous maximization of parking infrastructure capability (satisfaction of parking owners).

Lambrinos and Dosis [9] described a new SPS architecture based on the Internet of Things technology. The architecture of this system consists of a Zigbee Wireless Sensor Network (WSN), an IoT middleware layer and a front-end layer as the final user interface that provides data reporting to the user. However, there are disadvantage as it does not use a suitable application protocol for the transfer of data from the WSN to the server, such as the constrained application protocol (CoAP), there is no mathematical model for the system operations, and there

is no system performance evaluation. In Geng and Cassandras [10], an approach is presented for reservation of parking slots based on the driver's costs. But the proposed system can not fuse the parking reservation requests and statuses with other useful data which is gathered by the cars and parking slots.

This open opportunity can lead to propose an Internet of Things solution to connect every parking slots to every cars efficiently by using added value services. As the best of our knowledge there is no proposed IoT-based scheme for traffic-correlated parking reservation service. Parking reservation scheme without considering the estimated time of car arrivals and reservation cannot lead to an efficient solution.

Chapter 3

3.1 System Flow

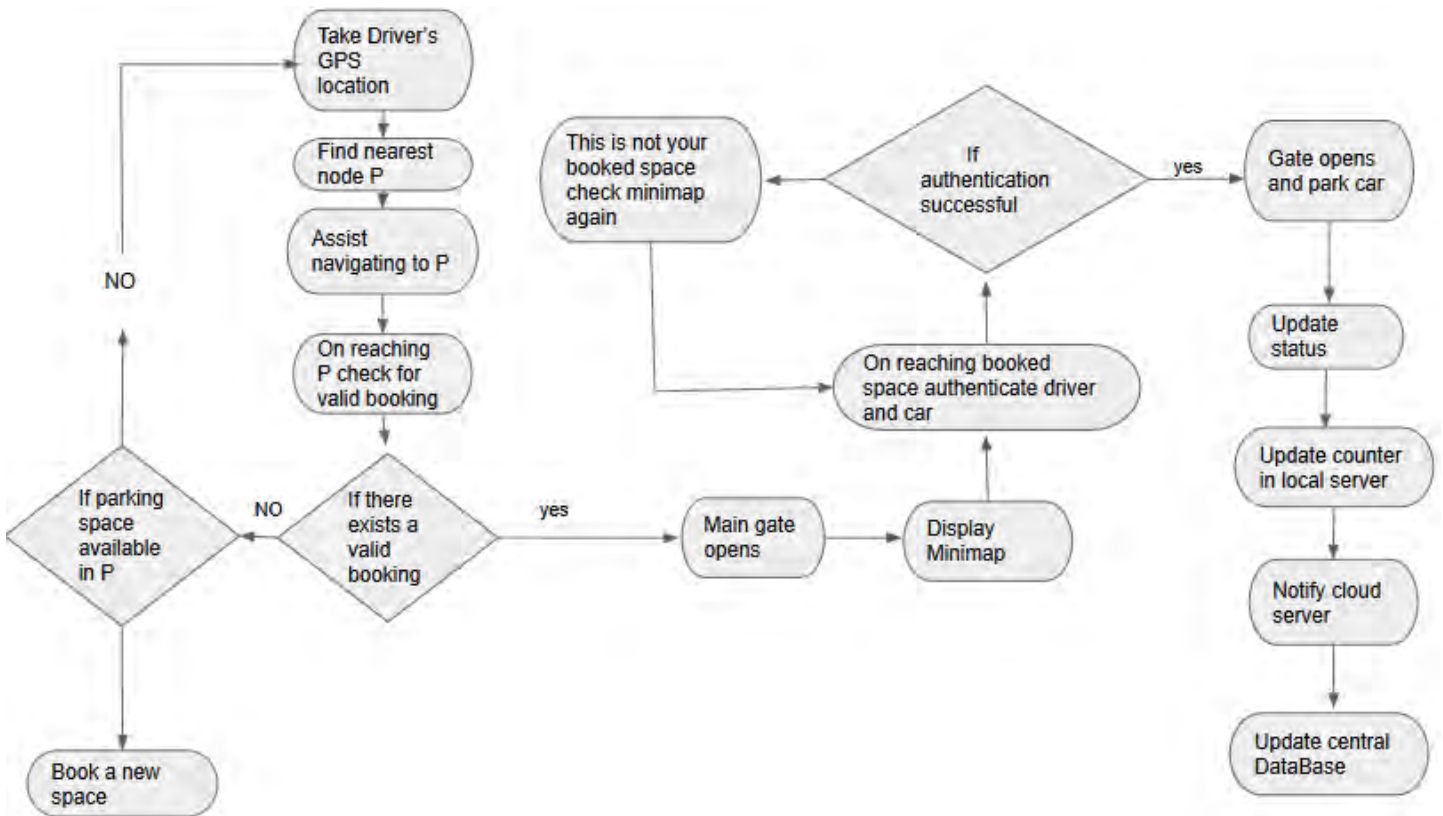


Figure 1: System Flow Chart

3.2 Algorithm

```
1. System Starts {
2. Int D , Time, pcount;
3. Pcount = 10;
4. Time = 9Mins;
5. ArrayList<LatLng> parking_locations;
6. parking_locations = new ArrayList();
7. parking_locations_n = new ArrayList();
8. //set existing parking locations on map.
9. parking_locations.add(new LatLng(latitude, longitude));
10. parking_locations_n.add(new LatLng(..... , .....));
11. // here -n” signifies the nth number of parking location
12. //display current location on the MapActivity on ButtonPress()
13. ShowCurrentLocation(){
14. //using inbuilt android location managers.
15. googleMap.SetLocationEnabled = true;
16. LocationManager locationManager = (LocationManager) getSystemService
    (Location_Service);
17. String Provider = locationManager.GetBestProvider(services, true);
18. Location location = locationManager.GetLastKnownLocation(Provider);
19. if (location not null){
20. //fetching latitude and longitude
21. double latitude = location.getLatitude();
22. double longitude = location.getLongitude();
23. //set marker
24. MarkerOptions options=new MarkerOptions().setTitle(“My Current
    Location”).setMarker(lat,long)
25. googleMap.addMarker(options);
26. return latitude, longitude;
27. }
28. Void buttonPress(){
29. ShowCurrentLocation(Lat, Long);
30. //sends HTTP requests through Google API to determine current location.
31. }
32. SearchForParkingLot(){
33. //using retrofit and google distance matrix API to match D within 100metres from current
    location from
    https://maps.googleapis.com/maps/api/distancematrix/outputFormat?parameters”
34. //retrieve green signal from the hardware.

35. If (D == minimumOfD(parking_locations1, parking_locations_n){
36. ArrayList parking_location = SetLocation(latitude, longitude)
```

```

37. //search for parking location within a specific radius e.g 100 metres and save coordinates
    in an arraylist called parking_location using inbuilt android methods.
38. if (pcount != 0){
39. //checks from hardware if the parking spots are available.
40. Toast : "nearest parking found"
41. }else{
42. Toast: "Valid parking spots not found. Searching for a 2nd best location"
43. restart();
44. //starts the process again if the desired parking location not found
45. }
46. }
47. void BookParking(){
48. if(pcount != 0){
49. //start booking process
50. showRoute(lat, long){
51. display ("Parking booked for 10 minutes");

52. time- -;
53. //time deducted for every seconds spent while reaching towards destination
54. pcount--;
55. //pcount is reduced by 1 for every parking spots booked or taken and this information is
    sent to the firebase and hence to the hardware.
56. D = CurrentLocation(lat, long) - Parking_Location(lat, long); // deducts current location
    from the destination and forms a route
57. }
58. }
59. D- - ;
60. //the D will decrease for every distance travelled towards the destination.
61. POPUP Message : This is your passcode "XXXX", please drive to your destination
    before time runs out;
62. //fetches keycode from the hardware with the help of cloud services and wireless
    modules.
63. if(D != 0 && Time == 0){
64. cancelRoute();
65. //user fails to reach destination in time, asks the user if he wants to find a spot again
66. }
67. When D = 0;
68. Toast: "you have reached your destination, enter the keycode to park the car";
69. If (application keycode == hardware generated keycode){
70. Gate.Open();
71. }else{
72. Gate.AccessDenied();
73. }
74. }
75. }
76. //enter the keycode again to open the gate during departure

```

```
77. DiscardKeycode(keycode){
78. keycode = null;
79. //cancels the keycode and it cannot be used again.
80. }
81. //automatically closes gate after a few seconds
82. exit();
```

3.1 System Architecture

Cloud-Based Server

This is a Web entity that stores the resource information provided by local units located centrally in the system. This server basically receives the request from clients placed through the application for free parking spaces. After receiving the request it checks for available nearest parking lots from the client's current location in the central database which is stored in it without letting the client directly access the local servers of each car park. On finding the nearest car park with the least cost in terms of both distance and estimated time needed to reach, it displays the client the information of the car park via the application. The cloud server also performs necessary computation on the data that has been routed to it from the local servers. After completing each calculation the server periodically updates the central database with data like total percentage of free space in each parking lot and also the total percentage of the overall parking system.

Local Unit

This unit is located in each car park and stores the information of each parking space. The server basically maintains a counter which keep track of the current occupancy status of the car park. The local zone contains a router to transmit sensor data. Each router in a zone is connected to pressure sensors to detect the presence or absence of a vehicle, to IR sensors to detect the occupancy of each parking spots and to a number pad attached to the entrance to detect the real user who has booked the specified spot. The information from the local server is then uploaded into a central cloud based server, which makes decisions to provide the suitable parking zone for a vehicle based on its vicinity. The local unit also contains a control unit, which is an Arduino module that connects with the cloud server through an Internet connection to transfer data from

the local car park to the cloud server database. The Arduino module will control the opening of the door for the vehicle to enter in the booked parking space.

Software Client

This is an application software system running on Android operating system. Users will install it on their smartphones and use it to find, reserve and navigate to the nearest parking spaces available. The users access the system via 3G/4G mobile connections. To make the system work, we have to locate the address on the exact position of the map which requires address geocoding a process of finding associated geographic coordinates (often expressed as latitude and longitude) of a location by the text of the address. For this we have used Class Client Geocoder in Google Maps API. A Client Geocoder object communicates with Google map servers to obtain geocodes for user-specified addresses. In addition, a Geocoder object maintains its own cache of addresses, which allows repeated queries to be answered without a round trip to the server.

After getting the vehicle location we need to display a route on the map. For this we have used Google Maps Direction API which retrieves the duration and distance values for multiple destinations and transport modes. In many systems, the distance between a customer pair is simplified to Euclidean distance. However, in real situations, the distance should be calculated based on the shortest road path between two locations. In Google Maps API for android, we can get the real distance from the distance property of the class Directions. A Direction object communicates with Google servers to obtain directions between two or more waypoints.

Chapter 4

4.1 Hardware Components

An instrumental part of our projected was the designing and implementation of a backend hardware system. This system will undertake the task of:

- Checking to see which parking spots are open and booking them
- Identifying vehicles
- Generating security codes to be sent to user for verification
- Mapping parking spots against vehicles
- Mapping verification code against user and parking spot
- Updating database

The entire system of a parking lot was scaled down to accommodate real-time simulations.

Arduino Uno vs. Raspberry Pi:

An Arduino is a microcontroller motherboard. A microcontroller is a simple computer that can run one program at a time, over and over again.

A Raspberry Pi is a general-purpose computer, usually with a Linux operating system, and the ability to run multiple programs.

We found the Arduino to contain sufficient memory and processing power to handle the codes and sensors used in the scaled down parking model.



Figure 2: Arduino Uno Front

Sharp gp2y0a21yk_e Infrared Reader and sensor:

The GP2Y0A21YK0F is a distance measuring sensor unit, composed of an integrated combination of PSD (position sensitive detector), IRED (infrared emitting diode) and signal processing circuit.

The sensor gives an analogue voltage reading corresponding to the detected distance.

Features

1. Distance measuring range: 10 to 80 cm
2. Analog output type
3. Package size: 29.5×13×13.5 mm
4. Consumption current: Typ. 30 mA
5. Supply voltage: 4.5 to 5.5 V

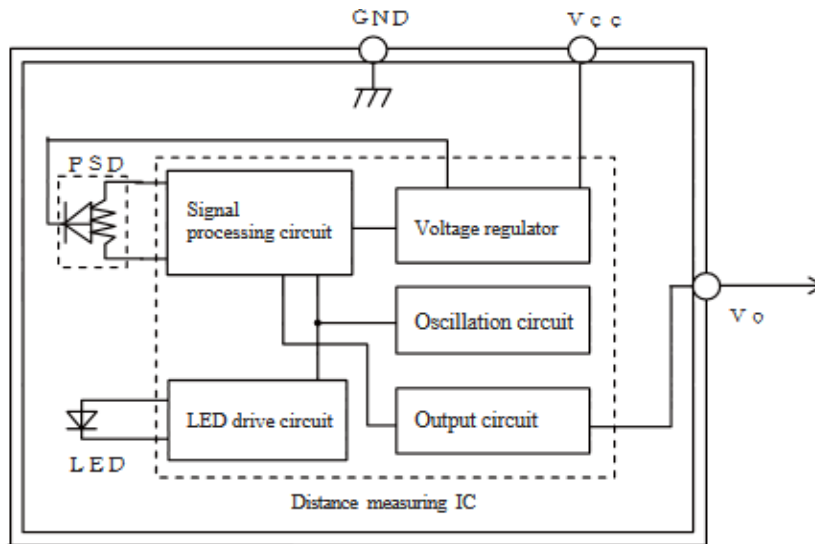


Figure 3: Block Diagram of IR sensor

The analogue readings taken from the sensor was converted to a distance using a simple mapping function in Arduino.

Example of distance measuring characteristics(output)

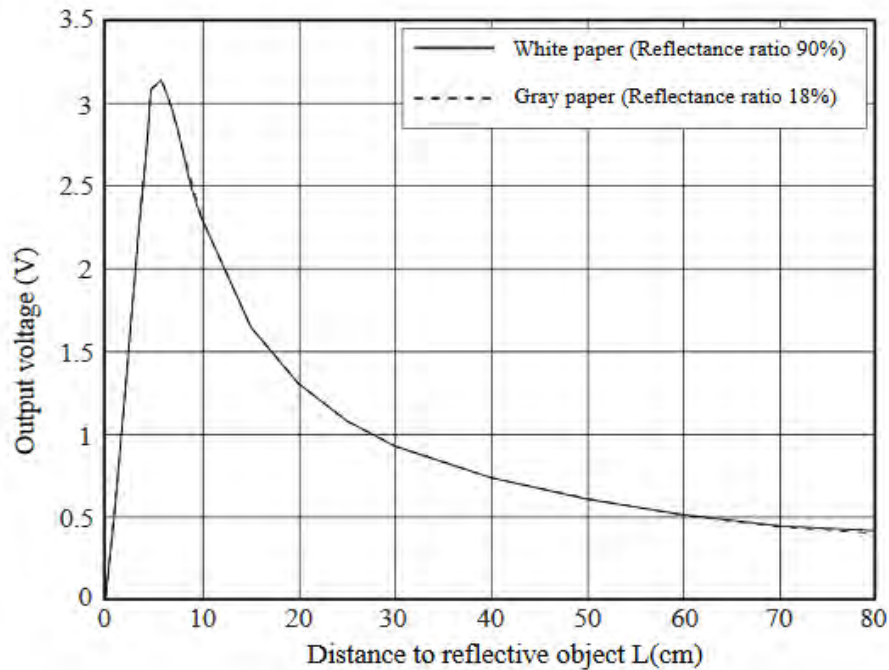


Figure 4: Output voltage vs. distance graph of IR sensor

Keypad

We used a matrix styled hex keypad and installed the **Keypad** [1] library. This took an input and was used for user verification before entering a booked parking spot.

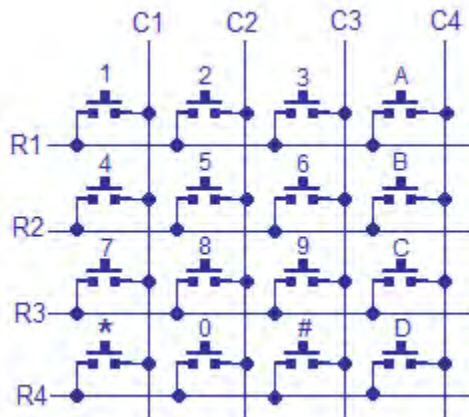


Figure 5: Matrix styled hex keypad

LED and Resistors

A RGB LED was used instead of an actual gate with the logic being that the LED will go red as soon as a parking is booked, indicating a closed gate. Once the user reaches the parking spot and enters the verification code given to him/her, the LED turns green, indicating an open gate. 1k resistors were used to control the flow of current.

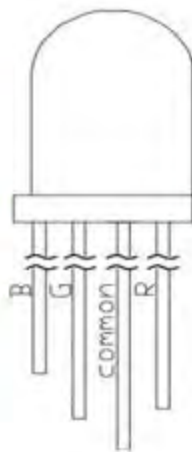
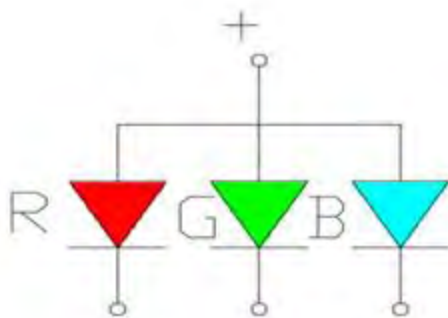


Figure 6: RGB LED and Resistors

4.2 Hardware Prototype

The hardware prototype provides back-end hardware support for users to book and access a parking spot. The algorithm was fine-tuned to reduce run time to an absolute minimum, making the user experience as smooth and fast as possible. To book the parking spot, the system conducts a 5 step process as follows:

- ➔ Read Data from sensors
- ➔ Check condition of all parking spots
- ➔ Book a parking spot
- ➔ Generate a random password
- ➔ Match input of password to open gate in front of parking

Reading Data from Sensors

For the prototype, three Sharp GP2Y0A21YK0F infrared sensors were connected to the parking spots and used to check if there are any cars in the parking. The sensor only gives an analogue voltage reading thus we used a mapping function to map voltage 'V' to distance 'cm'.

The variables val1, val2 and val3 store the analogue readings and dist1/2/3 stores the mapped distance variable.

```
val1 = analogRead(V1);  
val2 = analogRead(V2);  
val3 = analogRead(V3);  
  
dist1 = map(val1, 50, 400, 80, 10);  
dist2 = map(val2, 50, 400, 80, 10);  
dist3 = map(val3, 50, 400, 80, 10);
```

Check condition of all parking spots

This part of the system checks to see which parkings are open and which are used, prints the condition of the parking spaces and updates an integer array `parkingSlotsEmpty[]`. The array, in combination with another array called `parkingSlotsBooked[]` enables us to see which parking spots are available.

```
void pCond()
{
  if (dist1 < 10)
  {
    Serial.println("Car in parking slot p1"); //increase parking count
    parkingSlotsEmpty[0] = 1;
  }
  else
  {
    Serial.println("No car in parking slot p1");//decrease parking count
    parkingSlotsEmpty [0] = 0;
    if (parkingSlotsBooked[0] = 0)
      setColor(0, 255, 0);
    delay (1500);
  }
  if (dist2 < 10)
  {
    Serial.println("Car in parking slot p2"); //increase parking count
    parkingSlotsEmpty [1] = 1;
  }
  else
  {
    Serial.println("No car in parking slot p2");//decrease parking count
    parkingSlotsEmpty [1] = 0;
    if (parkingSlotsBooked [1] = 0)
      setColor(0, 255, 0);
    delay (1500);
  }
  if (dist3 < 10)
  {
    Serial.println("Car in parking slot p3"); //increase parking count
    parkingSlotsEmpty [2] = 1;
  }
  else
  {
    Serial.println("No car in parking slot p3");//decrease parking count
    parkingSlotsEmpty [2] = 0;
    if (parkingSlotsBooked [2] = 0)
      setColor(0, 255, 0);
    delay (1500);
  }
}
```

Book a Parking Spot

This function checks the two arrays that deal with booked slots and empty slots. If the function finds a spot that is open, it will call upon another function `book_parkingSpotName()`; [e.g. `bookP1()`]. The `book_parkingSpotName ()` function prints the name of the booked spot and updates `parkingSlotsBooked []` and sets the LED to red. A red LED represents a closed gate.

```
String bookParking()
{
  if ((parkingSlotsBooked [0] && parkingSlotsEmpty [0]) == 0)
  {
    bookP1();
    return "p1";
  }
  else if ((parkingSlotsBooked [1] && parkingSlotsEmpty [1]) == 0)
  {
    bookP2();
    return "p2";
  }
  else if ((parkingSlotsBooked [2] && parkingSlotsEmpty [2]) == 0)
  {
    bookP3();
    return "p3";
  }
  else
  {
    Serial.println("Parking is full!");
    return " ";
  }
}

void bookP1()
{
  parkingSlotsBooked [0] = 1;
  Serial.println("Parking slot p1 is booked");
  setColor(255, 0, 0);
  delay (1500);
}
void bookP2()
{
  parkingSlotsBooked [1] = 1;
  Serial.println("Parking slot p2 is booked");
  setColor(255, 0, 0);
  delay (1500);
}
void bookP3()
{
  parkingSlotsBooked [2] = 1;
  Serial.println("Parking slot p3 is booked");
  setColor(255, 0, 0);
  delay (1500);
}
```

```

void setColor(int red, int green, int blue)
{
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}

```

Generate and send a random password

This function makes a random 4 digit code using the 'True Random' [2] library and sends it to the central database.

```

int pwMake ()
{
  Pw = TrueRandom.random(1000, 9999);
  delay(500);
  return Pw;
}

```

Match input of password to open gate in front of parking

The final part of the system takes an input from the user using the physical keyboard in the parking spot. If the input matches the code sent to user then the validCode () function is called which sets led to green, resembling an open gate.

```

void pwInput()
{
  //c=0;
  while (c < 4)
  {
    char key = keypad.getKey();
    if (key != NO_KEY)
    {
      int a = int(key) - 48;
      Serial.print("* ");
      Chk[c] = a;
      c++;
    }
  }
  pwCheck();
  delay(3000);
}

```

```

void pwCheck()
{
    while (l < 4)
    {
        if (Chk[l] != PwArr[l])
        {
            invalidCode();
        }
        else {
            if (l = 3)
            {
                validCode();
            }
            l++;
        }
    }
}

```

```

void invalidCode()
{
    delay(500);
    Serial.println("Invalid code, Please enter correct password");
    c = 0;
    pwInput();
}

```

```

void validCode()
{
    delay(500);
    Serial.println("Code accepted");
    //Enter code for RGB HERE
    c = 4;
    setColor(0, 255, 0);
    delay(1000);
}

```

Chapter 5

5.1 Software Analysis

Feasibility

Technological feasibility: The app we have developed is very easy to use. It was built in a way that it meets the user's comfort zone. The current users of mobile phone application across the globe whether using a navigation app or rent a car app use an interface quite similar to ours thus making it possible to reach every user type. There is enough software components used in the backend to provide a robust service to the user.

Economic feasibility: As a mobile assistant application we had to make zero investment for now thus making it economic. And in the long run with more features added like premium database handling, still a better economic feasibility can be achieved. If we look carefully, we will notice that for maximum number of users, even a purchased database or cloud system can be very cost effective.

Organizational Feasibility: Car parking problem is quite major across the world and to solve this, our application can help users and reduce their trouble. If we imagine a short scenario with a car parking app in a remote city of Bangladesh, say for Dhaka, we would definitely see a major change in the traffic condition of the streets as we are very likely to park our cars anywhere we find space. This ensures our application's organizational feasibility.

Requirement

Functional Requirements

1. We are building an android mobile application. User will need a smartphone with android IOS.
2. As an IOT based mobile app, internet connection is a must.
3. The app needs user authorization.
4. Application should not contain a loophole and not let the users misuse it.
5. The mobile application requires a real-time database and cloud system. If any of these goes down, the application will collapse. Thus, a live support is necessary.

Nonfunctional requirements

1. The application should be user friendly.
2. The application when crashed will generate negative perception about the application. Therefore, every time the application crashes, a crash report processing is necessary.
3. Huge number of users may login at the same time. So, the application should be able to handle high traffic.
4. The application's data fetching and retrieval system should be faster.
5. Premium database plan should be undertaken.

5.2 Software Design

Platforms and Tools

Android

Android is a software stack for mobile devices that includes an operating system based on the Linux Kernel, middleware and key applications. At a larger scale, Android is an Ecosystem built by Google & its affiliates, which enables Android developers to build, deploy and monetize applications for a range of touchscreen Android devices. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android

Wear for wristwatches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.

JAVA

Java is the most widely used programming language by the android developers. It was originally developed by James Gosling at Sun Microsystems on 1995 which is now acquired by Oracle Corporation. One of the main reasons of using Java for android development is, the language's platform independent quality. Android runs on many different hardware platforms. So using any platform dependent language will lead you to compile and optimize your native code for each of these different platforms to see any real benefits. Besides Java has huge open source support, with many libraries and tools available to make developers life easier. Java also protects developers from many of the problems inherent in native code, like memory leaks, bad pointer usage, etc. Java allows creating sandbox applications and creating a better security model so that one bad App can't take down your entire OS. Lastly there are a large number of developers already proficient in Java which is one of the vital reasons of using Java in developing android applications.

Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built based on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development. We have used the current android studio version v2.3.0 for this project because it includes certain useful updates like- the two new Constraint Layout Features, the separate button to push changes with Instant Run, can now convert PNG, BMP, JPG, and static GIF files to WebP format. WebP file format which provides glossy compression (like JPEG) as well as transparency (like PNG) but can provide better compression than either JPEG or PNG.

Android SDK

The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.5.8 or later, and Windows 7 or later. For this project we have used sdk version v25.0.1 .

Google Developer Console

The Google Developer Console enables developers to deploy their applications to the Google Play Store and track usage and other relevant analytics.

Google Firebase

Firebase is a mobile and web application development platform developed by Firebase, Inc. in 2011, then acquired by Google in 2014. Firebase has grown inside Google and expanded their services to become a unified platform for mobile developers. Firebase now integrates with various other Google services to offer broader products and scale for developers. We have used a handful of features provided by this platform in our project which made our work for building the server side of our application a lot more easier. The Firebase services that we use for building the server side of our mobile application are briefly discussed below:

- **Authentication:** We have used the authentication module of Firebase to manage our users in a simple and secure way. Firebase Auth offers multiple methods to authenticate, including email/password, third-party providers like Google or Facebook, or using your existing account system directly. We used user's email and password for authentication and registration. We have built our own interface, instead of taking advantage of their open source, fully customizable UI in order to make our UI design consistent and indifferent to other applications.
- **Real-time Database:** Firebase provides a real-time database service for **storing** and syncing data between users and devices in real-time using a cloud-hosted, noSQL database. Updated data syncs across connected devices in milliseconds,

and data remains available if the app goes offline, providing a great user experience regardless of network connectivity. For the above reasons we have selected Firebase Database as our main database for storing all our user information, parking spot locations and status, user subscription history and passcodes.

- **Cloud Storage:** Google also provides a cloud storage service with their Firebase platform to Store and share images, audio, video, or other user-generated content easily with powerful, simple, and cost-effective object storage built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. For these reasons we took the advantage of this cloud storage for storing our user's profile pictures, which will help us to identify each user more accurately.
- **Google Analytics:** Another feature that comes with Firebase is the Analytics module that analyzes user attributions and behavior in a single dashboard to make informed decisions on the product roadmap. We can Gain real-time insights from reports, or export your raw event data to Google BigQuery for custom analysis. We got a lot of insights such as, the number of our monthly active users, percentages of user engagements on different activities of the application, region wise usage of our application and many more.

Third party Libraries and APIs:

Google Maps Android API

With the Google Maps Android API, one can add maps based on Google Maps data to their application. The API automatically handles access to Google Maps servers, data downloading, map display, and response to map gestures. One can also use API calls to add markers, polygons, and overlays to a basic map, and to change the user's view of a particular map area. These objects provide additional information for map locations, and allow user interaction with the map. The API allows to add these graphics to a map:

- Icons anchored to specific positions on the map (Markers).
- Sets of line segments (Polylines).
- Enclosed segments (Polygons).
- Bitmap graphics anchored to specific positions on the map (Ground Overlays).
- Sets of images which are displayed on top of the base map tiles (Tile Overlays).

We have used the API for displaying user's current location as well as the location of the designated parking spot for the user.

Google Maps Direction API

The Google Maps Directions API is a service that calculates directions between locations. One can search for directions for several modes of transportation, including transit, driving, walking, or cycling. The API returns multi-part directions using a series of waypoints. Specifies origins, destinations, and waypoints as text strings (e.g. "Chicago, IL" or "Darwin, NT, Australia"), or as latitude/longitude coordinates, or as place IDs. The API returns the most efficient routes when calculating directions. Travel time is the primary factor optimized, but the API may also take into account other factors such as distance, number of turns and many more when deciding which route is the most efficient. Thus by considering all the above mentioned features of this API we have integrated it in our application in order to find the nearest parking spot for the user from his current location.

Android Design Support Library

Material Design is a new design language that gives design guidelines for Android apps, was introduced with the release of Android 5.0 Lollipop. With it came new UI components such as the `_Floating Action Button`, `_Typography` which adds further guidance on style and line height for dense and tall languages, `_Cards` which includes more specs for laying out actions and content etc. Implementing these new components while ensuring backward compatibility was typically a tedious process. Third party libraries would usually be needed to streamline the process. Such a third party library is the Android Design Support Library. Since we have developed our whole app with material design we took the advantage of the Design Support library to streamline our design process. We have used version 25.3.1 of the android support design library instead of the latest version 26.0.1 due to compatibility issues with our other libraries.

Retrofit

Retrofit is a REST Client for Android and Java by Square. It makes it relatively easy to retrieve and upload JSON (or other structured data) via a REST based web service. In Retrofit we configure which converter is used for the data serialization. Typically for JSON you use GSON, but you can add custom converters to process XML or other protocols. Retrofit uses the OkHttp library for HTTP requests. To work with Retrofit you need basically three classes.

- Model class which is used to map the JSON data to
- Interfaces which defines the possible HTTP operations
- Retrofit.Builder class - Instance which uses the interface and the Builder API which allows defining the URL endpoint for the HTTP operation.

For this project we have used retrofit 2 version 2.1.0 .

Retrofit Converter

Retrofit can be configured to use a specific converter. This converter handles the data (de)serialization. Several converters are already available for various serialization formats.

- To convert to and from JSON:
 - Gson: com.squareup.retrofit:converter-gson
 - Jackson: com.squareup.retrofit:converter-jackson
 - Moshi: com.squareup.retrofit:converter-moshi
- To convert to and from Protocol Buffers:
 - Protobuf: com.squareup.retrofit:converter-protobuf
 - Wire: com.squareup.retrofit:converter-wire
- To convert to and from XML:
 - Simple XML: com.squareup.retrofit:converter-simplexml

Here in this project we have implemented a retrofit2 converter known as Gson Converter which uses Gson for serialization to and from JSON because whenever we receive or send any data to our firebase database it goes in JSON format. Our gson version is v2.1.0 .

Glide

Glide is a fast and efficient open source media management and image loading framework for Android that wraps media decoding, memory and disk caching, and resource pooling into a simple and easy to use interface. Glide supports fetching, decoding, and displaying video stills, images, and animated GIFs. Glide includes a flexible API that allows developers to plug in to almost any network stack. By default Glide uses a custom HttpURLConnection based stack, but also includes utility libraries plug in to Google's Volley project or Square's OkHttp library instead. Glide's primary focus is on making scrolling any kind of a list of images as smooth and fast as possible, but Glide is also effective for almost any case where you need to fetch, resize, and display a remote image. For developing our application we have used glide version v3.7.0.

EZ PhotoPicker Library

If some want to pick up a photo from the gallery and camera, store it somewhere then do something, this library will be the best choice for them. It will handle all the storing, scaling, rotating, threading, loading dialog. Easy to start a photo intent, easy to get the result, won't need to code a lot as what one used to do. It also help to handle about real-time permission without any lines of code. In our app a user can put his/her photo during registration. The photo will be stored in firebase database under user profile. For accomplishing the above mentioned task we have used EZ PhotoPicker library version v1.0.6 .

greenDAO

greenDAO is an open source Android ORM making development for SQLite databases fun again. It relieves developers from dealing with low-level database requirements while saving development time. SQLite is an awesome embedded relational database. Still, writing SQL and parsing query results are quite tedious and time-consuming tasks. greenDAO frees you from these by mapping Java objects to database tables (called ORM, "object/relational mapping"). This way someone can store, update, delete, and query for Java objects using a simple object oriented API. greenDAO's unique set of features:

- **Maximum performance** (probably the fastest ORM for Android); our benchmarks are open sourced too
- **Easy to use** powerful APIs covering relations and joins
- **Minimal** memory consumption
- **Small** library size (<100KB) to keep your build times low and to avoid the 65k method limit
- **Database encryption:** greenDAO supports SQLCipher to keep your user's data safe
- **Strong community:** More than 5.000 GitHub stars show there is a strong and active community.

The version of greenDAO used in this project is v3.1.1

5.3 User Interface and Design

Main Activity

This is the main landing screen of our application ‘Find My Parking’. From here a user can navigate both to the login and registration screens.

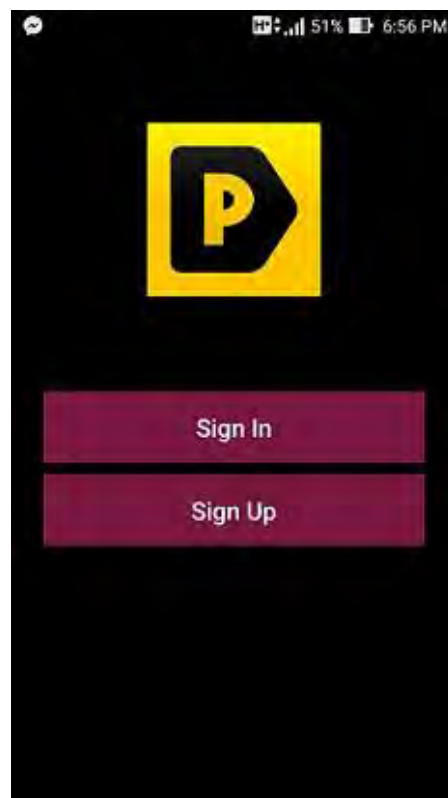
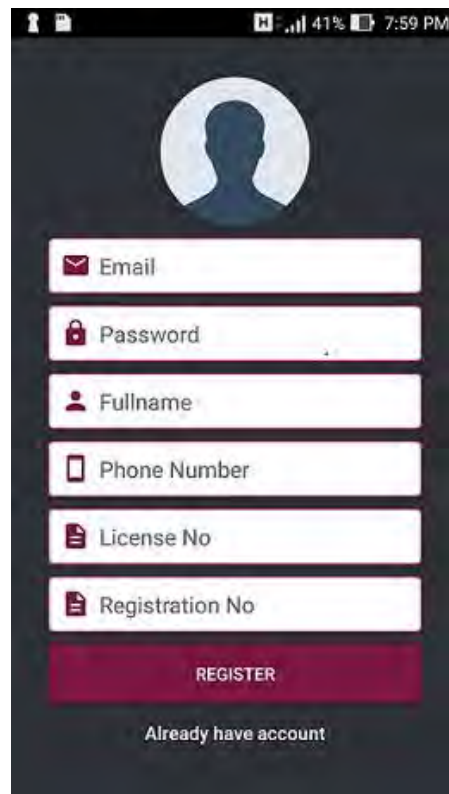


Figure 7: Main Activity

Registration Activity

This activity is used for completing the registration process. Here the user needs to provide their email, password, full name, phone number, driving license number, car registration number and their photo in order to successfully complete their registration.

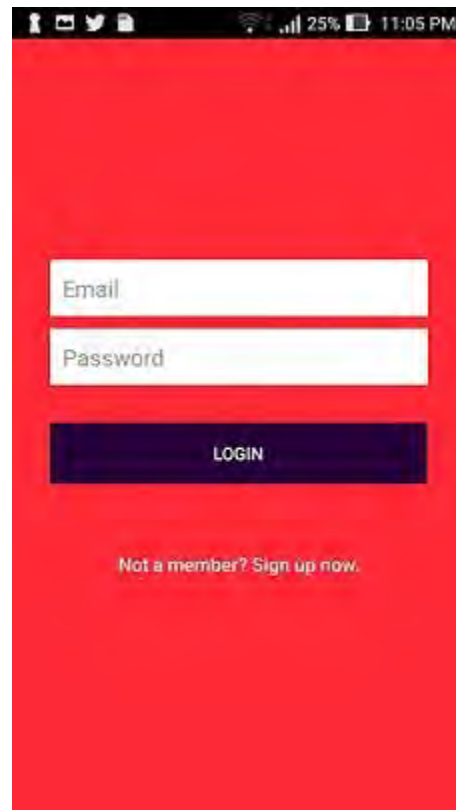


The image shows a mobile application registration screen. At the top, there is a status bar with icons for signal, battery (41%), and time (7:59 PM). Below the status bar is a dark header area containing a circular profile picture placeholder. The main content area consists of several white input fields stacked vertically, each with a small icon on the left: 'Email' (envelope icon), 'Password' (lock icon), 'Fullname' (person icon), 'Phone Number' (phone icon), 'License No' (document icon), and 'Registration No' (document icon). Below these fields is a prominent red button labeled 'REGISTER'. At the bottom of the screen, there is a link that says 'Already have account'.

Figure 8: Registration Activity

Login Activity

Users can log into their already registered accounts using this screen. Here insertion in the two input fields are mandatory for a successful login.



The image shows a mobile application login screen. At the top, there is a black status bar with icons for signal strength, Wi-Fi, and battery (25%), and the time 11:05 PM. The main background is a solid red color. In the center, there are two white rectangular input fields stacked vertically. The top field is labeled 'Email' and the bottom field is labeled 'Password'. Below these fields is a dark blue rectangular button with the word 'LOGIN' in white capital letters. At the bottom of the screen, there is a link that says 'Not a member? Sign up now.' in white text.

Figure 9: Login Activity

Home Activity

The main activity of our application is this activity. This is the screen form where a user can make a parking request, can check their previous subscriptions, their frequently used parkings, user profile and can logout of the application using the navigation drawer located at the top left corner. Route to the nearest parking space and travel time is also shown on this screen. User can also see the destination address here.

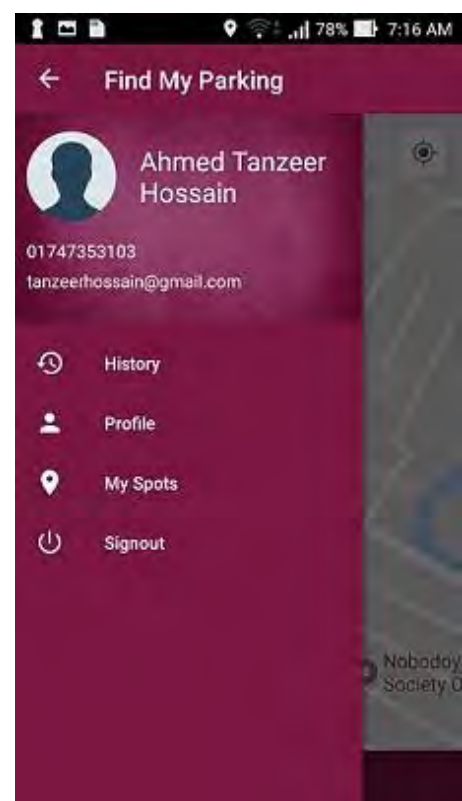


Figure 10: Home and Navigation Activity

History Activity

A user can reach this screen via the navigation drawer on home screen and can see his/her subscription history.

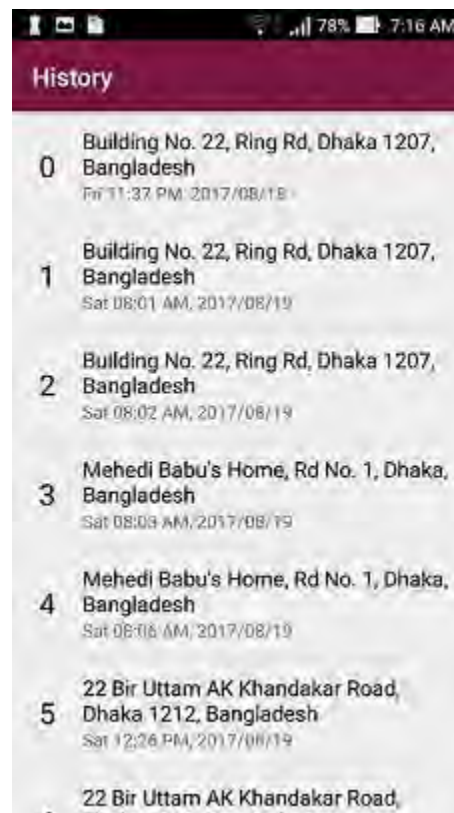


Figure 11: History Activity

My Spots Activity

This is where a user can find his/her frequently used parking space locations with a number associated to them indicating how many times each of these parkings has been assigned to the user.



Figure 12: My Spots Activity

Profile Activity

User's profile can be viewed from this screen. A user can navigate to this screen by using the navigation drawer.

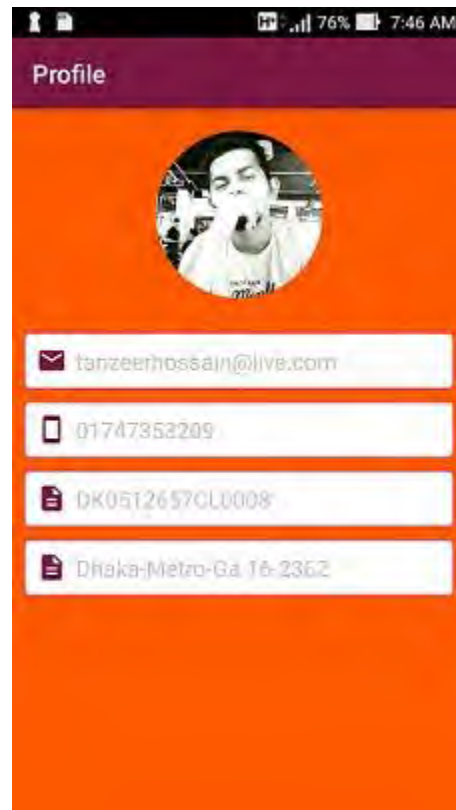


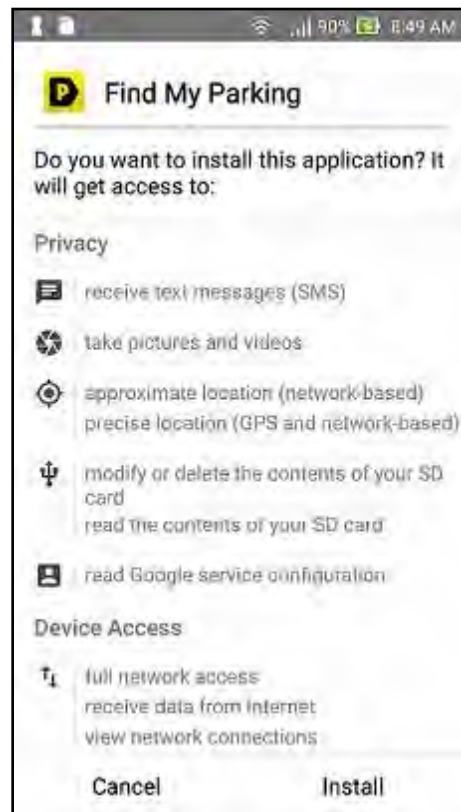
Figure 13: Profile Activity

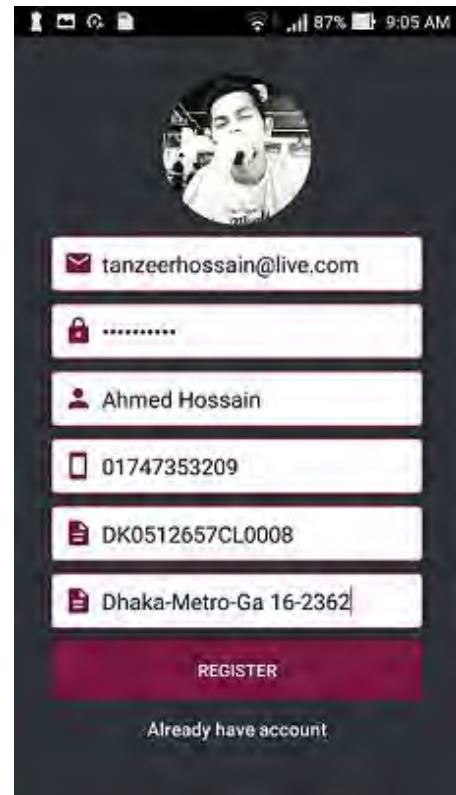
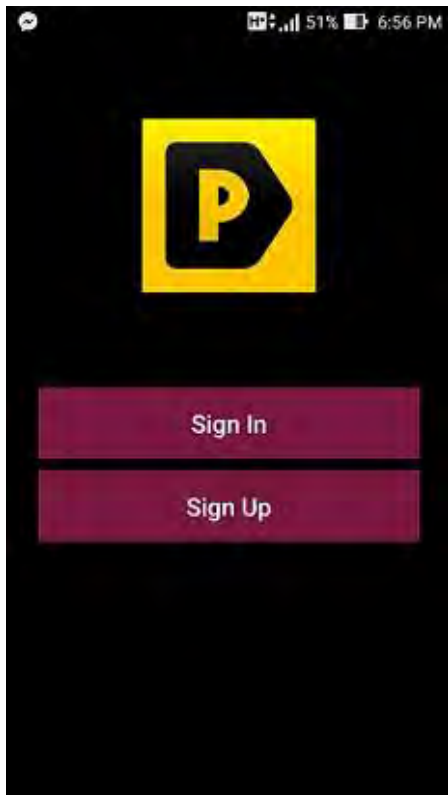
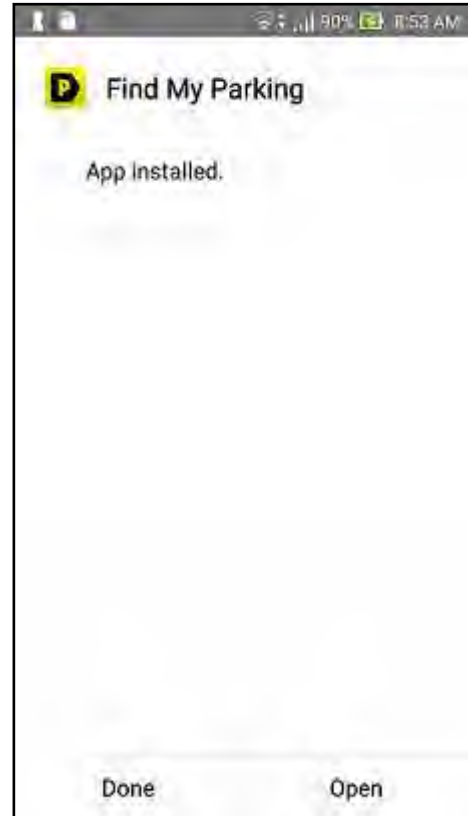
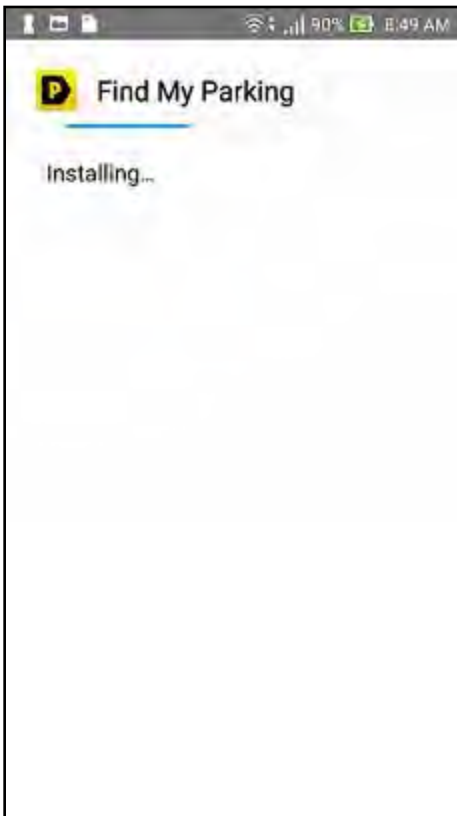
5.4 Application's Features

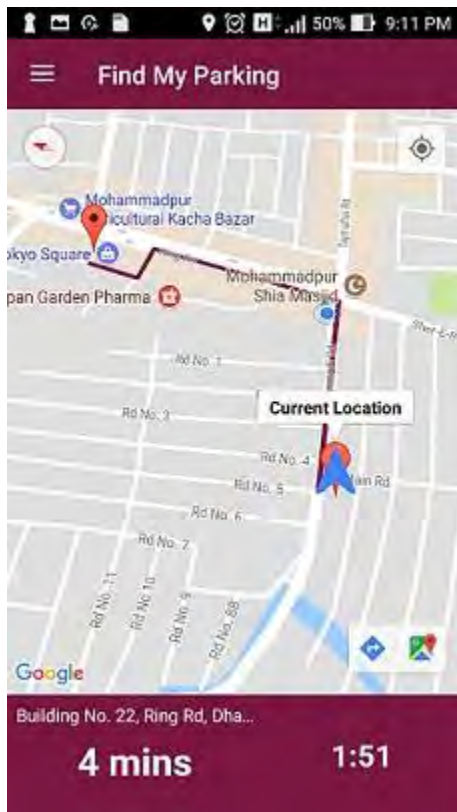
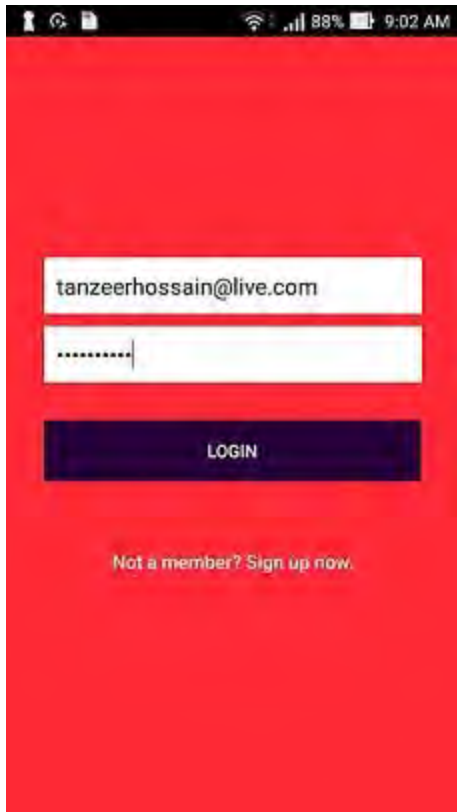
- Find the closest parking space from the user's current location in terms of distance and travel time.
- Guide vehicles to the nearby parking space through the shortest possible path considering travel time as the primary factor and further optimized by the number of turns in the route, real time traffic and some other parameters.
- Reserve a parking spot for the user and display the time to reach that spot.
- On exceeding the time limit let the user know that the booking has been cancelled and ask if he wants to make another request.
- Forwards the user to another car park if he makes another request.
- Store user subscription history.
- Store frequently used parking spaces under the tab called "My Spots".

5.5 Client's View of the whole process

For First Time Users







Time out pop up

An alert box pops up when the user fails to reach the designated parking on time. The system then asks if he/she wants to make another request since the previous one has been terminated.

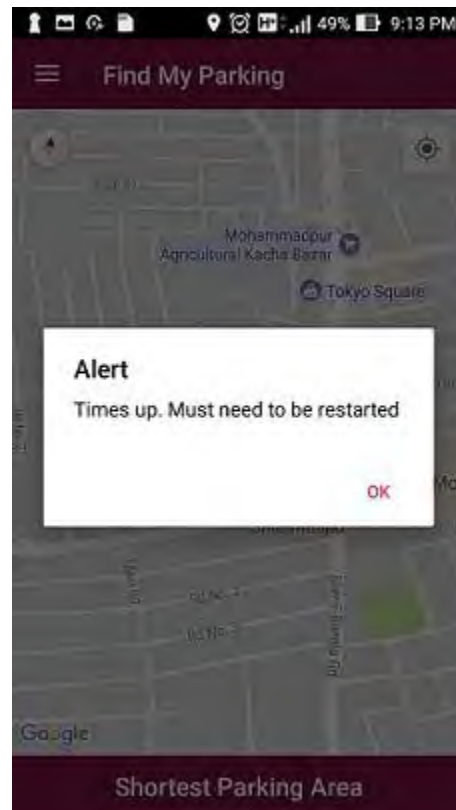


Figure 14: Time out pop up

5.6 Database Design

As stated above we have used Google's Firebase Database as our main database for the application. We have used Firebase because:

- **Real-time:** Instead of typical HTTP requests, the Firebase Realtime Database uses data synchronization—every time data changes, any connected device receives that update within milliseconds. Provide collaborative and immersive experiences without thinking about networking code.
- **Offline:** Firebase apps remain responsive even when offline because the Firebase Real-time Database SDK persists your data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronizing it with the current server state.
- **Accessible from Client Devices:** The Firebase Real-time Database can be accessed directly from a mobile device or web browser; there's no need for an application server. Security and data validation are available through the Firebase Real-time Database Security Rules, expression-based rules that are executed when data is read or written.

Hierarchical Structure of Database

As stated above we have used Google's Firebase Database as our main database for the application. We have used Firebase because:

- **Real-time:** Instead of typical HTTP requests, the Firebase Realtime Database uses data synchronization—every time data changes, any connected device receives that update within milliseconds. Provide collaborative and immersive experiences without thinking about networking code.
- **Offline:** Firebase apps remain responsive even when offline because the Firebase Real-time Database SDK persists your data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronizing it with the current server state.

- **Accessible from Client Devices:** The Firebase Realtime Database can be accessed directly from a mobile device or web browser; there's no need for an application server. Security and data validation are available through the Firebase Realtime Database Security Rules, expression-based rules that are executed when data is read or written.

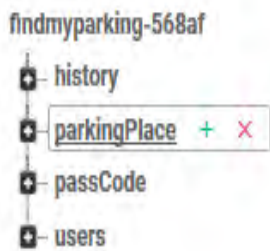


Figure 15: Root DB

findmyparking-568af

```
history
├── 1502025095058
│   ├── date: "Sun 07:11 PM, 2017/08/06"
│   ├── destination: "23.7805906,90.4092380"
│   ├── frequency: 1
│   ├── id: 1
│   ├── name: "13 Bir Uttam AK Khandakar Rd, Dhaka 1212, Bangl..."
│   ├── origin: "23.7744238,90.4268252"
│   ├── timeInLong: 1502025095058
│   └── uid: "eDfGqwTG2vgJetYfc4JmjibnVh73"
├── 1502025123557
│   ├── date: "Sun 07:12 PM, 2017/08/06"
│   ├── destination: "23.7805906,90.4092380"
│   ├── frequency: 2
│   ├── id: 2
│   ├── name: "13 Bir Uttam AK Khandakar Rd, Dhaka 1212, Bangl..."
│   ├── origin: "23.7744238,90.4268252"
│   ├── timeInLong: 1502025123557
│   └── uid: "eDfGqwTG2vgJetYfc4JmjibnVh73"
├── 1502025266357
├── 1502027324854
├── 1502028099255
├── 1502029123263
├── 1502029215033
├── 1502029299500
└── 1502029334373
```

Figure 16: History DB

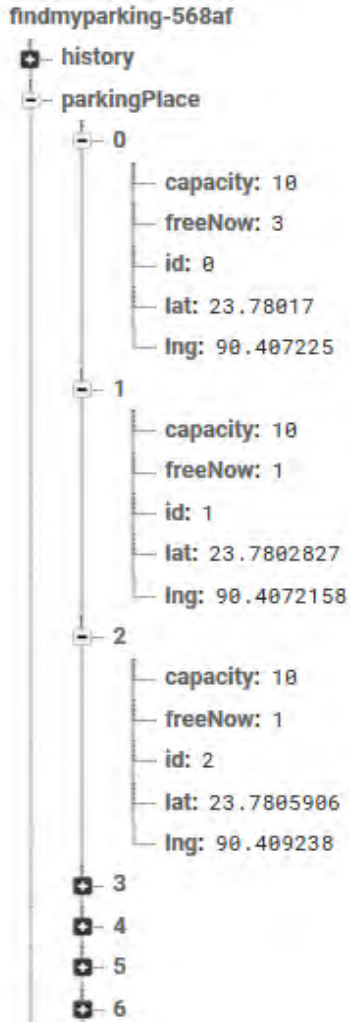


Figure 17: Parking Place DB

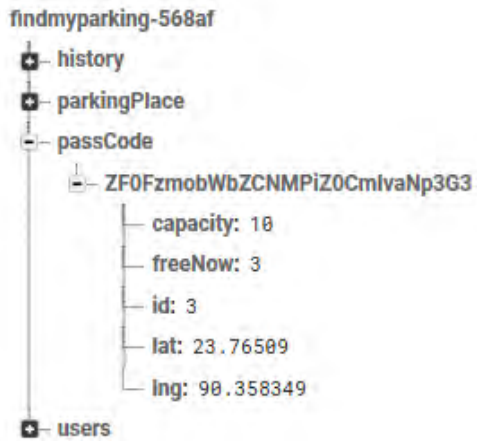


Figure 18: Passcode DB

findmyparking-568af



Figure 19: Users DB

5.6 System Implementation

Some important code snippets of our software system are included below:

Our Manifest File

Where we declared all our activities and take all the necessary permissions from the user that is required for the application to run smoothly.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="corp.tz.findmyparking">

    <permission
        android:name="corp.tz.findmyparking.permission.MAPS_RECEIVE"
        android:protectionLevel="signature"/>

    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="corp.tz.findmyparking.permission.MAPS_RECEIVE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/logo"
        android:label="Find My Parking"
        android:roundIcon="@drawable/logo"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

```

<application
    android:allowBackup="true"
    android:icon="@drawable/logo"
    android:label="Find My Parking"
    android:roundIcon="@drawable/logo"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="AIzaSyDlXtKoCIB9RtQzKn_ubMjttD-Rf4E85W-Q"/>

    <meta-data
        android:name="io.fabric.ApiKey"
        android:value="2f0dba9963664f988d312e2fffb6ffd364040f42e"/>

    <!-- <meta-data -->
    <!-- android:name="com.google.android.geo.API_KEY" -->
    <!-- android:value="AIzaSyBApSCVq6hhSttuXJ2YdLOyHsEaGhEvl_w"/> -->

    <activity
        android:name=".activity.MainActivity"
        android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>

            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <activity
        android:name=".activity.LoginActivity"
        android:screenOrientation="portrait"/>
    <activity
        android:name=".activity.RegisterActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme.NoActionBar"/>
    <activity
        android:name=".activity.HomeActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme.NoActionBar">
    </activity>
    <activity android:name=".activity.MapsActivity"/>
    <activity
        android:name=".activity.ProfileActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme.NoActionBar">
    </activity>
    <activity
        android:name=".activity.HistoryActivity"
        android:label="HistoryActivity"
        android:theme="@style/AppTheme.NoActionBar">
    </activity>

```

```

        <activity
            android:name=".activity.MySpotsActivity"
            android:label="MySpotsActivity"
            android:theme="@style/AppTheme.NoActionBar">
        </activity>
    </application>

</manifest>

```

App Dependencies

All the third party APIs and libraries that we have used to build our application.

```

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })

    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.google.firebase:firebase-database:10.0.1'
    compile 'com.google.firebase:firebase-core:10.0.1'
    compile 'com.google.firebase:firebase-auth:10.0.1'
    compile 'com.google.firebase:firebase-storage:10.0.1'
    compile 'com.android.support:mediarouter-v7:25.3.1'
    compile 'com.google.android.gms:play-services-maps:10.0.1'
    compile 'com.android.support:design:25.3.1'
    compile 'com.squareup.retrofit2:retrofit:2.1.0'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'
    compile 'com.intuit.sdp:sdp-android:1.0.3'
    compile 'de.hdodenhof:circleimageview:2.1.0'
    compile 'com.google.android.gms:play-services-location:10.0.1'
    compile 'org.greenrobot:greendao:3.1.1'
    compile 'com.siclo.ezphotopick:library:1.0.6'
    compile 'com.github.bumptech.glide:glide:3.7.0'
    compile('com.crashlytics.sdk.android:crashlytics:2.6.8@aar') {
        transitive = true;
    }
    testCompile 'junit:junit:4.12'
}

apply plugin: 'com.google.gms.google-services'

```

Taking User's Current Location

We have taken a user's current location using the android's native Location Manager class and then put a pin on the map showing the location using Google maps API.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    if (ActivityCompat.checkSelfPermission(this, android.Manifest.permission
        | .ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission(this, android.Manifest.permission
        | .ACCESS_COARSE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
        // TODO: Consider calling
        //     ActivityCompat#requestPermissions
        // here to request the missing permissions, and then overriding
        //     public void onRequestPermissionsResult(int requestCode, String[] permissions,
        //                                             int[] grantResults)
        // to handle the case where the user grants the permission. See the documentation
        // for ActivityCompat#requestPermissions for more details.
        return;
    }
    googleMap.setMyLocationEnabled(true);
    LocationManager locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
    Criteria criteria = new Criteria();
    String provider = locationManager.getBestProvider(criteria, true);
    Location location = locationManager.getLastKnownLocation(provider);
```

```
if(location != null){
    double latitude = location.getLatitude();
    double longitude = location.getLongitude();
    MarkerOptions options = new MarkerOptions()
        .title("My Current Location")
        .position(new LatLng(latitude, longitude));
    mMap.addMarker(options);

    LatLng ll = new LatLng(latitude, longitude);
    CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(
        ll, 14);
    mMap.animateCamera(cameraUpdate);
}
```

Chapter 7

7.1 Limitations

- A prototype of a broad idea has been presented. We need much more time to make it deliverable. At this moment, we are not following any ideal structure or model. We will create one in the future.
- Whenever we turn on location system in our app, the exact location of the user may get deviated a bit due to network interferences from around the environment.
- The hardware part was implemented with Arduino which has a very limited amount of ram and motivated us thinking critically to make the project work.
- As for now, our project covers a smaller span of people for a ideal scenario and therefore Arduino is working fine in here. But, in the near future, our app covering a wider group of people with much more complex scenario will definitely need a more stable system like Raspberry Pi.
- Firebase being a database platform is good for certain amount of data handling. Massive data handling may result to a cumbersome scenario. But, we shall make it robust in the near future with a dedicated database.
- The data request and data retrieval system is conducted among the app, the Arduino and the database which may make the process a bit slower.
- As researchers we lacked knowledge of cloud system beforehand. As a result we learnt about the system and then got to work.
- A transaction system could have been implemented with PayPal API which at present, is not included.

7.2 Future Work

- **Multi-Platform application:** Our project is currently an android-platform mobile application. It can be rebuilt in a multi-platform structure so that windows and iPhone users can also use this app.
- **Data Size:** With more time, our database handling features will improve and with premium database, our application will be able to provide more edgy service.
- **Improving the sample area:** Currently, our application is working for Dhaka city only. In future, parking spots across the globe can be considered as a sample area and our application will be able to handle that nicely.
- **VIN number and IPv6:** As the implementation of IPv6 becomes more prominent, we will map a vehicle's 132 bit alphanumeric VIN number (unique for each car worldwide) to a 128 bit IP, allowing us to identify and communicate with vehicles directly, without the need for human intervention. The VIN code can be split into 3 parts: WMI, VDS, and VIS. Each part gives a partial description of one vehicle's attributes: manufacturer ID, vehicle's description and vehicle's serial ID. There is an algorithm to perform in order to set an IPv6 Interface Identifier out of a VIN code (VIID). The method used is based on two short and powerful assumptions that allow for the compression of maximum number of VIN digits in a minimum number of bits. The algorithm if implemented properly, will surely take our application to a new level.
- **Car Application:** In the near future, with enough modifications, we can build up an application suitable for every car available on the planet if implemented with IPv6.
- **Walkthrough:** Someone with no prior knowledge of booking a parking lot might be interested in our application. For those people, snapshots of sequential steps can be included in the system.
- **Simultaneous Access:** The application is in a prototype version. So, it might not be able to handle many users at the same time. Handling more simultaneous users through improving application performance is a future scope.

- Security: We are currently using basic security protocols for the application and the cloud database. But when the application is considered for real life implementation, more versatile encrypted security system can be considered.
- Tracker: By using gps and such technology, user will no longer be required to give their location as input. Location of the user will be tracked immediately as the user logs in to the app.
- Feedback: Initially we will take feedback from the users in old fashioned way. Live conversation feature can be incorporated to the application.

7.3 Conclusion

The ease of parking system is quite a challenge in modern days. Since the advent of industrialized cities, number of cars has been increasing and day by day people are facing bigger trouble while trying to manage their cars into a parking lot. This scenario of parking crisis gives rise to new solutions with the help of Internet of things (IOT) thus managing car parking systems. Our paper addresses the crisis of car parking across a remote city and comes out with an IoT based assistant mobile application system. The proposed project provides real time information of a car parking lot and is able to coordinate with the mobile application thus giving user the feasibility of booking a parking lot staying at a distance.

References

- [1] Y. Geng and C. G. Cassandras, "A new smart parking system based on optimal resource allocation and reservations," in Proc. 14th Int. IEEE Conf. Intell. Transp. Syst. (ITSC), Oct. 2011, pp. 979–984.
- [2] Y. Geng and C. G. Cassandras, "New smart parking system based on resource allocation and reservations," IEEE Trans. Intell. Transp. Syst., vol. 14, no. 3, pp. 1129–1139, Sep. 2013.
- [3] X. Zhao, K. Zhao, and F. Hai, "An algorithm of parking planning for smart parking system," in Proc. 11th World Congr. Intell. Control Autom. (WCICA), 2014, pp. 4965–4969.
- [4] L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and R. Vergallo, "Integration of RFID and WSN technologies in a smart parking system," in Proc. 22nd Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM), 2014, pp. 104–110.
- [5] C. W. Hsu, M. H. Shih, H. Y. Huang, Y. C. Shiue, and S. C. Huang, "Verification of smart guiding system to search for parking space via DSRC communication," in Proc. 12th Int. Conf. ITS Telecommun. (ITST), 2012, pp. 77–81.
- [6] R. E. Barone, T. Giuffrè, S. M. Siniscalchi, M. A. Morgano, and G. Tesoriere, "Architecture for parking management in smart cities," IET Intell. Transp. Syst., vol. 8, no. 5, pp. 445–452, 2014.
- [7] C. Shiyao, W. Ming, L. Chen, and R. Na, "The research and implement of the intelligent parking reservation management system based on ZigBee technology," in Proc. 6th Int. Conf. Meas. Technol. Mechatronics Autom. (ICMTMA), 2014, pp. 741–744.

- [8] D. J. Bonde, R. S. Shende, K. S. Gaikwad, A. S. Kedari, and A. U. Bhokre, "Automated car parking system commanded by Android application," in Proc. Int. Conf. Comput. Commun. Inform. (ICCCI), Coimbatore, India, Jan. 2014, pp. 1–4.
- [9] L. Lambrinos and L. Dosis, "DisAssist: An Internet of Things and mobile communications platform for disabled parking space management," in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2013, pp. 2810–2815.
- [10] Geng, Y., and C. Cassandras. (2013). New Smart Parking system based on resource allocation and reservations. IEEE Transactions on Intelligent Transportation Systems, 14, 5, (2013) 1129-1139.
- [11] V. W. S. Tang, Y. Zheng, and J. Cao, "An intelligent car park management system based on wireless sensor networks," in Pervasive Computing and Applications, 2006 1st International Symposium on , 2006, pp. 65–70.
- [12] J. Chinrungrueng, U. Sunantachaikul, and S. Triamlumlerd, "Smart parking: An application of optical wireless sensor network," in Applications and the Internet Workshops, 2007. SAINT Workshops 2007. International Symposium on, 2007, pp. 66–66.
- [13] G. Yan, W. Yang, D. B. Rawat, and S. Olariu, "Smartparking: A secure and intelligent parking system," Intelligent Transportation Systems Magazine, IEEE, vol. 3, no. 1, pp. 18–30, 2011.
- [14] R. Lu, X. Lin, H. Zhu, and X. S. Shen, "An Intelligent Secure and Privacy-Preserving Parking Scheme Using Vehicular Communications," Engine, 2009.
- [15] K. C. Mouskos, J. Tsvantzis, D. Bernstein, and A. Sansil, "Mathematical formulation of a deterministic parking reservation system (PRS) with fixed costs," in Electrotechnical Conference, 2000. MELECON 2000. 10th Mediterranean, 2000, vol. 2, pp. 648–651.
- [16] Arduino.cc, "Functional application of Hex Keypad", 2014. [Online]. Available: <http://playground.arduino.cc/Code/Keypad> [Accessed: 14- Dec- 2016].
- [17] code.google.com, "Tinkerit-TrueRandom.wiki", 2015. [Online]. Available: <https://code.google.com/archive/p/tinkerit/wikis/TrueRandom.wiki> [Accessed: 25- Jan- 2017].

[18] [thedailystar.net, 'The Smartest Ways To Deal With Traffic Congestion in Dhaka', 2016.](http://www.thedailystar.net/op-ed/politics/the-smartest-ways-deal-traffic-congestion-dhaka-1220956) [Online]. Available:<http://www.thedailystar.net/op-ed/politics/the-smartest-ways-deal-traffic-congestion-dhaka-1220956> [Accessed: 16- Apr- 2017].

[19] [academia.edu, 'Managemental causes of traffic congestion in Dhaka city and its possible solution', 2017.](https://www.academia.edu/30649982/Managemental_causes_of_traffic_congestion_in_Dhaka_city_and_its_possible_solution?auto=download) [Online]. Available:https://www.academia.edu/30649982/Managemental_causes_of_traffic_congestion_in_Dhaka_city_and_its_possible_solution?auto=download [Accessed: 16- Apr- 2017].

