Smart Parking System using Cloud based Computation and Raspberry Pi

Conference Paper · October 2018

DOI: 10.1109/I-SMAC.2018.8653764

3 authors:

Akash Gupta
Shiv Nadar University
2 PUBLICATIONS **9** CITATIONS
SEE PROFILE

Priyansh Rastogi
Shiv Nadar University
1 PUBLICATION **9** CITATIONS
SEE PROFILE

Shaurya Jain
Shiv Nadar University
1 PUBLICATION **9** CITATIONS
SEE PROFILE

Proceedings of the Second International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC 2018)
IEEE Xplore Part Number:CFP18OZV-ART; ISBN:978-1-5386-1442-6

Smart Parking System using Cloud based Computation and Raspberry Pi

Akash Gupta
Dept. of Computer Science,
Shiv Nadar University
Greater Noida, India
ag102@snu.edu.in

Priyansh Rastogi
Dept. of Computer Science,
Shiv Nadar University
Greater Noida, India
pr266@snu.edu.in

Shaurya Jain
Dept. of Computer Science,
Shiv Nadar University
Greater Noida, India
sj762@snu.edu.in

***Abstract*—The internet of things plays a vital role in interconnection and automation of various physical devices, vehicles, home appliances and other things. With the help of software, various sensors, actuators, these objects connect and exchange data. This automation of devices enhances a person's standard of life and way of living, which is a need of future. A similar need is discussed in this paper. In this project a smart parking feature is discussed which enables a user of find a parking location and a free slot in that parking space inside a city. This project focuses on reducing time wasted on finding parking space nearby and on going through the filled parking slots. This in turn reduces the fuel consumption and standard of living.**

Keywords—Smart Parking, Location based parking, IoT, sensors, Raspberry Pi, Arduino, Node.js.

I. INTRODUCTION

It has been recorded that in coming year urbanization will skyrocket as people continue to migrate to the cities in the hope of better living. This will place a considerable strain on the existing infrastructure of the cities [7]. Such a strain will be of finding a parking space inside a city. This issue of parking is becoming day-by-day a key cause of traffic congestion, driver frustration and air pollution [2]. This system aims at solving this problem using internet of things. The internet of things aims at making things efficient and safer. Inside a smart city, a smart parking system is a much needed system to save money, time, and even environment. This internet of things system aims at developing a smart parking system inside a smart city which automatically finds the nearest available parking slot. Using progressive web app which user can login with Google and find the nearest available parking slot just with one tap and it tracks user's parking sessions in a parking using RFID tags. It is also used to recognize a user, his entry and exit date and time and based on this automatically calculating his charges for the parking. The user's parking invoice is sent to this email-id which he can pay later on. This system uses ultrasonic sensors and each unit of ultrasonic sensor acts as a slot inside a parking space.

II. RELATED WORK

In this section we have discussed about the work conducted till now on this topic of smart parking system. In paper [3], a smart parking system is proposed which detects and find a parking location for consumer's vehicle that works on wireless sensor network and Bluetooth of user's smartphone. It specifies for indoor and outdoor parking separately as it uses ultrasonic sensor for indoor and magnetic sensor for outdoor parking. As for the location of the user Bluetooth and USIM ID are used. This method provides a cheap solution than our Ultrasonic and RFID based solution but this is not an effective and a smart solution to this parking issue. The solution proposed in this paper is end-to-end solution with the use of latest technologies in the market which provides the user with a navigation option with which he can drive to the parking location in no time. There is a one-time cost involved in the solution of this paper but the future aspects of the proposed solution in our paper far more exceeds the future value of solution in paper [3]. These technologies used in this system can be used to fulfil the future aspects such as a solution of smart-parking in smart or driverless cars.

III. HARDWARE AND SOFTWARE USED

A. Hardware

a. Raspberry Pi

The raspberry pi is a low cost, credit-card sized system-on-chip(SoC) device that can be plugged into a computer monitor. It uses a standard keyboard and mouse. A SoC device is a complex integrated circuit that integrates the major functional elements into a single chip or chip-set [2]. The Raspberry Pi 2 Model B+ is used in this project.

b. Arduino

Arduino is a pre-assembled microcontroller board which have series of input/output(I/O) pins that may be interfaced to various expansions boards and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers [3]. Arduino model used here is Arduino Uno R3.

c. Sensors

i. Ultrasonic sensor(HC-SR04)

An Ultrasonic sensor is a device that can measure the distance to an object by using sound waves. It measures distance by sending out a sound wave at a specific frequency and listening for that sound wave to bounce back [1]. Considering the scope of this project, 4 HC-SR04 units are used.

ii. RFID(RC522)

RFID is radio frequency identification which uses electromagnetic fields to automatically identify and track tags attached to the objects. The tags contain electronically-stored information [4]. The operating frequency of RFID tags is 13.56MHz. RFID RC522 is used in this system.

B. Software

a. Raspberry Pi

- Node.js: It is an open-source, cross-platform JavaScript environment for developing server-side and networking applications [7].
- Socket.io-client: Socket.io enables real-time bidirectional event-based communication. It's an implementation of WebSockets in JavaScript.
- Pigpio: Pigpio is a library for the Raspberry which allows control of the General Purpose Input Outputs (GPIO). Pigpio works on all versions of the Pi.

b. Cloud Application

- Node.js
- Express: Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- Socket.io
- AWS-EC2: Amazon Web Service Elastic Compute Cloud (AWS EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. The system's cloud application is hosted on the AWS.
- MongoDB (mLab hosted): MongoDB is a NoSQL database which is written in C++ [5]. For this system, it is hosted on mLab. mLab is a cloud service dedicated to MongoDB.
- Helper node modules such as passport, Passport is Express-compatible authentication middleware for Node.js. Google authentication is done using passport.
- Nodemailer: Nodemailer is a module for Node.js applications which sends email easily. The system feature which includes sending invoice to users is implemented with this module.
- Google Distance Matrix API: The Google Maps Distance Matrix API is a service that provides travel distance and time for a matrix of origins and destinations.

c. Client Application

- React: React is a javascript library for building User Interfaces. The system's progressive web app is built with React.
- Redux: Redux is an open-source JavaScript library for managing application state. It is used with React library for building user interface for this smart parking system.

IV. ARCHITECTURE

A. Design

The application is divided into three parts: 1. Raspberry Pi application. 2. Cloud Application and 3. Client Application.

The Raspberry Pi Application senses the environment data, which in our case is, number of free slots in a parking and RFID tags. Ultrasonic sensors are used to determine if a parking slot is free or not. Each slot of the parking has an ultrasonic sensor which reads the distance of the obstacle every 5 second. If the distance is less than a threshold value,

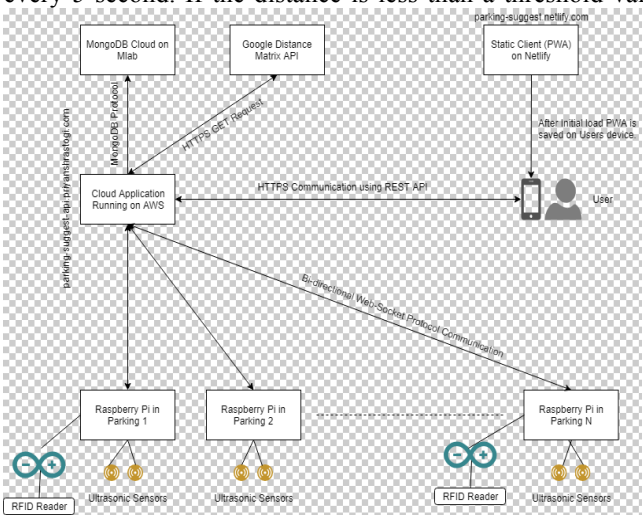


Figure 1: IoT Project Architecture

the parking slot is busy otherwise it is free. The Raspberry Pi application sends this data to cloud application.

Cloud Application reads the data sent by Raspberry Pi application and stores it in the MongoDB. As a user sends a request to find the nearest available parking, it finds the nearest parking with respect to user's current location and check the logs to find if this parking has any free parking slots, it does the same until it finds an available parking slots and sends the response back to the user. It also reads the triggers sent by Raspberry Pi application when a user enters the parking using RFID, it maintains the logs such as entering time, exit time, parking id, billed amount for the parking and emails the user invoice for a parking session.

Client Application is a progressive-web-app using which user can access our services. Using Google OAuth, User can sign in to our application using their Google Identity can find nearest available parking with one tap. Users current location is required for this operation, if user grants the location permission, client application will send request to the cloud and will show the response to the user. Using the Navigate button, user can navigate to the parking with Google Maps.

Connections and Data Flow

Raspberry Pi and Cloud application are connected using web socket protocol. It is a bidirectional communication protocol built on TCP. As Raspberry Pi Application boots up, it sends the connection request to the cloud with Raspberry Pi ID and authentication key, In the process of handshake, it authenticates the connection using this combination and establishes the connection after successful authentication. This connection works a tunnel. It is a bidirectional communication link. It is better than using HTTP or HTTPS protocol because they are heavy protocols and RPi have to wait for response after sending POST request to cloud.

Cloud application and Client Application are connected using HTTPS protocol, since user's location is a private information, encryption is needed in both client and server side. We are using Let's Encrypt free SSL certificates for both sides. Client application is hosted on Netlify, URL is: https://parking-suggest.netlify.com, Cloud application is hosted on Amazon Web Services EC2 instance (Ubuntu 16.04, 1GB RAM), URL is: https://parking-suggest-api.priyanshrastogi.com.

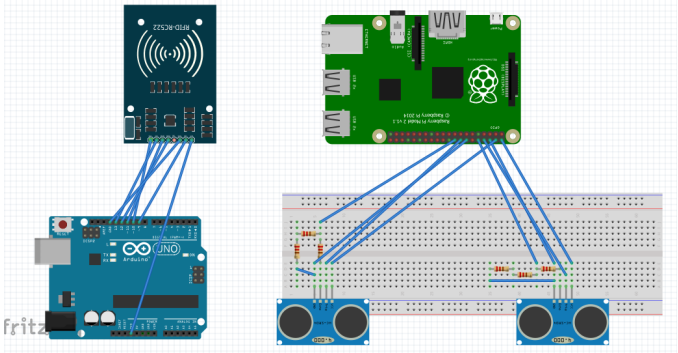


Figure 2: Fritzing Diagram of the Circuit

B. Implementation

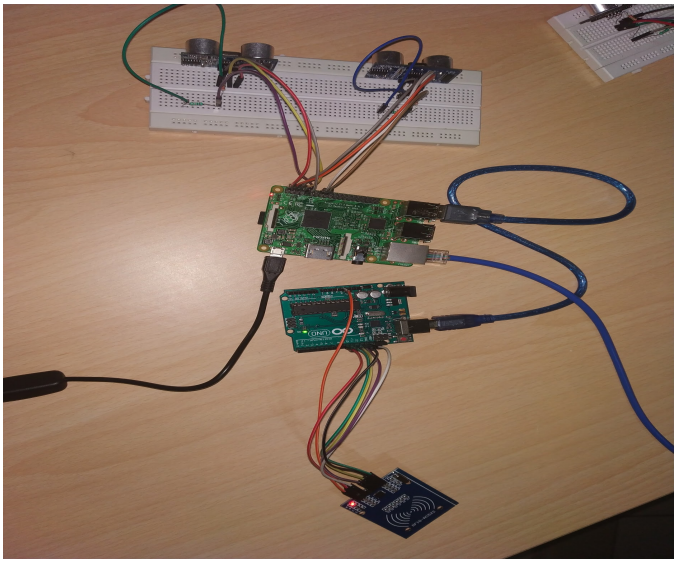


Figure 3(a): Circuit Diagram of one RPi module

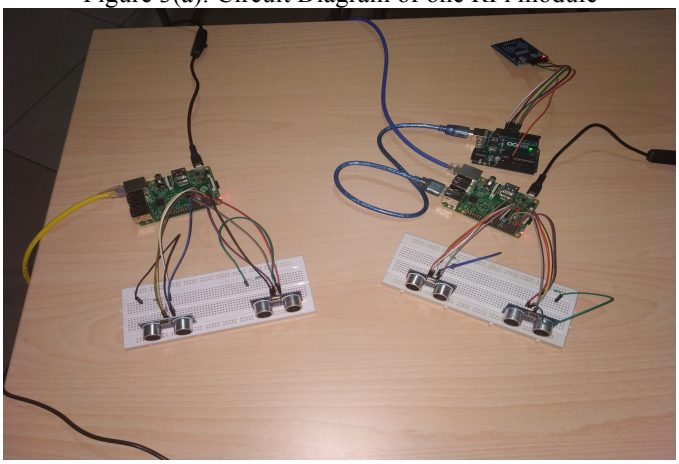


Figure 3(b): Circuit diagram of both RPi modules

In this project two raspberry pi 2 module has been used where each of them is analogous to a parking space according to their functionality. Ultrasonic sensors are used to find whether a parking slot is empty or not, which are eventually connected to the raspberry pi. RFID RC522 is used to find out the user's information and mapping a certain user to a parking slot. RFID module is connected to the Arduino. Arduino is then connected to the raspberry pi using the serial port communication. Figure 2 describes the fritzing diagram of the one raspberry pi module with Arduino and RFID sensor. Figure 3(a) and 3(b) describes the actual circuit diagram of smart parking system.

Developing the Raspberry Pi Application

Raspberry Pi acts as a parking location where each parking location is assigned a name, its location i.e. its longitude and latitude, its capacity. We are using ultrasonic sensor which behaves as a parking slot and using pi-gpio module (a simple node.js based library to help access the GPIO of the Raspberry Pi) we are connecting our ultrasonic

sensor to the node.js application, where ultrasonic sensor is sensed every 5 second to determine if the location is free or not. Arduino is connected serially to Raspberry Pi by using a node module namely **Serialport** which provides a stream interface for the low-level serial port code necessary to control arduino chipsets. Arduino here acts as a bridge between Raspberry Pi and RFID tag. Now whenever a RFID tag is read by the RFID module, the arduino will write the RFID tag to serial port and then the raspberry pi will read this data and send it to the cloud.

Developing the Cloud Application

A. **Server and Database Setup**

For cloud application we are using Amazon Web Services Elastic Computing Cloud Instance (Ubuntu 16.04, 1GB, 1 Core) and MongoDB which is a NoSQL database. The database is hosted on mLab (MongoDB as service provider). The application is using HTTPS protocol, so we generated Let's Encrypt SSL keys with Certbot.

B**. Database Schema**

- User Schema - googleId, name, email, phone, rfidTag.
- Parking Schema - name, location (lat, long), capacity, perhourprice, authToken, rpiId, sessionId (where sessionId is the id of current websocket connection of raspberry pi to cloud server, authToken is a random generated token used to authenticate a raspberry pi).
- ParkingLogs Schema - reference of a parking using parkingId, freeSlots, datatime (timestamp of this log)
- RPiLogs Schema - rpiId, sessionId, connectedTime, disconnectedTime, disconnectReason
- UserParking Schema - reference of a user using UserId, reference of a parking using parkingId, inTime(time when a user enters into a parking), outTime(time when a user exits a parking), bill, status(there are two types of status a)parked: user car is parked b)done:user has left the parking slot , here default value is parked).

C. **Routes**

Our Node.js cloud application has these routes.

- Authentication (POST /auth/google): We have used a node module named passport.js and passport-google-oauth20 that impelments Google OAuth2.0. By clicking on Sign In With Google button user is redirected to Google's OAuth flow and it returns a access token if user successfully authenticated with Google, using the access token we can get user's name, email and google ID, which we store in our database.
- Parking
    - Add Parking (POST /parkings/add) - This route is used to add a parking slot to the application. We need to give rpiId, parking name, latitude and longitude of location, capacity and price per hour of parking. Server will generate a 8 byte authentication token that will be used to authenticate the raspberry pi to server in order to send the logs.
    - Find Nearest Parking (GET /parkings/findnearest) - This route finds nearest parking location w.r.t to user's current location. Using Google Maps Distance Matrix API find distance of each parking in the database and make an array of JSON objects that contain parkingId and distance from user's location and it is sorted in ascending order w.r.t user's current location. Now find number of free slots available in first parking of the array using latest log as it is the nearest parking to the user. If free slots are available, return it to the user otherwise check next parking in the array until a free slot is found, if there aren't any free slots in any of the parking return not found to the user.
- User
    - GET users (GET /users/email) - This route finds the user with a given email.
    - Add RFID for a user (POST /users/email/RFID) - This route finds the user with a given email, if the user is found it adds RFID Tag to that user, otherwise returns an error response.

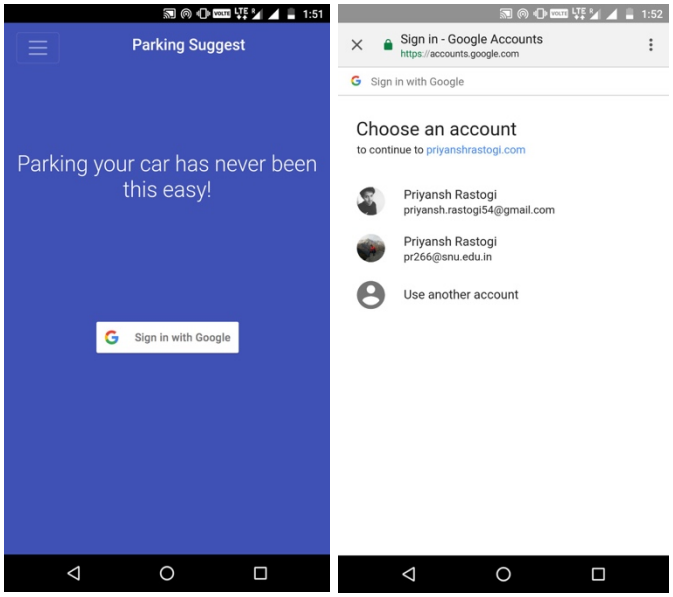
D. **Socket.io Events**

Socket.io is event-based, so we need to define some events. So as if data is received associated with these events this is to be done further. Following events are defined in our application.

- Logs: Socket.io will execute this event as data is received with this event name. Logs event is parking logs, as a parking log is received with a payload that is a JSON object with two keys parkingId and freeSlots, A document in collection ParkingLogs is created.
- RFID: As data is received with this event name with payload which contains rfidTag read by RFID reader, first we find the User associated with that RFID Tag and then we find a document in UserParking collection with userId, parkingId and 'parked' status. If find a document as result, we add outTime and calculate the billed amount and change the status to 'done' and mail user parking invoice with billed amount. If we don't get any document in result then we create a new document in UserParking with userId, parkingId, (inTime will be automatically set to current time and status will be set to 'parked' by default) and email user the parking alert.

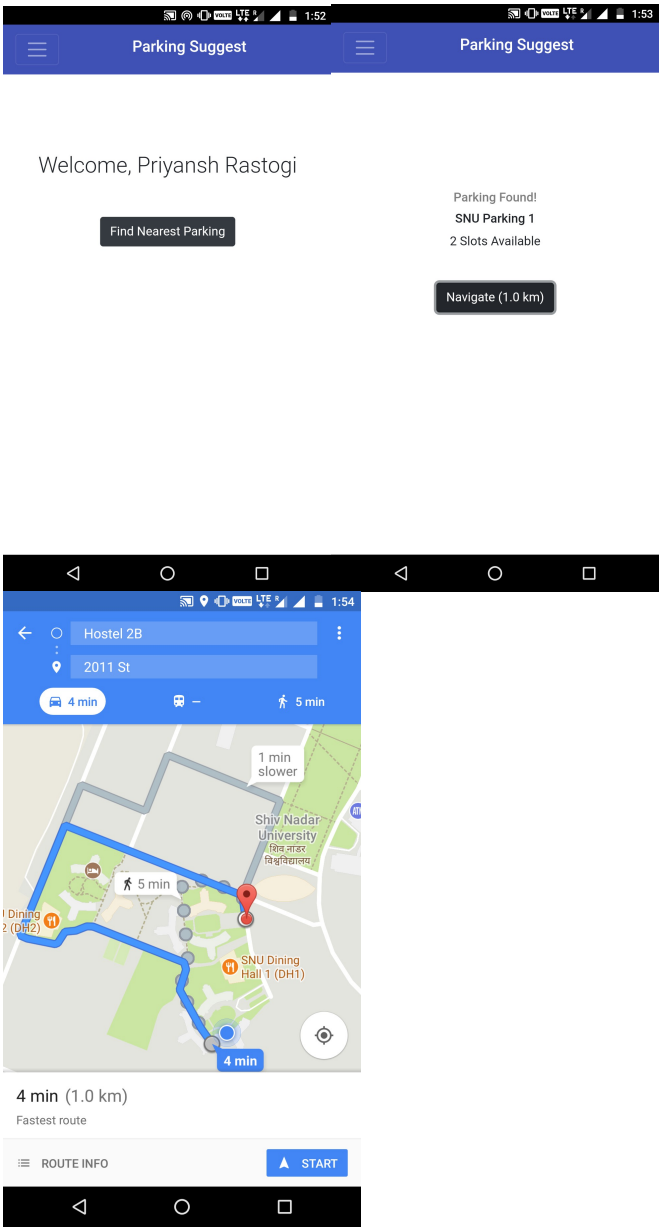
V. WORKING

This smart parking system works as follows:



Firstly, the user opens up the progressive web app or we can say user interface on the https://parking-suggest.netlify.com. Since this is a progressive web app, so the app will be saved on the user's devices after the initial load. So, whenever the user opens up the app for another session, it will not waste time in loading up the contents. The cloud server for this parking suggestion system is hosted on AWS EC2 on the https://parking-suggest-api.priyanshrastogi.com.The communication between the user's application and the cloud application will take place via HTTPS protocol using REST API. As user clicks on Sign In with Google button, a POST request will be sent to server on /auth/google and user will be redirected to Google OAuth flow, where user can select a Google Account, from where the server will now get the access to users name, email and google Id. If the user is a first time user, a new account will be created and a JSON web token will be generated which will be sent to client app and will be stored in browser's local storage. This JWT will be included in the header of the requests that user will make. Now as user is signed in, user can click on find nearest parking button and a POST request will be sent to server and using the Authorization header that contains JWT, server will authorize the user and will return the nearest parking location with respect to user's current location. The flow of the request to response will be the following:

Raspberry Pi is continuously sending free slots logs every 5 seconds which is being stored in MongoDB in ParkingLogs collection. As a POST request is received on route /parkings/nearestparking, if the request contains a valid JSON web token, the request will be authorized and the request will enter the next middleware. The request contains latitude and longitude of user's location and the Google Distance Matrix API will get the distance between user's location and all

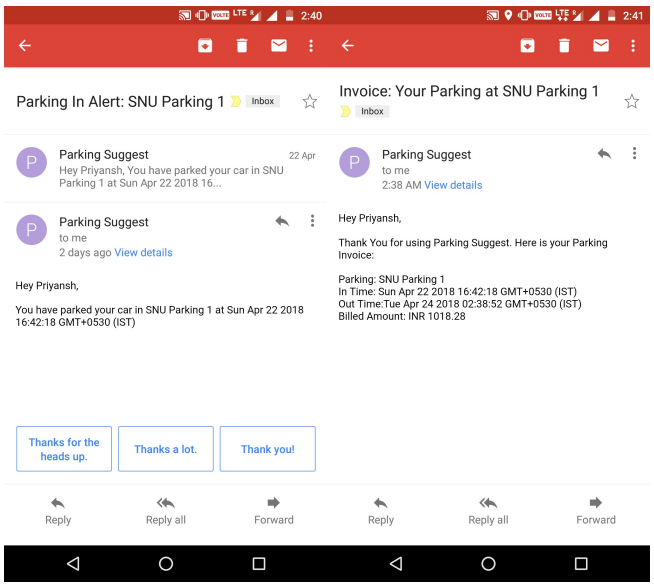


parking and it will sort the parking according to distance to user and will search if a parking has free space or not if a parking has free space it will be sent as response to client otherwise it will search for next parking and so on. If free slots are not available in any parking, the server will send a "Not found" response.

User can see the parking in this screen and by clicking on Navigate button Google Maps app will be opened and user will be navigated to nearest parking.

Now as user goes to a parking, if the user is using Parking Suggest parking for the first time, user will be assigned an RFID Card containing the tag. As user enters or exits the parking the RFID tag will be read and send to the cloud, now server will find the user associated with the RFID tag and will find a document in UserParking collection to see if the user has an active parking session in the parking, if finds the

document it will end it and calculate the billed amount and mail the invoice to the user otherwise it will create a new parking session for the user and send the mail to the user.



ASSUMPTIONS

1). Raspberry pi should be connected to the internet all the time.
2). Google matrix API, Ultrasonic sensors, RFID, Arduino, should be functional all the time.

LIMITATIONS

1). Ultrasonic sensor can give faulty readings whenever any obstacle other than the car in the parking is detected.

ACKNOWLEDGMENT

We are thankful to Dr. Debopam Acharya and Dr. Divya Lohani for the continued guidance and support which helped us towards the completion of the project.

REFERENCES

[1] D. Vakula and Y. K. Kolli, "Low cost smart parking system for smart cities," 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, 2017, pp. 280-284.

[2] K. Hassoune, W. Dachry, F. Moutaouakkil and H. Medromi, "Smart parking systems: A survey," *2016 11th International Conference on Intelligent Systems:* Theories and Applications (SITA), Mohammedia, 2016, pp. 1-6.

[3] C. Lee, Y. Han, S. Jeon, D. Seo and I. Jung, "Smart parking system for Internet of Things," *2016 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, 2016, pp. 263-264.

[4] S. Shinde, A. Patil, S. Chavan, S. Deshmukh and S. Ingleshwar, "IoT based parking system using Google," *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, 2017, pp. 634-636.

[5] N. R. N. Zadeh and J. C. D. Cruz, "Smart urban parking detection system," 2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Batu Ferringhi, 2016, pp. 370-373.

[6] P. Sadhukhan, "An IoT-based E-parking system for smart cities," *2017 International Conference on Advances in Computing, Communications and I*nformatics (ICACCI), Udupi, 2017, pp. 1062-1066.

[7] IoT for Smarter Cities
http://intelzone.com/smarter-cities/

[8] Node.js Documentation
https://nodejs.org/en/docs/

[9] Raspberry Pi Documentation
https://www.raspberrypi.org/

[10] Arduino Uno Rev3
https://store.arduino.cc/usa/arduino-uno-rev3/

[11] Arduino RFID reader using the RC522 Module
https://www.brainy-bits.com/card-reader-with-an-arduino-rfid-using-the-rc522-module/