



**ELEC 3848 Integrated Design Project  
Final Report**

**Smart Parking  
Easy Parking, Easy Life**

**Group 19**

**Ho Yu Hin(ElecE, 3035276696)  
[Hided for privacy]**

## **Abstract**

Parking is an essential part of a modern city. To construct a smart city, smart parking would inevitably take a vital role. Therefore, we aim to build a smart parking system in this project.

As one may notice in the title slogan “Easy Parking, Easy Life”, the project objective is to facilitate parking of drivers. It is a common phenomenon that the drivers realize a car park has no parking space vacancy at the very last minute. It turns out to waste a lot of time to find a proper car park with parking space vacancy. The major benefit of the smart car park system is to help avoid such kind of problem. A number of ultrasonic sensors would be installed in the parking space so that the vacancy status could be acquired. Various states of parking space would be sent to database as an intermediate process to update the vacancy of different car parks in the phone application. Basically, the users can use the phone application to check the availability of parking space in different car parks.

An embarrassing situation would happen if a driver is driving to the car park with available parking space but it is occupied by another car before he arrives. Therefore, the parking system allows the drivers to reserve a parking space for fifteen minutes through the phone application. This could be simply implemented by updating the state of the reserved parking space to avoid it being rented or reserved by other drivers.

Furthermore, if the size of a car park is large, it takes time for drivers to find the correct parking space. With a view to saving drivers time for searching parking space, LED lamps could be installed in the car park to provide a sufficient guidance. Basically, when the driver taps the card in the entrance of the car park, it does not only activate the gate to turn on, but it turns on a set of LED lamps by identifying the corresponding parking space received from the data of the card reader. A light path would be created to guide the driver to the correct parking space.

With all these features, the mere objective of the smart parking system would be achieved that it facilitates and improves the parking experience of the drivers.

## **Project Objectives**

1. To construct a lighting system with LED indicators, which guides the car to the corresponding parking place by identifying it from the data of the card reader.
  - 1.1 Data of the card reader would be sent to the lighting system
  - 1.2 Automatic LED lighting system would turn on the corresponding light path to guide the driver to the correct parking space by the data of the card reader.
2. To set a LCD monitor to show the vacancy of parking space in the carpark.
  - 2.1 It gets data from the database retrieved from the Raspberry Pi and shows the current number of parking space vacancy in the car park.
3. To build a smartphone application (app)
  - 2.1 Login/Register/Logout. It allows the users to login, register and logout in the App.
  - 2.2 Reservation function. The users could reserve a parking space in a car park with vacancy. Car park vacancy information will be collected from database for implementing the reservation function in the app.
  - 2.3 Record function. As the database has saved the reservation records of the users, they can track the reservation history using the function “Record”.
  - 2.4 Cancel Booking. After the reservation of the parking space, the system allows the users to cancel booking.
  - 2.5 Timeout. The time limit for the reservation is 15 minutes. After the time limit, the system would automatically cancel the booking, The reserved parking space would then become available again.
4. To build gates in the entrance or exit in the car park model
  - 3.1 A payment process will be modelled by tapping a card to open the gate
  - 3.2 It sends the data from the card reader so that the lighting system could be notified and turn on an appropriate light path to guide the driver to the correct parking space.

## Project Deliverable

### Automatic LED Lighting System

Function	<b>Provide light guidance to drivers</b> When the lighting system receives the signal from the card reader system, it would turn on the respective light path to guide the drivers to the correct parking space.
Structure	This module is constructed with an <b>Arduino UNO board</b> as a controller and <b>LED lamps</b> .
Input	<b>Control signal</b> (Arduino MEGA board ) <b>Data the card reader</b> (from another Arduino UNO board)
Output	<b>Power to the LED lamps</b> (on / off)

### Parking and App System

Function	<b>Register/Login/Logout</b> Users can register, login and logout in the app. <b>Show vacancy states</b> We collect all data from each ultrasonic sensor in the parking spaces. After that, we can provide drivers the vacancy states in the App <b>Reservation</b> Users can reserve available parking space through the App. <b>Record</b> Parking records through reservation via the App can be found in App for reference. <b>Cancel Booking</b> Users can cancel booking of reservation.
Structure	This module is constructed with <b>Android studio</b> and a <b>MySQL database</b> .
Input	<b>Data from the database, user inputs in the phone application</b>

Output	<b>Data from the database, user inputs in the phone application</b>
--------	---

#### Server / Database

Function	<p><b>Parking space vacancy</b></p> <p>A Apache server and a MySQL database are set up using Raspberry Pi. It would collect the vacancy state of the parking space getting from the ultrasonic sensors from Arduino and update the information in the database.</p> <p><b>Reservation Record</b></p> <p>The database will record all of the information about the current and past reservation for the implementation of the “Reservation” and “Record” function in the app</p>
Structure	This module is constructed with an <b>Raspberry Pi and an Arduino UNO board</b>
Input	<b>Data signal</b> (Arduino UNO board) and <b>the reservation data</b> (from the App)
Output	Data in database

#### Card Reader and Gate

Function	<p><b>Gate and LED lamps (On/Off)</b></p> <p>Gate and a green LED lamp would turn on if a correct card is tapped on the card reader. Otherwise, only a red LED lamp and the buzzer would be turned on.</p> <p><b>Determination of Light Path</b></p> <p>The card reader system would send a signal to notify the lighting system to turn on an appropriate light path to guide the driver to the correct parking space.</p>
Structure	This module is constructed with a <b>card reader, two LED lamps, a buzzer, a servo motor and an Arduino UNO board</b>

Input	<b>Control signal</b> (Arduino UNO board )
Output	LED lamps On/Off, Gate On/Off

### Ultrasonic Sensor

Function	<b>Detection of the car</b> The sensor is installed on the roof of the car park to check the vacancy of the parking space and whether there is any unauthorized parking.
Structure	This module is constructed with <b>ultrasonic sensors and an Arduino UNO board.</b>
Input	<b>Control signal</b> (Arduino UNO board )
Output	<b>Data signal to the database</b> (The vacancy of the parking space)

### LCD monitor

Function	<b>Show vacancy</b> It is installed in the entrance of the car park to show the current number of parking space vacancy in the car park.
Structure	This module is constructed with <b>a LCD monitor and an Arduino UNO board.</b>
Input	<b>Data from database</b> (No. of vacancy) and <b>Control signal</b> (Arduino UNO board )
Output	The <b>display of current number of parking space vacancy</b> in the car park

## **Project Background**

In the recent years, smart city has become a more popular idea, thanks to the rapid development of Internet of Things (IOT) protocols. The idea of smart city is to take advantage of the technology and use innovative solutions to address issues in the city, enhancing competitiveness and sustainable development of the city [1]. Hong Kong, as an international city, needs to catch up the global trend to build a smart city. This is the fundamental reason for the project that it provides a possible solution to tackle the parking problem in a smart manner, which contributes a part in the development of a smart city.

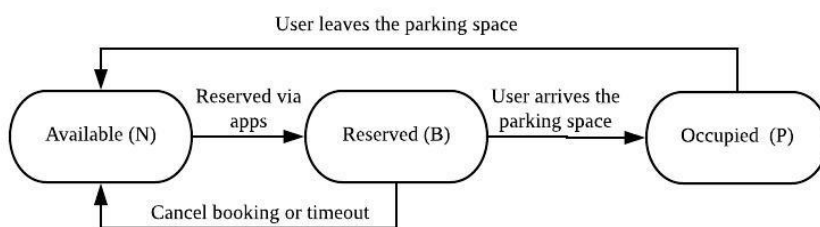
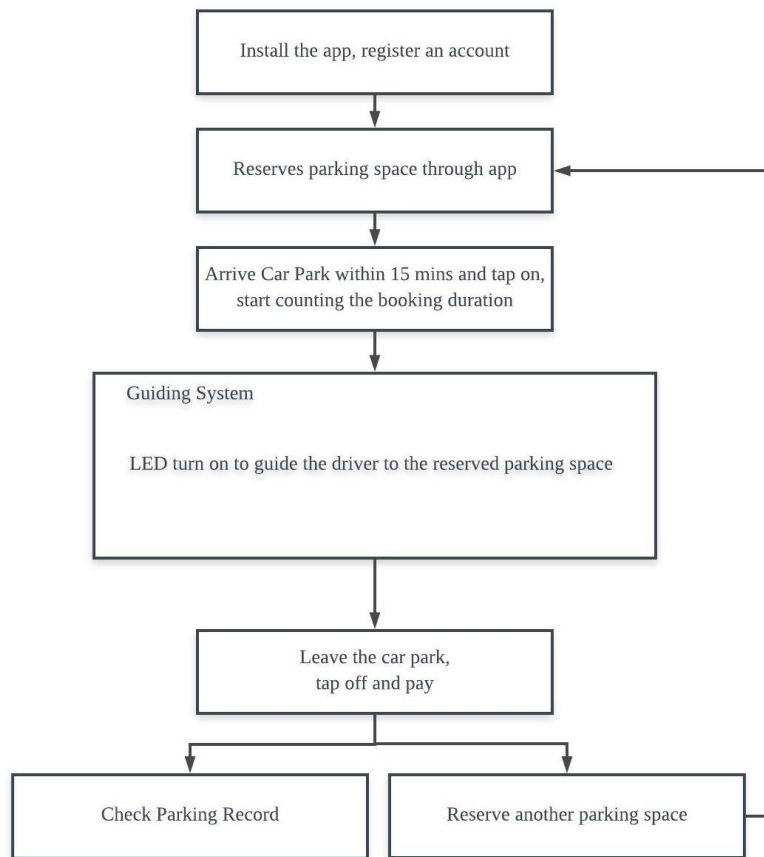
As we know, the population density of Hong Kong is extremely high. This makes little room for building a lot more car parks, which induces a problem of searching a parking space. It takes time and efforts in searching for a parking space, especially in the rush hour. According a document from Legislative Council, the amount of private cars had increased 45% while the parking spaces had only increased 9.5% from 2006 to 2016 [4]. It shows the parking demand in Hong Kong has kept rising.

Under this situation, it drives us to seek for solutions to deal with it. There are basically two focus in the project. The first one is to help the drivers to find a parking space. This aims at reducing the difficulty of searching a parking space. This idea is simple that the vacancy states in different car parks are collected, which enables the drivers to reserve an available parking space through the phone application. In this way, it can effectively save time for the drivers to search a parking space. Another focus is about the guiding system in a car park. Usually, it takes time for drivers to search a parking space in a large or limited parking space car park. The project proposes to build a guiding system with LED lights to guide the cars in a car park. This would be another way to improve the parking experience that the progress would be more efficient.

In short, this project aims at providing smart solutions to enhance the deficiency in the existing parking system.

## Project Overview and Milestones

### Project Work



The above block diagram shows the system implementation in our project “Smart Carpark”. Basically, the major purpose of the project is to facilitate the drivers to park. For this purpose, we have implemented an app with the main functions of reservation and record. The users could reserve a parking space and track the parking history in the App. To make these functions come true, a database is required so that information like vacancy of the parking space could be obtained in the App, which is especially important to the system. As a result, a number of



ultrasonic sensors would be installed on the roof of the model to check the vacancy of the parking space and whether there is any unauthorised parking. Plus, a LCD monitor that shows the current vacancy in the car park has been built. Furthermore, as the LCD monitor and ultrasonic sensors need to receive data from or send data to the database, Raspberry Pi has been used in our system serving as a server to communicate with the database. More than that, a card reader and gate system, and a lighting system to guide the drivers to the desired parking space have been constructed to further provide a more convenient parking experience. The details of each system would be discussed in the next part “Design Details”.

### **Project Budget and Equipments**

<b>Components</b>	<b>Cost (\$ HKD)</b>	<b>Components</b>	<b>Cost (\$ HKD)</b>
<b>Raspberry PI</b>	<b>291</b>	<b>LED indicator *10</b>	<b>120</b>
<b>Wifi model *1</b>	<b>15</b>	<b>Ultrasonic detector *6</b>	<b>28.5</b>
<b>Maped sticky notes</b>	<b>8</b>	<b>Wood Stick*2</b>	<b>120</b>
<b>Hard cardboard*2</b>	<b>12</b>	<b>Tape</b>	<b>4.5</b>
<b>Stationary</b>	<b>5</b>	<b>Total</b>	<b>604</b>

The total final budget meets the estimated budget. We have bought some items such raspberry pi and LED which we expected in the beginning. However, we has added two wood sticks in the list since we need to strengthen the model structure.

### **Project Milestones**

<b>Project Milestones of IDP -- Smart Parking</b>										
<b>Prototype Design Cycle</b>	<b>Task Owner</b>	<b>Week</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>
<b><u>R&amp;D</u></b>										
Material Preparation	Everyone									
Breadboard Design										
Design Specification										

<u>Prototype Design</u>										
App - Front-End	Law Kin Ping									
Login, Register, Menu										
Design of each page										
App - Back-End	Ho Yu Hin									
Login, Register, Database Design										
Getting Sensor Data from database										
Arduino Coding										
Sensors and Communication With database	Lee Chun Yuen									
Gate	Law Kin Ping									
Lighting Systemic	Lee Chun Yuen									
LCD monitor	Law Kin Ping									
<u>Complete Set</u>	Everyone									
Assembling										
Code Touch Up										
Circuit Touch Up										
<u>Complete Prototype Test</u>	Everyone									
Functional										
Performance										
Final Touch Up										

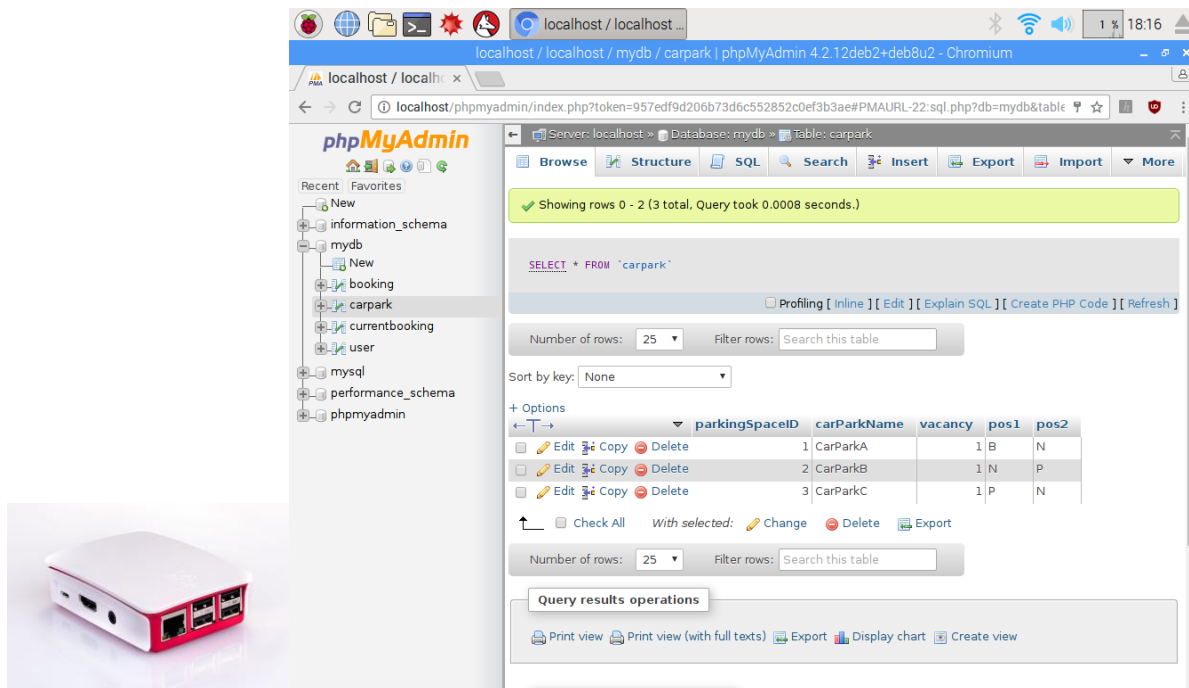
## Design details

### 1. Server and Database

#### 1.1 Raspberry Pi server

We use a Raspberry pi (Rpi) with Raspberry OS to create a Apache HTTP server. The reason of using Rpi is the fact that it consumes little energy comparing to a normal server. The computational power of the raspberry pi is obviously enough for our application. We use Apache

HTTP server because it is open source and free to use. A MySQL database is established in the Rpi to store user information, car park information and the booking status and record of each user. We use phpMyAdmin to management the mysql database as shown in below.



## 1.2 DDNS

It is usual to use Dynamic Domain Name System (DDNS) to map the local ip address of the raspberry pi to the domain name (e.g. idpparking.ddns.net) since the ip address is normally dynamic. However, this process involves port forwarding which allow external ip (user) to access the database through the default port 80 of the ip of raspberry pi, that we are not able to achieve since we have no control on the router setting in the lab. Therefore in the prototype, the app will directly connect to the database through ip address e.g.

jdbc:mysql://192.168.0.105/mydb where 192.168.0.15 is the internal ip address of the server and mydb is the name of database. The connection should work provided that the user device and the server are under the same subnet, such that they can directly exchange information to each other.

## 1.3 Connection

Due to the time limit on this project, we were unable to implement a better but time consuming

method to connect the apps with the MySQL database. The usual way is to use PHP to execute a command in external server to interact with the database. The main advantage is the username and password are not stored inside the apps, that is, if the hacker reverse-engineer our apps, he cannot gain unauthorized access to our database. Therefore, we should use PHP to implement the database connection for the security propose.

In our app, Java Database Connector (JDBC) is used to connect the app to the database. We simply create a class call ConnectionClass to setup the connection.

```
@SuppressWarnings("NewApi")
public Connection CONN() {
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
        .permitAll().build();
    StrictMode.setThreadPolicy(policy);
    Connection conn = null;
    String ConnURL = null;
    try {
        Class.forName(classss);
        conn = DriverManager.getConnection(url, user, password);
        conn = DriverManager.getConnection(ConnURL);
    } catch (SQLException se) {
        Log.e(tag: "ERROR", se.getMessage());
    } catch (ClassNotFoundException e) {
        Log.e(tag: "ERROR", e.getMessage());
    } catch (Exception e) {
        Log.e(tag: "ERROR", e.getMessage());
    }
    return conn;
}
```

For the execution of query to the MySQL database, we use AsyncTask method. This method is basically divided into three parts: onPreExecute, doInBackground and onPostExecute. As the name of the function suggest, they handle the action before the main task executed, the main task that perform in the background and the action after the execution of the main task respectively. I will use the Login page to illustrate.

### 1.3.1 AsyncTask

```
private class Dologin extends AsyncTask<String,String,String> {  
  
    String userName = etUserName.getText().toString();  
    String password = etPassword.getText().toString();  
  
    String z="Incorrect Password";  
    boolean isSuccess=false;  
  
    String user,pw, li;
```

There are three parameters for the AsyncTask, Params, Progress and Result. Params is the type of the parameters sent to the task upon execution; progress is the type of the progress units published during the background computation; result is the type of the result of the background computation (Source: Android developers). Here we set all the parameters be String because we manipulate String z to show the result.

### 1.3.2 onPreExecute

```
@Override  
protected void onPreExecute() {  
    progressDialog.setMessage("Loading...");  
    progressDialog.show();  
    super.onPreExecute();  
}
```

For onPreExecute, it is normally to use a progressDialog object that can display a message to user to inform that the app is connecting the server. For example, in the login task, we can set the message to be “loading...” and display it to the user.

### 1.3.3 doInBackground

```
@Override
protected String doInBackground(String... params) {
    if(userName.trim().equals("") || password.trim().equals(""))
        z = "Please enter all fields....";
    else
    {
        try {
            Connection con = connectionClass.CONN();
            if (con == null) {
                z = "Please check your internet connection";
            } else {
                String query=" select * from user where userName='"+userName+"' and password='"+password+"' ";
                Statement stmt = con.createStatement();
                ResultSet rs=stmt.executeQuery(query);
                while (rs.next())
                {
                    user = rs.getString(1);
                    pw =rs.getString(2);
                    li =rs.getString(3);
                    if(user.equals(userName)&& pw.equals(password))
                    {
                        isSuccess=true;
                        z = "Welcome!! "+userName;
                    }
                    else {
                        isSuccess = false;
                    }
                }
            }
        }
    }

    catch (Exception ex)
    {
        isSuccess = false;
        z = "Exceptions"+ex;
    }

    return z;
}
```

For doInBackground, this is the place where the main task is performed. Connection to the database, executing query, retrieving information, utilization of the information such as displaying them, will be executed in this part. Firstly, we will check if the user enabled internet

connection. If so, the app will try to connect to the database through the connection class as mentioned above. If the connection is success, we will further generate a statement of query and send it to the server. After receiving the response, we can read information from the result set. If needed, some information received in this part will be pass to the onPostExecute.

### 1.3.4 onPostExecute

```
@Override
protected void onPostExecute(String s) {
    Toast.makeText(getApplicationContext(), text: ""+z, Toast.LENGTH_LONG).show();
    if(isSuccess) {
        Intent intent = new Intent( packageContext: LoginActivity.this, MainPage.class);
        Bundle bundle = new Bundle();
        bundle.putString("userName", userName);
        bundle.putString("licenseNumber", li);
        intent.putExtras(bundle);
        startActivity(intent);
    }
    progressDialog.dismiss();
}
```

For the onPostExecute, it is mainly used to handle the change in activity or fragment.

For some pages such as carParkSelection, the data need to be displayed in the listView.

```
@Override
protected void onPostExecute(String s) {
    List<Map<String, Object>> items = new ArrayList<>();
    Map<String, Object> title = new HashMap<>();

    // 1st row: Add the title
    title.put( k: "Name", v: "Name");
    title.put( k: "Distance", v: "Distance");
    title.put( k: "Vacancy", v: "Vacancy");
    items.add(title);

    // nth row: data
    for (int i=0; i < carParkInfo.size()-1; i++){
        Map<String, Object> item = new HashMap<>();
        item.put( k: "Name", carParkInfo.get(i).get(0));
        item.put( k: "Distance", carParkInfo.get(i).get(1));
        item.put( k: "Vacancy", carParkInfo.get(i).get(2));
        items.add(item);
    }
}
```

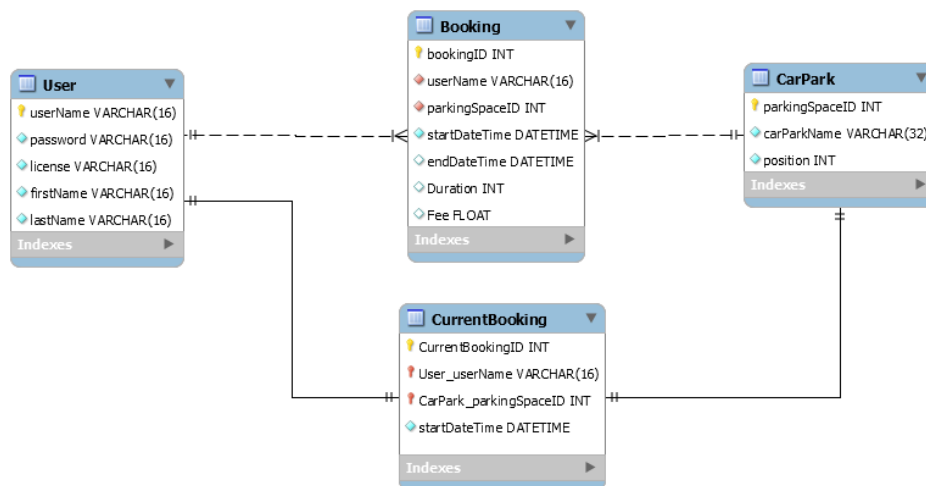
For each set of data, we will use HashMap ( Hash Table) to represent the data. HashMap is a specific data structure which implements a list, with the key mapped to the value by hash function. For instance, we create three keys: Name, Distance and Vacancy. They will be mapped to the data. Since each set of data is represented in a hashmap, we then create a list to store all maps.

```
// Load data to the listview
SimpleAdapter adapter = new SimpleAdapter(
    getActivity(),
    items,
    R.layout.style_listview,
    new String[]{"Name", "Distance", "Vacancy"},
    new int[]{R.id.tvName, R.id.tvDistance, R.id.tvVacancy}
);

listview.setAdapter(adapter);
```

To display the data in the listView, we can pass the list of hashmap into SimpleAdapter, which is the easiest way to handle listView.

## 1.4 Database Design

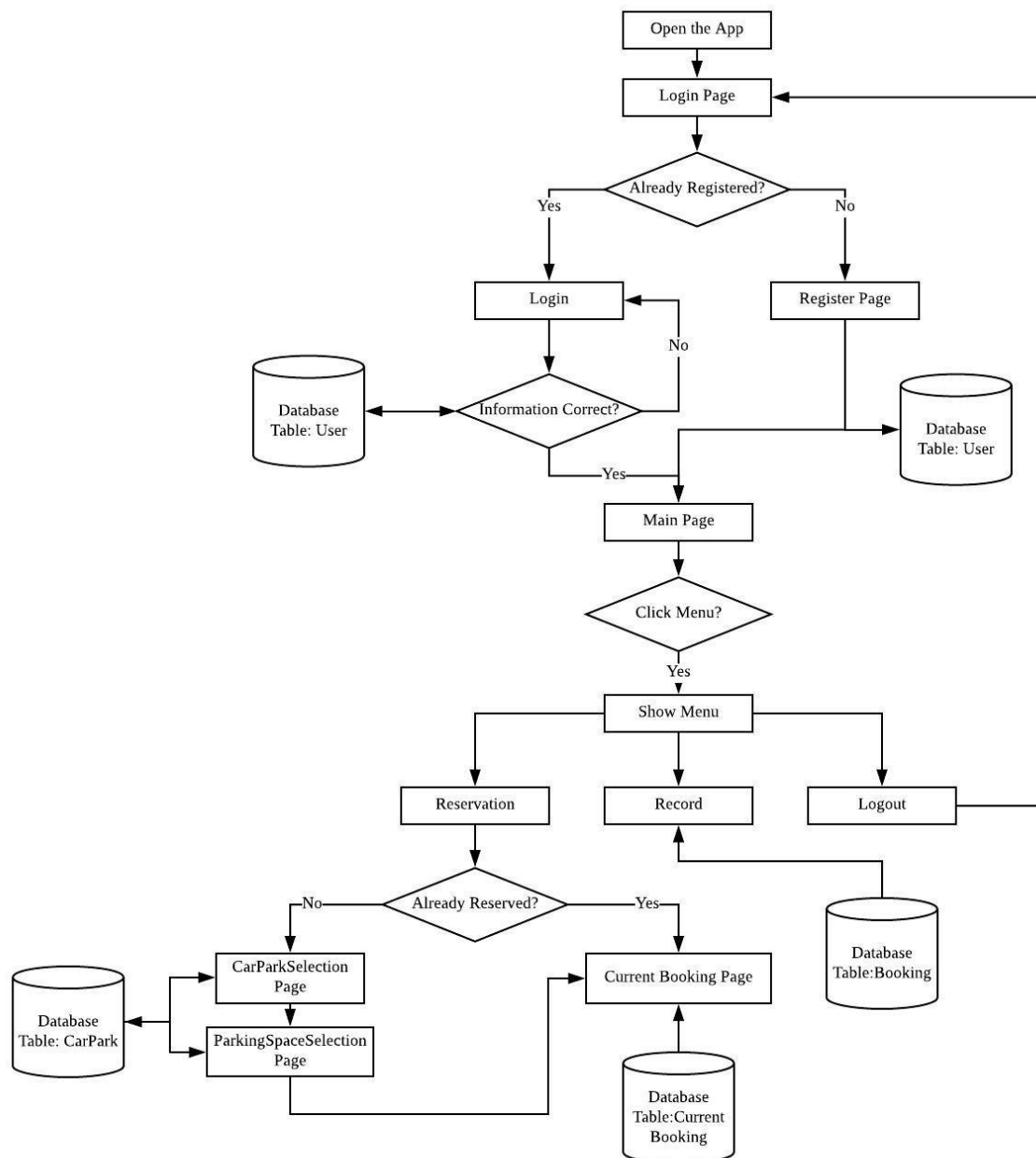


There are four tables in the database. User table stores the user information; CarPark table stores the name and vacancy of the car park; CurrentBooking table stores the information about the current booking, including the name of the user who book the parking space, which parking space the user is booking and the booking time; Booking table stores the booking record.

## 2. Phone Application

The Block Diagram of the App:

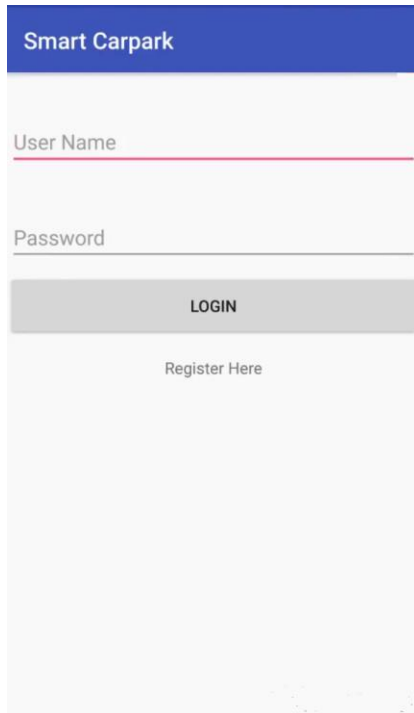




As shown in the above block diagram, the major function of the phone application is to implement the reservation and record functions for parking. For “Record” function, it is just a function that allows the users to track the history of parking. For “Reservation” function, it is implemented by allowing the users to choose the correct car park and the parking space in it. In case the user wants to cancel the booking, the App would allow him/her to do so. Also, there would be a timeout feature that the reservation would automatically cancel if the user does not show up after the reservation time limit. The details of each page would be presented in the

following.

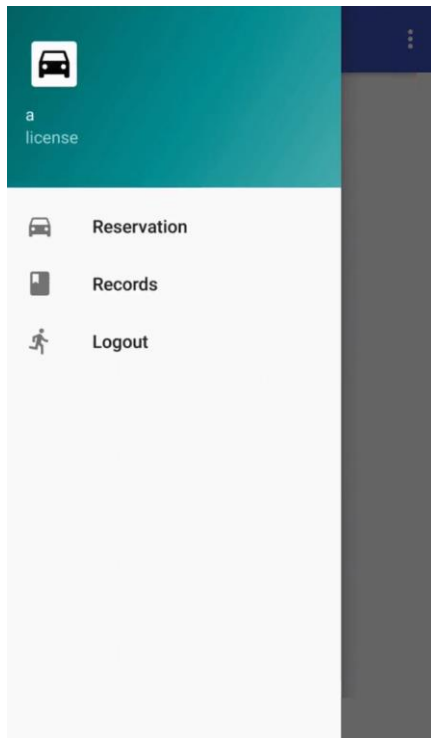
## 2.1 Login page/Register:



The image shows a mobile application interface for a 'Smart Carpark'. At the top is a blue header with the text 'Smart Carpark' in white. Below the header is a light gray background. There are two input fields: 'User Name' with a pink underline and 'Password' with a gray underline. Below these fields is a gray button labeled 'LOGIN'. At the bottom of the form is a link that says 'Register Here'.

“Login” and “Register” functions are implemented to allow the users to register and login in the phone application. After the login, the users can use the functions in the Main Page.

## 2.2 Main Page:



In the main page, there are three events.

1. Click the Reservation button
2. Click the Record button
3. Click the Logout button

For the reservation, the system will first determine whether the user booked a parking space or not after the user clicking the reservation button. If yes, the app will enter the “Current Booking” page, else enter the “Car Park Selection” page. After selecting a desired car park, the app enter the “Parking Space Selection” page where the user is required to select the desired parking slot. After selecting the parking slot, the app enter the “Current Booking” page.

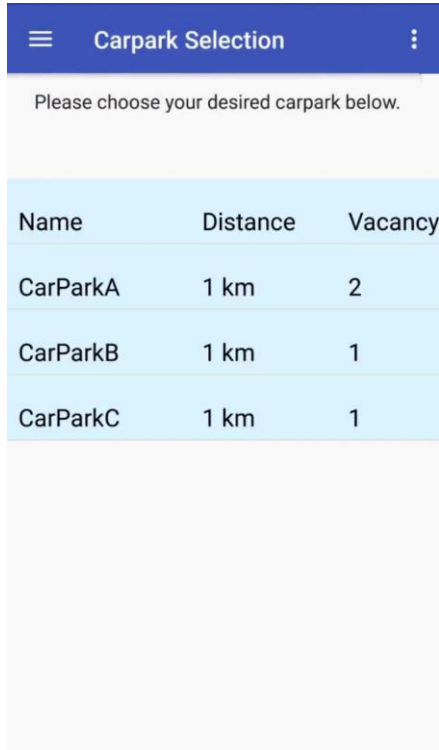
For the record, the app will display the past parking record. Logout is simply logout the system and return to the login page.

### **2.2.1 Reservation:**

1. After clicking the Reservation button, the app will then retrieve the information from the “current booking” table from the database to determine whether the current user has booked a parking space. This is done by checking whether there is a current booking item with the

userName equals the userName of the user. If the user has booked a parking space, the fragment will be changed to CurrentBooking. If the the database return null, the fragment will be changed to CarParkSelection.

#### 2.2.1.1 CarPark Selection:



Name	Distance	Vacancy
CarParkA	1 km	2
CarParkB	1 km	1
CarParkC	1 km	1

Here we assume that we have entered the information of the car park including the name, the location and the capacity of the car park into the “car park” table in the database.

In the project, we name our car park as CarParkA and set the capacity to 10. However, we only set the maximum vacancy to 2. After entering CarParkSelection fragment, the app will retrieve the the details of the car park. Three items including the name of car park, the distance of the car park from the user and the current vacancy would be shown in the list view.

#### 2.2.1.2 ParkingSpaceSelection:

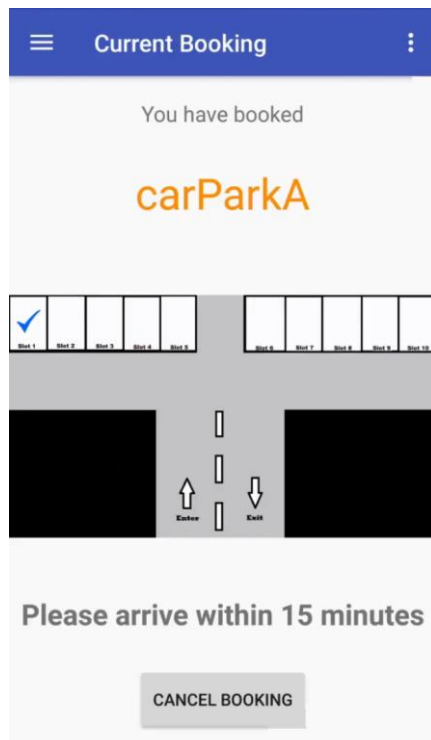
In this page, the floor plan of the selected car park and the status of parking slot would be shown. For simplicity, we only show the status or allow the selection of the parking slot 1 and 10. The parking space that is available and unavailable for booking will be represented by green button and red button respectively. Initially, all the buttons are set to be invisible. After retrieving the

current status of the parking space, the appropriate buttons become visible accordingly. For example, the parking slot 1 is booked/parked and the slot 10 is neither booked nor parked. A red button is shown on the slot 1 and a green button is shown on the slot 10.

### 2.2.2 Record:

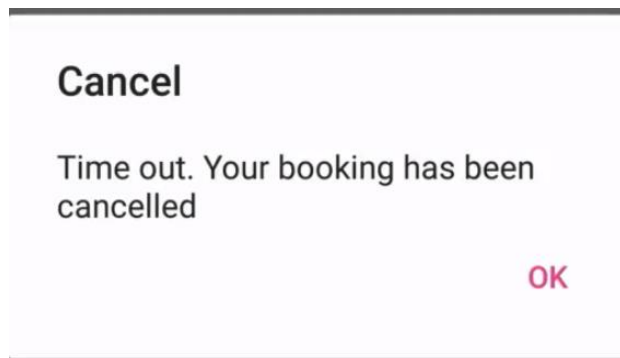
In this page, the record of the booking will be shown in the list view. The app will retrieve the record from the database. Four items would be shown, including the name of the car park, the start time and end time and the fee of parking.

### 2.2.3 Current booking/Time out/Cancel Booking



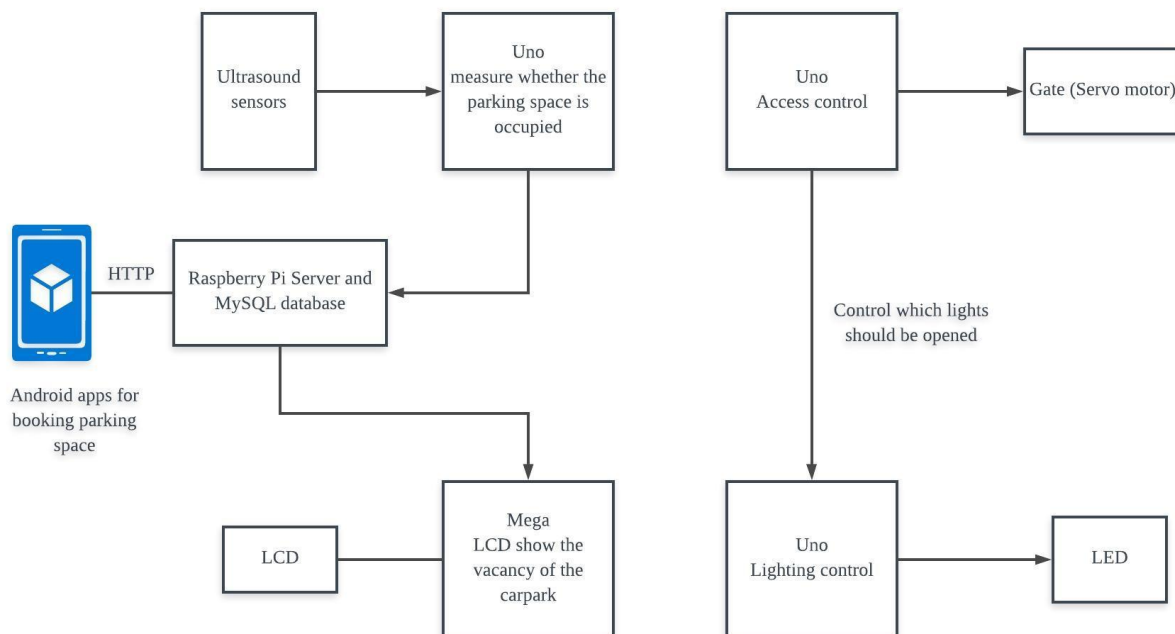
This page shows which parking space the user has booked. In this example, the user booked parking slot 1 of carParkA. The user has at most 15 mins to arrive the car park. If the count down timer reaches 0 minutes, the booking would be canceled automatically.

The user can click the “CANCEL BOOKING” button to cancel the booking. The state of that parking space will change from B to N and the vacancy would increase by one.



### 3. The Card Park Model

The running flow chart in the car park model:



The flow chart demonstrates the simple idea of how we implemented the car park model. In the following, it would talk about each component of the card park model in details.

#### 3.1 Card Reader and Gate

As mentioned, the payment process at the entrance of the car park is modelled by the card reader system. This system consists of a servo motor, a green LED, a red LED, a buzzer, and the RFID MFRC522 card reader and card. Basically, when a right card is tapped on the card reader, the green light, the buzzer and the gate would turn on at the same time. The allows the driver to enter

the car park. Conversely, the buzzer and the red light would turn on to notify the driver that there is something wrong with the card if the card tapping on the card reader is not the desired one. Moreover, this would be worth mentioning the reason why the RFID card reader and card are chosen. The fundamental reason is that RFID is a mature technology with high reliability. This implies the RFID card reader and card would be widely available in the market with acceptable price. This makes RFID very suitable for modeling or even the real application of the card reader system.

With a basic concept of how the system works, it then would take a look how we control the system explained with part of Arduino code in the following.

After calling necessary and defining the variables, some important features can be done.

```
void loop()
{
    // Look for new cards
    if ( ! mfrc522.PICC_IsNewCardPresent())
    {
        return;
    }
    // Select one of the cards
    if ( ! mfrc522.PICC_ReadCardSerial())
    {
        return;
    }
    //Show UID on serial monitor
    Serial.print("UID tag :");
    String content= "";
    byte letter;
    for (byte i = 0; i < mfrc522.uid.size; i++)
    {
        Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(mfrc522.uid.uidByte[i], HEX);
        content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
        content.concat(String(mfrc522.uid.uidByte[i], HEX));
    }
    Serial.println();
    Serial.print("Message : ");
    content.toUpperCase();
```

As every card has its own number, these codes enable Arduino to read the card number after the card is tapped on the card reader. The card number would be shown on serial monitor for further application.

```
if (content.substring(1) == "34 68 79 89" || content.substring(1) == "FA 3A 12 32")
{

    Serial.println("Authorized access");
    Serial.println();
    delay(500);
    digitalWrite(LED_G, HIGH);
    tone(BUZZER, 500);
    delay(300);
    noTone(BUZZER);
    myServo.write(90);
    delay(3000);
    myServo.write(0);
    digitalWrite(LED_G, LOW);
```

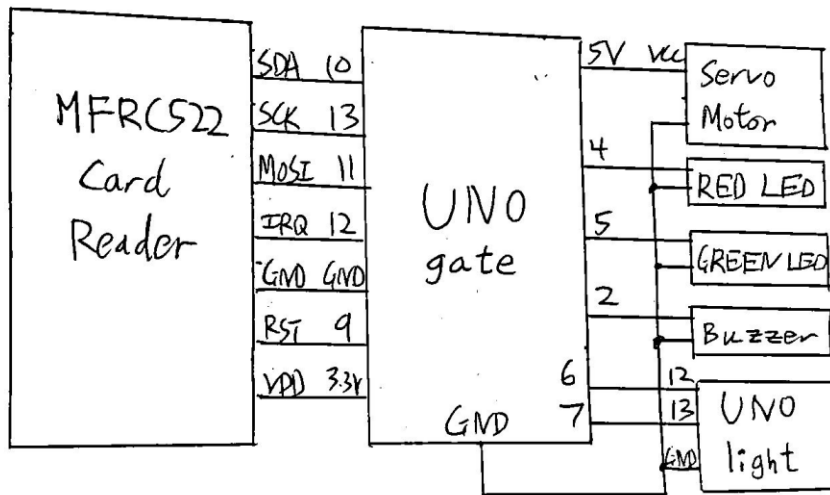
If the card number can be obtained, it could give access to the authorized card. In our system, the cards with card number “34 68 79 89” and “FA 3A 12 32” are authorized so that the green LED and buzzer would turn on after they are tapped on the card reader. The buzzer would just turn on for 0.3 second. After that, it also enables the gate to open by controlling the servo motor to turn 90 degrees upward for 3 seconds. Afterward, the gate would return to the original position controlled by the servo motor, and the green LED would also turn off.

```
else {
    Serial.println(" Access denied");
    digitalWrite(LED_R, HIGH);
    tone(BUZZER, 300);
    delay(1000);
    digitalWrite(LED_R, LOW);
    noTone(BUZZER);
}
```

This shows what happens when a wrong card is tapped on the card reader. It would turn on the buzzer and the red LED for a second and turn them off.



The following schematic diagram shows the connection of the card reader and gate system.



### 3.2 Lighting system

We use LED light to guide the driver to the corresponding parking space. Each of the LED installed on the roof of the model to provide light on the road. After the gate opened, the light which is installed on the parking space will open. Also, the light on the path turn on one by one to guide the driver to the parking space that he booked. When the car park in the right space, the light on the parking space will be turned off.

With a basic concept of how the system works, it would take a look how we control the system explained with part of Arduino code in the following.

```
int n=1500; // delay time
void setup() {
  pinMode(led1, OUTPUT);pinMode(led2, OUTPUT);pinMode(led3, OUTPUT);pinMode(led4, OUTPUT);pinMode(led5, OUTPUT);
  pinMode(led6, OUTPUT);pinMode(led7, OUTPUT);pinMode(led8, OUTPUT);pinMode(led9, OUTPUT);pinMode(led10, OUTPUT);
  pinMode(11, INPUT);pinMode(12, INPUT);

  Serial.begin(9600);
}
```

In the beginning, we set n as the delay time (1.5 second). In “void setup()”, we set the arduino mega pin2 to pin11 as the output in order to provide the power to each LED light. Also, we set pin 11 and 12 as the input which will receive the signal to determine the correct light path.

```

void loop() {

    //Serial.println(digitalRead(11));
    //Serial.println(digitalRead(12));

    // From Car Reader, pin6->11, pin7->12
    if(digitalRead(11)==HIGH)
        pathA();
    else if(digitalRead(12)==HIGH)
        pathB();
}

```

In the “void loop()”, it gets the signal from another arduino uno board. When the voltage of pin 11 or 12 is HIGH, the programme will run “pathA()” or “pathB()” respectively.

```

9      10
2 3 4 5 6 7 8
1

```

It is the light path of the model. 1 is the first light when driver comes from the gate. 9 and 10 are the lights in parking space.

```

void pathB(){
    analogWrite(10, 255);
    analogWrite(1, 255);
    delay(n);
}

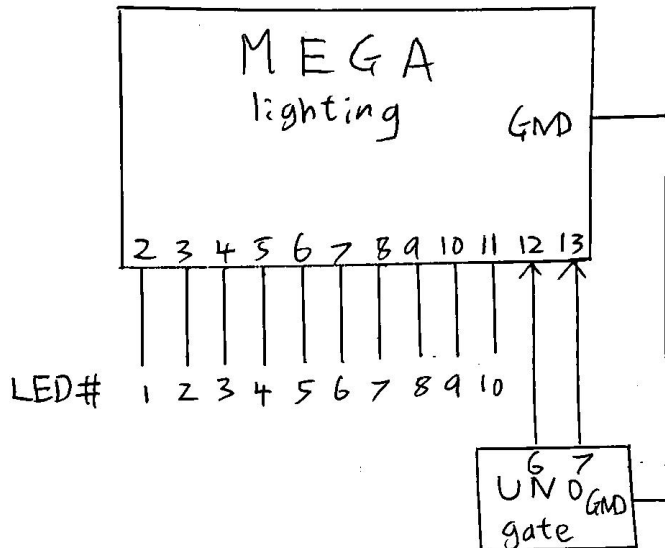
void pathA(){
    analogWrite(9, 255);
    analogWrite(1, 255);
    delay(n);
    analogWrite(1, 0);
    analogWrite(4, 255);
    delay(n);
    analogWrite(4, 0);
    analogWrite(5, 255);
    delay(n);
    analogWrite(5, 0);
    analogWrite(6, 255);
    delay(n);
    analogWrite(6, 0);
    analogWrite(8, 255);
    delay(n);
    analogWrite(8, 0);
    analogWrite(7, 255);
    delay(n);
    analogWrite(7, 0);
    analogWrite(9, 255);
    delay(3000);
    analogWrite(9, 0);
    delay(3000);
    analogWrite(10, 0);
}

```

pathA() and pathB() determine the light path showing on the road. For example, when pin 11 is HIGH, the arduino will run pathA(). path() will turn on the LED9 non-stop and show the correct

parking space to the driver. After that, the first LED (led1) will turn on and turn off after 1.5 second. When the light achieves to led9, there is a time delay 3.0 second for the driver parking in the space then led9 will be turned off. The process of pathB() is similar as pathA() but the final destination is led10.

The following schematic diagram shows the connection of the LED lights.



### 3.3 Ultrasonic Sensors and Communication with the server

We use ultrasound sensors to sense whether there is a car that has parked in to the parking space. The ultrasound sensor is installed on the roof of the model to measure the distance from the ground. When there is no car occupying the parking space, the distance would be a constant, which is around 33 cm in our model. When a car occupies the parking space, the measured distance would decrease, which is around 31 cm in the model. If the threshold value is set in between 31 cm to 33 cm, it could determine whether there is a car in the parking space. After setting the appropriate threshold, which is 32 cm in our model, we can get the currently measured parking space status. There are two states N stands for “NO car is occupying the space” and P stands for “someone is PARKING”. For example, the status of the left parking space and that of right parking space would be sent to the raspberry in the format of “N,P” for further process. “N,P” means the left parking space is not occupied and the right parking space is

occupied. The raspberry pi gets the current parking status from the Arduino and it also gets the parking space status from the database for comparison.

In the database, there is one more parking status other than “N” and “P”. It has a “B” status, which stands for “the parking space is BOOKED but the driver haven’t arrived yet”.

In the normal situation, the sequence of changing states should be N->B->P if someone reserves the parking space through the App and enters the car park on time. In order to achieve it, the parking space status in the database will change to P only when the data from the Arduino is P and data from the database is B. When a car is occupying a non-booked parking space (i.e. data from Arduino = P while data from database = N), there will be a warning showing on the screen of the server, indicating the security guard that an unauthorised parking has occurred.

With a basic concept of how the system works, it then would take a look how we control the system explained with part of Arduino code in the following.

```
void setup() {  
  
    pinMode(trigL, OUTPUT);  
    pinMode(trigR, OUTPUT); char ReadReceived_Char;  
    pinMode(echoL, INPUT);   const int trigL = 8;  
    pinMode(echoR, INPUT);   const int echoL = 9;  
    Serial.begin(9600);       const int trigR = 6;  
    }                         const int echoR = 7;  
                             long durationL = 0;  
void loop() {               long durationR = 0;  
    getData();              float distanceL = 0;  
                             float distanceR = 0;  
    }                       float threshold = 30;
```

In “void setup()”, we set the corresponding pin as input and output. Also, we set the data rate (baud) for serial data transmission as 9600. Then the program will loop “getData()”.

```

void getData(){
  // Trigger for Left
  digitalWrite(trigL,1);
  delayMicroseconds(15);
  digitalWrite(trigL,0);
  // Echo for Left
  durationL = pulseIn(echoL, HIGH);

  // Trigger for Right
  digitalWrite(trigR,1);
  delayMicroseconds(15);
  digitalWrite(trigR,0);
  // Echo for Right
  durationR = pulseIn(echoR, HIGH);

```

In “getData()”, “trigL” will turn on for sending the signal outside for 15 microseconds (us) and turn off. After that “echoL” will turn on for detecting the signal which sending from “trigL” before and record the time to “durationL” by function “pulseIn”.

```

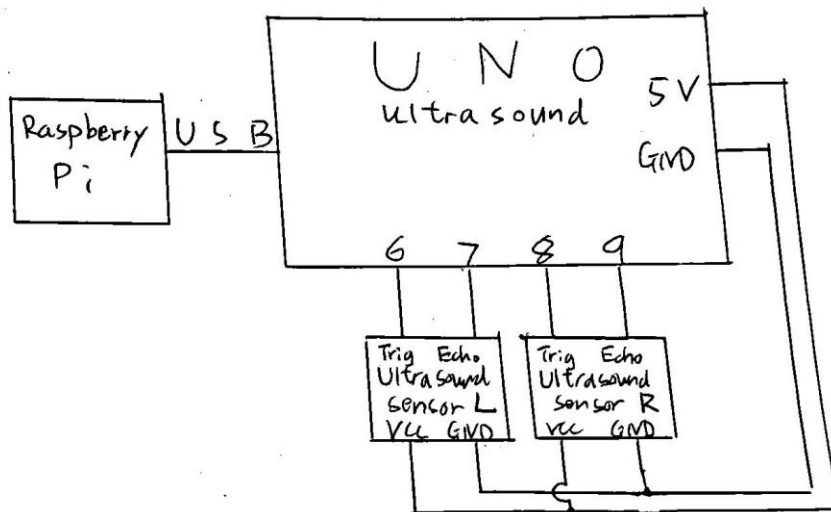
// Calculate the distance
distanceL = durationL/58 ;
distanceR = durationR/58 ;
String L,R,str;
if(distanceL>threshold)
  L="N";
else
  L="P";
if(distanceR>threshold)    // Reset parameters
  R="N";                  durationL = 0; distanceL = 0;
else                      durationR = 0; distanceR = 0;
  R="P";
str= L+", "+R;             // Set the sampling interval
Serial.println(str);      delay(2000);

```

If there is a car parking in the space, the distance between object and sensor will be decreased. Hence, when the distanceL(R) is larger than “threshold” which means the distance between the sensor and the road, there is no car on the space and program will send “N” to the database. Otherwise it will send “P”.

After sending the data to database, “durationL(R)” and “distanceL(R)” will be reset and the program will run again in the same procedure.

The following schematic diagram shows the connection of the ultrasonic sensor.



### 3.4 LCD monitor

As shown on the above picture, the LCD monitor shows the number of parking space vacancy in the car park. It could communicate with the database through Raspberry Pi. When anyone reserves a parking space in the car park through the App, the number of vacancy would reduce by one immediately actualized by the communication using USB. If the booking of the parking space is cancelled manually or automatically, the vacancy would update accordingly. Also, when the car leaves the parking space, the number of vacancy shown on the LCD monitor would update after a while. This process is done by the data received by the ultrasonic sensors. The sensors would check the availability of the parking space as mentioned in the previous part. The data would then be sent to the database through Raspberry Pi. As the LCD monitor could communicate with the database, it could update the number of vacancy after the car leaves the car park.

With a basic concept of how the system works, it would then take a look how we control the system explained with part of Arduino code in the following.

```

#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 10, 9, 8, 7);

String incomingByte;
void setup()
{
  lcd.begin(16,2);
  lcd.clear();
  delay(1000);
  Serial.begin(9600);
}

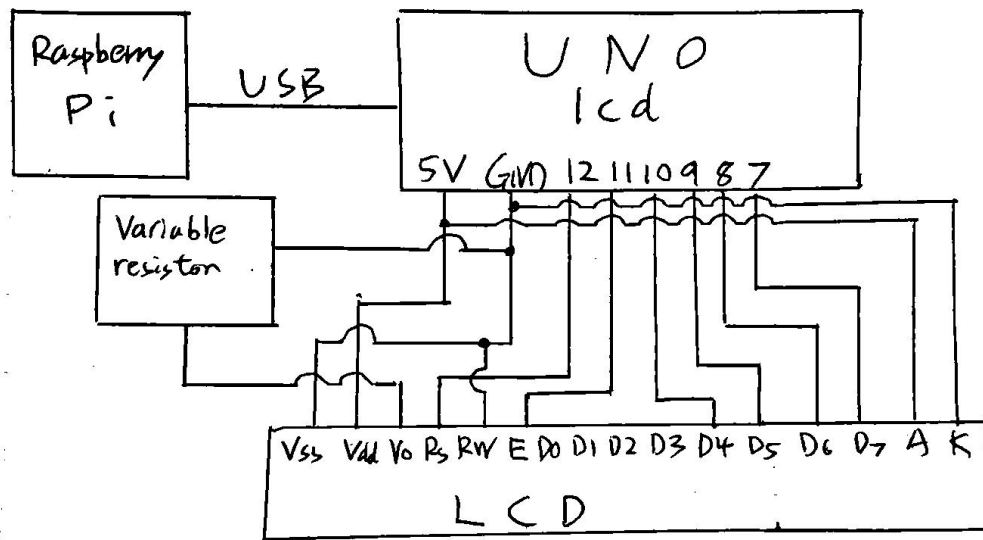
void loop ()
{
  if (Serial.available() > 0) {
    incomingByte=Serial.readString();
    int length = incomingByte.length();
    incomingByte = incomingByte.substring(0,length-2);
    Serial.println(incomingByte);
  }
  lcd.setCursor(0, 0) ;
  String show = "Vacancy:" +incomingByte;
  lcd.print(show);
  lcd.setCursor(0, 1) ;
  lcd.print("");
  delay(500);
}

```

In “void setup()”, it is just to set the LCD monitor clear initially and the starting location of the string.

In “void loop”, it gets the number of vacancy from the database from time to time. It then shows the updated number of vacancy on the LCD monitor in the format of “Vacancy: (Number of vacancy)”. As the LCD monitor would retrieve data from database from time to time, it could ensure the number of vacancy displayed on the LCD monitor would show the number without much delay.

The following schematic diagram shows the connection of the LCD monitor.



For the LCD monitor, we purposely choose LCD1602 simply because of the size suits the installation on top of the roof at the entrance. It also consists of a potentiometer, which serves to adjust the brightness of the string displayed by the LCD monitor.

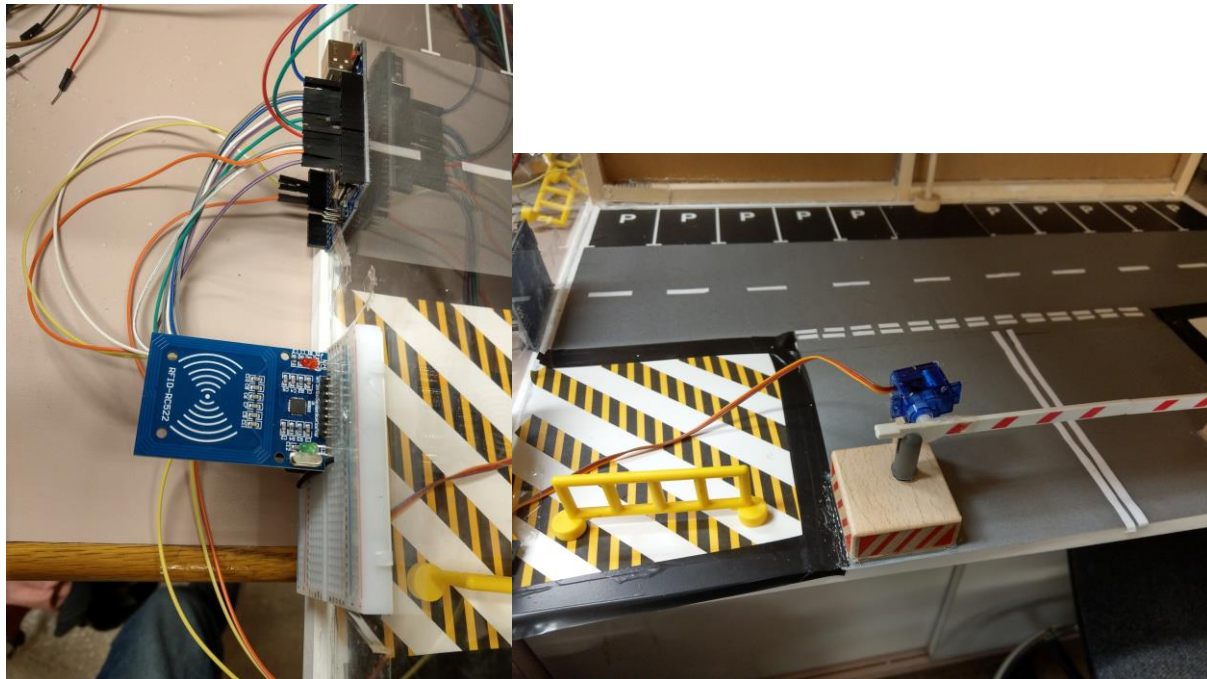
## Prototype and Performance

### 1.2. Server, database and phone application

The prototype is shown in detail in the previous section. The performance of the server and database are satisfactory since the computing power of the Raspberry Pi is sufficient for building the prototype. We found that the speed of the app is seriously influenced by the speed of the wifi. If we choose the wifi with lower throughput, the user would feel obvious lag when the app need to retrieve or update information in the database in the server. If we want to implement the system in a larger scale, it is crucial to enhance the speed of the network, so as to enhance the user experience.

## 3. Car Reader and Gate System



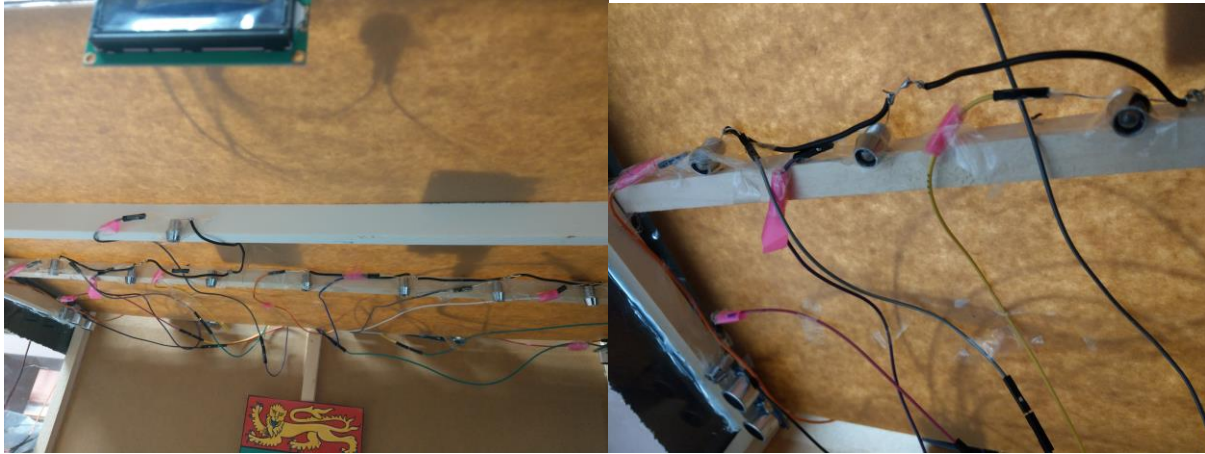


For the card reader, it is placed on the left side of the wall. This arrangement is to ensure that the Arduino UNO or breadboard would not lead to negative impact to the appearance of the model. For the sake of demonstration, the card reader is closed to the entrance of the car park model so that it is convenient for demonstration while it could also keep the model neat and tidy. Also, the buzzer and LED lights are placed neatly on the breadboard to improve the appearance.

For the gate, it is implemented by attaching a servo motor to the gate so that the servo motor movement would drive the gate to open or close. The difficulty of it is that it needs to ensure the servo motor could be stucked well to the gate. Otherwise, the gate would not follow the movement of the servo motor. This requires a careful treatment to make sure they are well attached.

In short, the system could run efficiently, thanks to the high reliability of the RFID card reader as mentioned. The system could respond quickly when the card is tapped on the card reader.

#### **4. Lighting System**



All of the LED light placed on the roof of model. It can show the clear light path on the road and guide the driver. For the good demonstration, we put the lights under the roof which is 33cm higher than the ground. We grouped the common ground of the LED to reduce the total number of wire. Also, we stuck the wire on the roof by tape in order to improve the appearance.

The LED light sometimes turned on and off unexpectedly. We finally found that the problem is bad connection. The ground of LED light was easily disconnected from the wire when we installed another components. The solution is that we added a little bit plastic tape in the front of the wire and they were not disconnected easily.

For installing the LED on the roof, we have two methods which are using the tape or thermal glue. We finally chose tape because the light path can be adjusted and more flexible compared to using glue. We have tried several times to fix the position of LED at last. Hence, using thermal glue will not able to adjust the location and direction of LED light.

For the performance, LED light can show the clear light path for the driver. Also, the time delay function is suitable for driver following the light path in enough time.

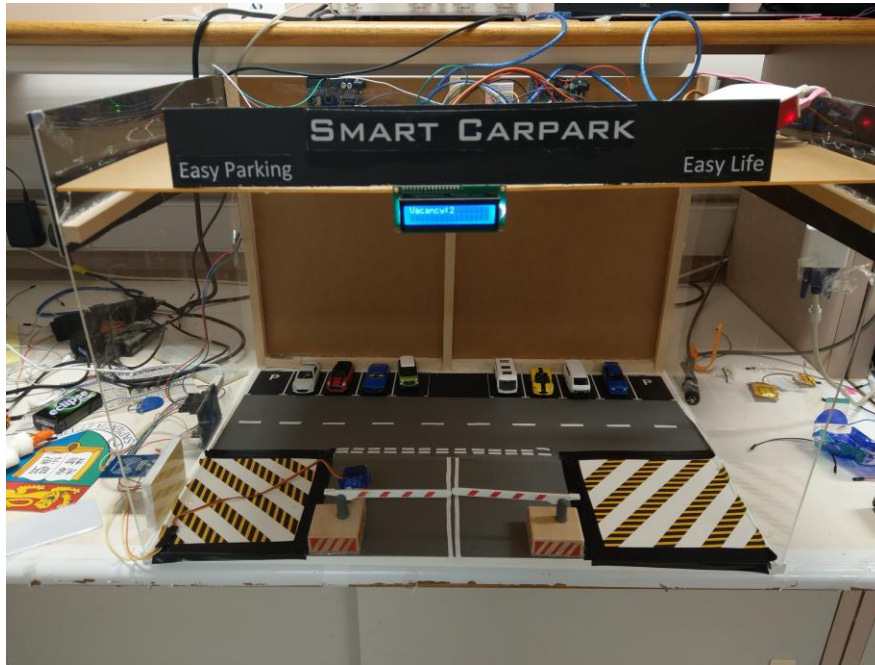
## 5. Ultrasonic Sensors and Communication with Database



The two ultrasonic sensors are installed on the top of the left and right parking space. It will detect the signal that is the time difference travelling between the object and sensor then send it back to the database. Hence, we can calculate the distance between them. The sensor must install directly on the top of the specific parking space and parallel to it. The reason is that the sensor consists of trigger (output signal) and echo (receive the signal from trigger) function. If it is not perfectly installed, the “echo” cannot receive the correct signal sending from “trigger” then cannot get the right travelling time and distance. We first used the tape to fix the position on the roof. However, the sensor is slightly heavy so we finally replaced it as thermal glue.

For the performance, the signal is clear and instantly communicating with the database.

## 6. LCD monitor



The LCD monitor is installed on top of the model at the entrance point. It is to show the current number of parking space vacancy in the car park, with a potentiometer to adjust the brightness of the LCD monitor. There are two fundamental reasons for the installation arrangement. The major one is that it is eye-catching and better for demonstration. Another one is because of the appearance consideration. The breadboard and Arduino UNO board could be placed on top of the model, which could be covered by the cardboard. This makes the appearance of the model more neatly, which turns out to be conducive to demonstrate the model.

For the performance, the string displayed on the LCD monitor is bright and clear. However, it could delay for a while to update the current number of parking vacancy in the car park. This is mainly attributed to the limited speed of the transferring data from the database. Fortunately, this is still bearable as the time delay is just several seconds. This makes a little impact on the model demonstration.

## Discussion and Future Work

In the project, it allowed us to apply and integrate the previous knowledge. There are several important lessons that we have learned in the project.

The major one is about the planning. It would save a lot of time if there is a well planning. The planning is not just about bring out an innovative idea, but thinking how the system could be implemented. At the early stage in the project, we had identified what are required in the system implementation, and had checked whether those components were available and easily obtained in the market. With this process, it turned out to save us a lot of time in the stage of prototyping as the materials we bought suited quite well for the implementation of the model.

Aside from the planning, this would be extremely important to ensure the system could be integrated with each other. Thanks to the planning work before starting to build the model, there would not be a fatal issue. Still, there were a lot of problems. Even each component works well on its own, it may not work really well in the system integration process. For example, the threshold value of the data received from ultrasonic sensors was set by trials and errors. It had to ensure that the ultrasonic sensor obtains an accurate data for the parking space vacancy determination without compromising the appearance of the model. This was a challenge to us that how we should strike a balance between the model appearance, the functionality and the reliability.

Therefore, if there is any chance to carry out similar project in the future, the first thing which should be put on top of the list would be the planning process. But another thing comes along with planning should be time management. It always has to reserve a room for the adjustment in integrating each component. Otherwise, it could be painful in the system integration. It must manage the schedule better so that it could have sufficient time for the process of system integration, such as trials and errors.

Fortunately, the core functions that we want to demonstrate in the App and the car park model could be implemented. However, the project was finished in a rush manner. This could remind us to pay more attention to the time, and perhaps certain parts in the project could be implemented in a more efficient and better manner.

## **Reference**

1. What is a “Smart City”?.(2015). Retrieved March 28, 2018, from <https://www.legco.gov.hk/research-publications/english/essentials-1415ise08-what->

[is-a-smart-city.htm](#)

2. **5 Parking Lot Management Problems And Their Solutions.** (2018, February 14). Retrieved March 28, 2018, from <https://www.trakaid.com/5-parking-lot-management-problems-solutions/>
3. **Background Information: Smart Parking: A System that Could ...** (2015). Retrieved March 28, 2018, from <https://www.bing.com/cr?IG=BDAA75677630434B8C100639B151D924&CID=3414BD3E49B8667C09C0B6FE481767CF&rd=1&h=C6qhkMttEqCszL7M8y5jXT8jw0hpeMIBPhssiV0PJyA&v=1&r=https://www.siemens.com/press/pool/de/events/2015/corporate/2015-09-iaa/background-smart-parking-e.pdf&p=DevEx,5068.1>
4. **LCQ17: Parking spaces.** (n.d.). Retrieved March 28, 2018, from <http://www.info.gov.hk/gia/general/201711/22/P2017112200307.htm?fontSize=1>  
Design details