Reg no: 510421104028

Name: DIVYA.A

Phase-5 Document submission

Project: *SMART PARKING*

ABSTRACT

This project's goal, "SMART PARKING SYSTEM USING IOT," is to give car owners access to free parking spaces while reducing traffic. Cities with rapid urbanization have problems. This issue can be solved by utilizing RFID technology along with an early parking slot booking option and a contemporary smart parking website. To regulate every aspect of the smart parking system's operation, an ESP WROOM 32 microcontroller is employed. Infra-red, the use of sensors allows for the determination of whether a vehicle is in or out of the slot. We can increase revenue for the organization by installing our smart parking system in important cities. For the end users, it's incredibly simple to keep track of all parking activity.

Introduction to Smart Parking:

- Smart parking is a modern technological solution aimed at revolutionizing the way we approach parking in urban and suburban environments.
- Smart parking leverages a combination of advanced technologies such as sensors, data analytics, mobile apps, and automation to provide a more efficient and user-friendly parking experience.
- In a smart parking system, sensors are installed in parking spaces to monitor their availability in real-time.
- In this age of increasing urbanization, where efficient use of space and resources is paramount, smart parking is a critical component of the smart city concept.

Objectives:

The Smart Parking System aims to revolutionize urban parking by leveraging IoT technology to provide real-time parking availability information to drivers. The objectives are as follows:

- Provide drivers with real-time information on available parking spaces.
- Optimize parking space utilization and reduce congestion.
- Enhance user experience through a user-friendly mobile application.
- Utilize Raspberry Pi for data processing and integration with IoT sensors.
- In all phases we discussed about the senors,raspberry pi integration, mobile app development using python and so on.

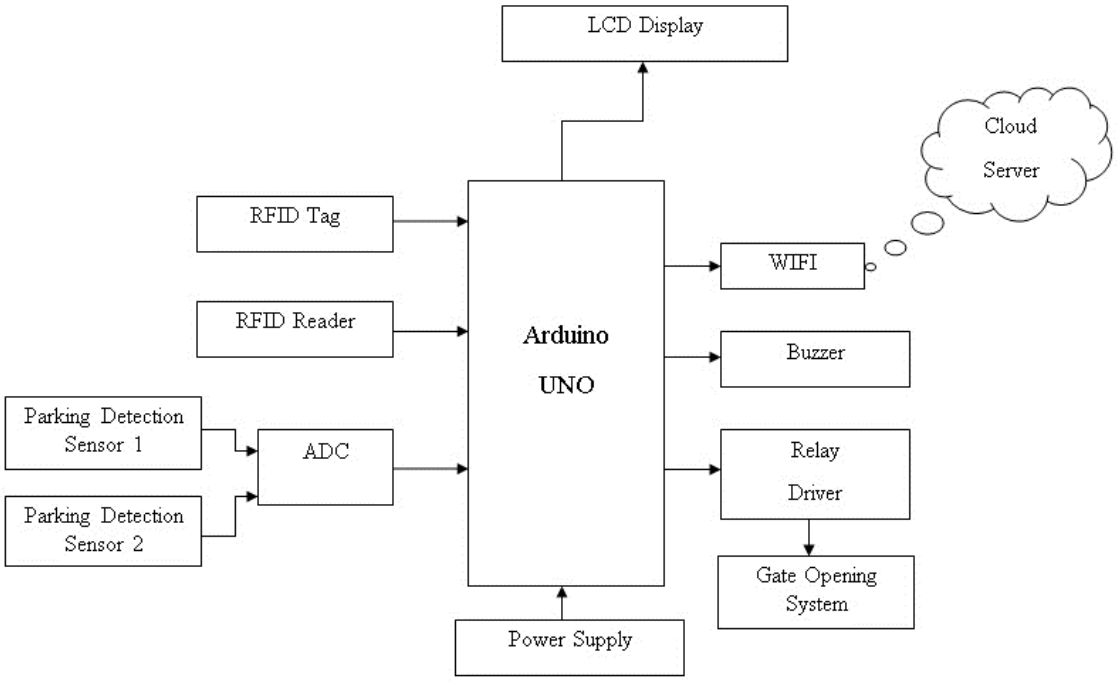
Phase 5: Project Documentation & Submission

- In this part you will document your project and prepare it for submission.
- Document the Smart Parking project and prepare it for submission.
- Describe the project's objectives, IoT sensor setup, mobile app development, Raspberry Pi integration, and code implementation.

- Include diagrams, schematics, and screenshots of the IoT sensors and mobile app.

- Explain how the real-time parking availability system can benefit drivers and alleviate parking issues.

BLOCK DIAGRAM



COMPONENTS:

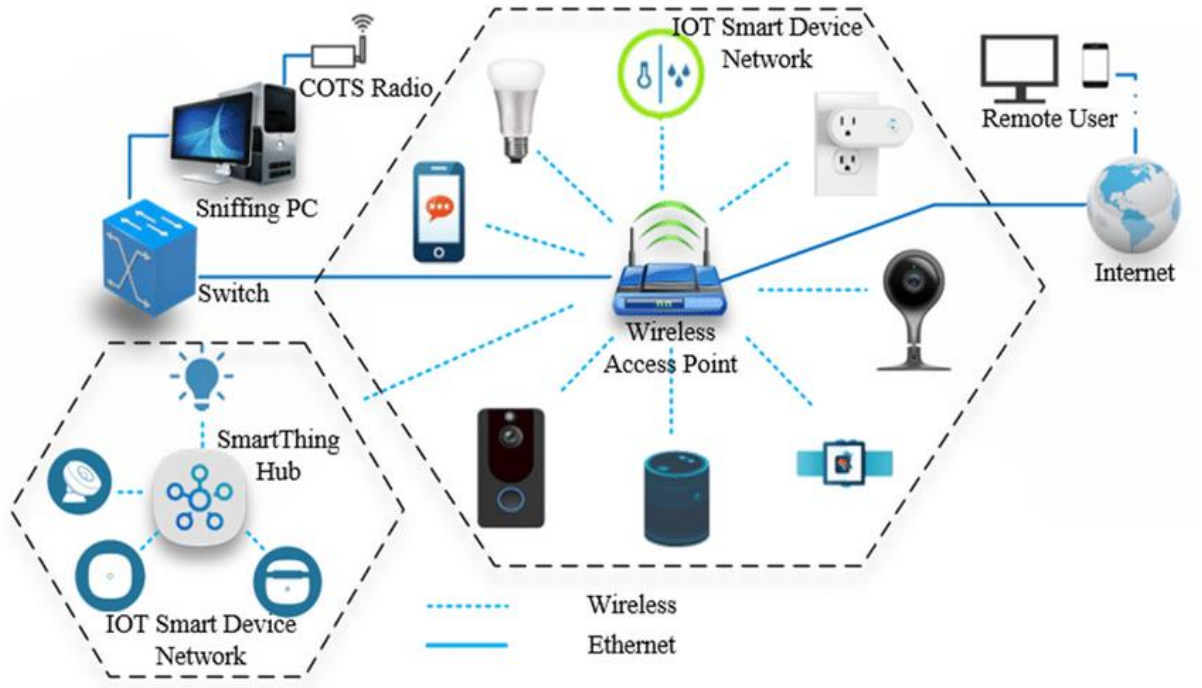
1) Hardware:
   - **a)** Ultrasonic Sensors
   - **b)** Microcontroller (e.g., Arduino or Raspberry Pi)
   - **c)** Wi-Fi Module
   - **d)** Power Supply (for sensors and microcontroller)
   - **e)** LEDs (optional, for visual indicators)
   - **f)** Cables and Breadboard (for prototyping)

2) Software:

a) Arduino IDE (for programming the microcontroller)
b) Python IDE (for programming Raspberry Pi)
c) Mobile App Development Environment (e.g., Android Studio or Swift for iOS)

Devices Setup:

- Setting up an IoT project using an ESP8266 NodeMCU board, ultrasonic sensor, DC servo motor, IR sensors, a 16x2 I2C LCD display, and jumpers involves several steps.
- I'll provide an overview of how you can set up this project, but please note that this is a complex project, and you may need to consult specific documentation and libraries for each component.
- Additionally, coding this project will require programming skills in platforms like Arduino IDE.



1)Gather the Required Components:

- ESP8266 NodeMCU board.
- Ultrasonic sensor (e.g., HC-SR04).
- DC servo motor.
- IR sensors (for object detection).
- 16x2 I2C LCD display.
- Jumper wires and breadboard.

2)Connect the Ultrasonic Sensor:

- Connect the VCC pin of the ultrasonic sensor to the 3.3V output of NodeMCU.
- Connect the GND pin of the ultrasonic sensor to the GND of NodeMCU.
- Connect the TRIG pin of the ultrasonic sensor to a GPIO pin (e.g., D2).
- Connect the ECHO pin of the ultrasonic sensor to another GPIO pin (e.g., D3).

3)Connect the DC Servo Motor:

- Connect the positive (red) lead of the servo motor to the 5V output of NodeMCU.
- Connect the negative (brown) lead of the servo motor to the GND of NodeMCU.

- Connect the signal (orange/yellow) lead of the servo motor to a GPIO pin (e.g., D4).

4)Connect the IR Sensors:

- IR sensors are usually analog sensors. Connect the VCC and GND pins to 3.3V and GND on the NodeMCU.

- Connect the signal pin of the IR sensors to analog GPIO pins (e.g., A0 and A1).

5)Connect the 16x2 I2C LCD Display:

- Connect the SDA (data) and SCL (clock) pins of the I2C LCD display to the corresponding pins on the NodeMCU (D1 and D2 on the NodeMCU, respectively).

- Connect the VCC of the I2C display to 5V on NodeMCU and GND to GND.

6)Write and Upload the Code:

- Write the Arduino code to control your project. This code will involve reading data from the ultrasonic sensor, processing it, controlling the servo motor, and displaying information on the LCD. You'll also need code to handle IR sensor inputs if they're used for object detection.

7)Power Supply:

- Make sure you have a suitable power supply for your servo motor, as the NodeMCU might not be able to provide enough power for it.

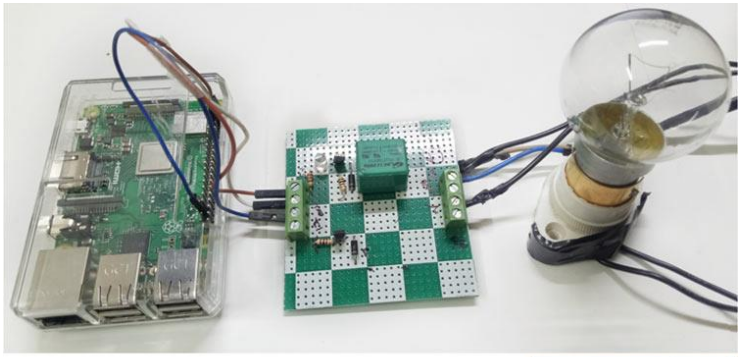
8. Assemble and Test:

- Connect all the components, upload the code to the NodeMCU, and assemble the project.
- Test each component and ensure that the system functions as expected.

Raspberry Pi

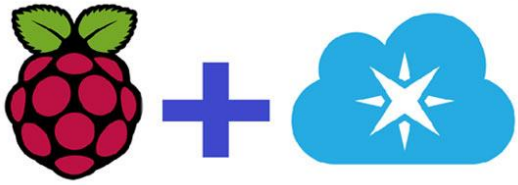
1)Central Controller:

- Use a Raspberry Pi or similar device as a central controller to collect and process data from parking sensors.



2)Connectivity:

- Ensure the Raspberry Pi is connected to the internet to transmit data to the mobile app and back-end server.



3)Control Logic:

- Implement control algorithms to manage traffic and provide real-time updates to users.

Mobile App Development:

The mobile application serves as the interface for end-users. It is designed for both Android and iOS platforms and includes the following features:

Real-time Parking Availability

Displays available parking spaces in a selected area.

Navigation Integration:

Provides directions to the chosen parking spot.

Booking and Payment (Optional):

Allows users to reserve and pay for parking spaces in advance.

Code Implementation:

1)Sensor Code:

Write code for the sensors to detect and transmit data to the central controller.

2)Central Controller Code:

Develop software for the Raspberry Pi to collect and process sensor data, communicate with the server, and control LED displays or other indicators.

3)Mobile App Code:

Create code for the mobile app, including front-end for user interaction and back-end to communicate with the central controller and payment gateway.

4)Server Backend:

Set up a server to store parking data, handle reservations, and communicate with both the mobile app and Raspberry Pi.

PROGRAM:

```
#include <ESP8266WiFi.h>

#include <Servo.h>

#include <LiquidCrystal_I2C.h>

#include <Wire.h>

#include <FirebaseArduino.h>

#define FIREBASE_HOST "smart-parking-7f5b6.firebaseio.com" project name address from firebase id
```

```
// the
#define FIREBASE_AUTH
"suAkUQ4wXRPW7nA0zJQVsx3H2LmeBDPGmfTMBHCT" // the secret key generated from firebase
#define WIFI_SSID "CircuitDigest"
or public wifi name
// input your home
#define WIFI_PASSWORD "circuitdigest101"
for Wifi
//password
String Available = "";
//availability string
String fireAvailable = "";
LiquidCrystal_I2C lcd(0x27, 16, 2); display
//i2c display address 27 and 16x2 lcd

int Empty;    //available space integer
int allSpace = 90;
int countYes = 0;
int carEnter = D0;  // entry sensor
int carExited = D4;//exi sensor
int TRIG = D7;      //ultrasonic trig  pin
int ECHO = D8;     // ultrasonic echo pin
int led = D3;// spot occupancy signal
int pos;
int pos1;
long duration, distance;
```

```
void setup() {
delay(1000);
Serial.begin (9600);        // serial debugging
Wire.begin(D2, D1);         // i2c start
myservo.attach(D6);         // servo pin to D6
myservos.attach(D5);        // servo pin to D5
pinMode(TRIG, OUTPUT);      // trig pin as output
pinMode(ECHO, INPUT);       // echo pin as input
pinMode(led, OUTPUT);// spot indication
pinMode(carExited, INPUT);    // ir as input
pinMode(carEnter, INPUT);     // ir as input
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

connect with wifi
Serial.print("Connecting to ");
Serial.print(WIFI_SSID);// display ssid
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");     // if not connected print this
delay(500);
//try to
}

Serial.println();
Serial.print("Connected to ");
Serial.println(WIFI_SSID);
Serial.print("IP Address is : ");
Serial.println(WiFi.localIP());    //print local IP address
```

```
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);// begin firebase
authentication

lcd.begin();  //begin lcd
lcd.home();
lcd.setCursor(0, 0);          // 0th row and 0thh column
lcd.print("Smart Parking");
}

void loop() {
digitalWrite(TRIG, LOW);
delayMicroseconds(2);

// make trig pin low
digitalWrite(TRIG, HIGH);       // make trig pin high
delayMicroseconds(10);
digitalWrite(TRIG, LOW);
duration = pulseIn(ECHO, HIGH);
distance = (duration / 2) / 29.1;  // take distance in cm
Serial.print("Centimeter: ");
Serial.println(distance);

int carEntry = digitalRead(carEnter);   // read ir input
if (carEntry == HIGH) {  // if high then count and send data
countYes++;//increment count
Serial.print("Car Entered = " ); Serial.println(countYes );
```

```
lcd.setCursor(0, 1);
lcd.print("Car Entered");

for (pos = 140; pos >= 45; pos -= 1) { // change servo position
myservos.write(pos);
delay(5);
}
delay(2000);

for (pos = 45; pos <= 140; pos += 1) {
// change servo position
in steps of 1 degree myservos.write(pos); delay(5);
}
11
Firebase.pushString("/Parking Status/", fireAvailable );    // send string to
firebase
lcd.clear();
}

int carExit = digitalRead(carExited);   //read exit ir sensor
if (carExit == HIGH) {    //if high then count and send
countYes--;  //decrement count
Serial.print("Car Exited = " ); Serial.println(countYes);
lcd.setCursor(0, 1);
lcd.print("Car Exited");

for (pos1 = 140; pos1 >= 45; pos1 -= 1) {     // change servo position
myservo.write(pos1);
```

```
delay(5);
}
delay(2000);

for (pos1 = 45; pos1 <= 140; pos1 += 1) {
// change servo position
in steps of 1 degree myservo.write(pos1); delay(5);
}
Firebase.pushString("/Parking Status/", fireAvailable ); // send string to firebase
lcd.clear();
}

if (distance < 6) {  //if distance is less than 6cm then on led
Serial.println("Occupied ");
digitalWrite(led, HIGH);
}

if (distance > 6) {  //if distance is greater than 6cm then off led
Serial.println("Available ");
digitalWrite(led, LOW);
}

Empty = allSpace - countYes;
//calculate available data
Available = String("Available= ") + String(Empty) + String("/") +
String(allSpace); // convert the int to string
fireAvailable = String("Available=") + String(Empty) + String("/") +
String(allSpace);
```

```
lcd.setCursor(0, 0);
lcd.print(Available);        //print available data to lcd
}
```

Real time parking availability :

A real-time parking availability system offers several benefits to drivers and effectively alleviates parking issues in smart parking:

1)Time and Stress Savings:

- Drivers can save time and reduce stress by quickly locating available parking spaces without the need to circle the parking lot or search for an empty spot.

2)Reduced Emissions:

- Efficient parking systems help decrease unnecessary driving, which in turn reduces emissions and environmental impact.

3)Cost Savings:

- Drivers can optimize their parking expenses by choosing available spaces based on cost or proximity to their destination.

4)Accessibility:

- Drivers with special needs can easily find and reserve accessible parking spots.

5)Enhanced User Experience:

- A user-friendly mobile app and real-time information improve the overall experience for drivers, making parking more convenient and enjoyable.

6)Economic Benefits:

The deployment of such systems can stimulate local economies by encouraging people to visit and shop in areas where parking was previously a deterrent

Raspberry Pi Data Transmission Output:

1)Sensor Data Processing:

Sensor 1: Occupied
Sensor 2: Available
Sensor 3: Occupied
Sensor 4: Available

2)Central Controller Logic:

Total Available Spaces: 2
Total Occupied Spaces: 2

3)Data Transmission to Mobile App:

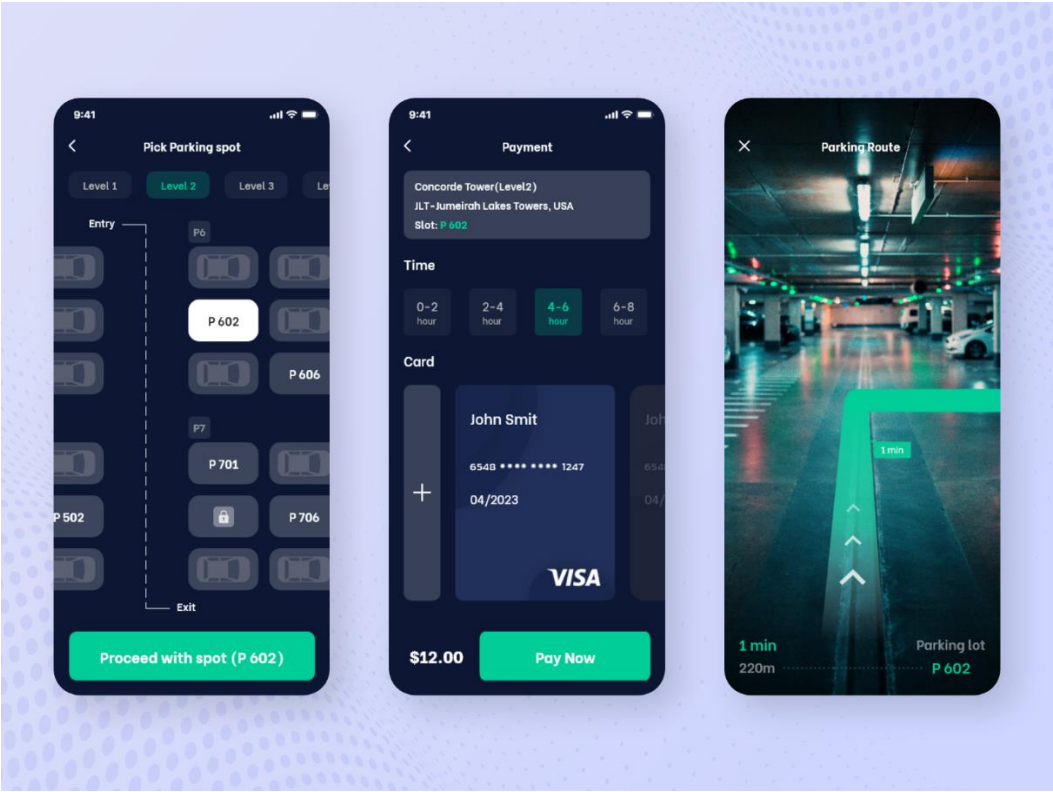
"Available Parking Spaces: 2"
"Occupied Parking Spaces: 2"
"Last Update: 10:30 AM"

- These are basic examples of the data sent from the Raspberry Pi to the mobile app, which would update in real-time as parking space statuses change.

Mobile App User Interface:

Main Screen:

- Displays a map with color-coded parking space markers:
    - Green for available spaces.
    - Red for occupied spaces.



- "Find Parking" button for real-time space availability.

Search and Reserve:

- Allows users to search for parking near a destination.
- Provides the option to reserve a parking spot and set a duration.

Navigation:

- Offers turn-by-turn navigation to the selected parking spot.

Payment:

- Enables users to pay for parking through the app using various payment methods.

Profile:

- User profile with payment history, reservations, and settings.

Notifications:

- Real-time updates on parking availability, reservation confirmations, and expiration reminders.

Feedback and Help:

- Options for user feedback and support.

Alerts and Warnings:

- Notifications for time-limited parking or special conditions (e.g., street cleaning).

Historical Data:

- Historical parking occupancy data for various time slots.

These examples are simplified to illustrate the key features of the system.

Conclusion:

The Smart Parking System leverages IoT technology to provide a seamless parking experience for drivers. With real-time availability information, drivers can save time and reduce the environmental impact of urban commuting. This project represents a significant step toward creating smarter, more efficient cities.