SMART PARKING

Introduction to Smart Parking

Smart parking is a modern technological solution aimed at revolutionizing the way we approach parking in urban and suburban environments. As cities around the world face increasing challenges related to traffic congestion, limited parking spaces, and environmental concerns, smart parking systems have emerged as a promising answer to these issues.

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. The key idea behind smart parking is to optimize the utilization of parking spaces, reduce the time and fuel wasted in the search for parking, and enhance the overall urban mobility and quality of life.

In a smart parking system, sensors are installed in parking spaces to monitor their availability in real-time. This data is then relayed to users through mobile applications or digital displays, allowing them to easily find and reserve parking spots. Additionally, smart parking solutions often include features like automated payment systems and integration with navigation apps to guide drivers to available parking spaces.

The benefits of smart parking extend beyond individual convenience. By reducing traffic congestion and the environmental impact of circling for parking, these systems contribute to improved air quality and reduced emissions. They also have the potential to boost revenue for municipalities or private operators through dynamic pricing and efficient enforcement. Furthermore, smart parking aligns with broader urban development goals by enhancing safety, promoting sustainable transportation alternatives, and integrating with other aspects of urban infrastructure, such as public transportation and city planning.

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. It represents a proactive and innovative approach to tackling the parking challenges that accompany urban growth, making cities more livable and sustainable for residents and visitors alike. As technology continues to advance, the future of smart parking promises even more innovative and intelligent solutions to improve the way we park and move within urban environments.

Project Objectives

Optimize Parking Space Utilization: The primary goal of smart parking is to maximize the use of available parking spaces. This involves reducing congestion and minimizing the time and fuel wasted by drivers searching for parking spots.

Reduce Traffic Congestion: By guiding drivers to available parking spaces and reducing the time spent circling for a spot, smart parking systems aim to alleviate traffic congestion in urban areas. This can lead to reduced emissions and improved air quality.

Enhance User Convenience: Smart parking systems should make it easier for drivers to find, reserve, and pay for parking. Mobile apps, online booking, and real-time availability updates contribute to a more convenient and user-friendly experience.

Improve Revenue Generation: Many smart parking initiatives are designed to increase revenue for municipalities or private operators. This can be achieved through dynamic pricing, efficient enforcement, and improved space turnover.

Enhance Safety and Security: Smart parking solutions can incorporate security features like surveillance cameras and emergency call buttons to enhance the safety of parking facilities. **Reduce Environmental Impact**: Reducing the time vehicles spend idling and circling for parking spots can contribute to lower fuel consumption and emissions, thus helping to combat air pollution and climate change.

Promote Sustainable Transportation: Smart parking projects often aim to encourage the use of public transport, carpooling, and non-motorized modes of transportation by making these options more accessible and convenient.

Data Collection and Analysis: Gathering data on parking space utilization, traffic patterns, and user behavior is a key objective. Analyzing this data can help urban planners make informed decisions and optimize parking policies.

Enhance Accessibility: Smart parking should be designed to cater to the needs of all users, including those with disabilities, by providing accessible parking spaces and user-friendly features.

Integration with Urban Infrastructure: Smart parking systems should be integrated with broader urban infrastructure, including traffic management, public transportation, and city planning, to ensure a cohesive and well-coordinated approach to urban mobility.

Sustainability and Green Initiatives: Some smart parking projects may include the implementation of sustainable infrastructure elements like electric vehicle charging stations, solar-powered parking meters, and green urban design.

Economic Development: In certain cases, smart parking projects are expected to stimulate economic growth by making it easier for people to access businesses and attractions in urban areas.

Enforcement and Compliance: Ensure that parking rules and regulations are effectively enforced through automated methods, such as license plate recognition or ticketing systems.

Feedback and User Engagement: Smart parking projects should encourage user feedback and engagement to continuously improve the system based on user experiences and preferences.

Scalability and Adaptability: The system should be designed to adapt to changing urban needs and to be scalable as the city or area grows **IoT Devices :**

1. ESP8266 NodeMCU
2. Ultrasonic Sensor
3. DC Servo Motor
4. IR Sensors
5. 16x2 i2c LCD Display
6. Jumpers

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:

1. ESP8266 NodeMCU board.
2. Ultrasonic sensor (e.g., HC-SR04).
3. DC servo motor.
4. IR sensors (for object detection).
5. 16x2 I2C LCD display.
6. Jumper wires and breadboard.
7. Power supply for the servo motor if needed.

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.

Platform Development:

1. **Define Your Goals and Objectives:**

Clearly define the objectives and goals of your smart parking platform. Understand what problems you want to solve, whether it's reducing congestion, enhancing revenue generation, improving accessibility, or promoting sustainability.

2. **Hardware Selection:**

Choose the appropriate hardware components, such as sensors (ultrasonic, infrared, camera-based), microcontrollers (like Raspberry Pi or Arduino), communication modules (Wi-Fi, LoRa, or cellular), and displays for realtime information.

3. **Sensor Installation:**

Install sensors in parking spaces to monitor availability. Ensure the sensors can detect the presence or absence of vehicles accurately. These sensors may be ultrasonic or infrared-based, depending on your project requirements.

4. **Communication Infrastructure:**

Set up a reliable communication infrastructure that connects the sensors to a central server or cloud platform. Wi-Fi, LoRa, or cellular networks are commonly used for data transmission.

5. **Central Server or Cloud Platform:**

Develop a central server or cloud-based platform to collect, process, and store data from the sensors. Use a robust database system to manage realtime parking space availability.

6. **User-Facing Mobile and Web Applications:**

Create user-friendly mobile and web applications that allow users to:

Find available parking spaces.

Reserve parking spots in advance.

Make payments for parking.

Receive real-time updates on parking availability and guidance.

7. **Payment Integration:**

Integrate payment gateways into the mobile app for user convenience. This can include options for cashless payments, credit card payments, and mobile wallets.

8. **Data Analytics and Prediction:**

Implement data analytics to analyze historical and real-time parking data. This can help in predicting parking demand, optimizing pricing, and improving operational efficiency.

9. **User Feedback and Support:**

Include features for user feedback, customer support, and assistance in case of issues or emergencies.

10. **Security and Access Control:**

Ensure the security of data and transactions. Implement user authentication and access control features to prevent unauthorized use or tampering with the system.

11. **Real-time Displays and Signage:**

Install displays at the parking facility entrances to guide drivers to available spaces and provide real-time updates on space availability.

12. **Environmental Considerations:**

For sustainability, consider incorporating features such as electric vehicle charging stations, solar-powered components, and green infrastructure.

13. **Scalability and Flexibility:**

Design your platform to be scalable, allowing for easy expansion to more parking areas as needed.

14. **Regulatory Compliance:**

Ensure your platform complies with local parking regulations, privacy laws, and other relevant regulations.

15. **Testing and Maintenance:**

Thoroughly test the system to ensure its reliability and accuracy. Develop a maintenance plan for regular sensor and system checks.

16. **User Education and Onboarding:**

Provide clear instructions to users on how to use the platform and make the transition to smart parking smooth.

17. **Marketing and Adoption:**

Promote your smart parking platform to attract users and encourage adoption. This may involve partnerships with local businesses and marketing campaigns.

18. **Continuous Improvement:**

Continuously monitor the system's performance and gather user feedback for ongoing improvement and innovation.

Code Implementation:

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" // the project name
address from firebase id

#define FIREBASE_AUTH
"suAkUQ4wXRPW7nA0zJQVsx3H2LmeBDPGmfTMBHCT" // the secret key
generated from firebase
```

```
#define WIFI_SSID "CircuitDigest" // input your home or
public wifi name
#define WIFI_PASSWORD "circuitdigest101" //password for Wifi
String Available = ""; //availability string

String fireAvailable = "";

LiquidCrystal_I2C lcd(0x27, 16, 2); //i2c display address 27 and 16x2 lcd display
Servo myservo; //servo as gate

Servo myservos; //servo as gate

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); //try to
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);

}

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); // make trig pin low

delayMicroseconds(2);
```

```
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);
```

```
        }
```

```
 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

}
```

Flutter UI code :

lib/main.dart ::

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http; // Import the HTTP package

void main() {
runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Smart Parking App',
      home: ParkingApp(),
    );
  }
}

class ParkingApp extends StatefulWidget {
  @override
  _ParkingAppStatecreateState() => _ParkingAppState();
}

class _ParkingAppState extends State<ParkingApp> {
  int availableSpots = 0;

  Future<void>fetchAvailableSpots() async {
    // Replace the URL with your server endpoint to fetch parking availability
    final response = await http.get(Uri.parse('https://your-server-
url.com/parking/availability'));
```

```
    if (response.statusCode == 200) {
setState(() {
availableSpots = int.parse(response.body);
      });
    } else {
      throw Exception('Failed to load data');
    }
  }

  @override
  void initState() {
super.initState();
fetchAvailableSpots();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
appBar: AppBar(
        title: Text('Smart Parking'),
      ),
      body: Center(
        child: Column(
mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
Text(
              'Find Available Parking Spots',
              style: TextStyle(fontSize: 24),
            ),
SizedBox(height: 20),
ElevatedButton(
onPressed: () {
                // Implement functionality to find parking spots
              },
              child: Text('Find Parking'),
            ),
SizedBox(height: 10),
Text(
              'Available Spots: $availableSpots',
              style: TextStyle(fontSize: 18),
            ),
          ],
        ),
      ),
    );
  }
}
lib/models/parking_spot.dart
// lib/models/parking_spot.dart


class ParkingSpot {

  final String id;

  final String name;

  final bool isAvailable;


ParkingSpot({

    required this.id,

    required this.name,

    required this.isAvailable,

  });


  factory ParkingSpot.fromJson(Map<String, dynamic>json) {
    return ParkingSpot(
      id: json['id'],
      name: json['name'],
isAvailable: json['isAvailable'],
    );
  }

  Map<String, dynamic>toJson() {
    return {
      'id': id,
      'name': name,
      'isAvailable': isAvailable,
    };
  }
}
Lib/services/api_services.dart
// lib/services/api_service.dart
import 'dart:convert';
import 'package:http/http.dart' as http;

class ApiService {
  final String baseUrl;

ApiService({required this.baseUrl});
```

```
  Future<ParkingSpot>fetchParkingSpot() async {
    final response = await http.get(Uri.parse('$baseUrl/parking/spot'));

    if (response.statusCode == 200) {
      final jsonData = json.decode(response.body);
      return ParkingSpot.fromJson(jsonData);
    } else {
      throw Exception('Failed to fetch parking spot data');
    }
  }

  // Add additional methods for making other API requests as needed
}
Lib/widgets/parking_card.dart
// lib/widgets/parking_card.dart
import 'package:flutter/material.dart';
import 'package:your_app/models/parking_spot.dart'; // Import your data model

class ParkingCard extends StatelessWidget {
  final ParkingSpot spot;

ParkingCard({required this.spot});
```

```
  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 3,
      margin: EdgeInsets.all(10),
      child: Padding(
        padding: EdgeInsets.all(10),
        child: Column(
          children: <Widget>[
Text(
              spot.name,
              style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
SizedBox(height: 5),
Text(
              'Available: ${spot.isAvailable ? 'Yes' : 'No'}',
              style: TextStyle(fontSize: 16),
            ),
          ],
        ),
      ),
    );
  }
}
```

Homescreen.dart

```
// lib/screens/home_screen.dart
import 'package:flutter/material.dart';
import 'package:your_app/widgets/parking_card.dart'; // Import your widgets
import 'package:your_app/services/api_service.dart'; // Import your API service

class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenStatecreateState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final ApiServiceapiService = ApiService(baseUrl: 'your-api-url'); // Replace with
your API URL
  late Future<ParkingSpot>parkingSpot;

  @override
  void initState() {
super.initState();
parkingSpot = apiService.fetchParkingSpot();
  }

  @override
  Widget build(BuildContext context) {
```

```
  return Scaffold(
appBar: AppBar(
      title: Text('Smart Parking App'),
    ),
    body: Center(
      child: Column(
mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
Text(
            'Available Parking Spots',
            style: TextStyle(fontSize: 24),
          ),
SizedBox(height: 20),
FutureBuilder<ParkingSpot>(
            future: parkingSpot,
            builder: (context, snapshot) {
              if (snapshot.hasData) {
                return ParkingCard(spot: snapshot.data!);
              } else if (snapshot.hasError) {
                return Text('Error: ${snapshot.error}');
              }
              return CircularProgressIndicator();
            },
          ),
        ],
       ),
      ),
    );
  }
}
Lib/screens/parking_details_screen.dart
// lib/screens/parking_details_screen.dart
import 'package:flutter/material.dart';

class ParkingDetailsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
appBar: AppBar(
      title: Text('Parking Details'),
     ),
     body: Center(
      child: Column(
mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
Text(
          'Details of the Selected Parking Spot',
          style: TextStyle(fontSize: 24),
          ),
          // Add more widgets to display parking details
        ],
      ),
    ),
  );
 }
}
```

Server.py

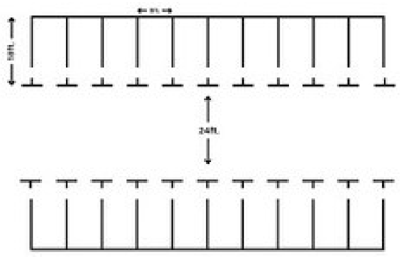
```
# server.py
from flask import Flask, request, jsonify
from datetime import datetime

app = Flask(__name)

# Sample data store for parking spots
parking_spots = [
   {"id": 1, "name": "Parking Spot 1", "isAvailable": True},
   {"id": 2, "name": "Parking Spot 2", "isAvailable": False},
]

@app.route('/parking/availability', methods=['GET'])
def get_parking_availability():
   return jsonify(parking_spots)
```

```
@app.route('/parking/spot', methods=['POST'])
def update_parking_spot():
   data = request.json
spot_id = data.get('id')
is_available = data.get('isAvailable')

   for spot in parking_spots:
      if spot['id'] == spot_id:
         spot['isAvailable'] = is_available
         break

   return jsonify({"message": "Parking spot updated"})

if __name__ == '__main__':
app.run(debug=True)
```

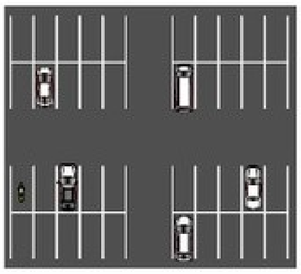
IOT sensor code:-

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char *ssid = "your_wifi_ssid";
const char *password = "your_wifi_password";
const char *server = "your_server_ip"; // or domain

WiFiClient client;
String postStr;

void setup() {
Serial.begin(115200);
delay(10);

  // Connect to Wi-Fi
WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
delay(1000);
Serial.println("Connecting to WiFi...");
  }
Serial.println("Connected to WiFi");

  // Set the server endpoint
server.begin();
}

void loop() {
  // Read sensor data
  float distance = readDistance(); // Implement your sensor reading logic
  // Send data to the server
  if (client.connect(server, 80)) {
postStr = "id=1&isAvailable=";
postStr += (distance >100 ? "true" : "false"); // Adjust the condition based on your
sensor data
client.println("POST /parking/spot HTTP/1.1");
client.println("Host: " + String(server));
client.println("Content-Type: application/x-www-form-urlencoded");
client.println("Connection: close");
client.print("Content-Length: ");
client.println(postStr.length());
client.println();
client.print(postStr);
  }

delay(5000); // Adjust the delay as needed
}
```

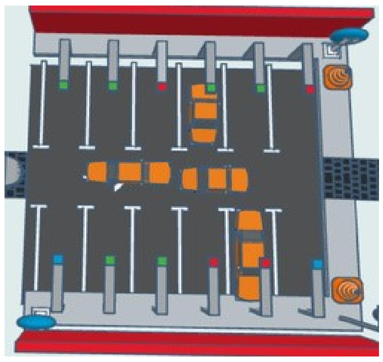
Diagram Representation



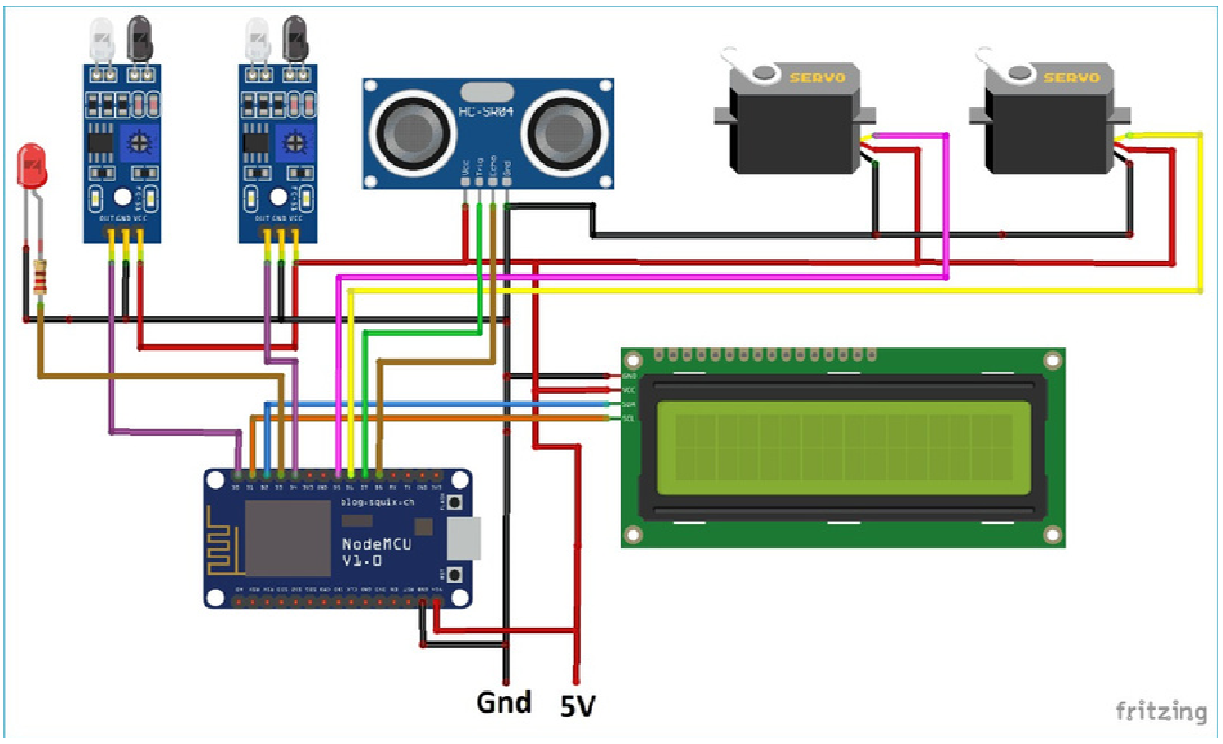
Schematic:



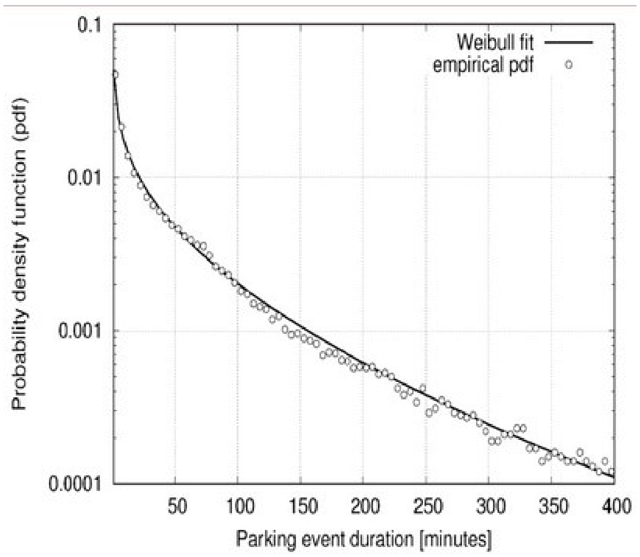
Screenshot:

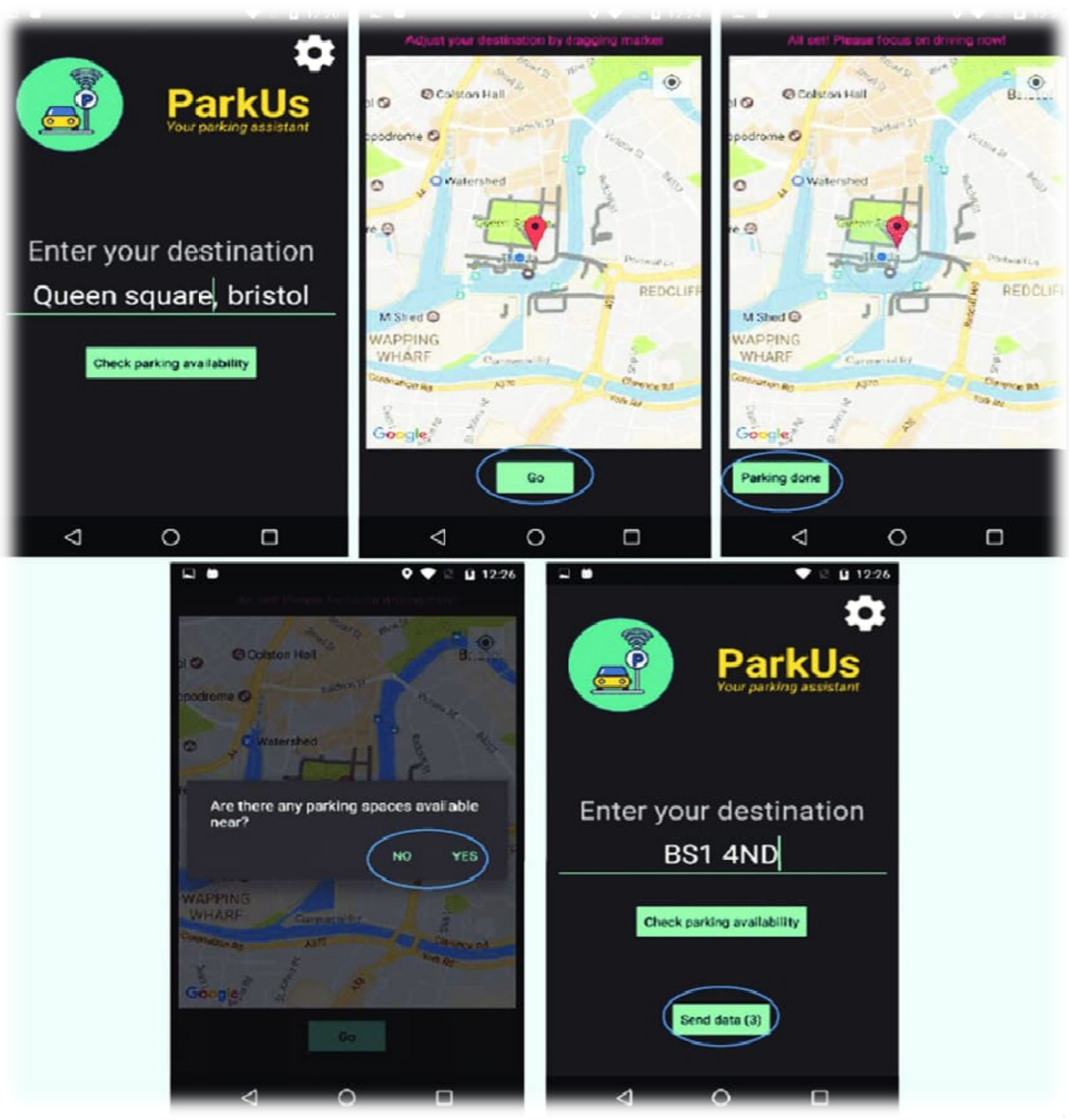


IoT Devices ss:



Data Sharing:




Enter your destination

Queen square, bristol

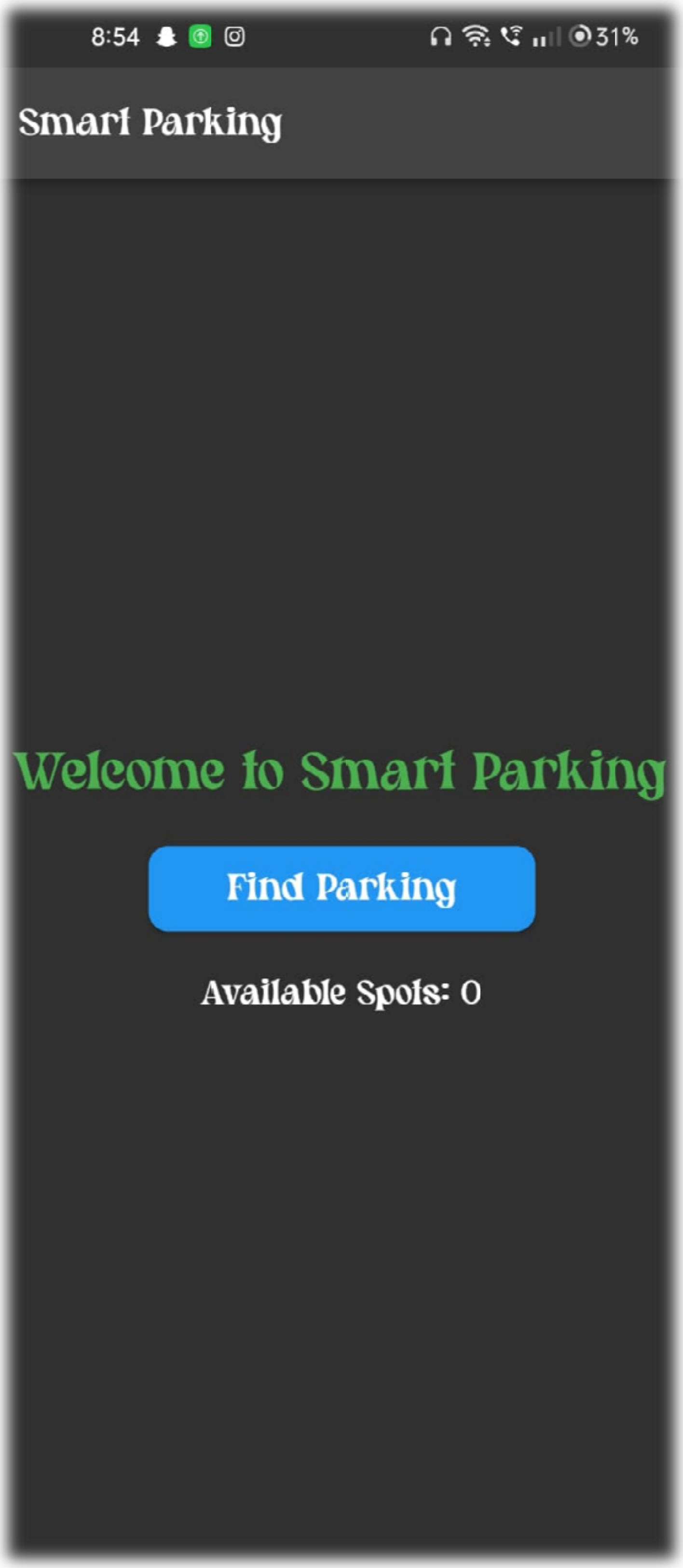
Adjust your destination by dragging marker

All set! Please focus on driving now!

Are there any parking spaces available near?

Enter your destination

BS1 4ND



Welcome to Smart Parking

Available Spots: 0