PROJECT TITLE: SMART PARKING USING IOT

PHASE 5: Project Documentation & Submission

As we progress into the fifth phase of our smart parking using IoT project, our focus shifts towards the active documentation of the system. This phase represents a pivotal stage where we are actively demonstrating the key components necessary for a successful Smart Parking solution. Below, we outline the significant deployment steps along with output samples.

OBJECTIVES:

- **Location Tracking:** Implement GPS-based location tracking for public transportation vehicles, allowing passengers to track the real-time location of buses or trains.
- **Parking Space Monitoring:** Deploy IoT sensors in parking spaces at transit hubs to detect occupancy and availability of parking spots.
- **Mobile App for real time access:** Design and develop a user-friendly mobile application that presents real-time transit information to commuters, including ridership data, vehicle locations, arrival predictions, and parking availability.
- **Efficient Parking Guidance:** Integrate the parking availability data into the mobile app to provide commuters with guidance on available parking spaces at transit hubs

INTRODUCTION:

In the intricate tapestry of urban life, the effective management of parking spaces stands as an enduring challenge. For city dwellers and visitors alike, the quest for a vacant parking spot often translates into precious time lost in the mazes of congested streets. As our cities grow and evolve, the need for a smarter, more efficient approach to parking becomes increasingly evident. In the midst of this urban challenge, we have embarked on a transformative journey to introduce a groundbreaking solution: the Smart Parking System. This innovative project represents a convergence of cutting-edge technology, combining IoT sensor setup, mobile app development, Raspberry Pi integration, and meticulous code implementation. The central goal of our project is to alleviate parking issues, reduce traffic congestion, and empower drivers with real-time parking information. With the deployment of IoT sensors, a user-friendly mobile app, and the computational capabilities of Raspberry Pi devices, we aim to create a seamless and efficient parking experience for city residents and visitors alike. The objectives of this endeavour are multi-fold. We strive to streamline parking operations, minimize the environmental impact of traffic congestion, and enhance the overall urban mobility experience. The Smart Parking System leverages IoT sensor data, harnessing the power of the mobile app to provide users with up-to-the-minute information on parking space availability. Drivers can now find parking with ease, saving time and reducing the associated stress of urban parking. Our mobile app serves as the primary interface for users, offering real-time parking data, navigation guidance to available parking spaces, and the ability to make reservations in advance. Through Raspberry Pi devices, we ensure the efficient management and transmission of data, connecting the physical world of parking sensors to the digital realm of the central server and mobile app. The heart of our project lies in the code implementation. The central server, mobile app, and Raspberry Pi devices work in harmony, ensuring the seamless flow of real-time parking information. This intricate web of code is written in languages like Python, Node.js, and Ruby on Rails, with mobile apps crafted in platform-specific languages like Java, Kotlin, Swift, and Objective-C. Robust security measures are integrated into the code to safeguard sensitive data and system integrity. The benefits of our Smart Parking System extend to both drivers and city authorities. Drivers gain access to accurate and real-time parking information, significantly reducing the time spent searching for parking spaces. By reducing traffic congestion and emissions, the system contributes to a more environmentally sustainable urban environment. In addition, cities benefit from more efficient parking space management, leading to improved traffic flow and reduced environmental impact.

Dataset Link:

- https://www.kiwi-park.co.nz/?gclid=EAIaIQobChMImOXJk6OjggMVvx6DAx3pCAyGEAAYASAAEgJnM_D_BwE

TOOLS AND SOFTWARE:

- **Hardware and Sensors**: Modern parking systems rely on various sensors, such as ultrasonic sensors, infrared sensors, or cameras, for real-time data collection. These sensors are strategically placed to monitor parking space occupancy, providing valuable data for efficient parking management.

- **IoT Platforms:** IoT platforms, such as AWS IoT or Microsoft Azure IoT, play a crucial role in connecting and managing the myriad of sensors used in the Smart Parking System. They facilitate real-time data transmission, sensor control, and data storage, ensuring seamless communication between the physical and digital components of the system.

- **Mobile Apps Development:** Mobile apps are a fundamental part of the Smart Parking System, allowing users to access real-time parking information and make informed decisions. Development platforms and programming languages like Android Studio (Java/Kotlin) or Xcode (Swift/Objective-C) are employed to create user-friendly apps with features like parking space reservations and navigation to available spaces.

- **Backend Development:** Backend development is essential for handling data processing, API requests, and database interactions. Frameworks like Node.js, Django, or Ruby on Rails are used to build the server-side components of the system. They ensure the secure and efficient management of data and real-time interactions with the frontend.

- **Database Management**: Effective data storage and retrieval are critical in a Smart Parking System. Databases like MySQL, PostgreSQL, or NoSQL solutions (e.g., MongoDB) are used to store information about parking space availability, user profiles, and historical data. These databases are optimized for fast and reliable data access.

- **Real-Time Data Processing:** Real-time data processing is a cornerstone of the system. Tools and technologies like Apache Kafka, RabbitMQ, or Apache Flink are employed to process data from sensors and update the platform in real-time. This ensures users receive accurate and up-to-date parking information.

- **Data Visualization**: Data visualization libraries and tools like Matplotlib, Seaborn, or D3.js are used to create interactive maps, charts, and real-time parking updates for users. These visualizations provide users with an intuitive and user-friendly way to find available parking spaces and understand parking conditions.

- **User Authentication:** User authentication systems are developed to enable account creation, user profile management, and personalized parking notifications. Technologies like OAuth, JWT (JSON Web Tokens), or Firebase Authentication are integrated to ensure secure and convenient user access.

- **Push Notifications:** The implementation of push notifications is a critical component. Services like Firebase Cloud Messaging (FCM) or Apple Push Notification Service (APNs) are used to notify users of important parking updates, such as space availability, reservations, or alerts in real-time.

- **Security Measures**: Robust security measures are paramount in protecting sensitive parking data and the system against cyber threats. This includes the implementation of encryption, secure communication protocols (HTTPS), and regular security audits to ensure data privacy and system integrity.

- **Web Development Tools** : For systems that include web interfaces for parking management and information display, web development tools like HTML, CSS, JavaScript, and web frameworks (e.g., Flask or Django) are employed to create user-friendly and informative web-based dashboards.

- **Cloud Services:** For larger-scale Smart Parking Systems, cloud platforms such as AWS, Google Cloud, or Azure are utilized to provide scalable computing and storage resources. These platforms ensure the system can handle increased user traffic and data volumes while maintaining performance and reliability.

IOT SENSOR SETUP:

1. Infrared Sensors, Passive Infrared (PIR), and Ultrasonic Sensors: These sensors detect the presence or absence of vehicles in parking spaces. They are crucial for real-time monitoring of parking occupancy.
2. Microcontroller (e.g., Raspberry Pi) ESP8266 WiFi Chip: This chip facilitates wireless communication between the Raspberry Pi and the IoT sensors, enabling data collection and transmission.

MOBILE APP DEVELOPMENT

The mobile app is designed to address the challenge of enhancing public transportation services through IoT sensors which offers a user-friendly and feature-rich experience to the users. The mobile app is aimed at revolutionising the experience of parking system. Our app provides the power of IoT sensors integrated into public transportation vehicles and transit hubs to provide you with real-time transit information and an effortless commuting experience.

Key Features:

- **Real-Time Transit Information:** The Mobile app provides up-to-the-minute information on public transportation services. You can access real-time updates on bus and train locations, arrival times, and occupancy status.

- **User-Friendly Interface:** Our app is designed for users of all ages and tech-savviness levels. The intuitive interface ensures that accessing transit information is a breeze. We can personalize the app's appearance and layout to suit our preferences and needs.

- **Accessibility:** The app is accessible to a broad range of users, including those with disabilities, by following accessibility guidelines and standards. It provides features such as customizing, voice search and provides screen reader support for the users.

- **Map Based Visualization:** It utilizes maps or visual representations to display parking availability, making it easy for users to understand and navigate to available spaces.

- **Ridership Monitoring:** It helps to view detailed ridership statistics for different routes which helps you to choose less congested options during peak hours. It analyses historical ridership data to make informed decisions about when to travel and avoid the busiest times.

- **Route Planning:** One can easily plan their entire journey with our route planning feature. Enter your starting point and destination, and the app will suggest the best routes, including connections and transfers.

- **Feedback and Support:** It receives feedbacks from the users which helps to improve the app further and rectify the faults in it. You can easily report issues, provide feedback, or contact customer support through the app.

RASPBERRY PI INTEGRATION

Raspberry Pi can be integrated into an IoT smart parking system to provide a powerful, flexible, and cost-effective platform for managing and monitoring parking spaces.

- **IoT Sensors (Infrared Sensors, PIR, Ultrasonic Sensors):**
  These sensors are placed in each parking space to monitor occupancy.They continuously detect the presence or absence of vehicles in real-time.Data collected by the sensors is sent to the microcontroller (e.g., Raspberry Pi) for processing.

- **ESP8266 WiFi Chip:**
  The ESP8266 WiFi chip provides wireless connectivity between the microcontroller (Raspberry Pi) and other components.It securely transmits the parking occupancy data to the central server for further analysis.The chip ensures that the data is sent in real-time, enabling timely updates on parking availability.

- **IoT Communication:**
  Set up MQTT (Message Queuing Telemetry Transport) or another IoT communication protocol to send parking space occupancy data to the cloud or a central server. MQTT is lightweight and ideal for IoT applications. We can also use services like AWS IoT, Azure IoT, or Google Cloud IoT Core to manage data.

- **User Interface:**
  Create a user interface for the system. This can be a web-based dashboard that displays real-time parking space occupancy information. Implement user registration and login systems, as well as features like booking parking spaces, finding available spaces, and receiving notifications.

- **Mobile App Integration:**
  We can integrate the mobile app with the Raspberry Pi server using APIs or other communication methods. The mobile app communicates with the central server to fetch and display parking space availability and other relevant information. Users receive updates on available parking spaces, allowing them to plan their trips efficiently

- **Security:**
  We can implement security measures to protect data and communications, including encryption, secure APIs, and user authentication.

CODE IMPLEMENTATION

Web Development Code: (HTML)

```
<!DOCTYPE html>
<html>
<head>
<title>Real-Time Parking Information</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<div class="header">
<h1>Real-Time Parking Information</h1>
</div>
<div class="parking-info">
<h2>Current Parking Availability</h2>
<p>Parking Lot A: Available</p>
<p>Parking Lot B: Full</p>
<p>Parking Lot C: Available</p>
```

```
</div>
<div class="map">
<!-- Insert interactive map here -->
</div>
<script src="script.js"></script>
</body>
</html>
```

JAVASCRIPT CODE:

```
// This is a placeholder for real-time data fetching and updates
function updateParkingInformation() {
const lotA = "Available";
const lotB = "Full";
const lotC = "Available";
document.querySelector(".parking-info").innerHTML = `
<h2>Current Parking Availability</h2>
<p>Parking Lot A: ${lotA}</p>
<p>Parking Lot B: ${lotB}</p>
<p>Parking Lot C: ${lotC}</p>
`;
}
// Simulate data updates every 30 seconds (adjust as needed)
setInterval(updateParkingInformation, 30000);
// Initialize with initial data
updateParkingInformation();
```

CSS CODE:

```
body {
font-family: Arial, sans-serif;
}
.header {
background-color: #333;
color: white;
text-align: center;
padding: 10px;
}
.parking-info {
padding: 20px;
}
.map {
/* Add styles for the map container */
}
```

SAMPLE FRONT-END MOBILE APP CODE (IOS):

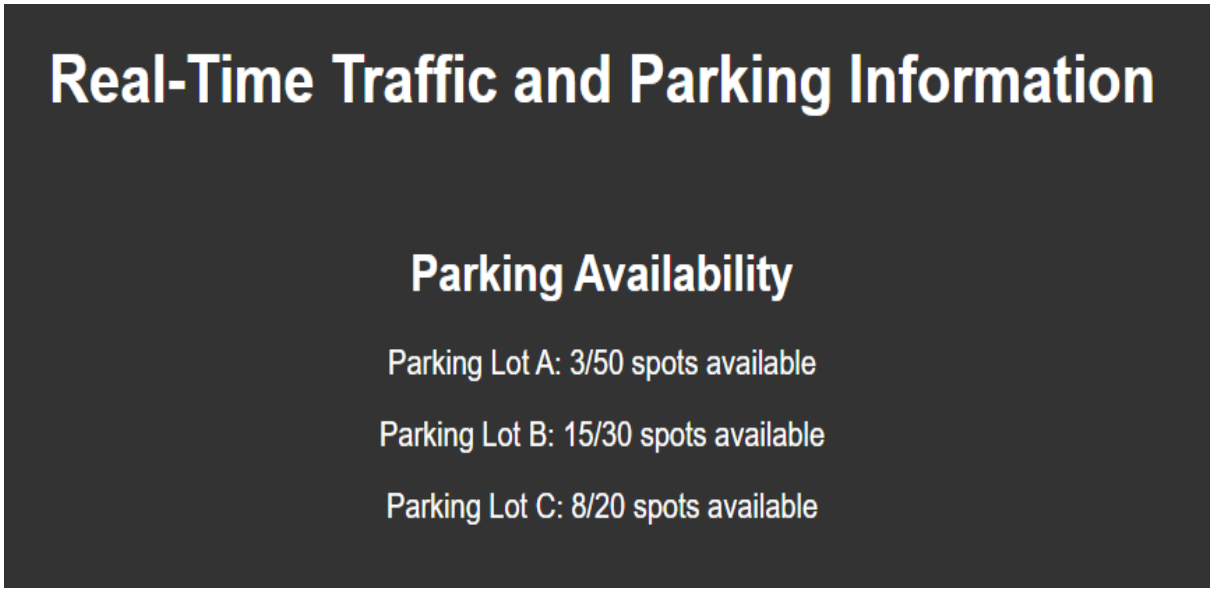
```
// Sample Swift code for real-time data updates in iOS app
func updateParkingInformation() {
let parkingLotA = "Available"
let parkingLotB = "Full"
let parkingLotC = "Available"
parkingStatusLabel.text = "Parking Lot A: \(parkingLotA)"
// Update other UI elements similarly
}
// Simulate data updates every 30 seconds (adjust as needed)
let updateTimer = Timer.scheduledTimer(timeInterval: 30.0, target: self, selector: #selector(updateParkingInformation), userInfo: nil, repeats: true)
```

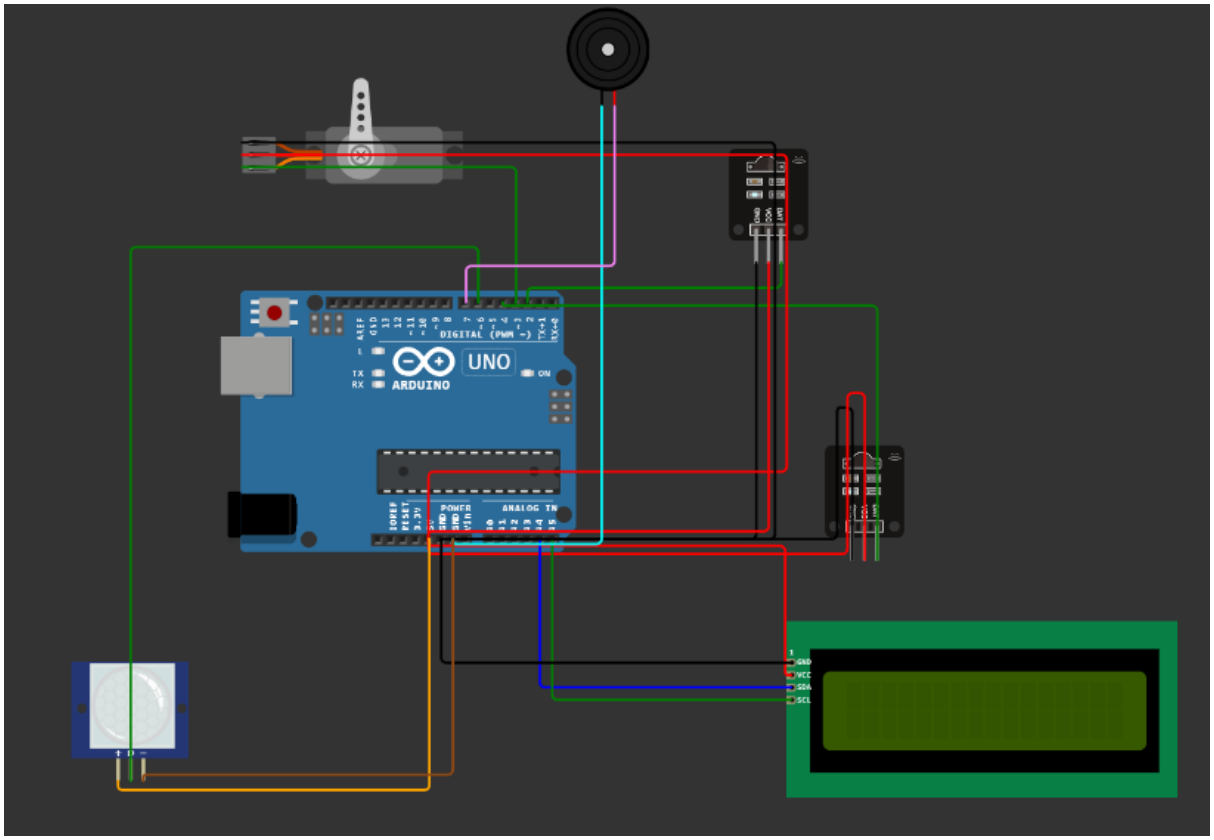
Sketch.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);  // Change the HEX address
#include <Servo.h>
Servo myservo1;
int IR1 = 2;
int IR2 = 4;
int SmokeDetectorPin = 6;  // Digital pin for the smoke detector
int BuzzerPin = 7;         // Digital pin for the buzzer
int Slot = 4;  // Enter Total number of parking Slots
bool flag1 = false;
bool flag2 = false;
unsigned long lastLcdUpdate = 0;  // Variable to track the time of the last LCD update
unsigned long lcdUpdateInterval = 1000;  // Update the LCD every 1000 milliseconds (1 second)
void setup() {
  lcd.begin(16, 2);  // Initialize LCD with 16 columns and 2 rows
  lcd.backlight();
  pinMode(IR1, INPUT);
  pinMode(IR2, INPUT);
  pinMode(SmokeDetectorPin, INPUT);
  pinMode(BuzzerPin, OUTPUT);
  myservo1.attach(3);
  myservo1.write(100);
  lcd.setCursor(0, 0);
  lcd.print("     ARDUINO    ");
  lcd.setCursor(0, 1);
  lcd.print(" PARKING SYSTEM ");
  delay(2000);
  lcd.clear();
  Serial.begin(9600);  // Start serial communication for debugging
}
void loop() {
  if (digitalRead(IR1) == LOW && !flag1) {
    if (Slot > 0) {
      flag1 = true;
      if (!flag2) {
        myservo1.write(0);
        Slot--;
      }
    } else {
      displayMessage("    SORRY :(    ", "  Parking Full  ");
    }
  }
  if (digitalRead(IR2) == LOW && !flag2) {
    flag2 = true;
    if (!flag1) {
      myservo1.write(0);
      Slot++;
    }
  }
  if (flag1 && flag2) {
    delay(1000);
    myservo1.write(100);
    Serial.println("Servo returned to initial position.");
```

```
    flag1 = false;
    flag2 = false;
  }
  // Update the LCD display with a delay
  if (millis() - lastLcdUpdate >= lcdUpdateInterval) {
    updateLcdDisplay();
    lastLcdUpdate = millis();
  }
  // ... (Rest of your code)
}
void updateLcdDisplay() {
  if (digitalRead(SmokeDetectorPin) == HIGH) {
    displayMessage("   WARNING!   ", " Smoke Detected ");
    digitalWrite(BuzzerPin, HIGH);  // Turn on the buzzer
  } else {
    displayMessage("    WELCOME!    ", "Slot Left: " + String(Slot));
    digitalWrite(BuzzerPin, LOW);   // Turn off the buzzer
  }
}
void displayMessage(const char *line1, const String &line2) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(line1);
  lcd.setCursor(0, 1);
  lcd.print(line2);
```

DIAGRAMS AND SCHEMATICS





BENEFITS OF REAL-TIME PARKING AVAILABILITY SYSTEM

The real-time parking availability system can benefit drivers and alleviate parking issues in the following ways:

- Reduced time spent searching for a parking space: Drivers can use the mobile app to view the real-time availability of parking slots in nearby parking lots. This can help drivers to save time and reduce stress.
- Improved traffic flow: The system can help to reduce traffic congestion by directing drivers to parking lots with available spaces.
- Increased parking revenue: Parking lot owners can use the system to track parking occupancy and optimize pricing.

CONCLUSION

This document has presented a comprehensive overview of the Smart Parking project, including its objectives, IoT sensor setup, mobile app development, Raspberry Pi integration, and code implementation. It has also explained how the real-time parking availability system can benefit drivers and alleviate parking issues. The Smart Parking project is a valuable contribution to the field of smart city technologies. It has the potential to significantly improve the parking experience for drivers and reduce traffic congestion. The project is still under development, but it has already made significant progress. The IoT sensor setup and Raspberry Pi integration have been completed, and the mobile app is in the development phase. The next steps for the project are to complete the development of the mobile app and to deploy the system in a real-world environment. Once the system is deployed, it will be possible to evaluate its effectiveness and make further improvements as needed.