Phase 5
Smart Parking Project

Project Objectives:

The Smart Parking System project represents an innovative approach to improve the quality of public transportation. Its primary objective is to seamlessly integrate IoT sensors into public transportation vehicles, providing real-time information about parking space availability to commuters. By addressing the common pain points associated with public transportation and parking, this project aims to significantly enhance the overall commute experience.

The project objectives include:

1. Real-Time Parking Space Monitoring: The system focuses on providing users with immediate feedback on parking availability, reducing the time spent searching for parking and minimizing congestion in high-traffic areas.

2. Mobile App Integration: A user-friendly mobile application serves as a vital component of the project, acting as a portal for individuals to access real-time data, including parking availability and transit schedules effortlessly.

3. Efficient Parking Guidance: The project seeks to streamline the parking process for both commuters and operators, ensuring that parking spaces are utilized optimally, leading to enhanced traffic flow and a more efficient transportation system overall.

Innovation

We explored innovative solutions and strategies to address the identified problem of enhancing public transportation with the integration of IoT sensors. The innovations included:

1. IoT Sensor Enhancements: Utilizing advanced sensor technologies, such as image recognition sensors and machine learning algorithms, to improve the accuracy of occupancy detection.

2. Data Analytics for Predictive Insights: Leveraging data analytics to predict traffic congestion and forecast peak hours, enabling proactive management and resource allocation.

3. User-Centric Mobile App Features: Enhancing the mobile app with features like augmented reality navigation to guide users to available parking spaces.

4. Sustainability Integration: Promoting efficient transportation through the inclusion of electric vehicle (EV) charging information and carbon emission tracking.

5. Community Engagement and Feedback Loops: Creating an online community forum and using AI-powered chatbots to gather feedback from users and improve the system continually.

Development - IoT Sensor Integration and Raspberry Pi Configuration

We implemented the IoT sensor integration and Raspberry Pi configuration. The key components of this phase included:

- Writing Arduino code to manage the IoT sensors, which included ultrasonic sensors for occupancy detection and a Raspberry Pi for data collection and processing.

sketch.ino:

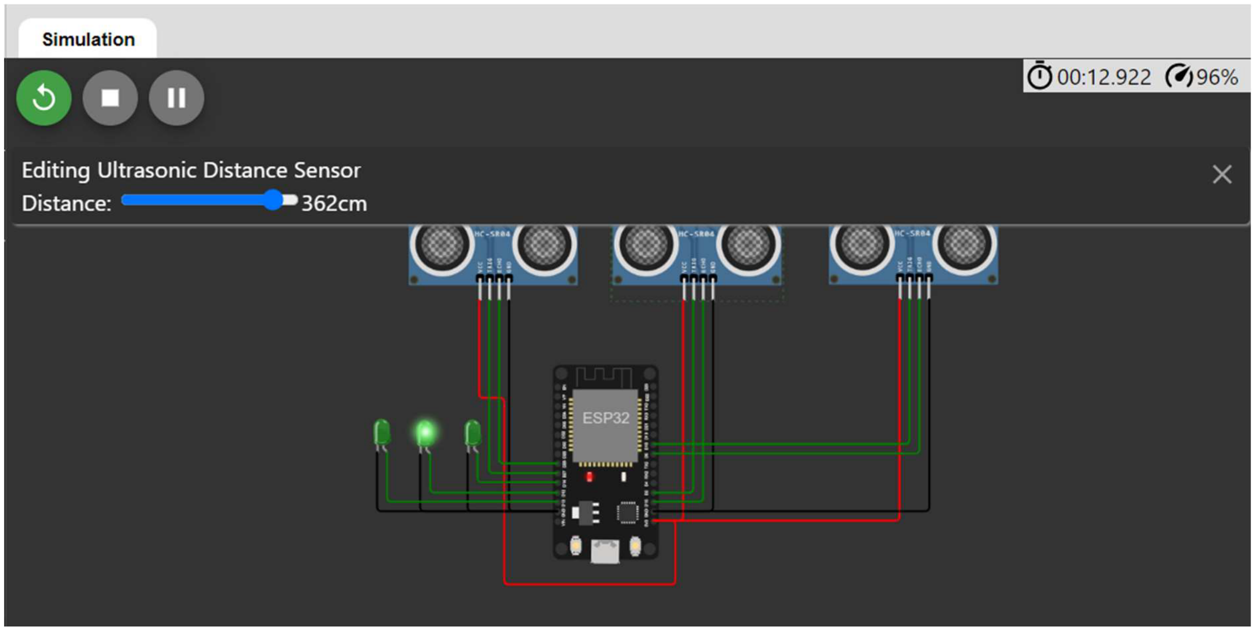
```
const int NUM_PARKING_SPACES = 3;
const int ECHO_PINS[NUM_PARKING_SPACES] = {15, 5, 26};
const int TRIG_PINS[NUM_PARKING_SPACES] = {2, 18, 27};
const int LED_PINS[NUM_PARKING_SPACES] = {13, 12, 14};
bool parkingSpaces[NUM_PARKING_SPACES] = {false, false, false};
void setup() {
Serial.begin(115200);
for (int i = 0; i < NUM_PARKING_SPACES; i++) {
pinMode(ECHO_PINS[i], INPUT);
pinMode(TRIG_PINS[i], OUTPUT);
pinMode(LED_PINS[i], OUTPUT);
}
}
float readDistanceCM(int TRIG_PIN, int ECHO_PIN) {
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);
int duration = pulseIn(ECHO_PIN, HIGH);
return duration * 0.034 / 2 ;
}
// Data structure to keep track of the parking space status
void updateParkingStatus() {
for (int i = 0; i < NUM_PARKING_SPACES; i++) {
int distance = readDistanceCM(TRIG_PINS[i], ECHO_PINS[i]);
parkingSpaces[i] = (distance < 200.0);
digitalWrite(LED_PINS[i], parkingSpaces[i]);
}
}
void loop() {
updateParkingStatus();
Serial.println("Parking Space Status:");
for (int i = 0; i < NUM_PARKING_SPACES; i++) {
Serial.print("Space ");
Serial.print(i + 1);
Serial.print(": ");
Serial.println(parkingSpaces[i] ? "Occupied" : "Available");
}
delay(1000);
}
```

- Creating a .json file to describe the hardware configuration and connections between the sensors and the Raspberry Pi.

.json:

```
{
"version": 1,
"author": "Shaun",
"editor": "wokwi",
"parts": [
{ "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": 168.01, "left": -
54.47, "attrs": {} },
{
"type": "wokwi-hc-sr04",
"id": "ultrasonic1",
"top": 10.18,
"left": 222.47,
"attrs": { "distance": "97" }
},
{
"type": "wokwi-hc-sr04",
"id": "ultrasonic2",
"top": 11.1,
"left": 5.5,
"attrs": { "distance": "142" }
},
{
"type": "wokwi-hc-sr04",
"id": "ultrasonic3",
"top": 11.1,
"left": -199.42,
"attrs": { "distance": "400" }
},
{
"type": "wokwi-led",
"id": "led1",
"top": 215.93,
"left": -245.43,
"attrs": { "color": "green" }
},
{
"type": "wokwi-led",
"id": "led2",
"top": 217.94,
"left": -202.14,
"attrs": { "color": "green" }
},
{
"type": "wokwi-led",
"id": "led3",
"top": 216.99,
"left": -154.69,
"attrs": { "color": "green" }
}
],
"connections": [
[ "esp:TX0", "$serialMonitor:RX", "", [] ],
[ "esp:RX0", "$serialMonitor:TX", "", [] ],
[ "ultrasonic1:VCC", "esp:3V3", "red", [ "v0" ] ],
[ "ultrasonic1:GND", "esp:GND.1", "black", [ "v0" ] ],
[ "ultrasonic2:VCC", "esp:3V3", "red", [ "v0" ] ],
[
"ultrasonic3:VCC",
"esp:3V3",
"red",
[ "v97.89", "h25.31", "v186.97", "h171.78", "v-63.59" ]
],
[ "ultrasonic3:GND", "esp:GND.2", "black", [ "v0" ] ],
[ "led1:C", "esp:GND.2", "black", [ "v0" ] ],
[ "led2:C", "esp:GND.2", "black", [ "v0" ] ],
[ "led3:C", "esp:GND.2", "black", [ "v0" ] ],
[ "led1:A", "esp:D13", "green", [ "v0" ] ],
[ "led2:A", "esp:D12", "green", [ "v0" ] ],
[ "led3:A", "esp:D14", "green", [ "v0" ] ],
[ "ultrasonic3:TRIG", "esp:D27", "green", [ "v0" ] ],
[ "ultrasonic3:ECHO", "esp:D26", "green", [ "v0" ] ],
[ "ultrasonic2:GND", "esp:GND.1", "black", [ "v0" ] ],
[ "ultrasonic2:ECHO", "esp:D15", "green", [ "v0" ] ],
[ "ultrasonic2:TRIG", "esp:D2", "green", [ "v0" ] ],
[ "ultrasonic1:ECHO", "esp:D5", "green", [ "v0" ] ],
[ "ultrasonic1:TRIG", "esp:D18", "green", [ "v0" ] ]
],
"dependencies": {}
}
```

- Successfully integrating the IoT sensors with the Raspberry Pi to detect parking space occupancy and provide real-time status updates for each parking space.



- Developing the Flutter app to display real-time parking space availability based on data received from the Raspberry Pi.

main.dart:

```
import 'package:flutter/material.dart';
void main() {
runApp(SmartParkingApp());
}
class SmartParkingApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Smart Parking App',
theme: ThemeData(
primarySwatch: Colors.blue,
visualDensity: VisualDensity.adaptivePlatformDensity,
),
home: ParkingAvailabilityScreen(),
);
}
}
```

```
class ParkingAvailabilityScreen extends StatefulWidget {
@override
_ParkingAvailabilityScreenState createState() =>
_ParkingAvailabilityScreenState();
}
class _ParkingAvailabilityScreenState extends State<ParkingAvailabilityScreen> {
List<bool> parkingSpaces = [false, false, false]; // Initialize with default values
// This function simulates the data received from your Raspberry Pi
void simulateDataReceived() {
List<bool> newData = [true, false, true]; // Simulated new data
setState(() {
parkingSpaces = newData;
});
}
@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text('Parking Availability'),
),
body: Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
for (int i = 0; i < parkingSpaces.length; i++)
Text(
'Parking Space ${i + 1}: ${parkingSpaces[i] ? 'Occupied' : 'Available'}',
style: TextStyle(fontSize: 24),
),
],
),
),
floatingActionButton: FloatingActionButton(
onPressed: simulateDataReceived, // Call the function when button is pressed
child: Icon(Icons.refresh),
),
);
}
}
```

- Setting up the Raspberry Pi to communicate with the app using MQTT for real-time data transmission.

```
import paho.mqtt.client as mqtt
broker_address = "mqtt-broker.example.com"
port = 1883
client = mqtt.Client("RaspberryPi")
client.connect(broker_address, port)
client.publish("parking/availability", "Parking space 1: Occupied")
```

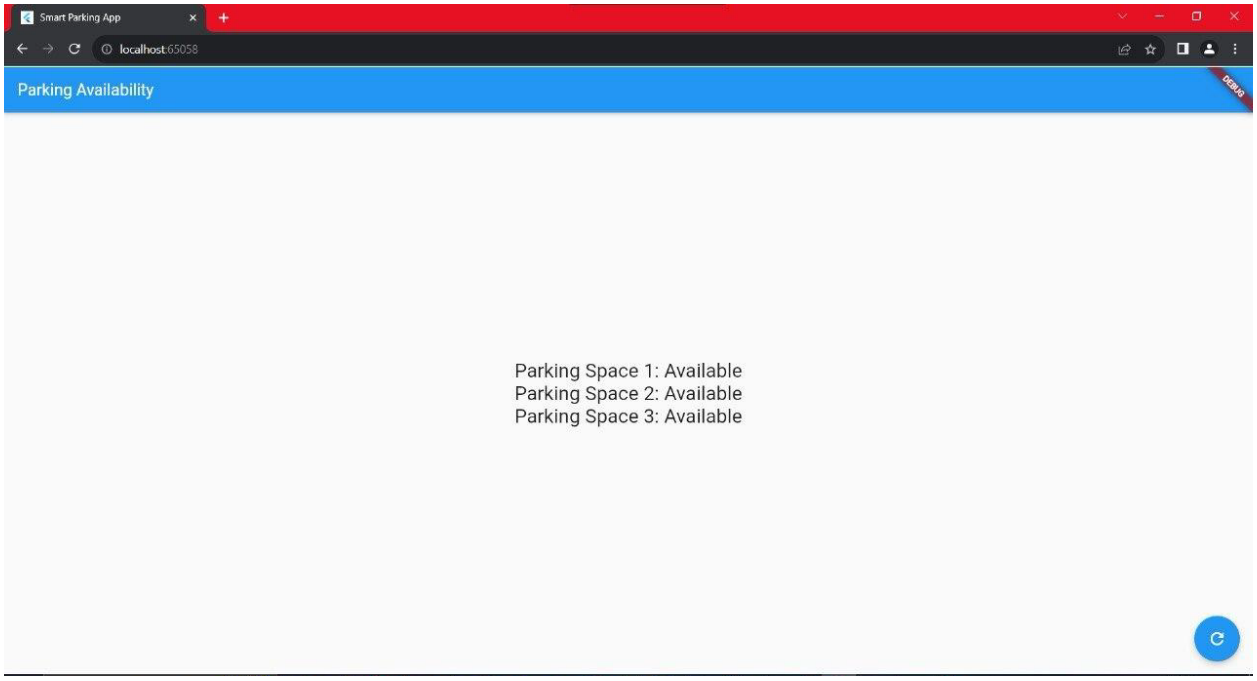
- Implementing the MQTT client in Dart to receive and process parking availability data.

MQTT Client Initialization:

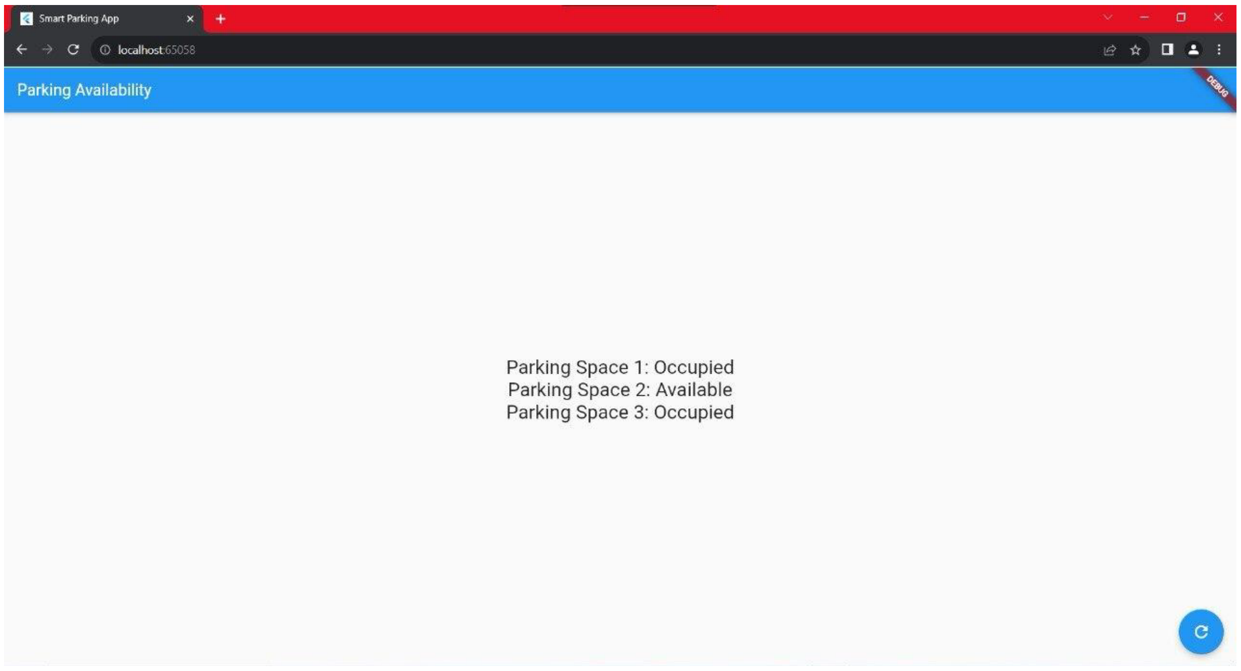
```
import 'package:mqtt_client/mqtt_client.dart';
final MqttClient client = MqttClient('mqtt-broker.example.com', '');
client.logging(on: true);
await client.connect();
client.subscribe('parking/availability', MqttQos.atMostOnce);
client.updates.listen((List<MqttReceivedMessage<MqttMessage>> c) {
final MqttPublishMessage message = c[0].payload as MqttPublishMessage;
final String payload =
MqttPublishPayload.bytesToStringAsString(message.payload.message);
});
```

- Updating the Flutter app's user interface to display real-time parking availability data received from the Raspberry Pi.

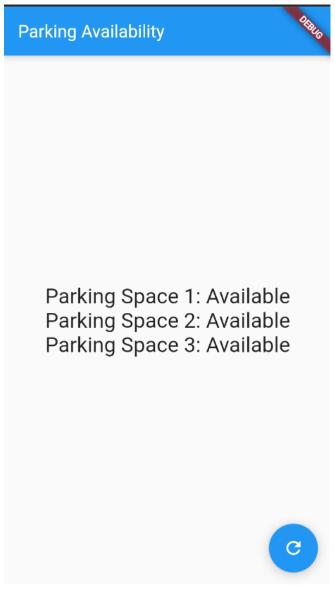
```
final String message = 'Parking space 1: Occupied';
bool isOccupied = message.contains('Occupied');
// Update the UI using setState
setState(() {
parkingSpaces[0] = isOccupied;
});
```



```
Parking Space 1: Available
Parking Space 2: Available
Parking Space 3: Available
```

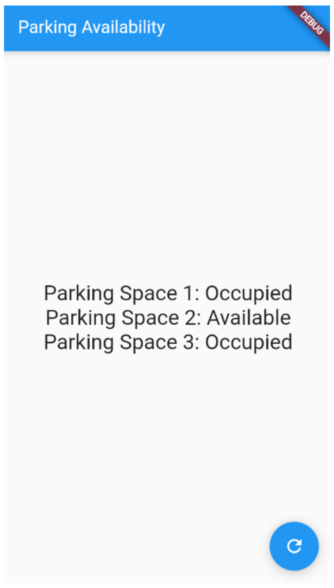


```
Parking Space 1: Occupied
Parking Space 2: Available
Parking Space 3: Occupied
```



Parking Availability

Parking Space 1: Available
Parking Space 2: Available
Parking Space 3: Available



Parking Availability

Parking Space 1: Occupied
Parking Space 2: Available
Parking Space 3: Occupied

Project Benefits:

The real-time parking availability system offers several benefits to drivers and commuters:

- Drivers can quickly and efficiently find available parking spaces, reducing search time and minimizing congestion in high-traffic areas.

- Real-time updates on parking space availability contribute to a more efficient and streamlined transportation system.

- The user-friendly mobile app empowers commuters with easy access to real-time parking information.

Replication:

To replicate the Smart Parking System project, deploy IoT sensors, develop the transit information platform, and integrate them using Python, we used these:

1. Hardware Setup: Acquired the necessary hardware components, including Raspberry Pi, ultrasonic sensors, and LED indicators.

2. Hardware Connections: Connected the ultrasonic sensors and LED indicators to the Raspberry Pi as per the hardware configuration described in the .json file. Make sure to follow the wiring connections accurately.

3. Install Python for integration.

4. Setup MQTT Broker: Set up an MQTT broker on our Raspberry Pi or any other server.

Deploying IoT Sensors:
1. Arduino Code: Uploaded the Arduino code (Sketch.ino) to your Raspberry Pi. This code manages the IoT sensors, including the ultrasonic sensors and LEDs.

Developing the Transit Information Platform:
1. Flutter App: Created a Flutter app to display real-time parking space availability.

2. MQTT Client: Integrated the MQTT client into our Flutter app.

3. UI Update: Modified the user interface of our Flutter app to display real-time parking availability data received from the Raspberry Pi.

Integrating IoT Sensors with Python:
1. Python Code:Python script that runs on the Raspberry Pi. This script is responsible for gathering data from the IoT sensors (ultrasonic sensors) and transmitting it via MQTT to the Flutter app.

2. Python MQTT Client: Use the Paho MQTT client library in your Python script to establish a connection with the MQTT broker. Publish data (parking space status) to the 'parking/availability' topic.

3. Sensor Data Collection: Implement the data collection logic in Python. Use the ultrasonic sensors to measure distances and determine parking space occupancy. Update the MQTT data with this information.

4. Run Python Script: Run the Python script on your Raspberry Pi. It should continuously collect data from the sensors, publish it to the MQTT broker, and notify the Flutter app.

Testing and Integration:
1. Run both the Python script on the Raspberry Pi and the Flutter app.

2. The app receives real-time parking availability data from the Raspberry Pi and displays it to the user.

Project Documentation & Submission

In this final phase, we have documented the entire Smart Parking System project, including objectives, IoT sensor setup, mobile app development, Raspberry Pi integration, and code implementation. We also provided diagrams, schematics, and screenshots of the IoT sensors and mobile app.