Smart Parking
Internet Of Things
Phase 5

Project Overview:

The Smart Parking System is an IoT-based project designed to address the challenges associated with parking in urban areas. This project aims to improve parking management, enhance the user experience, and optimise parking space utilisation by providing real-time information about parking space availability to drivers. This system leverages a combination of hardware and software components to achieve its objectives.

Project Objectives:

- This Provide drivers with real-time information about available parking spaces.
- This Helps drivers to find parking spaces more quickly and easily. This will be done by providing real-time information about parking availability, as well as guidance to available spaces.
- Reducing traffic congestion. By helping drivers to find parking spaces more quickly and easily, smart parking systems will reduce the amount of time that drivers spend circling around looking for parking. This will lead to reduced traffic congestion and emissions.
- Improvement parking management. Smart parking systems provide parking operators with real-time data about parking occupancy and usage. This data will be used to improve parking management decisions.
- Increasing revenue. Smart parking systems can help parking operators to increase revenue by providing drivers with more convenient and efficient parking options. For example, drivers are willing to pay more for parking if they can reserve a space in advance or pay for parking with their mobile phone.

IoT sensor setup:

● Sensor Introduction:

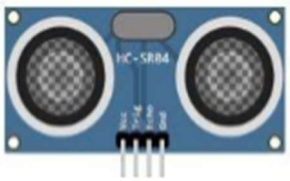
This is a simple implementation of a sensor data collection system using Raspberry Pi Pico. It utilises ultrasonic sensors to send the occupancy data to a cloud server. This project can be extended for applications such as parking management, occupancy detection, and more.

● Working Principle:

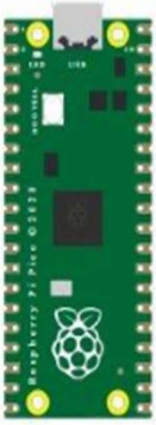
The code sets up three ultrasonic sensors, each consisting of a trigger pin and an echo pin, connected to the Raspberry Pi Pico. By emitting ultrasonic waves and measuring the time taken for the waves to bounce back, the sensors determine the distances of nearby objects. If the measured distance is less than a predefined threshold, it is considered as occupancy. The script then sends this occupancy data to a cloud server using the Wi-Fi module, allowing real-time monitoring and analysis of the sensor data for various applications

Components :

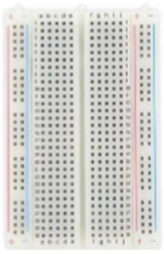
- HC-SR04 Ultrasonic Distance Sensor



- Raspberry Pi Pico



- Mini BreadBoard



ESP32

- Jumper wires

PIN Connection :

For the first sensor:

- **VCC Pin**: Connect to the 5V pin on the Raspberry Pi Pico (usually labelled as VBUS or VSYS).
- **GND Pin**: Connect to any of the ground pins on the Raspberry Pi Pico (labelled as GND).
- **Trigger pin**: Connect to Pin 0 (GP0) on the Raspberry Pi Pico.
- **Echo pin**: Connect to Pin 1 (GP1) on the Raspberry Pi Pico.

For the second sensor:

- **VCC Pin**: Connect to the 5V pin on the Raspberry Pi Pico.
- **GND Pin**: Connect to any of the ground pins on the Raspberry Pi Pico.
- **Trigger pin**: Connect to Pin 2 (GP2) on the Raspberry Pi Pico.
- **Echo pin**: Connect to Pin 3 (GP3) on the Raspberry Pi Pico.

For the third sensor:

- **VCC Pin**: Connect to the 5V pin on the Raspberry Pi Pico.
- **GND Pin**: Connect to any of the ground pins on the Raspberry Pi Pico.
- **Trigger pin**: Connect to Pin 4 (GP4) on the Raspberry Pi Pico.
- **Echo pin**: Connect to Pin 5 (GP5) on the Raspberry Pi Pico.

Steps involved:

IoT sensor setup in a smart parking project involves deploying sensors to detect the occupancy status of parking spaces. Here's a brief overview of the process:

- **Sensor Selection**: Suitable IoT sensors is chosen for detecting parking space occupancy. Common options include ultrasonic sensors, magnetic sensors, infrared sensors, or cameras.

- **Sensor Placement**: The selected sensors are installed in individual parking spaces, ensuring they are positioned to accurately detect the presence or absence of vehicles.

- **Wiring and Power**: The sensor is connected to a power source and, if necessary, to a communication module. Depending on the sensor type, this involves wiring the sensors to a central control unit or power source.

- **Communication Module**: If the sensors are not directly connected to the central control unit (e.g., a Raspberry Pi), Communication modules(e.g., Wi-Fi, LoRa, or Zigbee) are used to relay sensor data to the central system.

- **Calibration**: The sensors are Calibrated to ensure they provide accurate data. This will involve adjusting sensitivity settings or fine-tuning sensor parameters.

- **Testing**: The sensors are tested to confirm that they reliably detect vehicle presence and absence. Check for any false positives or false negatives and make necessary adjustments.

- **Data Transmission**: Ensure that sensor data is transmitted to the central control unit or gateway in a consistent and timely manner.

The IoT sensor setup is a critical component of a smart parking system, as it provides real-time data on parking space availability. Accurate and reliable sensors are essential for the successful operation of the system, allowing users to access up-to-date parking information via a mobile app or other interfaces.

Mobile app development and Raspberry Pi integration:

Source Code Example:

Here's an example of how you can start building your React Native mobile app to display parking availability data. Please note that this is a simplified example, and you'll need to adapt it to your specific requirements and integrate it with your Python code that communicates with the Raspberry Pi.

1. Install React Native: First, make sure you have React Native set up on your development environment. You can follow the official React Native documentation for installation instructions: https://reactnative.dev/docs/environment-setup

2. Create a New React Native Project Use the following command to create a new React Native project: npx react-native init ParkingAvailabilityApp

3. Design the User Interface: Create components and screens to display parking availability data. You can use libraries like react-navigation for navigation between screens and react-native-elements for UI components.

4. Fetch and Display Data: Use the fetch API or a library like axios to make HTTP requests to your Python server running on the Raspberry Pi

Here's a simplified example of fetching data and displaying it:

```
import React, { useState, useEffect } from 'react';
import { View, Text } from 'react-native';
function ParkingAvailabilityScreen() {
        const [availabilityData, setAvailabilityData] = useState(null);
        useEffect(() => {
         // Make an HTTP request to your Python server (replace
with your server URL)
        fetch('http://your-raspberrypi-server:port/availability')
        .then((response) => response.json())
        .then((data) => setAvailabilityData(data))
        .catch((error) => console.error(error));
}, []);
 return (
<View>
{availabilityData ? (
<Text> Parking Availability: {availabilityData.available}</Text>
) : (
<Text> Loading data...</Text>
 )}
<View>
 );
}
 export default ParkingAvailabilityScreen;
```

5. Run the App: Use the following commands to run your React Native app on connected device or emulator:

```
npx react-native start
npx react-native run-android  # For Android
npx react-native run-ios  # For iOS
```

6. Further Integration: Your Python code running on the Raspberry Pi should expose an API endpoint that provides real-time parking availability data. Make sure to replace the URL in the fetch request with the appropriate endpoint from your Python server. Remember that this is a basic example, and in a real-world scenario, you would want to handle error cases, optimize data fetching, and implement user-friendly UI. Additionally, security considerations should be taken into account when connecting to your Raspberry Pi. By following this example and customizing it to your project's needs, you can create a mobile app using React Native to display real-time parking availability data from your Raspberry Pi server.

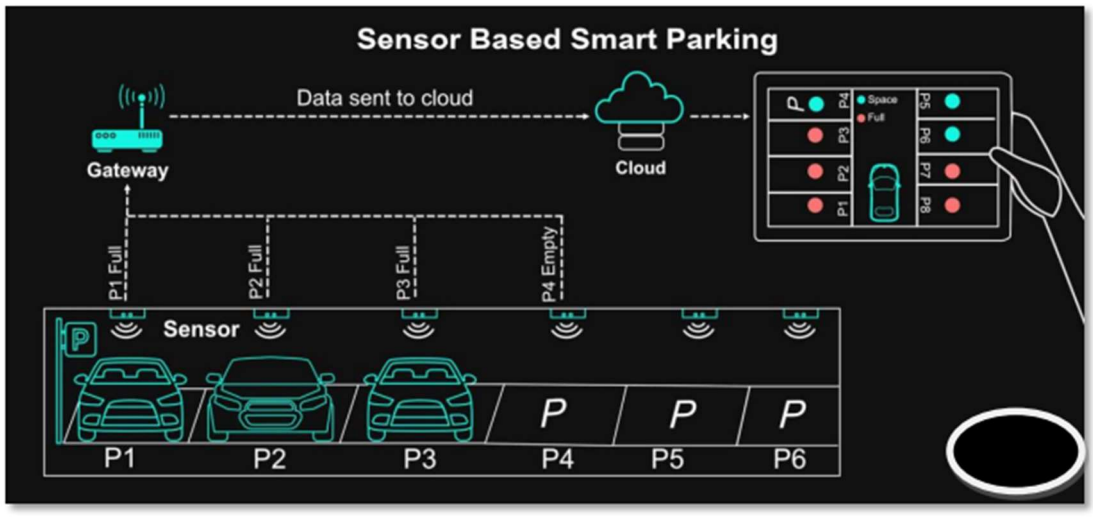
Kivy Framework For The Mobile App:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.network.urlrequest import UrlRequest
import json
class ParkingAvailabilityApp(App):
    def build(self):
        self.layout = BoxLayout(orientation='vertical')
        self.availability_label = Label(text="Parking Availability:
    Loading...")
        self.update_button = Button(text="Refresh Data")
        self.update_button.bind(on_press=self.update_availabil
    ity)
        self.layout.add_widget(self.availability_label)
        self.layout.add_widget(self.update_button)
```

```
        return self.layout
    def update_availability(self, instance):
        url = "https://your-api-endpoint.com/parking-availability"
        UrlRequest(url, self.parse_availability)
    def parse_availability(self, request, result):
        try:
            data = json.loads(result)
            available = data.get("available")
            last_updated = data.get("lastUpdated")
            self.availability_label.text = f"Parking Availability:
        {available} spots available\nLast updated:
        {last_updated}"
        except Exception as e:
            self.availability_label.text = "Failed to fetch data"
if __name__ == '__main__':
    ParkingAvailabilityApp().run()
```

Diagram :

Example output diagram:



Code implementation:

MicroPython Code

```
import machine import
time import urequests
# Define the pin configuration for each sensor sensor_pins = [(0, 1),
(2, 3), (4, 5)]
BEECEPTOR_ENDPOINT =
"https://cloudplatform.free.beeceptor.com"
# Function to measure the distance from the ultrasonic sensor def
measure_distance(trigger_pin, echo_pin): trigger =
machine.Pin(trigger_pin,
machine.Pin.OUT) echo = machine.Pin(echo_pin, machine.Pin.IN)
 trigger = machine.Pin(trigger_pin, machine.Pin.OUT) echo =
machine.Pin(echo_pin,
machine.Pin.IN)
 trigger.low() time.sleep_us(2)
trigger.high() time.sleep_us(10) trigger.low()
 while echo.value() == 0:
 signaloff = time.ticks_us() while echo.value()
== 1: signalon = time.ticks_us()
 timepassed = signalon - signal off distance = (timepassed *
0.0343) / 2 return distance
# Function to send data to Beeceptor endpoint def
send_data_to_beeceptor(data): headers =
{'Content-Type': 'application/json'} json_data = {'occupancy': data}
try: response =
urequests.post(BEECEPTOR_ENDPOINT, json=json_data,
headers=headers)
 print("Data sent successfully") print(response.text) except
Exception as
e: print("An error occurred while sending data:", e)
```

```
# Main loop to continuously measure and send data from all three
sensors while True:
 sensor_data = [] for trigger_pin, echo_pin in sensor_pins:
 distance = measure_distance(trigger_pin, echo_pin)
print(f"Distance:
{distance} cm from Sensor {sensor_pins.index((trigger_pin,
echo_pin)) + 1}")
occupancy = 1 if distance < 10 else 0
sensor_data.append(occupancy)
send_data_to_beeceptor(sensor_data) time.sleep(5) # Adjust the
sleep time as needed
```

Code Explanation:

- Definition of Pin Configuration: Three sets of trigger and echo pins are defined to connect the ultrasonic sensors to the Raspberry Pi Pico.
- Distance Measurement Function: The 'measure_distance' function calculates the distance based on the time taken for the ultrasonic signal to return.
- Data Sending Function: The 'send_data_to_beeceptor' function sends the occupancy data to the specified Beeceptor Endpoint.
- Main Loop: The main loop continuously measures data from all three sensors, determines the occupancy status based on the measured distance, and then sends this data to the Beeceptor endpoint.

Usage Scenario:

- Sensor Installation: Install the ultrasonic sensors at strategic locations within the parking lot, ensuring comprehensive coverage of the parking area.
- Raspberry Pi Pico Integration: Connect the ultrasonic sensors to the Raspberry Pi Pico, ensuring the

correct mapping of trigger and echo pins as outlined in the script.

- Data Collection and Transmission: Run the script on the Raspberry Pi Pico to collect and process realtime occupancy data. The script measures the distance between the sensor and any parked vehicle and
  sends this data to the cloud platform through the Beeceptor endpoint.
- Cloud-Based Analysis: Receive the occupancy data on the cloud platform, enabling parking lot
  managers to monitor and analyze parking space utilization trends. This data can be used to identify
  parking patterns, optimize parking allocation, and provide real-time parking availability updates to users.
- Optimized Parking Experience: With real-time occupancy data, users can easily locate available
  parking spaces, reducing the time spent searching for parking.

Python Code :

```
import time
Vehicle_Number=['XXXX-XX-XXXX']
Vehicle_Type=['Bike']
vehicle_Name=['Intruder']
Owner_Name=['Unknown']
Date=['22-22-3636']
Time=['22:22:22']
bikes=100
cars=250
bicycles=78
def main():
 global bikes,cars,bicycles
 try:
```

```
while True:
print("-----------------------------------------------------------------")
print("\t\tParking Management System")
print("-----------------------------------------------------------------")
print("1.Vehicle Entry")
print("2.Remove Entry" )
print("3.View Parked Vehicle ")
print("4.View Left Parking Space ")
print("5.Amount Details ")
print("6.Bill")
print("7.Close Programme ")
print("+---------------------------------------+")
ch=int(input("\tSelect option:"))
if ch==1:
no=True
while no==True:
Vno=input("\tEnter vehicle number (XXXX-XX-XXXX) - ").upper()
if Vno=="":
print("###### Enter Vehicle No. ######")
elif Vno in Vehicle_Number:
print("###### Vehicle Number Already Exists")
elif len(Vno)==12:
no=not True
Vehicle_Number.append(Vno)
else:
print("###### Enter Valid Vehicle Number ######")
typee=True
while typee==True:
Vtype=str(input("\tEnter vehicle
type(Bicycle=A/Bike=B/Car=C):")).lower()
if Vtype=="":
print("###### Enter Vehicle Type ######")
elif Vtype=="a":
Vehicle_Type.append("Bicycle")
bicycles-=1
typee=not True
elif Vtype=="b":
```

```
Vehicle_Type.append("Bike")
bikes-=1
typee=not True
elif Vtype=="c":
Vehicle_Type.append("Car")
cars-=1
typee=not True
else:
print("###### Please Enter Valid Option ######")
name=True
while name==True:
vname=input("\tEnter vehicle name - ")
if vname=="":
print("########Please Enter Vehicle Name ########")
else:
vehicle_Name.append(vname)
name=not True
o=True
while o==True:
OName=input("\tEnter owner name - ")
if OName=="":
print("###### Please Enter Owner Name ######")
else:
Owner_Name.append(OName)
o=not True
d=True
while d==True:
date=input("\tEnter Date (DD-MM-YYYY) - ")
if date=="":
print("###### Enter Date ######")
elif len(date)!=10:
print("###### Enter Valid Date ######")
else:
Date.append(date)
d=not True
t=True
while t==True:
```

```
time=input("\tEnter Time (HH:MM:SS) - ")
if t=="":
print("###### Enter Time ######")
elif len(time)!=8:
print("###### Please Enter Valid Date ######")
else:
Time.append(time)
t=not True
print("\n.....Record detail saved.....")
elif ch==2:
no=True
while no==True:
Vno=input("\tEnter vehicle number to Delete(XXXX-XX-XXXX) -
").upper()
if Vno=="":
print("###### Enter Vehicle No. ######")
elif len(Vno)==12:
if Vno in Vehicle_Number:
i=Vehicle_Number.index(Vno)
Vehicle_Number.pop(i)
Vehicle_Type.pop(i)
vehicle_Name.pop(i)
Owner_Name.pop(i)
Date.pop(i)
Time.pop(i)
no=not True
print("\n.....Removed Sucessfully.....")
elif Vno not in Vehicle_Number:
print("###### No Such Entry ######")
else:
print("Error")
else:
print("###### Enter Valid Vehicle Number ######")
elif ch==3:
count=0
print("--------------------------------------------------------------------------------")
print("\t\t\t\tParked Vehicle")
```

```
 print("--------------------------------------------------------------------------------
")
 print("Vehicle No.\tVehicle Type Vehicle Name\t Owner Name\t Date\t
\tTime")
 print("--------------------------------------------------------------------------------")
 for i in range(len(Vehicle_Number)):
 count+=1
print(Vehicle_Number[i],"\t ",Vehicle_Type[i],"\t ",vehicle_Name[i],"\t
",Owner_Name[i]," " ,Date[i]," ",Time[i])
 print("--------------------------------------------------------------------------------")
 print("---------------------- Total Records - ",count,"-----------------------------
----")
 print("--------------------------------------------------------------------------------")
 elif ch==4:
 print("--------------------------------------------------------------------------------")
 print("\t\t\t\tSpaces Left For Parking")
 print("--------------------------------------------------------------------------------")
 print("\tSpaces Available for Bicycle - ",bicycles)
 print("\tSpaces Available for Bike - ",bikes)
 print("\tSpaces Available for Car - ",cars)
 print("--------------------------------------------------------------------------------")
 elif ch==5:
 print("--------------------------------------------------------------------------------")
 print("\t\t\t\tParking Rate")
 print("--------------------------------------------------------------------------------")
 print("*1.Bicycle Rs20 / Hour")
 print("*2.Bike Rs40/ Hour")
 print("*3.Car Rs60/ Hour")
 print("--------------------------------------------------------------------------------")
 elif ch==6:
 print("..................................... Generating
Bill.....................................")
 no=True
 while no==True:
 Vno=input("\tEnter vehicle number to Delete(XXXX-XX-XXXX) -
").upper()
 if Vno=="":
```

```
print("###### Enter Vehicle No. ######")
elif len(Vno)==12:
if Vno in Vehicle_Number:
i=Vehicle_Number.index(Vno)
no=not True
elif Vno not in Vehicle_Number:
print("###### No Such Entry ######")
else:
print("Error")
else:
print("###### Enter Valid Vehicle Number ######")
print("\tVehicle Check in time - ",Time[i])
print("\tVehicle Check in Date - ",Date[i])
print("\tVehicle Type - ",Vehicle_Type[i])
inp=True
amt=0
while inp==True:
hr=input("\tEnter No. of Hours Vehicle Parked - ").lower()
if hr=="":
print("###### Please Enter Hours ######")
elif int(hr)==0 and Vehicle_Type[i]=="Bicycle":
amt=20
inp=not True
elif int(hr)==0 and Vehicle_Type[i]=="Bike":
amt=40
inp=not True
elif int(hr)==0 and Vehicle_Type[i]=="Car":
amt=60
inp=not True
elif int(hr)>=1:
if Vehicle_Type[i]=="Bicycle":
amt=int(hr)*int(20)
inp=not True
elif Vehicle_Type[i]=="Bike":
amt=int(hr)*int(40)
inp=not True
elif Vehicle_Type[i]=="Car":
```

```
 amt=int(hr)*int(60)
 inp=not True
 print("\t Parking Charge - ",amt)
 ac=18/100*int(amt)
 print("\tAdd. charge 18 % - ",ac)
 print("\tTotal Charge - ",int(amt)+int(ac))
 print(".....Thank you for using our
service……")
 a=input("\tPress Any Key to Proceed - ")
 elif ch==7:
 print(".....Thank you for using our
service.....")
 print(" **********(: Bye Bye :)**********")
 break
 quit
 except:
 main()
main()
```

Output for program:

```
Parking Management System
1.Vehicle Entry
2.Remove Entry
3.View Parked Vehicle
4.View Left Parking Space
5.Amount Details
6.Bill
7.Close Programme
+-----------------------------------------+
Select option: 1
Enter vehicle number (XXXX-XX-XXXX) - KA-01-HH-1234
Enter vehicle type(Bicycle=A/Bike=B/Car=C): b
Enter vehicle name - Honda Activa
Enter owner name - John Doe
Enter Date (DD-MM-YYYY) - 20-08-2023
Enter Time (HH:MM:SS) - 14:30:00
```

.....Record detail
saved.....
Select option: 3
Parked Vehicle
Vehicle No. Vehicle Type Vehicle Name Owner Name Date Time
KA-01-HH-1234 Bike Honda Activa John Doe 20-08-2023 14:30:00
---------- Total Records - 1 ----------
--

Select option: 6
..... Generating Bill
.....
Enter vehicle number to Delete(XXXX-XX-XXXX) - KA-01-HH-1234
Vehicle Check in time - 14:30:00
Vehicle Check in Date - 20-08-2023
Vehicle Type - Bike
Enter No. of Hours Vehicle Parked - 1
Parking Charge - 40
Add. charge 18 % - 7.2
Total Charge - 47.2
.....Thank you for using our
service.....
Press Any Key to Proceed

Benefits:

- Reduced time spent searching for parking: By providing real-time information about the availability of parking spaces, these systems will help drivers to find a spot more quickly and easily, reducing the amount of time they spend circling around a block or driving in and out of parking garages.
- Reduced fuel consumption: By helping drivers to find parking spots more quickly, real-time parking availability systems will also help to reduce fuel consumption. This is because drivers spend less time driving around looking for a place to park.
- Reduced traffic congestion: Real-time parking availability systems will also help to reduce traffic congestion. This is because drivers

are less likely to be circling around looking for a place to park, which can lead to congestion and delays.
- Reduced frustration: Finding a parking spot is a frustrating experience, especially in crowded urban areas. Real-time parking availability systems will help to reduce frustration by providing drivers with the information they need to find a spot quickly and easily.
- Improved air quality: Real-time parking availability systems can also help to improve air quality. This is because they can help to reduce the amount of time that drivers spend idling their vehicles while looking for a place to park.

Conclusion:

Real-time parking availability systems are a valuable tool for drivers, cities, and businesses. They offer a wide range of benefits, including:

- For drivers:
    - Reducing time spent searching for parking
    - Reducing fuel consumption
    - Reducing frustration
    - Improving air quality
    - Increasing convenience and efficiency
- For cities:
    - Reducing traffic congestion
    - Improving parking management
    - Increasing revenue from parking fees
    - Improving air quality
    - Enhancing urban planning
- For businesses:
    - Increasing customer satisfaction
    - Improving employee productivity
    - Reducing parking costs
    - Enhancing marketing and branding opportunities

Real-time parking availability systems are becoming increasingly widespread, as cities and businesses recognize the many benefits they offer. As these systems become more sophisticated and integrated with other smart city initiatives, they are likely to play an even more important role in improving the transportation and parking experience for everyone.

In addition to the benefits listed above, real-time parking availability systems can also help to reduce greenhouse gas emissions and other air pollutants. This is because they help drivers to find parking spots more quickly and easily, which reduces the amount of time they spend idling their vehicles. Real-time parking availability systems can also help to improve safety by reducing the number of vehicles circling around looking for parking.

Overall, real-time parking availability systems are a valuable tool that can provide a wide range of benefits to drivers, cities, and businesses. As these systems become more widespread and sophisticated, they are likely to play an even more important role in improving the transportation and parking experience for everyone.