**IOT BASED SMART PARKING SYSTEM**

**CODE:**

Import Kivy libraries

From kivy.app import App

From kivy.uix.boxlayout import BoxLayout

From kivy.uix.label import Label

From kivy.uix.button import Button

Create a simple parking app

Class SmartParkingApp(App):

Def build(self):

Main layout

Layout = BoxLayout(orientation=’vertical’, padding=10, spacing=10)

Title label

Title\_label = Label(text=’Smart Parking App’, size\_hint=(1, 0.1))

Parking information

Parking\_info\_label = Label(text=’Available parking spots: 10’, size\_hint=(1, 0.1))

Reserve button

Reserve\_button = Button(text=’Reserve Parking Spot’, size\_hint=(1, 0.1)) Reserve\_button.bind(on\_press=self.reserve\_parking)

Status label

Self.status\_label = Label(text=’’, size\_hint=(1, 0.1))

Add widgets to the layout

Layout.add\_widget(title\_label)

Layout.add\_widget(parking\_info\_label)

Layout.add\_widget(reserve\_button)

Layout.add\_widget(self.status\_label)

Return layout

Def reserve\_parking(self, instance):

Placeholder function to simulate parking reservation

Self.status\_label.text = ‘Parking spot reserved!’

Run the app

If\_\_name\_\_== ‘\_\_ main\_\_ ’:

SmartParkingApp().run()

**CODE FOR ULTRASONIC SENSOR**

import RPi.GPIO as GPIO

import time

import requests

GPIO pins for the ultrasonic sensor

TRIG = 23

ECHO = 24

Set up GPIO mode and pins

GPIO.setmode(GPIO.BCM)

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

def measure\_distance():

Trigger the ultrasonic sensor

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

Wait for the ECHO pin to go high

while GPIO.input(ECHO) == 0:

pulse\_start\_time = time.time()

Wait for the ECHO pin to go low

while GPIO.input(ECHO) == 1:

pulse\_end\_time = time.time()

Calculate distance based on the time taken for the pulse

pulse\_duration = pulse\_end\_time - pulse\_start\_time

distance = (pulse\_duration \* 34300) / 2 **# Speed of sound is 343 m/s**

return distance

try:

while True:

distance = measure\_distance()

print(f"Distance: {distance} cm")

Send the data to an IoT platform (e.g., ThingSpeak)

api\_key = 'YOUR\_THINGSPEAK\_API\_KEY'

url=f'https://api.thingspeak.com/update?api\_key={api\_key}&field1={distance}'

response = requests.get(url)

time.sleep(30)  **Delay between measurements (adjust as needed)**

except KeyboardInterrupt:

GPIO.cleanup()

**CODE FOR MOBILE APP :**

from kivy.app import App

from kivy.uix.boxlayout import BoxLayout

from kivy.uix.button import Button

from kivy.uix.label import Label

class SmartParkingApp(App):

def build(self):

self.status\_label = Label(text="Parking Status: Unknown")

self.check\_button = Button(text="Check Parking Status")

self.park\_button = Button(text="Park")

self.leave\_button = Button(text="Leave")

self.check\_button.bind(on\_press=self.check\_parking\_status)

self.park\_button.bind(on\_press=self.park)

self.leave\_button.bind(on\_press=self.leave)

layout = BoxLayout(orientation='vertical')

layout.add\_widget(self.status\_label)

layout.add\_widget(self.check\_button)

layout.add\_widget(self.park\_button)

layout.add\_widget(self.leave\_button)

return layout

def check\_parking\_status(self, instance)

Send a request to the backend to get the current parking status

Update self.status\_label.text with the response pass

def park(self, instance):

Send a request to the backend to reserve a parking spot

pass

def leave(self, instance):

Send a request to the backend to release the parking spot

pass

if \_name\_ == '\_main\_':

SmartParkingApp().run()