SMART PARKING

ABSTRACT

The concept of smart cities has recently received a lot of traction. The concept of a smart city today appears to be feasible, thanks to the advancement of the Internet of Things. Smart parking is a relatively recent concept that involves the use of sensors to offer information on the status of parking spaces. Sales of passenger automobiles climbed by 26.6 percent to 3.08 million units in 2021, compared to 2.43 million units in 2020. This will increase traffic congestion and slow down average travel speeds. While massive infrastructure improvements are planned to lower the number of cars on the road, this article presented a traffic space management approach to help speed up the process. Traffic congestion, limited automotive parking spots, and road safety are all concerns that the Internet of Things is tackling. We describe an IoT-based cloud-integrated smart parking system in this research. The Smart Parking system comprises an IoT module that is installed on-site and used to monitor and communicate the availability of each parking place. A mobile application is also available, allowing users to check parking spaces' availability. The study finishes with a use case that illustrates the accuracy of the proposed model.

DEVELOPMENT PART 1

KEY COMPONENTS AND STEPS

Sensor Integration:

Install IoT sensors in parking spaces to detect vehicle presence. These sensors can be ultrasonic, magnetic, or infrared, and they transmit data to a central server.

Data Communication:

Set up a robust communication network, such as Wi-Fi, Lora WAN, or cellular, to ensure data from sensors is transmitted to the central server in real-time.

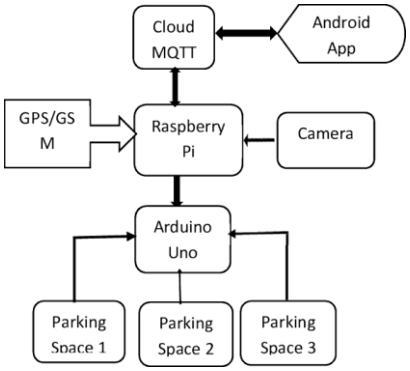
Mobile App and User Interface:

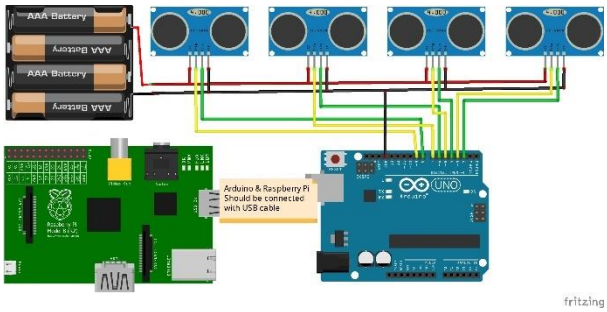
Create a user-friendly mobile app or web interface that allows users to check parking space availability, reserve spots, and make payments.

Real-time Updates:

Provide real-time updates to users, including directions to available spaces and notifications when their reservation is about to expire.

CONFIGURATION





PYTHON SCRIPT

```
import time
import RPi.GPIO as GPIO
import time
import os,sys
from urllib.parse import urlparse
import paho.mqtt.client as paho
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)


define pin for lcd

E_PULSE = 0.0005
E_DELAY = 0.0005
delay = 1



LCD_RS = 7
LCD_E = 11
LCD_D4 = 12
LCD_D5 = 13
LCD_D6 = 15
LCD_D7 = 16
slot1_Sensor = 29
slot2_Sensor = 31
GPIO.setup(LCD_E, GPIO.OUT)
GPIO.setup(LCD_RS, GPIO.OUT)
GPIO.setup(LCD_D4, GPIO.OUT)
GPIO.setup(LCD_D5, GPIO.OUT)
GPIO.setup(LCD_D6, GPIO.OUT)
GPIO.setup(LCD_D7, GPIO.OUT)
GPIO.setup(slot1_Sensor, GPIO.IN)
GPIO.setup(slot2_Sensor, GPIO.IN)
LCD_WIDTH =
LCD_CMD = False
LCD_LINE_1 = 0x80
LCD_LINE_2 = 0xC0
LCD_LINE_3 = 0x90
```

```
def on_connect(self, mosq, obj, rc):
self.subscribe("Fan", 0)
def on_publish(mosq, obj, mid):
print("mid: " + str(mid))

mqttc = paho.Client()
mqttc.on_connect = on_connect
mqttc.on_publish = on_publish

url_str = os.environ.get('CLOUDMQTT_URL', 'tcp://broker.emqx.io:1883')
url = urlparse(url_str)
mqttc.connect(url.hostname, url.port)

'''
Function Name :lcd_init()
Function Description : this function is used to initialized lcd by sending the different commands
'''
def lcd_init():
lcd_byte(0x33,LCD_CMD)
lcd_byte(0x32,LCD_CMD
lcd_byte(0x06,LCD_CMD)
lcd_byte(0x0C,LCD_CMD)
lcd_byte(0x28,LCD_CMD)
lcd_byte(0x01,LCD_CMD)
'''
Function Name :lcd_byte(bits ,mode)
Function Name :the main purpose of this function to convert the byte data into bit and send to lcd port
'''
def lcd_byte(bits, mode):
GPIO.output(LCD_RS, mode)
GPIO.output(LCD_D4, False)
GPIO.output(LCD_D5, False)
GPIO.output(LCD_D6, False)
GPIO.output(LCD_D7, False)
if bits&0x10==0x10:
GPIO.output(LCD_D4, True)
if bits&0x20==0x20:
GPIO.output(LCD_D5, True)
if bits&0x40==0x40:
GPIO.output(LCD_D6, True)
if bits&0x80==0x80:
GPIO.output(LCD_D7, True)
lcd_toggle_enable()
GPIO.output(LCD_D4, False)
GPIO.output(LCD_D5, False)
GPIO.output(LCD_D6, False)
GPIO.output(LCD_D7, False)
if bits&0x01==0x01:
GPIO.output(LCD_D4, True)
if bits&0x02==0x02:
GPIO.output(LCD_D5, True)
if bits&0x04==0x04:
GPIO.output(LCD_D6, True)
if bits&0x08==0x08:
GPIO.output(LCD_D7, True)

lcd_toggle_enable()
'''
Function Name : lcd_toggle_enable()
Function Description:basically this is used to toggle Enable pin
```

```
'''
def lcd_toggle_enable():
time.sleep(E_DELAY)
GPIO.output(LCD_E, True)
time.sleep(E_PULSE)
GPIO.output(LCD_E, False)
time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function Description :print the data on lcd
'''
def lcd_string(message,line):
message = message.ljust(LCD_WIDTH," ")
lcd_byte(line, LCD_CMD)
for i in range(LCD_WIDTH):
lcd_byte(ord(message[i]),LCD_CHR)

lcd_int()
lcd_string("welcome ",LCD_LINE_1)
time.sleep(0.5)
lcd_string("Smart Parking ",LCD_LINE_1)
lcd_string("System ",LCD_LINE_2)
time.sleep(0.5)
lcd_byte(0x01,LCD_CMD)
delay = 5

while 1:
rc = mqttc.loop()
slot1_status = GPIO.input(slot1_Sensor)
time.sleep(0.2)
slot2_status = GPIO.input(slot2_Sensor)
time.sleep(0.2)
if (slot1_status == False):
lcd_string("Slot1 Parked ",LCD_LINE_1)
mqttc.publish("slot1","1")
time.sleep(0.2)
else:
lcd_string("Slot1 Free ",LCD_LINE_1)
mqttc.publish("slot1","0")
time.sleep(0.2)
if (slot2_status == False):
lcd_string("Slot2 Parked ",LCD_LINE_2)
mqttc.publish("slot2","1")
time.sleep(0.2)
else:
lcd_string("Slot2 Free ",LCD_LINE_2)
mqttc.publish("slot2","0")
time.sleep(0.2)
```