SMART PARKING

Project's Objectives:

- ✓ Implement a system to continuously monitor the availability of parking spaces in real time.
- ✓ Develop a mobile application for users to access parking information, reserve spaces, and receive guidance.
- ✓ Provide efficient guidance to users to find and navigate to available parking spaces.
- ✓ Use data analytics to continually optimize parking operations.
- ✓ Encourage user engagement and gather feedback for ongoing improvements.
- ✓ Ensure that the parking system is accessible and inclusive for all users.
- ✓ Reduce the environmental impact of parking operations.

IOT Device Setup:

Setting up a smart parking IoT (Internet of Things) device involves a combination of hardware and software components to enable efficient and automated parking management. Below are the steps to set up a basic smart parking IoT device:

Hardware Setup:

1. Select Hardware Components:

- ✓ Parking Sensors: Ultrasonic or magnetic sensors can be used to detect the presence of vehicles in parking spots.
- ✓ Microcontroller or Single Board Computer (SBC): Raspberry Pi, Arduino, or similar devices can process sensor data and communicate with the server.
- ✓ Communication Module: Wi-Fi, LoRa, or GSM module to send data to the central server.
- ✓ Power Supply: Ensure a reliable power source for continuous operation.

2. Sensor Installation:

- ✓ Install parking sensors in each parking space to detect vehicle presence. Make sure they are securely mounted and properly calibrated

3. Connect Hardware:

- ✓ Connect the sensors and microcontroller/SBC.
- ✓ Ensure power is supplied to the device.

4. Operating System Installation:

- ✓ Install the required operating system on the microcontroller/SBC. For example, you can use Raspberry Pi OS for Raspberry Pi.

5. Sensor Data Processing:

- ✓ Write software code to read data from parking sensors.
- ✓ Implement algorithms to interpret sensor data and detect vehicle presence.

6. Connect to the Internet:

- ✓ Configure network settings to connect to the internet. You can use Wi-Fi, Ethernet, or cellular connectivity depending on the availability and location of the parking area.

7. Data Transmission:

- ✓ Develop code to send sensor data to a central server or cloud platform. You can use MQTT, HTTP, or other communication protocols.

8. Server/Cloud Setup:

- ✓ Set up a server or use a cloud platform to receive and store the data from parking devices. AWS, Azure, or Google Cloud can be used for cloud-based solutions.

9. Database Setup:

- ✓ Configure a database to store parking data. Use databases like MySQL, PostgreSQL, or NoSQL databases, depending on your requirements.

10. Backend Application:

- ✓ Develop a backend application to process and manage parking data, such as tracking available parking spots, handling reservations, and generating reports.

11. Frontend Interface:

- ✓ Create a user interface for both parking operators and users. This can be a web or mobile application showing parking availability and allowing reservations.

12. User Authentication and Security:

- ✓ Implement user authentication and encryption to ensure data security.

13. Alerts and Notifications:

- ✓ Set up alerts and notifications for parking availability, reservations, and any other important information.

14. Testing and Calibration:

- ✓ Test the entire system for accuracy and reliability. Calibrate sensors if needed.

15. Maintenance and Monitoring:

- ✓ Implement a maintenance plan and monitoring system to ensure continuous operation.

16. Scaling:

- ✓ If necessary, scale the system to cover more parking spaces by adding additional IoT devices.

17. Documentation and User Training:

- ✓ Create documentation for system maintenance and provide user training as needed.

18. Compliance and Regulations:

- ✓ Ensure that your smart parking system complies with local regulations and privacy laws.

Platform Development:

1. Hardware Setup:

- ✓ Install sensors (such as ultrasonic or magnetic) in parking spaces to detect vehicle presence.
- ✓ Implement cameras for license plate recognition (LPR) if needed.
- ✓ Set up a network infrastructure to connect all devices.

2. Data Collection:

- ✓ Collect data from sensors and cameras, including real-time occupancy information and license plate data.
- ✓ Utilize IoT (Internet of Things) protocols for efficient data transmission.

3. Data Processing:

- ✓ Use edge computing or cloud-based services to process and analyze the collected data.
- ✓ Apply algorithms to detect and track vehicles and determine parking spaces.

4. User Interface:

- ✓ Develop a user-friendly mobile app and/or web interface for drivers to access the system.
- ✓ Provide real-time information on available parking spaces and navigation to them.

5. Payment Integration:

- ✓ Implement a payment gateway for users to pay for parking.
- ✓ Enable various payment methods, such as credit cards, mobile wallets, or even cryptocurrencies.

6. Admin Dashboard:

- ✓ Create an admin dashboard to manage the system, monitor occupancy, and generate reports.
- ✓ Include features for configuring pricing, setting time limits, and handling exceptions.

7. Security:

- ✓ Ensure data security by encrypting sensitive information.
- ✓ Implement access control and authentication mechanisms for users and administrators.

8. Mobile Apps & Connectivity:

- ✓ Develop mobile apps for both Android and iOS platforms, ensuring seamless connectivity to the hardware components.

9. Notifications:

- ✓ Implement push notifications to inform users of parking availability and payment confirmations.

10. Machine Learning and AI:

- ✓ Utilize machine learning for predicting parking space availability based on historical data.
- ✓ Implement AI for license plate recognition and anomaly detection.

11. APIs and Integration:

- ✓ Offer APIs for third-party integration, such as navigation apps and local city services.

12. Testing:

- ✓ Rigorously test the system for accuracy, reliability, and scalability.
- ✓ Conduct field tests to ensure hardware and software work together seamlessly.

13. Regulatory Compliance:

- ✓ Ensure compliance with local regulations and data protection laws.

14. Scaling and Maintenance:

- ✓ Plan for system scalability as the number of users and parking spaces grow.
- ✓ Regularly maintain hardware and software to prevent system failures.

15. User Support:

- ✓ Provide customer support channels for users facing issues or needing assistance.

16. Feedback and Improvement:

- ✓ Continuously gather user feedback to improve the system and user experience.

Code Implementation:

```
#!/usr/bin/python

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
'''
# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
delay = 1
# Define GPIO to LCD mapping
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)  # E
```

```
GPIO.setup(LCD_RS, GPIO.OUT) # RS
GPIO.setup(LCD_D4, GPIO.OUT) # DB4
GPIO.setup(LCD_D5, GPIO.OUT) # DB5
GPIO.setup(LCD_D6, GPIO.OUT) # DB6
GPIO.setup(LCD_D7, GPIO.OUT) # DB7
GPIO.setup(slot1_Sensor, GPIO.IN)
GPIO.setup(slot2_Sensor, GPIO.IN)
# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
LCD_CMD = False
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x90# LCD RAM address for the 3nd line
def on_connect(self, mosq, obj, rc):
       self.subscribe("Fan", 0)
   def on_publish(mosq, obj, mid):
   print("mid: " + str(mid))
mqttc = paho.Client()                    # object declaration
# Assign event callbacks
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():

  # Initialise display

  lcd_byte(0x33,LCD_CMD) # 110011 Initialise

  lcd_byte(0x32,LCD_CMD) # 110010 Initialise

  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction

  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off

  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size

  lcd_byte(0x01,LCD_CMD) # 000001 Clear display

  time.sleep(E_DELAY)

'''

Function Name :lcd_byte(bits ,mode)

Fuction Name :the main purpose of this function to convert the byte data into bit
and send to lcd port
```

```
'''
def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  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)
```

```
  # Toggle 'Enable' pin
  lcd_toggle_enable()

  # Low bits
  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)

  # Toggle 'Enable' pin
  lcd_toggle_enable()
'''
Function Name : lcd_toggle_enable()
```

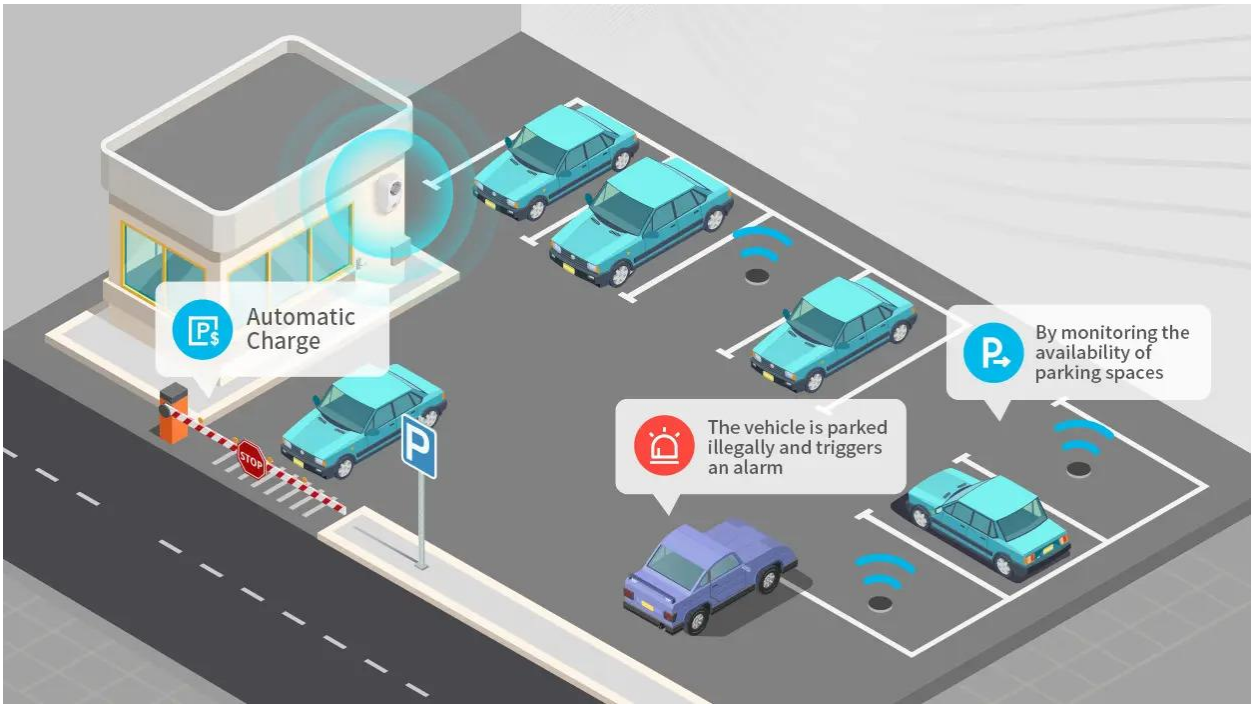
```
Function Description:basically this is used to toggle Enable pin
'''
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function  Description :print the data on lcd
'''
def lcd_string(message,line):
  # Send string to display
   message = message.ljust(LCD_WIDTH," ")
 lcd_byte(line, LCD_CMD)
   for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)
lcd_init()
lcd_string("welcome ",LCD_LINE_1)
time.sleep(0.5)
```

```
lcd_string("Car Parking ",LCD_LINE_1)
lcd_string("System ",LCD_LINE_2)
time.sleep(0.5)
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
# Define delay between readings
delay = 5
while 1:
  # Print out results
  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)
```

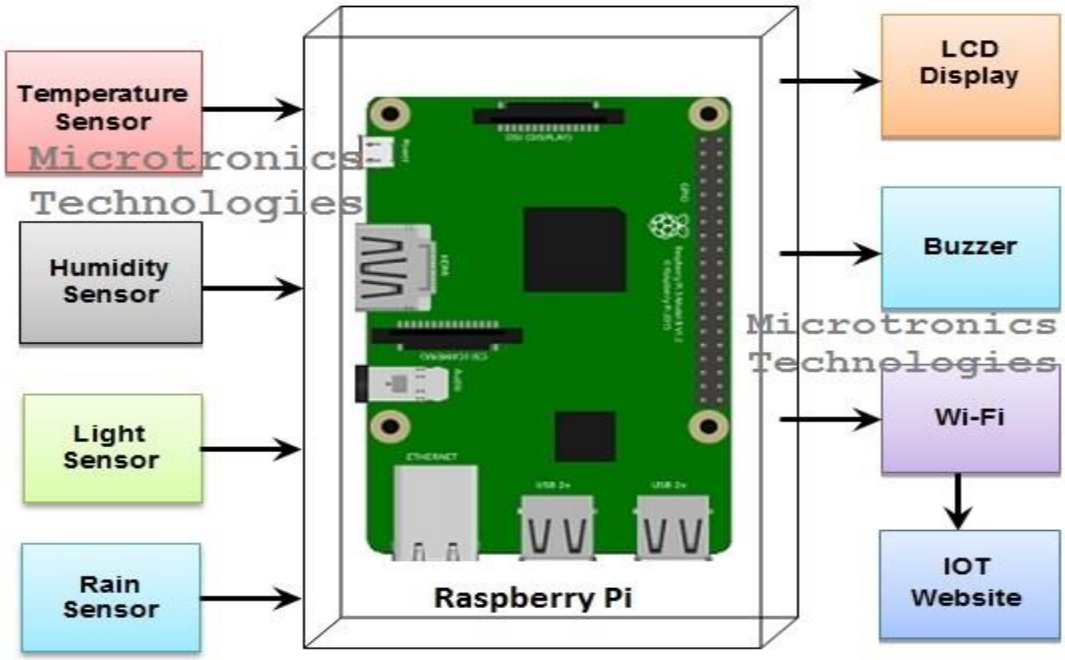
```
 if (slot2_status == False):
  lcd_string("Slot2 Parked  ",LCD_LINE_2)
  mqttc.publish("slot2","1")
  time.sleep(0.2)
 else:
   lcd_string("Slot2 Free  ",LCD_LINE_2)
   mqttc.publish("slot2","0")
   time.sleep(0.2)
```

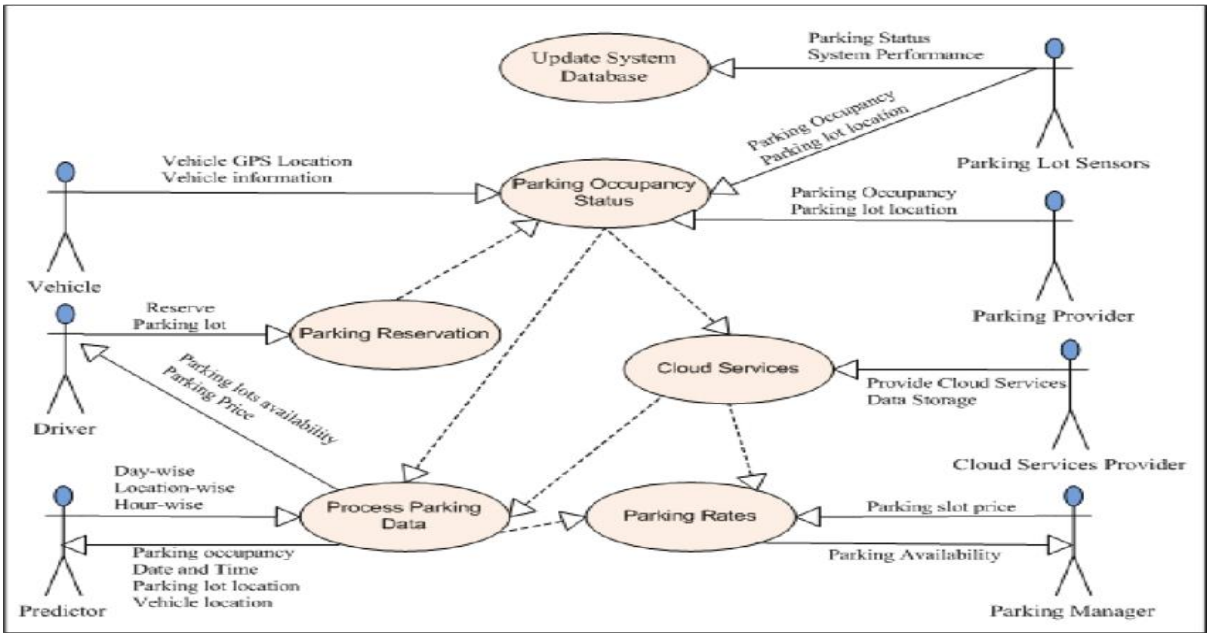
Diagrams:

SMART PARKING:

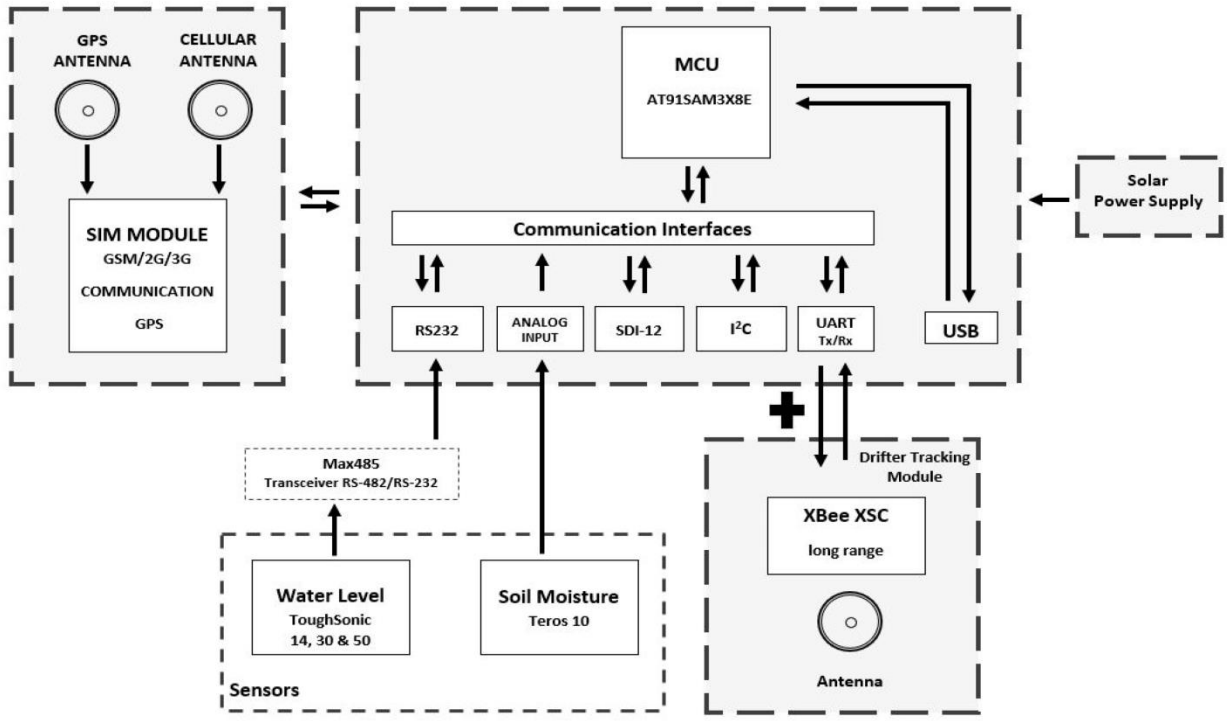


SCHEMATICS:


Temperature Sensor
Humidity Sensor
Light Sensor
Rain Sensor
Microtronics Technologies
Raspberry Pi
LCD Display
Buzzer
Wi-Fi
IOT Website
Microtronics Technologies



DATA-SHARING PLATFORM:



PROJECT OVERVIEW:

A smart parking system using IoT (Internet of Things) technology is designed to improve the efficiency and convenience of parking management for both parking operators and vehicle owners. It involves the integration of sensors, connectivity, and data processing to monitor, manage, and optimize parking spaces.

It should not be confused with mobile parking. This is only a way to pay with your phone or smartphone, to extend the parking time and find your car. Here you still need to find free spots by yourself.

The way smart parking works is, that a sensor detects if, or if not, a car is standing on a parking spot. This can either be a proximity sensor or often also a camera. The advantage of using a camera is that it can view a wider space. On the other hand, if the environment is not ideal for one of these, for example if trees or objects are in the way, then they can become quite expensive in comparison to sensors.

After the sensor detected a change (parking space free or not) on the parking spot, it sends a message to the cloud of the provider. Afterwards, the user receives a notification that there is a free parking space and gets the direction towards it.

When the user leaves the spot, the procedure gets repeated and other people get notified