**SMART PARKING**

**Components :**

**1. Parking Sensors:**

• Type: Ultrasonic sensors, infrared sensors, magnetic sensors, or video cameras.

• Function: Detect the occupancy status of parking spaces and relay real-time data to the central system.

**2. Communication Networks:**

• Type: Wi-Fi, Cellular (3G/4G/5G), LoRaWAN, or other wireless communication protocols.

• Function: Facilitate the transmission of data from parking sensors to the central server and communication with user devices.

**3. Central Server:**

• Function: Receives, processes, and stores data from parking sensors. Manages the overall system, including user requests, reservations, and data analytics.

**4. Mobile App:**

• Platform: iOS, Android, or cross-platform.

• Function: Allows users to check real-time parking availability, make reservations, and receive notifications. Integrates with the central server through APIs.

**5. User Interface (Web Application):**

• Access: Web browser.

• Function: Provides a visual representation of parking space availability, statistics, and management tools for administrators.

**6. Database Management System (DBMS):**

• Type: SQL (e.g., MySQL, PostgreSQL) or NoSQL (e.g., MongoDB).

• Function: Stores and manages data related to parking space availability, reservations, user information, and system logs.

**7. Parking Guidance System:**

• Components: LED displays, electronic signage, or mobile app integration.

• Function: Guides drivers to available parking spaces using real-time data and dynamic routing algorithms.

**8. Automated Payment System:**

• Integration: Payment gateways.

• Function: Facilitates secure and automated payment for parking, integrating with the mobile app and central server.

**9. Security and Surveillance:**

• Components: CCTV cameras, motion sensors.

• Function: Monitors and enhances security within parking facilities, assisting in the prevention of theft, vandalism, and unauthorized access.

**10. Data Analytics and Reporting:**

• Tools: Business intelligence tools.

• Function: Analyzes parking utilization patterns, generates reports, and provides insights for city planning and optimization.

**11. Integration with Navigation Systems:**

• Platforms: Google Maps, Waze, etc.

• Function: Integrates with navigation apps to provide turn-by-turn directions to available parking spaces.

**12. Environmental Sensors (Optional):**

• Type: Air quality sensors, noise sensors.

• Function: Monitors environmental conditions in and around parking facilities, contributing to smart and sustainable city planning.

**13. Emergency and Maintenance Systems:**

• Components: Emergency call buttons, maintenance alert systems.

• Function: Enables users to request assistance in emergencies and notifies maintenance personnel of issues within the parking facility.

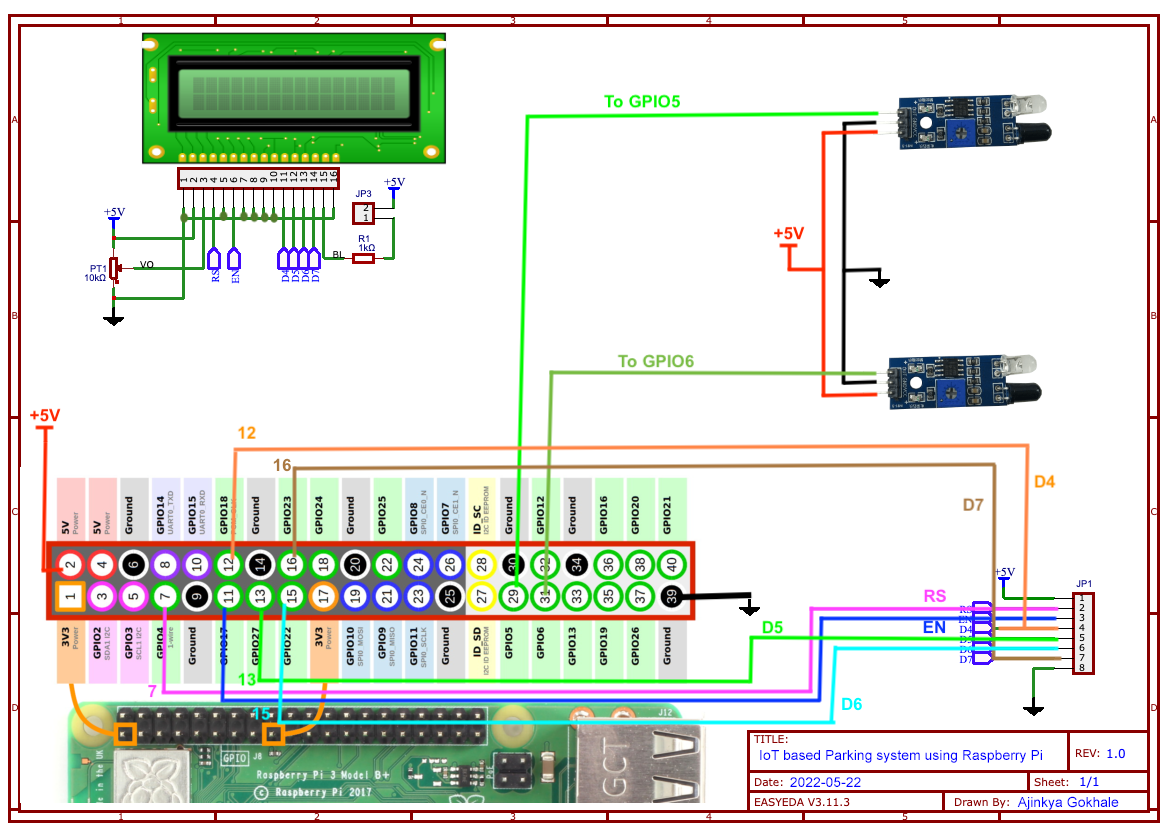
**14. Backend Infrastructure:**

• Hosting: Cloud-based (e.g., AWS, Azure) or on-premises.

• Function: Hosts the central server, databases, and other backend components, ensuring scalability and reliability.

**Technology**

This script represents a basic simulation of a smart parking system with options to check available spaces, reserve a parking space, release a parking space, and exit the system. In a real-world scenario, you would replace the simulation logic with actual data from parking sensors, integrate with databases, and possibly include a web or mobile interface for users. Additionally, security measures and error handling should be implemented for a production-level system.



1. Connect to Wi-Fi: The ESP32 connects to a Wi-Fi network using the provided credentials.
2. Read Sensor Data: A placeholder function read\_sensor\_data is used to simulate reading data from a sensor. Replace this function with your actual sensor reading logic.
3. Send Data to Server: The ESP32 sends the collected data to a server endpoint using HTTP POST. Ensure that your server is configured to receive and process data from the ESP32. Replace SERVER\_ENDPOINT with the actual endpoint of your server.
4. Main Loop: The main loop continuously reads sensor data, sends it to the server, and sleeps for a specified interval. Adjust the sleep time based on your monitoring requirements.

**Program:**

import machine

import time

# Pin assignments for the ultrasonic sensor

TRIGGER\_PIN = 23  # GPIO23 for trigger

ECHO\_PIN = 22     # GPIO22 for echo

# Pin assignment for the LED

LEAK\_LED\_PIN = 19  # GPIO19 for the LED

# Set the pin modes

trigger = machine.Pin(TRIGGER\_PIN, machine.Pin.OUT)

echo = machine.Pin(ECHO\_PIN, machine.Pin.IN)

leak\_led = machine.Pin(LEAK\_LED\_PIN, machine.Pin.OUT)

# Function to measure distance using the ultrasonic sensor

def measure\_distance():

    # Generate a short trigger pulse

    trigger.value(0)

    time.sleep\_us(5)

    trigger.value(1)

    time.sleep\_us(10)

    trigger.value(0)

    # Measure the echo pulse duration to calculate distance

    pulse\_start = pulse\_end = 0

    while echo.value() == 0:

        pulse\_start = time.ticks\_us()

    while echo.value() == 1:

        pulse\_end = time.ticks\_us()

    pulse\_duration = pulse\_end - pulse\_start

    # Calculate distance in centimeters (assuming the speed of sound is 343 m/s)

    distance = (pulse\_duration \* 0.0343) / 2  # Divide by 2 for one-way travel

    return distance

# Function to check for a water leak

def check\_for\_leak():

    # Measure the distance from the ultrasonic sensor

    distance = measure\_distance()

    # Set the threshold distance for detecting a leak (adjust as needed)

    threshold\_distance = 10  # Adjust this value based on your tank setup

    if distance < threshold\_distance:

        # If the distance is less than the threshold, a leak is detected

        return True

    else:

        return False

# Main loop

while True:

    if check\_for\_leak():

        # Blink the LED to indicate a leak

        leak\_led.value(1)  # LED ON

        time.sleep(0.5)

        leak\_led.value(0)  # LED OFF

        time.sleep(0.5)

    else:

        leak\_led.value(0)  # LED OFF

    time.sleep(1)  # Delay between measurements