SOFTWARE DOCUMENTATION

PROJECT: SMART GARAGE SENTINEL

Student: Grameni Alexia

TABLE OF CONTENTS

1. SOFTWARE OVERVIEW

The firmware for the "Smart Garage Sentinel" is developed using C++ within the Arduino Framework. The code is designed to run on the ESP32 DevKit V1 microcontroller. Its primary function is to act as an IoT Gateway: it reads digital data from physical sensors, processes the logic locally to eliminate false positives, and transmits the status in real-time to a Cloud Broker via Wi-Fi.

2. DEVELOPMENT ENVIRONMENT & LIBRARIES

The project was developed using **Visual Studio Code** equipped with the **PlatformIO** extension, ensuring better project management compared to the standard Arduino IDE.

Dependencies and Libraries:

- **<WiFi.h>**: Native ESP32 library used to manage the Wi-Fi station mode and network connection.
- **<PubSubClient.h>**: A lightweight library that implements the MQTT (Message Queuing Telemetry Transport) protocol, allowing the ESP32 to publish messages to the broker.

- **<OneWire.h> & <DallasTemperature.h>**: Specialized libraries required to decode the specific digital protocol used by the DS18B20 temperature sensor.

3. CODE ARCHITECTURE

The code is structured into modular functions to ensure readability and maintainability.

3.1. Hardware Abstraction & Configuration

To ensure easy maintenance, all hardware connections are defined as constants at the beginning of the file.

- **Broker Configuration:** The system connects to the public broker broker.emqx.io on port 1883.
- **Pin Mapping:**
  - PIN_OBSTACLE (GPIO 27): Mapped for the Proximity Sensor.
  - PIN_DOOR (GPIO 26): Mapped for the Line Tracking Sensor.
  - PIN_TEMP (GPIO 23): Mapped for the DS18B20 Data Bus.

3.2. Initialization Phase

This function runs once upon booting. It performs three critical tasks:

1. **Pin Configuration:** Sets GPIO 26 and 27 to INPUT_PULLUP. This is crucial for software stability (detailed in section 4).
2. **Sensor Startup:** Initializes the OneWire bus for the temperature sensor.
3. **Network Handshake:** Calls the setup_wifi() function to establish a connection with the local router and sets the MQTT server parameters.

3.3. Main Control Loop

The main loop runs continuously and is designed to be **non-blocking**. It performs three parallel tasks:

1. **Connection Watchdog:** Checks if the MQTT connection is active; if dropped, it calls reconnect() to restore it automatically.
2. **Security Monitoring:** Reads the IR sensors at high frequency.
3. **Telemetry:** Reads the temperature at low frequency.

4. KEY ALGORITHMS IMPLEMENTED

To ensure a responsive and efficient system, two specific software techniques were implemented:

4.1. State Change Detection (Edge Triggering)

Instead of continuously streaming data (e.g., sending "BLOCKED" 100 times per second while a car is parked), the code implements **State Change Detection**.

- The system stores the *previous state* of each sensor in a variable (`lastObstacolState`).
- A message is published to the Cloud **only** when the *current reading* differs from the *previous reading*.
- **Benefit:** This drastically reduces network bandwidth usage and processing load.

4.2. Non-Blocking Timers (millis)

The temperature sensor (DS18B20) is slow to read (approx. 750ms). Using a standard `delay()` function would freeze the entire security system during the temperature check.

- **Solution:** The code uses the `millis()` function to track time. It checks if 3000ms have passed since the last reading.
- **Benefit:** The security sensors (Obstacle/Door) are monitored continuously without interruption, while temperature is updated in the background every 3 seconds.

5. COMMUNICATION PROTOCOL (MQTT)

The system uses a Publish/Subscribe model. The ESP32 acts as a Publisher.

Topic Structure:

The topics are hierarchically organized under `proiect/garage/` to ensure logical grouping.

- **Obstacle Status** (Topic: `proiect/garage/obstacol`): Sends payloads `"BLOCAT"` (Blocked) or `"LIBER"` (Clear).
- **Door Status** (Topic: `proiect/garage/usa`): Sends payloads `"DESCHIS"` (Open) or `"INCHIS"` (Closed).
- **Environment** (Topic: `proiect/garage/temperatura`): Sends a numeric string (e.g., `"24.5"`).

6. CONCLUSION

The software implementation provides a robust foundation for the Smart Garage Sentinel. By utilizing internal Pull-Up resistors, the code mitigates hardware noise. The implementation of State Change Detection ensures efficient communication, while the non-blocking architecture guarantees that safety-critical sensors are prioritized over telemetry data. The result is a responsive, real-time IoT application.