**IOManager.cpp**

1. **Inclusions et définition de la constante :**
   * #include "IOManager.h" et #include "mqttManager.h" : Inclusions des fichiers d'en-tête pour les classes IOManager et mqttManager.
   * #include <Adafruit\_MCP23X17.h> : Inclusion de la bibliothèque Adafruit pour le MCP23017, un expandeur d'E/S.
   * #include <Wire.h> : Inclusion de la bibliothèque Wire pour la communication I2C.
   * constexpr uint64\_t pulseTime = 500; : Définition d'une constante pour la durée de l'impulsion à 500 ms.
2. **Instances des objets MCP23017 :**
   * Adafruit\_MCP23X17 mcp1; et Adafruit\_MCP23X17 mcp2; : Création de deux instances pour les deux expandeurs d'E/S MCP23017.
3. **Constructeur par défaut :**
   * IOManager::IOManager() = default; : Utilisation du constructeur par défaut pour la classe IOManager.
4. **Méthode setup() :**
   * switchState = false; : Initialisation de l'état des commutateurs à éteint.
   * Wire.begin(4, 5); : Initialisation de la communication I2C avec les broches SDA et SCL spécifiées pour l'ESP8266.
   * **Scanner I2C :** Boucle pour scanner et détecter les périphériques I2C connectés, affichant les adresses détectées ou les erreurs.
   * **Initialisation mcp1 (sorties) :** Configuration des 16 broches de mcp1 en sorties et les mettant toutes à HIGH
   * **Initialisation mcp2 (entrées) :** Configuration des 16 broches de mcp2 en entrées avec résistances de pull-up, puis lecture des positions initiales des aiguillages.
5. **Méthode loop() :**
   * Vérification si le temps d'impulsion est écoulé pour éteindre le relais correspondant.
6. **Méthode attachMqttManager() :**
   * void IOManager::attachMqttManager(MqttManager \*mngr) { mqttManager = mngr; } : Associer un objet MqttManager à l'instance IOManager.
7. **Méthode setLEDState() :**
   * void IOManager::setLEDState(int8\_t Id, bool on, String topic\_sub, String Payload\_sub) {...} :
     + Si on est vrai, le relais est activé, sinon il est désactivé.
     + Lors de la désactivation, les positions des broches mcp2 sont lues et comparées aux positions attendues, et des messages MQTT sont envoyés en conséquence.

***IOManager.cpp*** *configure et gère des expandeurs d'E/S via I2C pour contrôler et surveiller un système de commutateurs, probablement dans un contexte de modélisme ferroviaire. Le système envoie également des mises à jour d'état via MQTT*

**mqttManager.cpp**

1. **Inclusions et définition des constantes :**
   * #include "mqttManager.h" : Inclusion du fichier d'en-tête mqttManager.h.
   * static const String mqttTopicIn = "train/#"; : Définition du topic d'abonnement MQTT pour recevoir tous les messages sous le topic "train/#".
   * static const IPAddress mqtt\_server = {192, 168, 1, 32}; : Définition de l'adresse IP du serveur MQTT.
   * constexpr uint16\_t mqtt\_server\_port = 1883; : Définition du port du serveur MQTT.
2. **Pointeur statique vers IOManager :**
   * static IOManager \*s\_ioManager = nullptr; : Pointeur statique vers une instance de IOManager, utilisé pour accéder à ses méthodes statiquement.
3. **Fonction de rappel pour les messages MQTT :**
   * static void callback(char \*topic, byte \*payload, unsigned int length) : Cette fonction est appelée à chaque fois qu'un message est reçu.
     + Convertit le payload en une chaîne de caractères Payload.
     + Vérifie si le topic reçu est "train/cmd/aig".
     + Convertit le Payload en un entier id et le compare à un tableau de commandes.
     + Appelle setLEDState sur s\_ioManager pour allumer/éteindre les relais selon l'ID et l'état du message.
4. **Constructeur de MqttManager :**
   * MqttManager::MqttManager() : mqttClient(wifiClient) {} : Initialise mqttClient avec wifiClient.
5. **Méthode setup() :**
   * Configure le serveur MQTT, définit le keep-alive et la fonction de rappel.
6. **Méthode loop() :**
   * Vérifie si le client MQTT est connecté, sinon appelle connect() pour se reconnecter.
7. **Méthode connect() :**
   * Tente de se connecter au serveur MQTT et s'abonne au topic spécifié.
   * Envoie un message de bienvenue une fois connecté.
8. **Méthode senMessage() :**
   * Publie un message sur le topic spécifié.
9. **Méthode attachIOManager() :**
   * Attache une instance de IOManager à l'objet MqttManager et met à jour le pointeur statique s\_ioManager.

***mqttManager.cpp*** *configure et gère une connexion MQTT pour un ESP8266. Il écoute les messages sur un topic spécifique et utilise ces messages pour contrôler des relais via l'objet IOManager. La fonction de rappel traite les messages reçus et commande les relais en fonction du contenu des messages*

**Main.cpp**

1. **Préprocesseur :**
   * #ifdef ARDUINO et #ifndef ARDUINO : Ces directives conditionnelles permettent de vérifier si le code est compilé pour une carte Arduino ou pour un autre environnement (comme un PC). Si ARDUINO est défini, le code inclut Arduino.h pour les définitions spécifiques à Arduino.
   * static bool looping = true; : Si le code n'est pas compilé pour Arduino, une variable booléenne looping est définie pour contrôler la boucle principale.
2. **Inclusions de fichiers d'en-tête :**
3. **#include "WifiManager.h"**
   * #include "mqttManager.h"
   * Ces fichiers d'en-tête incluent les définitions des classes WifiManager, MqttManager, et IOManager.
4. **Définition des objets :**
   * WifiManager netMngr; : Objet pour gérer le réseau WiFi.
   * MqttManager mqttMngr; : Objet pour gérer la communication MQTT.
   * IOManager ioMngr; : Objet pour gérer les entrées/sorties.
5. **Fonction setup() :**
   * Serial.begin(115200); : Initialise la communication série à 115200 bauds.
   * Serial.println("Starting"); : Affiche un message de démarrage sur le moniteur série.
   * netMngr.setup(); : Appelle la méthode setup de netMngr pour configurer le WiFi.
   * mqttMngr.setup(); : Appelle la méthode setup de mqttMngr pour configurer MQTT.
   * ioMngr.setup(); : Appelle la méthode setup de ioMngr pour configurer les entrées/sorties.
   * mqttMngr.attachIOManager(&ioMngr); : Attache ioMngr à mqttMngr pour permettre au gestionnaire MQTT de contrôler les IO.
   * ioMngr.attachMqttManager(&mqttMngr); : Attache mqttMngr à ioMngr pour permettre au gestionnaire IO de publier des messages MQTT.
6. **Fonction loop() :**
   * netMngr.loop(); : Appelle la méthode loop de netMngr pour maintenir la connexion WiFi.
   * mqttMngr.loop(); : Appelle la méthode loop de mqttMngr pour gérer la communication MQTT.
   * ioMngr.loop(); : Appelle la méthode loop de ioMngr pour gérer les entrées/sorties.
7. **Fonction main() (pour les environnements non-Arduino) :**
   * setup(); : Appelle la fonction setup pour initialiser les composants.
   * while (looping ) : Boucle infinie pour exécuter la fonction loop en continu.
   * loop(); : Appelle la fonction loop pour maintenir les processus en cours.
   * return 0; : Retourne 0 à la fin de l'exécution (ne sera jamais atteint dans ce cas).

***main.cpp*** *configure et gère un système qui utilise le WiFi pour se connecter à un serveur MQTT, et gère les entrées/sorties (IO) via des modules WifiManager, MqttManager, et IOManager. Il initialise la communication série, configure le WiFi et MQTT, et attache les gestionnaires IO et MQTT pour permettre une communication bidirectionnelle entre eux. La fonction loop maintient les connexions et traite les messages en continu. Pour les environnements non-Arduino, une boucle infinie est utilisée pour émuler le comportement de la fonction loop d'Arduino*