🚪 Module 7 — Projet Final

La Porte de Garage Connectée

GarageController · FSM · Non-bloquant · Sécurité

📝 Changelog — V0.1.0

- Deck 7 : Projet final, cahier des charges, architecture GarageController, FSM porte de garage.

🎯 Contexte du projet

Mission : vous êtes développeurs embarqués chez un fabricant de systèmes domotiques.
Un client commande un **contrôleur de porte de garage** avec les exigences suivantes :

1. Ouverture/fermeture commandée par bouton et/ou via Wi-Fi
2. Détection automatique d'obstacle avec **arrêt d'urgence immédiat**
3. Signalisation visuelle pendant le mouvement (gyrophares)
4. **Aucun** `delay()` — le système doit rester réactif en permanence
5. Code structuré, testable, maintenable (normes industrie)

📋 Architecture matérielle

Entrées :

```
BTN_CMD   (GPIO 12) : Bouton de commande (INPUT_PULLUP)
SENSOR_IR (GPIO 14) : Faisceau IR — obstacle (INPUT_PULLUP)
LIMIT_UP  (GPIO 16) : Fin de course haut (INPUT_PULLUP)
LIMIT_DWN (GPIO 17) : Fin de course bas  (INPUT_PULLUP)
```

Sorties :

```
MOTOR_UP  (GPIO 26) : Moteur — sens montée
MOTOR_DWN (GPIO 27) : Moteur — sens descente
LED_BLINK (GPIO 25) : Gyrophare de signalisation
LED_STATUS(GPIO 33) : LED verte (état normal) / rouge (urgence)
```

📐 La FSM de la porte de garage

🔢 Décomposition des états

État	Moteur	Gyrophare	Déclencheur sortie
`FERM`	OFF	OFF	Btn pressé → `OUVERTURE`
`OUVERTURE`	UP ON	Clignote 500ms	Fin de course → `OUVERTE`
`OUVERTURE`	—	—	Obstacle → `URGENCE`
`OUVERTE`	OFF	OFF	Btn pressé → `FERMETURE`
`OUVERTE`	—	—	Timer 5min → `FERMETURE`
`FERMETURE`	DWN ON	Clignote 300ms	Fin de course → `FERMEE`
`FERMETURE`	—	—	Obstacle → `URGENCE`
`URGENCE`	OFF	LED rouge	Reset → `OUVERTE`

🏗️ Architecture de la classe `GarageController`

```
// lib/GarageController/GarageController.h

enum GarageState { FERMEE, OUVERTURE, OUVERTE, FERMETURE, URGENCE };

class GarageController {
  private:
    // Composants matériels
    Button& btnCmd;
    Button& sensorIR;
    Button& limitUp;
    Button& limitDwn;
    Led&    motorUp;
    Led&    motorDwn;
    Led&    gyrophare;
    Led&    ledStatus;

    // État courant de l'automate
    GarageState state;

    // Timers
    Timer ouvertureTimer;    // Délai auto-fermeture
    Timer blinkTimer;        // Clignotement gyrophare
    bool  blinkState = false;

    // Méthodes privées (un par état)
    void handleFermee();
    void handleOuverture();
    void handleOuverte();
    void handleFermeture();
    void handleUrgence();
    void transitionTo(GarageState next, const char* label);

  public:
    GarageController(Button& btn, Button& ir, Button& lUp, Button& lDwn,
                     Led& mUp, Led& mDwn, Led& gyro, Led& status);
    void begin();
    void update();           // À appeler dans loop()
    GarageState getState() const { return state; }
};
```

💻 Implémentation de `transitionTo()`

```
void GarageController::transitionTo(GarageState next, const char* label) {
    // onExit : actions de sortie de l'état actuel
    switch (state) {
        case OUVERTURE:
        case FERMETURE:
            motorUp.off();
            motorDwn.off();
            gyrophare.off();
            break;
        case URGENCE:
            ledStatus.off();
            break;
        default: break;
    }

    Serial.printf("FSM: %s\n", label); // Log de transition
    state = next;

    // onEnter : actions d'entrée dans le nouvel état
    switch (state) {
        case OUVERTURE:
            motorUp.on();
            blinkTimer.start(500); blinkState = false;
            break;
        case FERMETURE:
            motorDwn.on();
            blinkTimer.start(300); blinkState = false;
            break;
        case OUVERTE:
            ouvertureTimer.start(300000UL); // 5 min auto-close
            ledStatus.on();
            break;
        case URGENCE:
            // Clignotement rouge rapide
            blinkTimer.start(150);
            break;
        default: break;
    }
}
```

💻 État OUVERTURE

```
void GarageController::handleOuverture() {
    // Priorité 1 : détection d'obstacle (SÉCURITÉ CRITIQUE)
    if (sensorIR.isPressed()) {
        transitionTo(URGENCE, "OUVERTURE → URGENCE (obstacle)");
        return;
    }

    // Priorité 2 : fin de course atteinte
    if (limitUp.isPressed()) {
        transitionTo(OUVERTE, "OUVERTURE → OUVERTE (fin de course)");
        return;
    }

    // Gestion du clignotement gyrophare (non-bloquant)
    if (blinkTimer.isExpired()) {
        blinkState = !blinkState;
        blinkState ? gyrophare.on() : gyrophare.off();
        blinkTimer.reset();
    }
}
```

💻 État URGENCE

```
void GarageController::handleUrgence() {
    // Clignotement rouge rapide (toujours non-bloquant)
    if (blinkTimer.isExpired()) {
        blinkState = !blinkState;
        blinkState ? ledStatus.on() : ledStatus.off();
        blinkTimer.reset();
    }

    // Sortie : l'obstacle a été dégagé ET bouton reset pressé
    if (!sensorIR.isPressed() && btnCmd.wasClicked()) {
        gyrophare.off();
        ledStatus.off();
        transitionTo(OUVERTE, "URGENCE → OUVERTE (reset)");
    }

    // Sécurité absolue : moteurs à l'arrêt complet
    motorUp.off();
    motorDwn.off();
}
```

💻 La boucle principale (`main.cpp`)

```cpp
#include <Arduino.h>
#include "GarageController.h"

// Déclaration des composants
Button cmd(12), ir(14), lUp(16), lDwn(17);
Led    mUp(26), mDwn(27), gyro(25), status(33);

// Le contrôleur
GarageController garage(cmd, ir, lUp, lDwn, mUp, mDwn, gyro, status);

void setup() {
    Serial.begin(115200);
    garage.begin();   // Initialise tous les composants
    Serial.println("✅ Système porte de garage prêt.");
}

void loop() {
    garage.update();  // Une seule ligne — aucun delay() !
}
```

> *La boucle est **ultra-propre** : toute la logique est encapsulée dans `GarageController`.*

🏆 Critères d'évaluation

Critère	Points
Classe `GarageController` bien structurée	4 pts
Tous les états FSM implémentés	4 pts
Zéro `delay()` dans la boucle	3 pts
Gyrophare indépendant (non-bloquant)	3 pts
Arrêt d'urgence immédiat	3 pts
Auto-fermeture après 5 min	2 pts
Logs Serial clairs à chaque transition	1 pt
Total	**20 pts**

🚀 Extensions possibles

Niveau avancé (si temps disponible) :

- **Wi-Fi + MQTT** : publier l'état sur `garage/etat` et répondre à des commandes `garage/cmd` (`OPEN`, `CLOSE`, `STOP`)
- **Authentification** : code PIN via MQTT, ignorer les commandes non authentifiées
- **Mode Nuit** : désactiver le gyrophare entre 22h et 7h (via NTP + heure locale)
- **Dashboard Node-RED** : bouton virtuel + indicateur d'état en temps réel
- **Historique** : enregistrer les ouvertures sur carte SD ou dans InfluxDB

🎓 Soutenance finale

Format : présentation devant le groupe (10 min + 5 min questions)

Ce qu'on veut voir :

1. 🎬 Démo live sur matériel (ou simulation)
2. 📐 Présentation de la FSM (diagramme d'états)
3. 💻 Présentation du code `GarageController`
4. 🔒 Expliquer comment l'arrêt d'urgence est garanti
5. 💡 Un retour d'expérience : qu'est-ce qui a été difficile ?

Critère disqualifiant : présence de `delay()` dans la boucle principale.

🎯 Synthèse du Module 7 (et du cours)

Module	Concept clé
1 — Fondements IoT	Histoire, domaines, sécurité physique
2 — Architecture	MCU vs MPU, Deep Sleep, ESP32
3 — GPIO (TP 1)	setup/loop, pinMode, millis()
4 — POO (TP 2)	Classes, std::vector, pointeurs
5 — FSM (TP 3)	Automates : switch, state, lambdas
6 — Réseau	Debouncing, Wi-Fi, MQTT
7 — Projet	**Synthèse : GarageController complet**

🏁 Bonne chance pour la soutenance !

Réda BOUREBABA & Sébastien Antonico
Programmation de Microcontrôleurs & IoT — M2