Faculté sciences & sciences de l'ingénieur

Université Bretagne Sud

Université Bretagne Sud

PRJ1401 : PROJET PLURIDISCIPLINAIRE

CASIER INTELLIGENT

RAPORT FINAL

SNIO 2

2024 -2025

Participants

SERIGNE

MODOU

KASSE

MAMADOU

SALIOU

DIALLO

Encadrant

Mr Philippe Tanguy

Table des matières

I. Introduction

1. Contexte

A l'ère où la technologie impose sa domination, les institutions tournent alors leur regard vers ces ressources du genre pour améliorer sa propre expérience et celle de l'étudiant. Dans cette optique, à la recherche de la réalisation de son campus virtuel "campusco", l'Université Bretagne Sud prévoit d'installer des casiers connectés qui "auprès des utilisateurs du campus il permettra du "click et collect" entre utilisateurs ou simplement de consigne ".

2. Présentation du projet

C'est dans ce contexte que notre projet vise à mettre en place un système de **casiers intelligents** qui offrira aux étudiants et aux enseignants la possibilité de déposer et de récupérer des objets de manière simple, rapide et sécurisée.

On essaiera donc de proposer une **solution automatisée et connectée** pour faciliter l'échange de matériel entre les usagers du campus. On offrira une **expérience fluide et intuitive** avec lequel chaque utilisateur pourra accéder à un casier via un système d'identification efficace.

Dans ce contexte, ce projet contribue à l'innovation technologique et à une amélioration des infrastructures universitaires. S'il modernise indéniablement la logistique sur le campus, il fournit également une solution fiable et ergonomique qui répond aux attentes des étudiants et du personnel. Son incorporation contribue à la rendre l' espace universitaire plus intelligent, connecté et sécurisé.

II. Etat de l'art : Analyse des projets similaires

Plusieurs initiatives de casiers intelligents ont déjà été mises en place dans différents contextes. Ces projets, bien qu'innovants, présentent certaines limitations qui nous permettent d'identifier les améliorations et les adaptations nécessaires pour notre propre système.

1. Amazon Locker

Ces casiers sont conçus pour les livraisons e-commerce d'Amazon. Ils permettent aux clients de récupérer leurs commandes en saisissant un code reçu par courriel ou via l'application Amazon. Ce modèle est optimisé pour le commerce en ligne et fonctionne de manière entièrement automatisée.

- **Points positifs**
  - Solution robuste et éprouvée, utilisée à grande échelle.
  - Interface intuitive et simple pour l'utilisateur.
  - Sécurisation efficace des colis grâce à une gestion automatisée.
- **Limites**
  - **Solution propriétaire** exclusivement dédiée aux services Amazon, ne pouvant être adaptée à d'autres usages.
  - **Absence de flexibilité** pour une utilisation académique ou communautaire.
  - **Accès restreint** aux seuls clients Amazon, ce qui en fait un système fermé.

2. Smart Lockers de l'Université de Californie (UC Davis)

Ce projet universitaire propose des casiers connectés permettant aux étudiants d'emprunter ou de retourner du matériel (ordinateurs, outils de laboratoire, livres, etc.). L'accès aux casiers se fait via un code ou une carte d'étudiant, ce qui permet une gestion semi-automatisée du matériel.

- **Points positifs**
  - **Adapté au milieu universitaire**, facilitant le prêt et le retour d'équipements.
  - **Accès contrôlé** via une carte étudiante, renforçant la sécurité.
  - **Simplicité d'utilisation** pour les étudiants et le personnel universitaire.
- **Limites**
  - **Fonctionnalités limitées** : le système ne permet que le stockage et la récupération de matériel sans intégration avancée avec d'autres services numériques.
  - **Manque d'interopérabilité** : ne propose pas d'intégration avec des services externes pour une gestion plus centralisée des ressources universitaires.

3. Casiers connectés de l'Université de Lille

L'Université de Lille a introduit un système de casiers intelligents financé par la Contribution Vie Étudiante et Campus (CVEC). Ces casiers sont destinés au prêt de matériel sportif, permettant aux étudiants d'accéder aux équipements via un système automatisé.

- **Points positifs**
  - **Projet universitaire en France**, prouvant la faisabilité du concept dans un contexte académique national.
  - **Solution pratique** pour la gestion du prêt de matériel sportif.
  - **Financement institutionnel**, favorisant l'adoption et l'accessibilité.
- **Limites**
  - **Capacité limitée** : le nombre de casiers ne permet pas de répondre à une forte demande.
  - **Utilisation restreinte** uniquement pour le matériel sportif, alors que d'autres usages pourraient être envisagés (prêt de matériel informatique, stockage temporaire, etc.).
  - **Absence de gestion avancée des accès** pour permettre une plus grande flexibilité d'utilisation.

4. En quoi ces projets nous ont aidé dans le nôtre ?

L'examen des initiatives similaires comme les Amazon Lockers, les Smart Lockers de l'Université de Californie (UC Davis) et les casiers connectés de l'Université de Lille nous a permis d'en apprendre plusieurs leçons, économiques et d'ordre technique :

À un premier niveau, ces projets nous ont permis de mieux comprendre comment les utilisateurs interagissent réellement avec ce type de service. Leur étude nous a montré combien il est crucial de proposer une solution simple à utiliser, rapide d'accès et fiable, surtout dans un environnement universitaire où les utilisateurs souhaitent un service fluide et sans contraintes.

À un second niveau, ceux-ci nous ont également inspirés dans l'organisation de notre architecture technique : le découplage entre le microcontrôleur chargé des actions physiques (ESP32) et le serveur distant responsable des accès. L'utilisation du protocole MQTT, très courant dans les systèmes IoT, s'est imposée à nous pour sa légèreté et son efficacité.

III. Analyse des besoins

- **Système de verrouillage et de déverrouillage automatique des portes** :

La porte du casier doit s'ouvrir automatiquement grâce à un système de **loquets électriques discret** qui offre une sécurité et une ouverture rapide.

- **Ecran LCD** :

Il permet d'afficher des informations concernant l'ouverture et la fermeture de la porte (verrouillage et déverrouillage).

- **Système d'authentification (NFC)** :

Ce système permet de sécuriser les casiers. Avec celui-ci l'utilisateur pourra s'authentifier grâce à un **badge magnétique** afin de récupérer ou de déposer son colis ce qui garantit ainsi le fait que seuls les utilisateurs autorisés puissent accéder au casier afin d'éviter les vols.

- **Connexion au réseau de l'UBS** :

La connexion au réseau de l'Université Bretagne Sud (UBS) permettra au casier d'être connecté à Internet et aux serveurs locaux.

1. Exemples de scénario d'utilisation : Prêt d'un PC par la DSI à un étudiant

Contexte : Un étudiant a besoin d'un PC portable pour un projet, et demande à la DSI de lui en prêter un via un casier intelligent.

Déroulement :

- **Demande de l'étudiant (indépendant de notre projet)**
  - L'étudiant effectue une demande de prêt sur la plateforme universitaire.
  - La DSI valide la demande et attribue un PC à l'étudiant.
- **Dépôt du PC par la DSI**
  - Un agent de la DSI place le PC dans un casier disponible.
  - Il associe ce casier au badge NFC de l'étudiant via l'interface de gestion.
  - L'étudiant reçoit une notification l'informant que son PC est prêt à être récupéré. (Indépendant de notre projet)
- **Récupération du PC par l'étudiant**
  - L'étudiant se rend au casier et **scanne son badge NFC** sur le lecteur.
  - Le casier s'ouvre automatiquement.
  - Il prend le PC et referme la porte.
  - La DSI supprime l'accès au casier à l'étudiant.
- **Retour du PC**
  - Une fois l'utilisation terminée, la DSI lui attribue un casier
  - L'étudiant retourne le PC en allant au casier qui lui est attribué.
  - Il **scanne à nouveau son badge NFC**, ouvre le casier et replace l'ordinateur.
  - L'agent de la DSI peut ensuite récupérer le matériel ou le réattribuer à un autre étudiant si nécessaire.

2. Spécifications Techniques

a. Composants Matériels

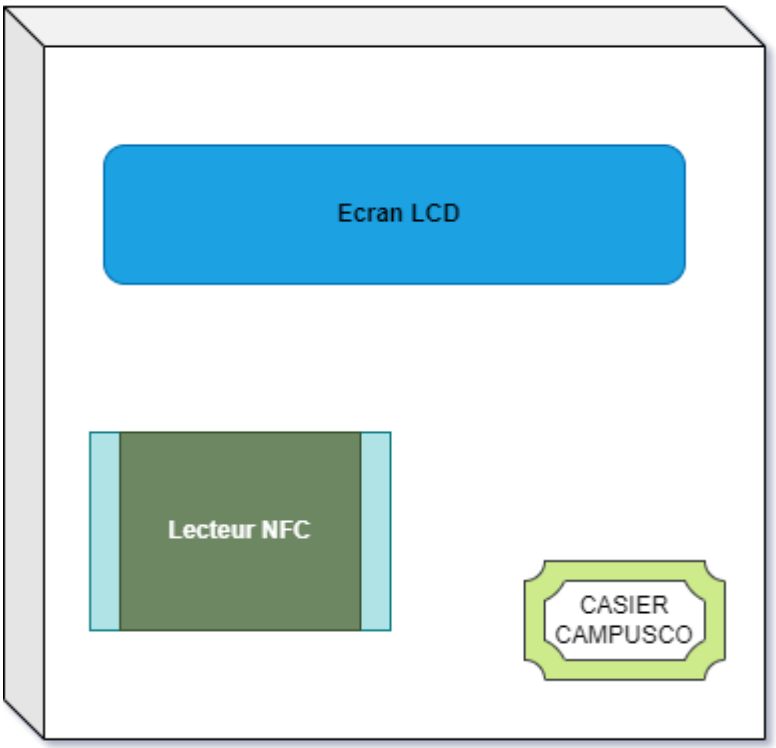
- **Microcontrôleur** : ESP32 Devkit v4 (WROOM 32)
- **Système de verrouillage** : Loquet électromagnétique
- **Module NFC** : Lecteur NFC DFRobot Gravity
- **Interface utilisateur** : Écran LCD + adaptateur I2C
- **Alimentation** : 5V/12V selon les composants

b. Fonctionnalités

- Identification des utilisateurs via **badge NFC**.
- Contrôle d'accès aux casiers via **une connexion entre le lecteur et l'ESP32 pour vérifier l'identité de l'utilisateur**.
- Affichage **dynamique** des informations sur **écran LCD**.
- Connexion au réseau de l'UBS via **Wi-Fi ou Ethernet.**

IV. Schémas d'Architectures

L'architecture comprendra une **interface externe** qui définit **la partie visible par l'utilisateur** et une **interface interne**, plus important qui montrera les **connexions entre les différents composants.**



Interface externe du casier

1. Interface externe

L'interface défini ci-contre correspond à l'apparence visible du casier intelligent pour les utilisateurs. Voici une explication des éléments présents :

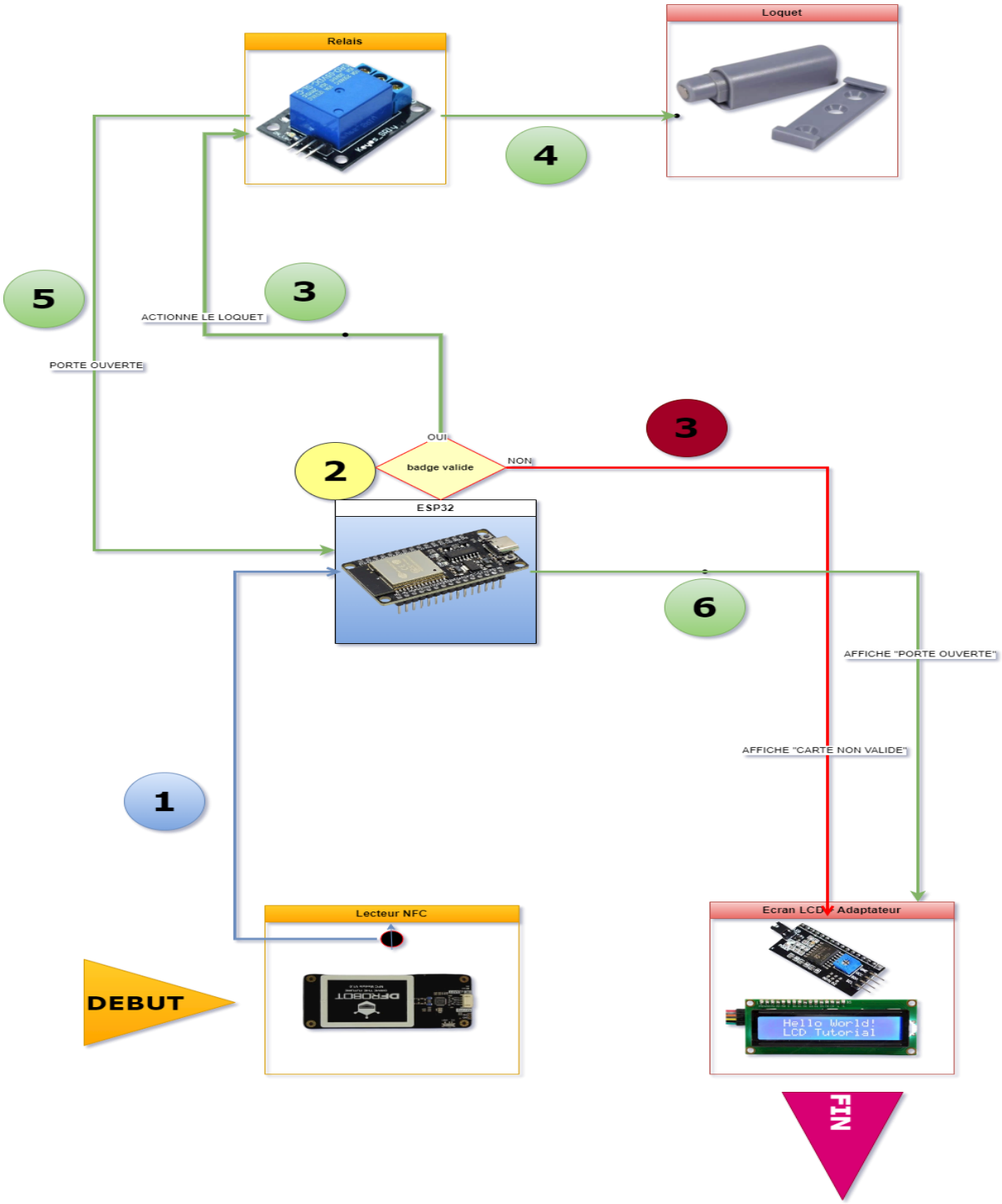
a. Écran LCD (en bleu)

b. Lecteur NFC (en vert)

c. Casier Campusco (en Jaune et noir)

- Indique le nom du casier.

2. Interface Interne



Interface interne du casier

3. Analyse de l'interface interne

- **Identification du badge NFC**
  - L'utilisateur scanne son badge sur le **lecteur NFC (Début)**.
  - Le lecteur envoie l'information à **l'ESP32** pour vérification (Etape 1)
- **Vérification du badge**
  - L'ESP32 (Étape 2) compare l'ID du badge avec la base de données intégrée.
  - Si le badge est valide, on passe à l'étape 3 (vert)
  - Si le badge est invalide, un message "Carte non valide" s'affiche sur l'écran LCD (Etape 3 rouge)
- **Activation du mécanisme d'ouverture**
  - Si le badge est valide, l'ESP32 active le **relais** (Étape 3 - vert).
  - Le relais actionne le **loquet** (Étape 4), permettant l'ouverture du casier.
- **Confirmation de l'ouverture**
  - Le casier est ouvert (Étape 5), et l'écran LCD affiche "Porte ouverte" (Étape 6).
  - L'utilisateur peut récupérer ou déposer son matériel (Fin).

4. Erreurs et Sécurité

La gestion des erreurs est tout aussi important pour ce projet. En effet, notre système pourra bien subir des désagréments et des solutions devront donc être envisagées.

De plus, la **protection des données** via un système de chiffrement des informations stockées devient une obligation au vu de l'importance que cela représente.

V. Contraintes du projet de PRJ1401

1. Lecteur NFC

Rôle : Permet l'authentification de l'utilisateur en scannant une carte NFC.

Contraintes :

- **Alimentation** : 5V, compatible avec l'ESP32.
- **Latence** : Le temps de lecture doit être rapide (<1 seconde).
- **Compatibilité** : Doit fonctionner avec les cartes NFC autorisées.
- **Sécurité** : Les données doivent être transmises de manière sécurisée pour éviter le clonage de cartes.

2. Microcontrôleur ESP32

Rôle : Cerveau du système, il reçoit les données du lecteur NFC, contrôle le loquet et affiche les informations sur l'écran.

Contraintes :

- **Alimentation** : 3,3V (nécessite un régulateur si alimenté par 5V ou plus).
- **Nombre de ports GPIO limité** : Doit gérer plusieurs périphériques sans conflit.
- **Latence** : Doit traiter les données en temps réel et envoyer rapidement les commandes.
- **Consommation électrique** : Modérée (~100-250 mA), doit être optimisée si le système est alimenté sur batterie.
- **Sécurité** : Connexions Wi-Fi/Bluetooth doivent être protégées contre les cyberattaques.

3. Relais

Rôle : Active et désactive l'alimentation du loquet magnétique en fonction des commandes de l'ESP32.

Contraintes :

- **Alimentation** : Nécessite du 5V pour fonctionner, compatible avec l'ESP32 via un transistor ou un circuit d'adaptation de tension.
- **Charge supportée** : Doit être capable de commuter la tension et l'intensité du loquet (souvent 12V ou 24V).
- **Temps de réponse** : <100 ms pour éviter tout retard dans l'ouverture.
- **Durée de vie** : Doit supporter plusieurs cycles d'activation/désactivation sans s'user rapidement.

4. Loquet Magnétique

Rôle : Verrouille et déverrouille le casier selon les instructions du relais.

Contraintes :

- **Alimentation** : Fonctionne souvent en 12V ou 24V → nécessite une alimentation adaptée.
- **Consommation électrique** : Peut consommer plusieurs ampères lorsqu'il est activé.
- **Sécurité physique** : Doit être robuste pour résister aux tentatives d'effraction.
- **Mode de verrouillage** : Certains modèles restent verrouillés même sans courant (fail-secure), d'autres se déverrouillent en cas de coupure de courant (fail-safe).
- **Temps de réaction** : Doit s'ouvrir instantanément après activation pour éviter un décalage perceptible par l'utilisateur.

5. Écran LCD (avec adaptateur I2C)

Rôle : Affiche les instructions et les statuts du casier (ex : "Casier ouvert", "Carte invalide", etc.).

Contraintes :

- **Alimentation** : Fonctionne généralement en 5V, nécessite une compatibilité avec l'ESP32 (3,3V).
- **Interface de communication** : Utilise l'I2C (moins de fils, mais nécessite des adresses spécifiques pour éviter les conflits avec d'autres périphériques).
- **Lisibilité** : Doit être visible même en conditions de faible luminosité (rétroéclairage requis).
- **Taille d'affichage** : Généralement limité à 16x2 ou 20x4 caractères → nécessite une gestion efficace des messages affichés.

6.Serveur

- **Rôle :** L'**ESP32** doit être connecté à un **serveur** contenant la **base de données** des cartes NFC autorisées. Cela ajoute des **contraintes supplémentaires** à prendre en compte :
- **Contraintes :**
  - **Connexion Wi-Fi de l'ESP32 :** Permet la communication avec le serveur --> Nécessite une connexion réseau stable et sécurisée (peut être limité en cas de coupure du Wi-Fi)
  - **Base de données :** Stocke les identifiants NFC autorisés --> Doit être rapide pour valider les cartes NFC en temps réel (latence minimale)
  - **Sécurité des échanges :** Protège contre les accès non autorisés --> Doit utiliser le chiffrement des données (HTTPS, authentification sécurisée)
  - **Synchronisation :** Mise à jour en temps réel des accès --> Assurer que la base de données reste à jour sans latence excessive
  - **Serveur hébergé (local ou cloud) :** Gère les requêtes et les accès aux casiers --> Si en cloud, dépend d'Internet ; si local, doit être robuste et accessible en interne

7. Impact des Contraintes sur le Projet

- **Optimisation des échanges serveur-ESP32** pour limiter la latence (utilisation d'un protocole léger comme MQTT).
- **Sécurisation des communications** pour éviter les failles de sécurité.
- **Prévoir un mode de secours** en cas de panne réseau (ex: cache local des cartes NFC).
- **Assurer une alimentation stable** pour les composants énergivores comme le loquet magnétique.

VI. Justification du choix des composantes du projet de PRJ1401

Le projet repose sur plusieurs composantes essentielles, chacune ayant été sélectionnée pour répondre à des besoins précis en matière de sécurité, d'automatisation et d'accessibilité. Voici une justification détaillée du choix de chaque élément du système.

1. Lecteur NFC

Pourquoi ?

- Il permet une **authentification rapide et sécurisée** des utilisateurs.
- La technologie NFC est **simple d'utilisation** : il suffit de passer une carte devant le lecteur.
- Elle est **largement utilisée** dans les systèmes d'accès et compatible avec les cartes étudiantes.
- Le NFC offre un **temps de réponse rapide** et réduit le risque d'erreurs d'authentification.

Alternatives possibles :

- Un clavier pour entrer un code → moins sécurisé et plus lent.
- Un QR code → nécessite un smartphone

2. ESP32 Devkit V4 (WROOM 32)

Pourquoi ?

- Ce microcontrôleur est **puissant** et **polyvalent**.
- Il intègre une **connexion Wi-Fi et Bluetooth**, essentielle pour la communication avec un serveur et la base de données.
- Il offre **une grande capacité de traitement** pour gérer les accès et contrôler les autres composants.
- Il est **économique et largement documenté**, facilitant son intégration et son développement.

Alternatives possibles :

- Arduino Uno → Pas de Wi-Fi natif, nécessiterait un module additionnel.
- Raspberry Pi → Plus puissant, mais aussi plus coûteux et surdimensionné pour cette application.

3. Relais

Pourquoi ?

- Nécessaire pour **contrôler l'ouverture et la fermeture des loquets** via le microcontrôleur.
- Fonctionne comme un **interrupteur électronique** permettant de gérer des tensions plus élevées.
- Compatible avec l'ESP32 et **facile à piloter via un signal numérique**.

Alternatives possibles :

- Un MOSFET → Trop complexe pour cette application.
- Un relais statique → Plus coûteux, sans réel avantage pour ce projet.

4. Loquets magnétiques

Pourquoi ?

- Ils assurent **une fermeture sécurisée** des casiers et peuvent être activés électroniquement.
- Ils sont **robustes et fiables**, évitant les ouvertures accidentelles.
- Fonctionnent bien avec un relais et offrent **une réponse rapide** à la commande d'ouverture.

Alternatives possibles :

- Un verrou motorisé → Plus complexe et plus énergivore.

5. Écran LCD + Adaptateur I2C

Pourquoi ?

- Permet d'afficher des **instructions claires** à l'utilisateur (exemple : "Présentez votre carte", "Casier ouvert", "Accès refusé").
- L'adaptateur I2C simplifie la connexion avec l'ESP32, réduisant le nombre de fils nécessaires.
- Il consomme **peu d'énergie** et reste **lisible même en plein jour**.

Alternatives possibles :

- Un écran OLED → Plus compact mais coûteux, et moins utile pour des affichages simples.
- Un écran tactile → Trop complexe et pas nécessaire pour cette application.

6. Connexion au serveur et base de données

Pourquoi ?

- Permet de **stocker et gérer les numéros de cartes NFC autorisées** en temps réel.
- Facilite **l'ajout et la suppression d'utilisateurs** sans modifier le code du microcontrôleur.
- Garantit une **sécurité renforcée** en limitant l'accès uniquement aux personnes autorisées.
- Assure une **traçabilité des ouvertures**, utile pour la gestion et la maintenance.

Alternatives possibles :

- Stockage local sur l'ESP32 → Limité en mémoire et plus difficile à mettre à jour.
- Utilisation d'une API externe → Peut poser des problèmes de latence ou de confidentialité.

VII. Conception Matérielle (Niveau Bas)

1. Architecture Générale

Le système de casiers intelligents repose sur une architecture matérielle simple, modulaire et fiable, articulée autour d'un microcontrôleur ESP32. Ce dernier assure la gestion des principales fonctions du système : identification de l'utilisateur via un badge NFC, commande d'ouverture du casier à l'aide d'un relais, et affichage d'informations sur un écran LCD.

Chaque sous-système joue un rôle précis :

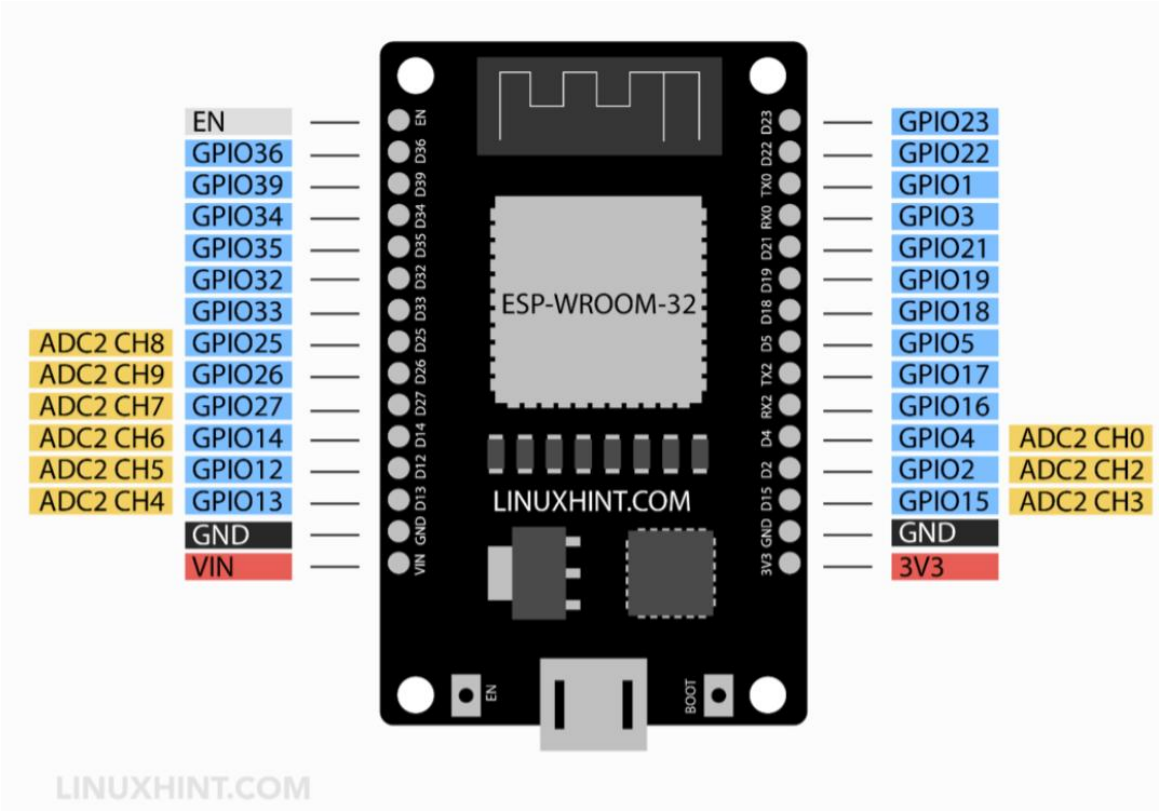
- Le **lecteur NFC** permet l'identification de l'utilisateur par lecture de badges sans contact ;
- Le **relais** commande le **loquet électrique**, assurant la sécurité physique du casier ;
- L'**écran LCD** informe l'utilisateur de l'état du casier et des actions à effectuer.

Cette organisation permet une séparation claire des responsabilités matérielles, facilitant la maintenance, l'évolution du système et l'intégration de nouvelles fonctionnalités.

2. Composants Principaux

a. Microcontrôleur ESP32

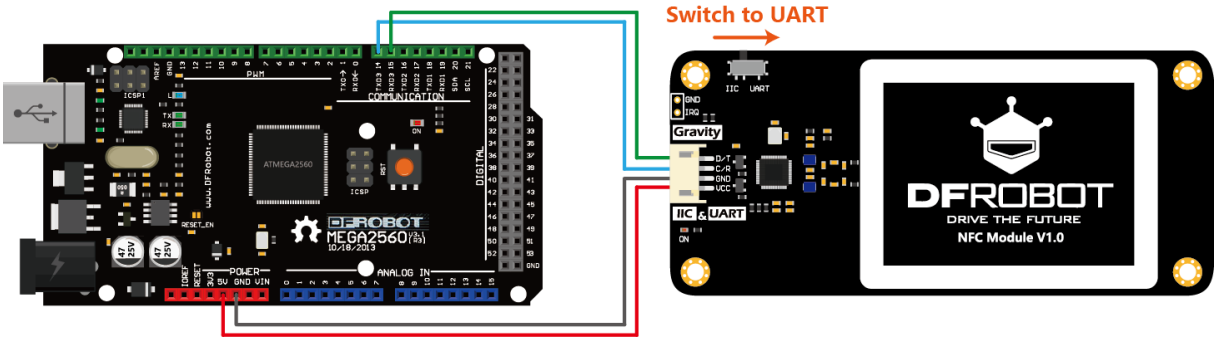
- **Fonction** : Unité centrale du système. Il coordonne la lecture des identifiants NFC, la communication avec le serveur via Wi-Fi/MQTT, et la commande d'ouverture.
- **Connexions** :
    - **I2C** : communication avec le lecteur NFC (broches SDA : GPIO 21, SCL : GPIO 22).
    - **GPIO** : commande du relais (GPIO 16 pour le casier 1, et GPIO 17 pour le casier 2).
    - **Wi-Fi** : liaison avec le serveur MQTT pour la validation des identifiants.



Pinout complet de l'ESP32

b. Lecteur NFC (PN532)

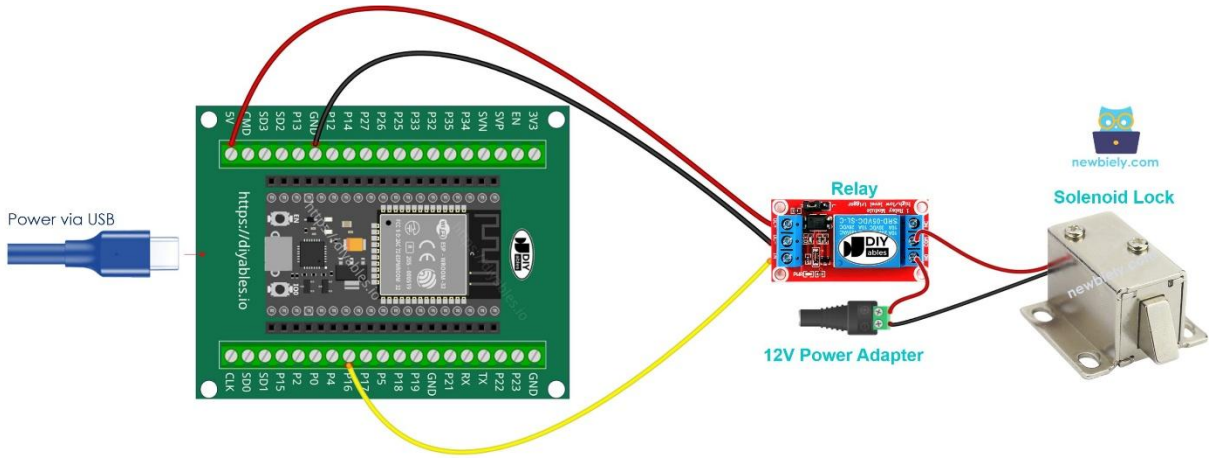
- **Modéle:** DFROBOT Gravity: UART & I2C NFC Module
- **Fonction** : Détection et lecture des badges NFC/MIFARE pour l'identification des utilisateurs.
- **Connexions** :
    - **I2C** : liaison avec l'ESP32 (broches IRQ et RESET configurées pour I2C logiciel).
    - **Alimentation** : en 3.3V ou 5V selon la version (le modèle Gravity de DFRobot accepte les deux).



- **Avantages** :
    - Détection rapide et fiable.
    - Facilité d'intégration avec Arduino.
    - Support étendu et documentation abondante.

c. Relais Électromécanique 5V

- **Fonction** : Interrupteur commandé logiquement pour activer le loquet du casier.
- **Connexions** :
    - **Entrée de commande** : GPIO 26 (ESP32).
    - **Alimentation** : 5V (avec masse commune).
    - **Sortie** : connectée au loquet (bornes COM et NO).



- **Comportement** : Le relais est activé uniquement si le badge est validé par le serveur. Le casier s'ouvre pendant cinq (5) secondes, puis se referme automatiquement.

d. Alimentation 5V (min. 2A)

- **Fonction** : Fournir une alimentation stable à l'ensemble du système.
- **Connexions** :
    - Via Vin pour le loquet et relais.
    - Distribution parallèle vers le relais et le loquet.
- **Justification** : Une alimentation fiable est indispensable pour éviter les redémarrages imprévus du microcontrôleur, notamment lors des pointes de courant induites par l'activation du loquet.

VIII. Conception Logicielle (Niveau Haut)

1. Microcontrôleur ESP32

La partie bas niveau du système repose sur l'utilisation d'un microcontrôleur **ESP32**, chargé de gérer la détection des badges NFC, la communication réseau, et le contrôle physique des casiers. Cette section détaille l'architecture logicielle embarquée et les interactions matérielles mises en œuvre.

a. Objectifs fonctionnels

Le microcontrôleur embarqué a pour rôle de :

- Établir une connexion Wi-Fi vers le réseau local.
- Lire en continu les badges NFC présentés devant le lecteur.
- Envoyer l'UID des badges détectés au serveur via le protocole MQTT.

- Recevoir les autorisations ou refus du serveur, et agir en conséquence (ouverture d'un casier via relais ou refus d'accès).
- Assurer une reconnexion automatique en cas de perte de connexion réseau.

b. Architecture logicielle embarquée

Le programme, développé avec l'IDE Arduino, suit une structure modulaire et réactive. Il repose sur les bibliothèques suivantes :

- Adafruit_PN532 pour la gestion du lecteur NFC ;
- WiFi.h pour la connexion réseau ;
- PubSubClient pour la communication MQTT.

À l'initialisation (**setup()**), les éléments suivants sont configurés :

- Les broches des relais sont initialisées en sortie et placées à l'état bas.
- Le module Wi-Fi est connecté au réseau local.
- Le client MQTT est initialisé et connecté au broker (serveur).
- Le lecteur NFC est activé en mode de lecture passive c'est-à-dire il attend qu'un badge (ou une carte NFC) s'approche pour lire les données qu'il contient, sans émettre activement une demande de communication en boucle

La boucle principale (**loop()**) exécute les opérations suivantes de manière cyclique :

- **Vérification des connexions** : si le Wi-Fi ou le MQTT est déconnecté, une reconnexion est tentée.
- **Lecture NFC** : la fonction badgeValide() détecte la présence d'un badge et lit son UID.
- **Transmission** : si un nouveau badge est détecté, son UID est publié sur le topic MQTT casier/ouverture.
- **Réaction au message serveur** : via un callback MQTT, le microcontrôleur reçoit une commande de type CASIER:x (*x* étant le numéro du casier) ou REFUS. Dans le premier cas, la fonction ouvrirCasier(int numero) active le relais correspondant pendant une durée de 5 secondes.

Une variable de garde (casierOuvert) permet d'éviter les doubles ouvertures sur un même signal.

c. Résilience et surveillance mémoire

Le système embarqué intègre plusieurs mécanismes de surveillance :

- Affichage du niveau de pile disponible pour la tâche principale (uxTaskGetStackHighWaterMark).
- Possibilité de surveiller la mémoire dynamique (heap) et la mémoire flash du module ESP32.
- Reconnexion automatique en cas de coupure Wi-Fi ou MQTT.

Ces fonctions nous ont grandement servies dans l'analyse de la mémoire.

Merci de la précision. Voici un **texte rédigé pour ton rapport** (style technique, sans emoji ni commentaires informels), qui **intègre la description du serveur Python, du protocole MQTT et de la base de données embarquée** :

2. Architecture côté serveur : gestion des autorisations via MQTT et base de données intégrée

Le serveur tient un rôle essentiel dans la validation des badges NFC lus par les microcontrôleurs. Il est programmé en Python à l'aide de la bibliothèque paho-mqtt, qui facilite une communication en temps réel en utilisant le protocole MQTT (Message Queuing Telemetry Transport).

Il fonctionne comme un client-broker au sein du réseau local, se connectant à un broker MQTT hébergé sur la machine locale à l'adresse IP définie. Il s'abonne au topic casier/ouverture, destiné à recevoir les identifiants de badges envoyés par les modules ESP32.

Lorsqu'un lecteur NFC détecte un badge et le transmet via MQTT, le serveur le compare à une base de données locale structurée sous forme de dictionnaire Python. Cette base associe chaque UID autorisé à un numéro de casier. Ce procédé simple mais efficace permet de gérer les accès de manière basique sans nécessiter l'utilisation d'une base de données externe.

Voici la base utilisée :

```
badges autorisés = {
  "44F77FA591490" : "1"   # Carte UBS personnelle
   "40F1ED56" : "2",  # Badge NFC générique
}
```

Lorsqu'un identifiant valide est reçu, le serveur publie en retour un message de type **CASIER:<numéro>** sur le même topic. Ce message est ensuite interprété par l'ESP32 pour déclencher l'ouverture physique du casier correspondant comme expliqué ci-dessus. Si le badge n'est pas reconnu, un message REFUS est envoyé pour signaler un accès non autorisé.

Cette architecture légère et modulaire permet une **gestion centralisée des droits d'accès**, tout en laissant la possibilité d'évoluer vers une base de données externe (MySQL, SQLite, etc.) ou un système d'authentification plus robuste à l'avenir.

IX. Implémentation microcontrôleur – serveur

L'implémentation technique du système repose sur trois éléments principaux : le microcontrôleur ESP32 pour la lecture des badges NFC, le serveur Python pour la gestion des droits d'accès, et la communication via le protocole MQTT sur un réseau Wi-Fi local.

1. Le microcontrôleur ESP32

Le programme de l'ESP32 est développé sous l'environnement Arduino. Il gère la lecture d'un badge via un lecteur NFC (DFRobot Gravity en mode I2C), la connexion au Wi-Fi, ainsi que l'abonnement/publication sur le broker MQTT.

Le microcontrôleur est configuré pour :

- Lire l'UID du badge présenté au lecteur NFC.
- Se connecter automatiquement au réseau Wi-Fi local.

- Publier l'UID lu sur le topic casier/ouverture.
- Réagir aux réponses reçues du serveur sur ce même topic (commande CASIER:x ou REFUS).
- Activer un relais pour ouvrir le casier si la commande reçue est valide.

2. Le Serveur Python et la Base de Données intégrée

Le serveur, exécuté sur une machine locale, est codé en Python à l'aide de la bibliothèque paho-mqtt. Il se connecte au broker MQTT et traite les messages entrants. À réception d'un UID, il le compare à une base de données locale (sous forme de dictionnaire Python) contenant les identifiants autorisés et les numéros de casiers associés.

Selon le cas :

- Si l'UID est reconnu, le serveur publie une commande CASIER:x.
- Sinon, il publie REFUS.
- Il ignore les messages de type CASIER:x pour éviter les boucles de publication.

Ce système centralisé permet une logique simple et évolutive de gestion d'accès.

3. Intégration Wi-Fi et MQTT

La communication entre le microcontrôleur et le serveur repose sur le protocole MQTT, adapté aux objets connectés pour sa légèreté et son fonctionnement en mode publication/abonnement.

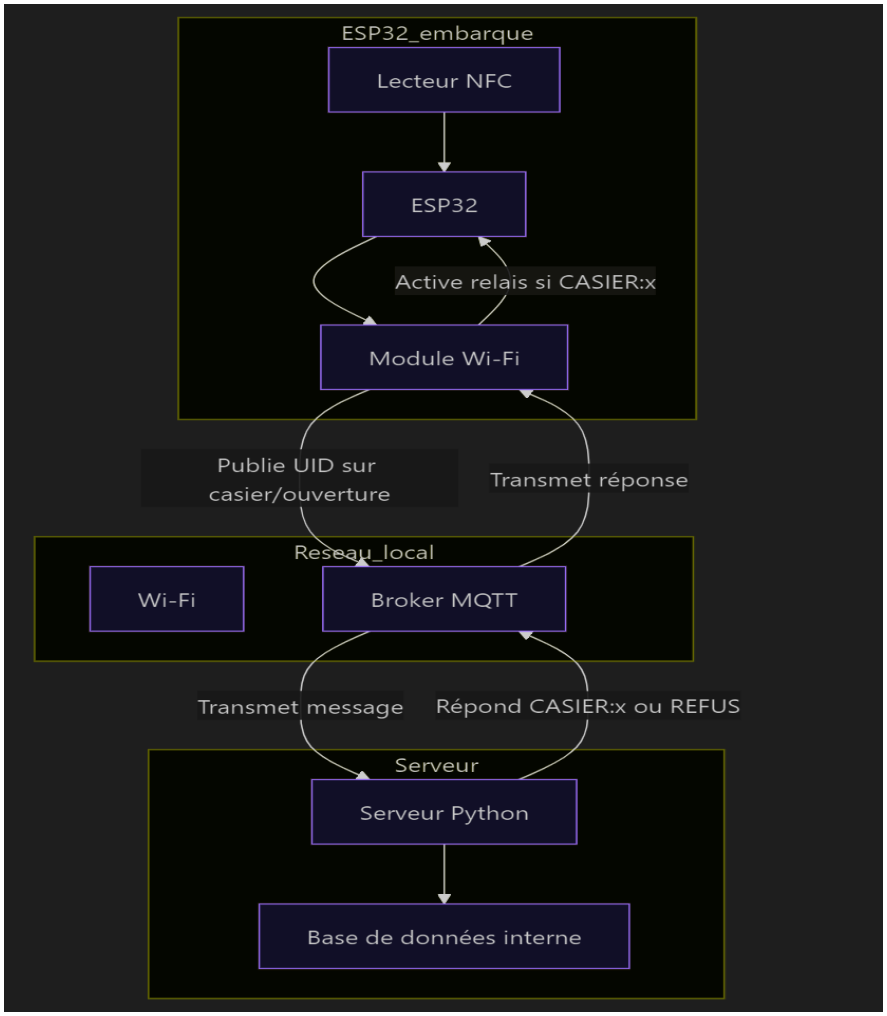
- **Wi-Fi** : l'ESP32 utilise une connexion Wi-Fi locale pour rejoindre le réseau du serveur.
- **MQTT** : l'ESP32 publie les identifiants des badges sur un topic dédié. Le serveur analyse ces messages et répond sur le même topic.
- Le broker utilisé est hébergé localement, sur l'ordinateur du développeur.

Cette architecture permet des échanges bidirectionnels efficaces entre les modules embarqués et le serveur de traitement.

4. Gestion des cas d'usage

Le système gère plusieurs scénarios principaux :

- **Badge valide** : l'ESP32 publie un UID reconnu, le serveur répond avec CASIER:x, ce qui déclenche l'ouverture du casier via un relais.
- **Badge invalide** : l'ESP32 publie un UID inconnu, le serveur répond par REFUS. Aucun relais n'est activé.
- **Déconnexion Wi-Fi ou MQTT** :
  - Côté ESP32 : des fonctions de reconnexion sont prévues pour tenter de rétablir la liaison automatiquement en cas de perte.
  - Côté serveur : les événements de déconnexion sont loggés, et la boucle MQTT reste active (loop_forever()).



- **Réception de commande non attendue** : l'ESP32 ignore les messages ne contenant pas un format attendu (CASIER:x), évitant ainsi les erreurs ou les actions non prévues.

X. Tests & Validations

1. Ciblage des fonctions critiques à tester

Les tests unitaires visent à vérifier individuellement les fonctions essentielles du système :

- lireCarte() : lecture d'un badge NFC
- envoyerUID() et recevoirRéponseServeur() : communication avec le serveur via MQTT
- ouvrirCasier(numero) : déclenchement du relais pour ouverture du casier
- Attribution UID → numéro de casier (dans la base de données)

2. Tests sur l'ESP32

a. Connexion Wi-Fi

Méthode :
Utilisation de la bibliothèque WiFi.h :

- WiFi.begin(ssid, password) pour se connecter.
- WiFi.localIP() pour vérifier l'attribution d'une IP.

Résultat attendu

Une adresse IP valide du même sous-réseau que notre PC (172.20.10.X)
Un SSID ou mot de passe erroné entraîne un échec de connexion → bon test de robustesse.

```
Carte réseau sans fil Wi-Fi :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv4. . . . . . . . . . . . . .: 172.20.10.9
   Masque de sous-réseau. . . . . . . . . : 255.255.255.240
   Passerelle par défaut. . . . . . . . . : 172.20.10.1
```

Resultat Obtenu

```
✅ WiFi connecté !
ESP connecte à l'adresse : 172.20.10.8
```

b. Connexion au Broker MQTT

Méthode :

Pour la connexion au MQTT, on commence par lancer mqtt sur l'invite de commande et autoriser les connexions extérieures, ensuite on procède relativement de la même manière.

Pour notre cas, la connexion se fait avec ce duo de commande :

cd C:\Program Files\mosquitto

mosquitto -v -c testconfig.txt

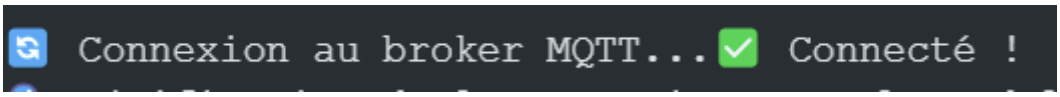
Avec la bibliothèque PubSubClient.h, on utilise la fonction client.setServer() pour se connecter au serveur mqtt en renseigner comme entrée le topic ou sujet du serveur et le port auquel il communique c'est-à-dire 1883.

Résultat attendu :

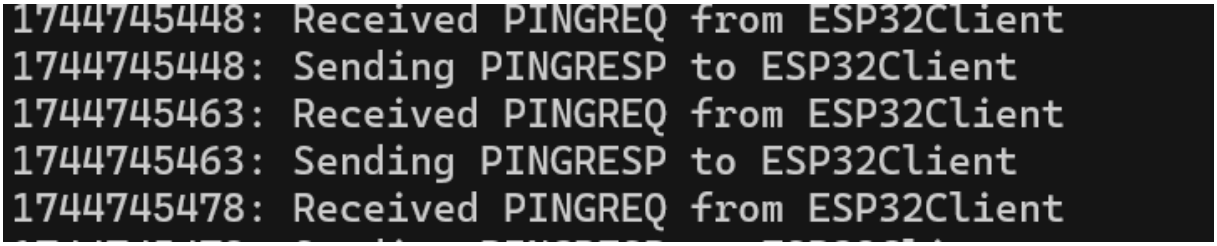
- Connexion confirmée dans les logs Mosquitto.
- En cas d'adresse ou port incorrect, échec de connexion → test d'erreur valide.

Résultat Obtenu :

Avec cette configuration, nous obtenons :



L'ESP arrive à se connecter à MQTT avec cette configuration et on peut observer les logs sur la console.

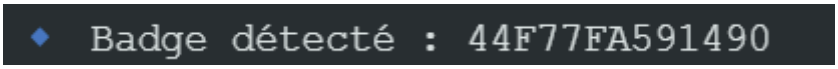


c. Lecture de badge NFC – lireCarte()

Tests effectués :

Scénario	Résultat
Badge NFC valide (ex: carte UBS)	UID correctement lu
Deux cartes en même temps	Une seule UID détectée
Carte sans NFC	Pas de détection
Carte emballée d'aluminium	Pas de détection (effet de blindage)

La carte, une fois détecté, retourne l'UID :



d. Ouverture du casier – ouvrirCasier(numero)

Principe de fonctionnement

À réception d'un UID valide, la fonction active le relais du casier correspondant.

Base de test :

badges_autorises = {

"44F77FA591490" : "1", # carte UBS

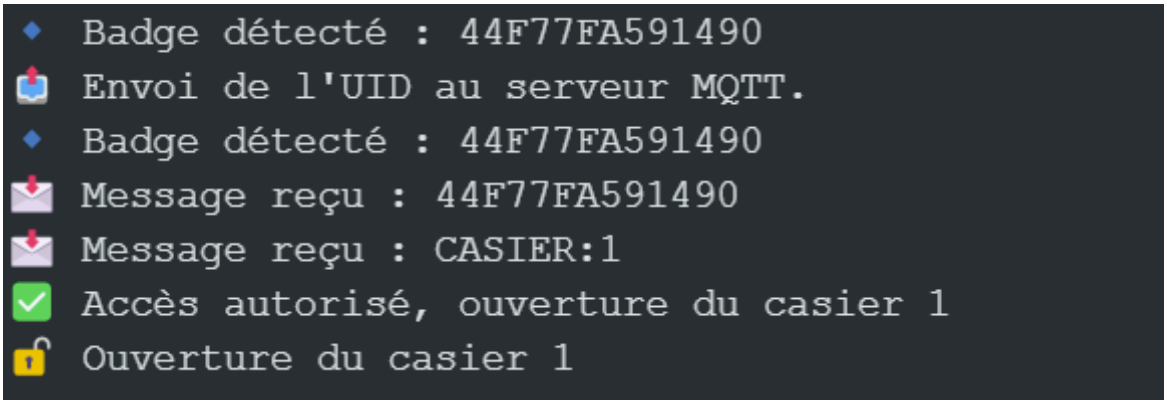
"40F1ED56" : "2" # tag NFC

}

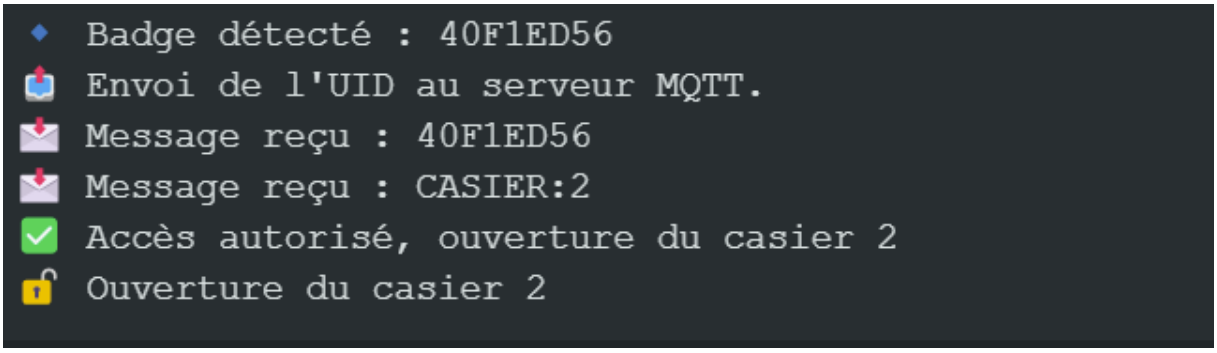
Tests effectués :

Badge	Casier attendu	Résultat
Carte UBS	Casier 1	Casier 1 activé
Tag NFC	Casier 2	Casier 2 activé
Badge inconnu	Aucun	Aucun relais déclenché

1er Test avec Carte UBS (casier 1)



2e Test avec tag NFC (casier 2)



3. Tests sur le serveur Python

a. Connexion au MQTT

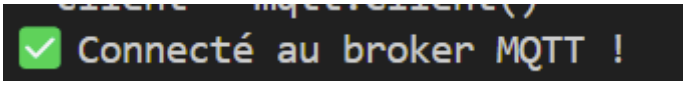
Bibliothèque : paho.mqtt.client

Vérification :

- client.connect(broker, port)
- Affichage des messages de connexion dans les logs Mosquitto (abonnement et publication)

Résultat attendu

Connexion réussie → pas d'erreurs visibles dans les logs.



b. Vérification UID / Casier – on_message()

Principe de fonctionnement

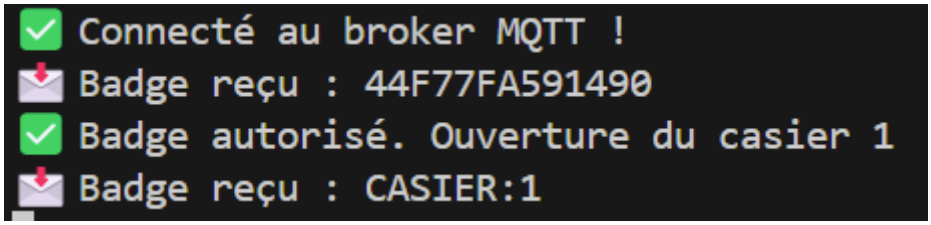
La fonction on_message() traite chaque message MQTT entrant contenant un UID.

Scénarios de test :

UID reçu	Présent en base ?	Réponse envoyée
"44F77FA591490"	Oui	"1"
"40F1ED56"	Oui	"2"
"XXINVALIDUIDXX"	Non	"REFUS"

Résultat attendu

Le serveur doit différencier correctement les badges autorisés et non autorisés, et publie une réponse adéquate.



4. Tests de cas limites

Cas	Comportement attendu
UID vide ou invalide	Ignoré ou message "REFUS"
Réponse serveur malformée ou vide	Aucun relais activé
Tentative d'ouverture de casier déjà ouvert	Aucun problème constaté, la commande est redondante mais sans risque
Déconnexion réseau (Wi-Fi ou MQTT)	Réessaie ou échec de transmission, selon configuration de PubSubClient

XI. Etude des performances du systèmes

1. Analyse de la mémoire

Pour notre projet, il est important de garantir une performance maximum du système en termes de mémoires. L'adéquate gestion de la mémoire garantie non seulement la stabilité et la réactivité du système, mais c'est aussi le facteur le plus important de la performance à long terme; néanmoins, des erreurs critiques, telles que les fuites de mémoire et les dépassements de pile, peuvent être faits et nuisent alors à la performance de l'ESP32 et du serveur.

L'ESP32, qui gère la communication avec les badges NFC et l'activation des casiers via relais, dispose de ressources limitées en mémoire vive (RAM) et mémoire flash. De même, le serveur, qui communique avec l'ESP32 via MQTT pour vérifier les badges et ouvrir les casiers, doit être optimisé pour gérer efficacement les connexions et les bases de données sans surcharger la mémoire.

Cette étude nous permet d'analyser et d'optimiser la gestion de la mémoire sur l'ESP32 et sur le serveur. En effet en surveillant la consommation de mémoire, en identifiant les points de consommation excessive et en appliquant des techniques d'optimisation adaptées, nous avons pu garantir un fonctionnement fluide et stable du système.

Dans cette partie, nous allons :

- Déterminer comment notre microcontrôleur utilise sa mémoire
- Détecter les surcharges potentielles
- S'assurer qu'une fuite mémoire ou dépassement de pile ne menace la stabilité du programme.

a. Mémoire interne ESP32

Le ESP32-D0WDQ6 contient deux microprocesseurs Xtensa® 32 bits LX6 à faible consommation d'énergie. La mémoire interne comprend :

- **448 KB de ROM** pour le démarrage et les fonctions principales.
- **520 KB de SRAM intégrée** pour les données et les instructions.
- **8 KB de SRAM dans le RTC**, appelée **mémoire RTC FAST**, utilisée pour le stockage de données ; elle est accessible par le processeur principal lors du démarrage RTC depuis le mode de veille profonde.
- **8 KB de SRAM dans le RTC**, appelée **mémoire RTC SLOW**, accessible par le coprocesseur pendant le mode de veille profonde.
- **1 Kbit de mémoire eFuse** : 256 bits sont utilisés pour le système (adresse MAC et configuration de la puce), et les 768 bits restants sont réservés pour les applications des utilisateurs, y compris le chiffrement de la mémoire flash et le CHIP-ID. ("Premiers pas avec la carte de développement ESP32 - Raspberryme")

b. Organisation de la mémoire & détection des fuites

L'ESP32 dispose de plusieurs zones mémoire :

- **Mémoire Flash** : pour le code et les données persistantes.
- **Mémoire RAM (SRAM)** : pour les variables, la pile, le tas (heap).

- **Mémoire Heap** : utilisée dynamiquement, par exemple pour les objets String, Wi-Fi, MQTT, etc.

c. Surveillance de la mémoire dynamique

Cette mémoire appelé *heap* permet les allocations dynamiques comme le malloc, les buffers MQTT ou encore la connexion Wi-Fi. Une baisse continue de la mémoire disponible peut indiquer une fuite.

- **Méthodes**

Pour faire cette étude, nous allons insérer dans la fonction loop() de notre code ESP32, ces lignes de code afin de suivre l'état de la mémoire dynamique :

Serial.print("Mémoire Heap disponible : ");

Serial.println(ESP.getFreeHeap());

- **Observation**

Au début du lancement, on a **232184,** puis la mémoire dynamique devient stable à **232472.**

```
📡 Version du firmware : 0x32010607
🎯 Lecteur NFC prêt !
Mémoire Heap disponible : 232184
Mémoire Heap disponible : 232472
Mémoire Heap disponible : 232472
Mémoire Heap disponible : 232472
```

Ceci montre que pour le moment la mémoire est stable et ne présente pas de fuite.

Maintenant nous allons vérifier la heap après lecture du badge et ouverture du casier.

```
Mémoire Heap disponible : 232516
Mémoire Heap disponible : 232516
🔹 Badge détecté : 44F77FA591490
📤 Envoi de l'UID au serveur MQTT.
Mémoire Heap disponible : 230632
Mémoire Heap disponible : 231876
📩 Message reçu : 44F77FA591490
Mémoire Heap disponible : 232516
📩 Message reçu : CASIER:1
✅ Accès autorisé, ouverture du casier 1
🔓 Ouverture du casier 1
Mémoire Heap disponible : 232516
Mémoire Heap disponible : 232516
Mémoire Heap disponible : 232516
```

Nous remarquons qu'avant lecture, la mémoire est stable à **232516 Ko.** Cependant, à l'ouverture, la mémoire chute considérablement à **230632.** Juste après, la mémoire disponible commence à remonter et redevient stable à **232516Ko.**

On en conclut que, la mémoire avant exécution est égale à celle après exécution. Ce qui justifie une absence de fuite dans la mémoire dynamique.

On note aussi que sur une longue durée sans événements, la mémoire ne diminue pas continuellement.

On peut donc conclure qu'il n'y a pas de fuite mémoire dans notre système.

d. Analyse de la mémoire Flash

La mémoire Flash nous permet de stocker en lecture seule le programme et certaines données. Il est donc utile de vérifier si on est proche de la limite.

- **Méthodes**

Pour vérifier la taille occupée, on utilisera dans la fonction setup() une fonction getter intégré :

Serial.print("Taille mémoire flash : ");

Serial.println(ESP.getFlashChipSize());

- **Observation**

```
Taille mémoire flash : 4194304
```

La mémoire totale de notre microcontrôleur est de **4Mo.** Ce qui implique que la taille de notre code ne doit pas dépasser **1,3Mo.**

La taille de notre programme :

Sur la console de l'IDE Arduino on peut observer :

```
Sketch uses 939726 bytes (71%) of program storage space. Maximum is 1310720 bytes.
Global variables use 47928 bytes (14%) of dynamic memory, leaving 279752 bytes for local variables. Maximum is 327680 bytes.
```

Ce qui nous montre la taille de notre programme qui est de **0,9Mo** sur **1,3Mo** de mémoire alloué au programme soit 71%. Donc la taille du code ne dépasse pas 1,3Mo qui correspond à la limite pour les modèles tel que le notre.

e. Vérification de la pile

L'ESP32 est multitâche et chaque tâche a sa propre pile ou stack en anglais. Un dépassement de stack peut provoquer un crash d'où l'intérêt de vérifier.

La vérification est principalement importante pour 3 raisons :

- L'utilisation d'opérations Wi-Fi et MQTT qui ont des librairies utilisant pas mal de ressources internes,
- La lecture NFC,
- La conversion de chaînes (ex: String badgeID),

En surveillant la pile, on s'assure que le système reste stable dans la durée et qu'on n'est pas dangereusement proches des limites. Ce qui est très important surtout dans le cadre des systèmes embarquées.

- **Méthodes**

Dans notre code, nous utilisons qu'une seule tâche *loop()*. Pour vérifier sa pile donc on insèrera la fonction dans cette fonction. On vérifiera aussi la pile restante avant la connexion au WiFi et MQTT.

Serial.print("Stack restante de loopTask : ");

Serial.println(uxTaskGetStackHighWaterMark(NULL));

Les précédentes lignes de code seront insérées dans le code pour vérifier la pile d'une tâche

- **Observation**

```
Stack restante de loopTask : 7212
..
✅ WiFi connecté !
ESP connecte à l'adresse : 172.20.10.8
🔄 Connexion au broker MQTT...✅ Connecté !
🔍 Vérification de la connexion avec le module NFC...
✅ Module NFC détecté !
📡 Version du firmware : 0x32010607
🎯 Lecteur NFC prêt !
Stack restante de loopTask : 6124
Stack restante de loopTask : 6124
Stack restante de loopTask : 6124
```

Cette capture nous montre qu'avant la connexion la pile était de **7212 octets** et après la connexion au WiFi et au MQTT, elle passe à **6124 octets.** Ce qui prouve l'utilisation de la pile dans les opérations Wi-Fi et MQTT.

```
Stack restante de loopTask : 6124
🔹 Badge détecté : 44F77FA591490
📤 Envoi de l'UID au serveur MQTT.
Stack restante de loopTask : 6124
Stack restante de loopTask : 6124
📩 Message reçu : 44F77FA591490
Stack restante de loopTask : 6124
📩 Message reçu : CASIER:1
✅ Accès autorisé, ouverture du casier 1
🔓 Ouverture du casier 1
Stack restante de loopTask : 5436
Stack restante de loopTask : 5436
Stack restante de loopTask : 5436
```

Ici, on remarque qu'après l'évènement lecture NFC qui contient une conversion en String de 4UID du badge détecté, la valeur de la pile baisse considérablement (de **6124** à **5436**) puis se stabilise à 5436 octets. Ceci montre donc qu'il n'y a pas de fuite ou de dysfonctionnement de la pile.

XII. Pistes d'amélioration et perspectives

Dans le cadre de l'évolution du projet des casiers intelligents, plusieurs axes d'amélioration et de développement peuvent être envisagés afin d'enrichir les fonctionnalités, de renforcer la sécurité, et de favoriser un déploiement à plus large échelle.

1. Implémentation d'une Double Authentification (2FA)

Afin de renforcer significativement la sécurité d'accès aux casiers connectés, nous proposons la mise en place d'une **authentification à deux facteurs (2FA)**. Ce mécanisme oblige l'utilisateur à prouver son identité à l'aide de deux éléments distincts, rendant les accès beaucoup plus difficiles à usurper.

Le processus envisagé est le suivant :

- **Premier facteur : authentification physique via badge NFC**
  L'utilisateur s'identifie dans un premier temps en scannant son badge NFC personnel sur le lecteur du casier. Ce badge doit être préalablement enregistré dans la base de données centrale.
- **Deuxième facteur : validation via code temporaire (OTP)**
  Une fois le badge reconnu, un **code temporaire** (One-Time Password, OTP) est généré automatiquement par le serveur et envoyé à l'utilisateur :
  - Soit par notification via une application mobile dédiée,
  - Soit par mail vers l'adresse institutionnelle de l'utilisateur (par exemple adresse universitaire).

  L'utilisateur devra ensuite entrer ce code sur une **interface mobile ou web** reliée au système pour valider définitivement l'ouverture du casier.

2. Interface mobile/web

Le développement d'une **interface utilisateur mobile ou web** offrirait une meilleure ergonomie et plus de flexibilité. Elle permettrait notamment aux usagers de :

- Réserver un casier à distance,
- Suivre l'état de disponibilité en temps réel,
- Recevoir des notifications lors de la mise à disposition ou du retrait d'un objet.

Une telle interface contribuera à une expérience utilisateur plus fluide et connectée.

3. Déploiement élargi

Enfin, le système de casiers intelligents a le potentiel d'être **déployé en dehors du cadre universitaire**, par exemple :

- Dans d'autres établissements d'enseignement supérieur (écoles, universités),
- Dans des entreprises pour la gestion de matériel partagé ou de consignes sécurisées,
- Dans des espaces publics (gares, bibliothèques, coworkings, etc.).

L'adaptation du prototype à des environnements variés nécessiterait une modularité accrue (nombre de casiers, types d'authentification, configuration logicielle) mais représenterait une excellente opportunité de valorisation du projet.

4. Mise en veille automatique pour limiter la consommation énergétique

Afin d'optimiser la consommation d'énergie du système de casiers intelligents, une **fonction de mise en veille automatique** peut être intégrée :

- Lorsque **aucune activité utilisateur** (pas de scan NFC, pas d'interaction réseau) n'est détectée pendant une période donnée (par exemple 5 minutes), l'ESP32 bascule automatiquement l'ensemble du système en **mode veille profonde**.
- Pendant ce mode, la majorité des composants (écran LCD, lecteur NFC, relais) seraient désactivés ou mis en mode faible consommation.
- Le système se **réactive automatiquement** dès qu'un événement externe est détecté, comme :
    - Une détection de badge NFC
    - La touche de l'interface utilisateur

XIII. Problèmes

1. Gestion et affichage avec l'écran LCD

L'un des principaux obstacles rencontrés au cours de notre projet a concerné l'utilisation de l'écran LCD. Dès le départ, il a été difficile d'obtenir une documentation fiable et complète sur ce composant. En effet, le constructeur ne proposait pas de datasheet officielle accessible sur son site, et les rares informations disponibles étaient présentées sous forme d'images dans des langues étrangères autres que l'anglais ou le français, ce qui rendait leur traduction et leur exploitation complexes.

Une autre difficulté importante a été l'identification de l'adresse I2C correcte pour notre écran. Contrairement aux valeurs standards attendues (telles que **0x27 ou 0x3F**), nous avons découvert une série d'adresses I2C inattendues qu'il a fallu tester manuellement une par une. Après plusieurs essais, l'adresse effective détectée s'est révélée être **0x78**, une valeur inhabituelle pour ce type de module, ce qui a soulevé des doutes sur la compatibilité réelle de notre I2C.

Une fois cette adresse trouvée, un autre souci est apparu : l'écran affichait des données incorrectes ou incohérentes, malgré un câblage validé à l'aide des schémas de l'ESP32 Devkit v4. Il était donc difficile d'identifier la cause : défaillance matérielle, incompatibilité logicielle, ou mauvaise gestion du protocole I2C.

Pour tenter de contourner ce blocage, nous avons opté pour **l'utilisation d'un autre écran LCD** mieux documenté. Cependant, ce dernier ne disposait pas d'un module I2C intégré et nécessitait une connexion en mode parallèle à 14 broches. De plus, ses broches de contrôle de rétroéclairage **(LED+)** et **(LED−)** n'étaient pas pré-soudées. Compte tenu de la nature partagée du matériel universitaire et par souci de prudence, nous avons choisi de ne pas souder ces broches, afin de ne pas endommager le matériel.

Le résultat fut un écran qui clignotait brièvement au démarrage puis s'éteignait, en raison de l'absence de rétroéclairage. Nous avons aussi tenté d'utiliser le premier écran LCD sans interface I2C, en mode parallèle classique. Néanmoins, ce mode exigeait également l'utilisation des broches **BLA (pin 19 : LED+)** et **BLK (pin 20 : LED−)**, qui, encore une fois, n'étaient pas soudées.

Ainsi, bien que le microcontrôleur transmettait correctement les données, l'absence de rétroéclairage empêchait toute lecture de l'information affichée, rendant l'écran inutilisable dans notre contexte.

Face à cette impasse, nous avons envisagé de **remplacer le module I2C** suspect en en commandant un nouveau, afin de vérifier si l'origine du dysfonctionnement venait de ce composant. Cependant, cette solution n'a pas pu être mise en œuvre en raison de **contraintes de temps** : la commande, l'intégration et la phase de tests/validation n'auraient pas pu être réalisées dans le délai imparti, la date butoir de remise du rapport final étant fixée au **2 mai**.

2. Gestion des publications

L'un des problèmes majeurs a concerné la **gestion des publications MQTT**. En effet, lors des premiers tests, une mauvaise implémentation du système de réponse provoquait une **publication en boucle continue** : l'ESP32 publiait un UID, le serveur répondait sur le même topic, ce qui déclenchait une nouvelle lecture côté ESP32, relançant ainsi la boucle indéfiniment.

Pour résoudre ce problème, nous avons mis en place un mécanisme de filtrage côté serveur afin d'**ignorer les messages de type CASIER:x** reçus sur le topic d'écoute. Ce filtrage conditionnel a permis de stabiliser la communication et d'éviter les réponses en chaîne non désirées.

XIV. Conclusion

1. Bilan du projet

Le casier intelligent visait essentiellement à développer une solution connectée permettant aux utilisateurs (étudiants, enseignants, personnel) d'échanger ou d'enregistrer des objets avec une sécurisation, une autonomie et fluidité dans l'utilisation. Grâce à une architecture sobre reposant sur l'ESP32, un lecteur NFC et un serveur MQTT local, le système a réussi avec succès à répondre aux exigences fonctionnelles de base : identification par badge, ouverture conditionnée du casier, et rappel d'état temps réel via la console.

Les principales étapes du projet, de la phase de conception à l'implémentation, ont permis de valider :

- la faisabilité technique avec des composants économiques et accessibles,
- la stabilité des communications entre le microcontrôleur et le serveur,
- la pertinence de l'approche centralisée via MQTT pour la gestion des accès.

Certaines limites ont toutefois été identifiées, notamment en matière de gestion des sessions multiples, de sécurité renforcée, ou encore de robustesse logicielle à long terme. Le projet reste néanmoins fonctionnel à son niveau de prototypage et ouvre la voie à de nombreuses extensions.

2. Apports personnels et techniques

Ce projet nous a permis d'acquérir et de consolider un ensemble de compétences tant sur le plan technique que personnel.

a. Apports techniques

Nous avons pu approfondir nos connaissances dans les domaines suivants :

- La programmation sur **ESP32**, incluant la gestion des entrées/sorties, la communication Wi-Fi, et l'interfaçage avec différents périphériques (NFC, relais, écran).
- L'utilisation d'un **lecteur NFC (PN532)** en mode passif pour l'identification des usagers par badge.
- La mise en œuvre d'un **système de communication MQTT** efficace entre le microcontrôleur et un serveur local Python.
- Le développement d'une **interface utilisateur embarquée** permettant de guider l'usager lors de l'interaction avec le casier.
- La structuration d'un projet en suivant une **démarche méthodologique claire (cycle en V)**, de l'analyse des besoins à l'implémentation.

b. Apports personnels

Sur le plan personnel, ce projet a renforcé notre :

- **Autonomie dans la recherche de solutions techniques** et la résolution de problèmes concrets,

- **Capacité à collaborer efficacement**, à répartir les tâches et à maintenir une cohérence globale dans le développement,
- Rigueur dans la **documentation, la validation des fonctions**, et l'organisation du travail selon une planification claire,
- Intérêt pour les **technologies de l'Internet des objets** et leur application dans des contextes réels (logistique, sécurité, automatisation).

XV. Références

Datasheets ESP32

https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

Lecteur NFC

https://wiki.dfrobot.com/Gravity:%20I2C%20&%20UART%20NFC%20Module%20SKU:%20DFR0231-H

Loquet Solenoïd

https://esp32io.com/tutorials/esp32-solenoid-lock

MQTT

https://esp32io.com/tutorials/esp32-mqtt

Relais

https://esp32io.com/tutorials/esp32-relay

RFID/NFC

https://esp32io.com/tutorials/esp32-rfid-nfc

Etat de l'art

https://lecasierfrancais.fr/

https://www.parcelpending.com/fr/

https://ipps.ucsd.edu/services/thetrove/about/index.html

Brochage ESP32

https://duckduckgo.com/?q=brochage+esp32&iax=images&ia=images&iai=http%3A%2F%2Falgo.tn%2Fesp32%2Fimg%2Fpins-carte-esp32.png

Logo SSI-UBS

https://duckduckgo.com/?q=logo%20ssi%20ubs%20Lorient&ko=-1&ia=images&iax=images&iai=http%3A%2F%2Fpluspng.com%2Fimg-png%2Fubs-png-ubs-faculte-ssi-logo-cmjn-2-250.png

Logiciels Utilisés

Pour les diagrammes : Draw IO & Mermaid

Pour la programmation : Arduino IDE, VS Code, Invite de Commandes Windows

Pour la saisie : Zotero, Obsidian, Word

XVI. Annexes

Les codes, datasheets et tout documents nécessaires à la reprise du projet est joint à ce rapport sous un fichier .zip