Lycée Alphonse BENOIT
Cours Victor HUGO
84800 L'ISLE SUR LA SORGUE
Téléphone : 04-90-20-64-20

ACADÉMIE D'AIX-MARSEILLE
Liberté
Égalité
Fraternité

BTS CIEL

Épreuve E6.2 – Session 2026

Dossier de revue finale

Projet DVIL

Damn Vulnerable Internet Lock



GitHub

https://gitlab.com/team-dvil26/projet-dvil-2026

https://framagit.org/dvil-2026/projet-dvil

GitLab

- Mohamed EL MADANI (IR1)
- Haitam El Kadmiri (IR2)
- Abdelrahmen Amdouni (ER)

Table des matières

Table des matières..... 2
Partie commune..... 5
1. Introduction & Contexte.....6
1.1 Expression du besoin..... 7
2. Spécifications et contraintes..... 7
2.1 Contraintes techniques.....7
3. Organisation et répartition des tâches.....7
4. Architecture Technique et Documentation..... 8
5. Suivi de projet..... 12
6. Synthèse et perspectives du projet..... 12
7. Glossaire.....12
Partie Personnelle EL MADANI Mohamed (IR1)..... 20
8. Objectifs..... 21
8.1 Rôle au sein de l'équipe et répartition des missions.....23
8.2 Tâches à réaliser (cahier des charges).....24
9. Planification Personnelle (diagramme de Gantt).....24
10. Analyse Technique.....25
10.1 Choix des technologies (interface web et clavier)..... 25
10.2 Contraintes techniques (matériels, logiciels et performances).....26
11. Réalisation Interface Web et journalisation..... 26
11.1 Implémentation de l'interface de supervision (API) [ANNEXE 3].....26
11.1.1 Fonctionalités Principales.....27
11.1.2 Exemple d'implémentation de la logique côté API : Enrôlement Badge..... 28
11.2 Modernisation et Migration : L'Architecture de Supervision Conteneurisée..... 30
11.2.1 Alignement Fonctionnel et Rôles Utilisateurs..... 30
11.2.2 Une Infrastructure Modulaire et Isolée (Docker)..... 31
11.3 La Chaîne de Commande IoT (Du Clic au Matériel)..... 32
11.4 Synchronisation des Données et Audit..... 33
11.5 Intégration de la logique de journalisation.....35
11.6 Sécurisation et Gestion des Flux.....40
11.6.1 Passage du protocole HTTP/S et création de certificat.....41
11.6.2 Mise en œuvre de Let's Encrypt.....41
11.6.3 Configuration de la Passerelle Nginx [annexe 6]..... 42
11.6.4 Résolution de Nom et Identité Numérique.....44
11.7 Configuration de l'Environnement et Sécurité.....46
11.8 Gestion des Sessions & Sécurité..... 47
11.9 Maintenance & Sauvegarde.....49
11.9.1 Gestion des erreurs et retour utilisateur.....49
11.9.2 Gestion des erreurs et expérience utilisateur..... 50
11.10 Gestion des Bases de Données [Annexe 5].....51

11.10.1 Serveur de supervision....51
11.10.2 Nœud de contrôle local (Raspberry Pi Zero) [Annexe 5]....52
11.10.3 Architecture de synchronisation (API REST)....53
11.11 Implémentation des challenges (CTF)....55
12. Sécurisation et bonnes pratiques....63
13. Développement de l'interface clavier [Annexe 8]....64
13.1 Protocole de communication I2C....66
13.2 Faille de vulnérabilité du clavier....67
13.3 Développement de l'application clavier....68
14. Partie Physique : Étude de la Couche Physique de Supervision (Wi-Fi 2,4 GHz)....70
14.1 Étude de la propagation et bilan de liaison Wi-Fi....70
14.2 Développement et Calculs Théoriques....70
15. Conclusion et perspectives du projet DVIL....73
15.1 Bilan personnel....73
16. Partie Spécifique IR2 : Interface NFC et Enrôlement (Haitam EL KADMIRI)....61
Cahier des charges et tâches à réaliser....63
16.1 Planification (Gantt)....64
16.2 Objectifs de la mission....65
16.1.1 Présentation du poste d'accueil....66
Présentation du poste d'accueil (Système d'enrôlement)....66
16.1.2 Développement des services IoT....68
Service de détection et lecture NFC (app-python-nfc)....68
Rôle technique et Hardware Ce service logiciel, fonctionnant en tâche de fond, assure la surveillance asynchrone du lecteur matériel PN532. La communication est établie via le bus SPI matériel de la carte Raspberry Pi (utilisant spécifiquement les broches physiques 19/MOSI, 21/MISO, 23/SCLK, et 24/CE0). Le choix du bus SPI garantit une communication Full-Duplex, minimisant la latence lors de la détection continue du badge....68
Algorithme de sécurité et logique....68
Service de pilotage de l'actionneur (app-python-serrure)....69
Structure mémoire des badges MIFARE Classic....72
Module Explorateur — Cartographie hexadécimale des 16 secteurs de la puce MIFARE 1K....73
Bilan technique....75
16.3 Analyse Technique (NFC, SPI, Mifare)....76
Environnement de développement....76
16.4 Journal des Incidents et Résolutions (Focus IR2)....77
Incident : Échec du clonage sur carte "Magic" CUID....78
Incident : Crash de l'IHM lors de l'affichage des logs....78
Incident : Collision sur le bus SPI et "Freeze" de l'interface....78
Interface de clonage intégral — Validation de l'algorithme d'injection de données sur badge CUID 81
16.5 PARTIE PHYSIQUE : Comparaison RFID et NFC....82
Positionnement technologique....82
Principe physique : Le couplage inductif....82

Justification du choix NFC pour DVIL....82
16.6 Conclusion....85
Partie Personnelle Abdelrahmen Amdouni (ER)....83
17-Objectifs....84
18-Planification....84
19-Matériel....84
20-Diagramme des blocs internes....86
21-Bus utilisés....87
21.1- Le bus I2C....87
21.2 Structure d'une trame I2C....88
21.3- Le bus SPI....89
22-réalisation....90
22.1- Câblage rapide de l'écran....92
23-Processus de fabrication....95
23.1 saisie de schéma....95
23.2 Routage de la carte....100
24-Fiche de commande des composants....108
Annexes....110
Annexes IR1....111
Annexe 1 : Vue de l'interface web et API....111
Annexe 2 : Dictionnaire des Endpoint....114
Annexe 3 : Documentation Utilisateur/ Administrateur....115
Annexe 4 : Mise en place de l'environnement virtuel Python....117
Annexe 5 : Bases de données....119
Annexe 6 : Infrastructure Réseau, DNS et Sécurisation HTTPS....123
Annexe 7 : Structure du Projet et Déploiement Docker....125
Annexe 8 : Guide de programmation du code clavier de l'Arduino à l'ATtiny1616....129
Annexe 9 : Fiche recette et audit de sécurité :....140
ANNEXES IR 2....142
Annexe 1 : import de fichier json pour cloner un badge....143
Annexe 2 : UUID....147

Partie commune

1. Introduction & Contexte

Dans le cadre du Brevet de Technicien Supérieur **Cybersécurité, Informatique et Réseaux, Électronique (BTS CIEL)** au lycée Alphonse Benoit, le développement de compétences opérationnelles en cybersécurité constitue un axe stratégique de formation. Face à l'essor des objets connectés (IoT) dans les environnements professionnels et industriels, il devient essentiel de former les étudiants à l'identification, l'analyse et la correction de vulnérabilités sur des systèmes réels.

Cependant, les supports pédagogiques actuellement utilisés reposent majoritairement sur des environnements simulés, limitant l'exposition des étudiants à des problématiques concrètes telles que la sécurité matérielle, les protocoles embarqués, la gestion des accès ou encore les attaques sur équipements physiques.





Ainsi, DVIL constitue un support technique professionnalisant, aligné sur les exigences du secteur de la cybersécurité et répondant aux besoins croissants en compétences spécialisées dans la sécurisation des objets connectés.

1.1 Expression du besoin

Cette section présente les besoins de base identifiés et partagés par l'ensemble des membres du projet :

Fonctionnalités essentielles :

- Contrôle de la serrure à travers différentes interfaces : clavier matriciel, badge NFC et interface web.
- Journalisation complète des actions, permettant le suivi détaillé des tentatives et opérations effectuées.
- Gestion des communications entre les modules, assurée via les protocoles MQTT et bus I2C/SPI.

2. Spécifications et contraintes

2.1 Contraintes techniques

- **Matérielles :** Assurer la compatibilité entre le microcontrôleur (Raspberry Pi / ATtiny), les modules NFC, le clavier matriciel et la serrure.
- **Logiciels :** Utilisation de frameworks éprouvés tels que Flask et SQLAlchemy, en veillant particulièrement à la sécurité des communications via HTTPS/TLS et à l'authentification par JWT.
- **Performances :** Garantir une réactivité optimale du système de commande de la serrure, avec un temps de réponse inférieur à 1 seconde, et une gestion efficace des journaux d'activité.
- **Budget :** Le projet doit être réalisé dans la limite de 150 euros.

3. Organisation et répartition des tâches

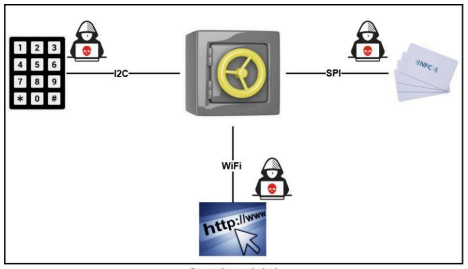
La répartition des tâches entre les différentes équipes est définie comme suit :

- **Étudiant IR1 :** Développement de l'interface web (API REST), gestion du clavier matriciel et journalisation des actions.
- **Étudiant IR2 :** Intégration de l'interface NFC et commande de la gâche électrique via MQTT.
- **Étudiant ER :** Conception électronique, câblage et mise en place de la carte (HAT) supportant l'ensemble du système ainsi que la carte clavier comportant le module NFC et le clavier couplés à l'écran OLED.

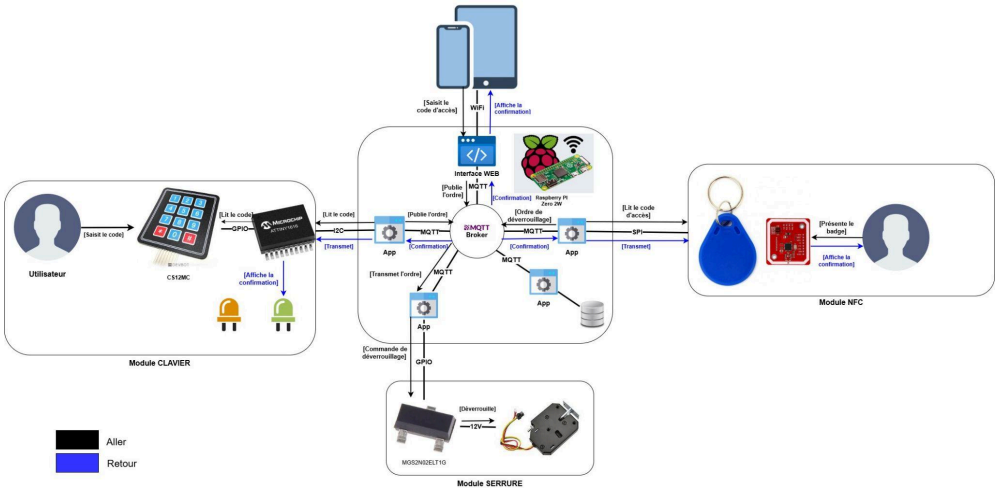
Chaque équipe collabore étroitement afin d'assurer la cohérence globale du projet et la communication efficace entre les différents modules, en utilisant des protocoles standards tels que MQTT, I2C et SPI.

4. Architecture Technique et Documentation

Le schéma présenté ci-dessous illustre l'architecture globale de la plateforme DVIL en 2025. Il met en évidence la répartition des différents modules, leurs interactions ainsi que le flux complet des opérations, depuis l'authentification de l'utilisateur jusqu'au contrôle physique de la serrure.

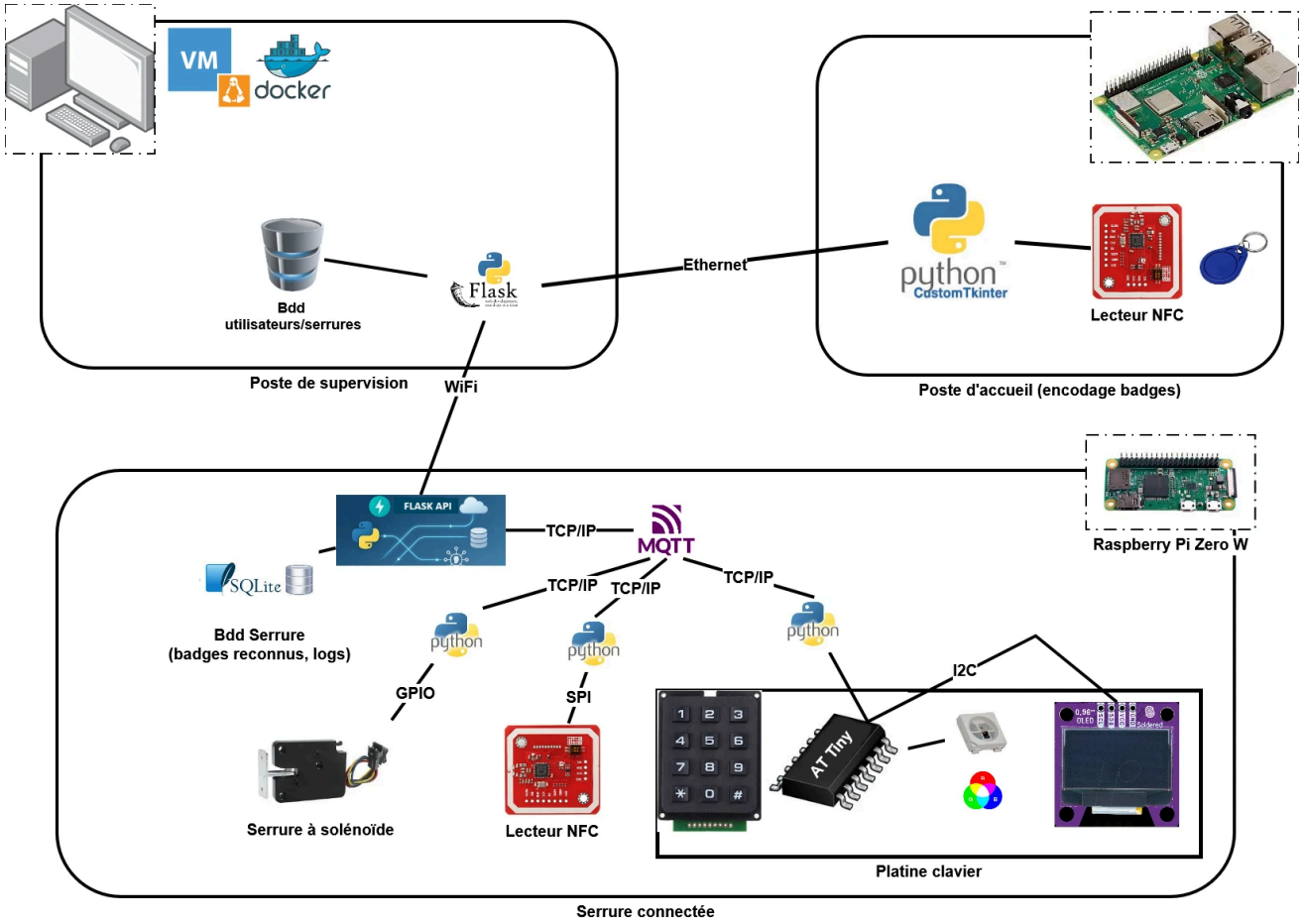


Synoptique général



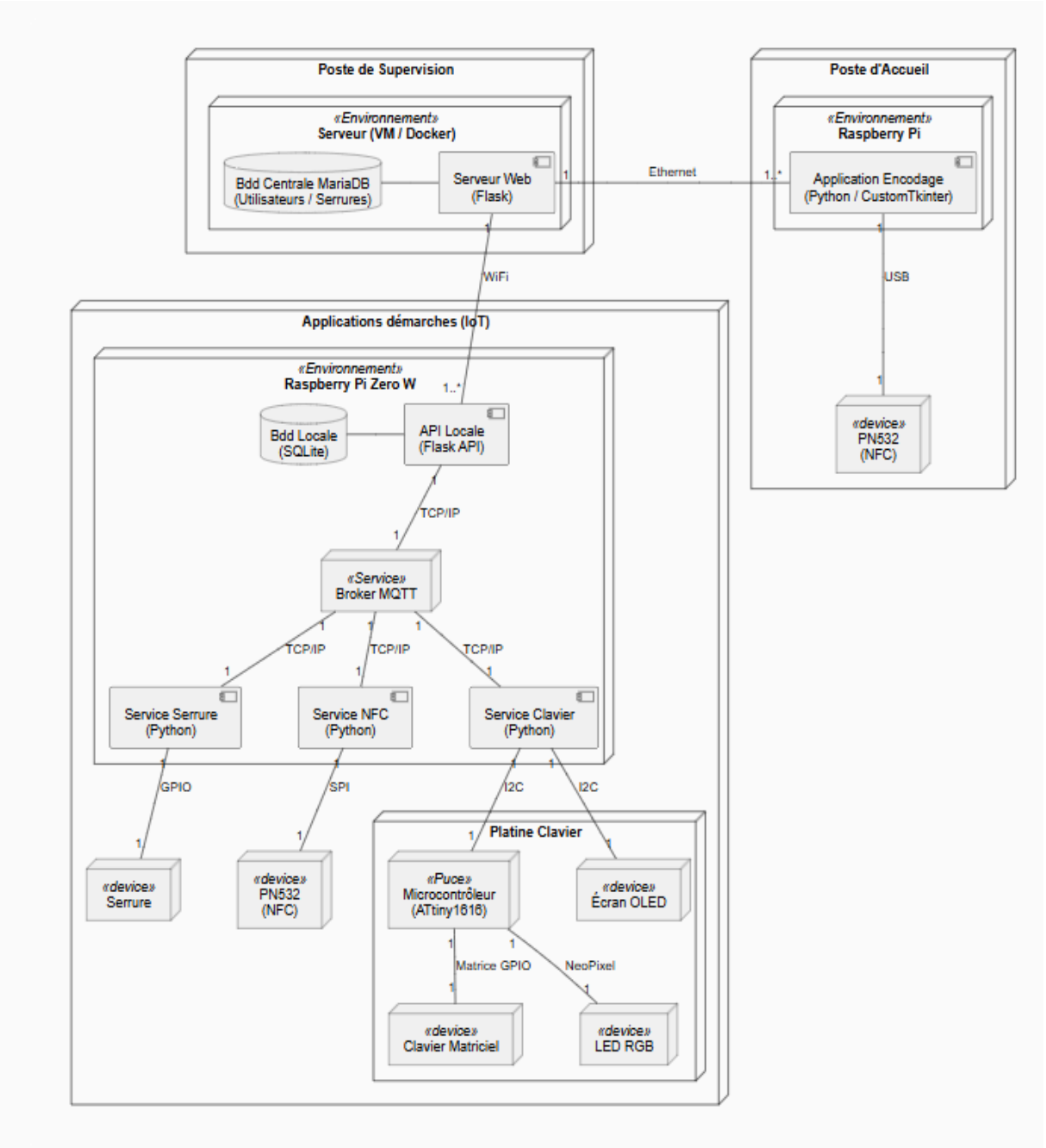
Pour le projet actuel (2026), l'architecture globale a été mise à jour afin d'intégrer des nouveaux éléments comme un clavier matriciel plus robuste, un **écran OLED SSD1306** et une **LED RGB (Neopixel)** en remplacement des deux leds verte et rouge.

Le synoptique ci-dessous présente la nouvelle architecture globale du projet **DVIL** 2026 actuellement :

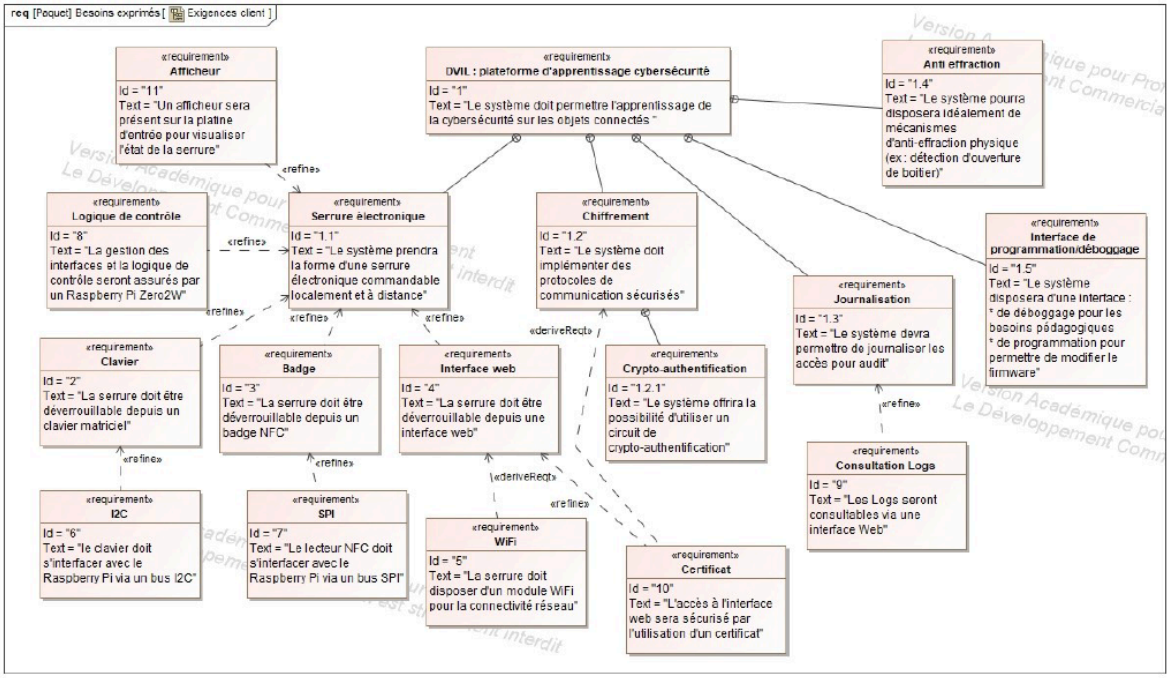


L'évolution de notre architecture marque le passage d'un système centralisé vers une structure distribuée plus robuste. En séparant différents postes (comme la supervision (VM) et l'accueil **NFC**) de l'exécution des applications des modules **IoT** (depuis la Raspberry Pi Zero 2W), nous optimisons la charge de travail de chaque unité et supprimons les risques de surcharge matérielle. Cette dissociation assure une meilleure performance globale et garantit surtout une autonomie totale de la serrure, qui demeure opérationnelle même en cas d'indisponibilité du serveur central.

Pour mieux préciser, ce diagramme de déploiement montre l'infrastructure globale du système pour ce projet :



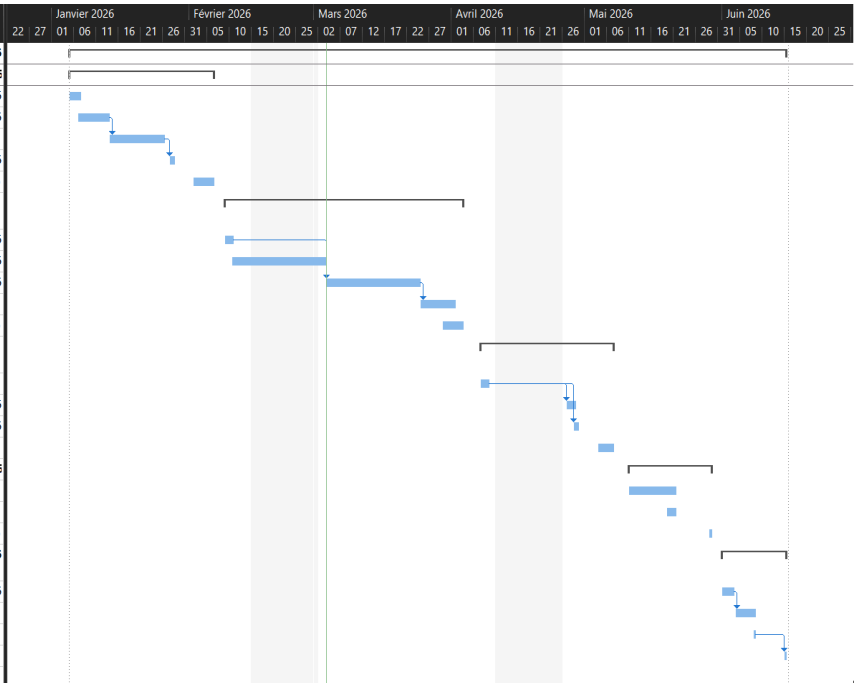
Ce diagramme d'exigences ci-dessous formalise les besoins fonctionnels et techniques de l'infrastructure DVIL, garantissant une traçabilité rigoureuse entre les objectifs pédagogiques de cybersécurité et les solutions matérielles et logicielles implémentées :



Le diagramme de Gantt présenté ci-dessous illustre la planification et le suivi des différentes tâches réalisées tout au long du projet. Il met en évidence la répartition temporelle des activités, les phases clés de développement ainsi que l'enchaînement des travaux jusqu'à l'achèvement du projet :

DIAGRAMME DE GANTT

	Mode Tâche	Nom de la tâche	Durée	Début	Fin
1		**PROJET GLOBAL DVIL**	**150h**	**Lun 05/01/26**	**Lun 15/06/26**
2		**Étude et Conception**	**40h**	**Lun 05/01/26**	**Ven 06/02/26**
3		Étude du cahier des	10h	Lun 05/01/26	Mer 07/01/26
4		Spécification des ex	12h	Mer 07/01/26	Mar 13/01/26
5		Conception de l'arcl	12h	Mer 14/01/26	Lun 26/01/26
6		Planification prévisi	6h	Mar 27/01/26	Mer 28/01/26
7		Revue 1	30min	Lun 02/02/26	Ven 06/02/26
8		**Conception Détaillée et Réalisation**	**70h**	**Lun 09/02/26**	**Ven 03/04/26**
9		Préparation environ	6h	Lun 09/02/26	Mar 10/02/26
10		Conception détaillé	14h	Mar 10/02/26	Mar 03/03/26
11		Réalisation des prot	35h	Mer 04/03/26	Mar 24/03/26
12		Tests unitaires	15h	Mer 25/03/26	Jeu 02/04/26
13		Revue 2	30min	Lun 30/03/26	Ven 03/04/26
14		**Intégration et Validation**	**20h**	**Mar 07/04/26**	**Jeu 07/05/26**
15		Intégration globale	8h	Mar 07/04/26	Jeu 09/04/26
16		Analyse des vulnéra	6h	Lun 27/04/26	Mar 28/04/26
17		Campagne de tests	6h	Mar 28/04/26	Mer 29/04/26
18		Revue 3 (certificativ	30min	Lun 04/05/26	Jeu 07/05/26
19		**Documentation**	**12h**	**Lun 11/05/26**	**Ven 29/05/26**
20		Rédaction de dossie	8h	Lun 11/05/26	Jeu 21/05/26
21		Remplissage des fich	4h	Mar 19/05/26	Jeu 21/05/26
22		Remise du projet	0h	Ven 29/05/26	Ven 29/05/26
23		**Clôture et Soutenance**	**10h**	**Lun 01/06/26**	**Lun 15/06/26**
24		Bilan et supports	6,5h	Lun 01/06/26	Mer 03/06/26
25		Preparaion de soute	2,5h	Jeu 04/06/26	Lun 08/06/26
26		Soutenance finale	1h	Lun 08/06/26	Lun 08/06/26
27		Livraison	0h	Lun 15/06/26	Lun 15/06/26



5. Suivi de projet

Le suivi de projet par l'ensemble des membres de l'équipe s'appuie sur différents éléments :

- Une planification détaillée, reposant sur un diagramme de Gantt et un découpage précis des tâches, permettant une optimisation du temps et des ressources.
- Echanges sur Discord (screenshot), compte rendu réunion pour trouver une solution à un problème technique
- L'exploitation de la forge logicielle **Framagit** comme référentiel centralisé qui garantit la traçabilité des différents codes sources applicatifs mais aussi des autres documents (schémas) via un versionnage distribué (**Git**).

6. Synthèse et perspectives du projet

Le projet DVIL constitue une expérience collective particulièrement enrichissante, ayant permis de mettre en application nos compétences en cybersécurité, en électronique et en développement IoT. Il nous a offert l'opportunité de consolider nos acquis, d'approfondir nos connaissances techniques et de développer notre capacité à travailler en équipe dans un contexte proche des réalités professionnelles.

La collaboration, la complémentarité des compétences et l'entraide ont été des facteurs déterminants dans la réussite du projet, notamment face aux contraintes de temps et de budget imposées. Cette réalisation nous a également sensibilisés à l'importance d'une approche « security by design », ainsi qu'à la nécessité d'une documentation rigoureuse et structurée tout au long du cycle de développement.

Nous tenons enfin à remercier nos encadrants pour leur accompagnement, leurs conseils et leur soutien, qui ont largement contribué à la bonne conduite et à l'aboutissement de ce projet.

7. Glossaire

Vocabulaire technique	Description
DVIL	**- Serrure connectée volontairement vulnérable, conçue pour apprendre la cybersécurité.**
IoT (Internet of Things)	**- Objets du quotidien connectés à internet pour échanger des données.**
Interface Web	**- Page ou application accessible dans un**

	navigateur pour interagir avec un service.
API REST (Application Programming Interface)	- Interface permettant aux applications de communiquer via des requêtes HTTP.
Module NFC (Near Field Communication)	- Capteur pour échanger des données sans contact sur une courte distance (badges, cartes).
Raspberry Pi	- Mini-ordinateur peu coûteux, populaire pour les projets d'électronique et de programmation.
Raspberry Pi Zero 2W	- Mini-ordinateur très compact et peu coûteux, doté du Wi-Fi et du Bluetooth, idéal pour des projets électroniques et embarqués.
Journalisation (Logs)	- Processus d'enregistrement automatique d'événements ou d'actions qui se produisent dans un système informatique (application, serveur, réseau, système d'exploitation, etc.).
TLS/HTTPS	- Protocole chiffré protégeant les échanges sur le web contre l'interception de données.
Edge Computing	- Approche qui consiste à traiter les données au plus près de leur source (appareils, capteurs) plutôt que dans un cloud distant, afin de réduire la latence et améliorer la réactivité.
Docker	- Plateforme de conteneurisation s'appuyant sur les mécanismes d'isolation du noyau Linux (espaces de noms, groupes de contrôle) pour exécuter des applications encapsulées de manière reproductible.
Nginx	- Serveur web et reverse proxy événementiel, utilisant une architecture asynchrone non bloquante pour gérer un grand nombre de connexions simultanées avec une faible consommation de ressources.
Reverse Proxy	- Composant intermédiaire côté serveur qui

	reçoit les requêtes clients, les redirige vers un ou plusieurs serveurs backend, puis renvoie la réponse, permettant répartition de charge, mise en cache et sécurisation des services.
MQTT (Message Queuing Telemetry Transport)	**- Protocole de messagerie léger utilisé pour le pilotage en temps réel de la serrure entre l'API locale et les scripts de contrôle matériel.**
Bus I2C (Inter-Integrated Circuit)	**- Bus de communication à deux fils (données et horloge) pour relier plusieurs composants.**
Bus SPI (Serial Peripheral Interface)	**- Protocole de communication rapide utilisant plusieurs fils (MOSI, MISO, SCK, SS).**
Clavier matriciel	**- Dispositif d'entrée constitué d'une grille de lignes et de colonnes, où chaque touche correspond à une intersection, détectée par balayage électrique pour identifier l'appui.**
Module NFC (Near Field Communication)	**- Capteur pour échanger des données sans contact sur une courte distance (badges, cartes).**
Arduino	**- Carte électronique programmable utilisée pour réaliser des projets embarqués, permettant de contrôler des capteurs et des actionneurs via un microcontrôleur simple à programmer.**
Arduino IDE	**- Logiciel permettant de programmer et téléverser du code sur une carte Arduino.**
Serrure	**- Dispositif mécanique ou électronique permettant de verrouiller une porte ou un accès, en empêchant son ouverture sans clé, code ou authentification appropriée.**
Gâche électrique	**- Mécanisme électromécanique permettant d'ouvrir ou bloquer une porte à distance.**
Relais	**- Interrupteur commandé électriquement**

	pour activer ou désactiver un circuit.
Authentification JWT (JSON Web Token)	- Méthode d'authentification basée sur un jeton signé utilisé pour vérifier l'identité d'un utilisateur sans session côté serveur.
XSS (Cross-Site Scripting)	- Faille web permettant d'injecter du code malveillant exécuté dans le navigateur de l'utilisateur.
Challenge CTF	- Compétition de cybersécurité consistant à résoudre des défis pour récupérer des "flags".
Flag	- Chaîne secrète à trouver pour valider un défi de cybersécurité.
Firmware	- Logiciel intégré au matériel qui en assure le fonctionnement de base.
Cross-compilation	- Compilation d'un programme pour une autre architecture que celle utilisée.
Python	- Langage de programmation informatique et polyvalent.
Flask (Python)	- Micro-framework Python pour créer des applications web, serveurs et API légères.
C++/Qt	- Langage (C++) et framework (Qt) pour créer des applications et interfaces graphiques multiplateformes.
Diagramme de Gantt	- Planning visuel des tâches d'un projet dans le temps.
Diagramme de séquence	- Diagramme UML qui représente l'ordre chronologique des interactions entre objets ou composants d'un système.
UML (Unified Modeling Language)	- Langage de modélisation visuelle pour représenter un système logiciel.

Schéma de routage	**- Schéma montrant les chemins de circulation des données dans un réseau.**
Master/Slave (Maître/Esclave)	**- Architecture où un maître démarre un contrôle de un ou plusieurs esclaves qui vont répondre.**
Driver	**- Logiciel qui permet au système d'exploitation de communiquer avec un périphérique matériel.**
Payload	**- Contenu utile d'un message ou d'une attaque, contenant les données ou instructions effectivement exécutées ou transmises.**
WebSocket (Socket.IO)	**- Protocole et bibliothèque permettant une communication bidirectionnelle en temps réel entre client et serveur.**
Bcrypt	**- Algorithme de hachage sécurisé des mots de passe.**
SSTI (Server-Side Template Injection)	**- Faille web permettant l'injection de code dans un moteur de templates côté serveur.**
SQL Injection (SQLi)	**- Faille permettant d'injecter des requêtes SQL malveillantes dans une base de données.**
IDOR (Insecure Direct Object Reference)	**- Faille d'accès où un utilisateur peut accéder à des ressources non autorisées en manipulant une référence directe.**
ATtiny1616 (Microship)	**- Microcontrôleur AVR compact utilisé pour des systèmes embarqués simples et à faible consommation.**
ATECC 608	**- Composant sécurisé dédié aux opérations cryptographiques (stockage de clés, authentification, signatures).**

MPLAB	**- Environnement de développement pour programmer et déboguer des microcontrôleurs Microchip.**
PICkit 4	**- Programmateur et débogueur pour microcontrôleurs Microchip.**
CH340	**- Convertisseur USB ↔ série utilisé pour programmer ou communiquer avec des microcontrôleurs.**
Afficheur OLED ssd1306	**- Contrôleur d'écran OLED utilisé pour piloter de petits afficheurs monochromes via I2C ou SPI.**
LED Adafruit NeoPixel	**- Led RVB (Rouge / Vert / Bleu) adressable individuellement pour effets lumineux.**
VM (Virtual Machine)	**- Environnement logiciel qui émule un ordinateur complet pour exécuter un système d'exploitation isolé sur une machine hôte.**
Enrôlement NFC	**- Enregistrement d'un badge NFC dans un système d'accès.**
Poste d'accueil	**- Station de connexion pour alimenter et relier un appareil à des périphériques.**
Poste de supervision	**- Système central permettant de surveiller et contrôler des équipements ou processus.**
MariaDB/SQLite	**- Systèmes de gestion de bases de données relationnelles, MariaDB serveur multi-utilisateur et SQLite embarqué sans serveur.**
SQL	**- Langage de requêtes pour interroger et manipuler des bases de données.**
SQLAlchemy	**- Bibliothèque Python permettant d'interagir avec des bases de données via SQL et des objets.**
DuckDNS	

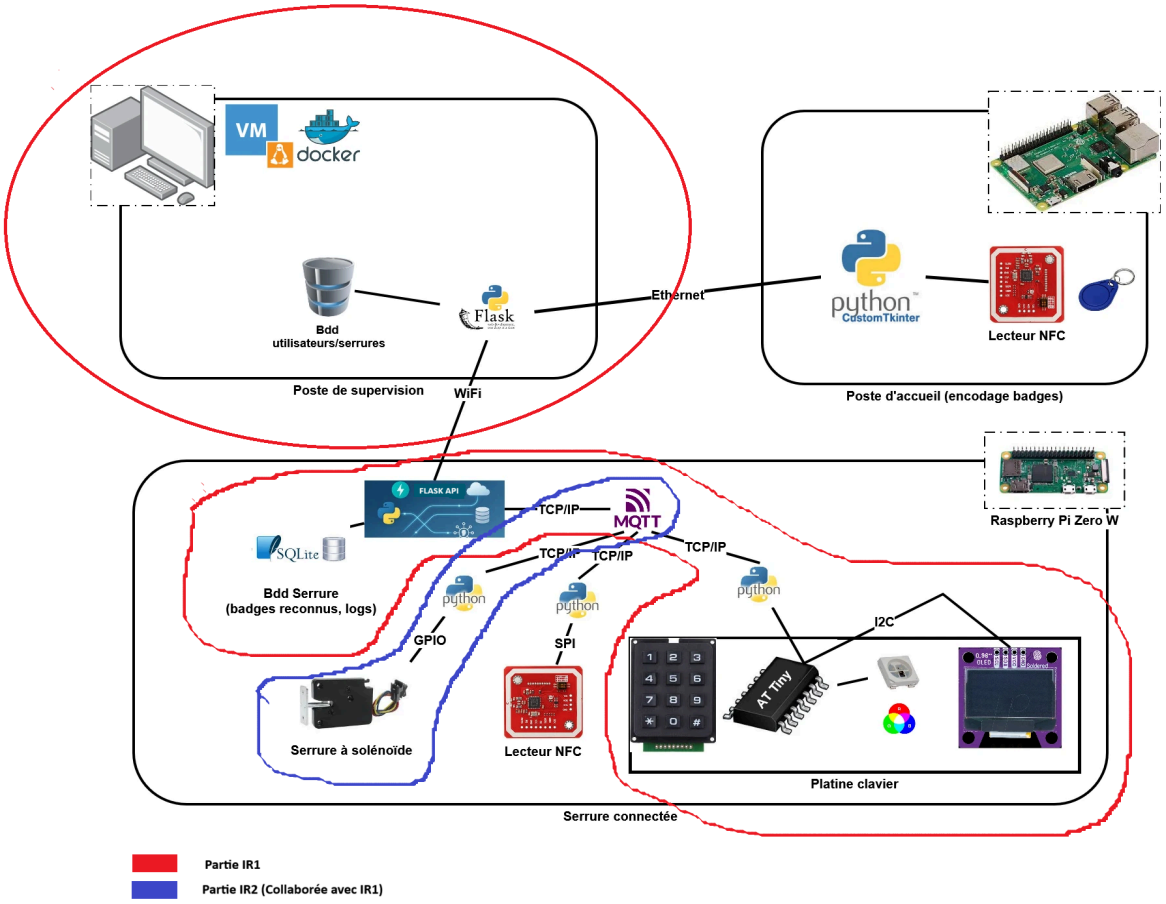
DNS	**- Service gratuit de DNS dynamique pour lier un nom de domaine à une IP changeante.**
Gateway (Passerelle)	**- Système qui traduit les noms de domaine (ex : google.com) en adresses IP.**
Let's Encrypt	**- Point d'entrée qui fait l'intermédiaire entre deux réseaux ou services.**
Serveur	**- Service gratuit qui fournit des certificats SSL/TLS pour sécuriser un site web (HTTPS).**
Software (logiciel)	**- Ordinateur (ou logiciel) qui fournit des services ou des données à d'autres machines (clients) sur un réseau.**
Proxy	**- Programme informatique qui permet à un ordinateur d'exécuter des tâches.**
Edge (périphérie)	**- Intermédiaire entre un client et un serveur.**
Service	**- Traitement des données au plus près de leur source, hors du cloud central.**
Breadboard	**- Programme qui fournit une fonction utilisable par d'autres systèmes ou utilisateurs.**
Flash Program	**- Plaque sans soudure qui permet de monter et tester des circuits électroniques rapidement.**
Conteneur (Docker)	**- Logiciel qui écrit (programme) une mémoire flash pour y installer ou mettre à jour un système embarqué (firmware).**
Image (Docker)	**- Instance exécutée d'une image, isolée et autonome.**
ISO	**- Modèle en lecture seule contenant une application et ses dépendances.**
	- Fichier image contenant une copie exacte d'un support (souvent un disque) pour installation ou sauvegarde.

Partie Personnelle

EL MADANI Mohamed (IR1)

8. Objectifs

Le synoptique ci-dessous illustre mes parties concrètes à réaliser et développer :



Dans le cadre de l'évolution du projet **DVIL** (**Damn Vulnerable Internet Lock**), mon rôle en tant qu'étudiant IR1 consiste à moderniser et sécuriser l'infrastructure logicielle du **poste de supervision** assurant la commande de la **gâche** via **interface web** ainsi que la validation des accès réalisés depuis le **clavier** et le **badge NFC**.

Mon travail consiste à faire évoluer un ancien système vers une architecture plus **flexible et évolutive**, à améliorer les interfaces de commande et à moderniser le système de journalisation des événements. Mes objectifs principaux sont :

1 - DÉVELOPPEMENT DES INTERFACES DE COMMANDE ET API

- **Interface Physique (Clavier) :** Concevoir et implémenter la gestion des saisies sur clavier matriciel (4x3), assurer la validation des codes et la remontée d'état via I2C / GPIO, mise en place de l'affichage de l'écran OLED qui permet une visualisation de la saisi du clavier et ses réponses des actions de clavier et un indicateur LED pour les réponses des actions faites au clavier.
- **Interface Web et API REST :** Développer une application web répartie – sécurisée via le protocole TLS – permettant le pilotage de la serrure à distance, l'authentification, la journalisation, la supervision de l'état du système et des services. Cette application donne aussi accès à quelques challenges de cybersécurité.

2 - INFRASTRUCTURE MODULAIRE ET CONTENEURISATION

- **Isolation par conteneurs** : Séparer les services (site web, base de données, proxy) dans des environnements isolés via **Docker Compose** pour sécuriser le serveur central.
- **Flexibilité et Maintenance** : Faciliter l'ajout de nouveaux modules et permettre la réinitialisation instantanée des exercices de Pentest sans impacter le système principal.

3 - OPTIMISATION ET SÉCURISATION WEB (REVERSE PROXY)

- **Centralisation et Sécurisation** : Utiliser un serveur **Nginx** comme point d'entrée unique sur le port **8443** pour chiffrer tous les échanges (SSL/TLS) et protéger l'accès aux services en passant du protocole **HTTP** en **HTTPS**.
- **Routage Intelligent** : Diriger automatiquement les flux vers le site principal ou les modules de challenge, tout en assurant la fluidité des données en temps réel (**WebSockets**).

4 - COMMUNICATION ET ORCHESTRATION IOT (MQTT)

Assurer la liaison entre les requêtes de l'**API** et le broker **MQTT** présent dans la serrure, permettant la redirection fiable des commandes de déverrouillage (ex : DVIL/Serrure, DVIL/Status) et la remontée instantanée des changements d'état physique de la serrure vers l'interface de supervision.

5 - DYNAMISME DE LA JOURNALISATION (LOGS)

- **Journalisation temps réel :** Intégrer la technologie **Socket.IO** pour permettre une remontée d'événements instantanés sur l'interface de supervision (accès, erreurs, sources).
- **Persistance et analyse :** Améliorer la structure de la base de données MariaDB pour assurer un historique fiable et exploitable des actions critiques de la serrure.

6 - SÉCURITÉ ET ASPECT PÉDAGOGIQUE

- **Laboratoires de Pentest (CTF)** : Créer des environnements d'entraînement avec des failles volontaires (ex: injections SQL) pour apprendre à identifier et corriger des cyberattaques en conditions réelles.
- **Confinement et isolation** : Utiliser la technologie **Docker** pour isoler chaque exercice, garantissant qu'une attaque réussie dans un laboratoire ne puisse pas se propager au serveur principal ou aux données du système.

7 - VALIDATION ET DOCUMENTATION

- **Tests de performance et fiabilité** : Valider la réactivité de la chaîne de commande (de l'interface web jusqu'au matériel) pour s'assurer que l'ouverture est instantanée et que chaque événement est correctement enregistré dans les journaux de sécurité.
- **Dossier d'exploitation technique** : Regrouper l'ensemble des schémas, des configurations réseaux et des procédures de déploiement dans une documentation structurée pour garantir une maintenance simplifiée et l'évolution future du système.

8.1 Rôle au sein de l'équipe et répartition des missions

Le projet DVIL (Damn Vulnerable Internet Lock) est un projet collaboratif dont l'objectif en 2026 est de faire évoluer une base existante vers une architecture professionnelle, sécurisée et évolutive. L'équipe se compose de trois membres, avec une répartition des tâches optimisée pour cette phase d'évolution :

- **IR1 EL MADANI Mohamed (moi) :** Responsable de l'évolution des interfaces de commande, de la mise en place de l'architecture système de l'interface web et de la journalisation des événements.
- **IR2 (EL KADMIRI Haitam) :** En charge de l'évolution de l'interface NFC, du pilotage de la gâche électrique, de la gestion des communications entre le microcontrôleur et la Raspberry Pi et d'une mise en place d'un poste d'accueil afin d'enrôler un badge pour un nouvel utilisateur.
- **ER (AMDOUNI Abdelrahmen) :** Responsable de la partie matérielle : conception électronique, câblage des bus de communication (I2C, SPI, GPIO) et gestion de l'alimentation électrique du système.

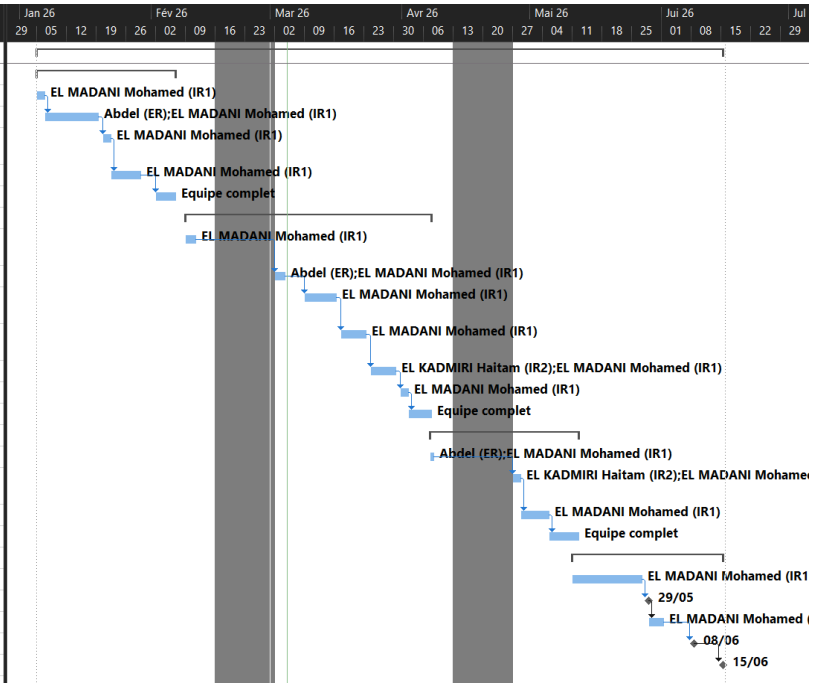
8.2 Tâches à réaliser (cahier des charges)

Étudiant n°1 **IR1**	**Interface Clavier/Interface Web/Journalisation** • S'approprier la communication via bus I2C et protocole MQTT • Définir avec **IR2** la messagerie entre les différentes applications du système • S'approprier le concept d'API Web • Concevoir/Coder/Tester une interface Web sécurisée (API ou site) permettant de communiquer via MQTT avec l'application de commande de la serrure • Concevoir/Coder/Tester une application permettant de s'interfacer via MQTT avec l'application de commande de la serrure depuis le clavier matriciel sur bus I2C (optionnellement chiffré) conçu par l'étudiant ER • Concevoir/Coder/Tester une application qui journalise toutes les demandes de déverrouillage de la serrure • Collaborer avec **IR2** pour imaginer des vulnérabilités et la manière de les exposer (firmwares différents, conteneurs Docker) • Collaborer étroitement avec les étudiants **IR2** et **ER** • Assurer la gestion de version logicielle (Forge logicielle Framagit) • Rédiger avec **IR2** un manuel de démarrage rapide pour l'installation et l'utilisation des différents constituants	**Installation/Configuration** : • Environnements de développement Raspberry Pi ZeroW • Broker MQTT **Mise en œuvre :** • Programmation Web (langage au choix) • Programmation Qt/C++ • TLS • Bus I2C • MQTT • Circuit de crypto-authentification **Réalisation :** • Application de pilotage de serrure via clavier matriciel • Application de pilotage de serrure via interface Web • Journalisation **Documentation :** • Guide de démarrage rapide • Gestion de version logicielle • Dossier de développement

9. Planification Personnelle (diagramme de Gantt)

DIAGRAMME DE GANTT

	Mode Tâche	Nom de la tâche	Durée	Début	Fin
1		**Projet Global DVIL**	**150h**	**Lun 05/01/26**	**Lun 15/06/26**
2		**ANALYSE ET CONCEPTION**	**40h**	**Lun 05/01/26**	**Ven 06/02/26**
3		Audit technique code/câblage 2025	8h	Lun 05/01/26	Mar 06/01/26
4		Analyse électrique & Schémas I/O	6h	Mer 07/01/26	Lun 19/01/26
5		Modélisation SysML (BDD / IBD / Séquence)	16h	Mar 20/01/26	Jeu 22/01/26
6		Architecture Docker & Nginx	10h	Jeu 22/01/26	Jeu 29/01/26
7		Revue 1	30min	Lun 02/02/26	Ven 06/02/26
8		**RÉALISATION SYSTÈME ET SÉCURITÉ**	**70h**	**Lun 09/02/26**	**Mar 07/04/26**
9		Reverse-Proxy Nginx & Certificats HTTPS	8h	Lun 09/02/26	Mer 11/02/26
10		Firmware Pico (Clavier / Serrure)	12h	Lun 02/03/26	Mer 04/03/26
11		Développement Module NFC (Python)	15h	Lun 09/03/26	Lun 16/03/26
12		Déploiement Docker et Base de Données	15h	Mar 17/03/26	Lun 23/03/26
13		Communication MQTT (IR1 / IR2)	10h	Mar 24/03/26	Lun 30/03/26
14		Tests unitaires matériels	10h	Mar 31/03/26	Jeu 02/04/26
15		Revue 2	30min	Jeu 02/04/26	Mar 07/04/26
16		**INTÉGRATION ET CYBER**	**20h**	**Mar 07/04/26**	**Mar 12/05/26**
17		Montage final dans la borne	8h	Mar 07/04/26	Mer 08/04/26
18		Audit Cybersécurité (Pentest & Firewall)	6h	Lun 27/04/26	Mar 28/04/26
19		Tests de robustesse Docker	6h	Mer 29/04/26	Mar 05/05/26
20		Revue 3	30min	Mar 05/05/26	Mar 12/05/26
21		**FINALISATION**	**20h**	**Lun 11/05/26**	**Lun 15/06/26**
22		Dossier Technique IR1	10h	Lun 11/05/26	Mer 27/05/26
23		Remis du projet	0h	Ven 29/05/26	Ven 29/05/26
24		Préparation Soutenance finale	9h	Ven 29/05/26	Lun 01/06/26
25		Soutenance finale	1h	Lun 08/06/26	Lun 08/06/26
26		Livraison	0h	Lun 15/06/26	Lun 15/06/26



10. Analyse Technique

Dans cette section, nous détaillons les choix technologiques opérés pour l'évolution de « l'Interface Clavier » et de «l'interface Web/API», en explicitant les contraintes matérielles et logicielles rencontrées.

10.1 Choix des technologies (interface web et clavier)

Pour le développement de l'interface clavier et la gestion locale du système, la langage Python a été choisi en remplacement du framework **Qt** en **C++** utilisé l'an dernier. Un environnement virtuel (**dvil_clavier**) a été mis en place pour l'application clavier afin de l'isoler des autres parties du projet **[voir annexe 4]**.

Ce choix a permis de faciliter le développement étant donné la richesse des bibliothèques et exemples de l'écosystème Python disponibles pour tous les composants/technologies de notre système : GPIOs, I2C, NFC, MQTT..

Côté interface Web/API, le backend utilise **Flask** pour orchestrer nos services via une **API REST** sécurisée, garantissant une intégration fluide avec le frontend (**HTML/CSS/JavaScript**) et une architecture modulaire prête pour de futures évolutions.

L'ensemble est sécurisé par un **Reverse Proxy Nginx**, garantissant une API REST et une interface web centralisées, évolutives et protégées par chiffrement TLS.

10.2 Contraintes techniques (matériels, logiciels et performances)

Catégorie	Composant / Logiciel	Contraintes / Spécifications
Matériel	Microcontrôleur	Raspberry Pi Zero W : ressources limitées en RAM/CPU → utiliser des protocoles légers (ex. MQTT ; SQLite ; OS Light Raspi...,).
	Lecteur NFC & Clavier	Bus I²C/SPI utilisés, respecter la vitesse I²C (100 à 400 kHz) ou vitesse maximale du SPI pour lecture/écriture.
	Alimentation Serrure	Prévoir une alimentation 5 V ou 12 V, capable de fournir le courant nécessaire à la gâche électrique.
Logiciel	Python / Flask	Utiliser Python 3.9 ou supérieur (version stable recommandée).
	Python	Multi environnements pour intégrer différents niveaux.
	Firmware	Compiler avec un cross-toolchain adapté si le microcontrôleur n'est pas classique (exotique).
Performances	Temps de Réponse Système	Le système doit réagir en moins d'1 seconde.
	Charge CPU & Logs	La charge CPU doit rester modérée sur Raspberry Pi Zero W . Éviter la surcharge par des logs excessifs ou des traitements lourds (ex. déchiffrement complexe).

11. Réalisation Interface Web et journalisation

Dans cette section, nous détaillons l'implémentation des différents modules qui constituent le système. Chaque sous-partie décrit un aspect clé du projet, de la modélisation initiale aux mécanismes de communication et de suivi en production.

11.1 Implémentation de l'interface de supervision (API) [ANNEXE 3]

L'API du projet DVIL ne se contente plus de répondre à des requêtes classiques ; elle agit comme une passerelle d'automatisation sécurisée. Elle centralise les interactions entre l'interface de supervision, le contrôleur matériel (Raspberry Pi Zero 2W), le poste d'accueil (NFC) et les laboratoires de Pentest conteneurisés.

- **Architecture RESTful :** L'API expose des endpoints sécurisés pour consulter, modifier et administrer les données. Ex :

```python
@app.route("/api/admin/users/<int:user_id>/reset_password", methods=["POST"])
@require_auth
@require_admin
def admin_reset_password(user_id):
    """ Action Admin : Modification du mot de passe d'un utilisateur cible. """
    data = request.json
    new_password = data.get("new_password", "").strip()
    if len(new_password) < 8:
        return jsonify({"success": False, "error": "8 caractères minimum requis"}), 400
    try:
        user = User.query.get(user_id)
        if not user: return jsonify({"success": False, "error": "Utilisateur introuvable"}), 404

        user.mot_de_passe = bcrypt.generate_password_hash(new_password).decode('utf-8')
        db.session.commit()

        admin_data = verify_token()
        write_log("API", f"Sécurité : Mot de passe de '{user.nom_utilisateur}' modifié par l'administrateur.", user_id=admin_data.get('id'))
        return jsonify({"success": True}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"success": False, "error": str(e)}), 500
```

- **Sécurité et authentification** : Les routes sensibles sont protégées par des décorateurs utilisés en Python (@require_auth, @require_admin), qui valident le token JWT de session, les rôles **RBAC** (***Role-Based Access Control*, ou contrôle d'accès basé sur les rôles**) et appliquent un mécanisme de limitation de débit par adresse **IP** (***rate limiting***) .

  L'authentification fonctionne par token **JWT**: chaque login crée un token qui va être stocké dans un cookie sécurisé, automatiquement vérifié sur chaque appel protégé.

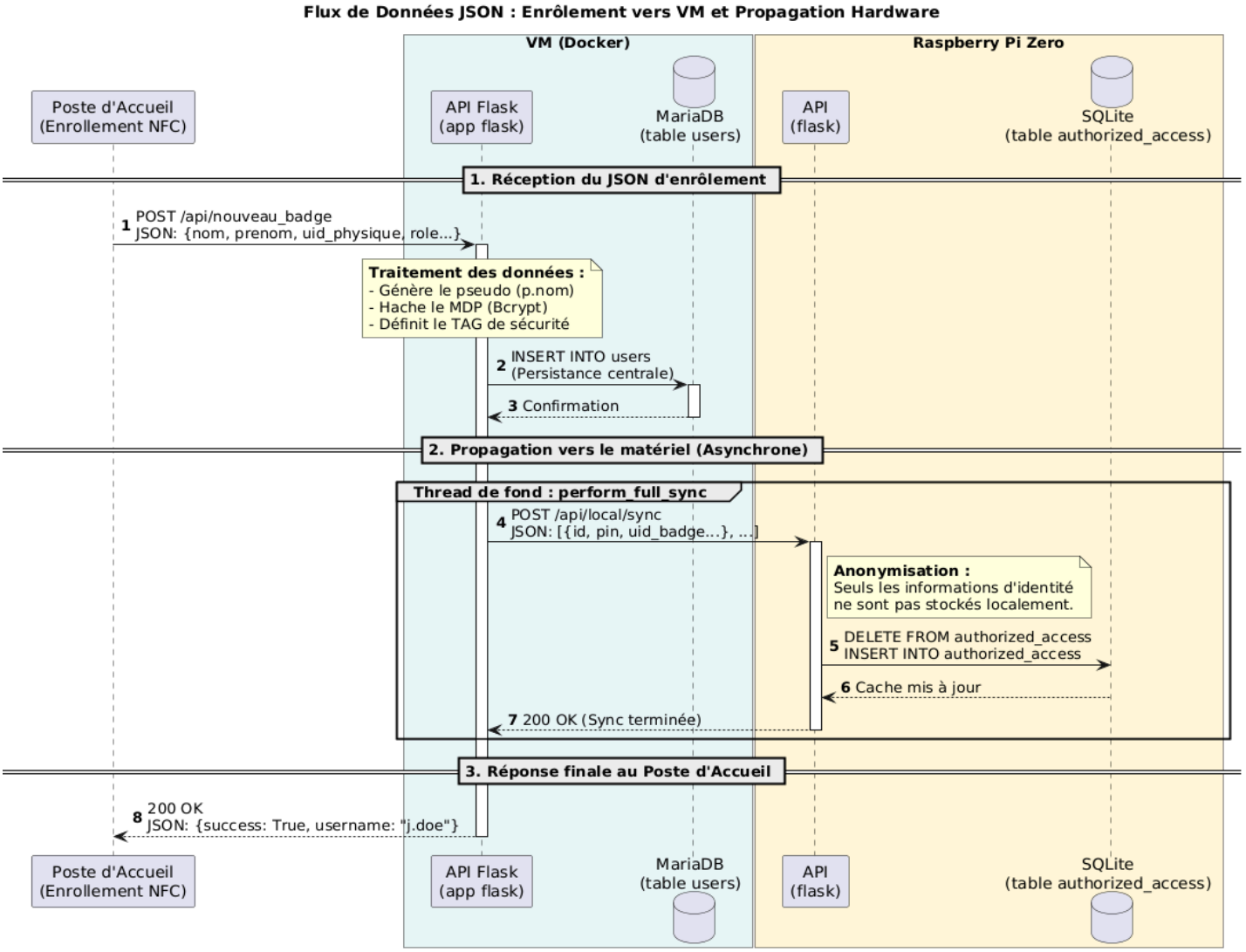
L'architecture et le fonctionnement de l'interface de supervision (interface web) reposent sur une **API REST** permettant d'assurer la gestion de l'authentification, le contrôle du verrouillage et du déverrouillage de la serrure par différents modules **IoT** (clavier, NFC, etc.) ainsi que l'administration, des utilisateurs, via différentes méthodes (POST, GET, etc.), chaque opération étant strictement régie par les droits associés à chaque utilisateur.

11.1.1 Fonctionalités Principales

L'ensemble des fonctionnalités offertes par l'API, ainsi que la description détaillée des endpoints, sont documentés en **[annexe 2]**. L'implémentation technique, accompagnée d'extraits de code et de captures d'écran illustrant le fonctionnement de l'API, sont également présentés en **[annexe 1]**, offrant une vue concrète de l'architecture et des usages du système.

11.1.2 Exemple d'implémentation de la logique côté API : Enrôlement Badge

Le diagramme de séquence ci-dessous illustre la démarche de la récupération des données **JSON** envoyé depuis le **poste d'accueil** d'après un enrôlement de **badge** par **NFC** :



L'API DVIL gère l'ajout d'un nouvel accès au système selon la séquence suivante :

Étape 1 : Demande administrateur :

- L'application cliente (faite par IR2) envoie une requête **POST** vers l'endpoint /api/nouveau_badge contenant les données d'identité, l'UID physique du badge et l'UUID logiciel.
- Le système extrait le nom, le prénom, le rôle et calcule la date d'expiration selon le rôle choisi (ex: 1 an pour un élève, 6 mois pour un stagiaire, 1 mois pour un intervenant et permanent pour un prof et administrateur).

Étape 2 : Vérification de sécurité et Génération dynamique :

- L'API vérifie l'unicité du profil dans la base de données **MariaDB** (doublons Nom&Prénom/UUID).
- **Tolérance au clonage :** Contrairement aux systèmes classiques, l'UID du badge n'est pas filtré, ce qui permet l'utilisation de badges clonés.
- **Automatisme :** En cas de succès,elle attribue une nouvelle ID, génère un code pin clavier et un pseudo (ex: j.dupont, pin: 1234), puis enregistre le nouvel utilisateur avec un mot de passe haché par **Bcrypt** afin d'attribuer un accès à l'interface de commande WEB en tant qu'utilisateur pour déverrouiller à distance ainsi que par interface clavier.

Étape 3 : Synchronisation Hybride :

- L'opération déclenche immédiatement un thread de synchronisation globale (fonction : perform_full_sync) voir le code en **[annexe 2]**.
- Ce thread pousse les nouvelles autorisations vers l'API de la Raspberry Pi Zero 2W, qui met à jour la base de données **SQLite locale** (.dvil_access.db).
- Cette étape garantit que le badge sera reconnu par la serrure, même en cas de coupure du réseau internet (Mode Hors-ligne). Donc les services des modules IoT deviennent indépendants de la Machine Virtuelle (Poste de supervision).

Étape 4 : Sécurisation et Journalisation :

- Le système définit le Tag de sécurité et la date d'expiration selon le rôle.
- Chaque étape est logguée (fonction : write_log voir **[annexe 2]**) et le succès est renvoyé à l'interface du d'accueil (**IR2**) pour confirmer la fin de l'enrôlement.

Format de la réponse API pour un élève crée :

```
{
  "success": true,
  "username": "j.dupont",
  "pin": "5678",
  "tag": "ÉLÈVE",
  "expiration": "2027-05-04"
}
```

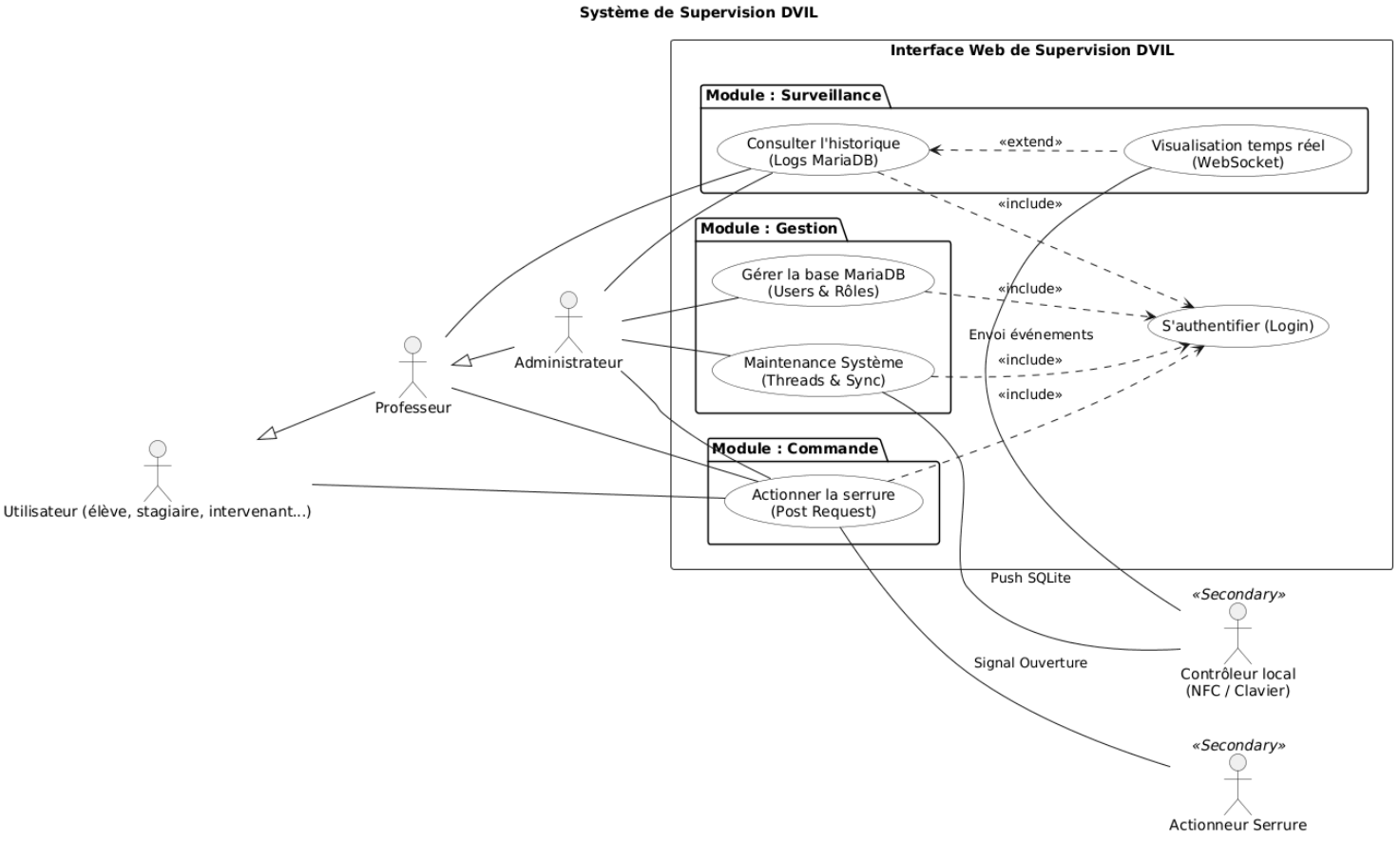
11.2 Modernisation et Migration : L'Architecture de Supervision Conteneurisée

Le projet DVIL 2026 marque une rupture avec l'approche de 2025. L'interface n'est plus une simple page de contrôle locale, mais un véritable **Centre de Supervision** déporté sur une Machine Virtuelle (VM) et géré par la technologie **Docker**.

11.2.1 Alignement Fonctionnel et Rôles Utilisateurs

Pour cette nouvelle version, nous avons redéfini les interactions entre les utilisateurs et le système. L'objectif était de créer une interface qui s'adapte aux responsabilités de chacun (Administrateur, Professeur ou Élève).

Le diagramme de cas d'utilisation ci-dessous, illustre la répartition des modules (Gestion, Commande, Surveillance) et comment chaque acteur interagit avec le système selon leur rôles covenants :



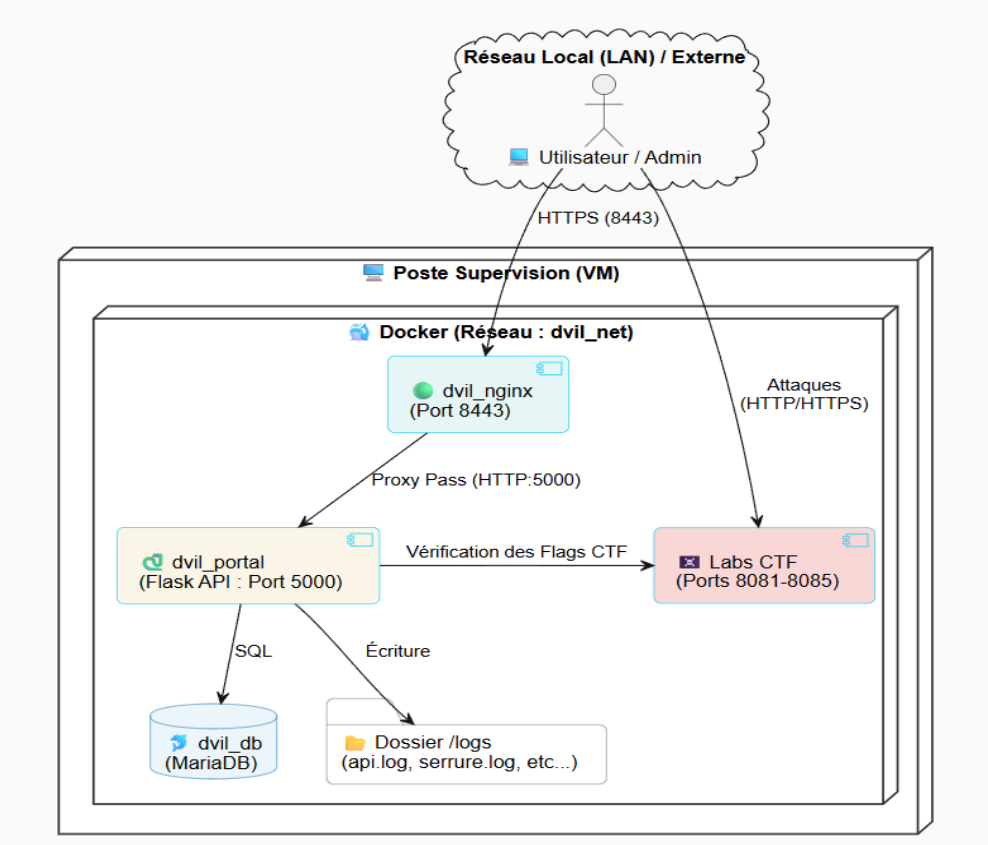
11.2.2 Une Infrastructure Modulaire et Isolée (Docker)

Pour garantir la sécurité et la stabilité, nous avons découpé le serveur en "différentes sections" appelées **conteneurs**. Si un apprenant (attaquant) parvient à s'entraîner dans le laboratoire de test, il reste bloqué dans ce compartiment sans pouvoir accéder à la base de données principale.

Les conteneurs du poste de supervision :

- **dvil_db (Image : MariaDB 10.11)** : Le cerveau central qui stocke les 7 profils utilisateurs et l'historique des accès.
- **dvil_portal (Image : Python 3.9-slim)** : Le cœur de l'application Flask qui gère l'API et la logique de déverrouillage.
- **dvil_nginx (Image : Nginx:alpine)** : Le garde du corps qui gère le chiffrement HTTPS et redirige les flux vers les bons services.
- **dvil_challenges (Conteneurs de Challenges)** : Des mini-serveurs isolés (ports 8081 à 8085) dédiés aux exercices de piratage (Injections SQL, failles IDOR, etc.).

Ce schéma illustre l'architecture de l'interface de supervision rangée par des conteneurs qui se situent à l'intérieur du réseau virtuel sécurisé nommé **dvil_net** :



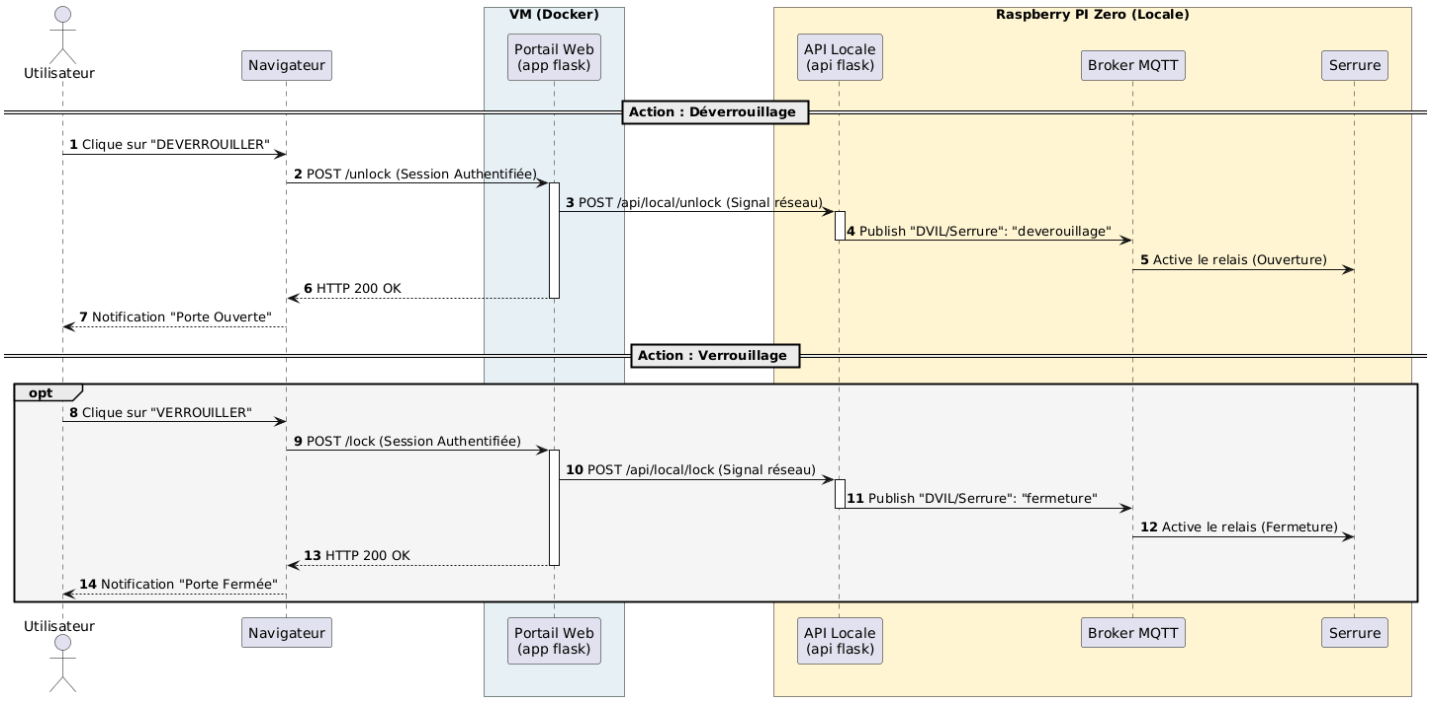
11.3 La Chaîne de Commande IoT (Du Clic au Matériel)

Le défi technique était de faire communiquer une page web située sur un serveur distant (VM) avec un objet physique (Raspberry Pi) de manière instantanée.

Les étapes d'une commande de déverrouillage :

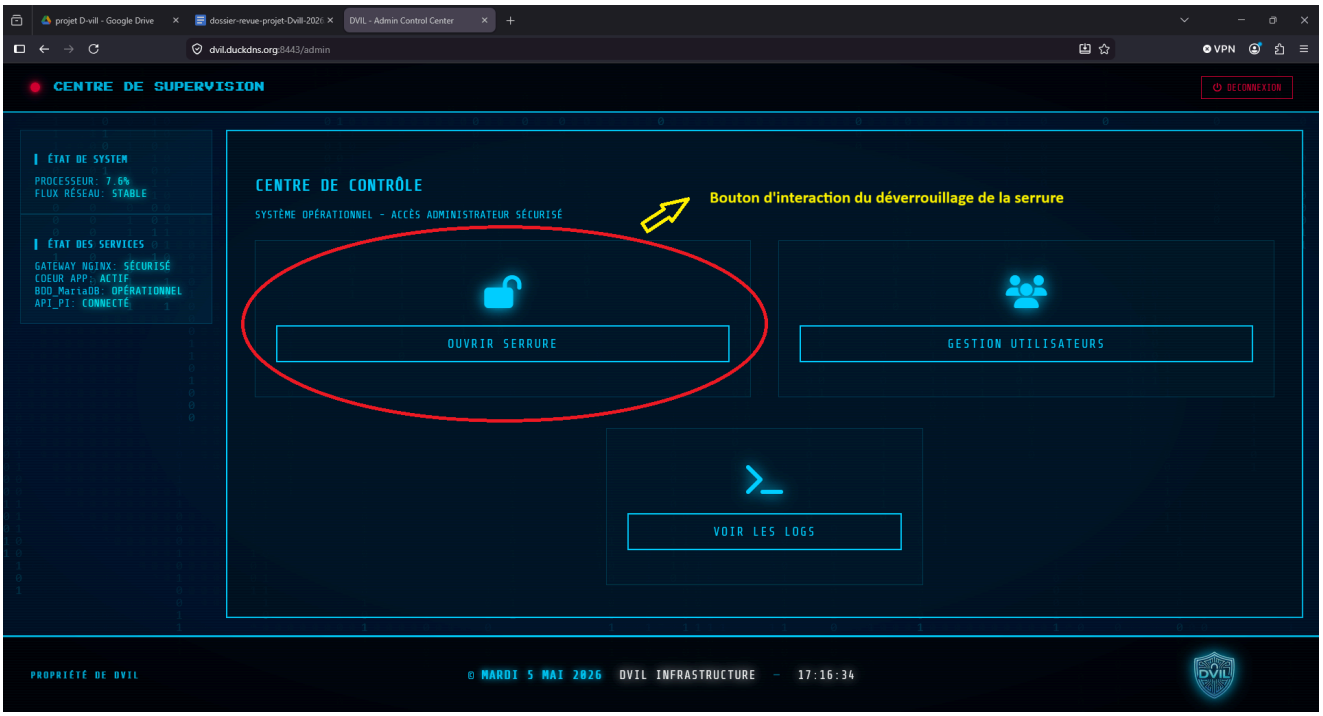
1. **Action Web** : L'administrateur clique sur "DÉVERROUILLER" depuis l'interface web (supervision).
2. **Validation API** : L' API Flask vérifie les droits de l'utilisateur.
3. **Transmission Réseau** : L'API Flask (VM) envoie un signal via le réseau à l'API locale de la Raspberry Pi.
4. **Action Physique** : La Pi publie un message sur le protocole **MQTT**, ce qui active le relais et ouvre la gâche électrique.

Ce diagramme de séquence détaille chaque étape du message, de la souris de l'utilisateur jusqu'au mouvement physique de la serrure :



Point à retenir : La serrure a été créée pour réaliser l'action de déverrouillage mais pour le moment il n'y a pas la méthode de verrouillage en manuelle puisque la serrure au moment du déverrouillage, elle se verrouille automatiquement. Dont le fait de mettre la partie de verrouillage manuelle optionnel d'après le diagramme de séquence ci-dessus.

Exemple de la vue de la vue de la commande de la serrure depuis le poste administrateur :



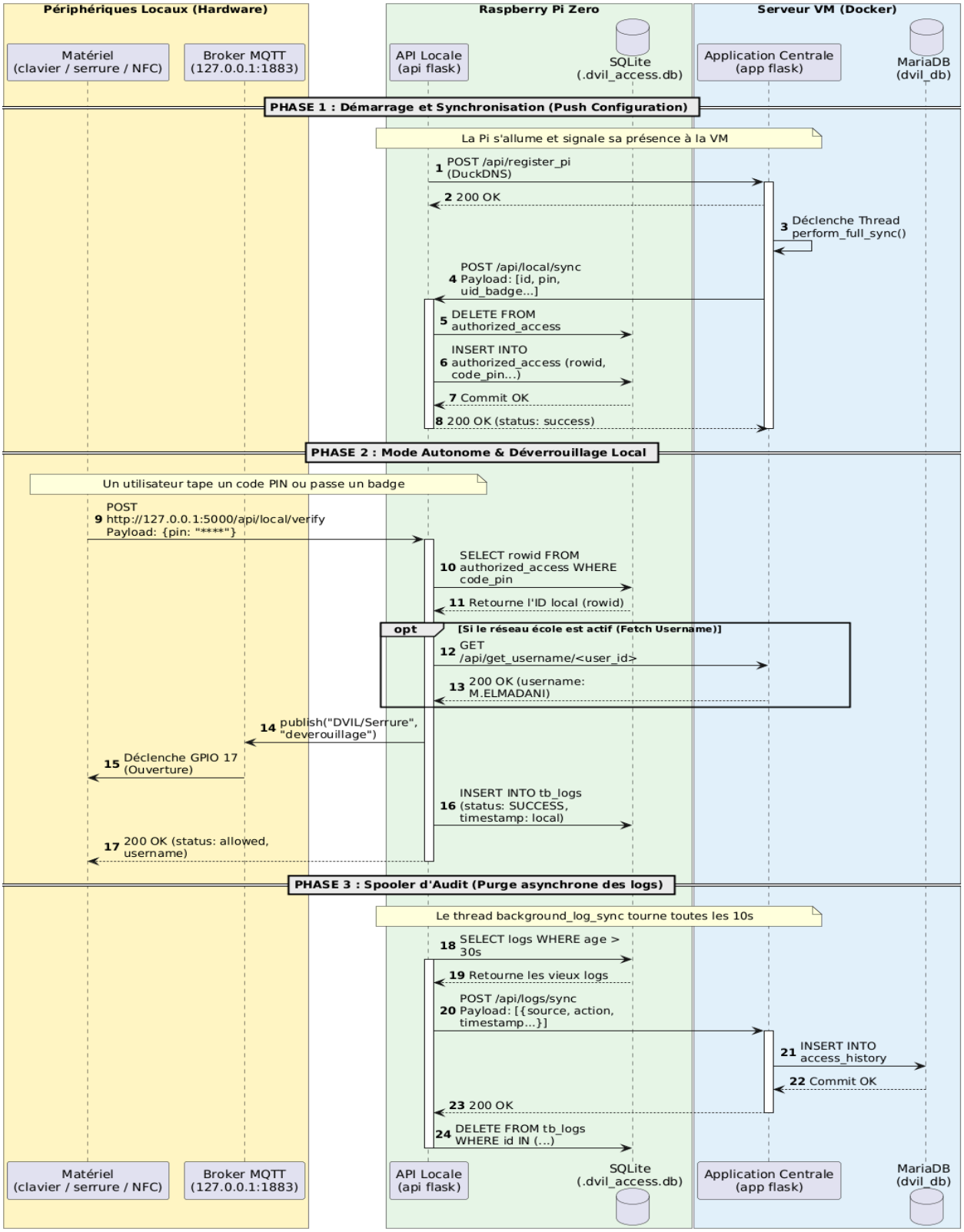
11.4 Synchronisation des Données et Audit

Pour que la serrure fonctionne même si le réseau est coupé, nous avons mis en place une **synchronisation hybride**. La VM possède la base de données "Maître" **(MariaDB)** et la Raspberry Pi possède une copie "Esclave" légère **(SQLite)**.

Fonctionnement en trois étapes :

- **Étape 1 (Push)** : Dès qu'un utilisateur est ajouté sur le serveur web, la l'API Flask (VM) pousse l'information vers l'API de la Raspberry Pi Zero 2W.
- **Étape 2 (Autonomie)** : L'API de la Raspberry Pi Zero 2W reconnaît les badges et codes PIN localement sans demander l'avis du serveur.
- **Phase 3 (Audit)** : Toutes les 2 secondes, un processus en arrière-plan (background_log_sync **[code annexe 5]**) envoie les logs locaux vers l'API de la VM pour archivage et les logs qui étaient dans la base de données **SQLite (Raspberry Pi Zero 2W)** seront effacé au bout de 30 seconds.

Ce diagramme de séquence montre comment les données sont circulé entre la base de données **MariaDB** et **SQLite** pour garantir l'autonomie du matériel :



11.5 Intégration de la logique de journalisation

L'interface de supervision fonctionne comme un tableau de bord. Grâce à Socket.IO (permettant une communication par **Websocket**), les informations (comme l'état des services du système ainsi que l'état du processeur et du réseau actuel) s'actualisent en temps réel, sans avoir besoin de recharger la page. **[Voir annexe 1 page de centre de contrôle]**

- **Surveillance :** Les voyants passent du bleu au rouge clignotant en cas de déconnexion d'un service.
- **Visualisations en temps réel :** On peut voir toutes les actions matériels comme le **clavier**, **NFC** et **serrure** ainsi que les mises à jour, synchronisations des **API**, modifications, tentatives d'authentification (qui s'est connecté, échecs de connexions, etc.

La journalisation occupe une place centrale dans notre système, car elle assure non seulement le suivi des opérations mais facilite aussi le débogage, la supervision en temps réel, et l'analyse des incidents de sécurité.

I. Objectifs et utilité de la journalisation

L'ensemble des modules (VM Supervision, Poste d'Accueil, Contrôleur local) génère des logs pour :

- **Tracer les flux d'accès** : Identifier quel utilisateur a utilisé quel moyen (NFC, Clavier, Web) pour actionner la serrure.
- **Audit de Sécurité** : Détecter les tentatives de "Brute Force" sur le clavier ou les badges inconnus.
- **Maintenance Préventive** : Monitorer l'état des services (Docker) de la VM et de la machine (CPU/NET) et la connectivité du contrôleur local (Raspberry Pi Zero 2W).

II. Mécanismes techniques mis en œuvre

Actuellement, l'application Flask (côté VM) utilise une fonction centralisée **write_log [code annexe 2]** qui assure une double persistance des données pour garantir à la fois un diagnostic technique rapide et un historique administratif fiable :

1. **Persistance SQL (MariaDB)** : Chaque événement critique est enregistré dans la table **access_history [annexe 5]**. Cela permet une conservation longue durée et des statistiques précises.
2. **Fichiers par secteur (Docker Volume)** : Les logs sont séparés par type d'interface (API, serrure, clavier, etc.) pour faciliter le diagnostic technique :
    - **api.log :** Centralise l'activité globale et les flux de l'API de supervision.
    - **serveur.log :** Cycle de vie de l'application Flask et sessions.
    - **serrure.log :** États physiques de la gâche électrique.
    - **clavier.log / nfc.log :** Détails des saisies et scans de badges.

Extrait du moteur de routage des logs (fonction **write_log**) :

```python
def write_log(category, message, user_id=None, status="SUCCESS"):
    """
    Système d'audit sectoriel avec anonymisation des composants techniques.
    Gère le masquage des données sensibles et le routage vers les fichiers logs.
    Double écriture : Fichiers texte (Web) + Table MariaDB access_history.
    """
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    cat_upper = category.strip().upper()
    msg_upper = message.upper()

    msg_clean = message.replace("Pi", "Contrôleur local").replace("PI", "Contrôleur local")

    if "[CLAVIER]" in msg_upper and "ACCORDÉ" in msg_upper:
        msg_clean = re.sub(r'\(.*?\)', '(****)', msg_clean)

    target_file = "api.log"
    color = C_CYAN

    if cat_upper == "SERVEUR":
        target_file = "serveur.log"
        color = C_BLUE
    elif "[CLAVIER]" in msg_upper:
        target_file = "clavier.log"
        if "ANNULÉE" in msg_upper: color = C_PURPLE
        elif "INCOMPLÈTE" in msg_upper: color = C_YELLOW
        else: color = C_GREEN if "ACCORDÉ" in msg_upper else C_RED
    elif "[NFC]" in msg_upper:
        target_file = "nfc.log"
        color = C_GREEN if "RECONNU" in msg_upper else C_RED
    elif "[SERRURE]" in msg_upper or "[ÉTAT]" in msg_upper:
        target_file = "serrure.log"
        color = C_PURPLE
```

III. Sécurisation et Anonymisation des Données

Contrairement à la version 2025, la version actuel applique un protocole d'anonymisation par **Expressions Régulières (Regex)** avant l'écriture pour protéger la vie privée des opérateurs :

Masquage dynamique :

- **Codes PIN (clavier)** : Les chiffres saisis sur le clavier sont remplacés par (****) dans les journaux à succès.
- **Adresses IP** : Les adresses IP distantes sont remplacées par le tag [REMOTE_IP] dans le fichier **api.log**.
- **Protection de la vie privée** : Ce mécanisme garantit qu'aucune donnée sensible ou information de traçage géographique ne soit stockée de manière durable dans les fichiers de logs.

Extrait du code de l'usage de **Regex** :

```
# Masquage du code PIN entre parenthèses
if "[CLAVIER]" in msg_upper and "ACCORDÉ" in msg_upper:
    msg_clean = re.sub(r'\(.*?\)', '(****)', msg_clean)
# Masquage de l'adresse IP dans le journal global
msg_anonyme = re.sub(r'\d{1,3}(\.\d{1,3}){3}', '[REMOTE_IP]',
msg_clean)
```

IV. Niveaux de log et gestion de la granularité

Pour assurer une surveillance précise, nous utilisons une combinaison de catégories (secteurs du système) et de **statut** (niveaux d'importance). Cela permet de filtrer les informations par origine matérielle ou par gravité de l'événement.

1. **Niveaux de sévérité (Paramètre status)**
   - **SUCCESS :** Valeur par défaut. Utilisée pour les actions nominales et validées (ex : déverrouillage autorisé, provisionnement de compte réussi, synchronisation terminée).
   - **DENIED :** Signalise un refus d'accès sécurisé ou une règle métier non respectée sans erreur logicielle (ex : badge inconnu, code PIN erroné, UUID déjà utilisé).
   - **ERROR :** Consigne de manière explicite les exceptions logicielles et les échecs matériels (ex : échecs de base de données, indisponibilité du contrôleur local).
   - **INFO :** Informe l'action et l'état de la serrure (si déverrouillé ou verrouillé).
2. **Secteurs d'activité (Paramètre category)**

   Le système trie les logs dans des fichiers distincts pour éviter la pollution des données :

   - **SERVEUR** : Événements liés au portail web Flask, au cycle de vie de l'application et aux ouvertures/fermetures de sessions.
   - **API** : Flux de données globaux et protocoles de synchronisation entre la Machine Virtuelle et la Raspberry Pi.
   - **SERRURE / CLAVIER / NFC** : Logs spécifiques aux interactions matérielles. Le système colorise automatiquement la sortie (ex: Vert pour "ACCORDÉ", Jaune pour "INCOMPLÈTE") en lisant le contenu du message.

Exemples d'implémentations [annexe 2] :

Ces exemples montrent comment le système journalise la création d'un nouvel utilisateur dans la base de données pour différents secteurs :

Action nominale (le paramètre status prend la valeur SUCCESS par défaut pour le secteur API):

```
write_log("API", f"Provisionnement : Nouvel accès '{role_select}' pour '{username}' créé.")
```

Action de refus :

```
write_log("API", f"Refus d'enrôlement : {prenom} {nom} existe déjà.", status="DENIED")
```

Défaillance technique (le paramètre status est forcé sur ERROR dans un bloc d'exception) si un problème de liaison entre la Machine Virtuelle et la Raspberry Pi Zero 2W rencontre un problème :

```
except Exception as e:
    db.session.rollback()
    write_log("SERVEUR", f"Erreur : {str(e)}", status="ERROR")
```

V. Télémétrie Asynchrone et Journalisation en Temps Réel

l'architecture de supervision abandonne la lecture bloquante de fichiers textes par des threads lourds. Le système met en œuvre une architecture hybride combinant Socket.IO pour les données critiques et l'AJAX (via l'API fetch) pour le traitement de l'historique textuel.

Cette séparation des canaux réseau permet d'optimiser l'utilisation de la bande passante et la réactivité du dashboard d'administration.

1. Le canal critique : Télémétrie instantanée (Socket.IO)

Le tunnel **WebSocket (Socket.IO)** est maintenu ouvert en permanence entre le navigateur et l'application Flask. Il est réservé exclusivement aux indicateurs de santé qui exigent une réaction instantanée à la milliseconde.

Dès qu'un changement d'état est détecté (ex: ouverture physique de la serrure ou pic d'utilisation du CPU), le serveur pousse l'information vers l'interface sans attendre d'être sollicité.

Exemple d'implémentation (Télémétrie matérielle) : Dans cet extrait **(route /unlock)**, le backend informe immédiatement les clients connectés que la gâche a été déverrouillée avec succès :

```python
@app.route("/unlock", methods=["POST"])
@require_auth
def unlock_door():
    """ Route de contrôle : Déverrouillage à distance """
    global lock_status
    user_data = verify_token()
    try:
        r = requests.post(f"http://{PI_IP}:5000/api/local/unlock",
                          json={"username": user_data.get('username')}, timeout=6)
        if r.status_code == 200:
            lock_status = "unlocked"
            state = SerrureState.query.get(1)
            if state:
                state.state = "unlocked"
                db.session.commit()
            socketio.emit('sys_update', {"status": "unlocked"})
            write_log("SERRURE", f"[SERRURE] État : Ouverte (Méthode : Interface Web par {user_data.get('role')} 
            '{user_data.get('username')}')", user_id=user_data.get('id'))
            return jsonify({"status": "success"}), 200

        return jsonify({"status": "error", "error": "Le contrôleur Pi a refusé l'ordre."}), 503
    except:
        return jsonify({"status": "error", "error": "Le Raspberry Pi est injoignable sur le réseau."}), 503
```

2. Le canal de stockage : "Smart Polling" des archives (AJAX)

Pour l'historique des logs, qui peut contenir des milliers de lignes, l'utilisation d'un tunnel permanent saturerait la mémoire du navigateur. Le système utilise donc le **Smart Polling** via l'API native JavaScript **fetch() (Architecture AJAX)**.

Le script client interroge le serveur en arrière-plan toutes les 1000 ms pour récupérer uniquement les nouveautés du secteur sélectionné par l'administrateur. Les données sont transmises au format **JSON**, permettant une mise à jour fluide du **DOM (Document Object Model)** sans aucun rechargement.

Extrait de l'implémentation **Front-end** de la page des logs en JavaScript :

```javascript
function loadLogs(type) {
    fetch("/admin/log_filter", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        credentials: "include", // Sécurité : envoi du cookie JWT
        body: JSON.stringify({ log_type: type })
    })
    .then(res => res.json())
    .then(data => {
        if (data.lines) {
            // Filtrage des répétitions (xN) et rendu dynamique dans le conteneur
            const filtered = filterRepeatedLines(data.lines);
            container.innerHTML = filtered.map((l) =>
                `<div class="${getLogClass(l, type)}">${l}</div>`
            ).join('');
        }
    });
}
```

3. Exploitation des logs concrètes :

Voici quelques exemples d'affichage des lignes générées dans les fichiers logs :

```
[2026-04-22 20:50:11] [SERVEUR] Ouverture de session : admin 'PentesterIR1'
[2026-04-22 21:38:24] [SERRURE] État: DÉVERROUILLÉ - Source: Actionneur Réseau (MQTT)
[2026-04-23 00:21:22] [INTERFACE] [CLAVIER] accordé par PENTESTERIR1 (Code: ****)
[2026-04-24 16:36:50] [API] Échec de synchronisation : contrôleur local injoignable sur [REMOTE_IP]
[2026-04-25 15:37:16] [INTERFACE] [CLAVIER] Accès refusé : (0101)
[2026-05-04 11:07:30] [INTERFACE] [NFC] reconnu par ALICE
```

Cela permet à tout moment de retracer les opérations, d'analyser les incidents et de préciser les actions réalisées.

11.6 Sécurisation et Gestion des Flux

L'interface web s'appuie sur une architecture en « oignon (couches) » où le serveur Flask est protégé par un **reverse proxy Nginx**, chargé de gérer efficacement la sécurité réseau et la cryptographie.

Pour sécuriser l'interface web avec **Nginx**, il faut réaliser certaines installations et configurations afin d'avoir une gestion de flux sécurisé.

Afin que **Nginx** puisse protéger les données en transit entre le client (Navigateur) et l'application **Flask,** il faut mettre en œuvre le passage du protocole **HTTP** en **HTTPS.**

11.6.1 Passage du protocole HTTP/S et création de certificat

Pour disposer d'un site web en protocole **HTTPS (HyperText Transfer Protocol Secure)**, il faut générer un certificat **SSL/TLS.**

Il existe 2 types de certificats :

1. **Certificats auto-signé :** créé et signé directement par son propriétaire, sans passer par une autorité de certification reconnue.

   Il permet de chiffrer une connexion **HTTPS** et de tester une configuration sécurisée, mais il n'est pas validé par une autorité externe, ce qui entraîne généralement des alertes de sécurité dans les navigateurs. Donc les navigateurs ne font pas confiance à ce type de certificats.

2. **Certificat délivré par une autorité de certification (CA) :** Ce type de certificat est vérifié par un tiers de confiance, ce qui garantit l'identité du site ou du serveur permettant d'activer le protocole **HTTPS** sans avertissement dans les navigateurs et est utilisé en production pour sécuriser les services accessibles publiquement.

Pour le projet **DVIL**, un certificat auto-signé n'est pas suffisant pour l'attente de l'exigence d'avoir un interface web sécurisé. Cependant, un vrai certificat nécessite un paiement à une organisation pour nous attribuer un passage **HTTPS**.

Il existe un moyen pour obtenir un certificat validé et reconnu par l'État.

11.6.2 Mise en œuvre de Let's Encrypt

Le chiffrement repose sur des certificats SSL/TLS délivrés par **Let's Encrypt**, une autorité de certification gratuite et automatisée.

- **Génération :** Les certificats ont été générés à l'aide de l'outil **Certbot**. Le défi HTTP-01 a été utilisé : **Let's Encrypt** vérifie que l'on contrôle bien le domaine en tentant d'accéder à un fichier spécifique sur le serveur.
- **Fichiers produits :** Une fois validée, l'autorité délivre **fullchain.pem** (le certificat et sa chaîne) et **privkey.pem** (la clé privée).

Ces informations sont gardé dans le répertoire suivant:

```
└─$ sudo ls /etc/letsencrypt/live/dvil.duckdns.org/
cert.pem  chain.pem  fullchain.pem  privkey.pem  README
```

- **Intégration Docker :** Pour garantir la sécurité, ces fichiers sont stockés sur la VM et montés dans le conteneur Nginx en mode lecture seule (:ro) via le fichier docker-compose.yml **[annexe 7].**

Extrait de la configuration **Docker** (docker-compose.yml) :

```yaml
nginx:
  image: nginx:alpine
  container_name: dvil_nginx
  restart: always
  ports:
    - "8443:8443"
  volumes:
    - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    - /etc/letsencrypt:/etc/letsencrypt:ro
  depends_on:
    - portal
  networks:
    - dvil_net
```

11.6.3 Configuration de la Passerelle Nginx [annexe 6]

Le fichier nginx/default.conf est le cœur de la sécurité réseau. Il gère trois aspects critiques :

A. Forçage du HTTPS et Redirection Standard

Le serveur écoute sur le port 80 (HTTP) et renvoie systématiquement l'utilisateur vers la version sécurisée **(Port 8443)** avec un code de redirection permanente 301.

Exemple de la redirection vers HTTPS :

```
# --- 1. REDIRECTION GLOBALE : HTTP (Port 80) vers HTTPS ---
server {
    listen 80 default_server;
    server_name _;

    return 301 https://dvil.duckdns.org:8443$request_uri;
}
```

B. Durcissement SSL et Correction du Port (Erreur 497)

Nginx est configuré avec des protocoles modernes pour interdire les anciennes méthodes de chiffrement vulnérables. De plus, la directive **error_page 497** est utilisée pour corriger une erreur courante : si un utilisateur tape http:// sur un port attendu en https://, Nginx le redirige automatiquement au lieu d'afficher une erreur.

Exemple de la configuration :

```
# --- 3. CONFIGURATION PRINCIPALE (L'application Flask) ---
server {
    listen 8443 ssl;
    server_name dvil.duckdns.org;

    ssl_certificate /etc/letsencrypt/live/dvil.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dvil.duckdns.org/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    # Si quelqu'un tape http://...:8443, ça va forcer en https sans que le navigateur affiche erreur 400
    error_page 497 =301 https://dvil.duckdns.org:8443$request_uri;
```

La configuration impose l'usage exclusif de protocoles modernes via la directive **(ssl_protocols TLSv1.2 TLSv1.3;)** :

- TLS 1.2 corrige les faiblesses des versions précédentes et reste aujourd'hui la norme la plus utilisée, offrant un bon niveau de sécurité et une large compatibilité avec les navigateurs.
- TLS 1.3 est la version la plus récente et la plus sécurisée. Elle améliore les performances en réduisant le temps de connexion et en supprimant les algorithmes obsolètes.

Le choix des suites de chiffrement **(ssl_ciphers HIGH:!aNULL:!MD5;)** permet d'utiliser uniquement des algorithmes sécurisés et d'exclure les méthodes faibles comme MD5, qui n'est plus fiable.

C. Masquage de l'Identité du Serveur (Anti-Scan)

Pour protéger l'infrastructure contre les outils de reconnaissance automatique (bots, scanners de vulnérabilités), Nginx intègre une politique de "rejet par défaut".

Les requêtes envoyées directement à l'adresse IP sont bloquées par Nginx grâce à une configuration par défaut. Seul le domaine dvil.duckdns.org **[voir section 11.6.4]** est accepté, les autres accès sont refusés (403), ce qui limite l'exposition aux scans.

Extrait de la protection d'identité :

```
# --- 2. SÉCURITÉ : Blocage de l'accès direct par adresse IP (HTTPS) ---
server {
    listen 8443 ssl default_server;
    server_name _;

    ssl_certificate /etc/letsencrypt/live/dvil.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dvil.duckdns.org/privkey.pem;

    # Intercepte les requêtes HTTP sur le port HTTPS et les redirige
    error_page 497 =301 https://dvil.duckdns.org:8443$request_uri;

    return 403 "ACCÈS INTERDIT : Veuillez utiliser le domaine sécurisé https://dvil.duckdns.org:8443";
}
```

D. Intégration Flask et Continuité d'Audit (ProxyFix)

Dans une architecture **Docker**, **Nginx** sert de passerelle : **Flask** reçoit les requêtes depuis l'IP interne de **Nginx**, et non directement depuis l'IP publique des utilisateurs.

Pour garder une traçabilité correcte, **Nginx** envoie la vraie IP du client via **X-Forwarded-For**. Flask utilise ensuite **ProxyFix** pour récupérer cette IP et identifier correctement l'utilisateur.

Exemple de l'implémentation :

```
from flask import Flask
from werkzeug.middleware.proxy_fix import ProxyFix


app = Flask(__name__)

# WSGI Middleware : Permet à Flask de récupérer l'IP réelle de l'utilisateur final.
# Sans cela, Flask verrait uniquement l'IP interne du conteneur Nginx.
app.wsgi_app = ProxyFix(app.wsgi_app, x_for=1, x_proto=1, x_host=1, x_port=1, x_prefix=1)
```

11.6.4 Résolution de Nom et Identité Numérique

Pour rendre l'interface web accessible en **HTTPS**, il faut un nom de domaine complet **FQDN (Fully Qualified Domain Name)**. Or, les connexions Internet utilisent souvent une adresse IP publique dynamique qui peut changer, rendant l'accès direct instable.

Le service **DuckDNS** a été utilisé comme solution de **DNS dynamique**. Il permet de maintenir le lien entre le nom de domaine dvil.duckdns.org et l'adresse IP du serveur, même lorsque celle-ci change. **[consulter l'annexe 6]**

Lien vers ce service : https://www.duckdns.org/

Sans ce service, l'accès distant serait interrompu dès que le fournisseur d'accès Internet **FAI (Fournisseur d'Accès à Internet)** changerait l'adresse IP publique du serveur.

I. L'interdépendance avec Let's Encrypt

L'utilisation d'un nom de domaine est indispensable pour obtenir un certificat SSL/TLS auprès de Let's Encrypt. Cette exigence repose sur trois raisons principales :

- **Validation de propriété (ACME) : Let's Encrypt** utilise le protocole **ACME (Automatic Certificate Management Environment)** pour vérifier que vous contrôlez bien le serveur. Le défi **HTTP-01** nécessite un nom de domaine pointant vers la machine afin de valider la demande de certificat.
- **Limitation des certificats sur IP :** Let's Encrypt ne délivre pas de certificats pour les adresses IP publiques car elles ne sont pas considérées comme des identifiants stables et fiables.
- **Mécanismes de confiance du navigateur :** Les navigateurs web modernes n'accordent le statut "Sécurisé" (cadenas vert) que si le certificat présenté correspond strictement au nom de domaine résolu.

II. Automatisation du maintien de l'identité

Pour assurer la disponibilité du service, un script met automatiquement à jour l'adresse IP publique auprès de **DuckDNS**.

Exemple de requête de mise à jour automatisée :

```
echo url="https://www.duckdns.org/update?domains=dvil&token=VOTRE_TOKEN_PRIVE&ip=" |
curl -k -o /var/log/duck.log -K -
```

III. Intégration technique dans le Reverse Proxy

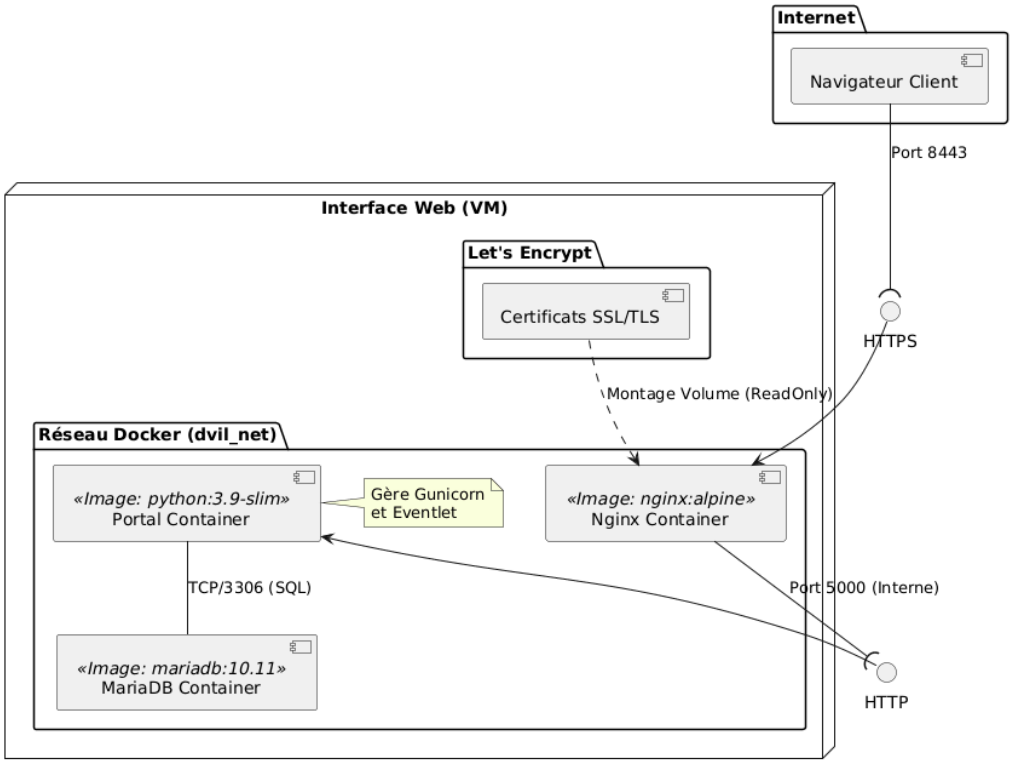
Une fois le nom de domaine stabilisé, Nginx utilise les certificats pour chiffrer les échanges, en les associant directement au domaine fourni par le service **DuckDNS**.

Extrait de la configuration :

```
# --- 3. CONFIGURATION PRINCIPALE (L'application Flask) ---
server {
    listen 8443 ssl;
    server_name dvil.duckdns.org;

    # Configuration SSL
    ssl_certificate /etc/letsencrypt/live/dvil.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dvil.duckdns.org/privkey.pem;
```

Le diagramme de composants ci-dessous illustre le fonctionnement de la sécurisation et gestion web grâce à au serveur **Nginx :**



11.7 Configuration de l'Environnement et Sécurité

L'un des piliers de la sécurité du projet **DVIL** est la séparation stricte entre la logique de l'application (le code) et les données de configuration sensibles. Pour cela, un fichier **.env** est utilisé comme coffre-fort de variables d'environnement.

La raison pour laquelle ce fichier est mis en place est :

1. **Sécurité :** Il évite de "coder en dur" **(*hardcoding*)** des données cachés comme les identifiants pour accéder à la base de données **(MariaDB)**. Cela protège ces informations si le code (application, api, templates, etc.) est partagé.
2. **Flexibilité :** Il permet de modifier facilement les paramètres de configuration, tels que l'adresse IP de la **Raspberry Pi Zero 2W** (pour communiquer avec son API) ou les identifiants d'accès à la base de données **(MariaDB)**, sans avoir à modifier le code source. Cela facilite l'adaptation de l'application à différents environnements (développement, test, production).

Exemple du fichier **.env :**

```
FLASK_ENV=production
SECRET_KEY=cle_secrete_inventee_pour_dvil_2026
DB_HOST=db
DB_USER=root
DB_PASSWORD=Admin
DB_NAME=dvil_db
PI_IP=192.168.4.47
```

Implémentation dans le code de l'application **Flask** :

```
from dotenv import load_dotenv
load_dotenv()

# Chargement des données sensibles depuis l'environnement
app.config["SECRET_KEY"] = os.getenv("SECRET_KEY")
PI_IP = os.getenv("PI_IP")
DB_PASSWORD = quote_plus(os.getenv("DB_PASSWORD"))
```

11.8 Gestion des Sessions & Sécurité

Le système du projet utilise une authentification basée sur les **JWT (JSON Web Tokens)**. Contrairement aux sessions classiques, le **JWT** est sans état **(stateless)** : toutes les informations qui sont liées à l'utilisateur sont intégrées directement dans un jeton, signé de manière **cryptographique** afin d'en garantir l'intégrité et de l'authentification.

I. Persistance des sessions et redémarrage avec Docker

Un des points importants est la survie de la session lors d'une maintenance des conteneurs.

- **Comportement :** Si vous redémarrez le conteneur **dvil_portal** (conteneur de l'application Flask), un utilisateur ayant une session ouverte peut toujours accéder à un chemin relatif d'une page web (ex : /accueil) sans se reconnecter.
- **Explication technique :** Tant que la SECRET_KEY dans le fichier .env reste identique et que le jeton n'a pas expiré (limite de 2 heures dans votre code), **Flask** peut valider la signature du cookie même après un redémarrage complet du service.

II. Journal d'incident : Audit des sessions et accès

Ci-dessous, les problèmes testés et les solutions de sécurité mises en place :

Nature	Observation technique	État / Correctif appliqué
Accès "Fantôme" après redémarrage	Un utilisateur non déconnecté accède à une page (ex : /accueil) après un redémarrage du conteneur.	La session est portée par le client. Sécurisé par une expiration courte (2h) dans app.py.
Tentative de Bypass d'URL	Accès direct aux routes /admin ou /accueil sans passer par le login.	Implémentation du décorateur @require_auth qui vérifie le jeton avant chaque rendu.
Persistance après une déconnexion de session	Le cookie restait actif après la fermeture de session, permettant un retour arrière.	Forçage de l'expiration du cookie à 0 dans la route /logout.

Extrait de la fonction du décorateur de vérification :

```python
def require_auth(f):
    """ Décorateur Python : Restreint l'accès aux utilisateurs authentifiés """
    @wraps(f) # Permet de conserver les métadonnées de la fonction décorée (nom, docstring)
    def decorated(*args, **kwargs):
        if not verify_token():
            if request.path.startswith('/api/'):
                return jsonify({"error": "Unauthorized", "redirect": "/login"}), 401
            return redirect(url_for("login_page"))
        return f(*args, **kwargs)
    return decorated
```

11.9 Maintenance & Sauvegarde

La gestion avec Docker permet de conserver les données importantes même après l'arrêt du système.

Le fichier de configuration docker-compose.yml définit des ponts entre la **VM (Machine Virtuelle)** et les conteneurs pour sauvegarder la base de données et les journaux d'audit.

Extrait de la configuration **Docker :**

```yaml
services:
  db:
    image: mariadb:10.11
    volumes:
      - dvil_db_data:/var/lib/mysql  # Persistance des utilisateurs et logs
  portal:
    build: .
    volumes:
      - ./logs:/app/logs             # Sauvegarde des fichiers .log sur l'hôte
```

11.9.1 Gestion des erreurs et retour utilisateur

L'interface doit informer l'administrateur de l'état de la communication entre la **VM (machine virtuelle)** et la **Raspberry Pi Zero 2W**.

Si la **Raspberry Pi Zero 2W** tombe en panne, l'administrateur doit être immédiatement averti. Un système de surveillance vérifie en continu que le contrôleur est toujours opérationnel grâce à un suivi en temps réel **(WebSocket)**.

Ce bloc gère le "checking" de l'état du matériel tels que les services (applications IoT, API, etc.), la connectivité réseaux, communication, etc. :

```python
def check_infrastructure():
    """
    Service de télémétrie et monitoring de l'état nominal des modules.
    Diffuse les métriques en temps réel pour synchronisation chromatique (Cyan).
    """
    while True:
        with app.app_context():
            try:
                # Analyse des ressources hardware via psutil
                cpu_usage = psutil.cpu_percent(interval=None)

                # Payload de santé : Les statuts correspondent aux cibles du frontend (Cyan)
                health_report = {
                    "cpu": f"{cpu_usage}%",
                    "net": "STABLE",
                    "nginx": "SÉCURISÉ",
                    "portal": "ACTIF",
                    "db": "OPÉRATIONNEL",
                    "pi": "CONNECTÉ"
                }

                # Diagnostic de la couche de persistance MariaDB
                try:
                    db.session.execute(db.text('SELECT 1'))
                except:
                    health_report["db"] = "OFFLINE"

                # Diagnostic de la connectivité avec l'API locale du contrôleur Pi
                try:
                    r = requests.get(f"http://{PI_IP}:5000/status", timeout=1.5)
                    if r.status_code != 200:
                        health_report["pi"] = "ERREUR"
                except:
                    health_report["pi"] = "OFFLINE"
```

Aussi un extrait du code qui prévient l'administrateur s'il essaie de déverrouiller la serrure alors que la Raspberry Pi Zero 2W est déconnecté du réseau ou éteinte :

```python
# --- Extrait de la section 8 : unlock_door ---
@app.route("/unlock", methods=["POST"])
@require_auth
def unlock_door():
    try:
        r = requests.post(f"http://{PI_IP}:5000/api/local/unlock",
                          json={"username": user_data.get('username')}, timeout=6)

        # ... (Logique de succès)

    except:
        # FEEDBACK UX : Message d'erreur personnalisé envoyé au navigateur
        return jsonify({"status": "error", "error": "Le Raspberry Pi est injoignable sur le réseau."}), 503
```

11.9.2 Gestion des erreurs et expérience utilisateur

L'application assure une gestion fluide des erreurs afin d'informer clairement l'utilisateur et de maintenir une navigation sécurisée et cohérente.

- **Redirection automatique :** Si le jeton expire pendant la navigation, l'utilisateur est instantanément envoyé vers la page /login grâce au décorateur de sécurité @require_auth.

- **Alertes visuelles :** Les erreurs de connexion, comme un mauvais mot de passe, sont affichées directement à l'utilisateur sous forme de messages, sans recharger la page. Cela est rendu possible grâce à des requêtes en arrière-plan **(AJAX/Fetch)**, ce qui rend l'expérience plus fluide et rapide.

11.10 Gestion des Bases de Données [Annexe 5]

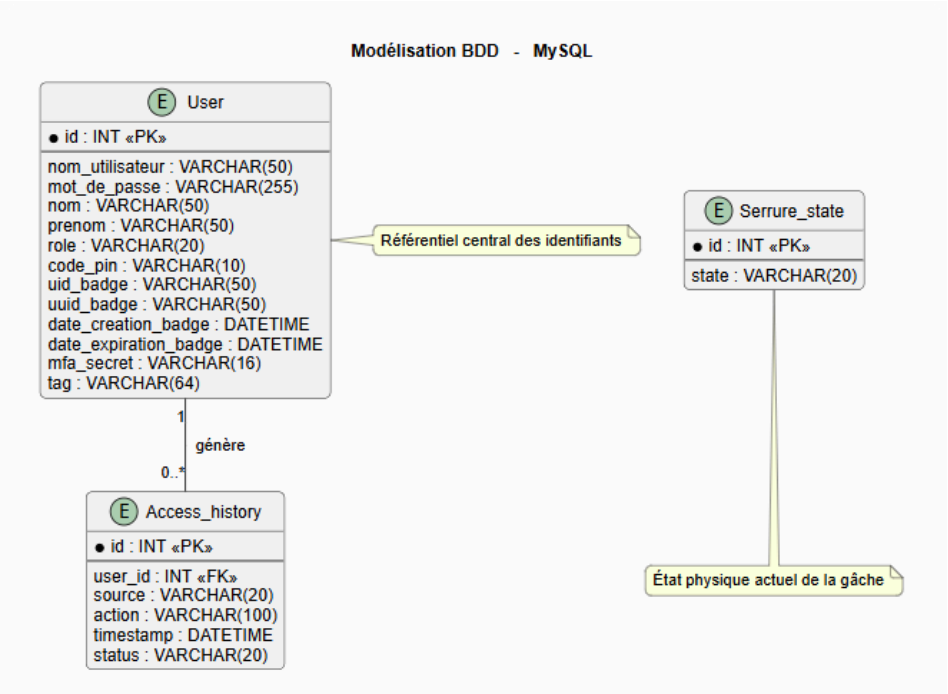
L'architecture de stockage du système repose sur une stratégie hybride visant à centraliser la gestion des utilisateurs sur le serveur **(VM)**, tout en garantissant l'autonomie du système de la **Raspberry Pi Zero 2W** en cas de coupure réseau.

11.10.1 Serveur de supervision

La base de données principale est gérée par **MySQL/MariaDB**. Elle centralise l'ensemble des données critiques : identités, mots de passe hachés et historique global.

- **Initialisation :** La structure est créée à partir du script dvil_db.sql **[consulter annexe 5]** . Ce fichier définit les tables users, access_history et serrure_state.
- **Gestion logicielle :** Côté serveur, l'application Flask utilise **SQLAlchemy** pour manipuler les données sans écrire de requêtes SQL brutes dans le code principal.

Ce modèle de base de données ci-dessous, montre la structure de la base de données **MariaDB (côté poste de supervision - VM)** :

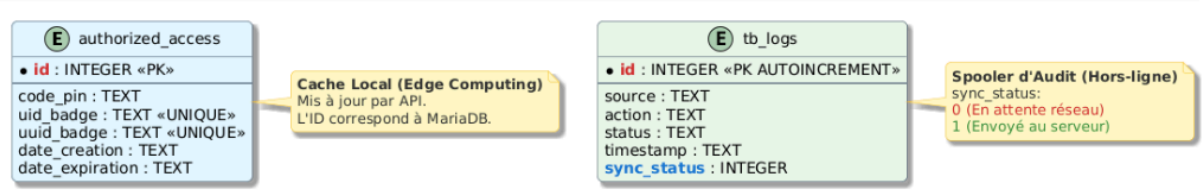


11.10.2 Nœud de contrôle local (Raspberry Pi Zero) [Annexe 5]

Sur la Raspberry Pi Zero, nous utilisons **SQLite (Structured Query Language Lite)**. Cette base de données est contenue dans un fichier caché (.dvil_access.db) pour assurer la discrétion et la sécurité locale.

- **Initialisation :** Contrairement au serveur, la base locale est initialisée par un script dédié init_db.py. Ce script crée des tables simplifiées et anonymisées pour respecter la confidentialité.
- **Autonomie :** Elle permet de vérifier l'accès d'un badge NFC ou d'un code PIN même sans connexion internet (pas de communication avec la **VM (Virtual Machine)** .

Ci-dessous le modèle de la base de données locale SQLite :



Extrait de code d'initialisation pour la table des données utilisateurs :

```
import sqlite3

# Chemin exact demandé : dossier /var/lib/SQLite/ et fichier caché .dvil_access.db
DB_PATH = "/var/lib/SQLite/.dvil_access.db"

def init_db():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()

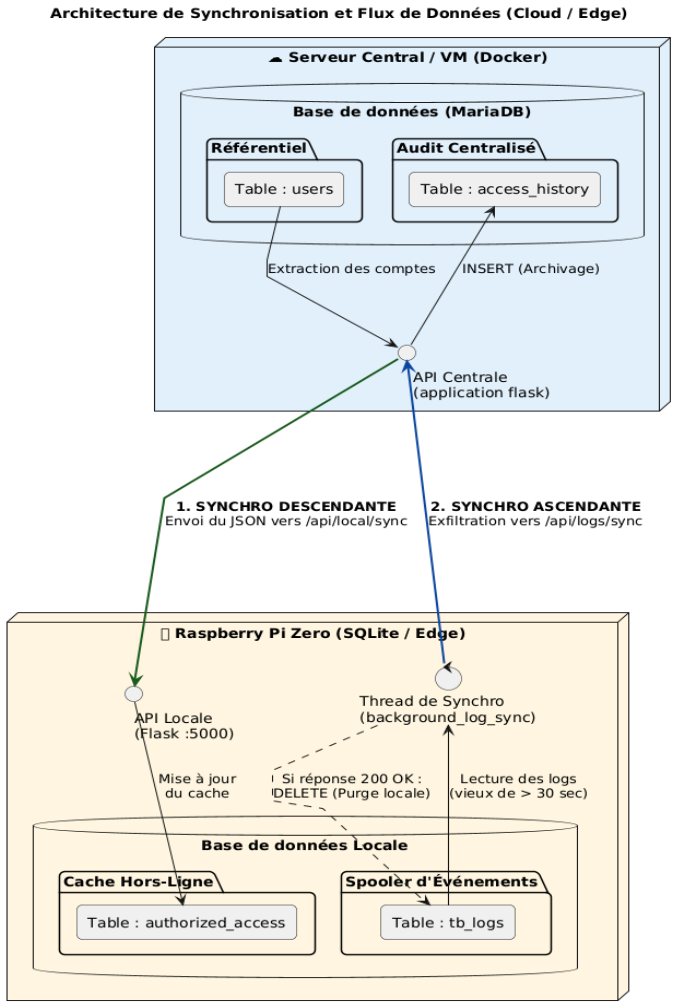
    # 1. Table AUTONOME (Anonyme : UUID + PIN + Dates)
    # Respect de la logique : pas de noms, pas de rôles sur la Pi
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS authorized_access (
            nfc_uuid TEXT PRIMARY KEY,
            pin_code TEXT,
            date_creation TEXT,
            date_expiration TEXT
        )
    ''')
```

11.10.3 Architecture de synchronisation (API REST)

La communication entre l'**API Flask (côté VM)** et l'**API Flask (côté locale Raspberry Pi Zero 2W)** est le point clé de la sécurité du système. Elle permet de maintenir les deux bases de données à jour.

- **Le Flux Descendant (VM → Pi) :** L'**API (côté Pi)** "tire" **(Pull)** régulièrement les données depuis la base de données **MariaDB** pour mettre à jour la base de données **SQLite** local. Cela inclut les nouveaux badges ajoutés ou les codes PIN modifiés par l'administrateur.
- **Le Flux Montant (Pi → VM) :** C'est la remontée des événements. Chaque fois qu'une personne utilise le clavier ou le badge, un log est créé localement. L'**API Flask (côté Pi)** envoie ensuite ces logs vers la base de données (table : acces_history) pour qu'ils soient enregistrés dans l'historique global.

Ce schéma illustre l'architecture et le flux de données entre les **API REST (VM -> Pi)** pour une synchronisation d'une base de données centrale et locale :



Extrait du code de la synchronisation des logs avec l'API REST (côté Pi) :

```python
def background_log_sync():
    """
    Gestionnaire automatique qui envoie les logs locaux vers la VM
    toutes les 30 secondes pour mettre à jour le superviseur.
    """
    while True:
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            # Sélection des logs ayant au moins 30 secondes d'existence locale
            cursor.execute("""
                SELECT id, source, action, timestamp, status
                FROM tb_logs
                WHERE timestamp <= datetime('now', '-30 seconds', 'localtime')
            """)
            logs = cursor.fetchall()

            if logs:
                payload = [
                    {"source": l[1], "action": str(l[2]).strip(), "timestamp": l[3], "status": l[4]}
                    for l in logs
                ]

                r = requests.post(VM_LOG_SYNC_URL, json=payload, timeout=5, verify=False)

                if r.status_code == 200:
                    # Suppression sélective des IDs synchronisés
                    ids = [str(l[0]) for l in logs]
                    cursor.execute(f"DELETE FROM tb_logs WHERE id IN ({','.join(ids)})")
                    conn.commit()
            conn.close()
        except:
            pass
        time.sleep(10)

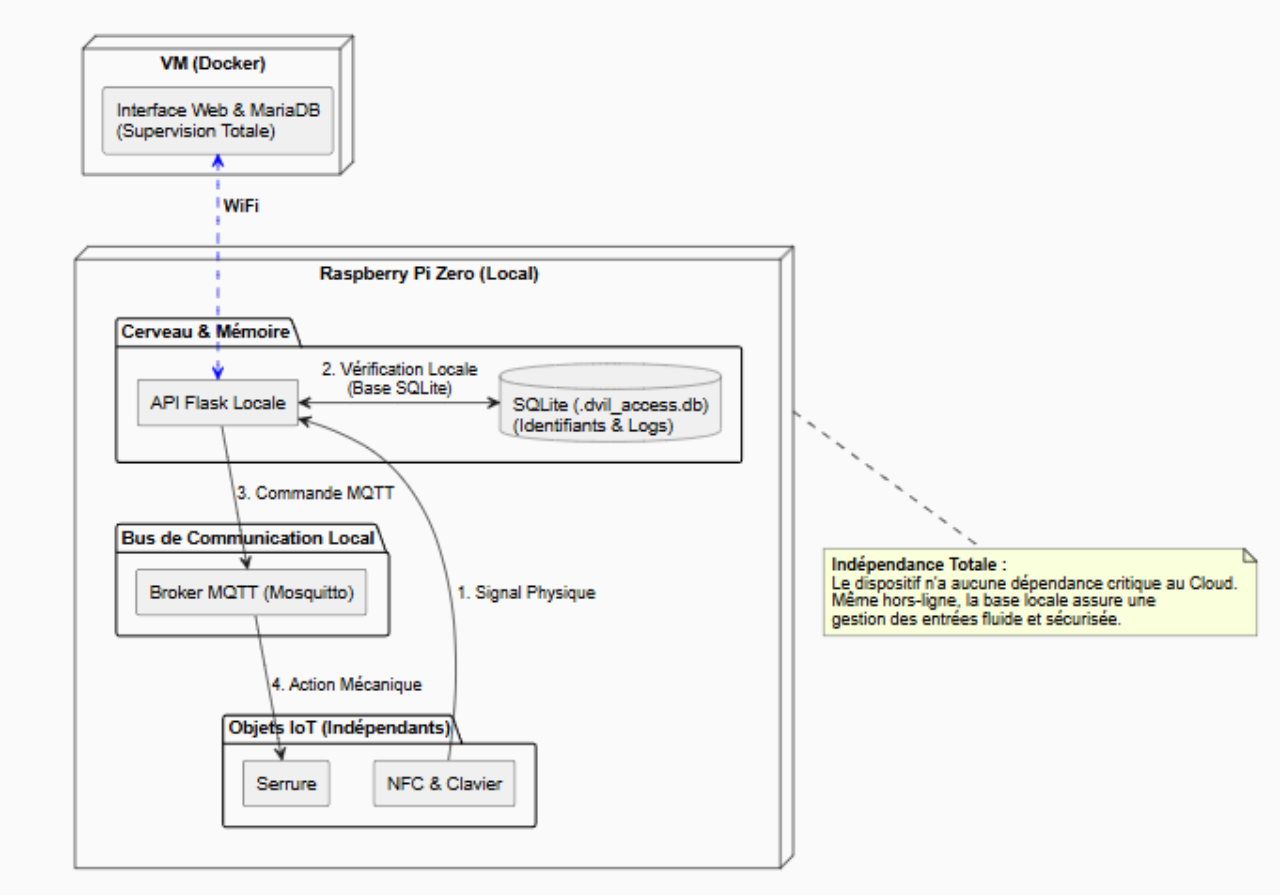
# Lancement des threads de synchronisation et de notification au démarrage
threading.Thread(target=background_log_sync, daemon=True).start()
```

Une synthèse des problèmes rencontrés durant le développement de la synchronisation et les solutions techniques apportées.

Problème rencontré	Risque associé	Solution technique appliquée
Coupure réseau **Wi-Fi**	Logs non recevable pour l'**API (VM)**	Stockage des logs dans une mini base de données **SQLite**
VM ou Serveur éteint	La serrure ne peut pas agir localement	Utilisation de l'autorisation de la base de données locale **SQLite**
Multiples connexions simultanées	Ralentissement (lag) de système **(Raspberry Pi Zero 2W)**	Utilisation d'un **Thread** avec une pause de 10s (time.sleep)
Tentative d'intrusion **(Clavier)**	L'alerte n'arrive pas au superviseur	L'incident est logué localement et envoyé dès le retour du réseau

L'architecture démontre l'indépendance opérationnelle des modules IoT (clavier, NFC et serrure) capable d'assurer la validation des accès et le stockage des événements en totale autonomie, sans dépendre d'une connexion permanente à l'API du serveur interne conteneurisé **(VM)**.

Ci-dessous un diagramme de dépendance du système interne de la **Raspberry Pi Zero 2W** :



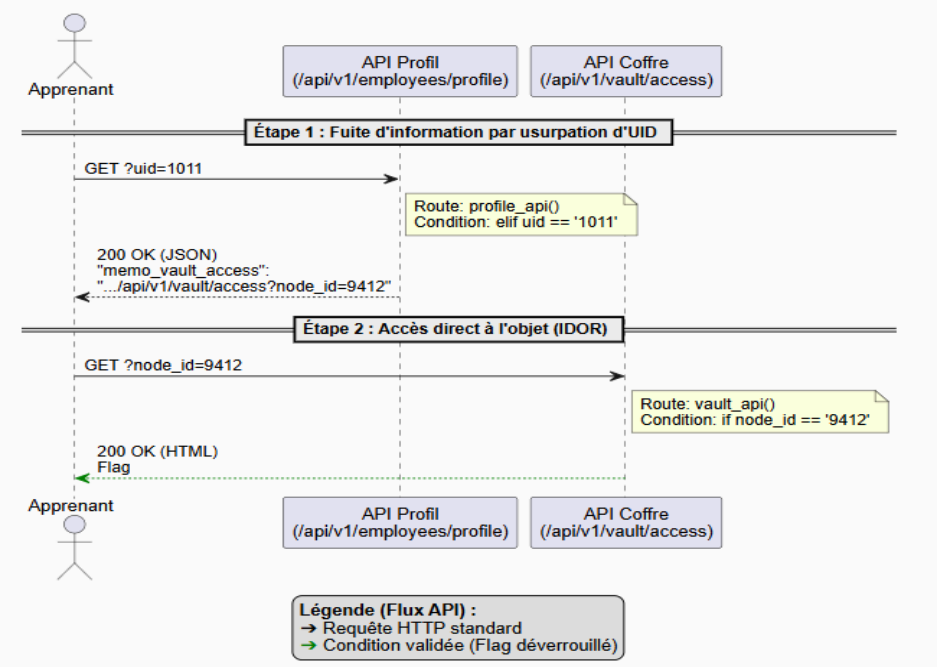
11.11 Implémentation des challenges (CTF)

Cette section décrit les 6 environnements d'entraînement déployés sur l'infrastructure de l'**interface web**. Chaque laboratoire est développé en **Python** avec **Flask** et une architecture asynchrone multithread. L'objectif est d'expliquer le fonctionnement du **backend,** le routage HTTP et les validations menant à la compromission du système et à l'obtention du flag.

1. **Exploitation de la faie IDOR (Insecure Direct Object Reference)**
   - **La faille :** L'API utilise le paramètre ?uid= sans vérifier les droits du jeton de session actif.
   - **L'exploitation :** L'attaquant usurpe l'UID administrateur **(1011)** pour fuiter le node_id, réutilisé pour ouvrir le coffre.
   - **Ce qu'il faut éviter (Remédiation) :**

- Ne jamais faire confiance aux identifiants manipulables par le client.
- Remplacer les identifiants séquentiels par des **UUID**.
- Implémenter un contrôle d'accès strict **(RBAC)** validé côté serveur.

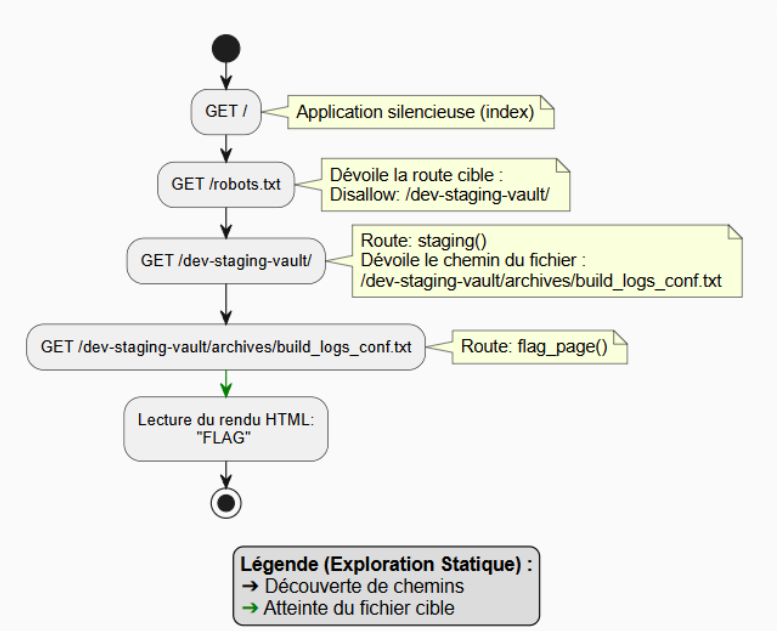
Ce diagramme de séquence ci-dessous illustre le parcour de l'exploitation de la faille à fin de trouver le flag de niveau 1 (facile):



2. Niveau 2 : Fuite d'Information

- **La faille :** Aucune faille dynamique. Un fichier robots.txt expose publiquement une arborescence cachée de pré-production.
- **L'exploitation :** Exploration manuelle de l'arborescence web (sans interface graphique) pour extraire un fichier de configuration.
- **Ce qu'il faut éviter (Remédiation) :**
  - Isoler les environnements de staging du réseau public **(VPN, filtrage IP)**.
  - Ne jamais stocker de secrets en clair dans les fichiers **(Hardcoding)**.

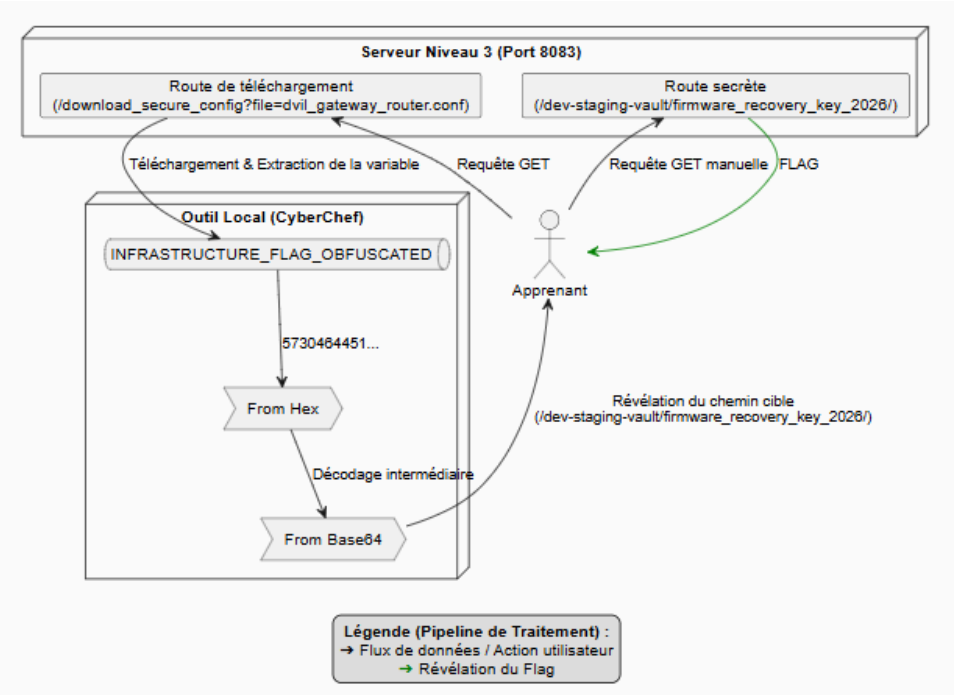
Ci-dessous un schéma logique de l'exploitation du niveau 2 :



3. Niveau 3 : Obfuscation Cryptographique

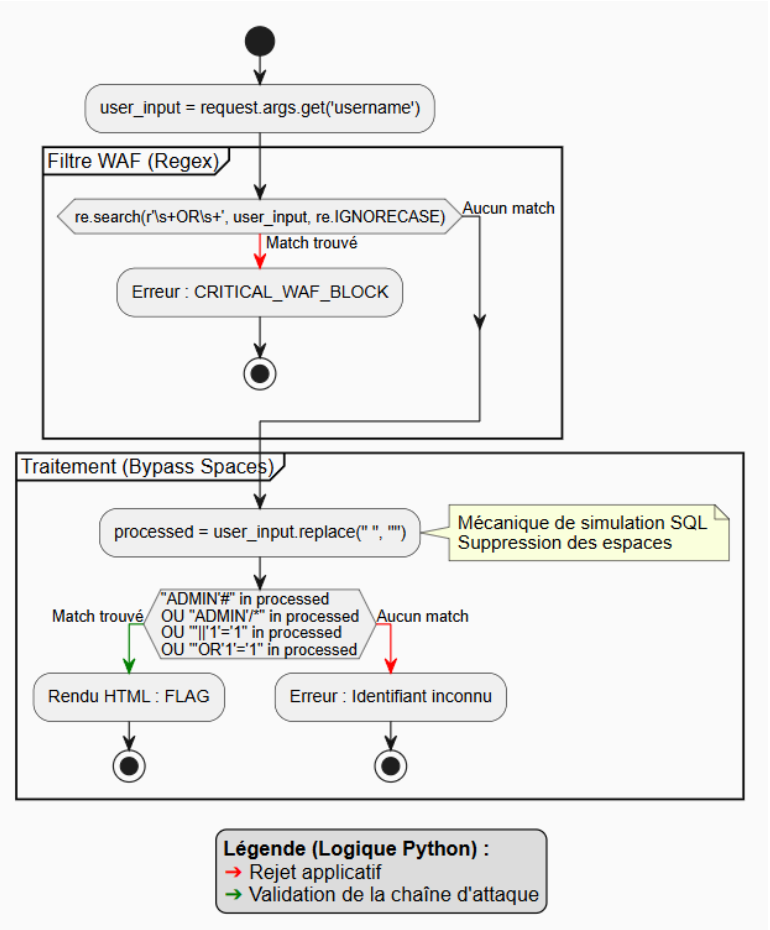
- **La faille :** Un fichier réseau expose une variable masquée par un double encodage **(Base64 + Hexadécimal)**.
- **L'exploitation :** Extraction de la chaîne et décodage inverse hors-ligne (ex: **CyberChef**) pour révéler l'URL cible.
- **Ce qu'il faut éviter (Remédiation) :**
    - Ne jamais confondre encodage **(formatage)** et chiffrement **(sécurité)**.
    - Proscrire la sécurité par l'obscurité **(Security through Obscurity)**.
    - Chiffrer toute donnée sensible au repos **(Data at Rest)** avec des standards robustes **(AES-256)**.

La diagramme de flux suivante montre le parcour du niveau 3 (facile) à exploiter :



4. **Niveau 4 : Injection SQL et Pare-feu WAF (Web Application Firewall)**
    - **La faille :** Un **WAF** filtre les espaces. La validation logicielle exige une équivalence stricte avec une liste d'approbation **(Allow-list)**.
    - **L'exploitation :** Forger une charge utile exacte (ex: ADMIN'--) en remplaçant les espaces par des commentaires **SQL**.
    - **Ce qu'il faut éviter (Remédiation) :**
        - Interdire la concaténation directe d'entrées utilisateur dans une **requête SQL**.
        - Ne pas se reposer uniquement sur un **WAF** ou des expressions régulières **(Regex)**.
        - Imposer systématiquement l'utilisation de requêtes préparées **(Prepared Statements)** ou d'un **ORM (*Object-Relational Mapping : permet de faire le lien entre une base de données et du code objet, sans écrire directement de SQL*** **)** .

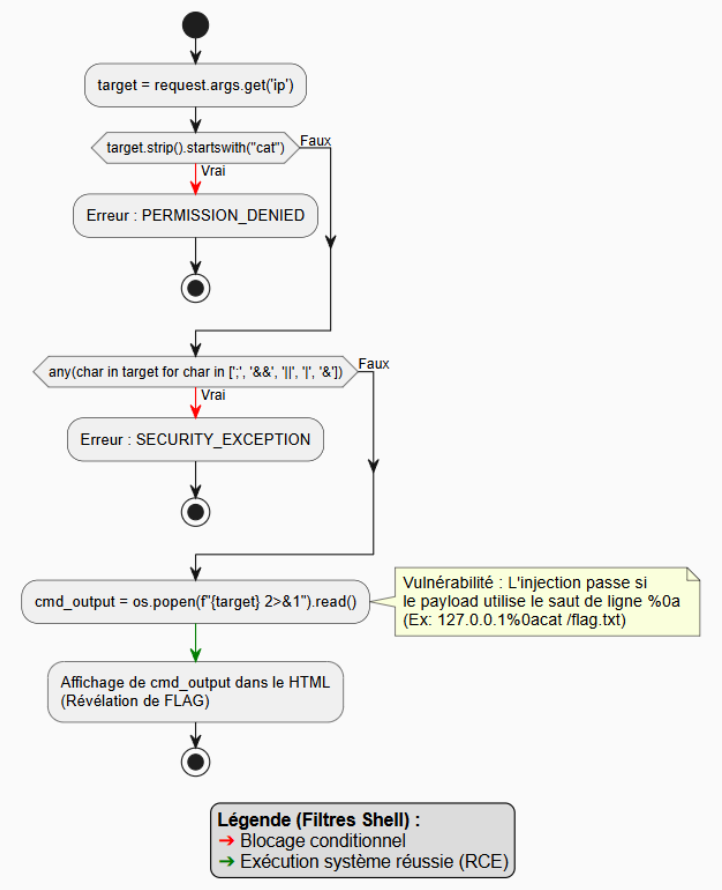
Ci-dessous la logique du parcour de l’exploitation du niveau 4 (moyen) :



5. Niveau 5 : Injection de Commandes RCE (Remote Code Execution)

- **La faille :** Le **backend** exécute un **ping système** (os.popen) avec l'entrée utilisateur. Une liste noire limite les opérateurs &&, ;.
- **L'exploitation :** Injecter un saut de ligne (%0a) pour forcer l'**OS** à traiter le payload comme une nouvelle commande isolée.
- **Ce qu'il faut éviter (Remédiation) :**
  - **Ne jamais évaluer des données non purgées via un shell système.**
  - **Ne faire aucune confiance aux blocages gérés côté Front-end (Navigateur).**
  - **Privilégier des bibliothèques de programmation natives plutôt que l'appel direct aux commandes de l'OS.**

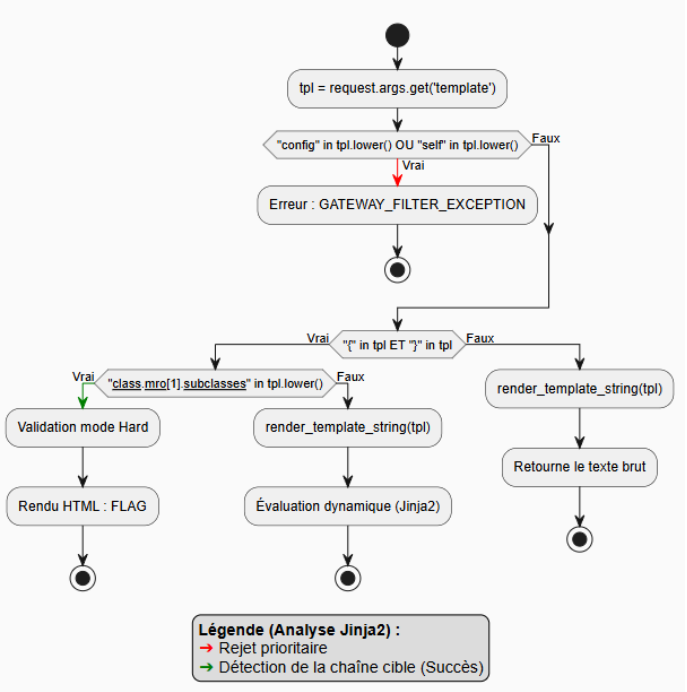
Le schéma suivante illustre l'exploitation du parcour de niveau 5 (moyen) :



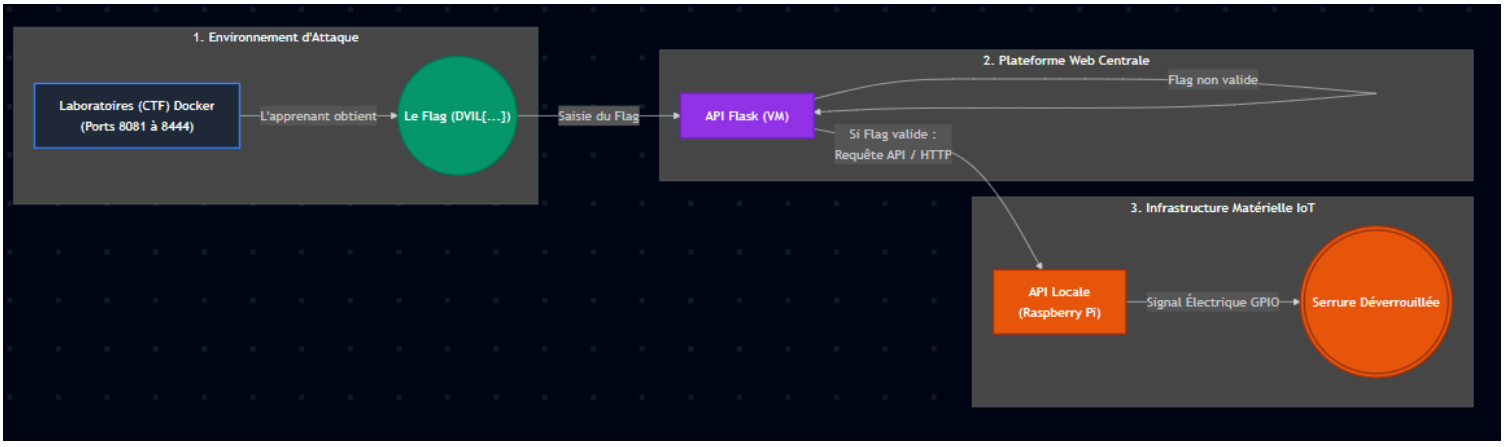
6. Niveau 6 : Introspection de Templates SSTI (Server-Side Template Injection)

- **La faille :** Sandboxing extrême. Le moteur dynamique (render_template_string) bloque tout mot-clé système **(os, import, class)**.
- **L'exploitation :** Soumettre un payload d'introspection Python complexe, unique et exact, pour contourner le filtre **Zero Trust** (Ne faire confiance à personne par défaut, même à l'intérieur du réseau).
- **Ce qu'il faut éviter (Remédiation) :**
  - Ne jamais compiler ou évaluer dynamiquement les données d'un utilisateur.
  - Séparer strictement la logique métier de la vue (**Logic-less Templates :** moteurs de templates qui limitent volontairement la logique dans les fichiers de template).
  - Transmettre les variables d'affichage passivement, sans exécution possible de code.

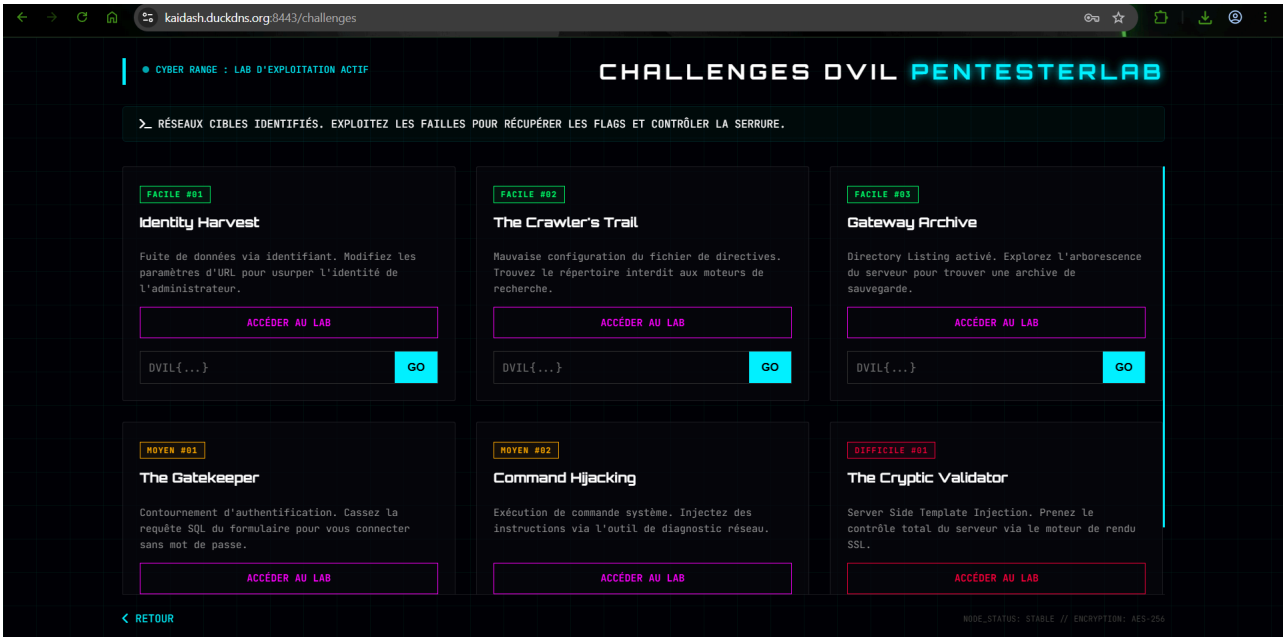
Ci-dessous l'algorigramme d'évaluation **JINJA2** pour niveau 6 (Difficile) :



7. Interprétation et représentation du déverrouillage avec la validation des flags :



Voici une vue (IHM) de l’interface de laboratoire des challenges :



12. Sécurisation et bonnes pratiques

Dans cette section, nous présentons les mécanismes de sécurité déjà implémentés dans le projet, ainsi que les pistes d'amélioration envisagées pour renforcer la robustesse et la fiabilité du système.

Plusieurs mesures de sécurisation ont été intégrées pour protéger l'application et limiter les risques d'attaques :

- **Authentification et Gestion des Sessions :**

  Les utilisateurs sont identifiés grâce à des tokens **JWT**, qui expirent automatiquement après un certain temps. Des fonctions spéciales (comme require_auth ou require_admin) contrôlent l'accès aux pages sensibles. Flask-Login est aussi utilisé pour gérer les connexions des utilisateurs.

- **Sécurisation de l'infrastructure réseau :**

  L'application **Flask** n'est jamais exposée directement sur internet. Elle est isolée derrière un serveur **Nginx**, qui fait office de bouclier, gère les flux **HTTP** et permet une future mise en œuvre de **certificats SSL (HTTPS)**.

  Un mécanisme de limitation **(Rate Limiting)** est appliqué sur l'**API** pour bloquer les tentatives d'attaques par force brute sur le code PIN ou les identifiants de connexion.

- **Confidentialité et conformité des données :**

  La base de données **SQLite** située sur la **Raspberry Pi Zero 2W** est volontairement incomplète. Elle ne contient aucune donnée nominative (nom, prénom), mais uniquement des identifiants techniques **(UUID)** et des codes PIN.

- **Résilience et intégrité du système :**

  En cas de coupure réseau ou de panne du serveur central, la serrure bascule automatiquement sur la base de données locale **(SQLite)**. Cela garantit que les utilisateurs autorisés ne restent jamais bloqués.

  L'utilisation de **Threads** (processus en arrière-plan) pour la remontée des logs assure qu'aucun événement n'est perdu. Si la connexion échoue, les logs sont conservés dans la file d'attente **SQLite** et renvoyés dès le rétablissement du lien.

- **Journalisation et Suivi des Événements :**

  Chaque tentative d'accès, qu'elle soit physique (clavier) ou numérique (API), est horodatée et archivée.

L'intégration de **SocketIO** permet à l'administrateur de surveiller les flux de données et l'état de la serrure en direct, facilitant la détection immédiate de comportements suspects.

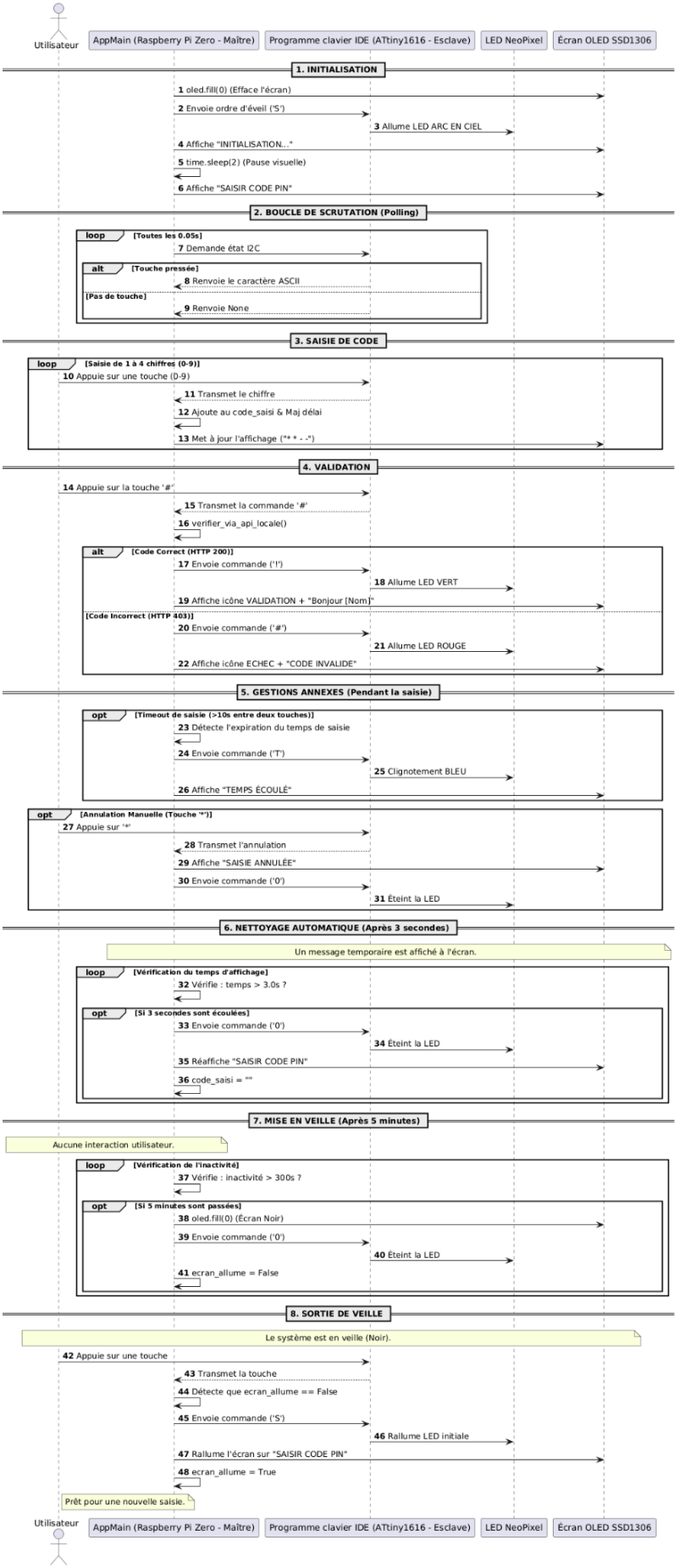
13. Développement de l’interface clavier [Annexe 8]

Le module clavier repose sur une architecture décentralisée. Contrairement à la version du projet 2025, la gestion physique des touches est déportée sur un microcontrôleur **ATtiny1616**. Ce choix technique permet de libérer des ressources sur la **Raspberry Pi Zero 2W** et de garantir une réactivité optimale du balayage (scanning) de la matrice.

L'application clavier met en oeuvre l'interface :

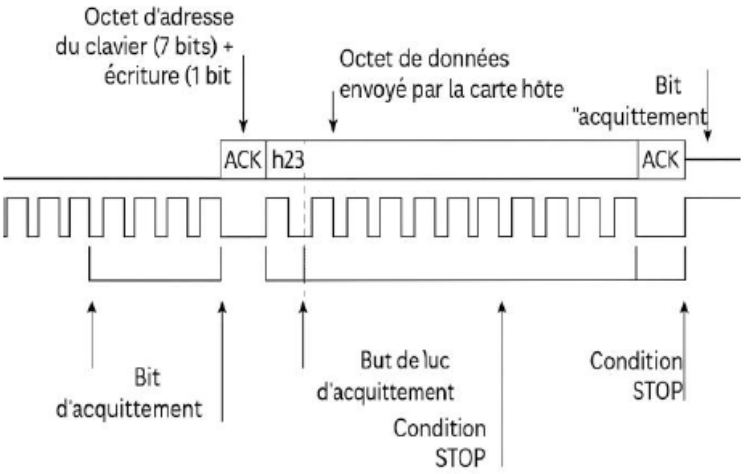
- **Acquisition :** Elle interroge l'**ATtiny** via le **bus I2C** pour récupérer les touches pressées.
- **Traitement :** Les codes PIN sont mémorisés temporairement.
- **Validation :** Une fois validé, le code est envoyé à l'API locale (`/api/local/verify`).
- **Feedback :** L'état (Succès/Échec) est renvoyé à l'utilisateur via l'afficheur **OLED** et la **LED** gérée par l'**ATtiny**.

Cette séquence illustre la chaîne de liaison complète du **clavier** physique à l'**ATtiny**, puis vers la **Raspberry Pi Zero 2W** via le **bus I2C**, incluant les périphériques de sortie **(OLED/LED)** :


Utilisateur
AppMain (Raspberry Pi Zero - Maître)
Programme clavier IDE (ATtiny1616 - Esclave)
LED NeoPixel
Écran OLED SSD1306
1. INITIALISATION
1 oled.fill(0) (Efface l'écran)
2 Envoie ordre d'éveil ('S')
3 Allume LED ARC EN CIEL
4 Affiche "INITIALISATION..."
5 time.sleep(2) (Pause visuelle)
6 Affiche "SAISIR CODE PIN"
2. BOUCLE DE SCRUTATION (Polling)
loop [Toutes les 0.05s]
7 Demande état I2C
alt [Touche pressée]
8 Renvoie le caractère ASCII
[Pas de touche]
9 Renvoie None
3. SAISIE DE CODE
loop [Saisie de 1 à 4 chiffres (0-9)]
10 Appuie sur une touche (0-9)
11 Transmet le chiffre
12 Ajoute au code_saisi & Maj délai
13 Met à jour l'affichage ("* * - -")
4. VALIDATION
14 Appuie sur la touche '#'
15 Transmet la commande '#'
16 verifier_via_api_locale()
alt [Code Correct (HTTP 200)]
17 Envoie commande ('I')
18 Allume LED VERT
19 Affiche icône VALIDATION + "Bonjour [Nom]"
[Code Incorrect (HTTP 403)]
20 Envoie commande ('#')
21 Allume LED ROUGE
22 Affiche icône ECHEC + "CODE INVALIDE"
5. GESTIONS ANNEXES (Pendant la saisie)
opt [Timeout de saisie (>10s entre deux touches)]
23 Détecte l'expiration du temps de saisie
24 Envoie commande ('T')
25 Clignotement BLEU
26 Affiche "TEMPS ÉCOULÉ"
opt [Annulation Manuelle (Touche '*')]
27 Appuie sur '*'
28 Transmet l'annulation
29 Affiche "SAISIE ANNULÉE"
30 Envoie commande ('0')
31 Éteint la LED
6. NETTOYAGE AUTOMATIQUE (Après 3 secondes)
Un message temporaire est affiché à l'écran.
loop [Vérification du temps d'affichage]
32 Vérifie : temps > 3.0s ?
opt [Si 3 secondes sont écoulées]
33 Envoie commande ('0')
34 Éteint la LED
35 Réaffiche "SAISIR CODE PIN"
36 code_saisi = ""
7. MISE EN VEILLE (Après 5 minutes)
Aucune interaction utilisateur.
loop [Vérification de l'inactivité]
37 Vérifie : inactivité > 300s ?
opt [Si 5 minutes sont passées]
38 oled.fill(0) (Écran Noir)
39 Envoie commande ('0')
40 Éteint la LED
41 ecran_allume = False
8. SORTIE DE VEILLE
Le système est en veille (Noir).
42 Appuie sur une touche
43 Transmet la touche
44 Détecte que ecran_allume == False
45 Envoie commande ('S')
46 Rallume LED initiale
47 Rallume l'écran sur "SAISIR CODE PIN"
48 ecran_allume = True
Prêt pour une nouvelle saisie.
Utilisateur
AppMain (Raspberry Pi Zero - Maître)
Programme clavier IDE (ATtiny1616 - Esclave)
LED NeoPixel
Écran OLED SSD1306

13.1 Protocole de communication I2C

Cette image représente une trame **I2C** interceptée, annotée pour en faciliter la lecture. Elle illustre une transmission typique d'un code clavier entre un périphérique maître **(carte hôte)** et un esclave **(clavier I2C)**. Le format respecte le protocole **I2C** standard, utilisant les lignes **SDA (data)** et **SCL (clock)**.



h09 WR — Octet d'adresse

- **h09 (binaire : 00001001)** correspond à l'adresse du périphérique clavier.
- Le bit **WR (Write)** à 0 indique une transmission de données du maître vers l'esclave.
- Le **maître (carte hôte)** s'adresse au clavier pour lui transmettre une donnée.

ACK — Acknowledge

- Le clavier répond à la requête en tirant **SDA** bas pendant un cycle d'horloge pour confirmer la réception de l'adresse.

h23 — Donnée transmise

- **h23 (binaire : 00100011)** est la valeur transmise — dans ce contexte, elle pourrait représenter un code PIN ou une touche clavier.
- C'est l'information sensible qu'un attaquant pourrait tenter de récupérer.

ACK — Deuxième acquittement

- Le périphérique confirme à nouveau la réception de la donnée envoyée **(h23)**.

Stop — Condition de fin

- SDA passe de bas à haut pendant que **SCL** est haut → signal de fin de transmission **I2C**.

13.2 Faille de vulnérabilité du clavier

Une faille peut toutefois survenir si la liaison **I2C** entre le clavier et la carte hôte est interceptée :

- Un attaquant équipé d'un sniffer **I2C** pourrait récupérer les codes transmis en clair.
- Cela mettrait en péril la confidentialité et l'intégrité du système.

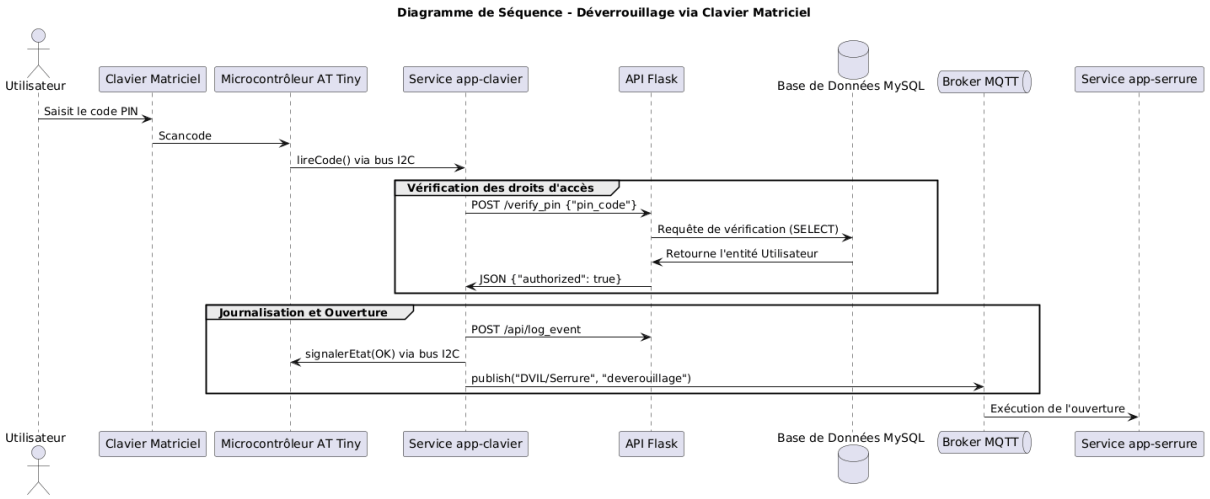
Contre Mesures recommandées :

1. Chiffrer ou obfusquer les données sur le bus I2C.
2. Protéger physiquement l'accès au matériel.
3. Journaliser et surveiller les accès inhabituels sur le bus.

Ci-dessous les risques et recommandations pour la sécurisation de l'infrastructure clavier :

Description	Risque	Solution
Lecture du code PIN via les fils **SDA/SCL**	Interception **I2C**	Protection physique du câblage
Simulation d'une touche "Validation" par un esclave pirate	Injection de données	Utilisation de résistances de pull-up spécifiques et contrôle de l'intégrité logicielle

Ci-dessous la séquence de déverrouillage depuis le clavier :

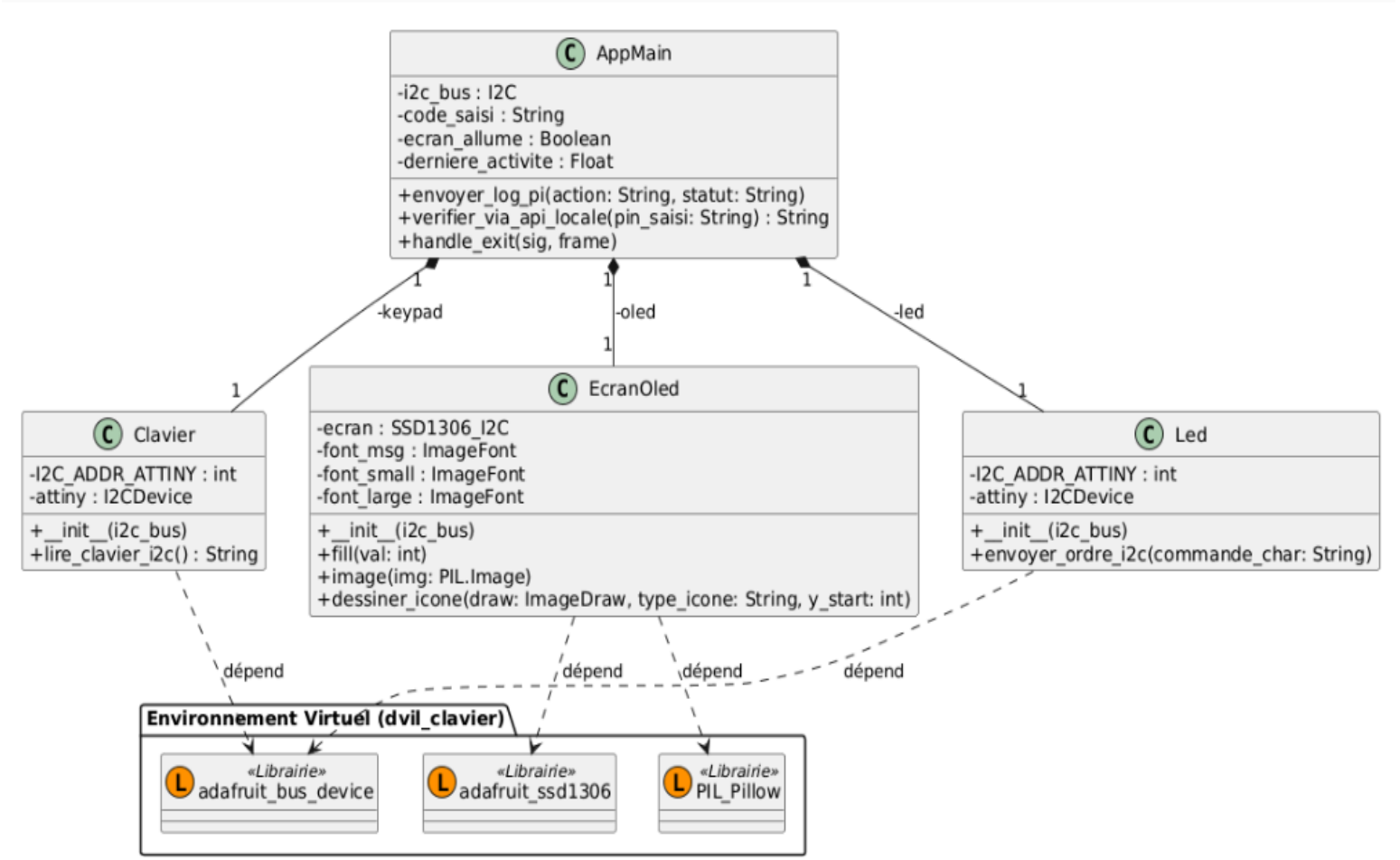


13.3 Développement de l’application clavier

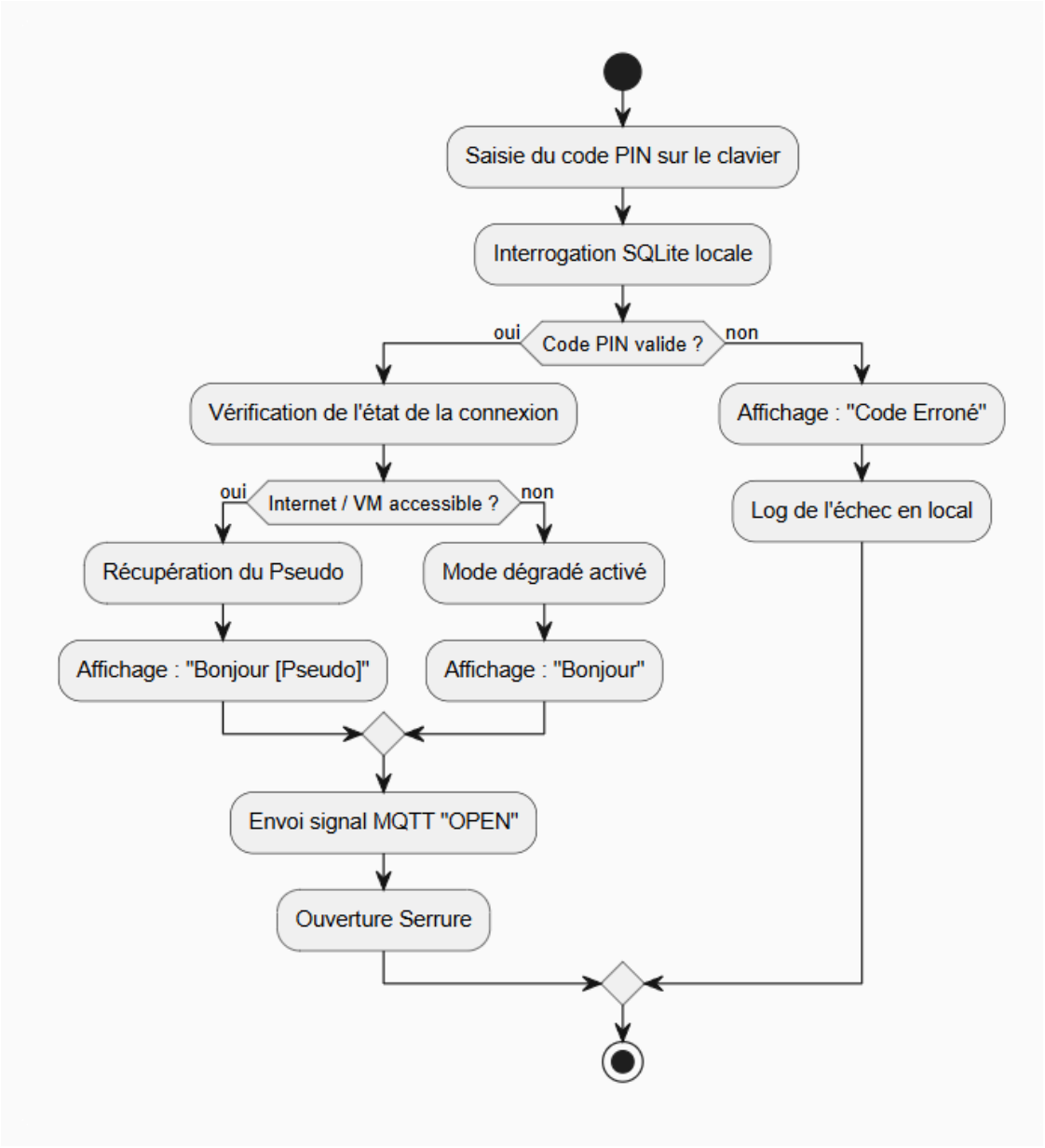
1. Modélisation logicielle

La modélisation logicielle détaille l'organisation des méthodes permettant l'acquisition des touches matricielles et le pilotage des indicateurs visuels via le bus **I2C**.

Ce diagramme de classes présente l’architecture logicielle du système de contrôle d’accès : On peut voir la répartition entre le fonctionnement **clavier**, l'affichage d’**écra**n et la signalisation de la **LED** par type d’action :



2. **Schéma logique du fonctionnement de l'interface clavier à fin de déverrouiller la serrure :**



14. Partie Physique : Étude de la Couche Physique de Supervision (Wi-Fi 2,4 GHz)

Cette section présente l'étude de la couche physique appliquée à la transmission **radiofréquence (RF)** du projet **DVIL**. Elle qualifie l'intégration de la **Raspberry Pi Zero 2W** à l'intérieur du boîtier pupitre final et analyse la propagation des ondes électromagnétiques dans l'environnement de la salle de classe jusqu'au point d'accès réseau.

La communication sans fil repose sur une **clé Wi-Fi USB compatible IEEE 802.11n**, connectée à la Raspberry Pi Zero 2W, l'ensemble étant entièrement intégré à l'intérieur du boîtier. **[voir annexe 8]**

14.1 Étude de la propagation et bilan de liaison Wi-Fi

Cette section vise à valider la viabilité de la liaison sans fil par une approche **théorique rigoureuse**. Nous étudierons les propriétés de propagation de l'onde électromagnétique au sein de notre environnement, puis nous établirons le bilan de puissance du signal afin de confirmer la stabilité de la transmission **Wi-Fi** entre le boîtier et le point d'accès.

14.2 Développement et Calculs Théoriques

Pour commencer, nous appliquons les lois fondamentales de la physique des ondes électromagnétiques **ultra-haute fréquence (UHF)**.

1. Détermination de la Longueur d'Onde

La longueur d'onde représente la distance physique parcourue par l'onde pendant une période du signal. Elle dépend de la vitesse de propagation **(célérité c)** et de la **fréquence (f)**.

- **Célérité dans l'air (c) :** On sait que la vitesse des ondes électromagnétiques est de 3×10^8 m/s .
- **Fréquence centrale du Wi-Fi (f) :** Pour le Wi-Fi, elle utilise la fréquence centrale de 2,4 GHz = $2,4 \times 10^9$ Hz .

À l'aide de la formule de la **longueur d'onde (λ)** on a :

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8}{2,4 \times 10^9} \approx 0,125 \text{ m (soit 12,5 cm)}$$

La longueur d'onde est d'environ **12 cm** permettant de caractériser l'échelle spatiale du signal **Wi-Fi**. Cette grandeur est ensuite utilisée comme référence pour analyser l'interaction de l'onde avec les matériaux du boîtier et justifier le choix du matériau.

2. Évaluation de la Perte en Espace Libre

Pour une distance d'utilisation entre le boîtier et le routeur de **2m** dans la salle de classe à la **fréquence de 2,4 GHz**, la formule de **Friis** normalisée de **perte en espace libre (FSL)** s'exprime ainsi :

$$FSL\ (dB) = -147.55 + 20 \times \log(f) + 20 \times \log(d)$$

- Avec **d** en **m**.
- Avec **f** en **Hz**.

Donc on a :

$$FSL(dB) = -147{,}55 + 20 \times \log(2{,}4 \times 10^{9}) + 20 \times \log(2)$$

$$\Leftrightarrow \quad FSL(dB) = -147{,}55 + 187{,}78 + 6{,}02 = 46{,}07\ dB$$

La puissance reçue est estimée à partir du modèle de **Friis**, en tenant compte de la puissance d'émission, des gains d'antennes et des pertes en espace libre :

$$P_r = P_e + G_r + G_e - FSL$$

- P_r : niveau de puissance reçue, exprimé en **dBm**.
- P_e : niveau de puissance transmise à l'antenne d'émission, exprimé en **dBm**.
- FSL : pertes en espace libre, exprimé en **dB**.
- G_e : gain de l'antenne d'émission (clé USB Wi-Fi 802.11n), exprimé en dBi.
- G_r : gain de l'antenne de réception, exprimé en **dBi**.

La **puissance reçue** par le routeur dépend de la **puissance d'émission** de l'antenne qui possède la **clé USB Wi-Fi 802.11n (estimation 18 dBm)** et des **gains cumulés des antennes d'émission et de réception** (pour chaque antenne on estime environ **2 dBi)** :

Donc on a :

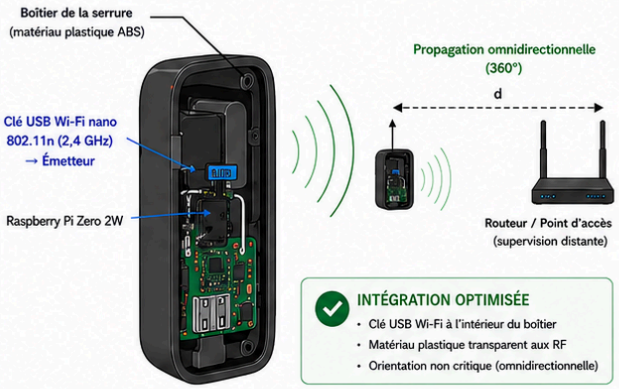
$$P_r = 18 + 2 + 2 - 46{,}07 = -24.07\ dBm$$

Finalement, l'atténuation en espace libre (FSL) donne une perte d'environ **46 dB** à une distance de **2 mètres**, ce qui nous donne une puissance reçue théorique d'environ **-24 dBm**. Le seuil critique de sensibilité d'un récepteur **Wi-Fi** pour décoder les données sans pertes se situe à **-70 dBm**, notre résultat offre une marge de sécurité confortable d'environ **45 dB**. Le choix du **Wi-Fi** Infrastructure est donc parfaitement validé pour assurer une supervision déportée stable et continue dans la salle informatique.

3. Document technique du système de communication Wi-Fi

1) POSITIONNEMENT PHYSIQUE : Wi-Fi 2,4 GHz

Le module Wi-Fi (Raspberry Pi Zero 2W + clé USB Wi-Fi nano 802.11) est intégré dans la serrure et communique en 2,4 GHz avec le routeur distant.
La clé USB Wi-Fi agit comme l'émetteur : elle diffuse l'onde omnidirectionnelle.



INTÉGRATION OPTIMISÉE
- Clé USB Wi-Fi à l'intérieur du boîtier
- Matériau plastique transparent aux RF
- Orientation non critique (omnidirectionnelle)

À ÉVITER : Boîtier métallique
Le métal crée une Cage de Faraday qui réfléchit le signal et empêche toute communication.



COMPARAISON DES MATÉRIAUX DU BOÎTIER

Critère	Plastique ABS (Diélectrique)	Métal (Conducteur)
Permittivité relative (ε_r)	≈ 2,5 – 3	Très élevée (non pertinente en RF)
Conductivité électrique (σ)	≈ 10^{-12} S/m	≈ 10^{6} S/m
Transmission RF à 2,4 GHz	Excellente (atténuation < 1 dB)	Nulle (réflexion totale)
Phénomène dominant	Transmission de l'onde	Courants de Foucault (Cage de Faraday)
Impact sur le signal Wi-Fi	Propagation libre	Signal confiné / Bloqué
Conséquence pour la supervision	Connexion stable	Connexion impossible
Conclusion	**Matériau recommandé**	**À éviter impérativement**

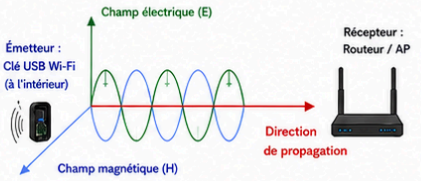
Le positionnement physique et le choix d'un boîtier plastique (ABS) garantissent la propagation libre de l'onde UHF (2,4 GHz) et une liaison stable avec le routeur distant (1 à 2 m en salle de classe).
La clé USB Wi-Fi nano agit comme l'émetteur omnidirectionnel.

2) PRINCIPE PHYSIQUE DU SIGNAL : PROPAGATION & ATTÉNUATION

Le Wi-Fi 2,4 GHz utilise des ondes électromagnétiques UHF.
Le signal se propage dans l'espace et subit une atténuation liée à la distance.

1 NATURE DE L'ONDE ÉLECTROMAGNÉTIQUE (UHF)

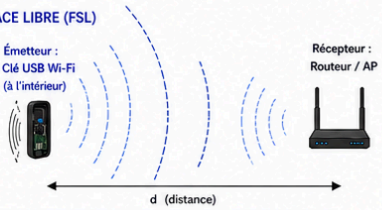
La clé USB Wi-Fi nano (à l'intérieur de la serrure) agit comme l'émetteur : elle génère des ondes électromagnétiques sinusoïdales composées d'un champ électrique (E) et d'un champ magnétique (H) perpendiculaires entre eux et à la direction de propagation.



2 PROPAGATION & ATTÉNUATION EN ESPACE LIBRE (FSL)

L'antenne de la clé USB Wi-Fi rayonne l'énergie de manière omnidirectionnelle. Plus la distance (d) augmente, plus l'énergie se répartit sur une surface sphérique de plus en plus grande ($4\pi d^2$).

La puissance reçue diminue donc avec le carré de la distance : c'est l'affaiblissement en espace libre (FSL).



Conséquence : Plus la distance augmente → plus la puissance reçue diminue → plus l'atténuation (FSL) augmente.

3 PARAMÈTRES PHYSIQUES CLÉS

LONGUEUR D'ONDE (λ)

$$\lambda = \frac{c}{f}$$

c : célérité de la lumière ≈ 300 000 000 m/s
f : fréquence Wi-Fi = 2,45 GHz = 2 450 000 000 Hz

$\lambda \approx 12{,}2$ cm

Pertinence : Détermine la taille idéale de l'antenne. Antenne quart-d'onde ≈ λ/4 ≈ 3 cm (taille typique de la clé USB Wi-Fi nano 802.11n intégrée).

AFFAIBLISSEMENT EN ESPACE LIBRE (FSL)

Formule (dB) :

$$L_{FSL} = 20\log_{10}(d) + 20\log_{10}(f) - 147{,}55$$

où :
d : distance (mètres)
f : fréquence (Hz)

Pertinence : Montre que l'atténuation augmente géométriquement avec la distance.

EXEMPLE NUMÉRIQUE
(f = 2,45 GHz = 2 450 000 000 Hz, d = 2 m)

$$L_{FSL} = 20\log_{10}(2) + 20\log_{10}(2{,}45 \times 10^{9}) - 147{,}55$$
$$L_{FSL} \approx 46{,}24 \text{ dB}$$

NIVEAU DE PUISSANCE REÇUE (MODÈLE DE FRIIS SIMPLIFIÉ)

$$P_r = P_e + G_e + G_r - L_{FSL}$$
$$P_r = 18 + 2 + 2 - 46{,}24$$
$$P_r \approx -24{,}24 \text{ dBm}$$

Marge de sécurité ≈ 45 dB (seuil récepteur ≈ -70 dBm)

RÉSUMÉ DU PRINCIPE PHYSIQUE

Le Wi-Fi 2,4 GHz se propage en espace libre et subit une atténuation liée à la distance.
Pour une liaison stable en salle de classe (1 à 2 m) :
- Le FSL à 2 m ≈ 46,24 dB → puissance reçue ≈ -24,24 dBm
- Large marge par rapport au seuil de sensibilité (≈ -70 dBm)

→ Connexion stable et fiable pour la supervision.

SÉCURITÉ DE LA COMMUNICATION
- Authentification : JWT (JSON Web Token)
- Chiffrement : TLS / HTTPS

15. Conclusion et perspectives du projet DVIL

15.1 Bilan personnel

Ce projet a été très enrichissant pour moi, autant au niveau technique que dans le domaine de la sécurité. J'ai travaillé sur plusieurs éléments : création d'un **poste de supervision (interface web)** conteneurisé avec **Docker,** communication **MQTT** et **API**, gestion des connexions utilisateurs et suivi des événements, et l'interaction avec du matériel physique **(IoT)**. Cela m'a permis d'améliorer mes compétences en **développement web** et en **protection des applications**.

Grâce aux exercices **(CTF)** proposés, j'ai pu mieux comprendre les vulnérabilités courantes comme les injections **SQL**, les failles **SSTI** ou encore la désérialisation non sécurisée. J'ai également appris l'importance de coder de manière sécurisée et rigoureuse. Ce projet m'a permis non seulement de progresser techniquement, mais aussi de prendre conscience qu'il faut être précis et vigilant lorsqu'on conçoit des systèmes informatiques ainsi que la reprise du projet déjà débuté afin de l'améliorer.

Évolutions envisagées

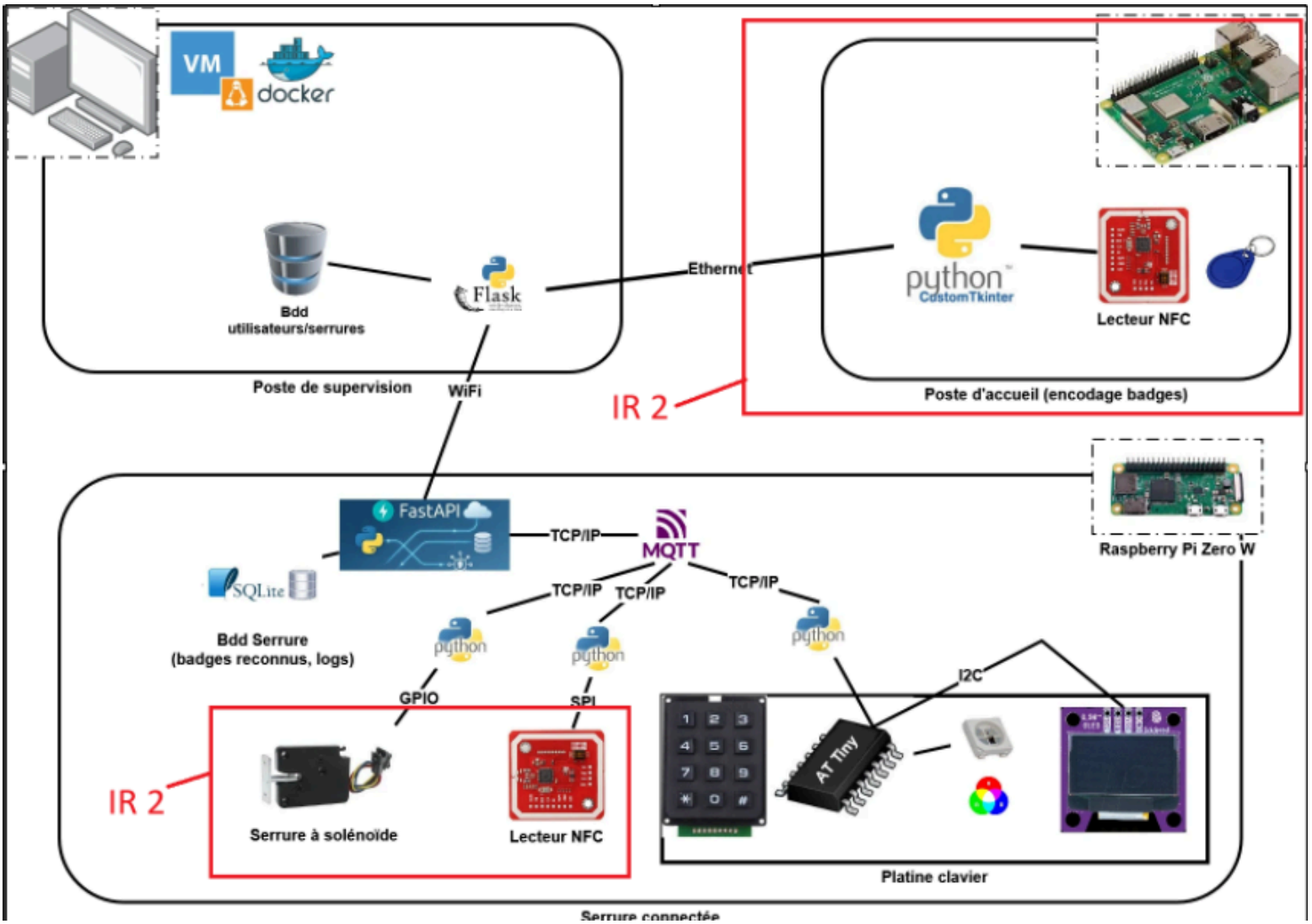
Bien que l'infrastructure du projet **DVIL** soit aujourd'hui fonctionnelle et réponde au cahier des charges initial, plusieurs axes d'amélioration peuvent être envisagés pour en faire une plateforme d'entraînement de niveau professionnel :

- **Sécurisation des communications :** Les échanges entre l'**API (VM)** et l'**API Locale (Raspberry Pi)** pourraient être protégés avec le protocole **HTTPS** afin de chiffrer les données, même pour les communication par **MQTT** qui ne sont pas chiffrés dont une sécurisation en **MQTTS** . Le système pourrait aussi être isolé sur un réseau dédié **(VLAN : Virtual Local Area Network)** pour renforcer la sécurité.
- **Amélioration des exercices :** Des "**Flags dynamiques**" pourraient être générés automatiquement pour chaque **utilisateur** (apprenant) afin d'éviter le partage des réponses. Un tableau des **scores** en direct pourrait également être ajouté.
- **Optimisation du déploiement :** Pour une utilisation avec une classe entière, l'infrastructure pourrait évoluer vers **Kubernetes (k8s)** afin de gérer automatiquement plusieurs environnements Docker simultanément.
- **Supervision et surveillance réseau :** L'ajout d'un système de monitoring permettrait de visualiser les activités des étudiants en temps réel (tentatives d'attaques, détection WAF, logs réseau, etc.).

Ces évolutions envisagées permettront non seulement de consolider la robustesse du système, mais également d'ouvrir la voie à des applications plus complexes et résilientes, en phase avec les exigences actuelles en matière de cybersécurité.

16. Partie Spécifique IR2 : Interface NFC et Enrôlement (Haitam EL KADMIRI)

Synoptique partie IR2 du projet DVIL



Cahier des charges et tâches à réaliser

Étudiant n°2 **IR2**	**Interface Serrure / Interface NFC** • Se renseigner sur la technologie NFC • Se renseigner sur le bus SPI • S'approprier le fonctionnement du lecteur NFC PN532 sur bus SPI • Définir avec **IR1** la messagerie entre les différentes applications du système • Concevoir/Coder/Tester une application permettant de s'interfacer via MQTT avec l'application de commande de la serrure depuis le lecteur NFC sur bus SPI • Concevoir/Coder/Tester une application de contrôle/commande de gâche électrique recevant les ordres d'un broker MQTT • Collaborer avec **IR1** pour imaginer des vulnérabilités et la manière de les	**Installation/Configuration** : • Environnements de développement Raspberry Pi ZeroW • Broker MQTT **Mise en œuvre :** • Programmation Qt/C++ et/ou Python • MQTT • Technologie NFC • Bus SPI • GPIO (libgpiod) **Réalisation :** • Application de pilotage de serrure via badge NFC • Application de contrôle/commande d'une gâche électrique **Documentation :** • Guide de démarrage rapide • Gestion de version logicielle

	exposer (firmwares différents, conteneurs Docker) • Collaborer étroitement avec les étudiants **IR1** et **ER** • Assurer la gestion de version logicielle (forge logicielle Framagit) • Rédiger avec **IR1** un manuel de démarrage rapide pour l'installation et l'utilisation des différents constituants • Collaborer étroitement avec étudiants **IR1** et **EC1**	• Dossier de développement

16.1 Planification (Gantt)



16.2 Objectifs de la mission

Dans le cadre de l'évolution du projet DVIL (Damn Vulnerable Internet Lock), ma mission principale consiste à fiabiliser et sécuriser le contrôle d'accès physique par technologie sans contact. Mes objectifs se concentrent sur les axes suivants :

Mise en œuvre d'une solution NFC sécurisée

- **Implémentation du bus SPI :** Le SPI a été choisi pour diversifier les moyens de communication sur le boîtier DVIL. Ce bus, natif sur la Raspberry Pi et pris en charge par le lecteur NFC, garantit une latence minimale lors de la détection.
- **Sécurisation logicielle (UUID) :** Développer une stratégie de protection contre le clonage d'identifiants physiques (UID) en intégrant un identifiant unique universel (UUID) chiffré dans les secteurs protégés de la puce.

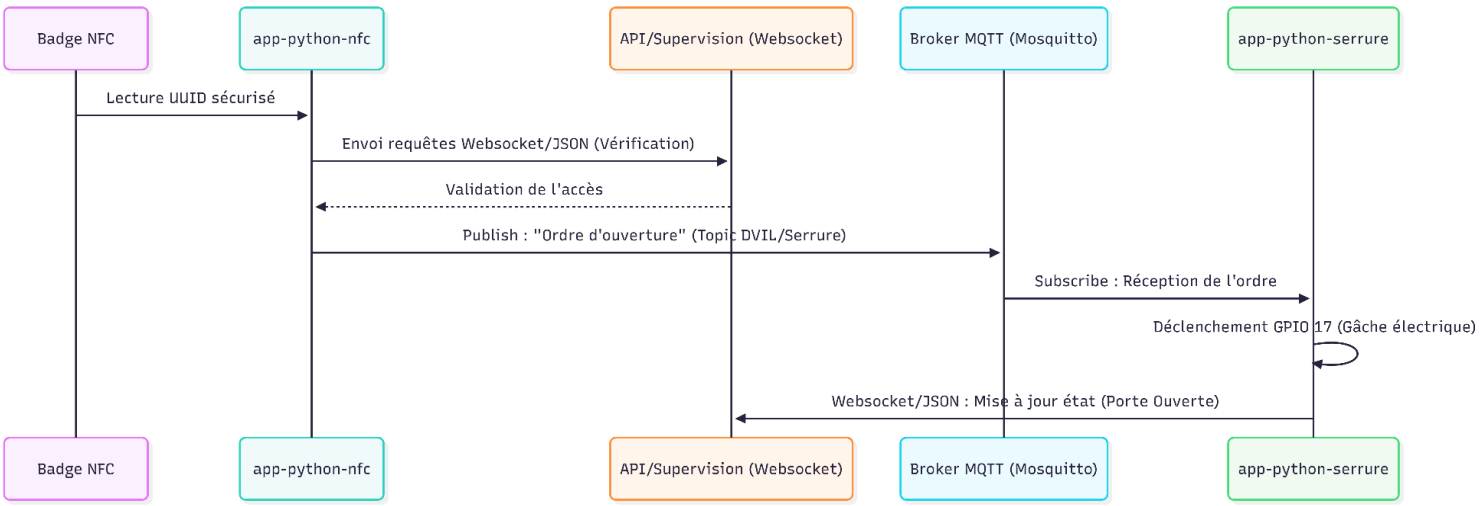
Orchestration IoT et contrôle de l'actionneur (MQTT)

Pilotage de la gâche : Développer l'application de contrôle/commande de la gâche électrique permettant l'ouverture sécurisée de la serrure.

Communication Broker : Configurer la réception des ordres sécurisés provenant du broker MQTT (Mosquitto) pour synchroniser l'état de la serrure avec l'interface de supervision en fonction des tentatives d'accès par clavier, badge et interface web.

Application NFC : Développer le service de lecture capable d'interroger l'API locale et de publier sur le broker MQTT.

Application Serrure : Concevoir le service de contrôle de la gâche électrique (GPIO 17) abonné aux ordres MQTT.



Développement du poste d’enrôlement

- Interface d'administration : Concevoir une IHM moderne (CustomTkinter) pour la création des profils utilisateurs et la gravure sécurisée des badges.
- Interfaçage API : Assurer la transmission des données d'enrôlement vers leposte de supervision via des requêtes JSON sécurisées.

16.1.1 Présentation du poste d'accueil

Présentation du poste d’accueil (Système d’enrôlement)

Dans le cadre du projet DVIL, le poste d'accueil est le dispositif spécifiquement dédié à l'encodage des badges d'accès (l'interface centrale d'administration étant gérée par le poste de supervision). Il assure le lien entre l'identité physique d'un utilisateur et son identité numérique enregistrée dans le système.

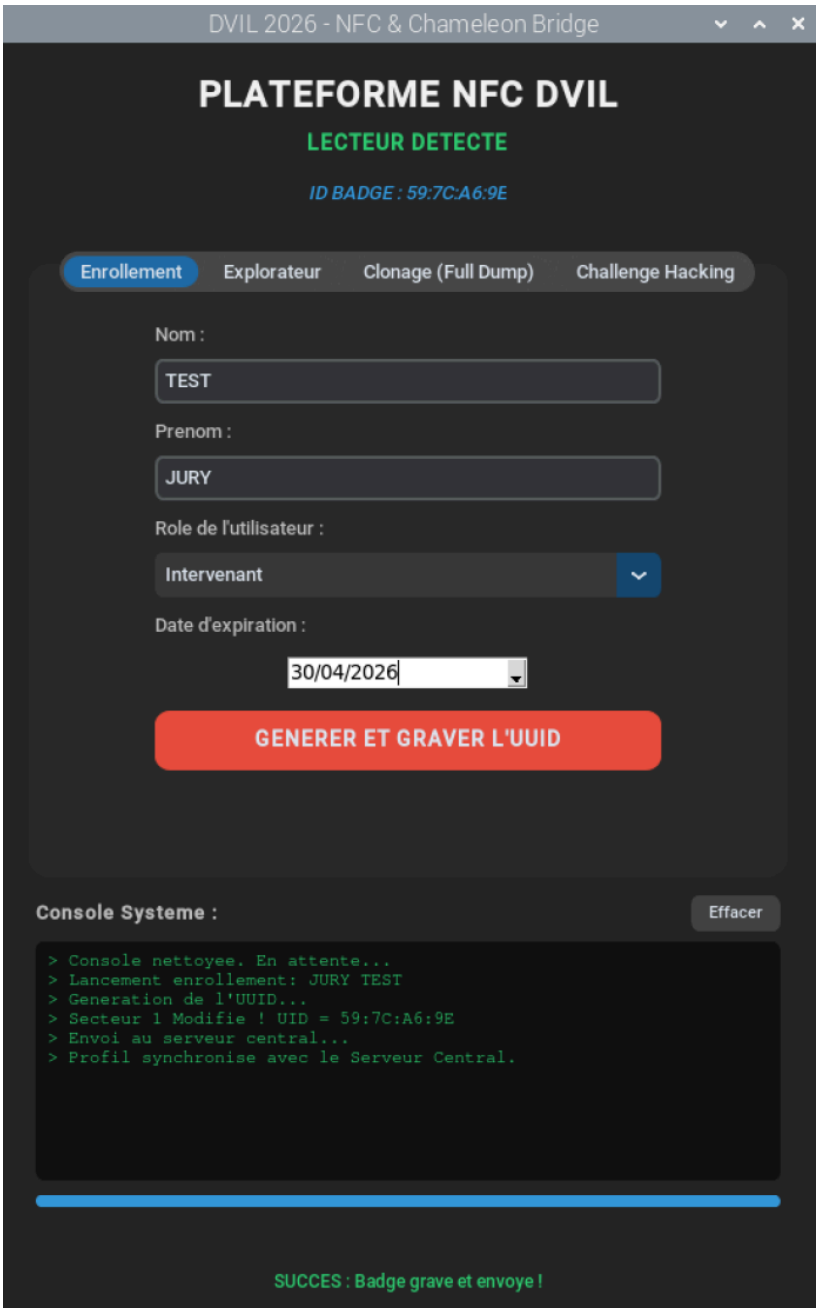
Sur le plan matériel, ce poste repose sur une Raspberry Pi associée à un lecteur RFID/NFC PN532, connecté via le **bus SPI** afin de garantir des performances optimales et une latence minimale.

Sur le plan logiciel, une application de bureau a été développée en Python selon une architecture modulaire. L’interface graphique, réalisée avec la bibliothèque **CustomTkinter**, offre un environnement ergonomique et moderne facilitant les opérations d’administration.

Les principales fonctionnalités du poste d’accueil sont les suivantes :

- **Enrôlement des utilisateurs** : création de profils, génération d’identifiants uniques et écriture sécurisée sur les badges.
- **Audit mémoire (mode explorateur)** : analyse de la structure interne des badges afin de vérifier la bonne configuration des secteurs et la validité des données enregistrées.
- **Gestion asynchrone des tâches** : utilisation du **multithreading** pour éviter le blocage de l’interface (IHM) lors des opérations NFC de lecture ou d'écriture.

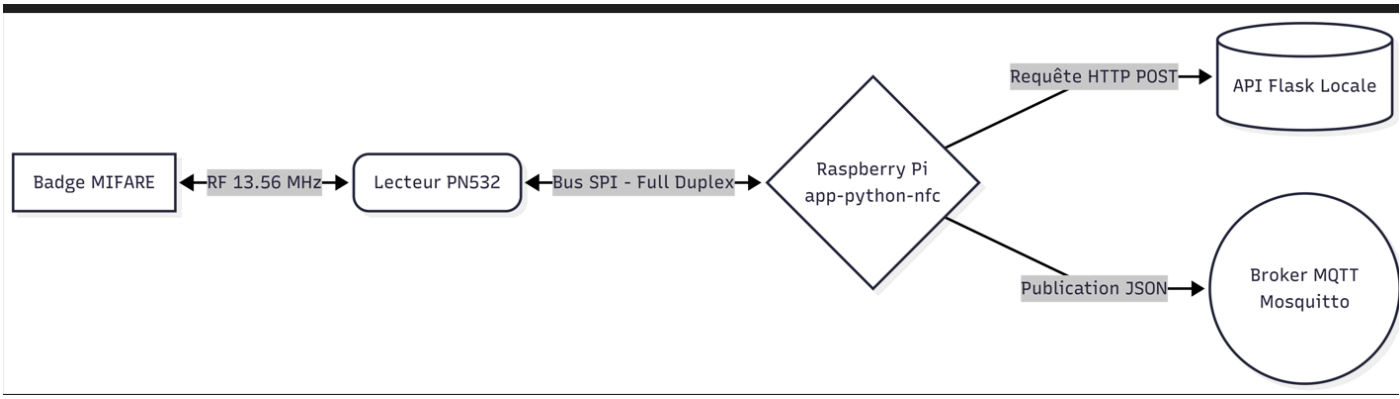
Interface d'administration (Custom Tkinter) — Succès de la gravure de l'UUID et synchronisation avec l'API



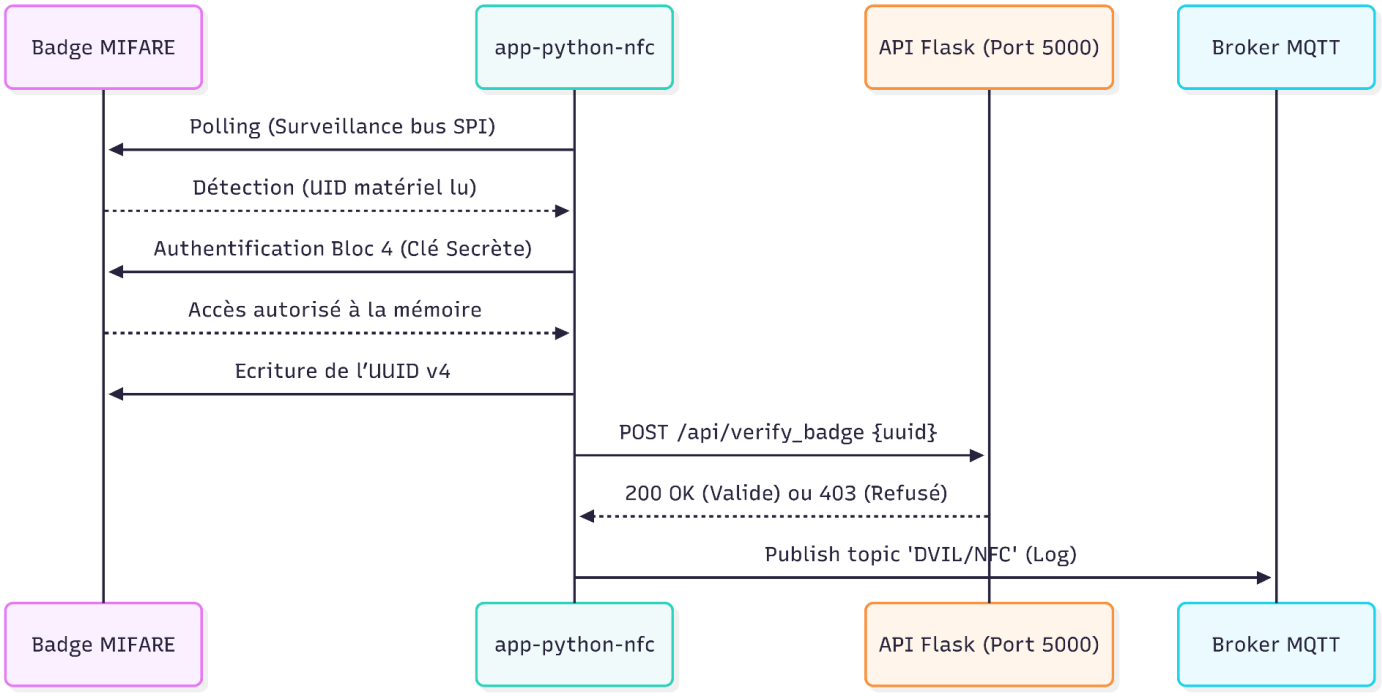
16.1.2 Développement des services IoT

Service de détection et lecture NFC (`app-python-nfc`)

Rôle technique et Hardware Ce service logiciel, fonctionnant en tâche de fond, assure la surveillance asynchrone du lecteur matériel PN532. La communication est établie via le bus SPI matériel de la carte Raspberry Pi (utilisant spécifiquement les broches physiques 19/MOSI, 21/MISO, 23/SCLK, et 24/CE0). Le choix du bus SPI garantit une communication Full-Duplex, minimisant la latence lors de la détection continue du badge.



Algorithme de sécurité et logique



Service de pilotage de l'actionneur (`app-python-serrure`)

Rôle technique et sécurité Ce script gère uniquement l'ouverture de la porte et écoute les ordres sur le réseau (abonnement au topic MQTT `DVIL/Serrure`). Un choix de sécurité important a été fait : le lecteur NFC et la serrure sont deux programmes totalement séparés qui ne se parlent jamais directement. Ainsi, même si le boîtier NFC est vandalisé ou piraté à l'extérieur, il est physiquement impossible de forcer l'ouverture de la porte depuis ce module.

Commande matérielle (GPIO) Le pilotage de la gâche électrique est géré par la bibliothèque `RPi.GPIO` sur la broche 17. Par défaut, la broche est maintenue à l'état BAS, ce qui maintient la porte verrouillée (principe de sécurité "Fail-Secure"). Quand le script reçoit un message valide sur MQTT, il passe le GPIO 17 à l'état HAUT pour activer le relais et ouvrir la porte. Il repasse ensuite automatiquement à l'état BAS après quelques secondes pour la reverrouiller.

Remontée d'information À chaque fois que la serrure s'ouvre ou se ferme, le script envoie une requête à l'API. Cela permet à la machine virtuelle de supervision d'être notifiée immédiatement et d'afficher l'état exact de la porte en temps réel.

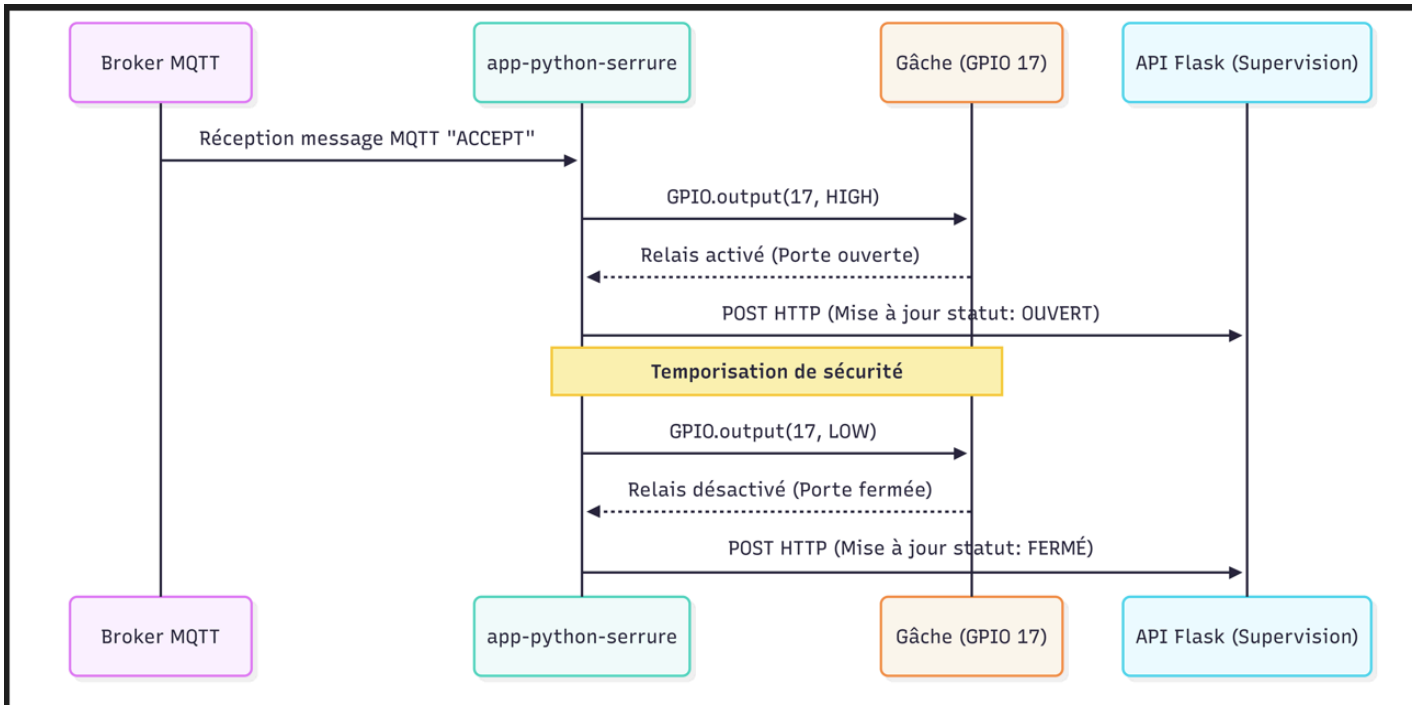


Diagramme de classes de l'application du poste d'accueil

L'application du poste d'accueil repose sur une séparation des responsabilités inspirée du modèle **MVC** (Modèle – Vue – Contrôleur) :

- **Classe AppEnrollement (Vue/Contrôleur)** : Gère l'interface graphique, les interactions utilisateur et le déclenchement des tâches asynchrones via des threads.
- **Classe GestionnaireNFC (Modèle)** : Encapsule la communication avec le module PN532 via la bibliothèque `adafruit_pn532` et gère les opérations de bas niveau (authentification, lecture, écriture).

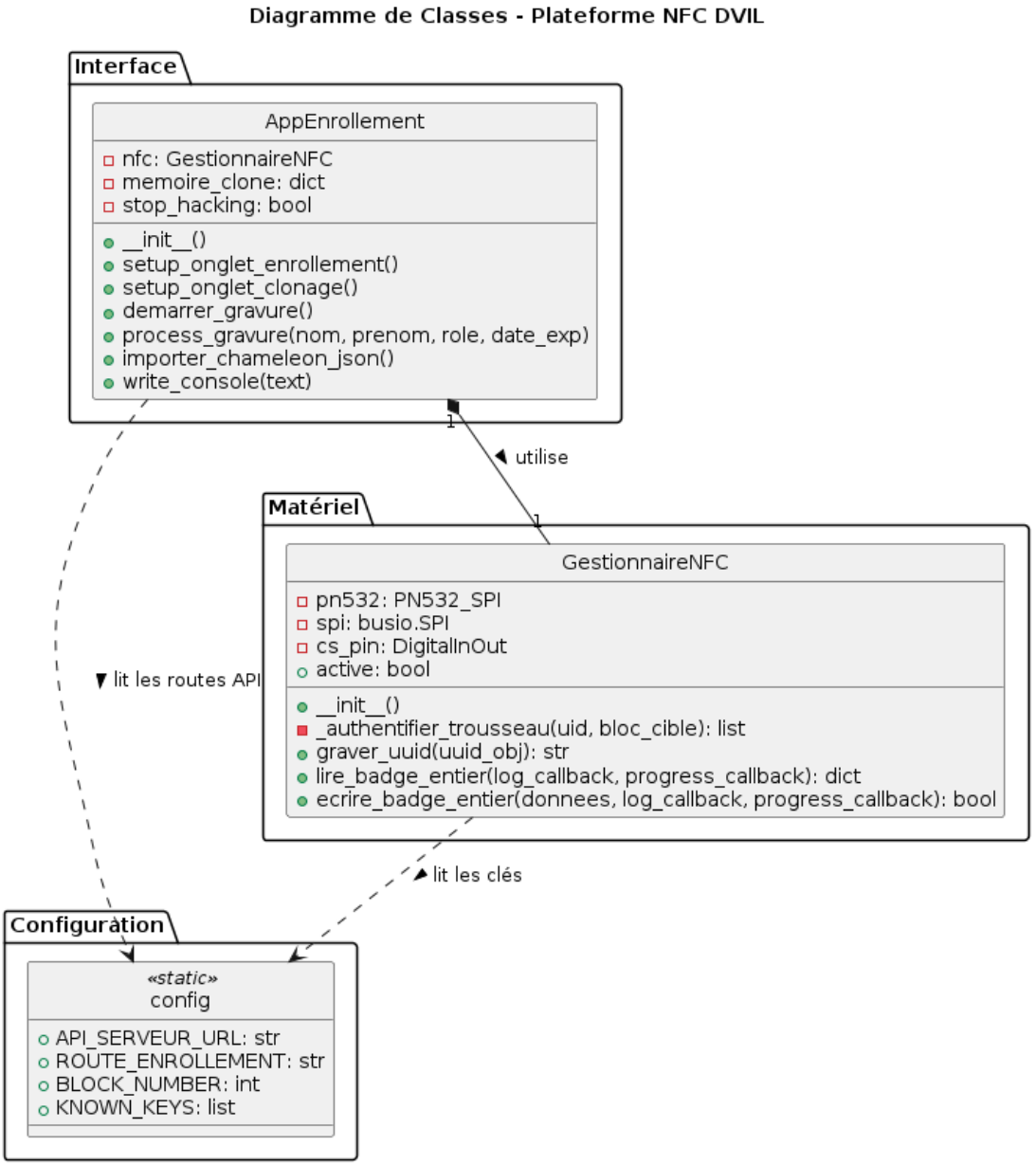
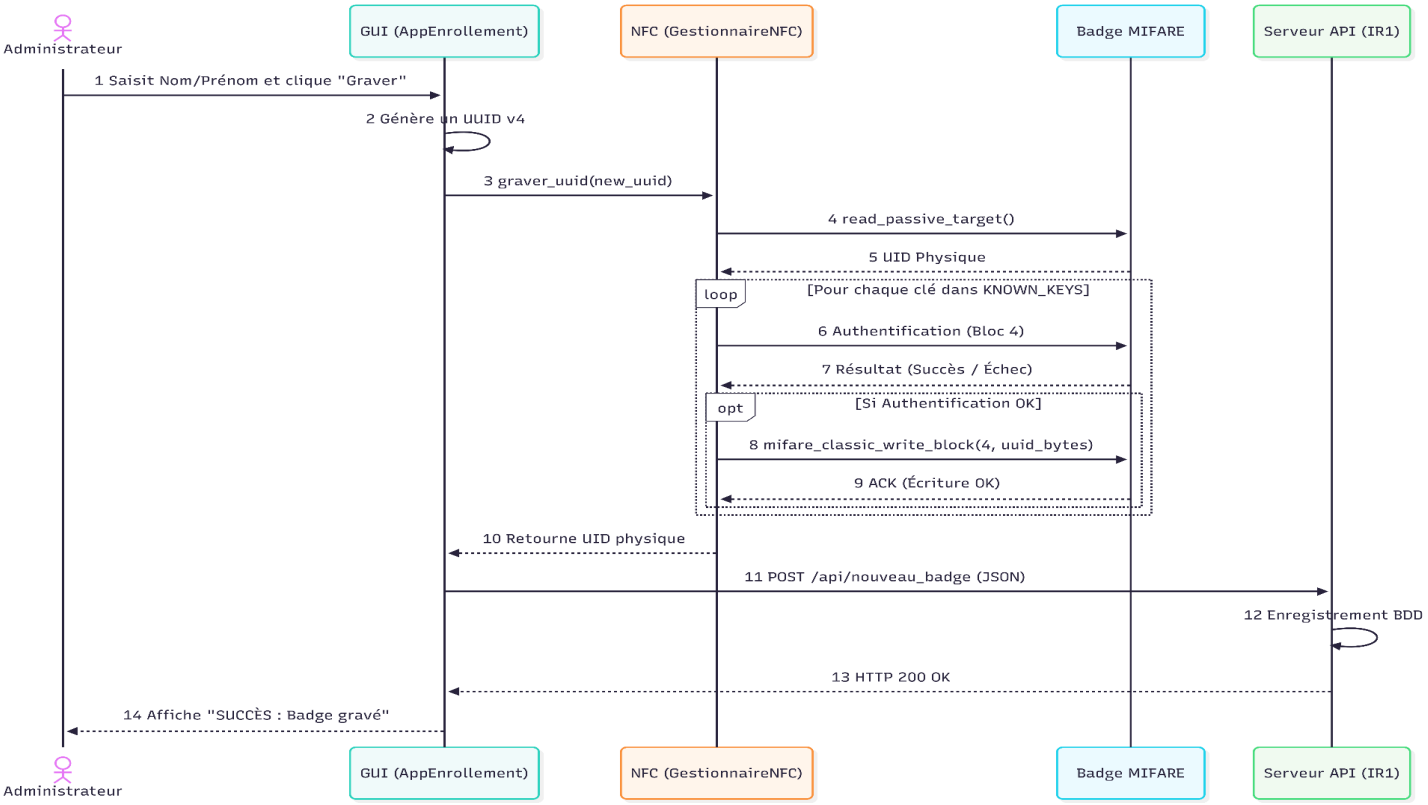


Diagramme de séquence (Processus d'enrôlement)

Le processus d'enrôlement d'un badge suit un flux logique strict :

1. **Saisie** des informations utilisateur via l'interface graphique.
2. **Détection** du badge et récupération de l'identifiant matériel unique (**UID**).
3. **Authentification** sur le secteur sécurisé du badge à l'aide d'une clé cryptographique.
4. **Écriture** de l'identifiant logique universel (**UUID**).
5. **Synchronisation** : Envoi des données vers l'API de supervision via une requête HTTP POST.



Principe de fonctionnement du NFC

Le système utilise la technologie **NFC** (*Near Field Communication*), fonctionnant à une fréquence de **13,56 MHz**. La communication repose sur le principe du **couplage inductif** :

- Le lecteur PN532 génère un champ magnétique.
- Le badge, passif, récupère cette énergie par induction pour s'alimenter.
- Les données sont transmises via une **modulation de charge**, permettant au badge de perturber le champ magnétique pour envoyer des informations au lecteur.
- **Protocole et mécanisme d'anti-collision** Dans un environnement où plusieurs badges pourraient être présentés simultanément, la norme impose un protocole d'anti-collision. Le lecteur envoie une requête de type REQA. Si plusieurs badges répondent en même temps, une collision se produit sur le bus de données. Le lecteur

utilise alors une boucle d'anti-collision (commandes `ANTICOLLISION` et `SELECT`) : il interroge les badges bit par bit sur leur numéro d'identification (UID) jusqu'à isoler un badge unique avec lequel il établira une communication exclusive.

5 - Choix du bus SPI

Le bus **SPI** (*Serial Peripheral Interface*) a été privilégié pour la communication entre la Raspberry Pi et le module PN532 :

- Un débit élevé (plusieurs Mbps) et une communication **Full-Duplex**.
- Une latence extrêmement faible, essentielle pour une détection fluide. Contrairement au bus I2C, le SPI ne nécessite pas de gestion d'adressage complexe sur le bus, ce qui améliore la rapidité des échanges lors des phases d'authentification.

6 - Sécurisation des badges RFID

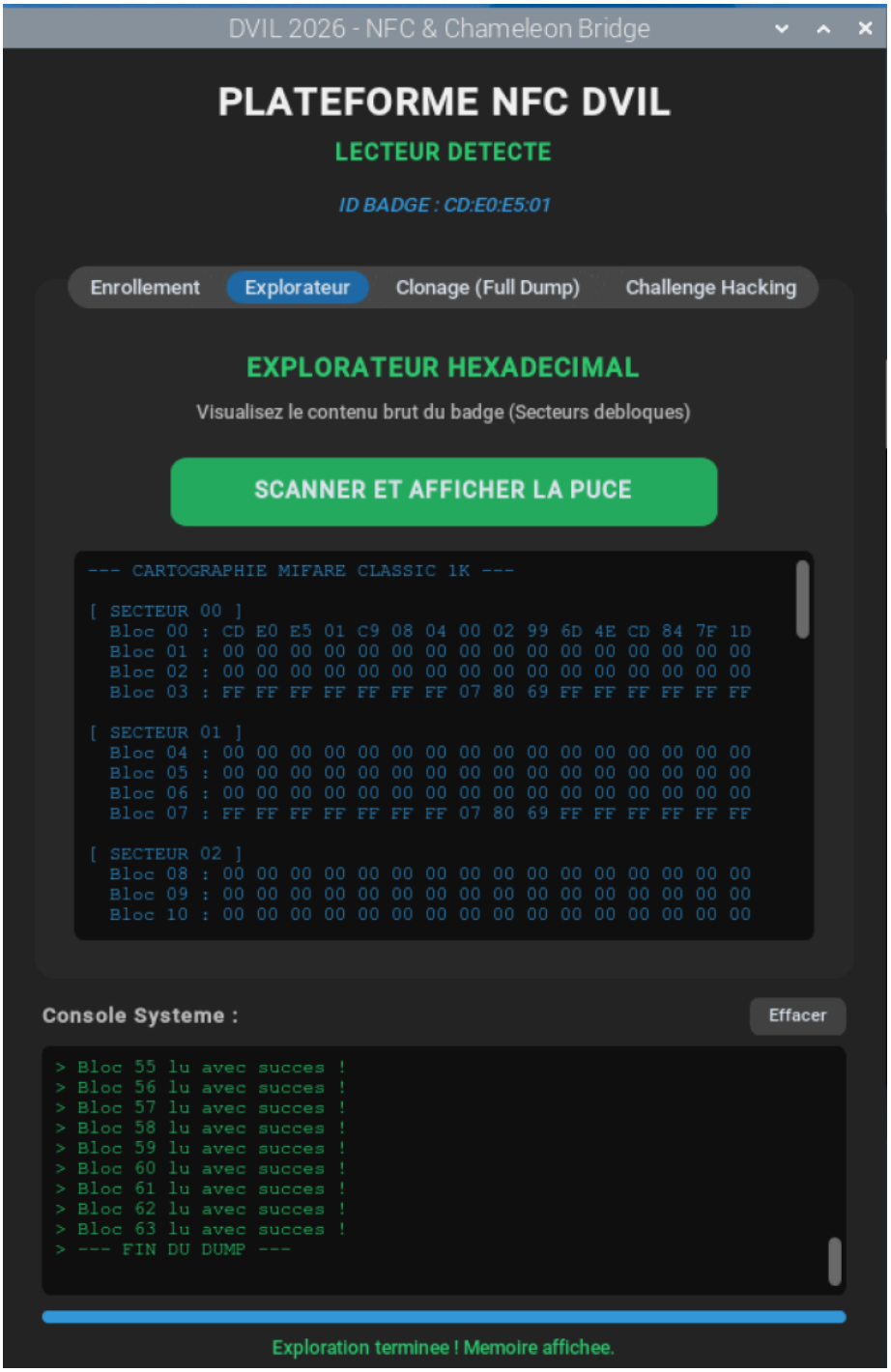
Structure mémoire des badges MIFARE Classic

<table>
<tr><td colspan="16">Secteur 0</td></tr>
<tr><td>Bloc 0</td><td colspan="5">UID</td><td colspan="10">Manufacturer Data</td></tr>
<tr><td>Bloc 1</td><td>1 octet</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Bloc 2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Bloc 3</td><td colspan="5">KEY A</td><td colspan="4">ACCESS</td><td colspan="6">KEY B</td></tr>
<tr><td colspan="16">Secteur 1</td></tr>
<tr><td>Bloc 4</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Bloc 5</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Bloc 6</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Bloc 7</td><td colspan="5">KEY A</td><td colspan="4">ACCESS</td><td colspan="6">KEY B</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

Les badges MIFARE Classic 1K disposent d'une mémoire de 1024 octets organisée en **16 secteurs** de **4 blocs** chacun (16 octets par bloc). Chaque secteur contient :

- Des blocs de données utilisateur.
- Un bloc de sécurité (**Sector Trailer**) comprenant la **Clé A**, les **Bits d'accès** et la **Clé B**. L'accès aux données nécessite une authentification préalable via l'algorithme de chiffrement **Crypto1**.
- L'**UID** (identifiant physique du badge) est transmis en clair lors de l'établissement de la communication NFC. Cette caractéristique constitue une faille de sécurité majeure, car l'UID peut être facilement cloné sur des badges reprogrammables, rendant l'usurpation d'identité possible.

Module Explorateur — Cartographie hexadécimale des 16 secteurs de la puce MIFARE 1K



Solution : Authentification renforcée(UUID)

Pour renforcer la sécurité, nous avons implémenté une solution basée sur une double vérification :

1. **Génération** d'un **UUID v4** (128 bits) aléatoire pour chaque utilisateur lors de l'enrôlement.
2. **Authentification** sur un secteur sécurisé du badge avec une clé personnalisée différente de la clé d'usine.
3. **Stockage** de cet UUID dans un bloc protégé de la mémoire (Bloc 4). Ainsi, même en cas de clonage de l'UID physique, l'accès sera refusé si l'UUID sécurisé n'est pas détecté par le lecteur de la serrure.

```python
import uuid

def enrollement_nouveau_badge(pn532, uid):
    # 1. Génération de l'UUID v4 (128 bits d'entropie)
    nouvel_uuid = uuid.uuid4()
    print(f"UUID généré pour l'utilisateur : {nouvel_uuid}")

    # 2. Conversion de l'UUID en tableau de 16 octets (bytes)
    # C'est le format brut requis pour l'écriture dans un bloc MIFARE
    uuid_bytes = nouvel_uuid.bytes

    # 3. Écriture dans le bloc protégé (Bloc 4)
    # (L'authentification préalable avec la clé secrète doit être réussie)
    bloc_cible = 4
    try:
        pn532.mifare_classic_write_block(bloc_cible, uuid_bytes)
        print(f"Succès : UUID écrit avec succès dans le Bloc {bloc_cible}")
        return str(nouvel_uuid)
    except Exception as e:
        print(f"Erreur lors de l'écriture sur le badge : {e}")
        return None
```

Qu'est-ce qu'un UUID v4 ? Un UUID (*Universally Unique Identifier*) est un identifiant standard de 128 bits utilisé pour identifier de façon quasi unique des éléments, sans avoir besoin d'un serveur central. Il est généralement représenté sous forme de chaîne de caractères, par exemple : `550e8400-e29b-41d4-a716-446655440000`.

Pour la sécurité du projet DVIL, nous utilisons spécifiquement la **version 4 (UUID v4)**. Celle-ci est entièrement générée de manière aléatoire (ou pseudo-aléatoire). Concrètement :

- Elle ne contient aucune information sur le temps, la machine ou l'adresse réseau.
- La probabilité de collision (générer deux UUID identiques) est extrêmement faible en pratique.

. *(Une explication plus détaillée de la répartition des bits de l'UUID est disponible en annexe).*

Communication avec l'API de supervision

Une fois le badge configuré, les informations sont transmises au serveur via une API REST sécurisée.

- **Méthode** : POST
- **Endpoint** : `https://<IP_SERVEUR>:8443/api/nouveau_badge`
- **Format** : JSON

Exemple de payload transmis :

JSON

```
{
 "nom": "Dupont",
 "prenom": "Jean",
 "role": "Eleve",
 "expiration": "15/06/2026",
 "uuid": "0e0716af-50d7-4050-95ca-28780a181b6d",
 "uid_physique": "B2:7E:41:3D"
}
```

Bilan technique

La mise en place du poste d'accueil NFC constitue une évolution majeure du projet DVIL. Elle permet :

- Une sécurisation avancée des accès par l'usage d'un identifiant logique chiffré (**UUID**).
- Une fiabilité accrue des communications matérielles via le bus **SPI**.
- Une interface administrateur ergonomique et performante grâce au **multithreading**.

Cette architecture répond aux problématiques de sécurité des systèmes RFID classiques et s'inscrit pleinement dans une démarche de **"Security by Design"**.

16.3 Analyse Technique (NFC, SPI, Mifare)

Cette section détaille les choix technologiques pour l'interface de commande physique et la gestion de la serrure.

Environnement de développement

- Python

Choix technologiques : Enrôlement NFC et Interface d'Accueil (IR2)

Le protocole de communication (SPI vs USB) : Pour le pilotage du lecteur RFID PN532, le choix s'est porté sur le bus **SPI** natif de la Raspberry Pi (Pins 19, 21, 23, 24) plutôt que sur une connexion série/USB. Le bus SPI offre une latence extrêmement faible, essentielle pour une détection fluide des badges, et élimine les problèmes de déconnexion inopinée fréquents sur les ports USB virtuels. L'intégration logicielle a été réalisée via la bibliothèque `adafruit_pn532`.

Sécurisation de la mémoire du badge (MIFARE Classic EV1) : Plutôt que d'utiliser l'UID d'usine (qui peut être cloné facilement), le système génère un **UUID de 16 octets** (Universal Unique Identifier) de manière logicielle. Cet UUID est gravé physiquement dans le **Bloc 4 (Secteur 1)** du badge. Ce secteur est protégé par une clé secrète personnalisée (`0xD3, 0xF7...`), nécessitant une authentification cryptographique explicite avant toute lecture ou écriture, empêchant ainsi l'usurpation.

Développement de l'Interface Graphique (IHM) : L'outil d'accueil a été développé en Python à l'aide de la bibliothèque **CustomTkinter**, permettant d'offrir un "Dark Mode" moderne et professionnel.

- **Multithreading :** Pour éviter que l'interface ne fige lors de l'attente du badge physique, les fonctions de scan et de gravure ont été isolées dans des "Threads" (tâches asynchrones).
- **Interopérabilité :** Une fois le badge gravé avec succès, l'application consolide les données (Nom, Prénom, Rôle, Date d'expiration, UUID et UID physique) et les expédie à l'API distante (Flask) via une requête HTTP POST au format JSON, assurant le lien entre la couche physique (NFC) et la base de données centrale.

16.4 Journal des Incidents et Résolutions (Focus IR2)

Incidents et Résolutions sur l'Interface d'Accueil NFC (IR2)

Incident	Cause Technique	Solution Apportée
Erreur de communication PN532 (ModuleNotFoundError / No module named 'busio')	Absence de la couche d'abstraction matérielle Blinka requise par les scripts CircuitPython sur Raspberry Pi OS (PEP 668).	Installation forcée de l'environnement matériel adafruit-blinka et configuration des Switchs DIP du module en mode SPI (SET0: 0, SET1: 1).
Erreur SPI (RuntimeError: Response frame preamble does not contain 0x00FF!)	Collision sur le bus SPI. L'interface graphique tentait de lire (scan boucle) et d'écrire (gravure) simultanément sur le même bus de données.	Refonte du code pour ignorer silencieusement les exceptions de lecture (try...except) pendant que le thread de gravure monopolise le bus SPI.
Crash IHM (SyntaxError: 'utf-8' codec can't decode byte)	Conflit d'encodage des caractères accentués (é, à, ô) entre l'éditeur de code et le terminal d'exécution Linux de la Raspberry Pi.	Modernisation du code avec l'en-tête # -*- coding: utf-8 -*- et suppression des caractères spéciaux critiques dans les labels pour garantir la portabilité du script.

5. Journal des Incidents et Résolutions

Cette section documente les difficultés techniques rencontrées lors du développement du poste d'enrôlement et les solutions apportées pour garantir la fiabilité du système.

Incident : Échec du clonage sur carte "Magic" CUID

- **Description du problème :** Lors de l'utilisation de la fonction "Injection" (clonage intégral), le processus s'arrêtait après l'écriture du Bloc 0. Les autres blocs (contenant les clés et les données) restaient inchangés, rendant le clone inopérant sur la serrure.
- **Analyse Technique :** Les cartes CUID (Generation 2) changent instantanément d'identité dès que le Bloc 0 (UID) est modifié. Le lecteur PN532 perd alors l'authentification en cours car l'UID ne correspond plus à celui utilisé pour ouvrir la session sécurisée.
- **Résolution :** J'ai modifié l'algorithme d'écriture dans `nfc_core.py`. Désormais, le script identifie le Bloc 0 et le déplace à la toute fin de la file d'attente d'écriture. Ainsi, toutes les données et les clés secrètes sont gravées pendant que la carte a encore son ancienne identité, et la mutation de l'UID se fait à la dernière milliseconde.

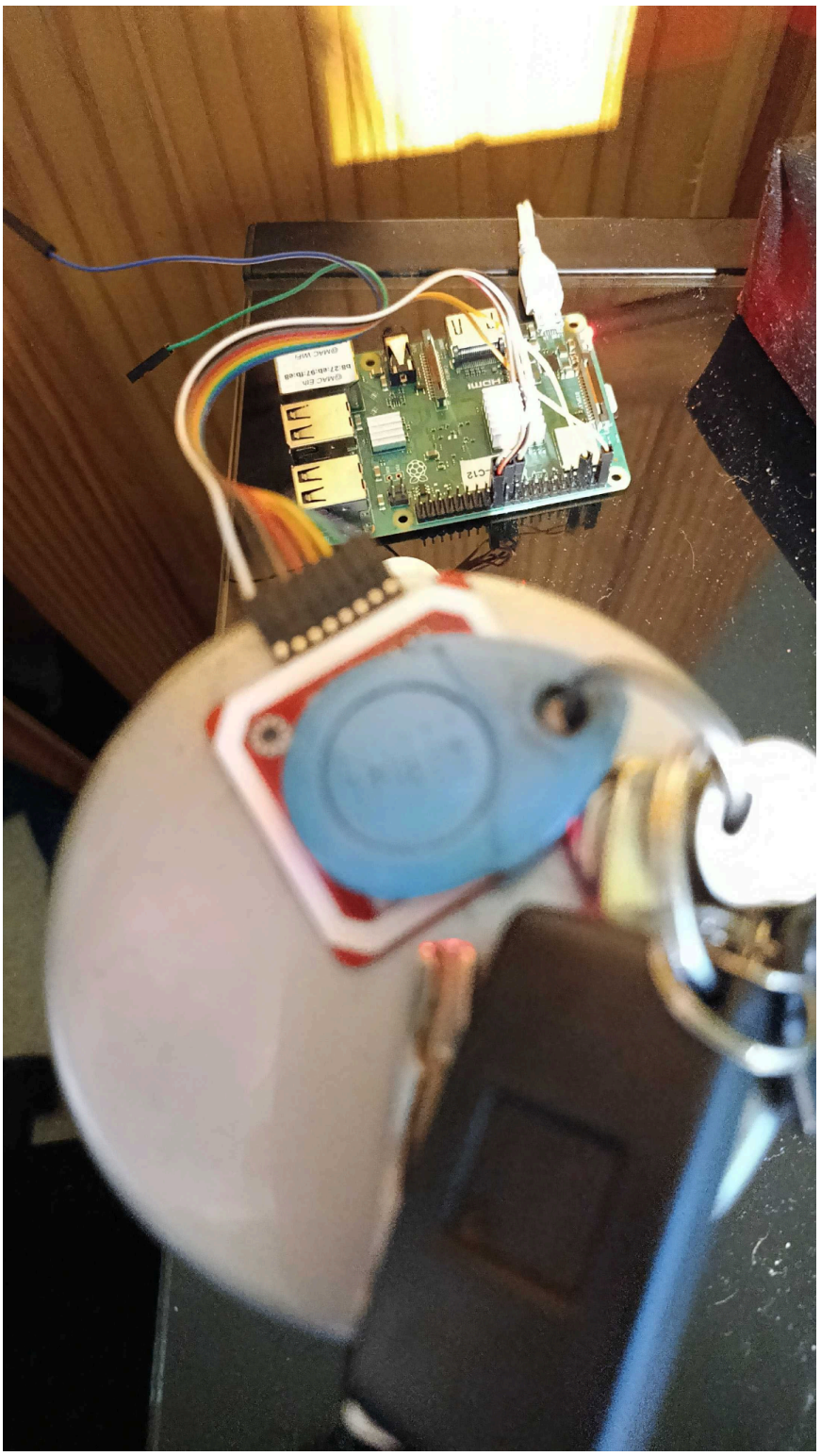
Incident : Crash de l'IHM lors de l'affichage des logs

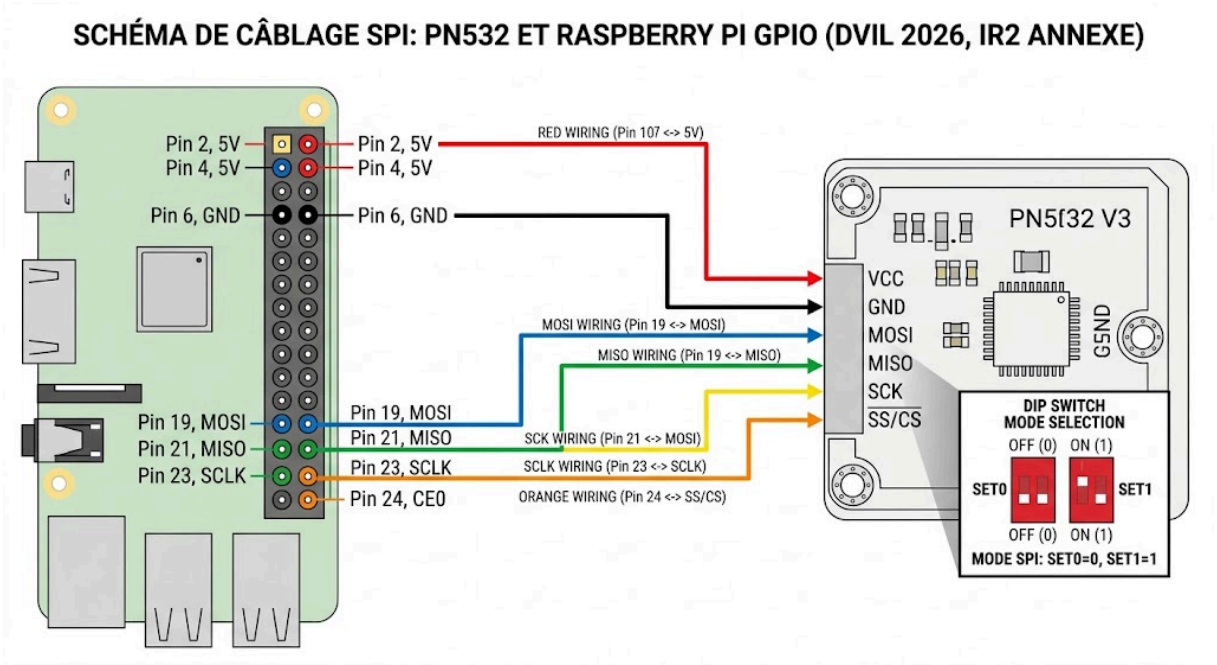
- **Description du problème :** L'interface graphique `CustomTkinter` plantait systématiquement avec une erreur de type `UnicodeDecodeError` ou `SyntaxError` dès qu'un caractère accentué (é, à, è) était envoyé à la console de debug.
- **Analyse Technique :** Par défaut, l'interpréteur Python sur Raspberry Pi OS peut rencontrer des conflits d'encodage entre les chaînes de caractères de l'IHM et les flux de données provenant du matériel.
- **Résolution :** J'ai forcé l'encodage en ajoutant la directive `# -*- coding: utf-8 -*-` au sommet de tous les scripts. De plus, j'ai normalisé les messages envoyés à la fonction `write_console()` pour supprimer les caractères spéciaux critiques, garantissant ainsi la portabilité du code sur différents systèmes Linux.

Incident : Collision sur le bus SPI et "Freeze" de l'interface

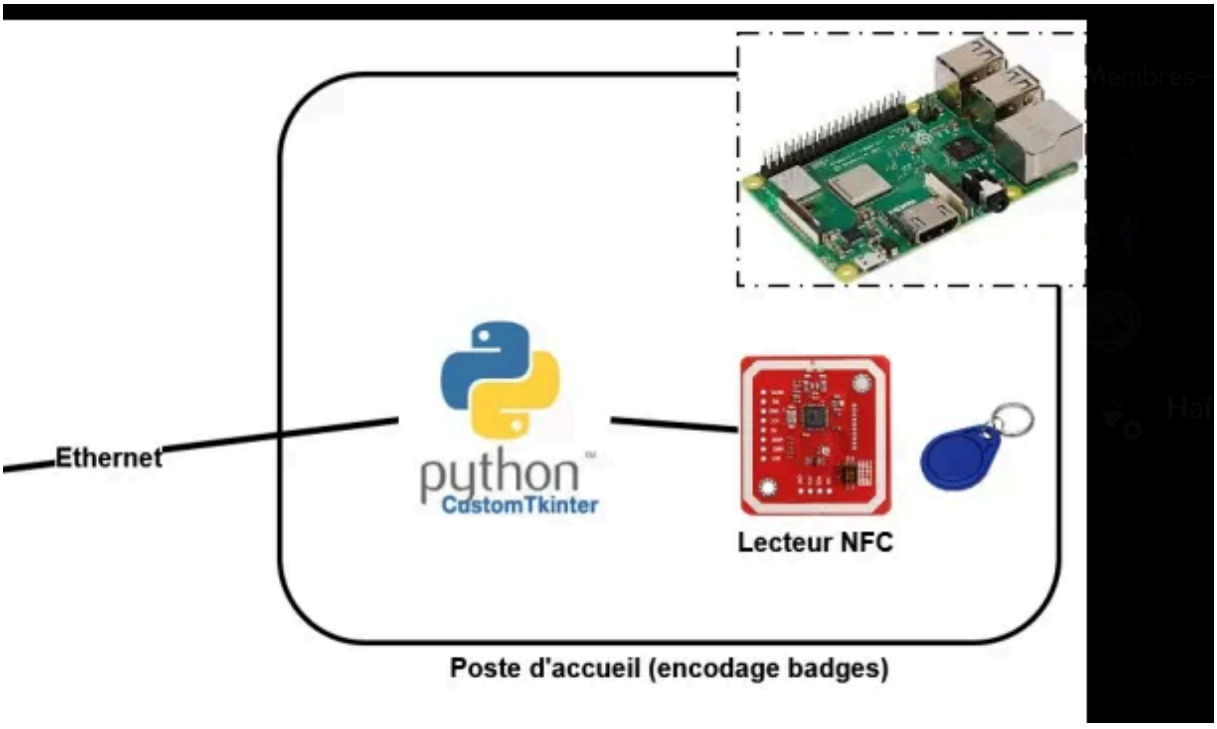
- **Description du problème :** Lorsque l'utilisateur lançait un scan du badge tout en essayant de cliquer sur un autre onglet, l'application se figeait totalement ("Not Responding").
- **Analyse Technique :** Le bus SPI est synchrone. Si le processeur est occupé à attendre une réponse du PN532 (qui peut être longue si le badge est mal placé), il ne peut plus traiter les rafraîchissements de l'interface graphique.
- **Résolution :** J'ai implémenté le **Multithreading** via la bibliothèque `threading`. Toutes les fonctions critiques du `GestionnaireNFC` (lecture, gravure, hacking) sont désormais exécutées dans des threads séparés du thread principal (Main Loop). J'utilise des "callbacks" pour mettre à jour l'interface sans jamais bloquer l'interaction utilisateur.

Banc de test Interfaçage du lecteur PN532 avec la Raspberry Pi via les lignes MOSI/MISO/SCK du bus SPI

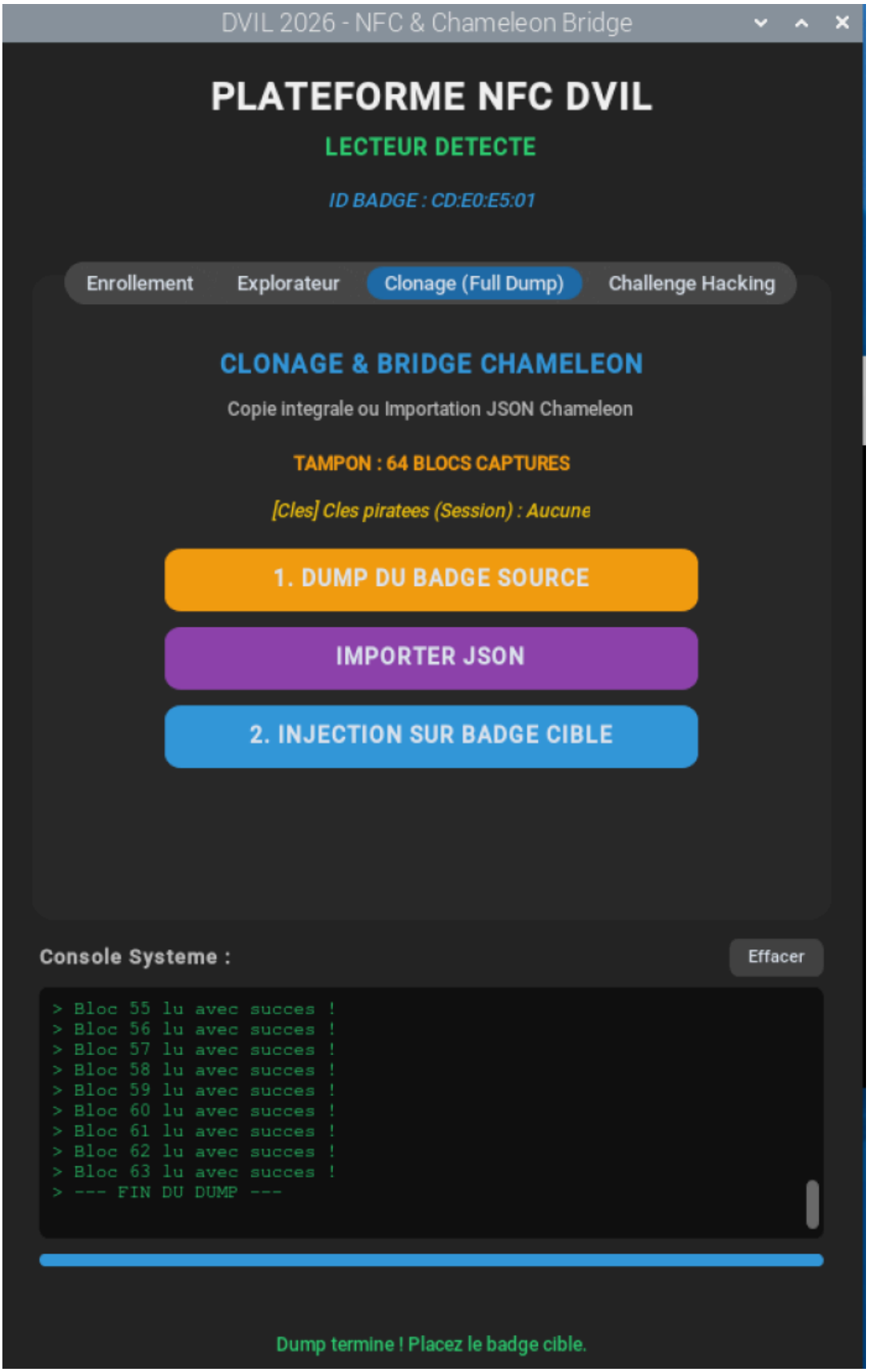




POST ACCUEIL



Interface de clonage intégral — Validation de l'algorithme d'injection de données sur badge CUID



16.5 PARTIE PHYSIQUE : Comparaison RFID et NFC

Dans le cadre du projet DVIL, le choix de la technologie sans contact est crucial pour garantir la sécurité de l'accès physique. Il est important de distinguer le **NFC** (Near Field Communication) utilisé ici de la **RFID** (Radio Frequency Identification) classique.

Positionnement technologique

La **RFID** est un terme générique désignant l'identification par radiofréquence. Elle regroupe plusieurs familles de fréquences (125 kHz, 13,56 MHz, 860 MHz). Le **NFC** est une évolution spécifique de la RFID haute fréquence, opérant exclusivement à **13,56 MHz**. Contrairement à la RFID classique qui peut fonctionner à plusieurs mètres, le NFC est limité à une distance de moins de 10 cm, ce qui constitue une première barrière de sécurité physique contre l'interception de données à distance.

Principe physique : Le couplage inductif

Le fonctionnement repose sur le principe du **couplage inductif** entre deux antennes : celle du lecteur PN532 et celle du badge passif.

1. **Génération du champ** : Le lecteur PN532 émet un champ magnétique alternatif à 13,56 MHz.
2. **Télé-alimentation** : Lorsque le badge entre dans ce champ, l'énergie magnétique est convertie en courant électrique par induction, permettant d'alimenter la puce du badge.
3. **Modulation de charge** : Pour répondre au lecteur, le badge ne génère pas sa propre onde radio. Il modifie la charge de sa propre antenne (en ouvrant ou fermant un circuit de commutation). Ces micro-variations perturbent le champ magnétique du lecteur, qui les traduit en données binaires (UID, UUID).

Justification du choix NFC pour DVIL

Le choix du NFC 13,56 MHz via le bus SPI offre des avantages déterminants par rapport à une solution RFID 125 kHz :

- **Communication bidirectionnelle** : Permet non seulement de lire un identifiant, mais aussi d'écrire des données sécurisées.
- **Sécurité cryptographique** : Le protocole Mifare Classic 1K utilisé permet une authentification par clé (Crypto1), indispensable pour protéger l'écriture de l'**UUID v4** que j'ai implémentée.
- **Débit et Latence** : Le couplage à 13,56 MHz permet des échanges de données rapides, garantissant un temps de réponse système inférieur à 1 seconde, conformément au cahier des charges.

Architecture physique et fonctionnement du NFC 13,56 MHz dans le projet DVIL

1) POSITIONNEMENT TECHNOLOGIQUE : RFID vs NFC

Le NFC (13,56 MHz) est une forme évoluée de la RFID Haute Fréquence.
Il se distingue par sa portée très courte et ses capacités de sécurité.

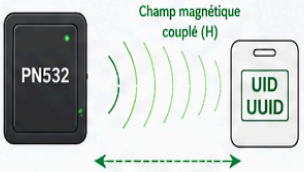
RFID CLASSIQUE (Basse Fréquence)
125 kHz



Portée typique
0.5 m à 1 m

⚠ Portée longue
Risque d'interception à distance

NFC (Haute Fréquence)
13,56 MHz



Portée sécurisée
< 10 cm

Portée très courte
Sécurité physique intégrée contre l'interception

COMPARAISON TECHNIQUE

Critère	RFID 125 kHz	NFC 13,56 MHz
Fréquence de travail	125 kHz	13,56 MHz
Portée typique	0.5 m à 1 m	< 10 cm
Couplage	Inductif (faible)	Inductif (fort)
Débit de données	Faible (~1 à 5 kbps)	Élevé (106 à 424 kbps)
Communication	Généralement unidirectionnelle (lecteur → tag)	Bidirectionnelle (lecteur ↔ tag)
Sécurité	Faible (UID lisible)	Élevée (authentification, chiffrement)
Écriture de données	Limitée ou inexistante	Oui (écriture sécurisée)
Exemple d'application	Contrôle d'accès basique	Contrôle d'accès sécurisé (DVIL) UUID v4, authentification
Norme / Protocole	ISO 11784/11785	ISO/IEC 14443 A – Mifare Classic
Sécurité des données	Aucune / Faible	Crypto1 (clé secrète) Chiffrement possible

★ Le NFC 13,56 MHz offre une portée réduite (< 10 cm) et des fonctions de sécurité avancées, idéales pour un contrôle d'accès fiable et protégé comme dans le projet DVIL.

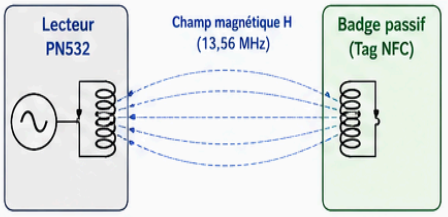
2) PRINCIPE PHYSIQUE DU SIGNAL : COUPLAGE INDUCTIF & MODULATION

La communication NFC repose sur le couplage inductif à 13,56 MHz entre le lecteur (PN532) *et le badge passif, suivant 3 étapes physiques clés.*

1 GÉNÉRATION DU CHAMP PAR LE LECTEUR

Le PN532 génère un courant alternatif à 13,56 MHz dans sa bobine (antenne).

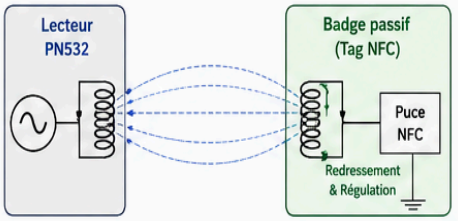
Cela crée un champ magnétique alternatif H qui se propage dans l'espace.



2 TÉLÉ-ALIMENTATION (INDUCTION)

Lorsque le badge entre dans le champ H, le flux magnétique traverse sa bobine.
Par induction électromagnétique, une tension est générée aux bornes de l'antenne du badge.

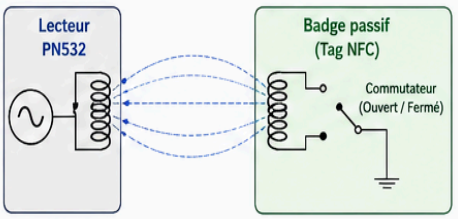
Cette énergie est redressée et régulée pour alimenter la puce du badge.



3 MODULATION DE CHARGE (RÉPONSE DU BADGE)

Pour répondre, le badge ne génère pas sa propre onde radio. Il modifie la charge de son antenne en ouvrant ou fermant un commutateur.
Ces variations de charge perturbent le champ H du lecteur.

Le PN532 détecte ces micro-variations de tension et les décode en données binaires (UID, UUID, etc.).

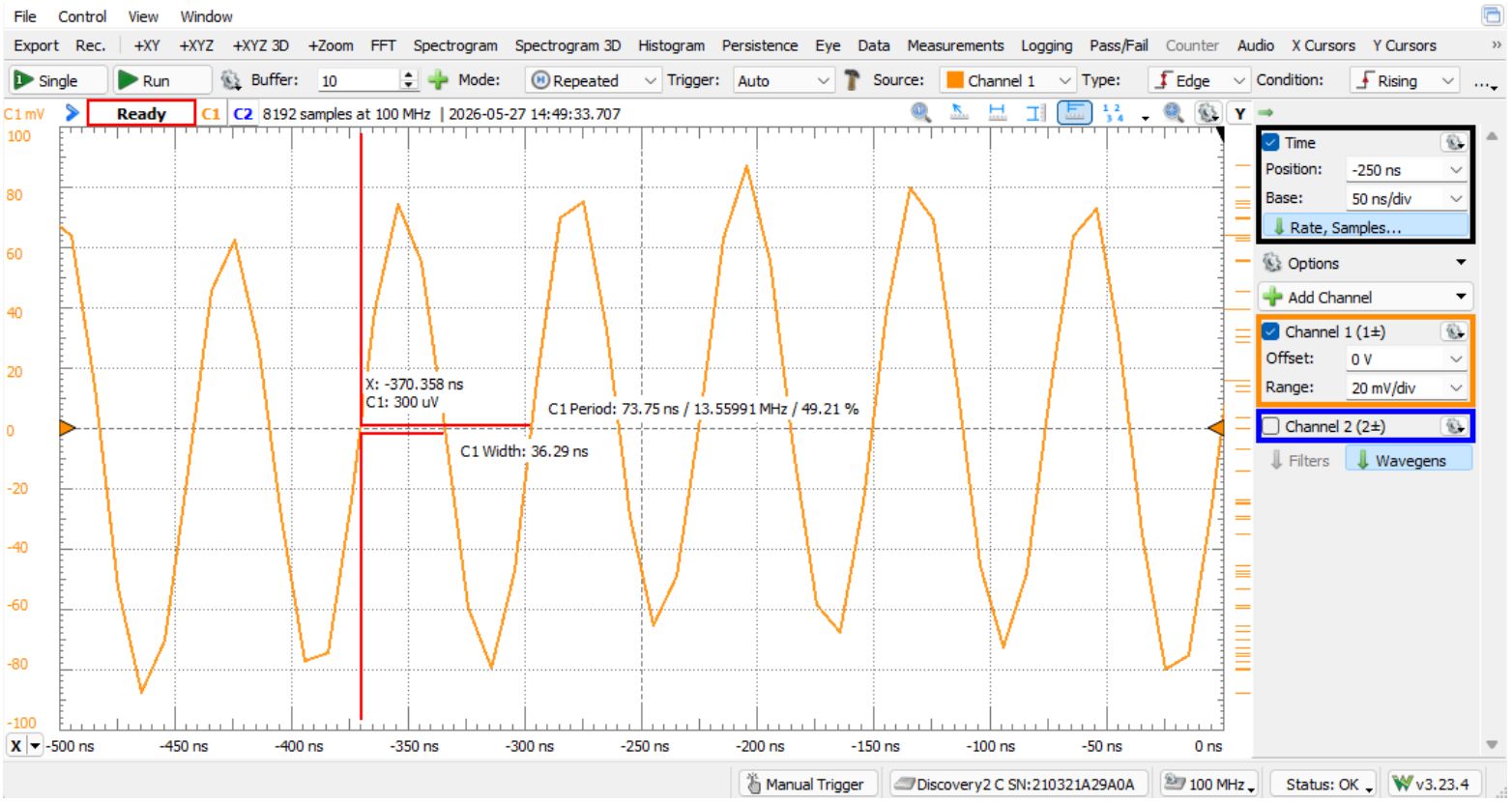




RÉSUMÉ DU PRINCIPE PHYSIQUE

Le NFC utilise le couplage inductif pour l'énergie et la donnée.
Le badge passif "parle" en modifiant sa charge, et le lecteur "écoute" ces perturbations pour lire les données.
Ce principe garantit une communication sécurisée à très courte portée (< 10 cm).


File Control View Window
Export Persistence Time Measurements Components X Cursors Y Cursors Markers Logging Notes
Single Run
Freq. Range: Auto Start: 7.7955415 MHz Stop: 19.75569775 MHz BINs: Default Algorithm: FFT
Scale: Linear Center: 13.77561965 MHz Span: 11.96015625 MHz Samples: Default Resolution: 6.10352 kHz
dBm Ready T1 T2 Scope: 100 MHz | 2026-05-27 14:40:14.072
Y1: -9.76247 dBm
Trace 1: C1 5.532 Vpk2pk
7.796 MHz 8.992 MHz 10.188 MHz 11.384 MHz 12.58 MHz 13.776 MHz 14.972 MHz 16.168 MHz 17.364 MHz 18.56 MHz
Magnitude
Units: dBm
Top: 10 dBm
Bottom: -120 dBm
Reference: 223.607 mV
Range: 5 V
Channel options
Add Trace
Trace 1
Channel: C1
Type: Sample
Count: 100
Trace 2
Manual Trigger
Discovery2 C SN:210321A29A0A
100 MHz
Status: OK
v3.23.4


File Control View Window
Export Rec. +XY +XYZ +XYZ 3D +Zoom FFT Spectrogram Spectrogram 3D Histogram Persistence Eye Data Measurements Logging Pass/Fail Counter Audio X Cursors Y Cursors
Single Run Buffer: 10 Mode: Repeated Trigger: Auto Source: Channel 1 Type: Edge Condition: Rising
C1 mV Ready C1 C2 8192 samples at 100 MHz | 2026-05-27 14:49:33.707
X: -370.358 ns
C1: 300 uV
C1 Period: 73.75 ns / 13.55991 MHz / 49.21 %
C1 Width: 36.29 ns
-500 ns -450 ns -400 ns -350 ns -300 ns -250 ns -200 ns -150 ns -100 ns -50 ns 0 ns
Time
Position: -250 ns
Base: 50 ns/div
Rate, Samples...
Options
Add Channel
Channel 1 (1±)
Offset: 0 V
Range: 20 mV/div
Channel 2 (2±)
Filters
Wavegens
Manual Trigger
Discovery2 C SN:210321A29A0A
100 MHz
Status: OK
v3.23.4

16.6 Conclusion

Ce projet DVIL conclut parfaitement mes deux années de BTS CIEL. J'ai particulièrement apprécié l'approche très "terrain" de ce travail, car j'ai pu toucher à tout : du câblage matériel (lecteur NFC, relais) jusqu'au développement Python sous Linux, en passant par le réseau avec MQTT.

Au-delà de faire fonctionner une simple porte, j'ai voulu pousser l'aspect sécurité beaucoup plus loin. La mise en place de l'anti-clonage (avec la gestion des UUID et le verrouillage matériel des secteurs) m'a donné un aperçu très concret de ce qu'est la cybersécurité matérielle aujourd'hui.

Bien sûr, assembler toutes ces technologies ne s'est pas fait sans mal. J'ai rencontré pas mal de bugs, notamment lors du déploiement des micro-services sur la Raspberry Pi ou avec la gestion du multithreading. Mais au final, c'est justement en cherchant l'origine de ces pannes et en les corrigeant que j'ai le plus appris.

Aujourd'hui, je suis très fière de livrer un prototype d'accès non seulement fonctionnel, mais surtout sécurisé. C’est une expérience vraiment motivante qui me donne une excellente base technique pour la poursuite de mes études.

Partie Personnelle
Abdelrahmen Amdouni
(ER)

17-Objectifs

- Prendre connaissance du dossier final de la version 2025 du projet DVIL.

-Analyser, et tester les cartes HAT et Clavier de cette version. Travail à effectuer en collaboration avec les étudiants IR.

- Modifier les cartes HAT et Clavier, afin qu'elles intègrent une connectique permettant l'interconnexion d'un afficheur OLED et d'un module nfc aussi bien sur l'une que sur l'autre (deux bus mis en oeuvre)

-Produire le schéma structurel de chacune des cartes. Effectuer un routage de ces cartes et produire les fichiers afin que la fabrication du PCB soit sous-traitée.

- Prévoir la mise en boîtier lors du routage des cartes aussi bien au niveau de la forme que de l'accès aux connectiques Câbler la carte et effectuer les essais.

18-Planification

-Pour la planification, j'ai utilisé un diagramme de Gantt afin de structurer les tâches définies. Cette méthode permet de réaliser les objectifs étape par étape, selon un calendrier précis.

Il est toutefois important de noter que cette planification reste une **estimation théorique**. L'avancement réel peut varier en fonction :

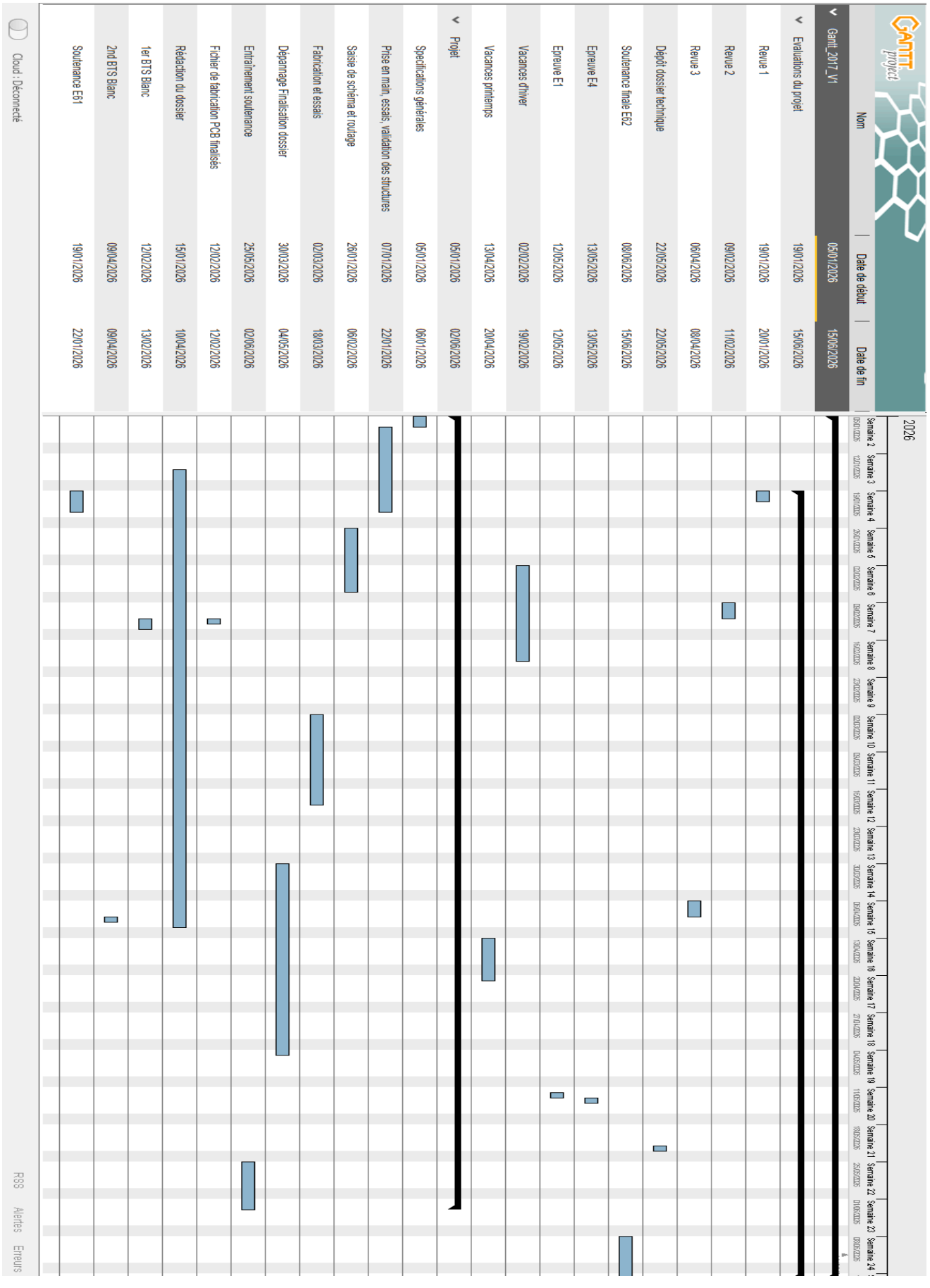
- Des imprévus techniques.
- Des difficultés rencontrées lors du déroulement.
- De l'ajustement des ressources nécessaires.

19-Matériel

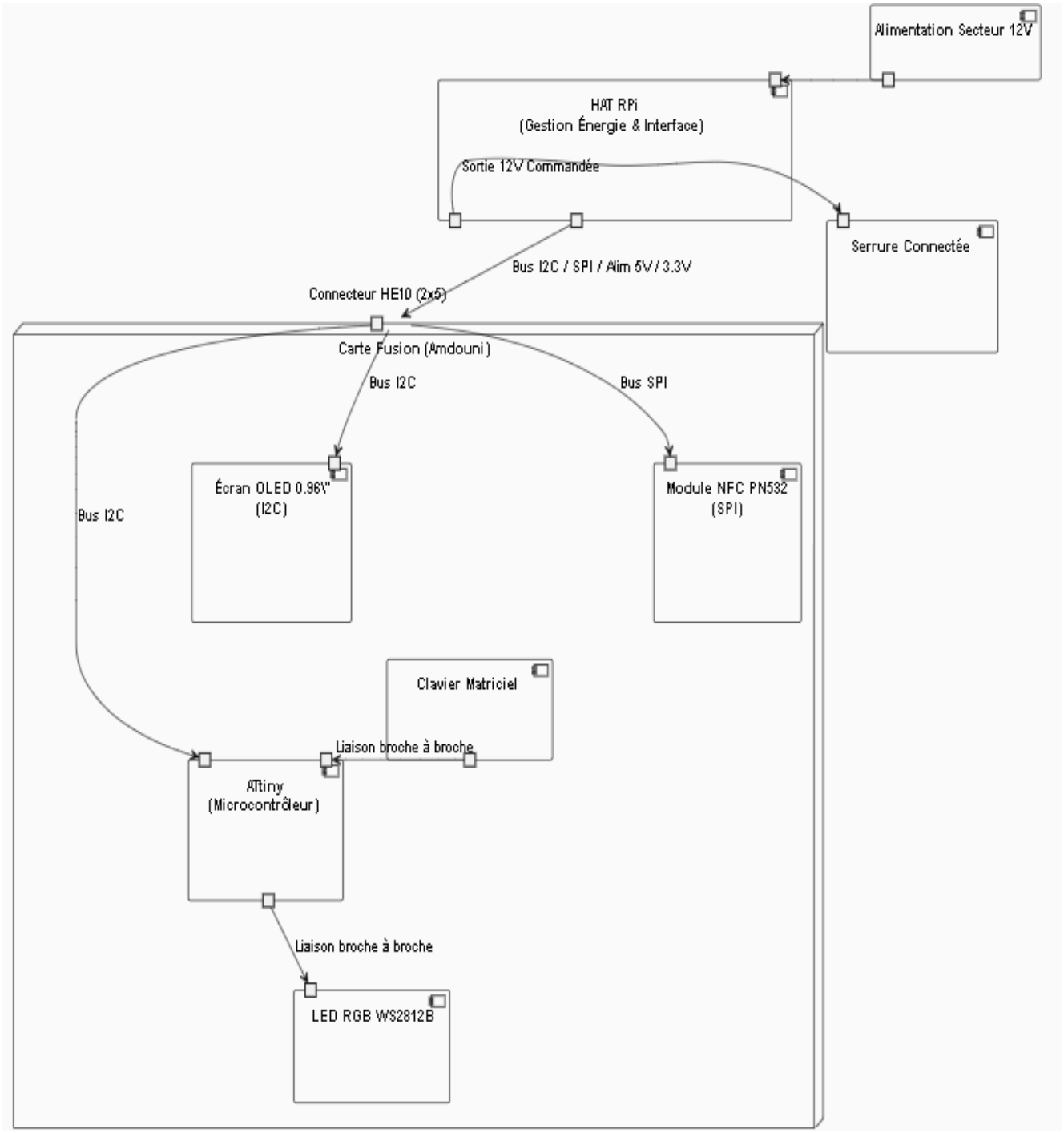
-Pour ce projet je dispose donc de ;

- Raspberry pi Modèle Zero
- ATtiny 1616
- Module NFC PN532
- Afficheur OLED 128x64
- LED adafruit programmable
- Serrure connectée

Gantt prévisionnel



20-Diagramme des blocs internes



21-Bus utilisés

21.1- Le bus I2C

-Le bus I2C (Inter-Integrated Circuit) est un bus de données utilisé principalement pour de la domotique ou de l'électronique domestique.

C'est un bus en série et asynchrone en half-duplex, utilisant deux lignes pour communiquer SDA (Serial Data Line) et SCL (Serial Clock Line), le tout avec une masse commune.

Ce bus utilise une relation de maître/esclave avec un maître (ou plusieurs mais rarement) et un ou tous les esclaves avec une initiative toujours pour le(s) maître(s).

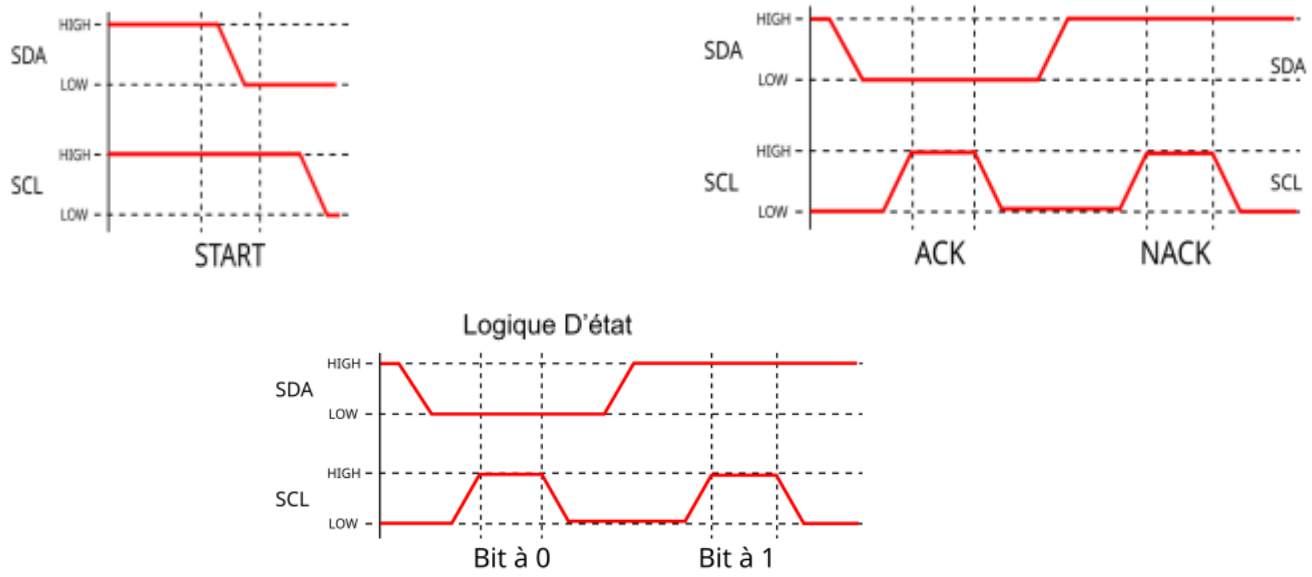
-Utilisation du bus I2C :

L'I2C présente l'avantage d'être un bus de communication série simple à implémenter, car il est nativement supporté par la quasi-totalité des microcontrôleurs du marché.

Cette universalité explique sa présence sur un très grand nombre de circuits intégrés (capteurs, mémoires, afficheurs). Il offre différentes vitesses de transmission, allant de 100 kbit/s en mode standard jusqu'à 5 Mbit/s en mode ultra-rapide.

21.2 Structure d'une trame I2C

Le bus I2C repose sur deux lignes (SDA et SCL) et fonctionne selon des règles de synchronisation précises :



- **Validation des données :** L'état de la ligne **SDA** doit impérativement rester stable pendant que l'horloge **SCL** est au niveau haut. Les changements d'état de **SDA** se font lorsque **SCL** est au niveau bas.
- **Conditions de START et STOP :** Ce sont les seules exceptions. Elles sont définies par une transition de **SDA** alors que **SCL** est maintenu au niveau haut :
    - **START :** Front descendant de SDA.
    - **STOP :** Front montant de SDA.

Gestion de l'acquittement (ACK/NACK)

Chaque octet transmis est suivi d'un neuvième bit d'acquittement pour valider la réception :

- **ACK (Acknowledge) :** Le récepteur maintient **SDA** au niveau bas. Cela confirme la bonne réception de l'octet.
- **NACK (Not-Acknowledge) :** Le récepteur laisse **SDA** au niveau haut. Cela indique soit une erreur, soit la fin de la communication (si le maître est en réception).

Adressage et Données

Adressage (7 bits + R/W) :

- Le maître initie la communication en envoyant l'adresse de l'esclave. Le 8e bit **(R/W)** définit le sens : **0** pour l'écriture (maître vers esclave) et **1** pour la lecture (esclave vers maître).

Données (8 bits) :

- Les informations circulent par paquets d'un octet. Pour des volumes plus importants, les données sont segmentées en plusieurs octets successifs.

Points clés retenus :

- L'adresse sur 7 bits.
- La fonction du bit R/W (0 = écriture, 1 = lecture).
- Le format standard de 8 bits par paquet.

21.3- Le bus SPI

Architecture du bus SPI

Le **SPI** (*Serial Peripheral Interface*) est un bus de communication série synchrone qui repose sur une architecture de type **Maître/Esclave**. Il utilise quatre lignes de signaux électriques pour transférer les données.

Les lignes de signaux

- **SCLK (Serial Clock) :** L'horloge générée par le maître qui cadence la transmission.
- **MOSI (Master Out Slave In) :** Ligne transportant les données du maître vers l'esclave.
- **MISO (Master In Slave Out) :** Ligne transportant les données de l'esclave vers le maître.
- **SS / CS (Slave Select / Chip Select) :** Ligne de sélection gérée par le maître pour activer un esclave spécifique (généralement active à l'état bas).

Mécanisme de transmission

Sélection de l'esclave

Pour initier une communication, le maître place la ligne **SS/CS** de l'esclave visé au niveau logique **0**. Tant que cette ligne reste à l'état bas, l'esclave écoute le bus et active sa sortie MISO. Les autres esclaves présents sur le bus, dont la ligne SS reste à l'état haut, maintiennent leur sortie MISO en haute impédance (isolée) pour ne pas perturber le signal.

Échange de données (Registre à décalage)

Le transfert fonctionne comme un double registre à décalage circulaire. À chaque impulsion de l'horloge SCLK :

1. Le maître décale un bit vers l'esclave via la ligne **MOSI**.
2. Simultanément, l'esclave décale un bit vers le maître via la ligne **MISO**.

Ce fonctionnement permet un échange de données en mode **Full-Duplex**, signifiant que l'émission et la réception se produisent exactement en même temps.

22-réalisation

Il a été décidé de procéder par étapes pour l'intégration des nouveaux composants :

1. -Installation de la connectique commune entre le hat rpi et la carte clavier..
2. - remplacer le clavier par un autre plus rigide.
3. proposer une structure de carte a usage commun.

Implantation matérielle :

- **Positionnement :** L'écran sera placé directement au-dessus du clavier.
- **Connectique :** L'écran sera branché sur la même carte que le clavier.

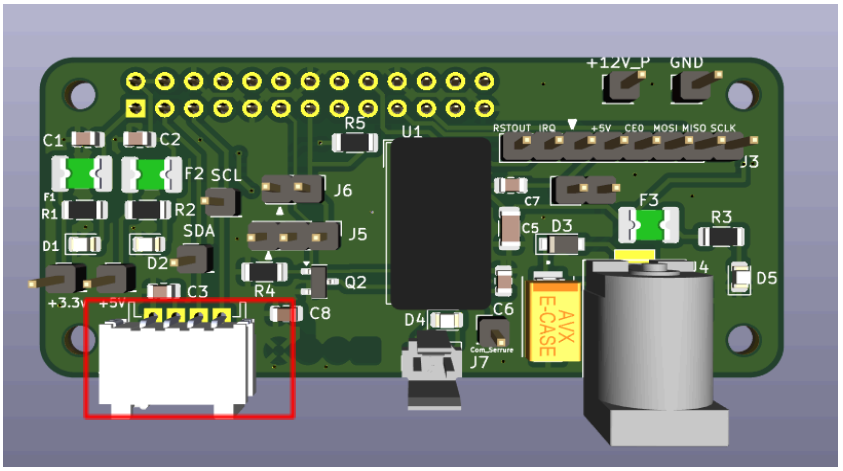
Modifications techniques : Après concertation, nous décidons de remplacer les ports I2C/SPI actuels par un **port tout en 1 qui** est nécessaire pour accueillir le bus i2c (ecran et clavier) et le bus spi (module NFC) ainsi que les tensions d'alimentation (5V/3.3V) . Ce port devra être implémenté sur deux éléments :

- Le HAT Raspberry Pi (RPI).
- La carte clavier.

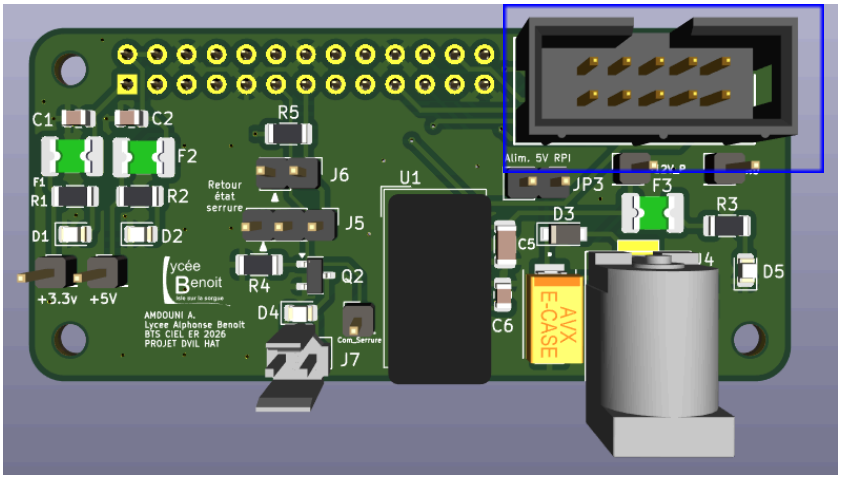
1 -Changement du connecteur sur le Hat RPI:

visualisation 3d

(ancien connecteurs)



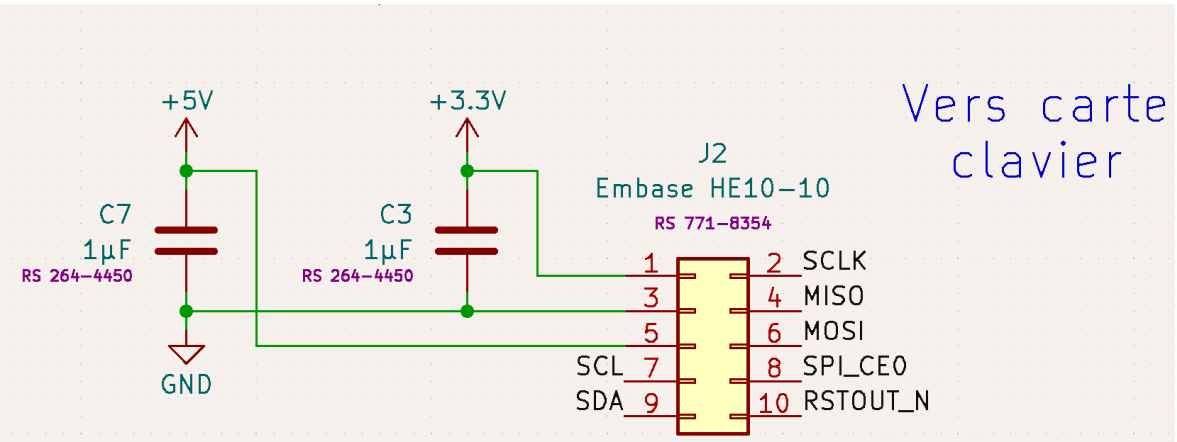
visu 3d (nouvelle version)



Analyse des changement ;

Le changement de connecteur 4 broches est en grande partie lié à la simplification d'utilisation. En effet, ici les cartes seront donc câblées l'une à l'autre avec un système plug & play qui permet donc de faire passer à la fois les bus I2C et SPI en un seul câble. Le modèle utilisé est un connecteur 2x5 broches droit HE10 équipé d'un détrompeur, d'où son placement à l'extrémité de la carte.

répartition des broches (scmat kicad) :



22.1- Câblage rapide de l’écran

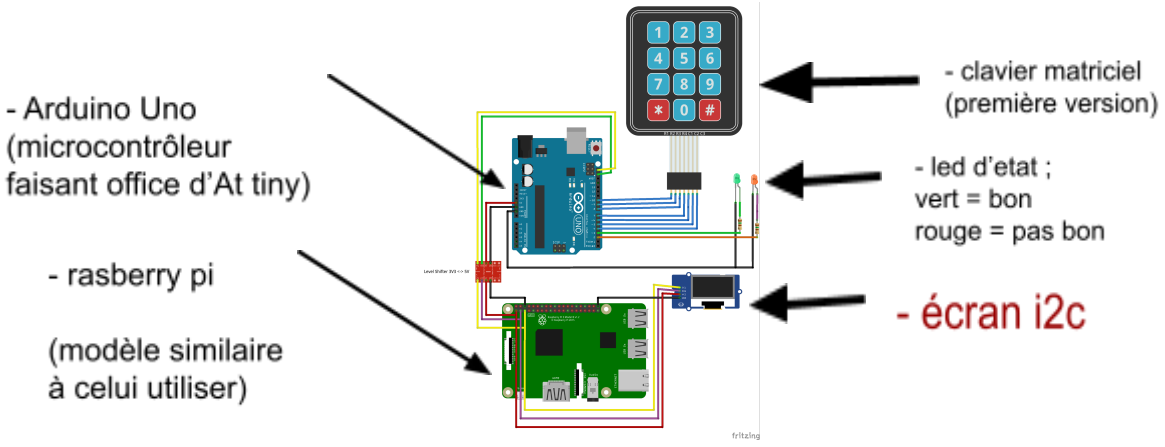
Réalisation du prototype : Câblage Fritzing

Dans le cadre du prototypage rapide, j'ai réalisé un schéma de câblage sous **Fritzing**. Ce montage met en œuvre les composants suivants :

- Un **écran LCD I2C**.
- Une carte **Raspberry Pi**.
- Une carte **Arduino Uno** (utilisée temporairement comme substitut à l'ATtiny pour valider la communication).

Détails techniques du montage : L'objectif est de tester l'interopérabilité entre les unités de calcul et l'affichage. L'utilisation de l'Arduino Uno permet de simplifier le débogage avant le passage définitif sur microcontrôleur ATtiny.

capture d’écran fritzing



-Analyse du montage de prototypage (frtizing) :

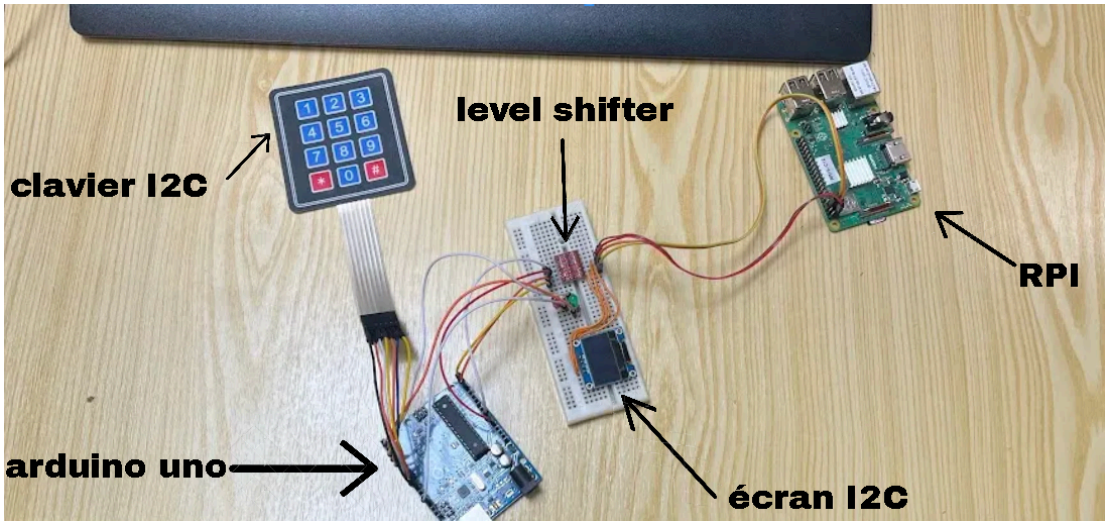
Le schéma Fritzing met en évidence deux aspects cruciaux du circuit :

- **Adaptation de tension (Level Shifter) :** Nous avons intégré un **convertisseur de niveau logique** (Level Shifter) afin d'adapter les signaux entre l'**Arduino Uno**, qui fonctionne sous **5V**, et la **Raspberry Pi**, qui fonctionne sous **3,3V**. Cette protection est indispensable pour ne pas

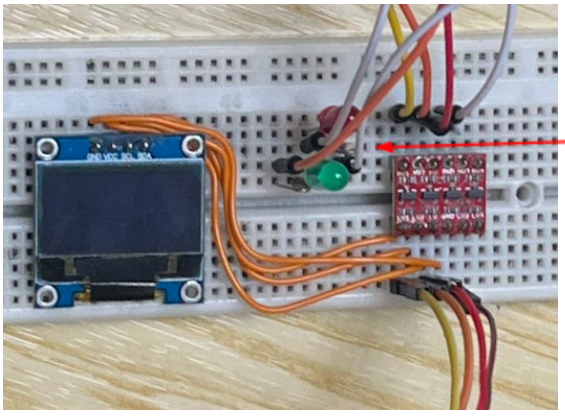
endommager les entrées/sorties de la Raspberry Pi.A noter que le level shifter ne sera pas présent sur la version définitive car la carte At-tiny fonctionne sous 3,3V donc les tensions seront compatibles entre cette dernière et la rasberry pi.

- **Indicateurs visuels (LED) :** Deux LEDs ont été ajoutées pour fournir un retour utilisateur immédiat :
    - **LED Verte :** S'allume si le code saisi sur le clavier est **correct**.
    - **LED Rouge :** S'allume si le code saisi est **erroné**.

-Mise en oeuvre dans la réalité :

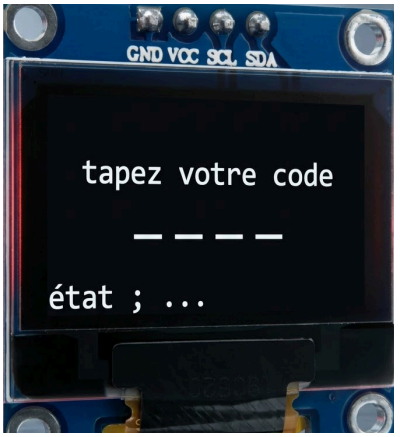


montage vue rapproché (breadboard)



Utilité de la mise en oeuvre :

Cette mise en œuvre dite rapide nous a permis d'en apprendre plus sur le fonctionnement de l'écran et nous avons pu calibrer une communication entre ce dernier et le clavier.



Par la suite, mes collègues IR ont pu programmer une interface interactive afin de pouvoir taper son code de manière sécurisée, tout en ayant un retour de l'état du code et de sa véracité sur l'écran.

23-Processus de fabrication

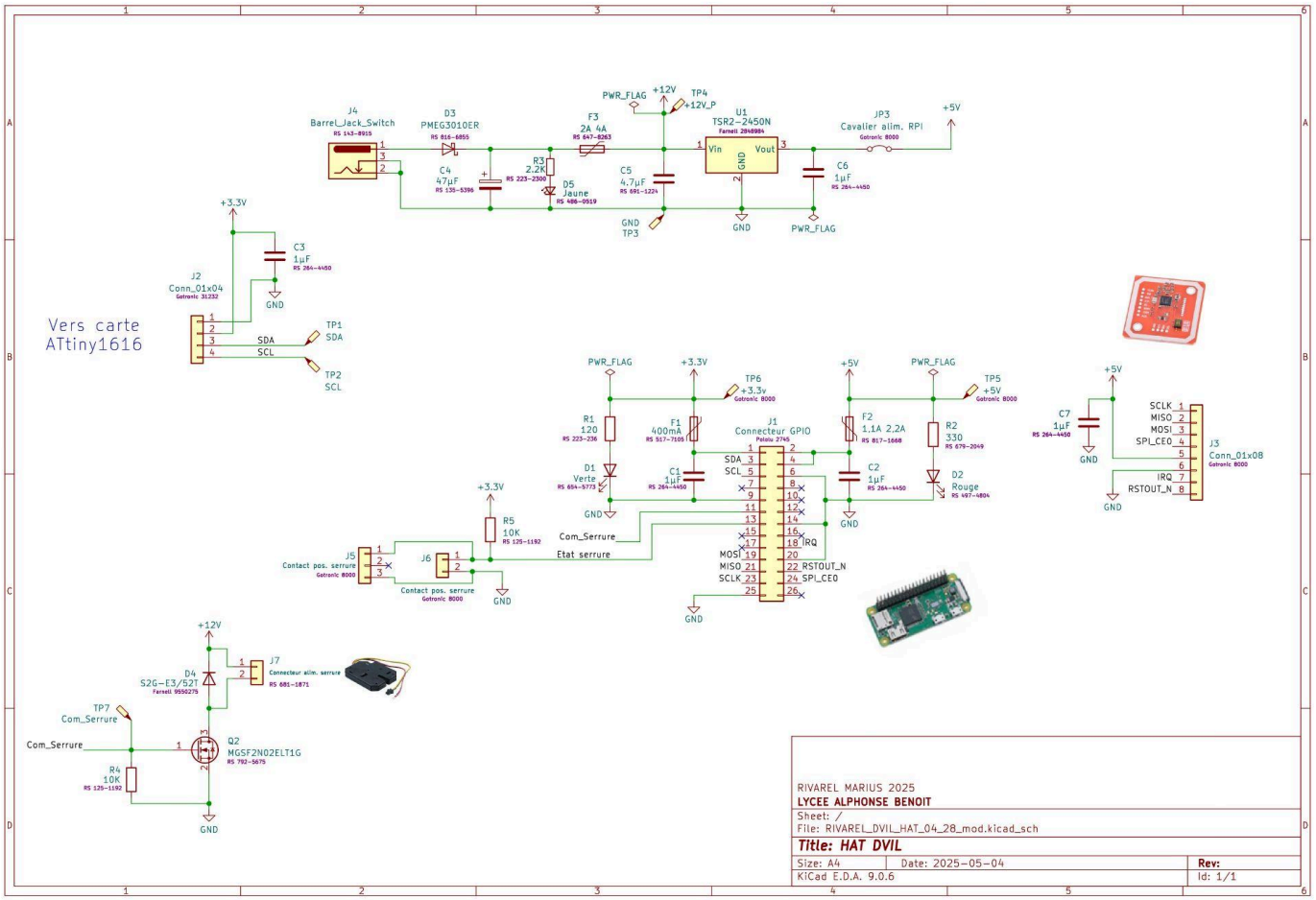
23.1 saisie de schéma

Pour la fabrication des nouvelles cartes de la session 2026, j'ai réalisé la saisie des schémas sous **KiCad 9.0**. Ce logiciel a permis d'assurer l'intégralité du processus de conception des deux cartes, de la saisie schématique jusqu'à la finalisation du routage.

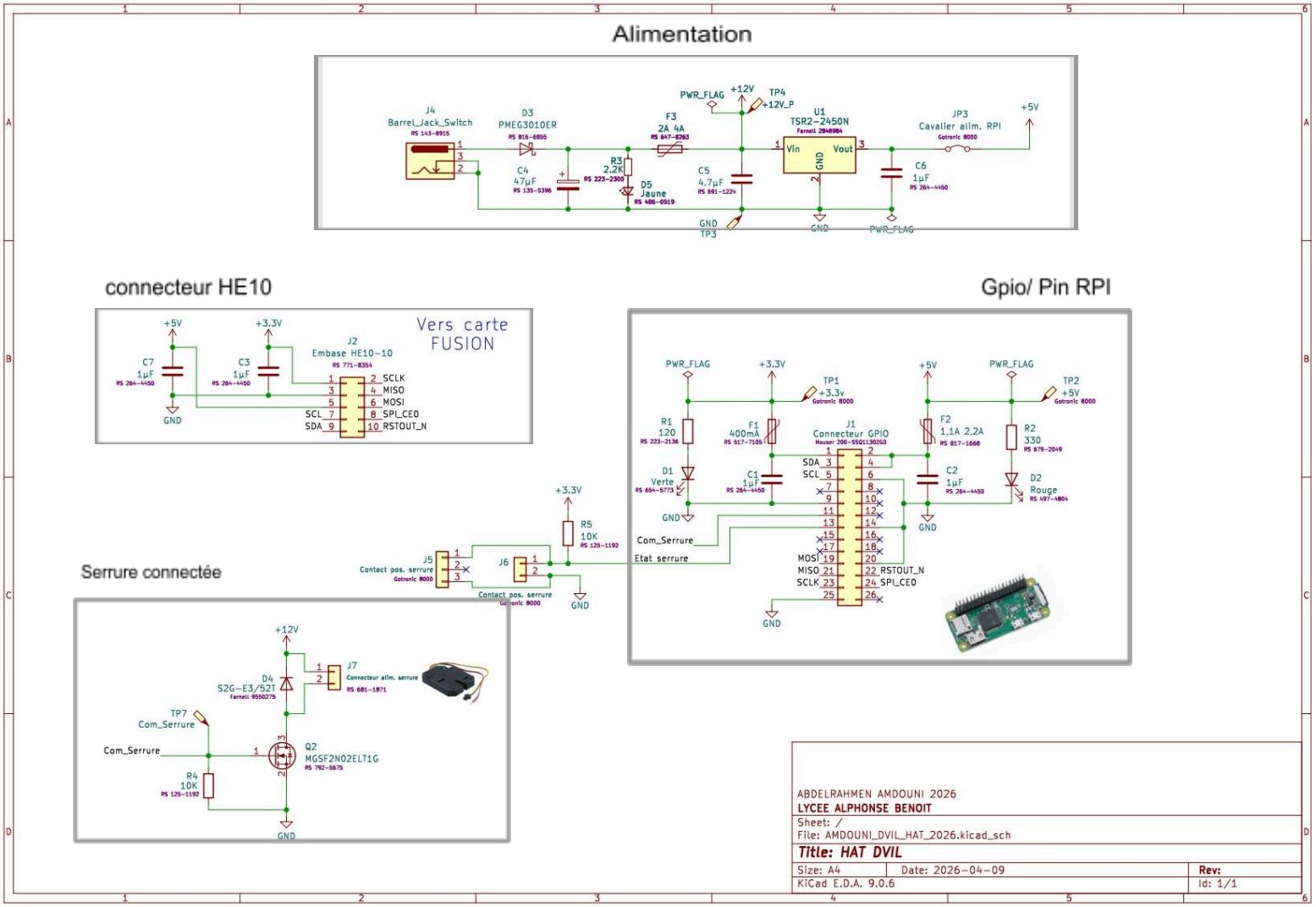
Méthodologie de conception

Bien que ce projet soit une reconduction, j'ai choisi de reprendre la conception à la base. Je me suis ainsi appuyé sur la version de l'année précédente pour m'en inspirer, tout en restructurant l'ensemble afin d'optimiser les nouvelles cartes.

- schéma de la carte hat (version 2025) :



- schéma de la carte hat (version 2026) :



- Comparaison des cartes HAT ;

1. Centralisation de la connectique

- **Version 2025 :** La communication vers la carte Fusion était étalée sur plusieurs connecteurs (J2 pour l'I2C vers l'ATtiny et J3 pour le bus SPI).
- **Version 2026 :** Tout est regroupé sur **un seul connecteur HE10-10 (J2)**. Ce connecteur unique transporte désormais :
  - L'alimentation (5V, 3.3V, GND).
  - Le bus SPI (SCLK, MISO, MOSI, SS).
  - Le bus I2C (SDA, SCL).
  - Le signal de Reset (RST OUT N).

2. Gestion de l'alimentation

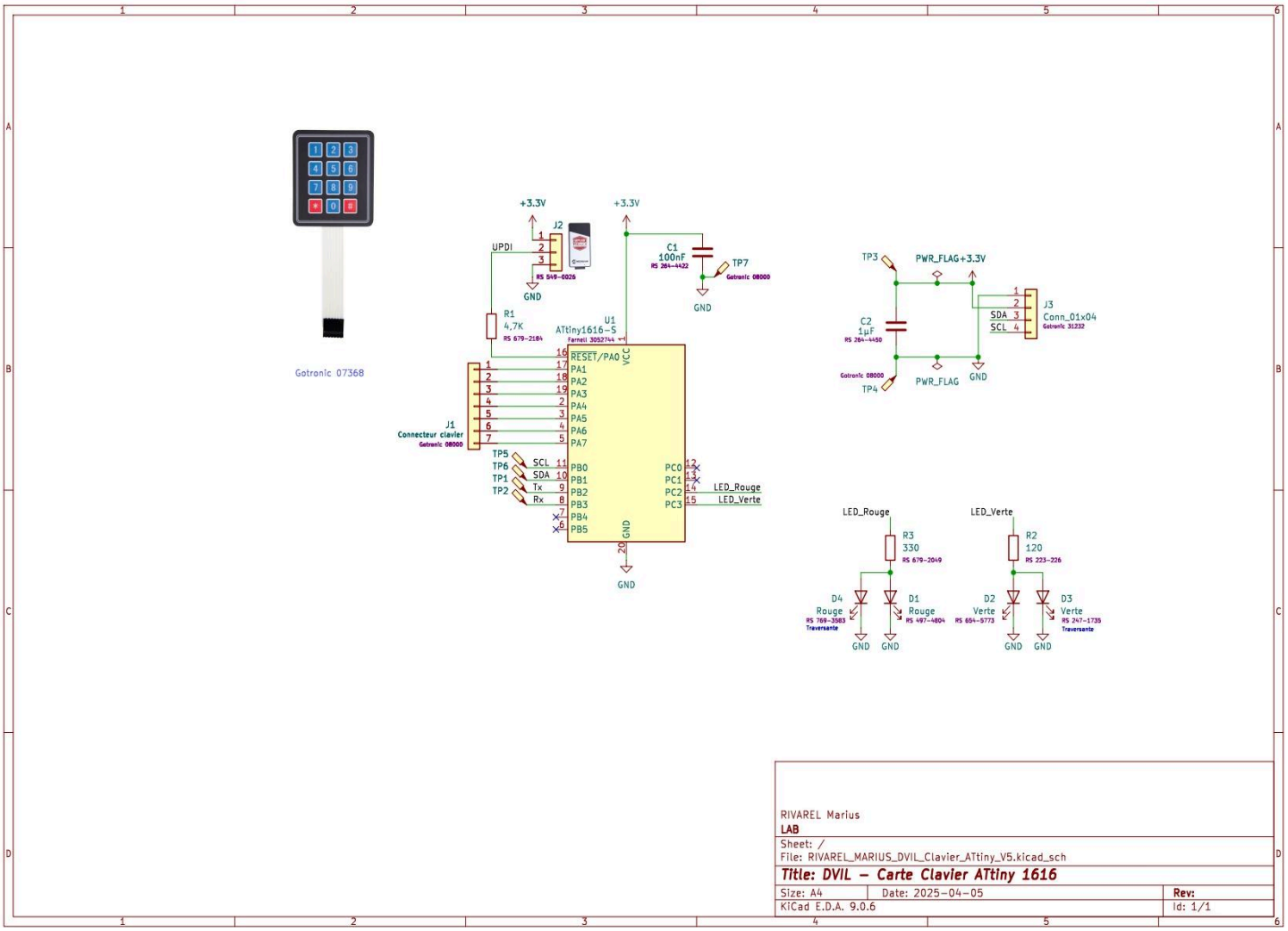
- **Régulateur :** Le régulateur **TSR2-2450N** (U1) reste identique pour convertir le 12V en 5V.

- **Protection :** Les deux versions utilisent des fusibles réarmables (F1, F2) pour protéger les lignes 3.3V et 5V.

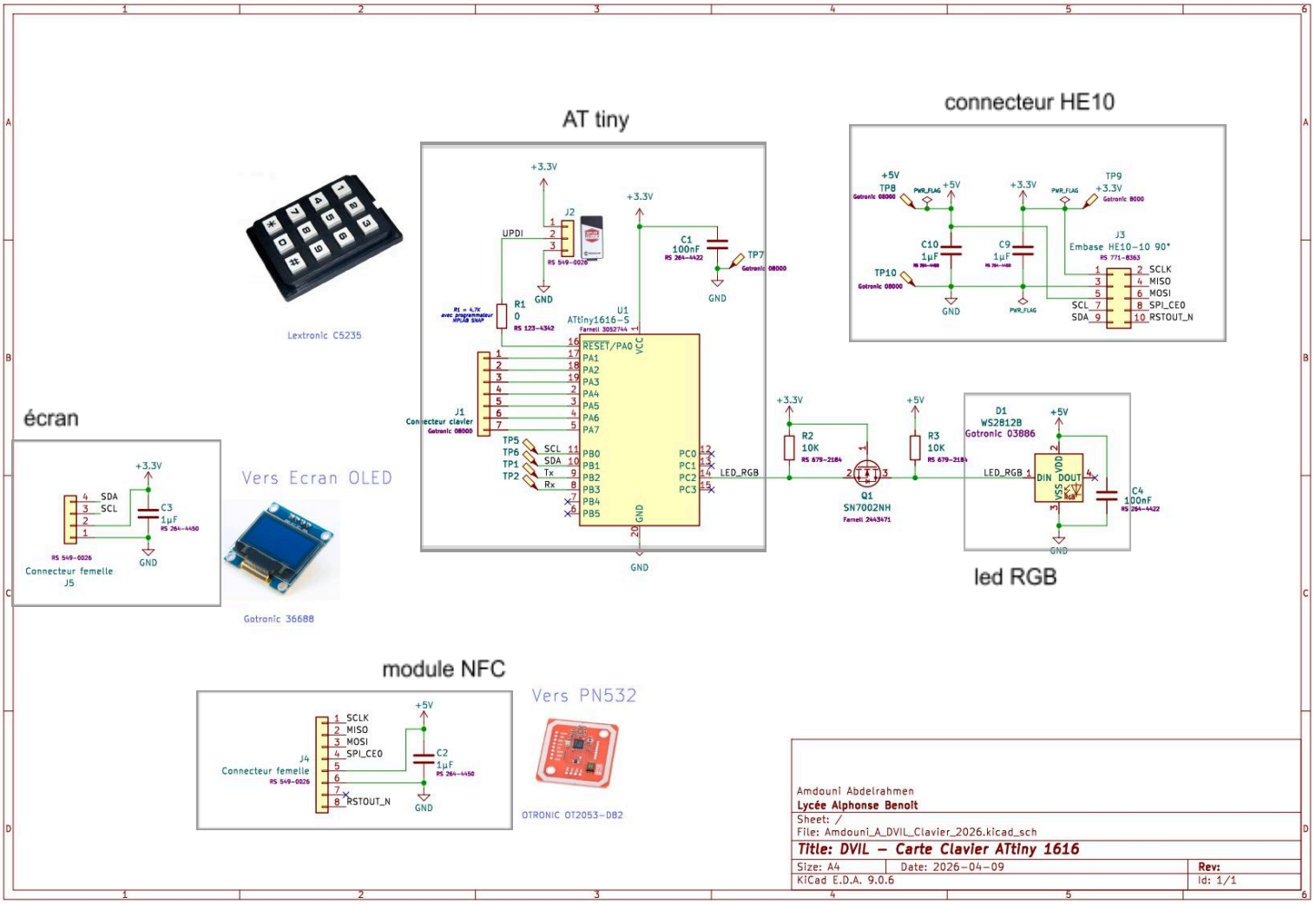
3. Interface Raspberry Pi (J1)

- Les deux cartes utilisent un connecteur GPIO partiel (26 pins).
- **Nouveauté 2026 :** On note l'apparition de signaux plus spécifiques sur le connecteur J1, mieux reliés au connecteur HE10 pour assurer que la Raspberry Pi puisse piloter directement le module NFC via le bus SPI traversant le HAT.

[- schéma de la carte clavier (version 2025) :]



- schéma de la carte clavier (version 2026) :



Comparaison Carte Clavier (2025 et 2026)

- **Centralisation :** En 2025, la carte ne gérait que le clavier. En 2026, elle devient une **Carte Fusion** qui regroupe tout (Clavier + Écran OLED + NFC + LED RGB).
- **Câblage :** On passe de plusieurs connecteurs éparpillés à un **connecteur unique HE10-10**. C'est plus propre, plus rapide à monter et plus fiable.
- **Technologie LED :** Passage de LEDs classiques à une **LED programmable WS2812B** (RGB), offrant plus d'états visuels (clignotements, couleurs variées).

2. Pourquoi la résistance de 0 Ohm sur la ligne UPDI ?

C'est un **pont de configuration** :

- **Flexibilité :** Elle permet de choisir quel programmateur a la priorité sur L'ATtiny.
- **Isolation :** En ne soudant pas R1, cela déconnecte physiquement le connecteur J2. Cela permet d'injecter le signal de programmation par un autre port sans que le connecteur J2 ne crée de conflits électriques ou de parasites sur la ligne de données UPDI.
- **Sécurité :** Une fois le projet fini, retirer cette résistance "verrouille" l'accès à la programmation via J2.

3. Étude technique du Level Shifter (Q1)

Le signal de commande de la LED WS2812B est extrêmement sensible. Une simple chute de tension ou un front montant trop lent corrompt la donnée.

- **Le déséquilibre des seuils logiques :**
  - **Sortie ATtiny :** VOH (Tension de sortie haute) = **3,3V**.
  - **Entrée WS2812B :** ViH (Tension d'entrée haute minimale) = 0,7 xVcc. Avec une alimentation à **5V**, le seuil est de **3,5V**.
  - **Problème :** 3,3V est inférieur à 3,5V. Le signal est dans une zone d'ombre (zone indéterminée), ce qui cause des erreurs de bits.
- **Le rôle du MOSFET (SN7002NH) :**
  Il est utilisé en **commutation rapide**. Le transistor va piloter la ligne LED RGB en tirant vers le **5V** via une résistance de pull-up (R3).
-
  - **Vgs (Gate-Source Voltage) :** Le SN7002NH à une tension de seuil VGSth située entre **1V et 2,5V**. Les 3,3V de l'ATtiny sont donc largement suffisants pour saturer complètement le transistor.
  - **Temps de montée (Rise Time) :** Ce modèle a un temps de commutation très faible (quelques nanosecondes), ce qui permet de conserver l'intégrité du signal PWM/Data nécessaire à la WS2812B sans déformer les impulsions.

- **En résumé technique :** Q1 sert d'interface d'adaptation d'impédance et de tension. Il transforme une commande **basse tension (3,3V)** à faible courant en un signal **pleine échelle (0V - 5V)** capable de piloter l'entrée de la première LED

Assignation des empreintes :

Une fois que tous les composants sont placés sur le schéma et reliés électriquement, l'étape cruciale consiste à **assigner une empreinte** à chaque symbole. Cette phase est le pont indispensable entre le schéma théorique et la conception physique du circuit (le PCB).

- **L'empreinte (Footprint) :** Elle définit les dimensions réelles du composant, la forme des pastilles de cuivre (pads) et l'espacement entre les broches (**pitch**).
- **Le choix technologique :** C'est à ce moment que l'on tranche entre les composants **CMS** (Composants Montés en Surface), pour gagner de la plac,e et les composants **traversants** (Through-Hole) , plus robustes mécaniquement (comme le HE10 ou le bornier de la serrure).

23.2 Routage de la carte

Après la saisie de schéma vient le routage des composants, c'est ce qui permet de disposer les différents composants ou on le souhaite sur une surface limitée pour ensuite générer les fichiers de fabrication.

-Voici ce qu'il y a à savoir sur les différentes fonctionnalité quand on route une carte ;

1. Pistes et Dimensions

Les **pistes** sont les chemins de cuivre qui transportent le courant.

- **Taille :** On adapte la largeur de la piste selon son rôle. Les pistes de signaux (data) peuvent être fines (ex: **0.25 mm**), mais les pistes de puissance (Alim 12V de la serrure, 5V, GND) doivent être plus larges pour supporter l'intensité sans chauffer.

2. Double face et Vias

- **Faces :** On utilise généralement deux couches : la **Face Avant** (F.Cu, rouge) et la **Face Arrière** (B.Cu, bleu). Cela permet de croiser des signaux sans créer de court-circuit.

- **Vias :** Un **via** est un petit trou métallisé qui permet de faire passer une piste d'une face à l'autre.

3. Plan de masse (Zone de remplissage)

Au lieu de tirer des pistes pour chaque connexion à la masse (GND), on crée un **plan de masse**.

- On "remplit" toute une face (souvent l'arrière) de cuivre relié au GND.
- **Uniformisation :** On place des "vias de couture" (via stitching) pour relier les plans de masse des deux faces. Cela réduit les parasites (CEM) et améliore la stabilité électrique.

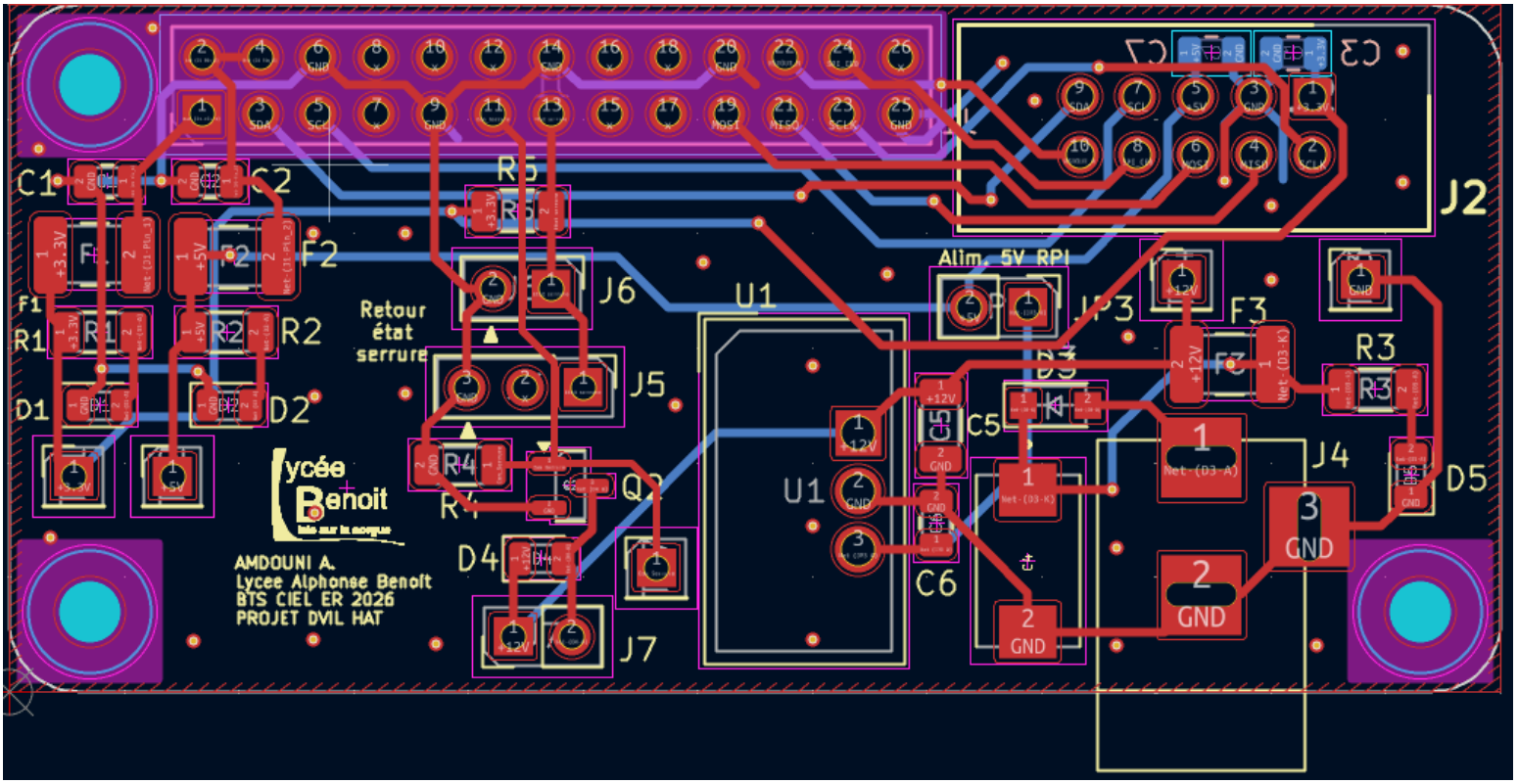
4. Règles de conception (DRC)

KiCad intègre un outil de **vérification des règles électriques (DRC - Design Rule Check)**.

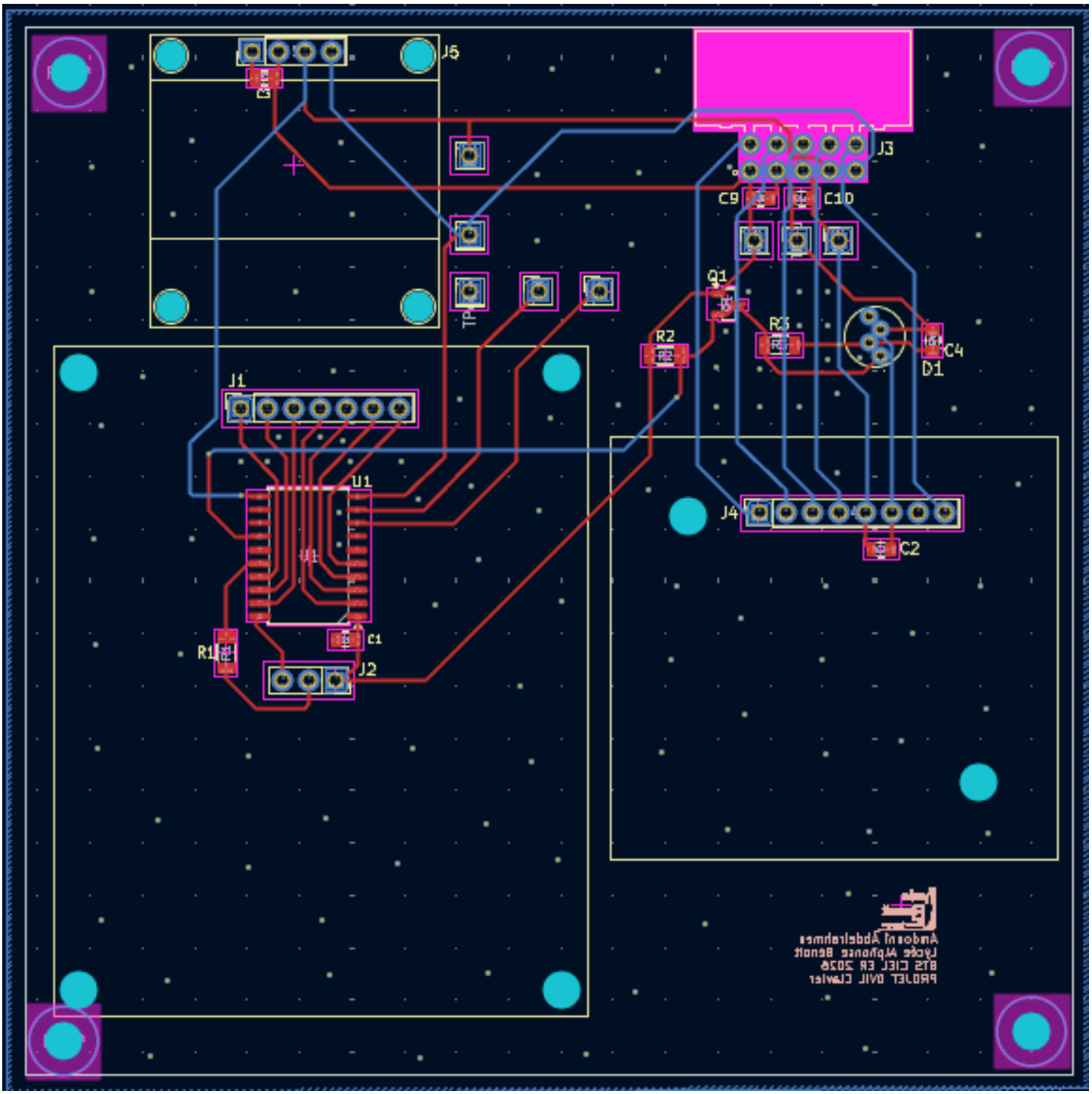
- Il vérifie en temps réel ou à la demande que les pistes ne sont pas trop proches (isolement), qu'aucune connexion n'est oubliée et que les largeurs minimales sont respectées. C'est le garde-fou indispensable avant d'envoyer la carte en fabrication.

routage des carte d-vill et clavier:

carte hat



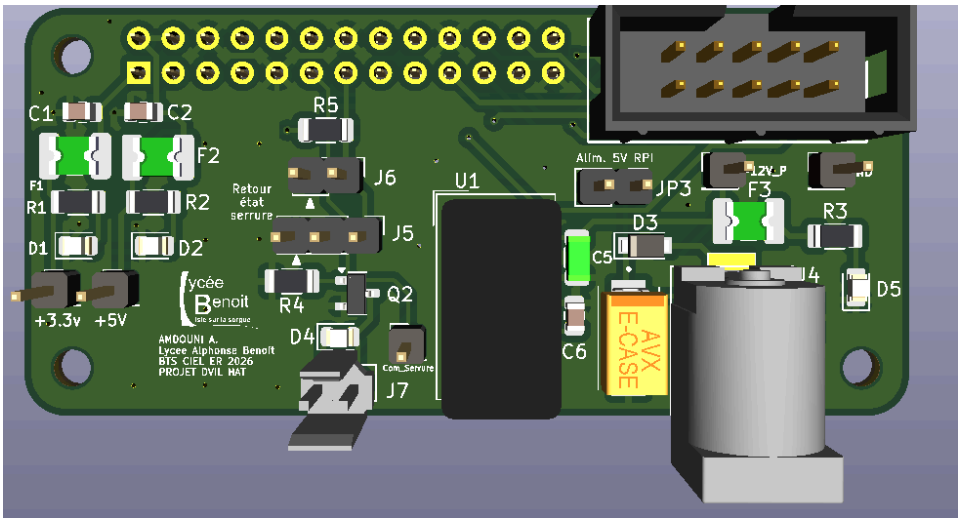
carte clavier



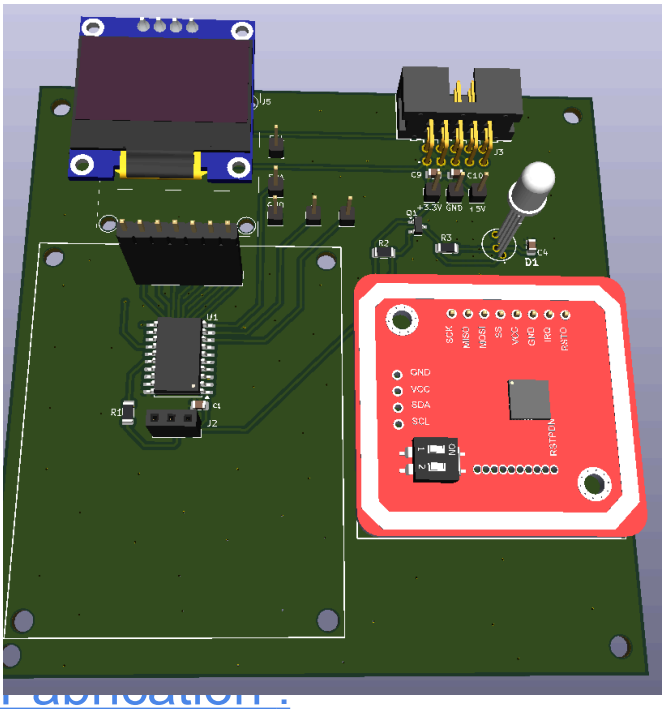
Rendu 3D:

Dans KiCad, la visualisation 3D permet de vérifier en temps réel si le positionnement théorique des empreintes est cohérent avec la réalité physique. Cet outil est indispensable pour contrôler l'encombrement des composants, s'assurer qu'ils ne se chevauchent pas et vérifier que les connecteurs sont bien accessibles pour le montage final dans le coffre-fort.

rendu carte HAT ;



rendu carte clavier ;



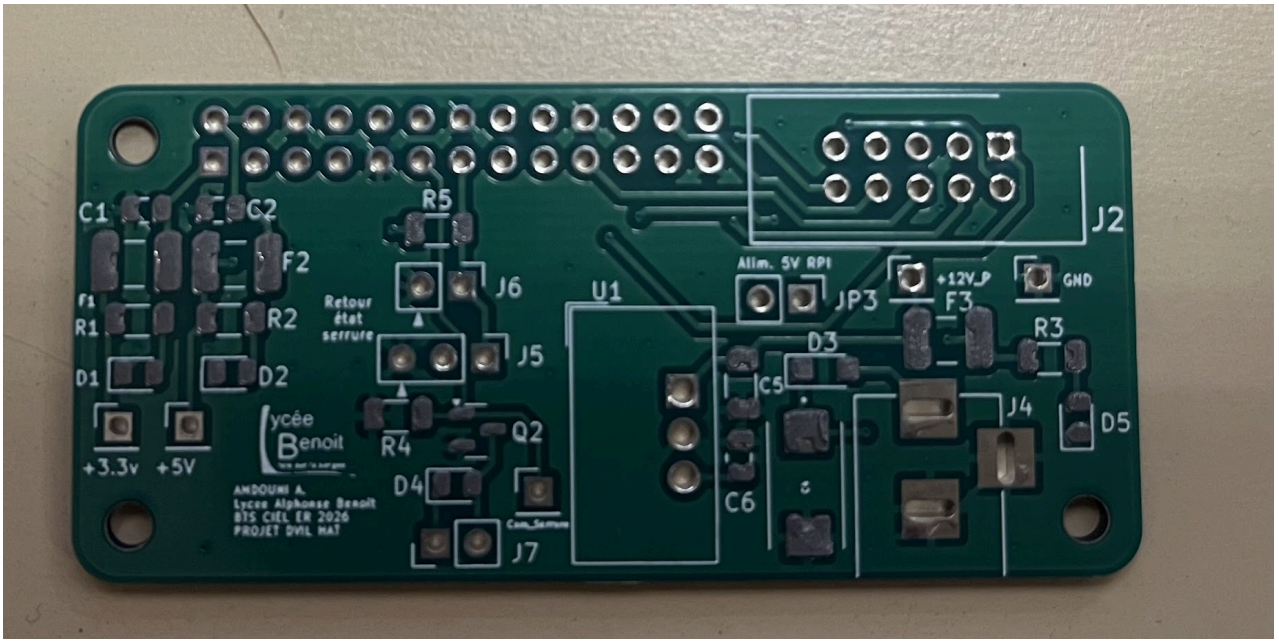
Fabrication :

Commande PCB :

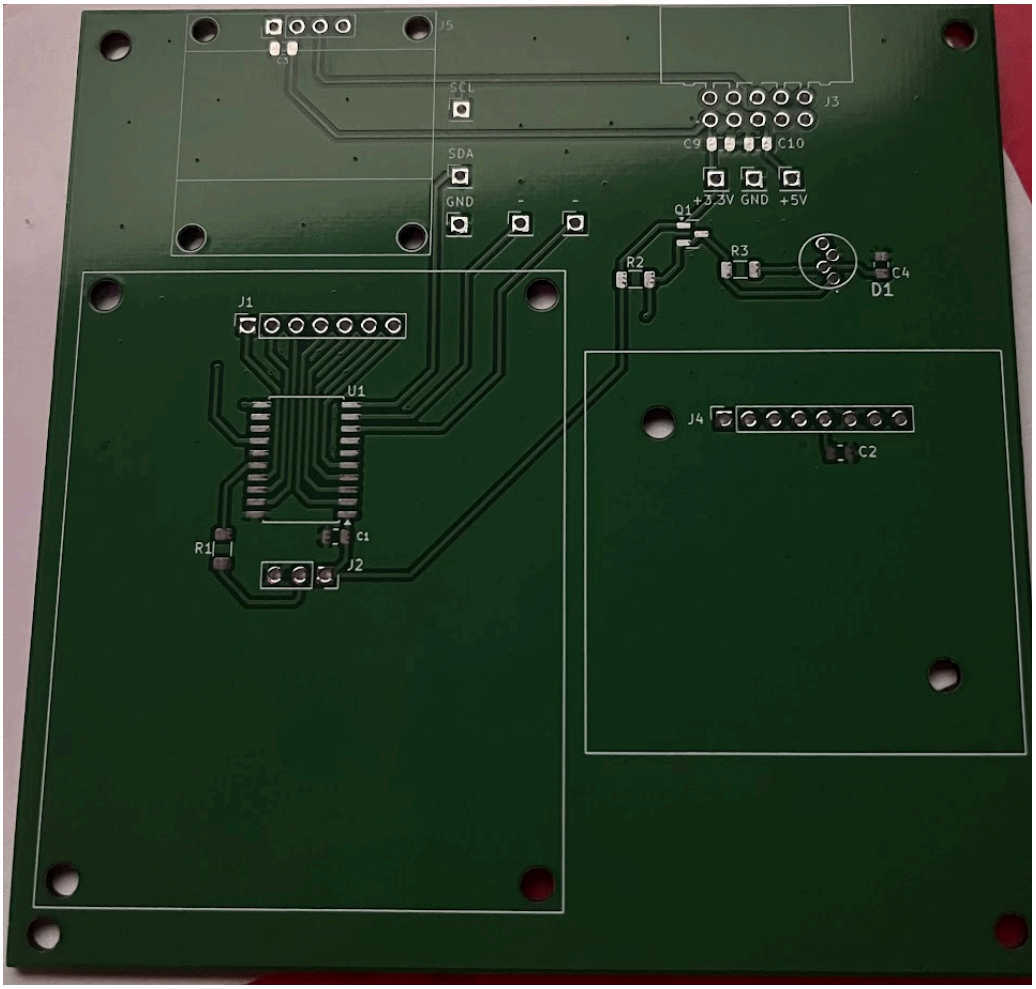
Après le routage, je génère les fichiers de fabrication **Gerber** pour les envoyer à **JLC-PCB**. Ce sous-traitant fabrique les cartes du lycée pour environ **5 euros** l'unité (hors livraison). Afin d'optimiser le budget, nous regroupons les commandes de plusieurs projets pour réduire l'impact des frais de port sur le coût total.

Voila ce les cartes que j'ai reçu après 15 jours d'attente environ ;

-Carte HAT (vierge) :



-Carte clavier vierge :



-Câblages des cartes ;

Assemblage des composants CMS :

Après réception, j'ai assemblé les cartes en commençant par les **composants montés en surface (CMS)** avant de passer au brasage par refusion. Le processus suit trois étapes précises :

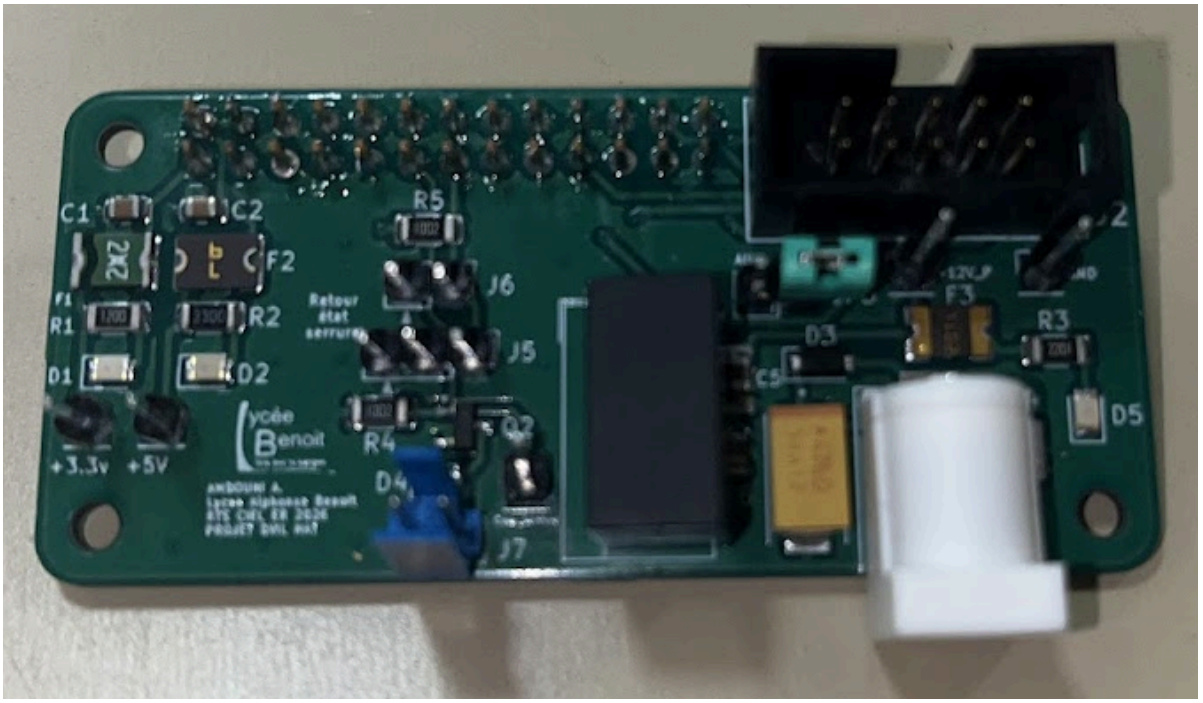
1. **Sérigraphie de la pâte à braser :** J'ai utilisé un **stencil** (pochoir en acier pré-découpé) pour appliquer la pâte à braser uniquement sur les plages d'accueil (pads) des composants. Cela garantit une précision maximale et un dosage constant.
2. **Placement (Pick and Place) :** Une fois le stencil retiré, j'ai positionné chaque composant CMS sur sa couche de pâte à l'aide d'une **pince brucelle**. La viscosité de la pâte permet de maintenir les composants en place avant la chauffe.
3. **Passage au four à refusion :** La carte est ensuite placée dans un four suivant une courbe de température précise (un modèle).Chauffée a 275°C

pendant 2min50s, La pâte fond pour créer une liaison électrique et mécanique solide entre le composant et le circuit imprimé.

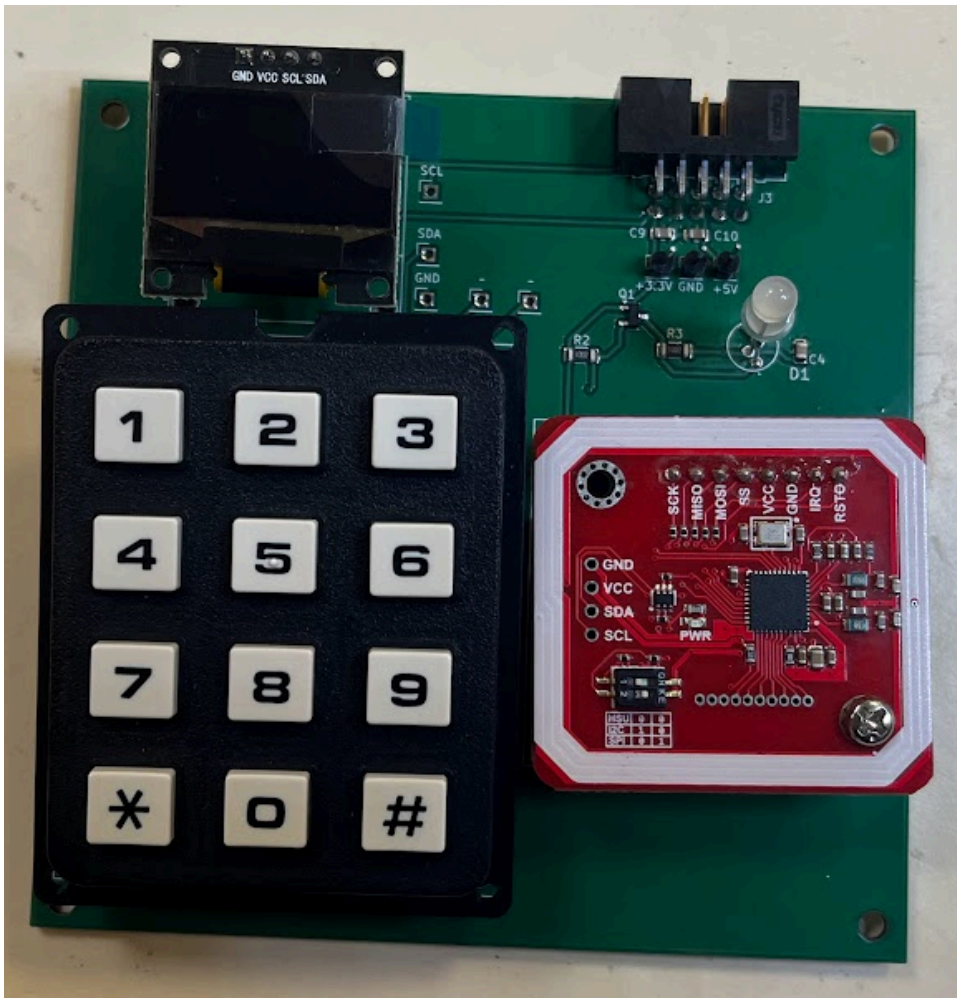
Note technique : On commence toujours par les CMS car ils sont plus petits et nécessitent un passage au four. Les composants traversants (plus gros) sont soudés manuellement à la fin pour éviter d'endommager les éléments sensibles lors de la refusion.

Carte HAT et Clavier après pose cms et traversant;

carte HAT

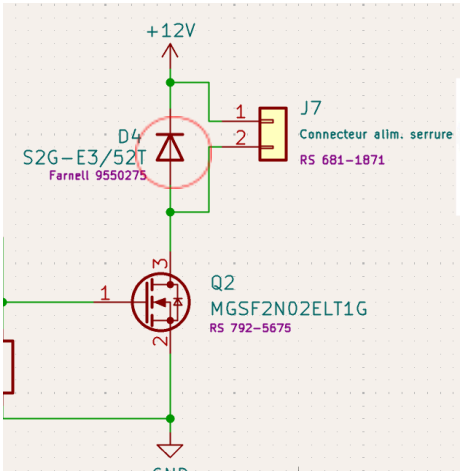


carte clavier



-Défaut mineur ;

Un défaut mineur a été relevé sur la carte **DVIL 2025**. En effet, bien que présente sur le schéma, la **diode de roue libre D4** n'était pas câblée sur la version antérieure. Cette diode est pourtant indispensable pour protéger le transistor **Q2**. Sans elle, lors de l'arrêt de l'alimentation de la serrure (charge inductive), une **surtension importante** (force contre-électromotrice) est générée, ce qui expose le transistor à un risque de destruction immédiate.



24-Fiche de commande des composants

carte hat dvill ;

Désignation (Ref)	Valeur	Code Commande	Prix Unit. (est.)	Total
Régulateur (U1)	TSR2-2450N	Farnell 2848984	20,00€	20,00€
Transistor (Q2)	MGSF2N02ELT1	RS 792-5675	0,15€	0,15€
Connecteur GPIO (J1)	Connecteur RPi	Pololu 2745	1,50€	1,50€
Embase 90° (J2)	HE10-10	RS 771-8363	0,55€	0,55€
Diode Roue Libre (D3)	PMEG3010ER	RS 816-6855	0,05€	0,05€
Diode (D4)	S2G-E3/52T	Farnell 9550275	0,05€	0,05€
Fusible (F2)	1.1A / 2.2A	RS 817-1668	0,30€	0,30€
Résistances (R4, R5)	10K	RS 125-1192	0,03€	0,06€
Condensateurs (C1-C7)	1µF	RS 264-4450	0,03€	0,15€
Condensateur (C4)	47µF	RS 135-5396	0,05€	0,05€
Fabrication PCB	Unité	JLCPCB-2LAYER	2,00€	2,00€
--	--	--	--	--
TOTAL (par carte)				24,86€

carte clavier ;

Reference	Qty	Value	Code commande	Prix(à l'unité)	prix total :
C1,C4	2	100nF	RS 264-4422	0.25	27.21
C2,C3,C9,C10	4	1µF	RS 264-4450	0.18	
D1	1	WS2812B	Gotronic 03886	1.32	
J1	1	Connecteur clavier	RS 549-0026	5.5	
J2	1	Conn_01x03	RS 549-0026	5.5	
J3	1	Conn_02x05_Odd_Even	RS 771-8363	0.7	
J4	1	Connecteur femelle	RS 549-0026	5.5	
J5	1	Connecteur femelle	RS 549-0026	5.5	
Q1	1	SN7002NH	Farnell 2443471	0.25	
R1	1	0	RS 123-4342	0.017	
R2,R3	2	10K	RS 679-2184	0.033	
TP1	1	Tx	Gotronic 08000	0.18	
TP2	1	Rx	Gotronic 08000	0.18	
TP5	1	SCL	Gotronic 08000	0.18	
TP6	1	SDA	Gotronic 08000	0.18	
TP7,TP10	2	GND	Gotronic 08000	0.18	
TP8	1	+5V	Gotronic 08000	0.18	
TP9	1	+3.3V	Gotronic 08000	0.18	
U1	1	ATtiny1616-S	Farnell 3052744	1.2	
PCB			JLPCB	2	

gant réel

GanttProject *

Projet Éditer Affichage Tâches Ressources Aide

Ctrl + F

Diagramme de Gantt | Diagramme des Ressources

Zoom avant | Zoom arrière | Début du projet | Reculer | Avancer | Afficher le chemin critique | Etats de référence du projet ...

2025 2026

Nom	Date de début	Date de fin
Gantt_réel	05/01/2026	15/06/2026
Evaluations du projet	19/01/2026	15/06/2026
Revue 1	19/01/2026	20/01/2026
Revue 2	09/02/2026	11/02/2026
Revue 3	06/04/2026	08/04/2026
Dépôt dossier technique	22/05/2026	22/05/2026
Soutenance finale E62	08/06/2026	15/06/2026
Epreuve E4	13/05/2026	13/05/2026
Epreuve E1	12/05/2026	12/05/2026
Vacances d'hiver	02/02/2026	19/02/2026
Vacances printemps	13/04/2026	20/04/2026
Projet	05/01/2026	15/06/2026
Specifications générales	05/01/2026	06/01/2026
Prise en main, essais, valida...	07/01/2026	22/01/2026
Saisie de schéma et routage	26/01/2026	06/02/2026
Fabrication et essais	06/04/2026	22/04/2026
Dépannage Finalisation doss...	11/05/2026	15/06/2026
Entraînement soutenance	25/05/2026	02/06/2026
Fichier de fabrication PCB fi...	07/04/2026	07/04/2026
Rédaction du dossier	15/01/2026	08/05/2026
1er BTS Blanc	12/02/2026	13/02/2026
2nd BTS Blanc	09/04/2026	09/04/2026
Soutenance E61	19/01/2026	22/01/2026

Cloud : Déconnecté

RSS Alertes Erreurs

Fiche compte rendu Amdouni Abdel

Chronologie du projet (Janvier – Mai 2026)

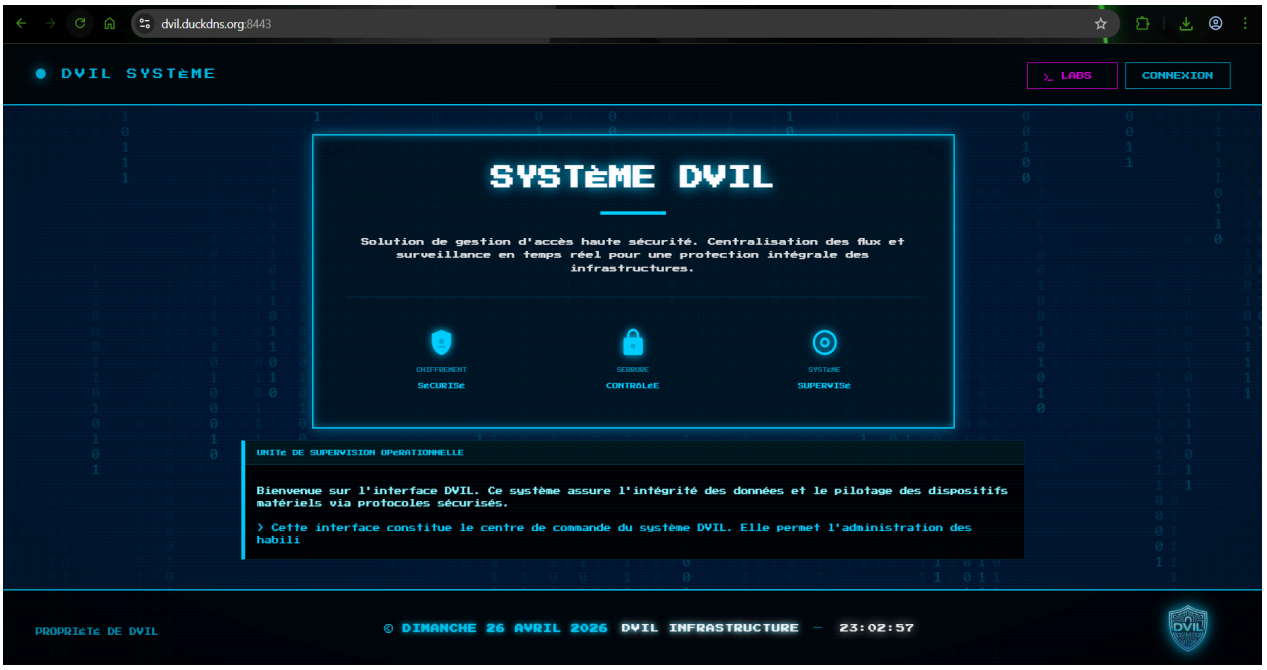
- **Mi-Janvier : Initialisation et Cadrage**
  - Analyse de la version précédente (2025) et visualisation du matériel existant avec l'équipe IR.
  - Estimation des durées et finalisation du planning Gantt personnel.
- **Fin Janvier : Réarchitecture et Validation Connectique**
  - Réunion ER/IR : choix d'intégrer l'OLED en I2C (au-dessus du clavier) puis le module de cryptage (module finalement abandonné). Décision d'ajouter un second port (HE10) sur le HAT et la carte Clavier.
  - Modification du schéma structurel du HAT RPi.
  - *Alerte technique (30 janv.) :* Validation du câblage de la nappe 2x5. Si la torsade d'origine est supprimée, les broches s'inversent, imposant une modification des empreintes sur le schéma.
- **Début Février : Prototypage Rapide**
  - Maquette virtuelle sous Fritzing (Rpi + Arduino Uno à la place de l'ATtiny + écran I2C) puis validation par un câblage physique sur platine d'essai (*breadboard*).
- **Février – Mars : Conception Finale & Fabrication**
  - Finalisation des schémas de la carte Clavier et routage complet du PCB.
  - Lancement de la fabrication des circuits imprimés.
- **Mars – Avril : Câblage & Programmation**
  - Réception des composants et soudure (câblage matériel) de la nouvelle carte Fusion.
  - Développement de l'algorithme et programmation de l'ATtiny (gestion du clavier matriciel, de la LED RGB et de l'écran).
- **Mai : Phase de Tests & Intégration**
  - Tests unitaires des composants (communication I2C, et SPI).
  - Tests d'intégration globaux avec la partie des étudiants IR (liaisons avec le Raspberry Pi, module NFC et Serrure) pour finaliser le coffre-fort connecté.

Annexes

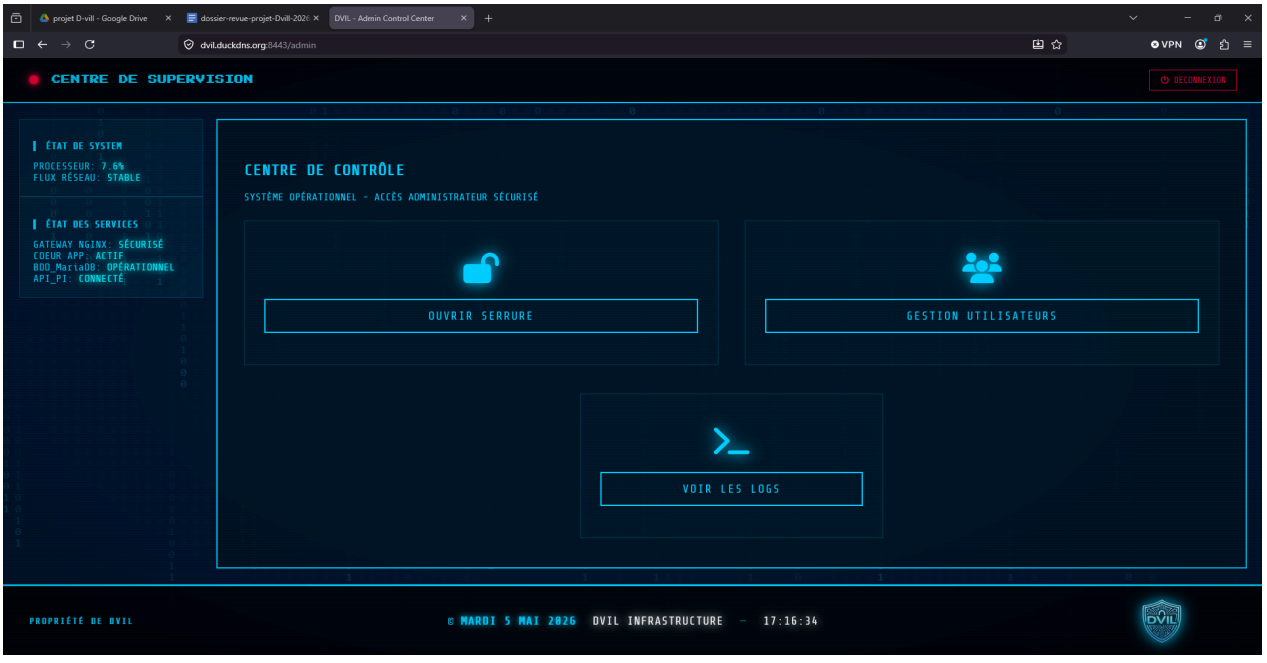
Annexes IR1

Annexe 1 : Vue de l'interface web et API

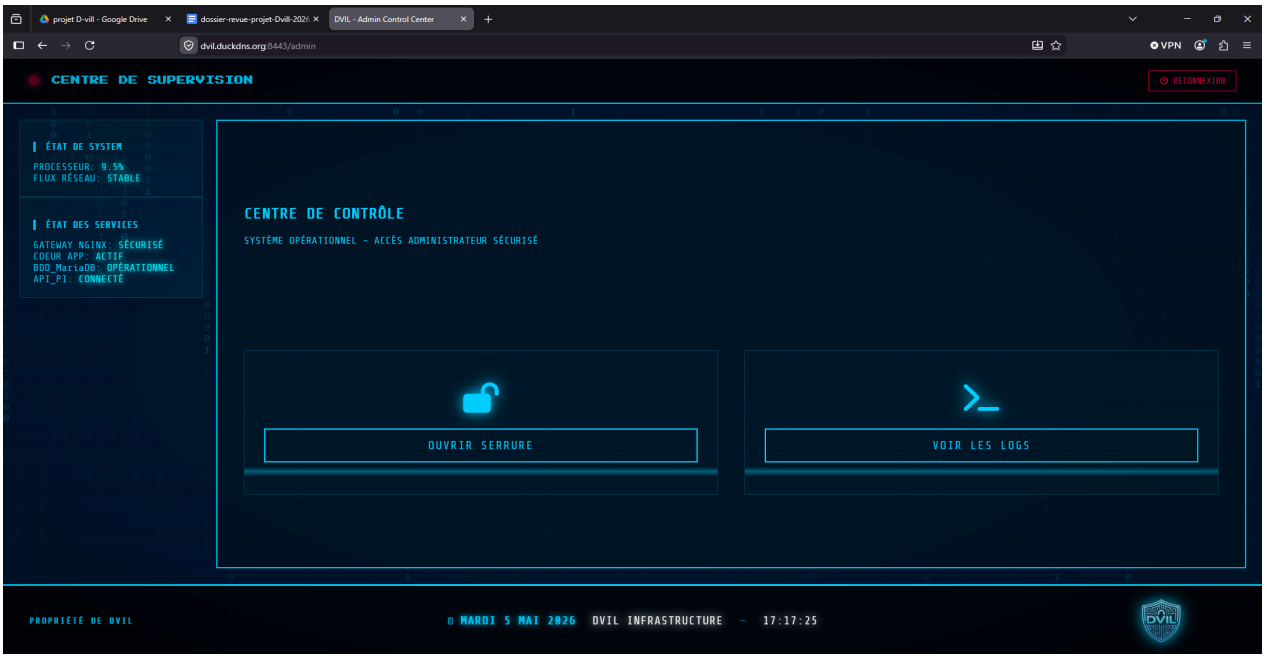
Page de présentation :



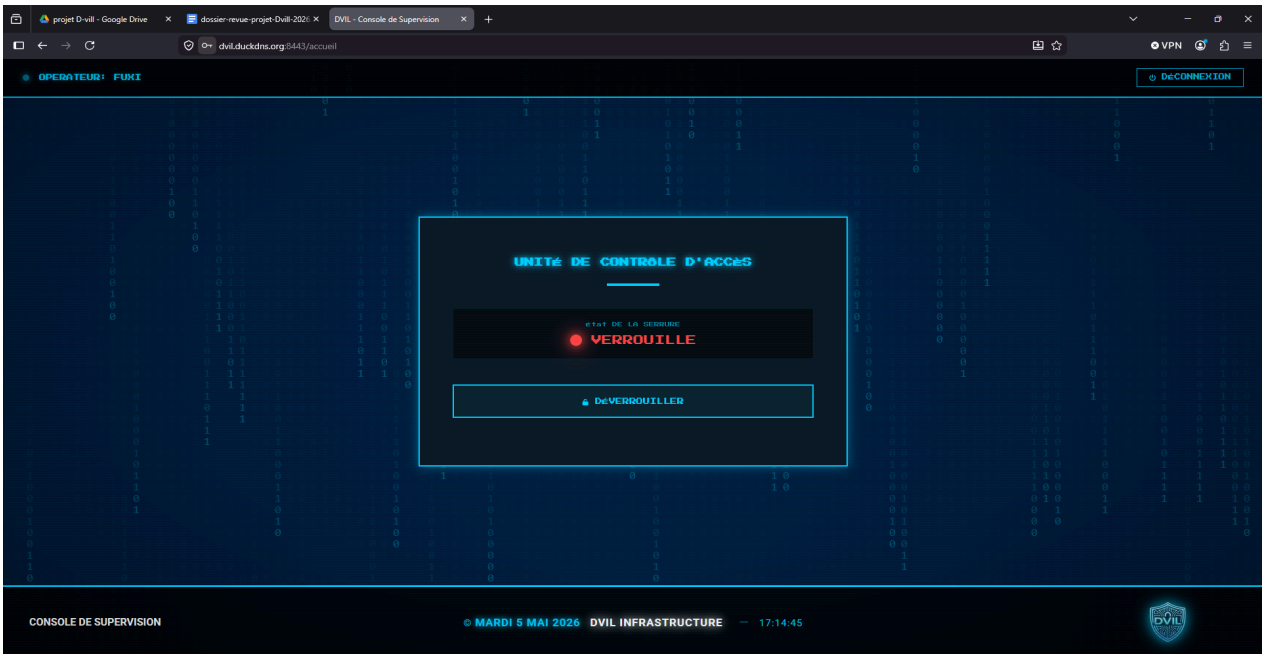
Page d'accueil (pour admin) :



Page d’accueil (pour professeur) :



Page d'accès au contrôle de la serrure pour les utilisateurs :



Page de la gestion des utilisateurs :



Page des logs :



Annexe 2 : Dictionnaire des Endpoint

Endpoint (Route)	Méthode	Description
/login	**POST**	Authentification des administrateurs (génère un token **JWT**)
/admin/users	**GET**	Interface de gestion des utilisateurs et des badges
/admin/logs	**GET**	Consultation de l'historique complet des accès remontés par l'API locale
/api/users/sync	**GET**	Envoie la liste des accès autorisés à l'API locale
/api/logs/sync	**POST**	Reçoit et enregistre les logs envoyés par l'API locale
/api/serrure/unlock	**POST**	Commande d'ouverture manuelle forcée depuis l'interface de supervision (web)
/api/serrure/status	**GET**	Renvoie l'état actuel de la serrure (verrouillée/déverrouillée)
/api/challenge/verify	**POST**	Validation du "Flag" lors des tests d'infiltration (Cybersécurité)

/api/local/verify	**POST**	Commande d'ouverture manuelle forcée depuis l'interface de supervision (web)
/api/local/status	**GET**	Renvoie l'état actuel de la serrure (verrouillée/déverrouillée)
/api/local/config	**GET**	Validation du "Flag" lors des tests d'infiltration (Cybersécurité)

-> Code API Flask (VM - app.py) : Voir le code source depuis le lien framagit ou GitLab dans la page 1 (chemin : /interface WEB/flask_app/app.py)

Annexe 3 : Documentation Utilisateur/ Administrateur

DVIL SYSTEM 2026 — Documentation Administrateur

1. Introduction & Architecture Hybride

Le projet **DVIL** 2026 est une solution de contrôle d'accès sécurisée et résiliente. Contrairement aux versions précédentes, elle repose sur une architecture hybride :

- **Supervision Centrale (VM) :** Gestion globale des utilisateurs et archivage des logs via une base **MariaDB**.
- **Intelligence Locale (Raspberry Pi) :** Décision d'ouverture instantanée et stockage de secours via une base **SQLite** anonymisée. Cette structure permet de maintenir le service même en cas de coupure internet (mode dégradé).

2. Authentification et Sécurité des Accès

L'accès à l'interface de gestion est protégé par des standards de cybersécurité modernes :

- **JWT (JSON Web Token) :** Utilisé pour maintenir la session administrateur de manière sécurisée (stocké en cookie HTTPOnly).
- **Bcrypt :** Les mots de passe ne sont jamais stockés en clair dans la base MariaDB, mais sous forme de hashs cryptographiques.
- **Rôles :**
  - **Utilisateur (élève, stagiaire et intervenant) :** Accès à l'accueil et consultation de l'état de la serrure.

- **Prof :** Accès à l'historique des logs et consultation de l'état de la serrure
- **Admin :** Accès complet (Panel de gestion, création de badges, historique des logs) et consultation de l'état de la serrure.

3. Synchronisation et Endpoints API

Le système utilise une **API REST** pour faire dialoguer la **VM (Virtual Machine)** et la **Raspberry Pi Zero 2W**.

A. Supervision (VM Server)

- **GET `/api/users/sync` :** Utilisé par la l'**API locale** pour récupérer les dernières autorisations (Flux Descendant).
- **POST `/api/logs/sync` :** Reçoit les événements de la Pi pour les enregistrer dans MariaDB (Flux Montant).
- **GET `/admin/users` :** Interface CRUD (Créer, Lire, Mettre à jour, Supprimer) pour la base utilisateur.

B. Contrôle Local (Raspberry Pi Zero 2W)

- **POST `/api/local/verify` :** Route critique appelée par le clavier ou le lecteur NFC pour vérifier un code dans la base **SQLite locale**.
- **GET `/api/local/status` :** Indique l'état de santé du module IoT et sa connectivité avec la **VM (Virtual Machine)**.

4. Bases de Données

Le système gère la redondance des données :

- **MariaDB (VM) :** Contient les profils complets (Nom, Prénom, Badge, PIN, etc.). C'est la base de référence pour l'administrateur.
- **SQLite (Raspberry Pi Zero 2W) :** Base de données locale située dans `/var/lib/SQLite/.dvil_access.db`. Elle est **anonymisée** (UUID et PIN uniquement) pour protéger la vie privée en cas de vol de la carte SD.

5. Interfaces Matérielles (Hardware)

L'administrateur peut surveiller l'état des périphériques connectés en **I2C** :

- **Contrôleur Clavier (ATtiny1616) :** Gère la matrice de touches et les animations **LED** (Adresse 0x09).
- **Écran OLED (SSD1306) :** Affiche les instructions et le **feedback (retour)** en temps réel (Adresse 0x3C).
- **Lecteur NFC :** Interfacé pour la lecture des badges de type **MiFare**.

6. Traçabilité et Journaux d'incidents

Le système génère des logs multidimensionnels consultables dans le panneau admin :

- **api.log :** État des synchronisations et erreurs réseau.
- **clavier.log :** Historique des saisies et tentatives de force brute.
- **nfc.log :** Détection des badges autorisés ou inconnus.
- **serveur.log :** Actions effectuées par les administrateurs sur l'interface web.

7. **Maintenance & Bonnes Pratiques**
    - **Sauvegardes :** Il est conseillé d'exporter régulièrement la base **MariaDB** de la **VM (Virtual Machine)**.
    - **Mode dégradé :** En cas de maintenance serveur, vérifiez que le service local de la Raspberry Pi Zero 2W est actif via /api/local/status.
    - **Sécurité physique :** Le **bus I2C** doit être confiné dans le boîtier pour éviter tout **sniffing (interception)** via un analyseur logique.

DVIL SYSTEM 2026 — Manuel Utilisateur

1. **Accès par Badge NFC**
    - Présentez votre badge devant le lecteur.
    - **Accepté :** Accès accordé, la serrure se déverrouille.
    - **Refus :** Badge non reconnu ou expiré. Contactez l'administrateur.
2. **Accès par Clavier (Code PIN)**
    - Saisissez votre code à 4 chiffres sur le **clavier matriciel**.
    - Suivez les instructions sur l'**écran OLED**.
3. **Interface Web (Utilisateurs autorisés)**
    - Connectez-vous sur https://dvil.duckdns.org:8443/login avec vos identifiants.
    - Vous pouvez consulter si la porte est actuellement verrouillée ou déverrouillée.
    - Si vous avez le rôle Admin, vous pouvez avoir accès totale du serveur.

Annexe 4 : Mise en place de l'environnement virtuel Python

1. **Objectif de l'environnement virtuel**

    L'utilisation d'un **environnement virtuel (venv)** permet d'isoler les bibliothèques nécessaires aux applications Serrure **(app-serrure)** et Clavier **(app-clavier)**. Cette méthode garantit la stabilité du système en évitant les conflits de versions entre les modules **Python** globaux de la **Raspberry Pi Zero 2W** et ceux requis par le projet.

2. **Procédure d'installation technique**

    Les commandes suivantes ont été exécutées pour initialiser l'environnement de travail:

    - **Création de l'environnement virtuelle :** python3 -m venv [nom_de_venv]
    - **Activation de l'environnement :** source [nom_de_venv]/bin/activate
    - **Mise à jour de l'outil de gestion de paquets :** pip install --upgrade pip

- **Installation des dépendances critiques :** pip install flask requests paho-mqtt adafruit-circuitpython-ssd1306 adafruit-circuitpython-busdevice pillow python-dotenv

3. Inventaire des modules installés et rôles

Le tableau suivant récapitule les modules **IoT (Internet of Things)** utilisant les bibliothèques installées dans le **venv (Virtual Environnement)** et leur utilité pour l'application clavier :

Élément	Caractéristiques	Rôle Technique
Matrice Clavier	Membrane 12 touches (4x3)	Interface de saisie physique (Chiffres 0-9, *, #).
Microcontrôleur	**ATtiny1616** (Microchip)	Esclave I2C (**0x09**) assurant le balayage (scanning).
Bus de données	Liaison I2C (SDA / SCL)	Transmission série synchrone des touches vers la Pi.
Indicateurs LED	LED RGB (Neopixel)	Feedback visuel (Clignotement, changement de couleur).

Annexe 5 : Bases de données

Base de données (MariaDB - VM) :

Table	Colonne	Type	Propriétés	Description
users	id	INT	PK, AI	Identifiant unique de l'utilisateur
	nom_utilisateur	VARCHAR(50)	UNIQUE	Login de l'interface d'administration
	mot_de_passe	VARCHAR(255)	-	Empreinte (Hash) sécurisée Bcrypt
	role	VARCHAR(20)	-	Niveau de privilèges (admin/user)
	code_pin	VARCHAR(10)	-	Code de secours pour le clavier
	uid_badge	VARCHAR(50)	UNIQUE	Identifiant physique du tag NFC
	date_creation_badge	DATETIME	NULLABLE	Date de création

	date_expiration_badge	DATETIME	NULLABLE	Date de fin de validité de l'accès
access_history	id	INT	PK, AI	Identifiant de l'enregistrement
	user_id	INT	FK	Référence vers users.id
	source	VARCHAR(20)	-	Origine de l'accès (NFC, CLAVIER, WEB)
	action	VARCHAR(100)	-	Description (ex: "Ouverture porte")
	timestamp	DATETIME	DEFAULT NOW	Horodatage de l'événement
	status	VARCHAR(20)	-	Résultat (ACCÈS ACCORDÉ / REFUSÉ)
serrure_state	id	INT	PK	Identifiant de l'état (toujours début 1)
	state	VARCHAR(20)	-	État actuel (locked / unlocked)

Base de données (SQLite - Local) :

Table	Colonne	Type	Propriétés	Description
authorized_access	nfc_uid	TEXT	PK	UID du badge
	nfc_uuid	TEXT	PK	UUID du badge (anonymisation nominative).
	pin_code	TEXT	-	Code secret associé au badge
	date_creation	TEXT	-	Date de synchronisation VM -> Pi
	date_expiration	TEXT	NULLABLE	Validité temporaire en mode hors-ligne
tb_logs	id	INTEGER	PK, AI	Identifiant de l'incident local
	source	TEXT	-	Capteur utilisé (Clavier ou NFC)
	action	TEXT	-	Type d'événement détecté

	timestamp	DATETIME	-	Horodatage local de la Raspberry Pi
	status	TEXT	-	Statut avant synchronisation vers la VM

-> Code API Flask (Local - api-flask.py) : Voir le code source depuis le lien framagit ou GitLab dans la page 1 (chemin : /api-localPI/api-flask.py)

Exemples de requêtes SQL (MariaDB) :

- Vérifier les administrateurs :

```
SELECT nom, prenom, nom_utilisateur
FROM users
WHERE role = 'admin';
```

- Afficher les 5 dernières tentatives d'accès refusées :

```
SELECT timestamp, source, action
FROM access_history
WHERE status = 'REFUSÉ'
ORDER BY timestamp DESC
LIMIT 5;
```

Exemples de requêtes SQL (SQLite) :

- Vérifier si un badge est présent dans le cache local :

```
SELECT nfc_uid
DELETE FROM tb_logs;
WHERE nfc_uid = '84:BA:4C:16';
```

- Vider la table des logs après une synchronisation réussie :

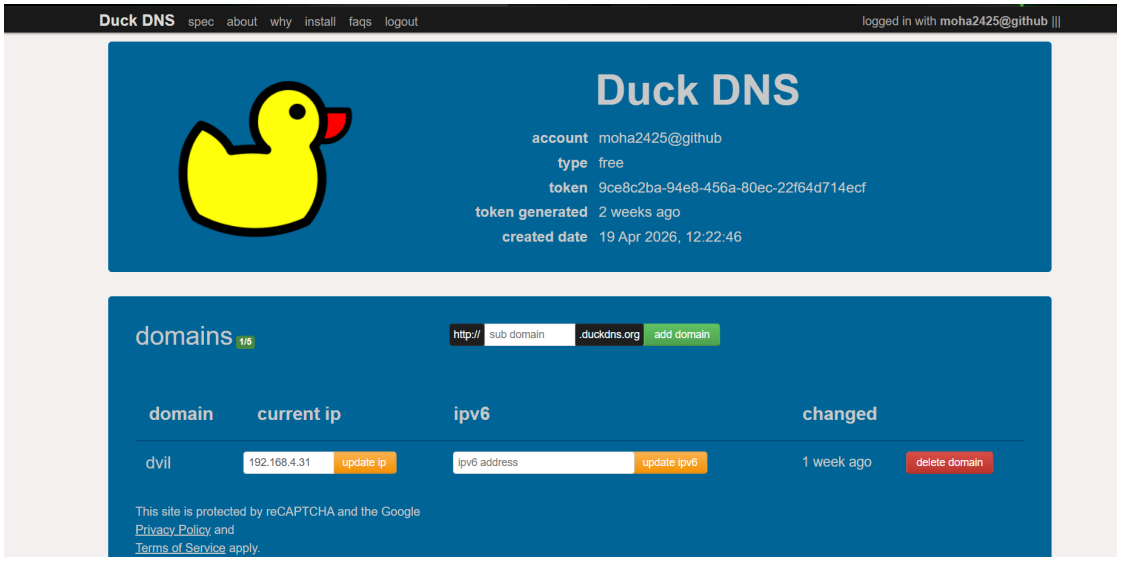
```
DELETE FROM tb_logs;
```

Annexe 6 : Infrastructure Réseau, DNS et Sécurisation HTTPS

1. Attribution du nom de domaine (DuckDNS)

- **Domaine configuré :** dvil.duckdns.org
- **lien vers le site DuckDNS :** https://www.duckdns.org

Page DuckDNS (vue de l'attribution de sous domaine) :



2. Résolution locale

Dans le fichier /etc/hosts de la **Raspberry Pi Zero 2W**, on ajoute cette ligne :

```
192.168.1.31      dvil.duckdns.org
```

Cela permet à l'API locale de traduire instantanément le nom de domaine en adresse IP sans dépendre d'Internet.

3. Serveur HTTP et Reverse Proxy (Nginx)

Avantages de cette configuration :

1. **Sécurité : Nginx** masque la signature du serveur web **Flask**.
2. **Gestion SSL : Nginx** prend en charge le chiffrement à la place de l'application.
3. **Stabilité :** Capacité à gérer plusieurs connexions simultanées de manière plus robuste.

Extrait de la configuration **Nginx** :

```
server {
    listen 80;
    server_name dvil.duckdns.org;

    location / {
        proxy_pass http://127.0.0.1:5000; # Redirection vers Flask
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

4. Sécurisation par Certificat SSL/TLS (HTTPS)

- **Outil utilisé** : Certbot (Let's Encrypt) .

Impact sur la configuration **Nginx (Port 443)** :

```
# --- 3. CONFIGURATION PRINCIPALE (L'application Flask) ---
server {
    listen 8443 ssl;
    server_name dvil.duckdns.org;

    # Configuration SSL
    ssl_certificate /etc/letsencrypt/live/dvil.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dvil.duckdns.org/privkey.pem;
```

Annexe 7 : Structure du Projet et Déploiement Docker

Arborescence complète de l'interface web conteneurisé :

```
flask_app/                          # Racine du projet Flask
├── app.py                          # Point d'entrée principal de l'application Flask
├── db_init
│   └── dvil_db.sql                 # Script SQL d'initialisation de la base de données
├── docker-compose.yml              # Configuration des services Docker
├── Dockerfile                      # Instructions de création de l'image Docker
├── logs                            # Dossier contenant les journaux d'exécution
│   ├── api.log                     # Logs des appels API
│   ├── clavier.log                 # Logs liés au clavier/code d'accès
│   ├── nfc.log                     # Logs des lecteurs NFC
│   ├── serrure.log                 # Logs du système de serrure connectée
│   └── serveur.log                 # Logs du serveur Flask
├── nginx
│   ├── default.conf                # Configuration du serveur Nginx
│   └── ssl                         # Dossier des certificats SSL/TLS
├── pentestlab
│   ├── challenges.py               # Gestion des challenges de cybersécurité
│   └── Dockerfile                  # Image Docker dédiée au laboratoire Pentest
├── requirements.txt                # Liste des dépendances Python du projet
├── static                          # Ressources statiques du site web
│   ├── css
│   │   ├── accueil.css             # Style de la page d'accueil
│   │   ├── admin.css               # Style de l'interface administrateur
│   │   ├── admin_logs.css          # Style de la gestion des logs
│   │   ├── admin_users.css         # Style de la gestion des utilisateurs
│   │   ├── challenges.css          # Style des pages de challenges
│   │   ├── index.css               # Style de la page principale
│   │   ├── login.css               # Style de la page de connexion
│   │   └── style.css               # Feuille de style globale
│   ├── js
│   │   ├── accueil.js              # Scripts de la page d'accueil
│   │   ├── admin.js                # Scripts du panneau administrateur
│   │   ├── admin_logs.js           # Gestion dynamique des logs
│   │   ├── admin_users.js          # Gestion dynamique des utilisateurs
│   │   ├── binary.js               # Fonctions liées aux challenges binaires
│   │   ├── login.js                # Gestion JavaScript de l'authentification
│   │   └── script.js               # Script JavaScript global
│   └── logo
│       └── logo.png                # Logo officiel de l'application
└── templates                       # Templates HTML Flask (Jinja2)
    ├── accueil.html                # Interface de la page d'accueil
    ├── admin.html                  # Interface administrateur
    ├── admin_logs.html             # Interface de consultation des logs
    ├── admin_users.html            # Interface de gestion des utilisateurs
    ├── challenges.html             # Interface des challenges
    ├── index.html                  # Page principale du site
    └── login.html                  # Page de connexion utilisateur
```

Toutes les commandes sont à exécuter depuis le répertoire flask_app de la **VM (Virtual Machine)**.

Lancement de l'infrastructure :

- docker-compose up -d --build

Arrêt et nettoyage :

- docker-compose down

Consultation des journaux (Logs) manuelle :

- docker-compose logs -f

Accès direct à la base de données (MariaDB) :

- docker exec -it dvil_db mariadb -u [nom_user_db] -p [mot_de_passe]

Code source de la configuration Docker (docker-compose.yml) :

```
services:
 # --- COUCHE DE PERSISTENCE ---
 db:
  image: mariadb:10.11
  container_name: dvil_db
  restart: always
  ports:
   - "3306:3306"
  environment:
   MYSQL_ROOT_PASSWORD: Admin
   MYSQL_DATABASE: dvil_db
   TZ: Europe/Paris
  volumes:
   - ./db_init:/docker-entrypoint-initdb.d  # Scripts SQL d'initialisation (7 utilisateurs)
   - dvil_db_data:/var/lib/mysql          # Persistance des données MariaDB
  networks:
   - dvil_net

 # --- COEUR APPLICATIF (API & PORTAIL) ---
 portal:
  build: .
```

```
  container_name: dvil_portal
  restart: always
  env_file: .env                    # Variables sensibles (SECRET_KEY, PI_IP)
  depends_on:
   - db
  volumes:
   - ./app.py:/app/app.py           # Code source dynamique
   - ./templates:/app/templates     # Interface Jinja2
   - ./static:/app/static           # Assets (CSS/JS/Logo)
   - ./logs:/app/logs               # Stockage des journaux d'audit
   - /etc/localtime:/etc/localtime:ro
   - /etc/timezone:/etc/timezone:ro
  environment:
   - TZ=Europe/Paris
  networks:
   - dvil_net

# --- SÉCURITÉ ET TERMINAISON SSL/TLS ---
nginx:
  image: nginx:alpine
  container_name: dvil_nginx
  restart: always
  ports:
   - "8443:8443"                    # Point d'entrée HTTPS unique
  volumes:
   - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
   - /etc/letsencrypt:/etc/letsencrypt:ro   # Certificats SSL DuckDNS
  depends_on:
   - portal
  networks:
   - dvil_net

 dvil_challenges:
  image: dvil/ctf-labs-2026:latest        # Environnement dédié aux laboratoires vulnérables
  container_name: dvil_challenges_lab
```

```
    restart: always
    ports:
      - "8081:8081"                # Accès Labo Facile #01
      - "8082:8082"                # Accès Labo Facile #02
      - "8083:8083"                # Accès Labo Facile #03
      - "8084:8084"                # Accès Labo Moyen #01
      - "8085:8085"                # Accès Labo Moyen #02
    networks:
      - dvil_net
    depends_on:
      - portal                     # Durcissement du conteneur (Hardening)

# --- CONFIGURATION RESEAU ISOLEE ---
networks:
  dvil_net:
    driver: bridge                 # Communication interne sécurisée via noms d'hôtes

# --- VOLUMES PERSISTANTS ---
volumes:
  dvil_db_data:                    # Conservation des données MariaDB au redémarrage
```

Contenu dans Dockerfile :

```
FROM python:3.9-slim

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

WORKDIR /app

RUN apt-get update && apt-get install -y gcc libmariadb-dev pkg-config && rm -rf
/var/lib/apt/lists/*

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

RUN touch pi_ip.txt

EXPOSE 5000

CMD ["gunicorn", "--worker-class", "eventlet", "-w", "1", "--bind", "0.0.0.0:5000",
"--access-logfile", "-", "--error-logfile", "-", "app:app"]
```

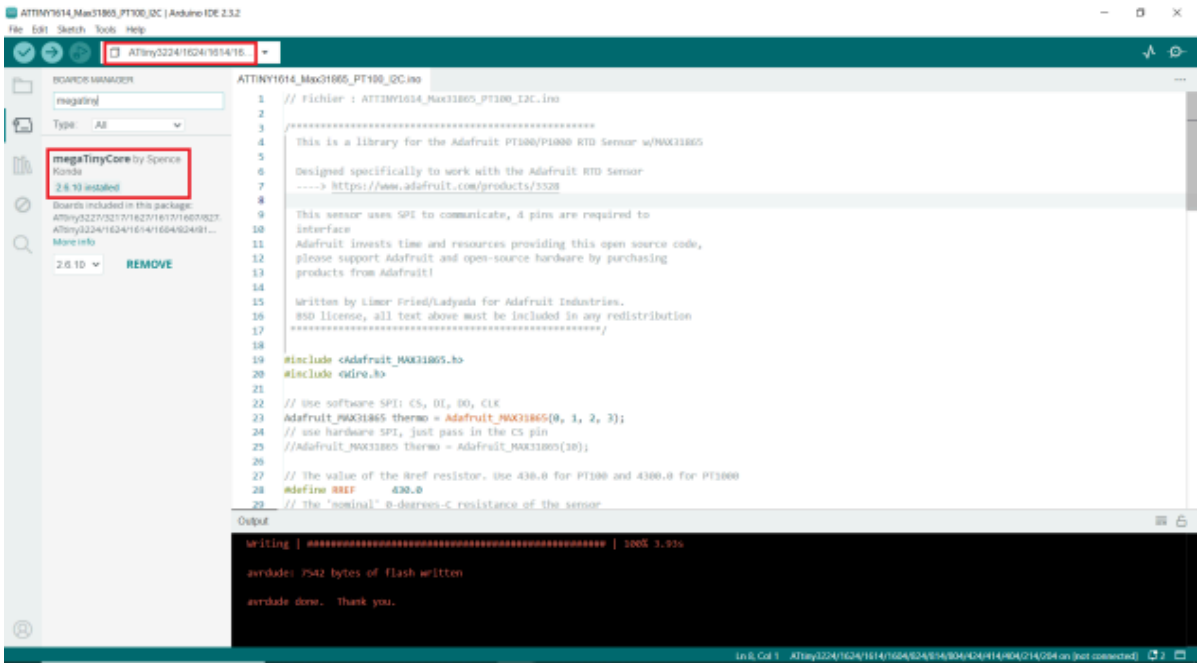
Annexe 8 : Guide de programmation du code clavier de l'Arduino à l'ATtiny1616

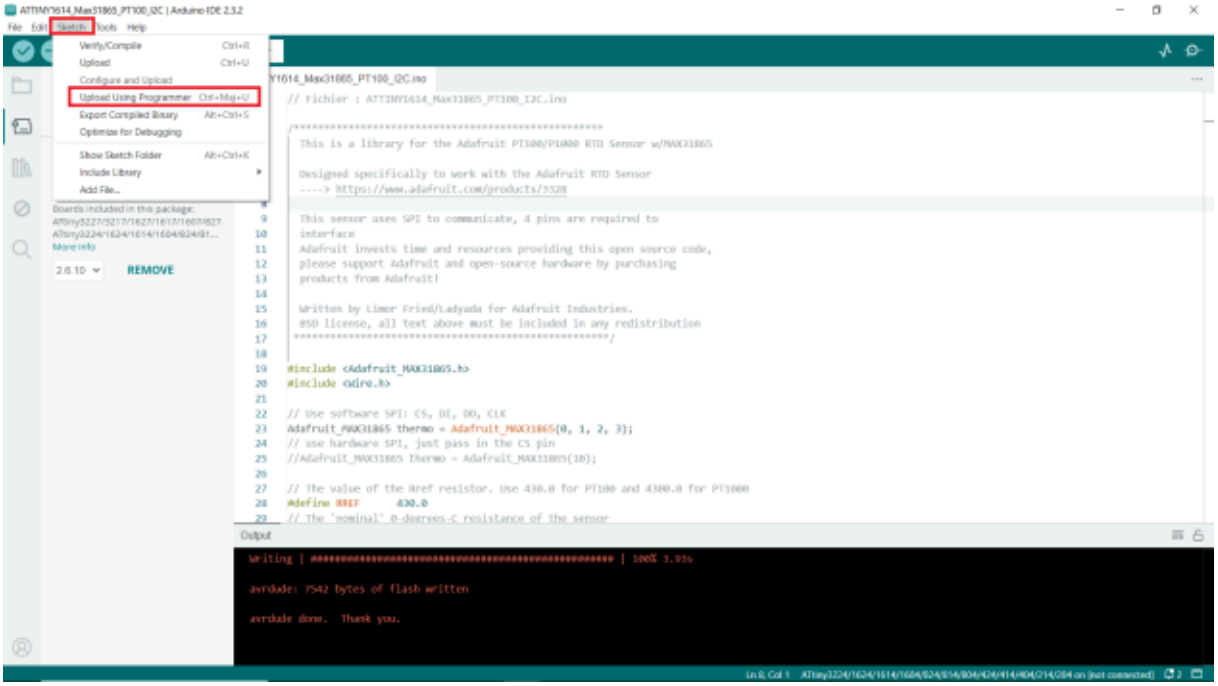
Le flashage du microcontrôleur **ATtiny1616** (utilisé pour la gestion du clavier et de la **LED**) a été réalisé selon deux méthodes distinctes, adaptées aux différentes phases du projet (développement et déploiement industriel).

Méthode 1 : Programmation professionnelle (PICkit 4 + MPLAB IPE)

Configuration et Utilisation de l'environnement **IDE Arduino** pour programmer un **ATTiny1616**

- Ajouter cette ligne dans le menu Préférences de l'**IDE Arduino** pour pouvoir télécharger les cartes **MegaTinyCore** de Spence Konde : **http://drazzy.com/package_drazzy.com_index.json**







Programmateur MPLAB PicKit4

Guide du programmeur :

MPLAB® PICkit™ 4 User's Guide

2.2.1 Standard ICSP™ Device Communication

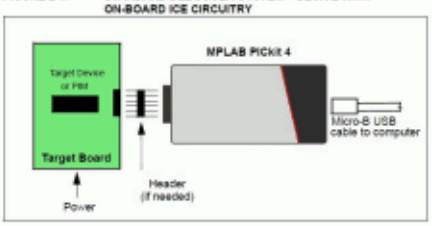
The debugger system can be configured to use standard ICSP communication for both programming and debugging functions.

Make sure to align the Pin 1 on the debugger to Pin 1 on the target. The programming connector can be inserted into either:

- A matching connector at the target, where the target device is on the target board (Figure 2-1).
- A standard adapter/header board combo (available as a Processor Pak), which is then plugged into the target board (Figure 2-2).

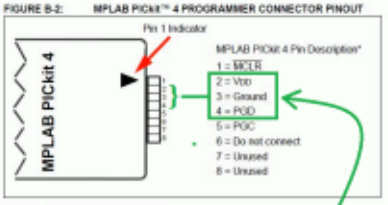
For more on standard communication, see Section B.4.1 "Standard Communication".

FIGURE 2-1: STANDARD DEBUGGER SYSTEM – DEVICE WITH ON-BOARD ICE CIRCUITRY



VTG = Voltage Target

FIGURE B-2: MPLAB PICkit™ 4 PROGRAMMER CONNECTOR PINOUT



B.4.2 Pinouts for Interfaces

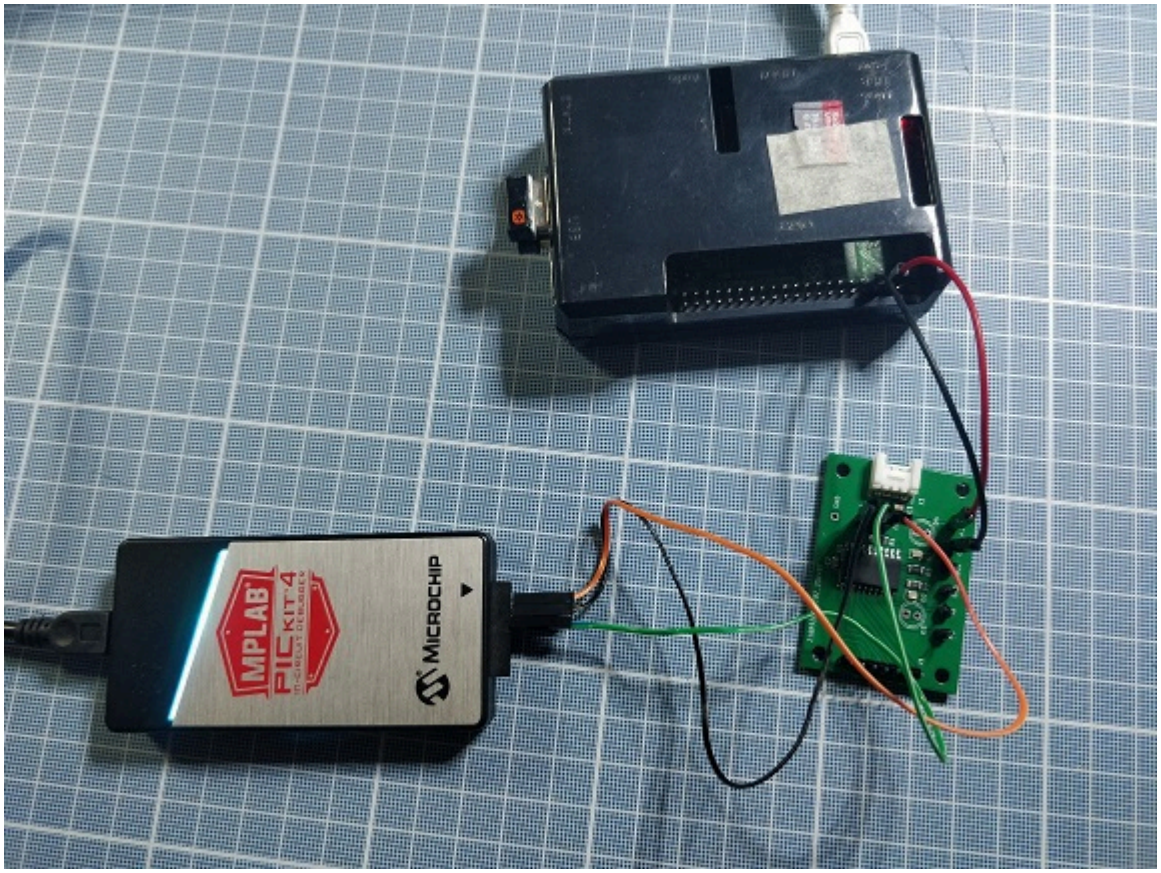
The programming connector pin functions are different for various devices and interfaces. Refer to the following pinout tables for debug and data stream interfaces.

Note: Refer to the data sheet for the device you are using as well as the application notes for the specific interface for additional information and diagrams.

TABLE B-5: PINOUTS FOR DEBUG INTERFACES

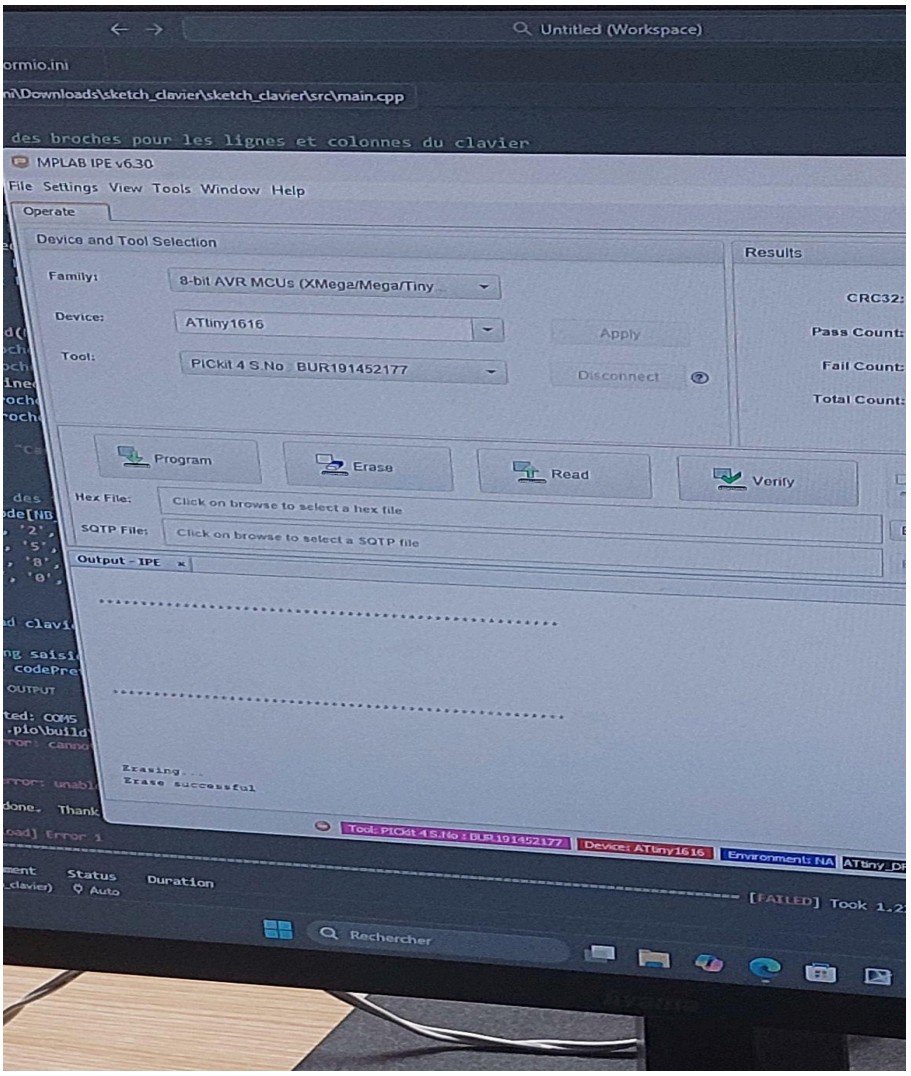
MPLAB PICkit 4	DEBUG									
Pin #	ICSP	MIPS EJTAG	CORTEX SWD	AVR JTAG	AVR ISP	UPDI	PDI	aW	dW	TPI
1	VPU/ MCLR									
2	VDD	VIO_REF	VTG	VTG	VTG	VTG	VTG	VTG	VTG	VTG
3	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND
4	PGD	TDO	SWO	TDO	MISO	UPDI	PDI_DAT	DATA		TPI_DAT
5	PGC	TCK	SWCLK	TCK	SCK					TPI_CLK
6	AUX	NRESET	NRST	RESET	RESET		PDI_CLK		dW	RST
7	TDI	TDI		TDI	MOSI					
8	TMS	TMS	SWDIO	TMS						
Module:		BB	BB	BB	SPI	UART	USART		UART	BB

Exemple de câblage pour flasher un programme d'une version différentes à l'ATtiny :



1. **Compilation :** Le programme clavier est compilé dans l'**IDE Arduino** en sélectionnant la carte **ATtiny1616** (via **megaTinyCore**). L'option « Exporter les binaires compilés » permet de générer le fichier de compilation **.hex**.
2. **Configuration MPLAB :** Le logiciel **MPLAB IPE** (Integrated Programming Environment) est utilisé pour configurer le programmateur **PICkit 4**.
3. **Connexion UPDI :** Le **PICkit 4** est relié à la puce via trois broches : VCC (5V), GND, et la broche **UPDI** (Unified Program and Debug Interface).
4. **Flashage :** Une fois la cible détectée sous **MPLAB**, le fichier hexadécimal est importé et "poussé" dans la mémoire Flash du microcontrôleur.

Cette image figure l'interface du logiciel du flashage du programme clavier :



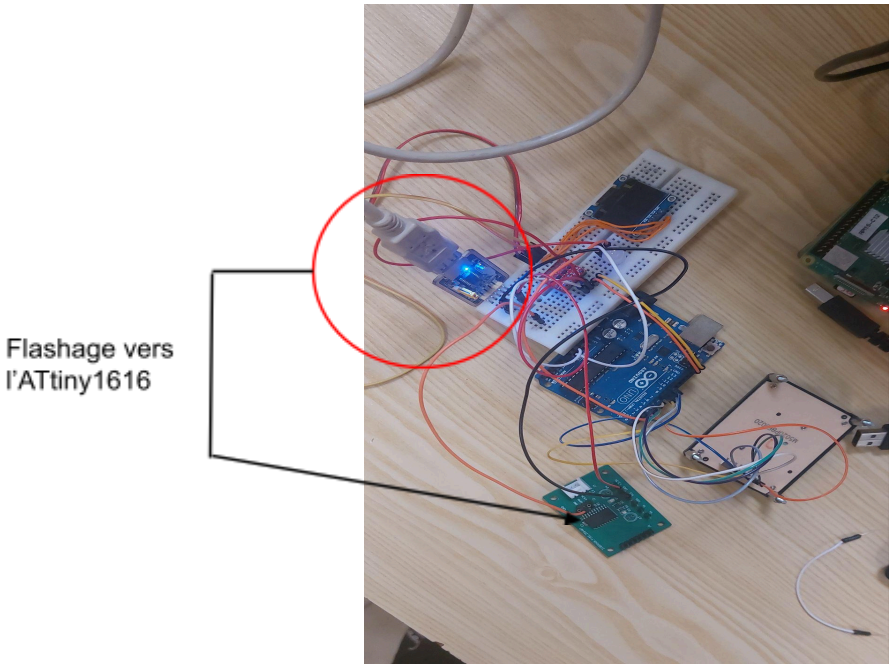
Méthode 2 : Programmation rapide (USB CH341 + Interface UPDI)

Pour les phases de tests rapides et de prototypage, une interface série simplifiée a été mise en œuvre. Elle utilise un adaptateur **USB-Série (CH341)** converti en programmateur **UPDI**.



USB-Série (CH341)

Exemple de montage en utilisant un breadboard :



1. **Configuration du pilote (Driver) et vérification système**

   Le bon fonctionnement de cette méthode dépend strictement de la version du pilote utilisé par le convertisseur USB-Série.

   - **Installation de version :** L'installation est réalisée via le logiciel officiel du constructeur (**CH341SER.EXE**) en prenant la version 2.7.
   - **Vérification (Gestionnaire de périphériques) :** Il est impératif de vérifier dans le gestionnaire de périphériques de Windows que le composant est reconnu sous le nom « **USB-SERIAL CH340 (COMx)** » .

2. **Génération du firmware (PlatformIO)**

   Au lieu de l'**IDE Arduino** classique, l'environnement **PlatformIO (depuis VScode)** a été utilisé pour sa gestion rigoureuse des bibliothèques.

   - **Action :** Exécution de la compilation (build) sur le projet (répertoire) "Clavier ATtiny1616".
   - **Résultat :** Génération du fichier binaire compilé à l'emplacement :

     ...\Projects\Clavier ATtiny1616\.pio\build\ATtiny1616\firmware.hex

3. **Optimisation matérielle (Le "Pontage")**

   Lors des premiers tests, des erreurs de type "**Timeout**" apparaissent.

   - **Solution technique :** Pour stabiliser le signal **UPDI** sur le fil unique de données, il a été nécessaire de **court-circuiter la résistance de couplage** entre les broches **TX** et **RX** de l'adaptateur **CH341**. Ce "**dérivation**" permet d'abaisser l'impédance et de garantir que les niveaux logiques sont correctement interprétés par l'**ATtiny1616** lors du flashage à haute vitesse.

4. **Flashage manuel en ligne de commande (PowerShell)**

   Pour un contrôle total sur les Fuses (bits de configuration interne) et la vitesse de transmission, le **flashage** est effectué via le script **Python** prog.py du package **megaTinyCore**.

   Exemple de commande utilisé :

```
PS C:\Users\melmadani> python -u "C:\Users\melmadani\AppData\Local\Arduino15\packages\megaTinyCore\hardware\megaavr\2.6.12\tools\prog.
py"  -t uart -u COM7 -b 57600 -d attiny1616 --fuses 0:0b00000000 2:0x01 6:0x04 7:0x00 8:0x00
"-fC:\Users\melmadani\Documents\PlatformIO\Projects\Clavier ATtiny1616\.pio\build\ATtiny1616\firmware.hex" -a write -v
```

   - **-t uart / -u COM7 : Utilisation de l'interface série sur le port identifié (CH341).**
   - **--fuses : Configuration critique des registres internes de la puce.**
   - **-a write -v : Écriture et vérification (V) du succès de l'opération.**

Attention : veuillez bien mettre le bon port COM, le bon chemin du fichier firmware.hex ainsi que le chemin du script de flashage avec CH341 et la bonne version de l'ATtiny dans ce cas ATtiny1616.

Incident rencontré	Cause identifiée	Solution technique appliquée
Échec de détection "Target not found"	Version du pilote CH341 obsolète.	Installation du pilote constructeur v2.7 via le gestionnaire de périphériques.
Erreur de synchronisation (Timeout)	Impédance trop élevée sur le bus UPDI (résistance).	Mise en place d'une nouvelle résistance à 0 Ω.
Fichier .hex introuvable	Erreur de compilation ou mauvais répertoire.	Build complet sous PlatformIO et vérification du dossier .pio/build.

Tests précédentes avant le passage final du programme clavier

Images de câblage version **Arduino UNO** clavier en **C++** avec la carte Arduino :

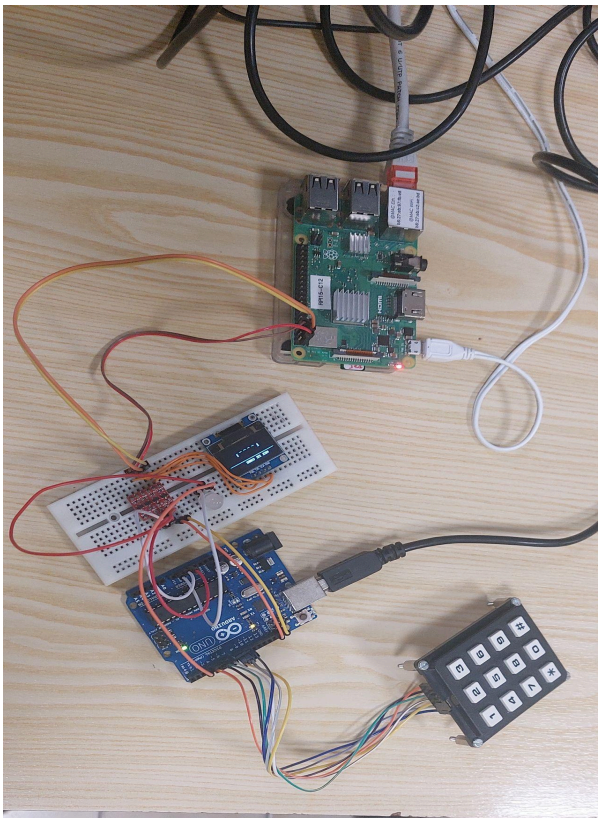


Schéma du câblage de test Arduino version 1 (LEDs vert et rouge) :

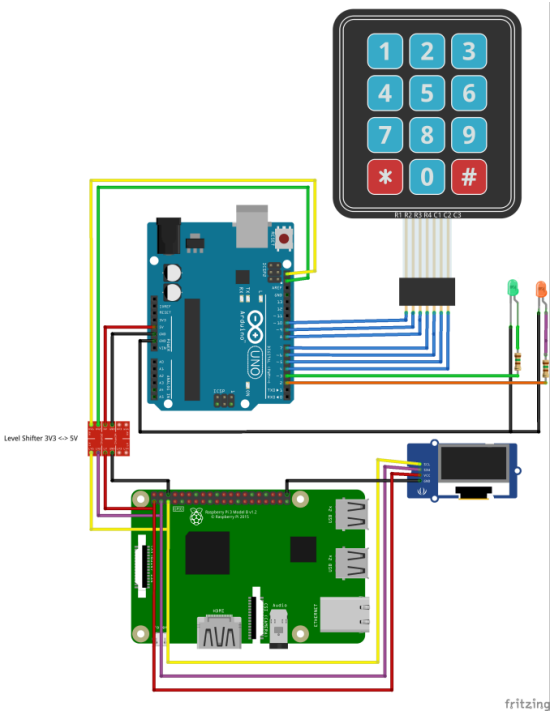


Schéma du câblage de test Arduino version 2 (LED NeoPixel Multicolore):

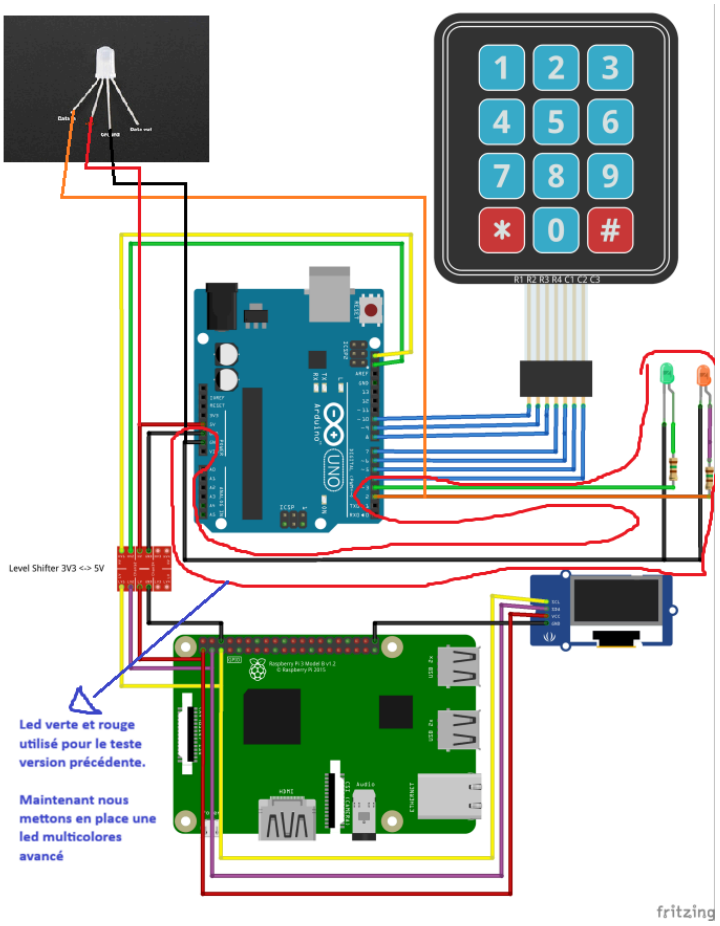


Image du montage du projet avant la version finale :

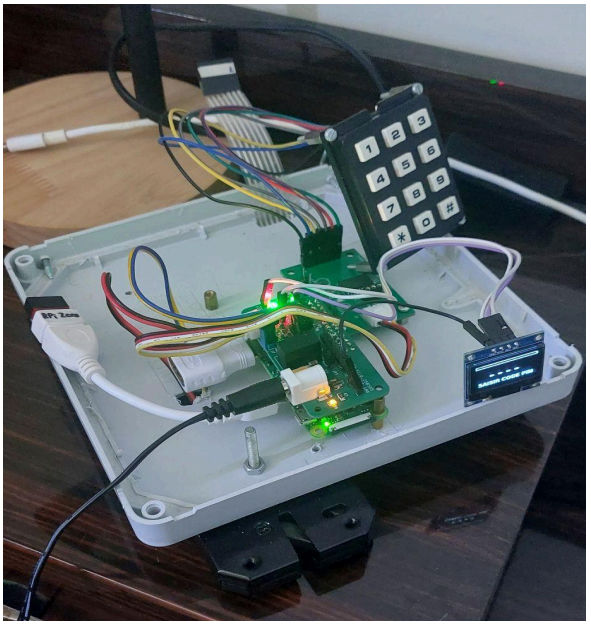
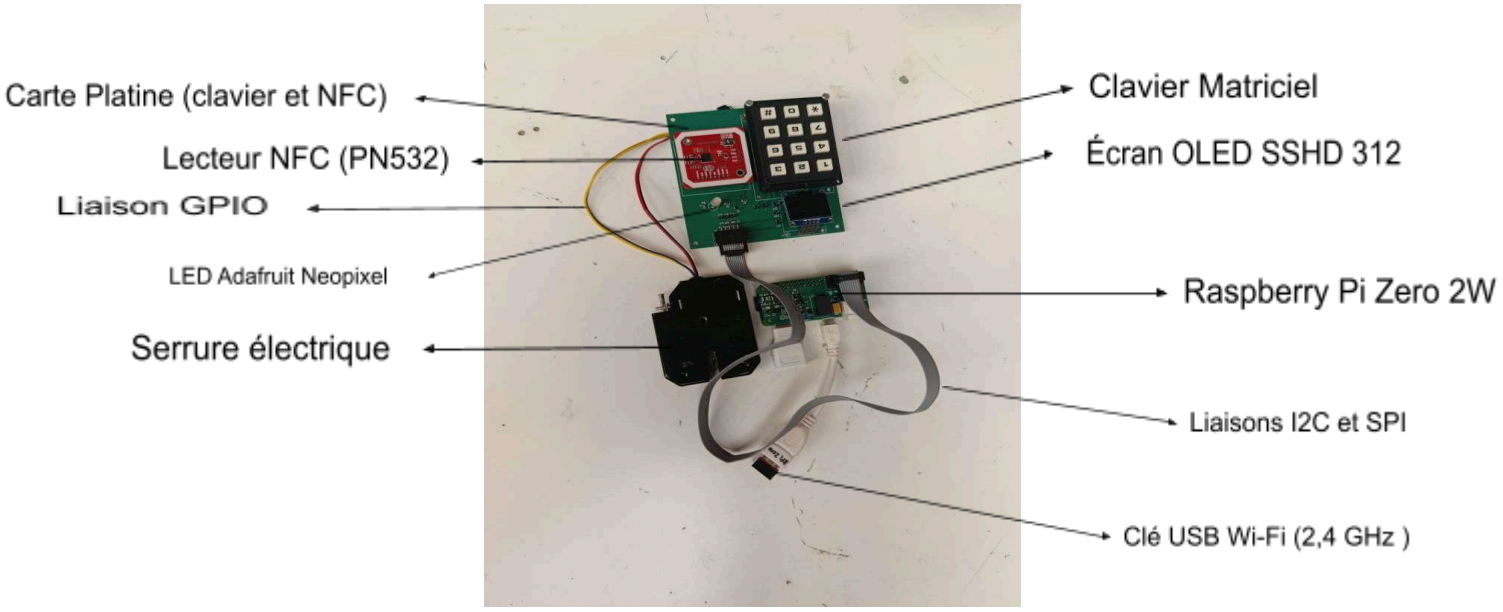


Image du montage finale du projet **DVIL** (**cartes mise à jour par les anciens et cette version lui manque le boitier qui sera terminé pour l'année prochaine)** :



Rappel : La Raspberry Pi Zero 2W utilisait une antenne Wi-Fi d'elle même soudé en utilisant un connecteur UFL pour le brancher mais malheureusement le connecteur s'est déchiré le fait de l'utilisation alternative d'une clé USB Wi-Fi suivi d'un adaptateur micro-USB -> USB. Ci-dessous l'exemple d'antenne précédente.



Antenne Wi-Fi de base



Connecteur UFL

Ci-dessous la structure du boîtier designé pour le montage du projet :

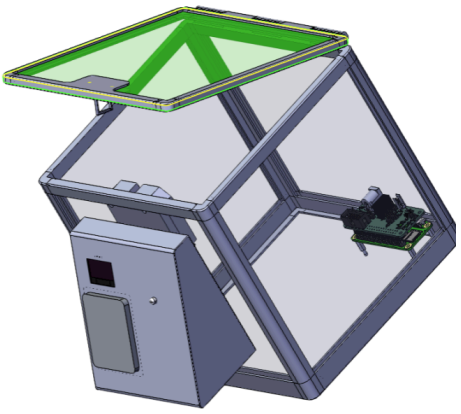


Illustration de la maquette du design des différentes affichages de l'écran OLED :



Annexe 9 : Fiche recette et audit de sécurité :

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	Plateforme DVIL (Damn Vulnerable Internet Lock)	Code de la campagne de test :	REC-DVIL-2026-V1.0
Technicien 1 : Technicien 2 : Technicien 3 : Technicien 4 :	Mohamed EL MADANI Haitam EL KADMIRI Abdel AMDOUNI	Date :	03/05/2026

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Validation du chiffrement des communications web (Protocole HTTPS / TLS)
Objectif du scénario :	S'assurer que l'interface web force l'utilisation du protocole sécurisé HTTPS, que les certificats sont reconnus, et que les données des requêtes entre le client (navigateur) et l'application (Flask) soient transité de manière chiffrée et illisibles par l'attaquant qui essaie de les analyser par la technique de **Man on the middle** (l'homme du milieu).

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

- Le serveur web Nginx est configuré pour écouter sur les ports 80 (HTTP) à l'intérieur localement et 443 (HTTPS) à l'extérieur du serveur.
- Un certificat SSL/TLS valide et un attribution de domaine est installé et lié à l'application.
- Le poste client dispose d'un navigateur web et d'un analyseur de trafic réseau (type Wireshark ou Burp Suite).
- Le conteneur qui contient le service web est en cours d'exécution.

EXÉCUTION DU TEST

Description courte :	**1 - Test de redirection :** Saisir l'URL non sécurisée http://[IP_du_serveur] dans le navigateur client. **2 - Test du certificat :** Cliquer sur l'icône de sécurité (le cadenas) à côté de l'URL dans la barre d'adresse. **3 - Test d'interception (Sniffing) :** Lancer une capture de trames réseau **(Wireshark)**, soumettre un formulaire de connexion (login) et de visualiser les données chiffrées correspondant aux requêtes **HTTP**.
Résultats attendus :	**Pour l'étape 1 :** Le serveur doit interdire la connexion en clair et rediriger automatiquement l'utilisateur vers https://[IP_du_serveur]. **Pour l'étape 2 :** Le navigateur doit afficher une connexion chiffrée (protocole TLS 1.2 ou 1.3 actif). **Pour l'étape 3 :** L'analyse réseau doit montrer uniquement du trafic chiffré (paquets "Application Data" en TLS). Le contenu du formulaire saisi ne doit jamais apparaître en clair dans la capture.

BILAN

La redirection HTTP vers HTTPS est automatique et fonctionnelle. L'analyse des trames réseau avec Wireshark confirme que toutes les commandes envoyées à la serrure transitent de manière chiffrée. La confidentialité du système est assurée, aucun attaquant ne peut intercepter l'ordre d'ouverture sur le réseau local.

REMARQUES

- Le certificat utilisé pour des petits projets est auto-signé (génère une alerte de sécurité sur le navigateur).
- Pour un déploiement réel en entreprise, il faudra installer un certificat signé par une autorité reconnue (comme Let's Encrypt) en lui attribuant un nom de domaine gratuit (comme DuckDNS).

CONCLUSION

VALIDE	X	NON VALIDE	

(Mettre une croix dans la case correspondante)

Fiche d'audit de sécurité.

Réf. FAS867470

Projet	***Auteur***	***Date***
Projet DVIL	EL MADANI Mohamed	08/05/2026

Partie de projet

Développement de l'interface clavier et interface web

Désignation	***Risque***	***Solution***
Exposition du port API	**Attaque par Force Brute / DDoS : :** DuckDNS rend l'URL publique. N'importe qui peut tenter d'inonder l'API de requêtes pour saturer le serveur	**Filtrage IP (Whitelist)** : Configurer le pare-feu (UFW) de la VM pour n'accepter les requêtes sur le port 443 **que** si elles proviennent de l'IP de la Raspberry Pi.
Trame I²C/SPI	Malgré le HTTPS sur le réseau, un attaquant physique peut sniffer les touches du clavier via les broches SDA/SCL de la Pi	Implémenter un algorithme de permutation des octets entre l'ATtiny et la Pi ou chiffrer la trame I2C avec une clé partagée
Accès physique à la carte SD	Un individu volant la Raspberry Pi peut lire les jetons d'accès et les codes PIN directement sur la carte SD	Utiliser une extension comme *SQLCipher* pour chiffrer le fichier .db ou activer le chiffrement intégral de la partition de données
Authentification (API & Web)	Un attaquant peut tester des milliers de combinaisons de codes PIN ou de mots de passe sans être interrompu, finissant par trouver l'accès	Intégrer *Flask-Limiter* pour bloquer une IP après 3 tentatives infructueuses et imposer un délai d'attente exponentiel entre chaque essai
Accès de Maintenance (SSH)	Si un attaquant accède au réseau local, il peut tenter de se connecter en SSH à la Raspberry Pi via le port 22 par défaut.	Désactiver l'authentification par mot de passe au profit de clés SSH (RSA), changer le port 22 et installer *Fail2Ban* pour bannir les IP suspectes

Synthèse

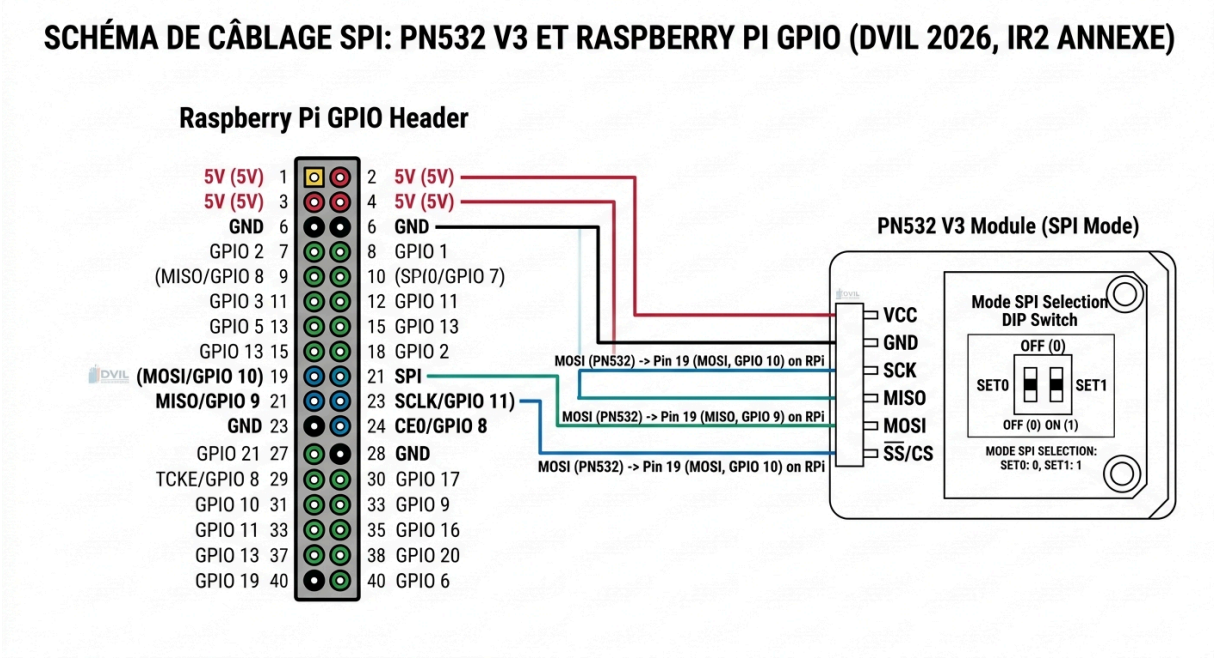
L'architecture DVIL 2026 présente une base solide grâce au chiffrement des flux (**HTTPS**) et des mots de passe (**Bcrypt**). Cependant, l'audit révèle deux vulnérabilités majeures à corriger : l'exposition du bus **I2C** au sniffing physique (codes PIN circulant en clair) et l'absence de **Rate Limiting** sur l'API, rendant le système vulnérable aux attaques par force brute via DuckDNS.

Prochainement une mise en développement du plateforme des challenges type (CTF) pour s'entraîner à du pentesting et la cybersécurité.

Attention : les failles seront volontaires pour l'apprentissage mais devront être supprimées ou désactivées en production.

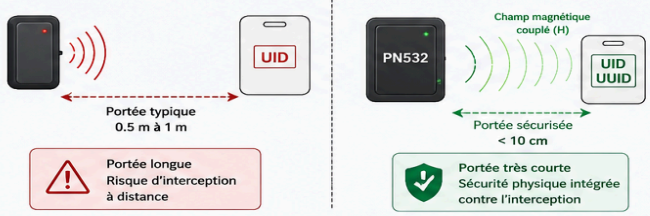
ANNEXES IR 2

SCHÉMA DE CÂBLAGE SPI: PN532 V3 ET RASPBERRY PI GPIO (DVIL 2026, IR2 ANNEXE)



1) POSITIONNEMENT TECHNOLOGIQUE : RFID vs NFC

Le NFC (13,56 MHz) est une forme évoluée de la RFID Haute Fréquence.
Il se distingue par sa portée très courte et ses capacités de sécurité.



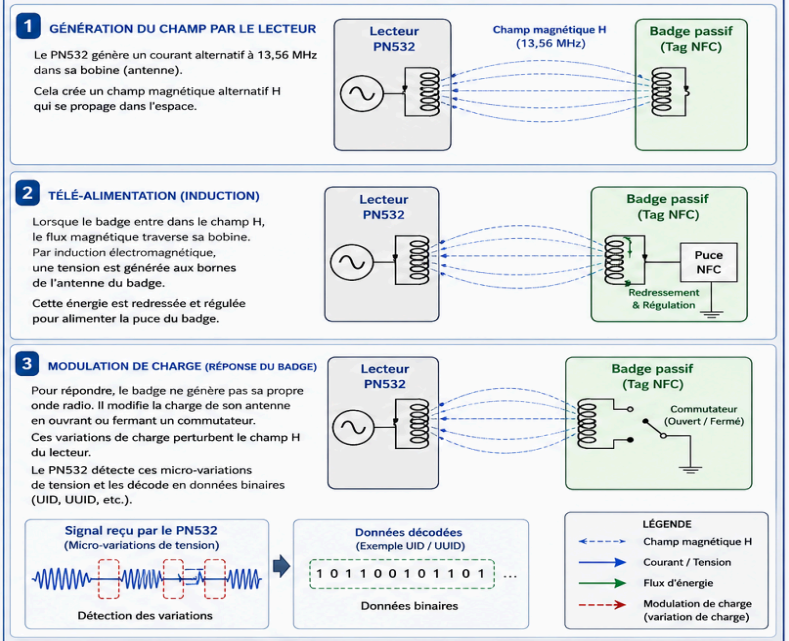
COMPARAISON TECHNIQUE

Critère	RFID 125 kHz	NFC 13,56 MHz
Fréquence de travail	125 kHz	13,56 MHz
Portée typique	0.5 m à 1 m	< 10 cm
Couplage	Inductif (faible)	Inductif (fort)
Débit de données	Faible (~1 à 5 kbps)	Élevé (106 à 424 kbps)
Communication	Généralement unidirectionnelle (lecteur → tag)	Bidirectionnelle (lecteur ↔ tag)
Sécurité	Faible (UID lisible)	Élevée (authentification, chiffrement)
Écriture de données	Limitée ou inexistante	Oui (écriture sécurisée)
Exemple d'application	Contrôle d'accès basique	Contrôle d'accès sécurisé (DVIL) UUID v4, authentification
Norme / Protocole	ISO 11784/11785	ISO/IEC 14443 A – Mifare Classic
Sécurité des données	Aucune / Faible	Crypto1 (clé secrète) Chiffrement possible

★ Le NFC 13,56 MHz offre une portée réduite (< 10 cm) et des fonctions de sécurité avancées, idéales pour un contrôle d'accès fiable et protégé comme dans le projet DVIL.

2) PRINCIPE PHYSIQUE DU SIGNAL : COUPLAGE INDUCTIF & MODULATION

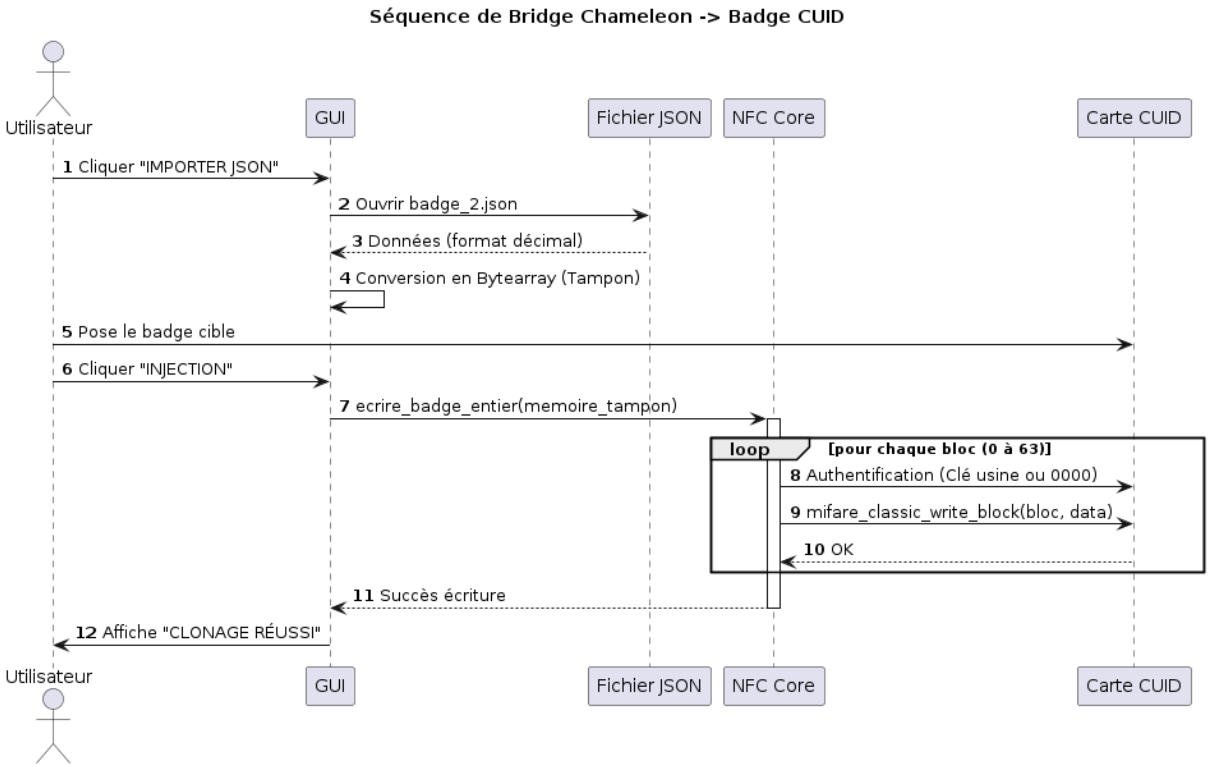
La communication NFC repose sur le couplage inductif à 13,56 MHz entre le lecteur (PN532) *et le badge passif, suivant 3 étapes physiques clés.*

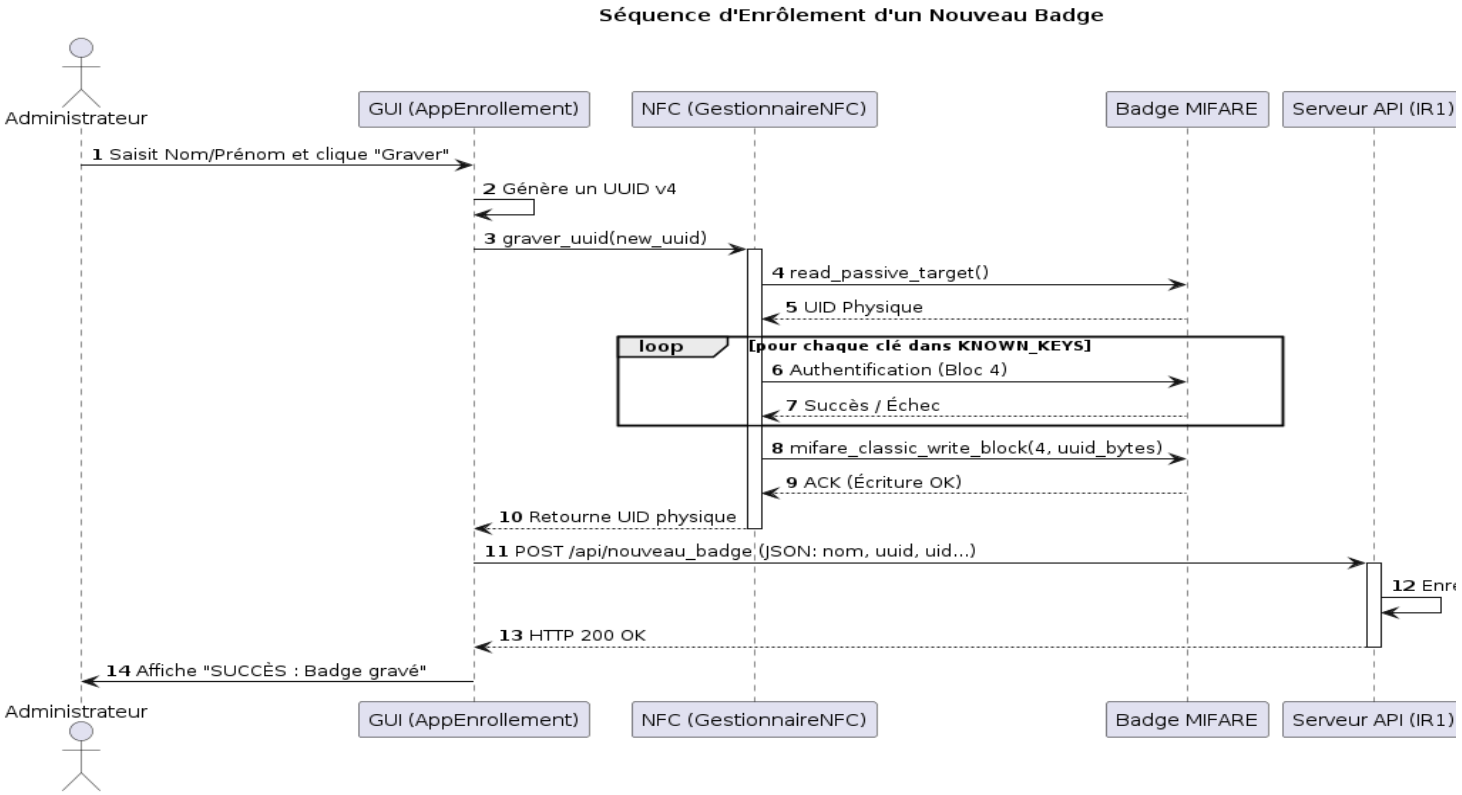


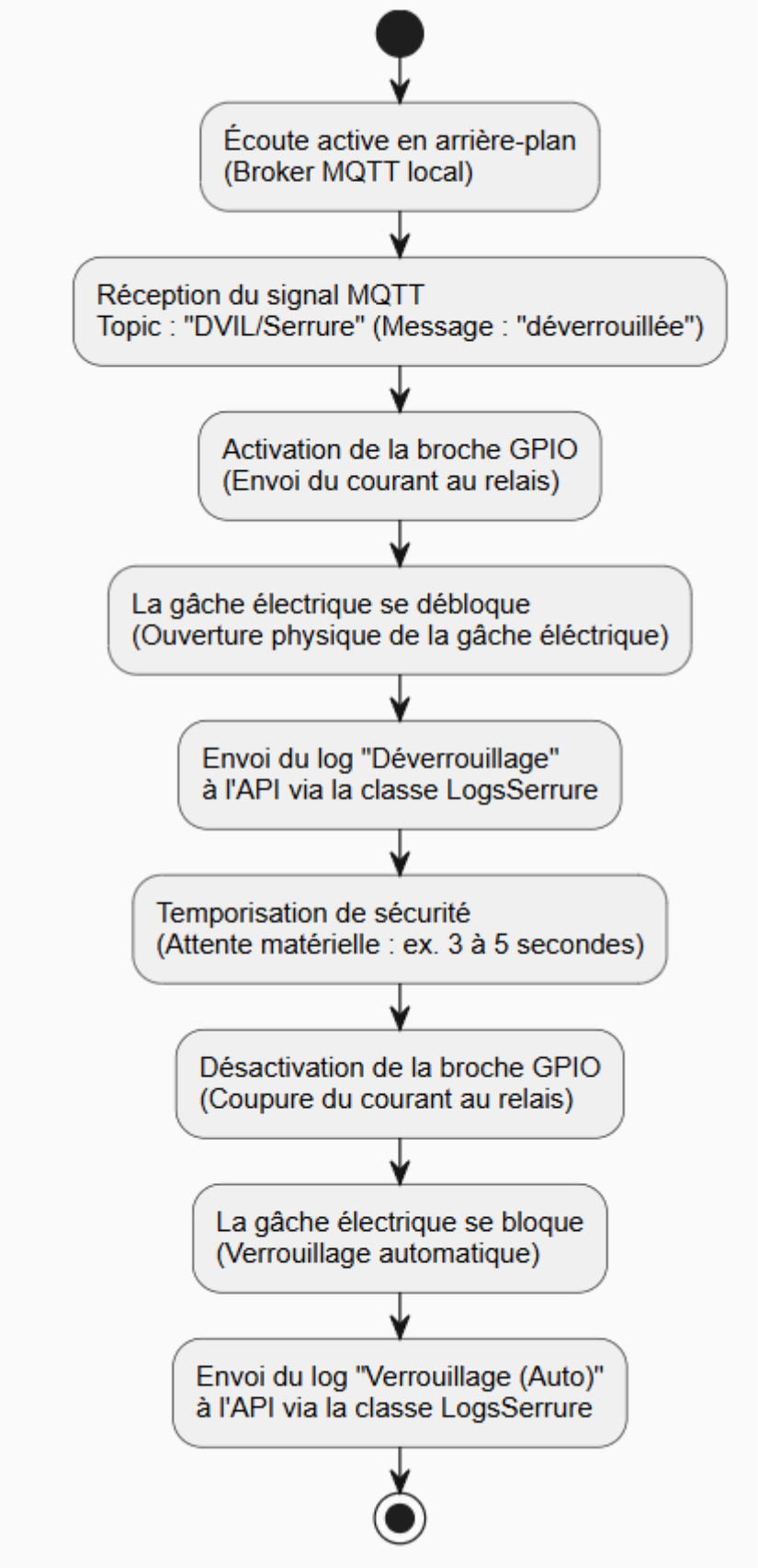
RÉSUMÉ DU PRINCIPE PHYSIQUE

Le NFC utilise le couplage inductif pour l'énergie et la donnée.
Le badge passif "parle" en modifiant sa charge, et le lecteur "écoute" ces perturbations pour lire les données.
Ce principe garantit une communication sécurisée à très courte portée (< 10 cm).

Annexe 1 : import de fichier json pour cloner un badge

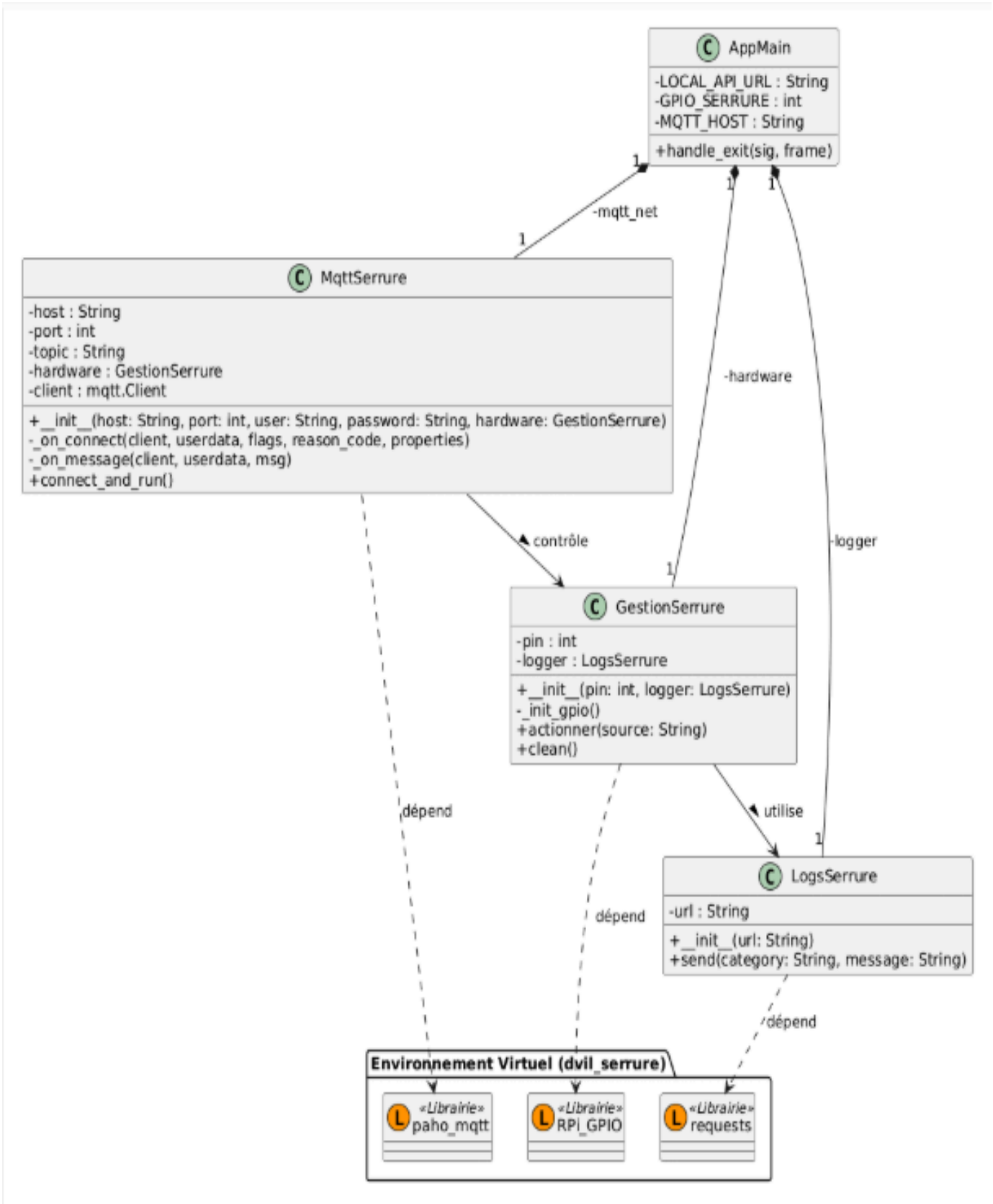





Écoute active en arrière-plan
(Broker MQTT local)
Réception du signal MQTT
Topic : "DVIL/Serrure" (Message : "déverrouillée")
Activation de la broche GPIO
(Envoi du courant au relais)
La gâche électrique se débloque
(Ouverture physique de la gâche éléctrique)
Envoi du log "Déverrouillage"
à l'API via la classe LogsSerrure
Temporisation de sécurité
(Attente matérielle : ex. 3 à 5 secondes)
Désactivation de la broche GPIO
(Coupure du courant au relais)
La gâche électrique se bloque
(Verrouillage automatique)
Envoi du log "Verrouillage (Auto)"
à l'API via la classe LogsSerrure

la serrure

serrure


AppMain
-LOCAL_API_URL : String
-GPIO_SERRURE : int
-MQTT_HOST : String
+handle_exit(sig, frame)
-mqtt_net
MqttSerrure
-host : String
-port : int
-topic : String
-hardware : GestionSerrure
-client : mqtt.Client
+__init__(host: String, port: int, user: String, password: String, hardware: GestionSerrure)
-_on_connect(client, userdata, flags, reason_code, properties)
-_on_message(client, userdata, msg)
+connect_and_run()
-hardware
-logger
contrôle
GestionSerrure
-pin : int
-logger : LogsSerrure
+__init__(pin: int, logger: LogsSerrure)
-_init_gpio()
+actionner(source: String)
+clean()
dépend
utilise
dépend
LogsSerrure
-url : String
+__init__(url: String)
+send(category: String, message: String)
dépend
Environnement Virtuel (dvil_serrure)
«Librairie»
paho_mqtt
«Librairie»
RPi_GPIO
«Librairie»
requests

Fiche d'audit de sécurité

Projet BTS DVIL 2026 – Serrure connectée NFC/RFID

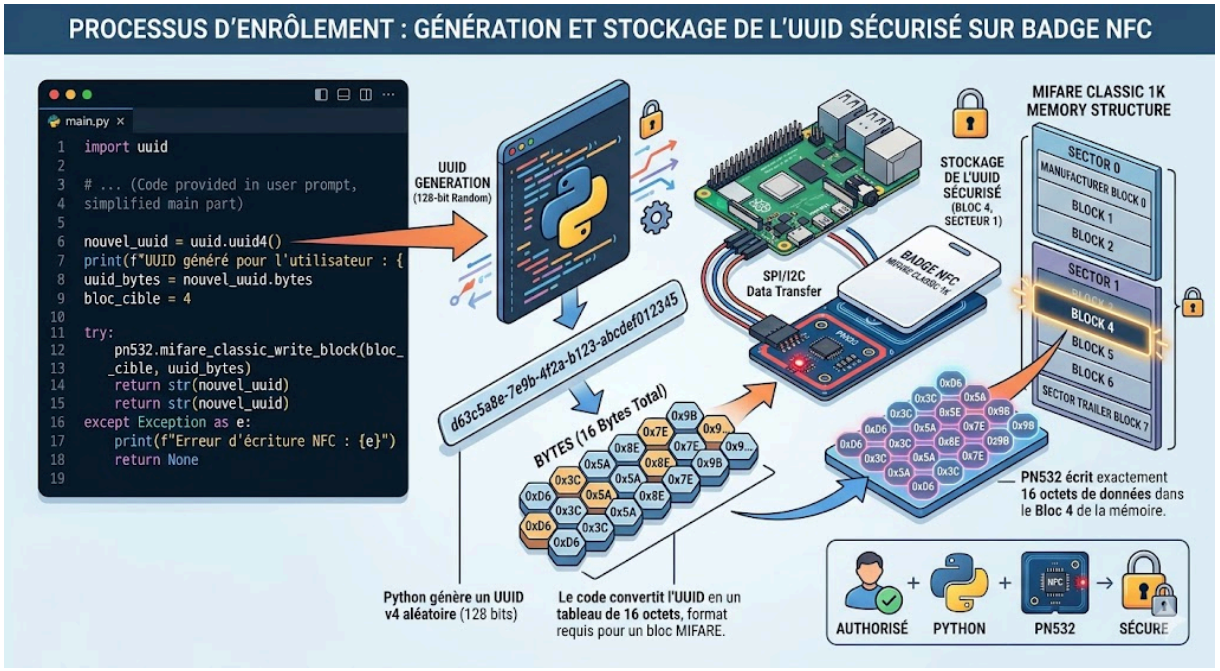
Auditeur : Haitam EL KADMIRI (Responsable IR2)

ID	Désignation	Risque	Solution	Statut
IR2-01	Clonage d'UID physique	L'identifiant matériel d'usine du badge NFC peut être copié facilement, permettant une usurpation d'identité et un accès non autorisé.	Remplacement de l'UID physique par un UUID v4 logique de 128 bits stocké dans une zone mémoire sécurisée.	■ RÉSOLU
IR2-02	Clés cryptographiques par défaut	Les badges utilisent des clés MIFARE d'usine connues publiquement et cassables en quelques secondes.	Modification des clés A et B du secteur 1 avec diversification cryptographique personnalisée.	■ RÉSOLU
IR2-03	Blocage de l'IHM (Déni de Service local)	L'interface Python se bloque lors des lectures NFC à cause d'un conflit d'accès au bus SPI.	Implémentation du multithreading Python avec séparation des traitements NFC.	■ RÉSOLU

Synthèse :

L'audit de sécurité du projet DVIL 2026 met en évidence plusieurs vulnérabilités critiques liées à l'authentification NFC et à la stabilité de l'interface de contrôle. Les principales failles identifiées ont été corrigées grâce à des mécanismes de sécurisation avancés : remplacement des UID physiques par des UUID v4 sécurisés, diversification des clés cryptographiques MIFARE et mise en place d'un système multithread pour stabiliser l'IHM Python.

Annexe 2 : UUID



UUID v4 (Universally Unique Identifier)

Identifiant unique universel basé sur des nombres aléatoires

UUID v4 = aléatoire

Les UUID v4 sont générés aléatoirement.
La probabilité de collision est extrêmement faible.

STRUCTURE D'UN UUID v4

Version 4 — Les 4 bits de poids fort sont fixés à 0100

Variant (RFC 4122) — Les 2 bits de poids fort sont fixés à 10

`550e8400 - e29b - 41d4 - a716 - 446655440000`

32 bits | 16 bits | 16 bits | 16 bits | 48 bits

COMMENT ÇA FONCTIONNE ?

1. **Génération aléatoire** — 122 bits sont générés aléatoirement.
2. **Application des règles** — Les bits de version (0100) et de variant (10) sont définis.
3. **UUID v4 prêt** — Identifiant unique, au format xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx (y = 8, 9, a ou b)

BON À SAVOIR — Avec 122 bits aléatoires, il y a ~5.3 × 10^{36} combinaisons possibles. Même en générant des milliards d'UUID par seconde, les collisions sont pratiquement impossibles.

À QUOI ÇA SERT ?

- **Clés primaires en base de données** — Évite les conflits et les fusions.
- **Identifiants distribués** — Unique sans coordination centrale.
- **Tokens, sessions, références** — Sécurisés et imprévisibles.

CARACTÉRISTIQUES

- 122 bits aléatoires
- Unique avec une probabilité très élevée
- Aucune signification temporelle
- Non séquentiel
- Recommandé pour la plupart des cas généraux

EXEMPLE

`550e8400-e29b-41d4-a716-446655440000`

Version 4 (4) — Variant ($a = 1010_2$)

Matrice de Sécurité MIFARE (Access Bits)

Analyse du bloc de contrôle & verrouillage des droits

TABLE DES CONDITIONS D'ACCÈS

Extrait documentation constructeur (Data Blocks)

Access Bits			KEY A (Porte)		KEY B (Admin)	
C1	C2	C3	Read	Write	Read	Write
0	0	0	A|B	A|B	A|B	A|B
0	1	0	A|B	never	A|B	never
1	0	0	A|B	never	A|B	KEY B
0	1	1	A|B	KEY B	A|B	KEY B
1	1	1	A|B	never	A|B	never

* C1, C2, C3 sont paramétrés via les octets 6, 7 et 8 du Sector Trailer.

STRUCTURE & RÉSULTAT

Byte 0-5	Byte 6-8	Byte 10-15
KEY A	ACCESS BITS	KEY B

⚠ État par défaut (Usine)

La configuration usine **0-0-0** permet à n'importe qui possédant la Clé A de lire ET d'écrire sur le badge. L'identifiant est totalement vulnérable.

`C1=0, C2=0, C3=0`

État Sécurisé (Ma Solution)

La configuration **1-0-0** impose de nouvelles règles physiques :

✓ **Lecture :** Autorisée pour l'authentification (Key A|B).
🔒 **Écriture :** Strictement verrouillée, réservée à l'administrateur (Key B).

`C1=1, C2=0, C3=0`

FICHE RECETTE

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	**Plateforme NFC DVIL - Contrôle d'accès sécurisé**	Code de la campagne de test :	REC-DVIL-01
Technicien 1 : Technicien 2 : Technicien 3 : Technicien 4 :	MOHAMED EL MADNI HAITAM EL KADMIRI ABDEL AMDOUNI	Date :	25/05/26

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Authentification renforcée (UUID) et ouverture de la serrure (Fail-Secure)
Objectif du scénario :	Valider que le passage d'un badge enrôlé déclenche bien la lecture de l'UUID dans la mémoire protégée, la validation par l'API, l'envoi de l'ordre MQTT et l'activation temporisée du relais de la gâche électrique.

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

La carte Raspberry Pi est sous tension et connectée au réseau.
Le lecteur PN532 est branché et opérationnel sur le bus SPI.
Les micro-services Python (`app-python-nfc` et `app-python-serrure`) tournent en tâche de fond.
Le broker MQTT (Mosquitto) et l'API Flask locale sont actifs.
Un badge MIFARE Classic 1K préalablement enrôlé (contenant un UUID v4 dans son Bloc 4) est à disposition.

EXECUTION DU TEST

Description courte :	Présenter le badge enrôlé devant le lecteur PN532 Observer les logs dans le terminal pour l'authentification. Observer l'état matériel de la gâche électrique (GPIO 17)
Résultats attendus :	Le lecteur lit l'UUID sécurisé et le transmet. L'API valide l'accès et publie le message "ACCEPT" sur MQTT. Le terminal affiche "Accès Autorisé". La gâche s'ouvre (claquement du relais) pendant la durée de la temporisation, puis se referme automatiquement.

BILAN

Le test est nominal. La communication MQTT est fluide et sans latence apparente. La gâche s'active exactement comme prévu par la logique Fail-Secure

REMARQUES

L'interface de supervision distante (sur la VM) reçoit bien la requête de mise à jour HTTP et affiche le changement d'état "Porte Ouverte" puis "Porte Fermée" en temps réel.

CONCLUSION

VALIDE	X	NON VALIDE	

(Mettre une croix dans la case correspondante)

CAISSON FINAL

