

BTS CIEL

Épreuve E6.2 – Session 2025

Dossier de revue n°4

Projet DVIL

Damn Vulnerable Internet Lock



https://github.com/lysandre84/PROJECT_DVIL_25.git

https://framagit.org/vialette/project_dvil_25.git

- VIALETTE Lysandre (IR)
- BRUSSET Abel (IR)
- RIVAREL Marius (ER)



Table des matières

Table des matières	2
1. Introduction & Contexte.....	4
Partie commune.....	4
1.1 Expression du besoin.....	6
2. Spécifications et Contraintes	6
2.1 Contraintes Techniques.....	6
3. Organisation et Répartition des Tâches	6
4. Schémas, Diagrammes et Documentation Technique	7
5. Validation et Suivi du Projet.....	8
6. Conclusion.....	8
7. Terminologie	9
Partie Personnelle VIALETTE Lysandre (IR 1).....	12
8. Objectifs	13
8.1 Rôle dans l'équipe et positionnement du travail	14
8.2 Résultats attendus (cahier des charges personnel).....	14
9. Planification	15
9.1 Diagramme de Gantt.....	15
10. Analyse Technique	16
10.1 Choix des technologies pour l'interface Web et le clavier	16
10.2 Contraintes techniques (matérielles, logicielles, performances)	16
11. Réalisation Interface Web, journalisation	17
11.1 Implémentation de l'interface Web (API) VOIR CODE ANNEXE 3	17
11.2 Diagrammes et représentation graphique.	23
11.3 Développement du multi environnements	26
11.4 [API] Implémentation Challenge 1.....	28
11.5 [API] Implémentation Challenge 2.....	30
11.6 Endpoint vulnérable.....	31
11.7 [API] Implémentation challenge 3.....	35
11.8 Endpoint vulnérable.....	37
12. Développement de l'application Clavier.....	40
12.1 Faille vulnérable sur le clavier.....	40
12.2 PC Frame.....	41
12.3 Diagramme et représentation graphique.....	42
13. Sécurisation et bonnes pratiques.	46

14.	Conclusion et perspectives.....	47
14.1	Bilan personnel.....	47
Partie Personnelle BRUSSET Abel (IR 2)		48
14.2	Les étapes à réaliser.....	49
14.3	Scénario.....	49
15.	La serrure.....	50
15.1	Test de la serrure sur Arduino.....	50
15.2	Test de la serrure sur Raspberry Pi 4.....	52
16.	Connexion avec le broker MQTT.....	54
17.	Le module NFC.....	55
Partie Personnelle RIVAREL Marius(ER)		60
18.	Introduction :.....	61
19.	Analyse du fonctionnement des bus de transmission :.....	62
20.	Le bus I2C :.....	62
20.1	Utilisation de l'I2C :.....	62
20.2	Structure d'une trame I2C :.....	63
21.	Début du projet :	65
22.	1ère réunion de projet :	68
23.	Schéma KICAD :	69
24.	Compte rendu.....	75
ANNEXES		80
25.	L'interface Web (API) Python ANNEXE [1].	81
26.	Dictionnaire des Endpoint ANNEXE [2].	84
27.	Base de données.....	103
28.	Documentation Utilisateur/ Administrateur ANNEXE [3].	104
29.	Fiche d'audit de sécurité ANNEXE [4].....	112
30.	Branchement du projet.....	113

Partie commune

1. Introduction & Contexte

Dans le cadre du Brevet de Technicien Supérieur « Cybersécurité, Informatique et Réseaux, Electronique » (BTS CIEL) au lycée Alphonse-Benoit, la maîtrise de la cybersécurité appliquée aux objets connectés est un enjeu majeur. Or, les supports actuels sont limités à des simulations logiciel, ce qui ne permet pas aux étudiants de confronter leurs connaissances à des cas pratiques réels.

Afin de combler ce manque, nous avons lancé la conception d'un dispositif IoT volontairement vulnérable : le projet DVIL – *Damn Vulnerable Internet Lock*.

Composants

Raspberry-pi Zero

AT-Tiny

Gâche électrique

Clavier Matriciel

Récepteur NFC

Moyen d'entrée

Badge NFC

Clavier Matriciel

Interface Web



Les étudiants manquent en effet d'un support matériel pour observer — et exploiter — les failles réelles d'une serrure connectée : mots de passe faibles, communication non chiffrée, mise à jour firmware non sécurisée, etc.

DVIL fournira ce terrain de jeu sécurisé : une petite plateforme physique, documentée et modulaire, permettant d'apprendre à attaquer puis à renforcer un objet connecté comme on le ferait en conditions professionnelles.

1.1 Expression du besoin

La partie commune définit les besoins initiaux, partagés par tous les membres du projet :

- **Fonctionnalités essentielles :**
 - Commande de la serrure via plusieurs interfaces : clavier matriciel, badge NFC, interface web.
 - Journalisation et suivi des actions (logs détaillés des tentatives et opérations).
 - Gestion des communications entre les modules (via MQTT et bus I²C/SPI)

2. Spécifications et Contraintes

2.1 Contraintes Techniques

- **Matérielles :** Compatibilité entre le microcontrôleur (Raspberry Pi | ATtiny), les modules NFC, le clavier matriciel, et la serrure.
- **Logiciels :** Utilisation de frameworks éprouvés (Flask, SQLAlchemy, etc.), avec une attention particulière portée à la sécurité des communications (HTTPS/TLS, authentification par JWT).
- **Performances :** Réactivité de la commande de la serrure (temps de réponse inférieur à 1 seconde) et gestion efficace des logs.
- **Budget alloué :** 150 euros.

3. Organisation et Répartition des Tâches

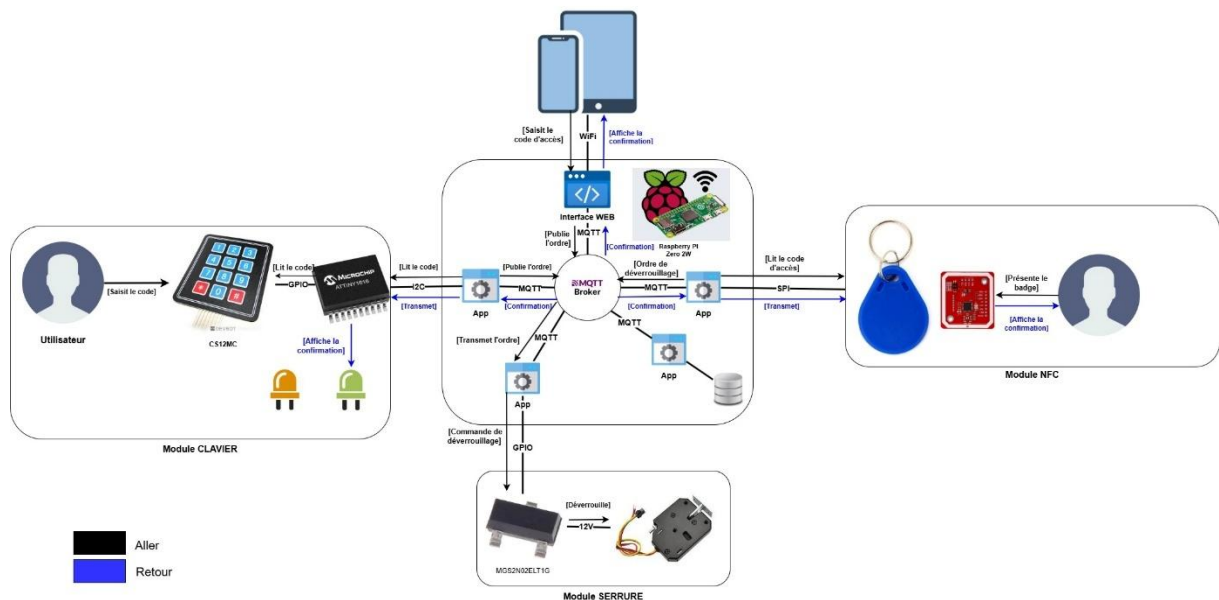
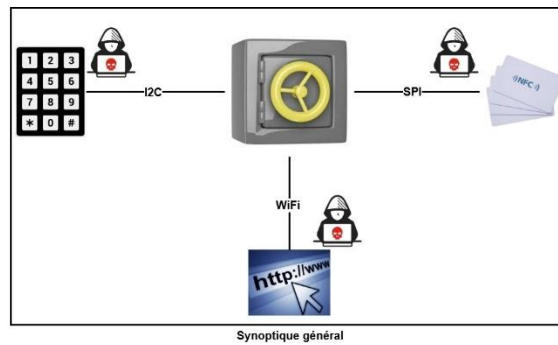
La **partie commune** définit la répartition des tâches entre les différentes équipes :

- **Étudiant IR1 :** Développement de l'interface web (API REST), gestion du clavier et journalisation.
- **Étudiant IR2 :** Intégration de l'interface NFC, commande de la gâche électrique via MQTT.
- **Étudiant ER :** Conception électronique, câblage, et mise en place de la carte (Hat) pour l'ensemble du système.

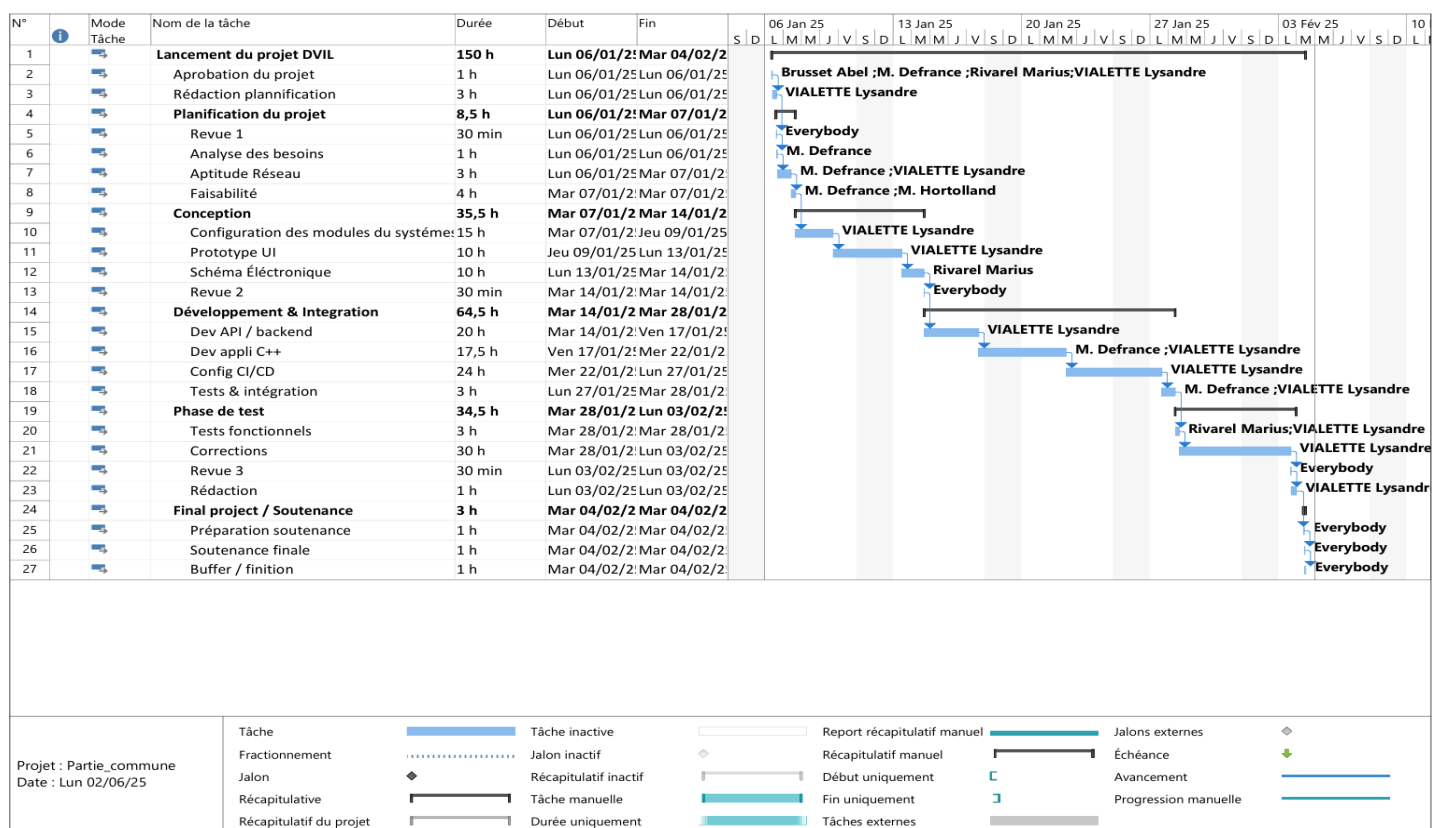
Chaque équipe doit collaborer étroitement pour assurer la cohérence du projet et la communication entre les différents modules via des protocoles standards (MQTT, I²C, SPI).

4. Schémas, Diagrammes et Documentation Technique

Le schéma ci-dessous présente l'architecture de la plateforme DVIL, faisant ressortir la répartition des modules, leurs interactions et le flux complet — de l'authentification utilisateur jusqu'au contrôle physique de la serrure. Cette vue d'ensemble offre une synthèse claire du fonctionnement global du système.



Ci-dessous un diagramme de Gant qui illustre nos tâches effectuées durant le projet :



5. Validation et Suivi du Projet

Pour garantir un pilotage rigoureux, la partie commune combine :

- **une planification détaillée** (diagramme de Gantt et découpage fin des tâches) pour optimiser le temps ;
- **des grilles de conformité** alignées sur le référentiel BTS CIEL, appliquées à chaque jalon pour valider les livrables ;
- **des réunions périodiques avec comptes-rendus** assurant la traçabilité des décisions et le respect des exigences initiales.

6. Conclusion

Ce projet DVIL a été une véritable aventure collective qui nous a permis de concrétiser nos connaissances en cybersécurité, électronique et développement IoT. Chacun a pu mettre en avant ses compétences, découvrir de nouveaux aspects techniques et apprendre des autres. L'entraide et la collaboration ont été essentielles pour surmonter les difficultés et avancer efficacement, malgré les contraintes de temps et de budget. Nous sommes fiers du travail accompli : cette expérience nous a montré l'importance de la sécurité "by design" et de la rigueur documentaire. Nous remercions aussi nos encadrants pour leur soutien précieux tout au long de ce parcours.

7. Terminologie

MOTS TECHNIQUES

Définitions

DVIL	Serrure connectée volontairement vulnérable, conçue pour apprendre la cybersécurité.
IoT (Internet of Things)	Objets du quotidien connectés à internet pour échanger des données.
Serrure connectée	Système de verrouillage pouvant être contrôlé à distance via un dispositif électronique.
API REST (Application Programming Interface)	Interface permettant aux applications de communiquer via des requêtes HTTP.
Interface Web	Page ou application accessible dans un navigateur pour interagir avec un service.
MQTT (Message Queuing Telemetry Transport)	Protocole léger de messagerie utilisé surtout pour les objets connectés (IoT).
Broker MQTT	Serveur qui gère et distribue les messages MQTT entre différents appareils.
Clavier matriciel	Clavier dont chaque touche est détectée à l'intersection d'une ligne et d'une colonne.
Module NFC (Near Field Communication)	Capteur pour échanger des données sans contact sur une courte distance (badges, cartes).
Bus I ² c (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).
Raspberry Pi	Mini-ordinateur peu coûteux, populaire pour les projets d'électronique et de programmation.
Arduino	Carte électronique programmable, pratique pour prototype et robotique.
Gâche électrique	Mécanisme électromécanique permettant d'ouvrir ou bloquer une porte à distance.
Relais	Interrupteur commandé électriquement pour activer ou désactiver un circuit.

Journalisation (Logs)	Bobine électromagnétique qui produit un mouvement mécanique (ex. serrure électrique).
Authentification JWT (JSON Web Token)	Enregistrement d'événements pour le suivi et l'analyse des actions dans un système.
TLS/HTTPS	Système pour prouver son identité à l'aide d'un jeton numérique temporaire.
XSS (Cross-Site Scripting)	Protocole chiffré protégeant les échanges sur le web contre l'interception de données.
Injection SQL	Faible où un attaquant peut injecter du code malveillant dans une page web.
Path Traversal	Attaque où l'on insère du code SQL non prévu pour accéder ou modifier une base de données.
Injection de templates (SSTI)	Faible permettant d'accéder à des fichiers sensibles en manipulant les chemins d'accès.
Désérialisation non sécurisée (Pickle)	Injection de code dans un moteur de rendu (comme Jinja2) côté serveur.
Challenge de sécurité	Exploitation de la conversion d'objets en données pour exécuter du code malveillant.
Firmware	Exercice où l'on doit identifier et exploiter des failles pour s'entraîner à la cybersécurité.
Cross-compilation	Logiciel embarqué dans un composant électronique (microcontrôleur, etc.).
Python/Flask	Compilation d'un programme sur un type de machine pour l'exécuter sur un autre processeur.
C**/Qt	Langage et framework web pour créer rapidement des serveurs et APIs.
Diagramme de Gantt	Langage et bibliothèque pour développer des applications graphiques multiplateformes.
UML (Unified Modeling Language)	Planning visuel montrant l'évolution des tâches d'un projet dans le temps.
Diagramme de séquence	Langage de modélisation décrivant la structure et le fonctionnement d'un système.
Schéma de routage	Schéma illustrant comment des objets ou composants interagissent au fil du temps.
	Plan de câblage sur une carte électronique, montrant comment les composants sont reliés.

Master/Slave (Maître/Esclave)	Canal où l'envoi et la réception de données s'effectuent à tour de rôle, pas simultanément.
Driver	Mode de contrôle où un maître initie la communication, et les esclaves répondent.
Payload	Programme qui permet au système d'exploiter un composant matériel ou périphérique.

Partie Personnelle
VIALETTE Lysandre
(IR 1)

8. Objectifs

Dans le cadre du projet D-VIL (Damn Vulnerable Internet Lock), mon rôle en tant qu'étudiant IR1 s'articule autour des interfaces de commande et la journalisation des événements de la serrure. Mes objectifs principaux sont :

1. MISE EN ŒUVRE DE DEUX INTERFACES DE COMMANDE.

- Interface Clavier :

Concevoir et implémenter la gestion des saisies calvier (4x3), valider les codes via I²C ou GPIO, et assurer la sécurité (limitation des tentatives, prévention des contournements).

- Interface Web / API :

Développer une API REST sécurisée permettant le contrôle de la serrure, avec gestion de l'authentification (session ou JWT) et chiffrement TLS.

2. COMMUNICATION ET ORCHESTRATION VIA MQTT

Configurer le broker MQTT (Mosquitto) et gérer les échanges entre les modules (ex : DVIL/Serrure, DVIL/Status) pour relier interface web et clavier matriciel. Implémenter la couche logicielle assurant la synchronisation entre les commandes et la serrure

3. JOURNALISATION (LOGS) DES ACTIONS

Développer un système de logs enregistrant les événements critiques (accès, échecs, changement d'état) au format exploitable (fichier, base de données, syslog), avec informations de date, utilisateur, IP, et une interface de consultation rapide.

4. SECURITE ET VULNERABILITES

Mettre en place les failles prévues pour le challenge (ex : mot de passe en dur, communication non chiffrée) en les localisant précisément dans le code/documentation, et proposer des pistes d'amélioration sans les corriger (pour l'aspect pédagogique).

5. TESTS ET VALIDATIONS

Définir des scénarios de tests (réactivité, qualité des logs), effectuer des tests d'intégration et analyser la fiabilité globale (pertes MQTT, redondance).

6. DOCUMENTATION ET LIVRABLES

Rédiger une documentation claire et versionnée (installation, dépendances, utilisation), centralisée sur Github ou Framagit pour une gestion efficace des versions et une traçabilité optimale via Git ou wiki.

8.1 Rôle dans l'équipe et positionnement du travail

L'équipe du projet D-VIL (Damn Vulnerable Internet Lock) se compose de trois membres, chacun en charge d'un domaine spécifique :

- IR1 VIALETTE Lysandre (moi-même)
Responsable du développement de l'Interface clavier, de l'interface web/API et de la journalisation des événements.
- IR2 BRUSSET Abel
En charge de l'interface NFC, du contrôle direct de la gâche électrique et de la communication entre le microcontrôleur et le Raspberry Pi.
- ER (RIVAREL Marius)
Responsable de la conception électronique, câblage des bus (I2C, SPI, GPIO), de l'alimentation et du dimensionnement matériel.

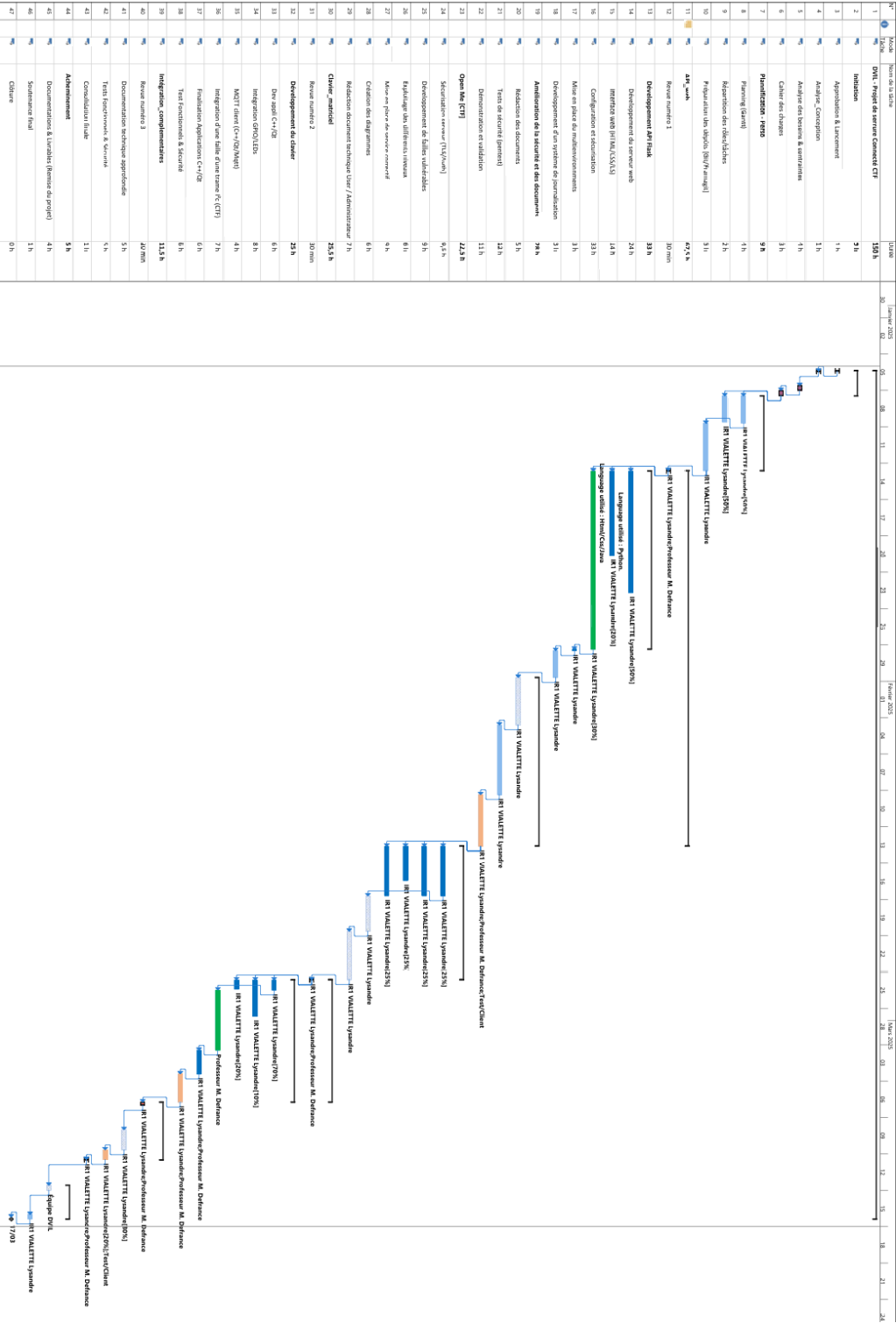
8.2 Résultats attendus (cahier des charges personnel)

Au sein de l'équipe, mon cahier des charges personnel se décline comme suit :

N°	Fonctionnalité / Exigence	Description / critères	Livrable associé
1	Interface Clavier	✓ Lecture des touches, validation du PIN. ✓ Communication MQTT/I2C avec la serrure. ✓ Intégration à la base de données.	Code source clavier
2	Interface Web / API	✓ API REST sécurisée (consultation de l'état, commande serrure). ✓ Implémentation des défis liés à la cybersécurité. ✓ Authentification (sessions/JWT). ✓ Chiffrement TLS recommandé.	Code source interface web
3	Journalisation (Logs)	✓ Enregistrement centralisé : horodatage, origine, résultat, utilisateur, etc. ✓ Accès aux logs via console ou interface web.	Module de logs
4	Sécurité & Vulnérabilités	✓ Mise en place d'une sécurité minimale ✓ Documentation et inclusion de failles intentionnelles à des fins pédagogique.	Rapport de sécurité
5	Tests & Validation	✓ Rédaction de scénarios de test (clavier, web) ✓ Vérification des logs ✓ Tests de robustesse des accès et commandes	Rapport de tests
6	Livrables	✓ Code source versionné sur GitLab/GitHub. ✓ Rapports de conception/tests/vulnérabilités ✓ Documentation installation et d'utilisation. Voir annexe 7	Dépôt GitLab + documentation

9. Planification

9.1 Diagramme de Gantt



10. Analyse Technique

Dans cette section, nous détaillons les choix technologiques opérés pour la partie « Interface Clavier » et « Interface Web/API », en explicitant les contraintes techniques (matérielles, logicielles et de performance).

10.1 Choix des technologies pour l'interface Web et le clavier

Pour l'interface clavier et la gestion locale, nous avons choisi Qt en C++ afin de faciliter l'intégration matérielle (clavier, serrure, badge NFC) et la communication via MQTT. Côté interface Web/API, nous utilisons Python (Flask) en backend pour sa simplicité d'intégration avec le reste du système et sa robustesse dans la gestion des requêtes http et HTML/CSS/JavaScript pour le frontend, garantissant une API REST simple, évolutive et sécurisée.

10.2 Contraintes techniques (matérielles, logicielles, 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 I ² C/SPI utilisés, respecter la vitesse I ² C (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	Qt	Version minimale Qt 5.15 requise pour QtMqtt (ou utiliser une bibliothèque tierce compatible).
	Python / Flask	Utiliser Python 3.9 ou supérieur (version stable recommandée).
	Python	Multi environnements pour intégrer différents niveaux.
	Firmware IR2	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, 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 Web (API) | VOIR CODE

ANNEXE 3

L'API RESTful du projet DVIL constitue le cœur de la communication entre l'interface utilisateur (web/mobile), les modules lot (clavier, badge, serrure connectée), et les outils d'administration. Elle est conçue pour être sécurisée, évolutive et observable en temps réels.

- **Architecture RESTful** : Les différents endpoints de l'API sont structurés pour permettre la consultation, la modification et l'administration des données en toute sécurité. Ex :

```
@app.route("/unlock", methods=["POST"])

@require_auth
@require_user_or_admin
def unlock():
    # Déverrouille la serrure via API, MQTT et met à jour l'état global
    try:
        mqtt_client.publish(MQTT_TOPIC_UNLOCK, "deverouillage")
        lock_state["locked"] = False
        serveur_logger.info("Déverrouillage par API")
        return jsonify({"status": "success", "message": "Serrure déverrouillée."})
    except Exception as e:
        serveur_logger.error(f"Erreur lors de la publication MQTT (unlock): {e}")
        return jsonify({"status": "fail", "message": "Erreur lors de l'envoi de la commande MQTT."}), 500
```

- **Sécurité et authentification** : Les routes sensibles sont protégées par des décorateurs Python (@require_auth, @require_admin), qui valident le token JWT de session, le rôle et appliquent un rate limit IP.

L'authentification se fait par token JWT : chaque login crée un token stocké dans un cookie sécurisé, automatiquement vérifié sur chaque appel protégé.

L'architecture et le fonctionnement de l'interface Web reposent sur une API REST, qui assure la gestion de l'authentification, le contrôle du verrouillage et du déverrouillage de la serrure, 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 Principales fonctionnalités

L'ensemble des fonctionnalités offertes par l'API, ainsi que le détail des endpoints disponibles, sont présentés en annexe 2. L'implémentation technique, accompagnée d'extraits de code et de captures d'écran illustrant l'API en fonctionnement, figure également en annexe 1 afin d'offrir une vision concrète de l'architecture et de l'utilisation du système.

11.1.2 Exemple de logique côté API

L'API DVIL orchestre la gestion de la serrure connectée selon la séquence suivante :

Étape 1 : Demande utilisateur

- L'utilisateur saisit un code PIN ou scanne un badge NFC.
- Une requête POST est envoyée vers l'API (/verify_pin ou /verify_nfc), qui transmet les données nécessaires.

Étape 2 : Vérification serveur *L'API vérifie la validité du code dans la base de données

- Si la vérification est réussie, elle publie une commande via MQTT sur le topic approprié pour le piloter la serrure physique

Étape 3 : Journalisation et suivi en temps réel

- Chaque action (succès ou échec) est inscrite dans les fichiers appropriés dans /logs
- Les logs sont diffusés en temps réel à l'interface administrateur grâce à socketIO.

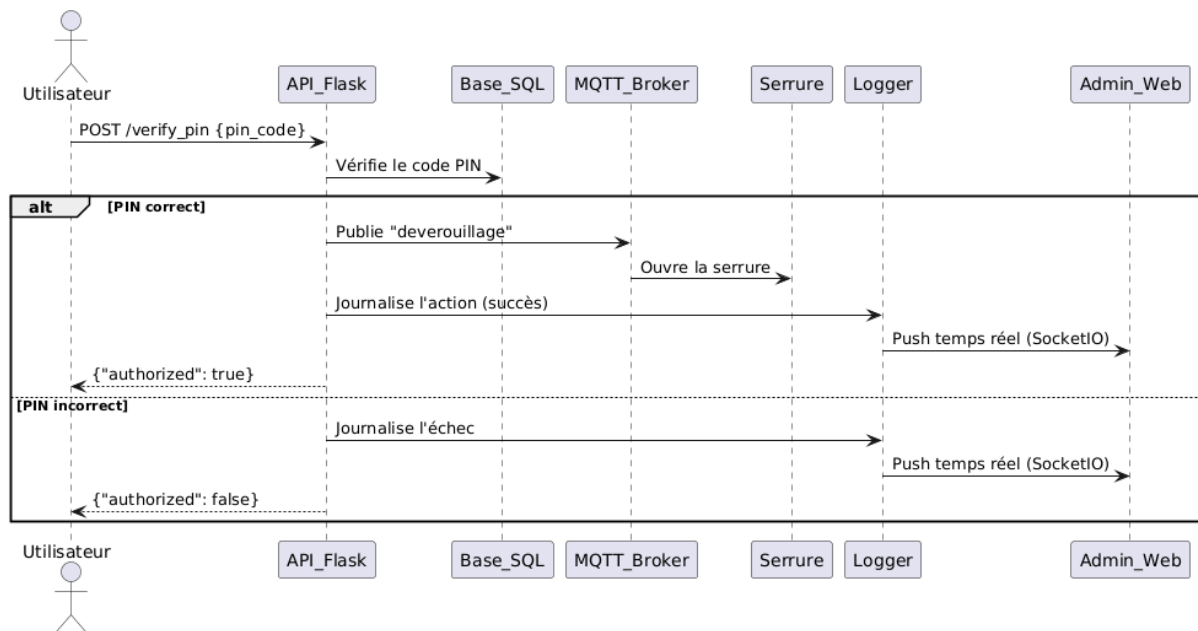
Étape : 4 Sécurisation

Chaque route sensible est protégée par des décorateurs (@require_auth, @require_admin) qui contrôlent le token JWT, le rôle de l'utilisateur, et limitent les abus par IP

La réponse API indique le status de l'opération :

```
{
  "authorized": true,
  "username": "lysandre",
  "user_id": 2,
  "role": "user",
  "message": "Déverrouillage réussi."
}
```

Le schéma ci-dessous illustre la chaîne complète : vérification, action matérielle, traçabilité en temps réel et sécurisation à chaque étape.



11.1.3 Gestion MQTT pour la communication

Pour assurer une communication efficace entre les différents modules, le projet intègre le protocole MQTT.

- **Principe du protocole MQTT** : MQTT est un protocole de messagerie léger qui écoute sur le port 1883 par défaut, et efficace, adapté aux environnements IoT. Il permet des échanges rapides, avec une consommation minimale de bande passante, et assure une faible latence, même sur des systèmes embarqués ou à ressources limitées.
- **Architecture de communication** : Un broker MQTT (tel que Mosquitto) centralise la gestion des topics et des messages. Chaque module du système (interface web, module clavier, etc.) publie ou s'abonne aux topics qui lui sont dédiés.
 - **DVIL/Serrure** : pour les commandes d'ouverture/fermeture,
 - **DVIL/Status** : pour la diffusion de l'état courant,
 - **DVIL/Clavier** : pour les interactions clavier.
- **Avantages** : Ce choix permet une communication asynchrone, facilitant la scalabilité du système et l'intégration d'applications IoT. Les tests de communication ont confirmé la robustesse de l'implémentation et la faible latence dans l'échange des messages.

Exemple de configuration :

```
# Place your local configuration in /etc/mosquitto/conf.d/
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example
```

```
pid_file /run/mosquitto/mosquitto.pid
```

```
persistence true
persistence_location /var/lib/mosquitto/
```

```
log_dest file /var/log/mosquitto/mosquitto.log
```

```
include_dir /etc/mosquitto/conf.d
```

```
#default.conf/
```

```
listener 1883
```

```
password_file /etc/mosquitto/passwd
```

```
allow_anonymous false
```

11.1.4 Gestion de la base de données pour la validation des accès

Pour centraliser la gestion des droits d'accès et des identifiants, le système s'appuie sur une base de données relationnelle légère (SQLite), adaptée aux environnements embarqués.

Technologie choisie :

SQLite, intégrée directement sur le Raspberry Pi hébergeant l'API Flask, permet de stocker toutes les données nécessaires (utilisateurs, codes PIN, badges NFC, rôles, etc.) dans un simple fichier local (dvil_db.db).

Ce choix garantit une installation simple (aucun serveur externe requis) et de bonnes performances pour un système embarqué ou expérimental.

Initialisation :

La base de données est initialisée à l'aide d'un script SQL qui crée, par exemple, la table users :

```
class User(UserMixin, db.Model):
    # Table des utilisateurs, héritée de Flask-Login UserMixin pour la gestion de session
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False, index=True)
    password = db.Column(db.String(255), nullable=False)
    pin_code = db.Column(db.String(10), nullable=True)
    nfc_code = db.Column(db.String(32), nullable=True)
    tag = db.Column(db.String(64), nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.datetime.utcnow)
    role = db.Column(db.String(20), default="user")
    mfa_secret = db.Column(db.String(16), unique=True, nullable=True)
```

Fonctionnement typique :

Transmission du code PIN : Lorsqu'un appareil physique (clavier, badge) reçoit un code saisi, il renvoie à l'API via une requête http POST (ex : /verify_pin).

Validation côté API :

L'API interroge la base avec une requête SQL telle que :

```
SELECT * FROM users WHERE pin_code = ?;
```

```
POST /verify_pin
{
  "pin_code": "1234"
  "nfc_code": "79:75:54"
}
```

Pour consulter la structure détaillée de la base de données, les scripts d'initialisation SQL et des exemples de contenu, se reporter à l'Annexe 4.

11.1.5 Intégration de la logique de journalisation

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 du projet (API Flask, interface web, modules IoT) intègre un mécanisme de logs. Cela permet de :

- Tracer toutes les actions importantes (authentification, déverrouillage, erreurs...)
- Identifier rapidement l'origine d'une anomalie ou d'un comportement inattendu
- Garantir la traçabilité pour l'analyse post-incident et la conformité

II. Mécanismes techniques utilisés

Nous avons utilisé le module Python logging, configuré pour écrire dans différents fichiers selon la nature de l'événement :

- api.log pour les actions liées à l'API
- serrure.log pour les actions physiques sur la serrure
- clavier.log pour les interactions clavier, etc.

Exemple de script :

```
LOGS_DIR = "logs"
os.makedirs(LOGS_DIR, exist_ok=True)

def create_logger(name, file, level=logging.INFO, fmt=None):
    logger = logging.getLogger(name)
    if not logger.handlers():
        fhandler = logging.FileHandler(os.path.join(LOGS_DIR, file))
        formatter = logging.Formatter(fmt or '%(asctime)s [%(levelname)s]
%(message)s')
        fhandler.setFormatter(formatter)
        logger.addHandler(fhandler)
        logger.setLevel(level)
    return logger

api_logger = create_logger("api", "api.log")
serrure_logger = create_logger("serrure", "serrure.log")
```

III. Niveaux de log et gestion de la granularité

Nous utilisons différents niveaux de log :

- DEBUG pour les détails techniques pendant le développement
- INFO pour les actions normales (ex : ouverture, connexion)
- WARNING pour les comportements inattendus
- ERROR pour les exceptions et échecs critiques

Exemples :

```
api_logger.info("Utilisateur connecté avec succès")
api_logger.warning("Tentative de connexion échouée pour %s", username)
serrure_logger.error("Erreur lors du déverrouillage : %s", str(e))
```

IV. Journalisation en temps réel pour l'administration

Un thread Python lit en continu les logs critiques (serrure.log) et envoie toute nouvelle ligne à l'interface admin via SocketIO.

Ainsi, l'administrateur visualise en temps réel les événements du système sur son dashboard web.

Extrait de code :

```
socketio = SocketIO(app)

def tail_logs():
    with open("logs/serrure.log", "r") as f:
        f.seek(0, 2)
        while True:
            line = f.readline()
            if line:
                socketio.emit("log_update", {"log": line.strip()})
            else:
                time.sleep(0.5)

threading.Thread(target=tail_logs, daemon=True).start()
```

5. Exploitation concrète des logs

Voici quelques exemples de lignes générées dans les fichiers de log :

```
2025-06-01 16:02:18 [API] Utilisateur connecté avec succès
2025-06-01 16:05:09 [SERRURE] Déverrouillage par API réussi
2025-06-01 16:09:47 [API] Tentative de connexion échouée pour 'alice'
2025-06-01 16:12:11 [API] Erreur lors de la publication MQTT : TimeoutError
```

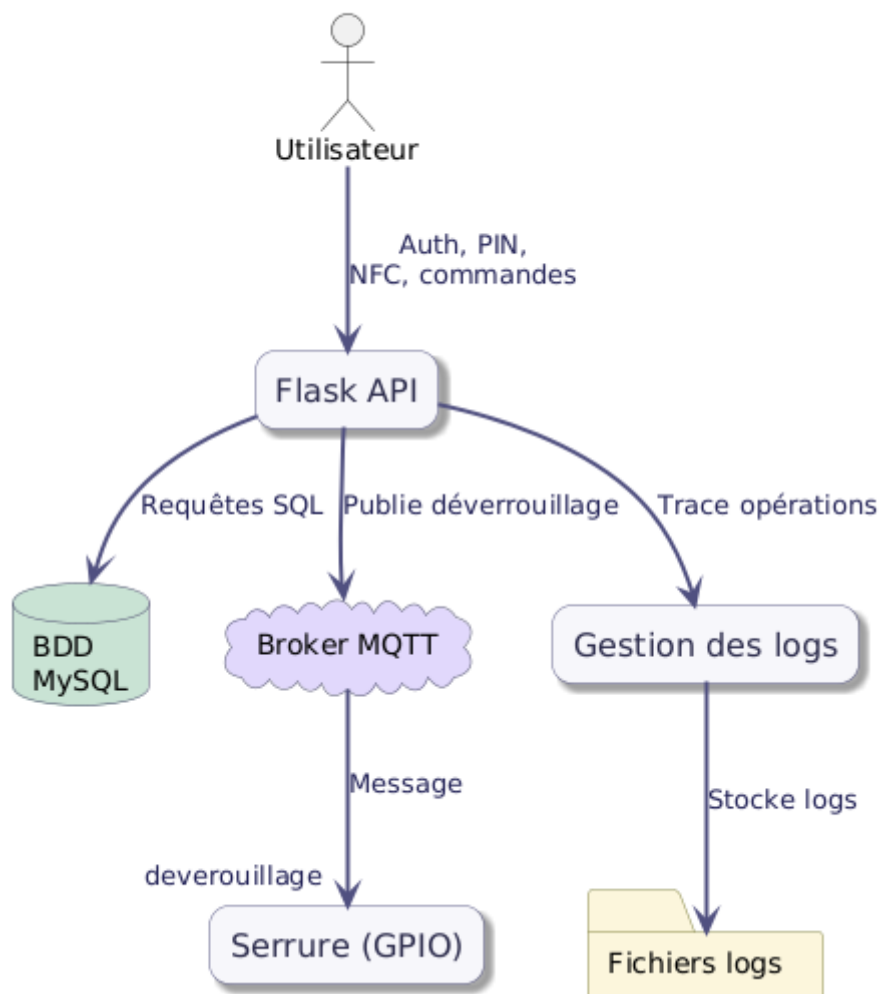
Cela permet à tout moment de retracer les opérations, d'analyser les incidents et de justifier les actions réalisées.

11.2 Diagrammes et représentation graphique.

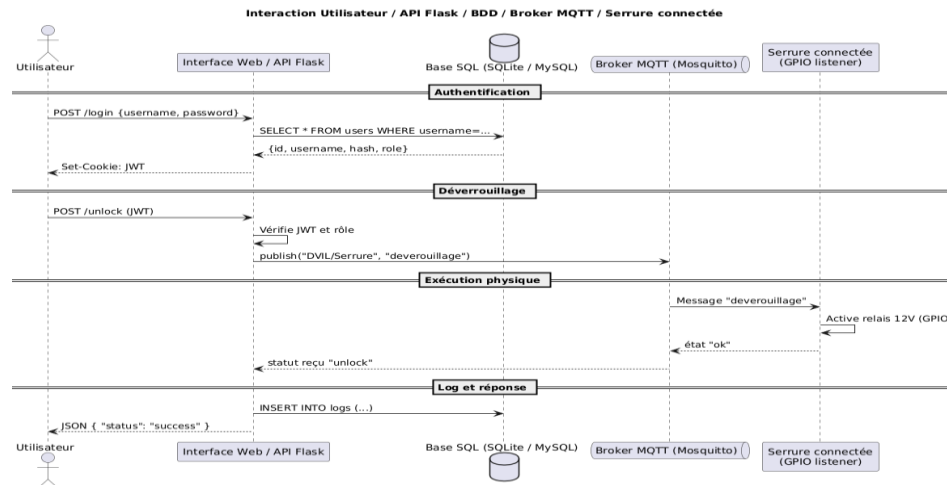
Ce diagramme de flux de données (DFD) met en avant la circulation des informations entre l'utilisateur, l'API, la base de données, le broker MQTT, les logs et la serrure connectée.

Il facilite la compréhension des échanges et des interactions techniques lors d'une utilisation typique du système.

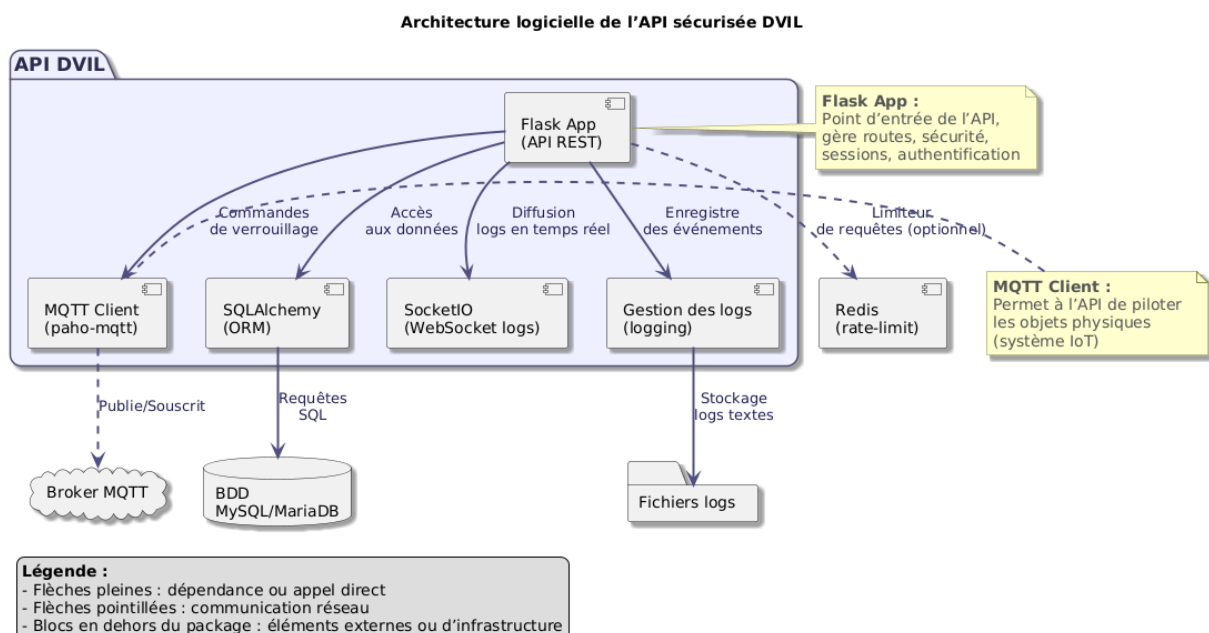
Flux de données — API DVIL



Ce diagramme de séquence détaille, étape par étape, le processus complet d'une demande de déverrouillage : de l'authentification de l'utilisateur jusqu'à l'activation physique de la serrure et la traçabilité de l'action via les logs.

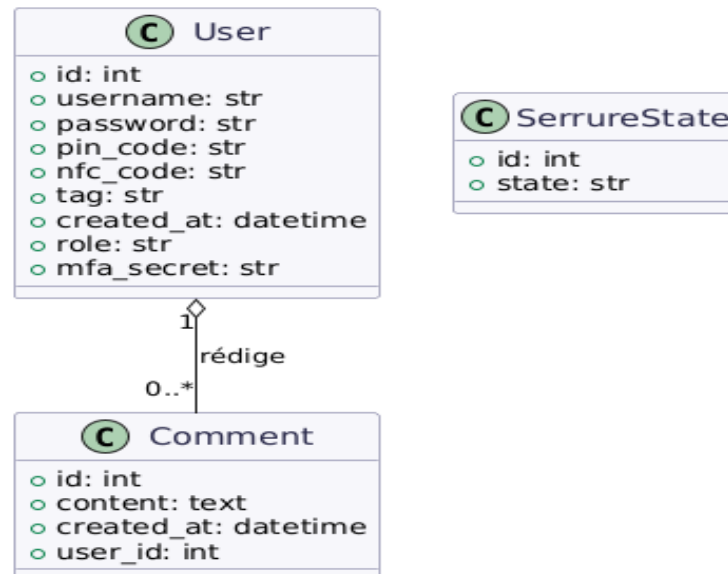


Grâce à ce diagramme de composants, on visualise clairement l'organisation technique de l'application, les interactions entre ses différents modules, et la manière dont l'API DVIL s'intègre à son environnement matériel et logiciel. Cette architecture modulaire et sécurisée facilite la maintenance, l'évolutivité et la traçabilité des accès au système.



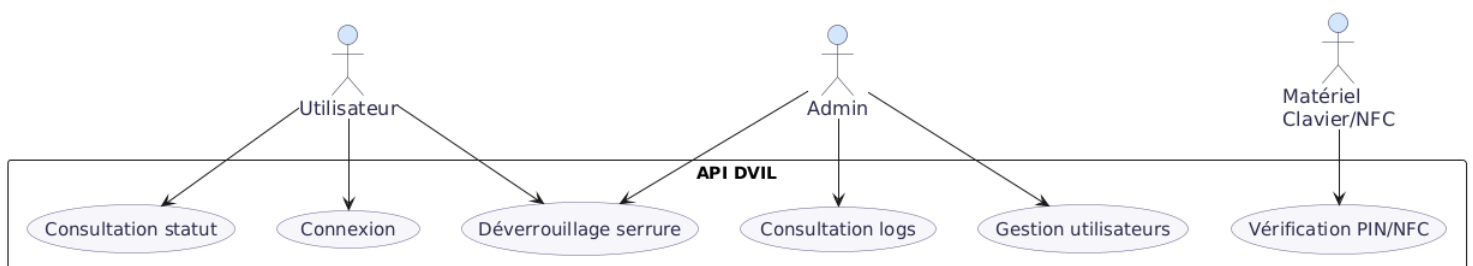
Ce diagramme de classes (ERD) présente la structure des données de l'application : il décrit les principales tables, leurs attributs et les liens entre elles. Grâce à cette représentation, on visualise la logique de stockage et la cohérence des données côté back-end.

Modèle de données (ERD) — DVIL



Ce diagramme de cas d'utilisation illustre les différents acteurs du système (utilisateur, admin, matériel) et leurs interactions principales avec l'API DVIL. Il permet de résumer les fonctionnalités essentielles proposées par l'application selon chaque profil d'utilisateur.

Cas d'utilisation — Plateforme DVIL



11.3 Développement du multi environnements

Dans un contexte pédagogique, il est essentiel de pouvoir basculer rapidement entre une version sécurisée et différents scénarios de faille sans perturber le projet. C'est pourquoi j'ai adopté une architecture **multi-niveau** : chaque version (sécurisée ou vulnérable) est isolée dans son propre dossier, et l'administrateur peut activer le niveau souhaité simplement en modifiant une variable et en redémarrant le service concerné.

Ci-dessous, l'organisation du dossier /home/Ad_@min/flask_app :

```
/home/Ad_@min/flask_app/
├─ API.py                # Script principal (lanceur ou API Flask général)
├─ arborescence.txt      # Documentation de l'arborescence
├─ cookies.txt           # Fichier de cookies, si besoin
├─ levels/               # Répertoire des différents niveaux/challenges
│   └─ __init__.py
│   └─ __pycache__/
│   └─ serrure.log
│   └─ version_0/        # Niveau sécurisé (aucune faille)
│   └─ version_1/        # Challenge niveau 1 (faille simple)
│   └─ version_2/        # Challenge niveau 2 (failles multiples)
│   └─ version_3/        # Challenge niveau 3 (failles avancées)
├─ logs/                 # Journaux structurés de l'API
│   └─ api.log
│   └─ clavier.log
│   └─ keypad_raw.log
│   └─ nfc.log
│   └─ serveur.log
├─ migrations/           # Gestion des versions de la base de données
├─ myenv/                # Environnement virtuel Python (librairies isolées)
├─ README.md             # Documentation du projet
├─ requirements.txt       # Liste des dépendances Python
├─ serrure.log            # Log général du système
└─ wsgi.py               # Point d'entrée pour serveur WSGI (prod)
```

Le répertoire **levels/** contient un sous-dossier pour chaque version du projet, de la version sécurisée (version_0/) aux différents challenges pédagogiques (version_1/ à version_3/).

Chaque dossier regroupe les scripts et configurations propres à ce niveau, permettant d'isoler parfaitement chaque environnement.

Le **niveau actif** est sélectionné simplement par l'administrateur, en modifiant la variable LEVEL dans le fichier .env ou dans la configuration du service systemd (ex : /etc/systemd/system/app-api.service).

Il suffit ensuite de redémarrer le service pour appliquer le changement :

```
LEVEL=3
```

Puis :

```
sudo systemctl restart app-api.service
```

Chaque module du système (API, NFC, serrure, clavier) dispose de son propre service systemd dédié, ce qui assure :

- ✓ une isolation complète entre les versions,
- ✓ une facilité de test et de déploiement,
- ✓ une flexibilité pédagogique optimale.

Ainsi, on peut passer d'un challenge à un autre ou restaurer un environnement en un seul clic, sans risque de conflit ou de perte de configuration.

11.4 [API] Implémentation Challenge 1.

Définition : On parle généralement de **faille de contrôle d'accès** (ou *Broken Access Control*) lorsque des fonctionnalités sensibles sont accessibles sans vérification des droits ou sans authentification appropriée. Dans ce cas précis, l'application dispose d'une route **/v1/unlock_public** qui autorisait le déverrouillage sans exiger de token ni de session valide, entraînant un risque majeur de sécurité.

```
@app.route("/v1/unlock_public", methods=["POST"])
def vuln_unlock_public():
    """
    Faille de contrôle d'accès : aucun mécanisme d'authentification.
    Cet endpoint permet à n'importe qui de déverrouiller la serrure.
    """
    mqtt_client.publish(MQTT_TOPIC_UNLOCK, "deverouillage")
    lock_state["locked"] = False
    logging.warning("[VULN] Déverrouillage public via /v1/unlock_public !")
    return jsonify({"status": "unlocked", "message": "Serrure déverrouillée (faille niveau 1)"})
```

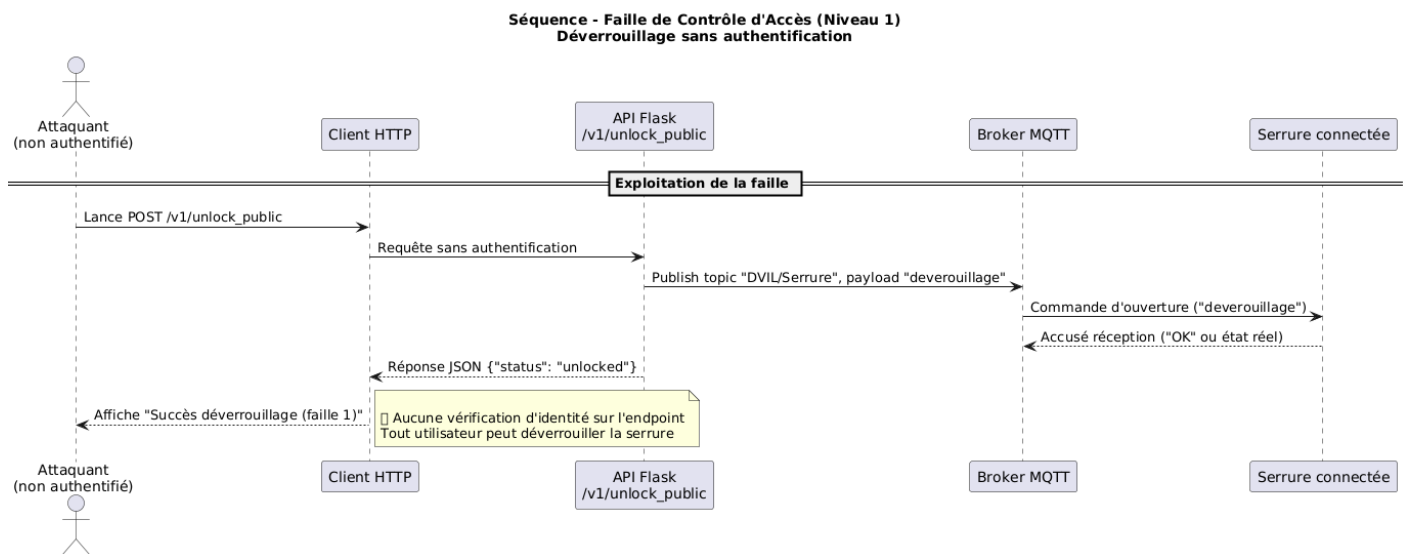
- **Vecteur d'attaque :**
Tout utilisateur, sans privilèges ou même inconnu, peut accéder à cette route et publier une commande MQTT pour déverrouiller la serrure.
- **Impact potentiel :**
Cette vulnérabilité permet à un attaquant d'accéder physiquement à une zone protégée en contournant l'authentification, compromettant ainsi la sécurité globale du système.

Mesures Correctives :

- Supprimer ou désactiver cet endpoint dans un environnement de production.
- Implémenter des contrôles d'accès stricts (par exemple, utiliser les décorateurs d'authentification déjà présents dans les autres endpoints).
- S'assurer que toute action sensible requiert une authentification et des droits appropriés.

11.4.1 Diagrammes et représentation graphique.

Ce diagramme de séquence met en évidence la faille de contrôle d'accès du niveau 1 : une requête POST anonyme sur /v1/unlock_public suffit à déclencher l'ouverture physique de la serrure, sans aucune vérification d'authentification. L'API relaie directement la commande via MQTT, et la réponse JSON confirme le succès de l'opération. Le challenger aura comme indication le nom exact de l'endpoint dans le script du frontend.



Ce type d'endpoint accessible publiquement sans authentification est **une erreur majeure**, car il ne vérifie **ni l'identité** ni **le rôle** du client.

Le serveur Flask (FlaskApp) reçoit la requête et exécute immédiatement la commande de déverrouillage.

Aucune vérification de token ou de session n'est effectuée. Cela signifie que **n'importe qui connaissant l'URL peut déverrouiller la serrure à distance**.

Le broker MQTT reçoit l'ordre et l'envoie à la serrure.

Résultat : la serrure s'ouvre, et une **réponse positive** est retournée à l'hacker :

```
{ "status": "unlocked" }
```

11.5 [API] Implémentation Challenge 2.

Dans le Challenge 2 du projet DVIL, deux vulnérabilités complémentaires sont introduites pour confronter les étudiants à des **scénarios d'attaque réalistes et combinés**.

Ces failles ne sont pas indépendantes : elles sont exploitables via un même point d'entrée vulnérable, illustré dans le diagramme de séquence "Attaque (XSS + Path Traversal)".

I. Cross-Site Scripting (XSS)

Permet à un attaquant d'injecter du code JavaScript dans une page web légitime.

Conséquences :

Le code malveillant s'exécute dans le navigateur d'autres utilisateurs, pouvant mener au vol de cookies, à des redirections ou à la modification du contenu.

II. Path Traversal (Directory Traversal)

Faible permettant à un attaquant de manipuler des chemins de fichiers (ex : ../) afin d'accéder à des fichiers/dossiers en dehors de l'espace prévu.

Conséquences :

Accès à des fichiers sensibles du système (configurations, logs, etc.) normalement inaccessibles.

Point d'entrée vulnérable et scénario d'exploitation

L'endpoint /vuln/template utilise la fonction `render_template_string` en passant directement la donnée du paramètre `q` sans validation.

Ce comportement ouvre la voie à deux types d'attaques simultanées :

Injection de Template (SSTI) : possibilité d'injecter du code dans le template côté serveur.

Path Traversal : possibilité d'utiliser des séquences de chemins pour accéder à des fichiers critiques.

Un attaquant peut ainsi concevoir des payloads spécifiques pour :

- exécuter du code côté serveur,
- lire/exfiltrer des fichiers sensibles.

Mesures correctives recommandées

- Valider et assainir toutes les entrées utilisateur avant tout rendu dans un template.
- Remplacer l'utilisation directe de `render_template_string` par des méthodes plus sécurisées, ou restreindre sévèrement les données pouvant être injectées.
- Implémenter des contrôles sur les chemins de fichiers pour empêcher toute tentative de path traversal.

11.6 Endpoint vulnérable.

```
@app.route("/v2/xss", methods=["GET", "POST"])
def vuln_xss():
    """
    Faille XSS : GET affiche le formulaire, POST enregistre le commentaire
    sans filtre (XSS possible).
    Active le flag de session xss_done.
    """
    if request.method == "POST":
        user_comment = request.form.get("comment", "")
        new_c = Comment(content=user_comment)
        db.session.add(new_c)
        db.session.commit()
        session["xss_done"] = True # Flag : faille XSS exploitée
        return render_template_string(f"""
        <h2>Votre commentaire :</h2>
        <div style="color:blue;">
            {user_comment}
        </div>
        <p><a href="/v2/xss">Retour</a></p>
        """)
    return """
    <h1>Faille XSS</h1>
    <form method="POST" action="/v2/xss">
        <label>Entrez un commentaire :</label><br/>
        <textarea name="comment" rows="4" cols="50"></textarea><br/><br/>
        <button type="submit">Envoyer</button>
    </form>
    <p>(Les commentaires s'afficheront sans protection... XSS possible)</p>
    """
```

Faille : XSS (Cross-Site Scripting)

Fonctionnement :

- GET : Affiche un formulaire HTML pour poster un commentaire
- POST :
 - Enregistre un commentaire en base **sans aucun filtrage**
 - Active session["xss_done"] = True (flag d'exploitation réussie)
 - Ré-affiche le commentaire injecté via render_template_string (danger : exécution de balises <script>)

```

@app.route("/v2/traversal")
def vuln_traversal():
    """
    Faille Path Traversal : lit un fichier du dossier uploads sans contrôle du
    chemin.
    Nécessite que la faille XSS ait été exploitée (flag session).
    Active le flag path_done si exploit réussi.
    """
    if not session.get("xss_done"):
        return " Vous devez d'abord exploiter la faille XSS avant d'accéder à
        celle-ci.", 403

    filename = request.args.get("file", "")
    base_folder = os.path.join(BASE_DIR, "uploads")
    filepath = os.path.join(base_folder, filename)

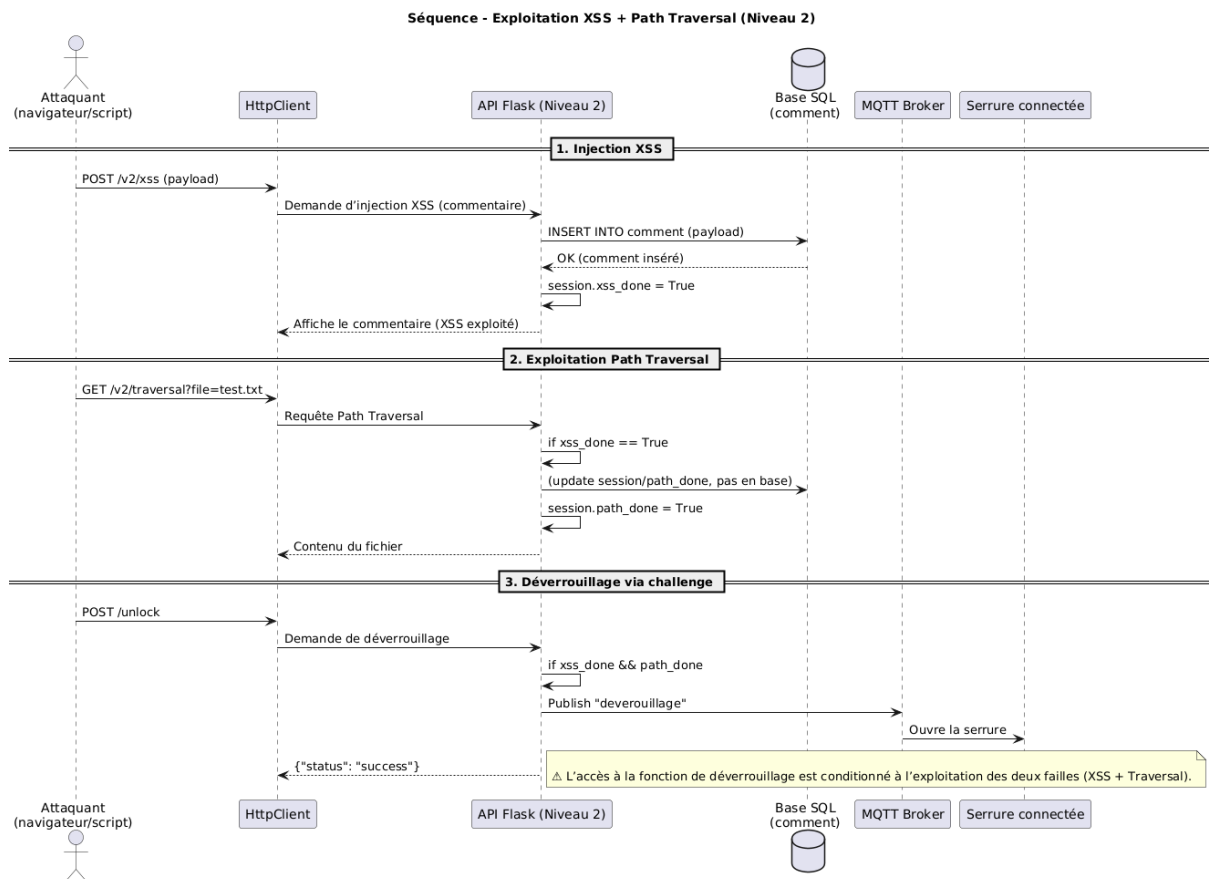
    try:
        with open(filepath, "r") as f:
            content = f.read()
            session["path_done"] = True # Flag : faille Path Traversal exploitée
            return f"<pre>{content}</pre>"
    except Exception as e:
        return f"Erreur: {str(e)}", 404

```

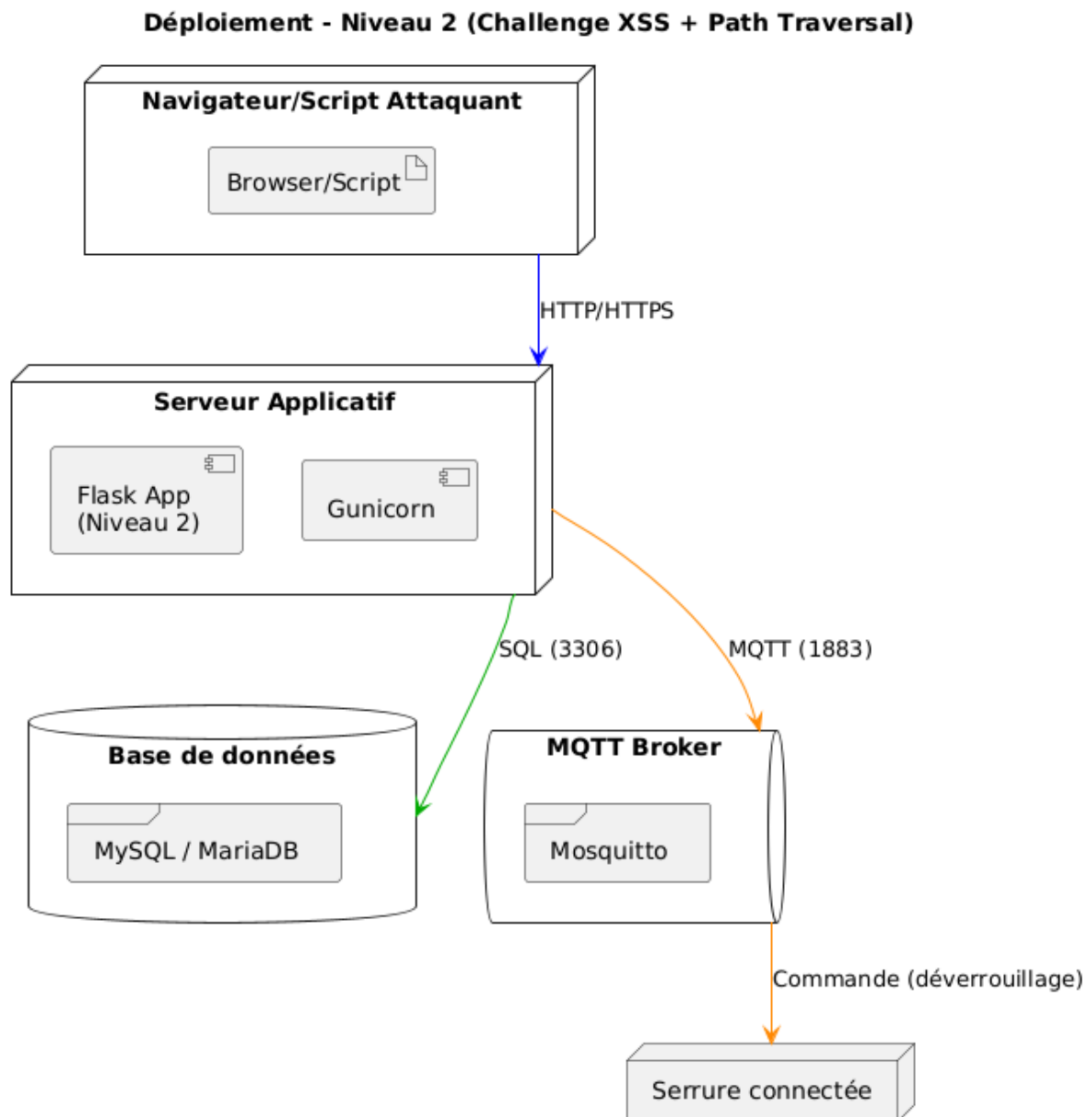
- ❖ Lit un fichier à partir du dossier uploads/, **sans filtrer ni vérifier le chemin**.
- ❖ Le nom de fichier est transmis en paramètre GET (?file=...).
- ❖ Permet d'accéder à des fichiers en dehors du dossier uploads/ via ../ (ex: ../../../etc/passwd).
- ❖ **Path Traversal** classique.
- ❖ **Condition préalable :**
- ❖ La faille n'est exploitable que si la session contient xss_done=True.
- ❖ Deuxième étape de l'exploitation. Si le fichier existe, path_done est activé.
- ❖ Simulation d'une chaîne de failles réalistes.

11.6.1 Diagramme et représentation graphique.

Ce diagramme de séquence détaille le scénario d'exploitation du niveau 2 : l'attaquant envoie d'abord un payload XSS via la route /v2/xss, puis exploite une faille de path traversal sur /v2/traversal. Ces deux étapes activent des flags de session côté serveur (xss_done et path_done), condition indispensable pour débloquent la fonctionnalité de déverrouillage. Enfin, la requête /unlock permet de publier la commande de déverrouillage via MQTT, ouvrant la serrure. Ce scénario illustre concrètement la nécessité de cumuler plusieurs failles pour compromettre la sécurité, et met en évidence l'importance de contrôler l'ensemble des flux de données au sein d'une API.



Ce diagramme de déploiement présente l'architecture du projet DVIL en mode challenge niveau 2. Il montre l'articulation entre les différents composants : l'attaquant interagit avec l'application Flask (hébergée via Gunicorn), laquelle échange avec une base de données MySQL/MariaDB pour la gestion des commentaires, et utilise un broker MQTT (Mosquitto) pour transmettre les commandes à la serrure connectée.



11.7 [API] Implémentation challenge 3.

Dans le niveau 3 du projet DVIL, plusieurs vulnérabilités critiques sont présentes, pouvant être exploitées individuellement ou en chaîne. Ces failles sont illustrées dans les diagrammes d'attaque associés.

1) Injection de Templates (SSTI – Jinja2)

L'injection de templates consiste à insérer du code malveillant dans le moteur de rendu de templates (ici, Jinja2) côté serveur.

Danger :

Si l'entrée utilisateur est passée directement à une fonction comme `render_template_string`, un attaquant peut injecter des instructions Jinja2 (`{{ ... }}`), accéder à des variables sensibles ou exécuter du code arbitraire.

Exemple :

Une requête du type `/vuln/template?q={{config.items()}}` peut exposer la configuration du serveur.

2) Désérialisation non sécurisée (Pickle)

Survient lorsqu'un serveur utilise `pickle.loads()` sur des données non contrôlées fournies par l'utilisateur.

Danger :

Un attaquant peut créer un objet picklé contenant du code Python malveillant, qui sera exécuté dès la désérialisation, compromettant ainsi tout le serveur.

Exemple :

En envoyant un payload spécialement conçu à `/vuln/deserialize`, l'attaquant peut obtenir une exécution de commande système.

3) Injection SQL

L'injection SQL consiste à manipuler des entrées utilisateur pour modifier ou injecter des requêtes SQL exécutées par le serveur.

Danger :

Cela permet de lire, modifier, supprimer des données, ou de contourner les contrôles d'accès.

Exemple :

Sur l'endpoint /vuln/serrure, la variable `update_value` non filtrée est injectée directement dans la requête SQL, permettant à l'attaquant de modifier l'état de la serrure ou d'accéder à d'autres informations sensibles.

Dans ce niveau, un attaquant peut exploiter ces failles isolément ou en chaîne pour :

- Exécuter du code arbitraire sur le serveur (SSTI, désérialisation Pickle)
- Accéder, modifier ou supprimer des données en base (Injection SQL)
- Contrôler l'état physique de la serrure connectée (via MQTT détourné)

L'impact est majeur : la compromission totale de l'application et du système, avec perte de confidentialité, d'intégrité et de contrôle.

Mesures correctives

- ✓ Ne jamais passer de données utilisateur à `render_template_string` sans validation stricte.
- ✓ Bannir l'utilisation de Pickle pour des données venant de l'extérieur.
- ✓ Utiliser des requêtes SQL paramétrées (requêtes préparées) pour toute entrée utilisateur.

11.8 Endpoint vulnérable.

```
# Endpoint volontairement vulnérable SSTI pour la pédagogie
@app.route("/vuln/template")
def vuln_template():
    # Démontre la vulnérabilité Server Side Template Injection
    user_input = request.args.get("q", "")
    return render_template_string(user_input, open=open)
```

L'entrée utilisateur (q) est directement injectée dans `render_template_string`, une fonction Jinja2 qui interprète dynamiquement le contenu.

Elle est capable d'exécuter des expressions Jinja2 côté serveur, comme :

```
?q={{ config.items() }}
?q={{ cycler.__init__.__globals__.os.popen('ls').read() }}
```

Permet d'accéder à des variables internes du serveur, voire d'exécuter des commandes système selon le contexte.

```
# Endpoint volontairement vulnérable (injection pickle)
@app.route("/vuln/deserialize", methods=["POST"])
def vuln_deserialize():
    import pickle
    global OVERRIDE_TOKEN
    try:
        obj = pickle.loads(request.data)
        if isinstance(obj, dict) and "set_token" in obj:
            OVERRIDE_TOKEN = obj["set_token"]
        return "Objet désérialisé: " + str(obj)
    except Exception as e:
        return str(e), 400
```

`pickle.loads()` est exécuté **sans aucune vérification** sur des données fournies par le client.

Un attaquant peut injecter un **objet picklé contenant du code malveillant**, qui sera exécuté automatiquement à la désérialisation.

```

# Endpoint vulnérable SQLi pour CTF
@app.route("/vuln/serrure", methods=["POST"])
def vuln_serrure():
    token = request.args.get("token")
    update_value = request.args.get("update")
    correct_token = app.config.get("SECRET_KEY")
    if OVERRIDE_TOKEN:
        correct_token = OVERRIDE_TOKEN

    if token != correct_token:
        return "Accès non autorisé", 403

    # INJECTION SQL VOLONTAIRE (danger : ne pas faire en prod réelle)
    query = f"UPDATE serrure_state SET state = '{update_value}' WHERE id = 1"

    try:
        state_entry = SerrureState.query.get(1)
        if not state_entry:
            state_entry = SerrureState(id=1, state="lock")
            db.session.add(state_entry)
            db.session.commit()
        db.session.execute(text(query))
        db.session.commit()
        clean_value = update_value.split("'")[0]
        # Action MQTT
        if "unlock" in clean_value.lower():
            mqtt_client.publish(MQTT_TOPIC_UNLOCK, "deverrouillage")
        elif "lock" in clean_value.lower():
            mqtt_client.publish(MQTT_TOPIC_LOCK, "Mauvaise manipe")
        # Thread reset
        logging.info("Lancement du thread de reset dans 10s...")
        threading.Thread(target=reset_challenge_10s, daemon=True).start()
        return f"Serrure mise à jour avec: {update_value}"
    except Exception as e:
        return str(e), 500

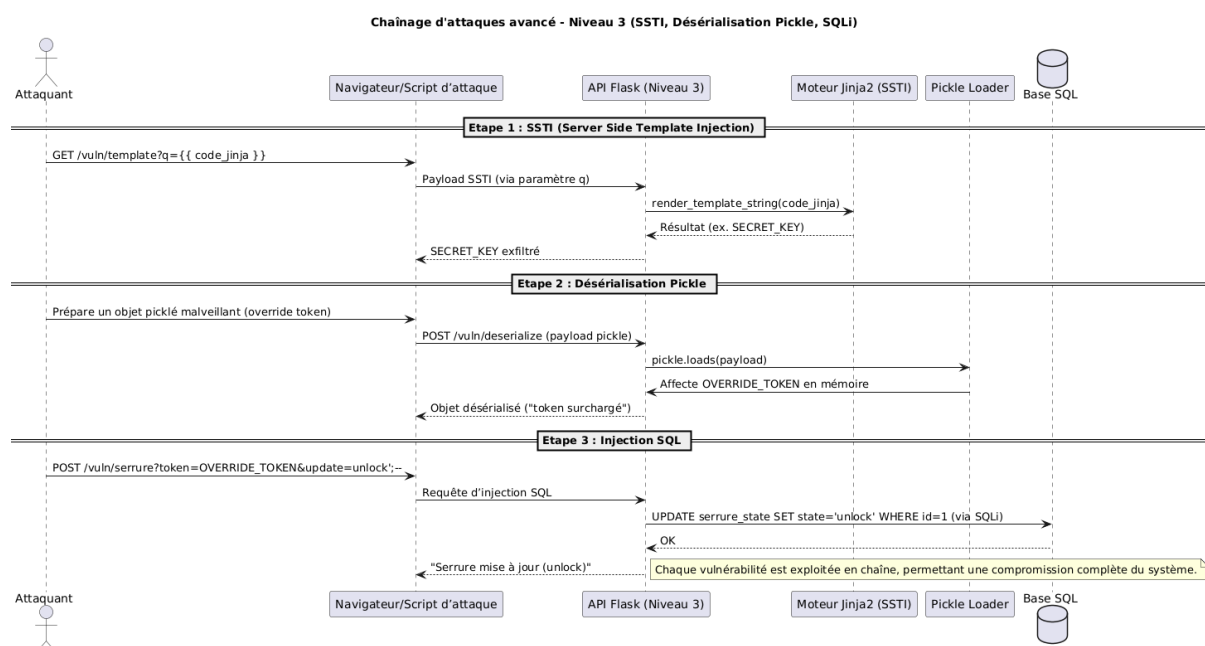
```

Si unlock est présent dans la valeur, une commande **MQTT de déverrouillage** est envoyée.

Cela permet de **prendre le contrôle physique de la serrure** sans authentification.

L'utilisation de `OVERRIDE_TOKEN` (surchargé via Pickle) permet de **contourner la vérification du token**, même sans authentification réelle.

11.8.1 Diagrammes et représentation graphique.



Ce diagramme illustre le chaînage des trois vulnérabilités critiques du niveau 3 de DVIL : SSTI (Server Side Template Injection), désérialisation non sécurisée (Pickle) et injection SQL.

L'attaquant commence par exploiter une SSTI pour obtenir le SECRET_KEY du serveur. Ce secret lui permet ensuite de générer ou surcharger un jeton via une désérialisation Pickle malveillante, autorisant l'accès à l'endpoint sensible `/vuln/serrure`. Enfin, l'injection SQL sur ce dernier endpoint permet de modifier directement l'état de la serrure dans la base, aboutissant à une compromission totale du système.

Ce scénario met en avant la dangerosité de la combinaison de failles, où chacune facilite l'exploitation de la suivante, et rappelle l'importance d'une défense en profondeur sur toutes les couches de l'application.

12. Développement de l'application Clavier.

Le module clavier, développé en C++ avec le framework Qt, constitue le point d'entrée principal pour les interactions physiques dans le système DVIL.

Il offre une interface ergonomique et réactive, gère les événements clavier (saisie, validation, effacement) et s'appuie sur une architecture orientée objet qui facilite sa maintenance et son évolution.

Lorsqu'un utilisateur saisit un code sur le clavier et valide, le module **interroge automatiquement une route dédiée de l'API Flask** (par exemple, /verify_pin).

Ce dialogue se fait généralement par une requête HTTP POST :

- **Le code PIN saisi** est envoyé à l'API.
- **L'API vérifie le code** en consultant la base de données centralisée (MySQL, SQLite, etc.).
- **La réponse de l'API** (autorisé/refusé) est alors affichée à l'utilisateur ou transmise à la partie contrôle/serrure.

Ce mécanisme permet :

- De **centraliser la validation des accès** dans la base de données du serveur.
- D'éviter que la logique d'authentification soit dupliquée ou exposée côté client/clavier.
- De **journaliser** chaque tentative dans les logs du serveur, pour le suivi et la sécurité.

12.1 Faille vulnérable sur le clavier.

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

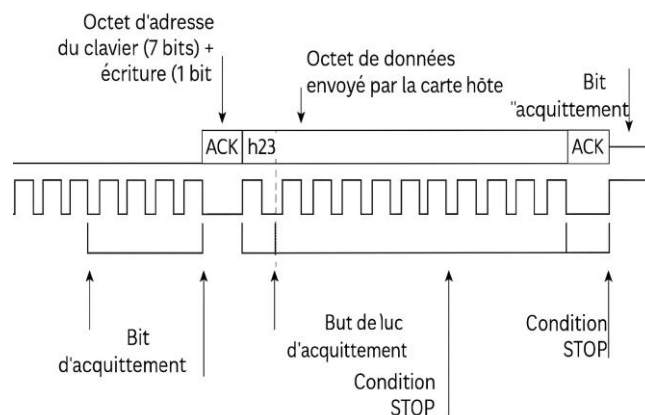
- Un attaquant équipé d'un sniffer I²C pourrait récupérer les codes transmis en clair.
- Cela mettrait en péril la confidentialité et l'intégrité du système.

Contremesures 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.

12.2 I²C Frame.

Cette image représente une **trame I²C réelle 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.

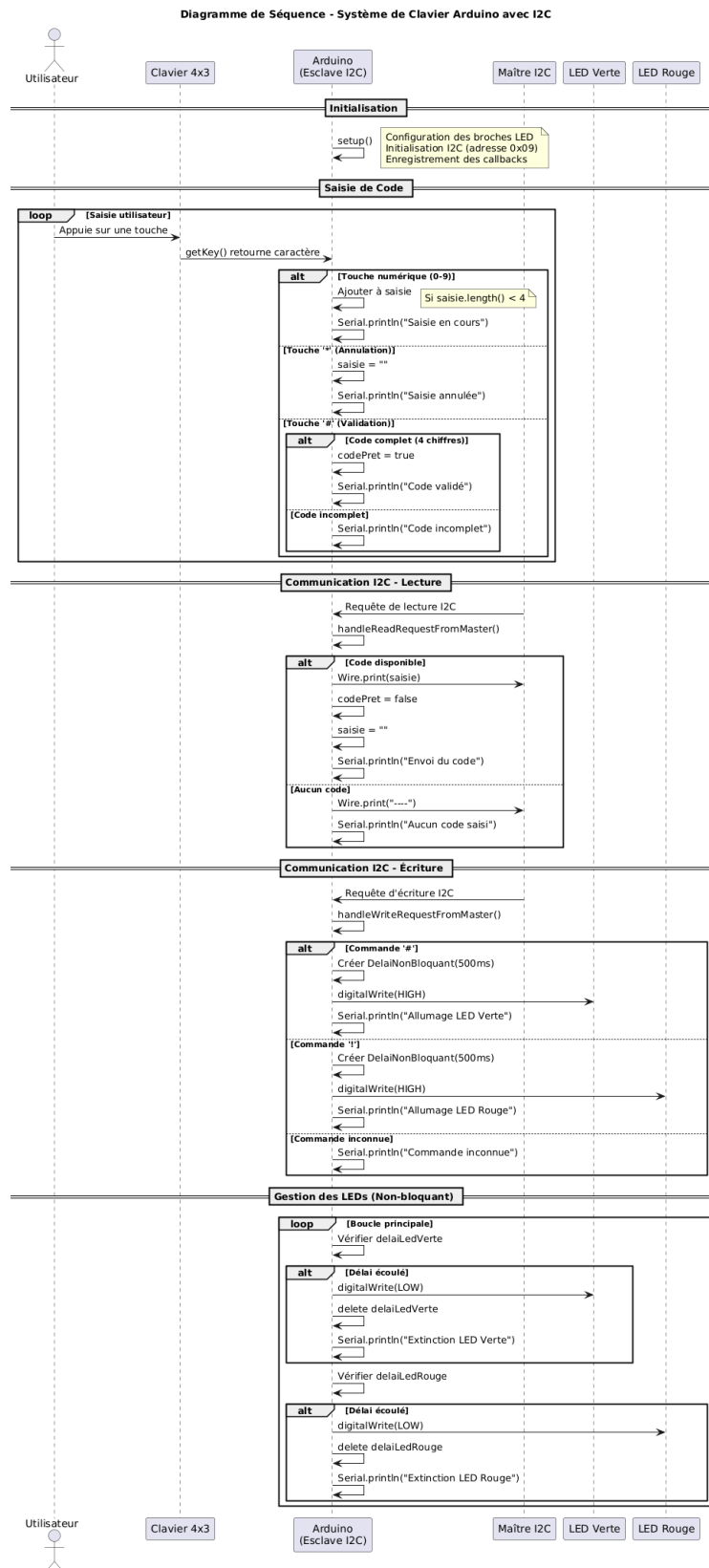
ACK — Deuxième acquittement

- 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.

12.3 Diagramme et représentation graphique.



Ce diagramme illustre le fonctionnement d'un système de saisie de code sécurisé avec un clavier matriciel 4x3 relié à un microcontrôleur (Arduino/ATtiny) agissant comme esclave I2C.

- L'utilisateur saisit un code à 4 chiffres, validé par # ou annulé par *.
- Le maître I2C (ex : RPi) peut lire le code saisi ou envoyer des commandes pour allumer les LEDs verte ou rouge (retour visuel).
- La gestion des LEDs se fait sans bloquer la boucle principale, grâce à un délai non bloquant.

Code explicatif :

```
#include <Keypad.h>
#include <Wire.h>

Keypad clavier = Keypad(makeKeymap({
  {'1','2','3'}, {'4','5','6'}, {'7','8','9'}, {'*','0','#'}
}), lignes, colonnes, 4, 3);

String saisie = "";
bool codePret = false;

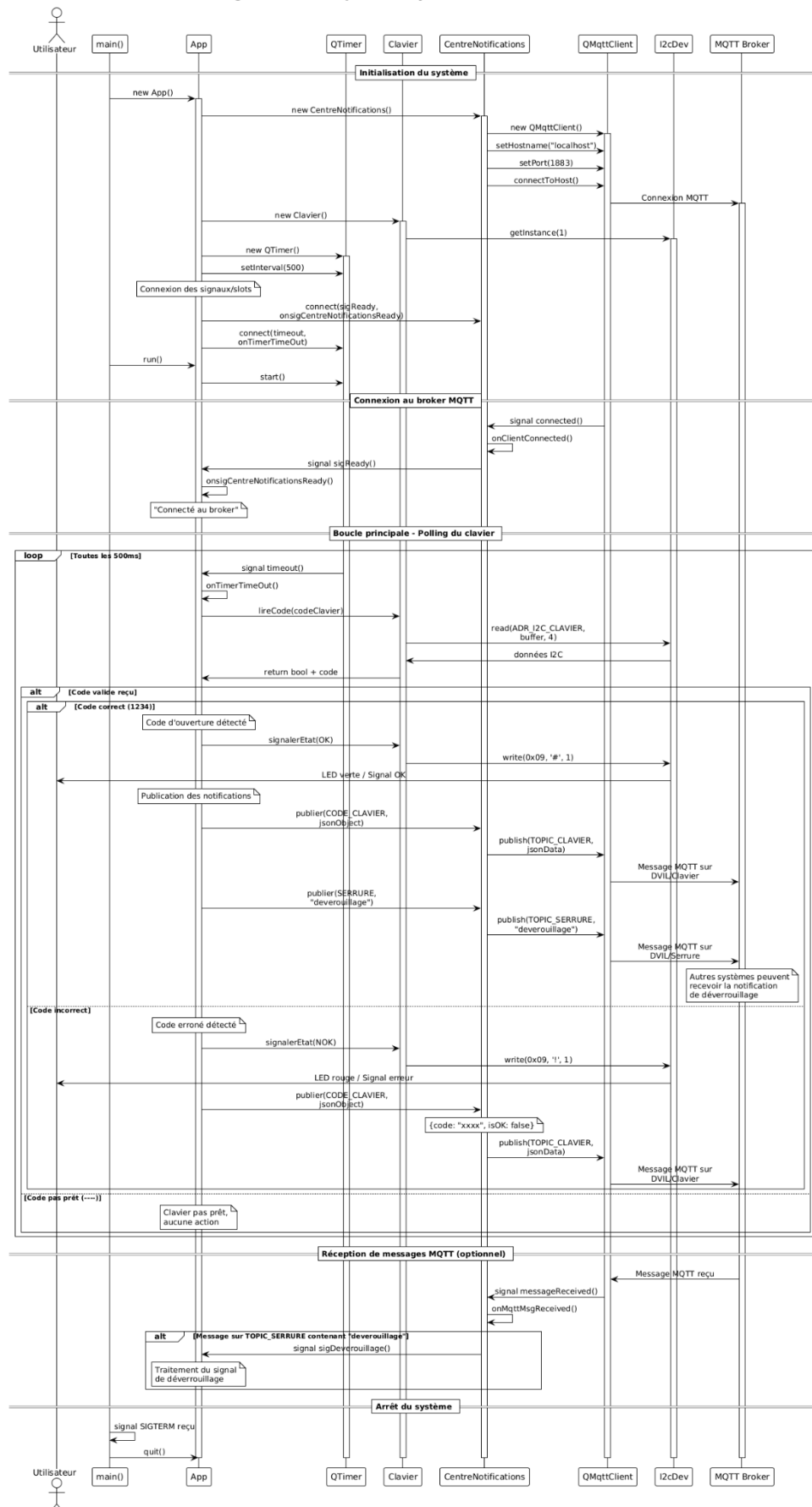
void setup() {
  Wire.begin(0x09);
  Wire.onRequest([]() { // Lecture par le maître
    Wire.print(codePret ? saisie.c_str() : "----");
    if (codePret) { codePret = false; saisie = ""; }
  });
  Wire.onReceive([](int) { // Commande LED par le maître
    char cmd = Wire.read();
    digitalWrite(cmd == '#' ? LED_VERTE : LED_ROUGE, HIGH);
  });
  pinMode(LED_VERTE, OUTPUT); pinMode(LED_ROUGE, OUTPUT);
}

void loop() {
  char t = clavier.getKey();
  if (t == '#') codePret = (saisie.length() == 4);
  else if (t == '*') saisie = "";
  else if (t && saisie.length() < 4) saisie += t;
  // ... gestion délais LEDs (non bloquant) ...
}
```

VIALETTE Lysandre (IR1)

Diagramme de séquence - Système de contrôle d'accès

S



Ce diagramme de séquence illustre le déroulement des opérations lors de la saisie d'un code utilisateur et de la communication avec le microcontrôleur. On y suit étape par étape la lecture des touches, la validation, la transmission du code via I2C, ainsi que le pilotage des retours lumineux. Ce schéma met en lumière la logique temporelle du système, depuis l'interaction physique jusqu'aux notifications réseau.

13. 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.

- **Limitation du Taux de Requêtes :**

Un système avec Flask-Limiter et Redis permet de limiter le nombre de requêtes par adresse IP, ce qui aide à éviter les attaques de type déni de service (DoS).

- **Sécurisation des Communications :**

Des en-têtes de sécurité (headers) sont ajoutés avec Flask-Talisman ou manuellement pour se protéger contre certaines attaques (comme le clickjacking).

La communication entre les modules se fait via le protocole MQTT avec des identifiants spécifiques pour sécuriser les échanges.

- **Accès à la Base de Données :**

L'utilisation de SQLAlchemy permet d'écrire des requêtes sécurisées, ce qui limite les risques d'injection SQL (sauf sur les endpoints créés volontairement vulnérables pour les exercices).

- **Journalisation et Suivi des Événements :**

Les actions importantes (connexion, ouverture de la serrure, actions d'administration...) sont enregistrées dans un fichier log.

Ces événements peuvent aussi être suivis en temps réel grâce à SocketIO.

- **Séparation des Endpoints Vulnérables pour le Challenge :**

Des pages spécifiques ont été créées pour les challenges de cybersécurité. Elles sont isolées du reste de l'application pour ne pas compromettre son bon fonctionnement.

14. Conclusion et perspectives.

14.1 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'une interface web, communication MQTT, gestion des connexions utilisateurs et suivi des événements. Cela m'a permis d'améliorer mes compétences en développement web et en protection des applications.

Grâce aux exercices 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.

Evolutions envisagées

Pour améliorer le projet, plusieurs pistes peuvent être envisagées :

- **Renforcement de la sécurité :**

Pour renforcer la sécurité, il serait utile de mieux contrôler les données saisies par l'utilisateur, et d'utiliser des méthodes plus sûres pour protéger les mots de passe, comme le hachage renforcé.

Des tests de sécurité réguliers, comme des audits ou des tests d'intrusion, permettraient aussi de détecter rapidement les failles du système.

- **Optimisation des performances :**

La gestion des connexions utilisateurs et les échanges via MQTT pourraient être améliorés pour rendre le système plus rapide et fluide.

Revoir certaines parties sensibles du code aiderait aussi à mieux gérer les erreurs et éviter les bugs.

- **Extension fonctionnelle :**

Il serait intéressant d'ajouter un outil pour centraliser les journaux (logs) et surveiller en temps réel ce qui se passe dans le système.

On pourrait aussi créer un tableau de bord avec des alertes et des statistiques pour mieux suivre l'utilisation et repérer les attaques.

- **Mise en production sécurisée :**

Avant de mettre le projet en ligne, il faudrait sécuriser la configuration avec le HTTPS et une bonne gestion du serveur MQTT.

Il est aussi important de rédiger une documentation claire et de former les utilisateurs pour assurer un bon suivi du système.

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.

**Partie Personnelle
BRUSSET Abel
(IR 2)**

14.2 Les étapes à réaliser.

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 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**

Installation/Configuration :

- Environnements de développement Raspberry Pi ZeroW

Mise en œuvre :

- Programmation Qt/C++ et/ou Python
- Technologie NFC
- Bus SPI
- GPIO (libgpod)

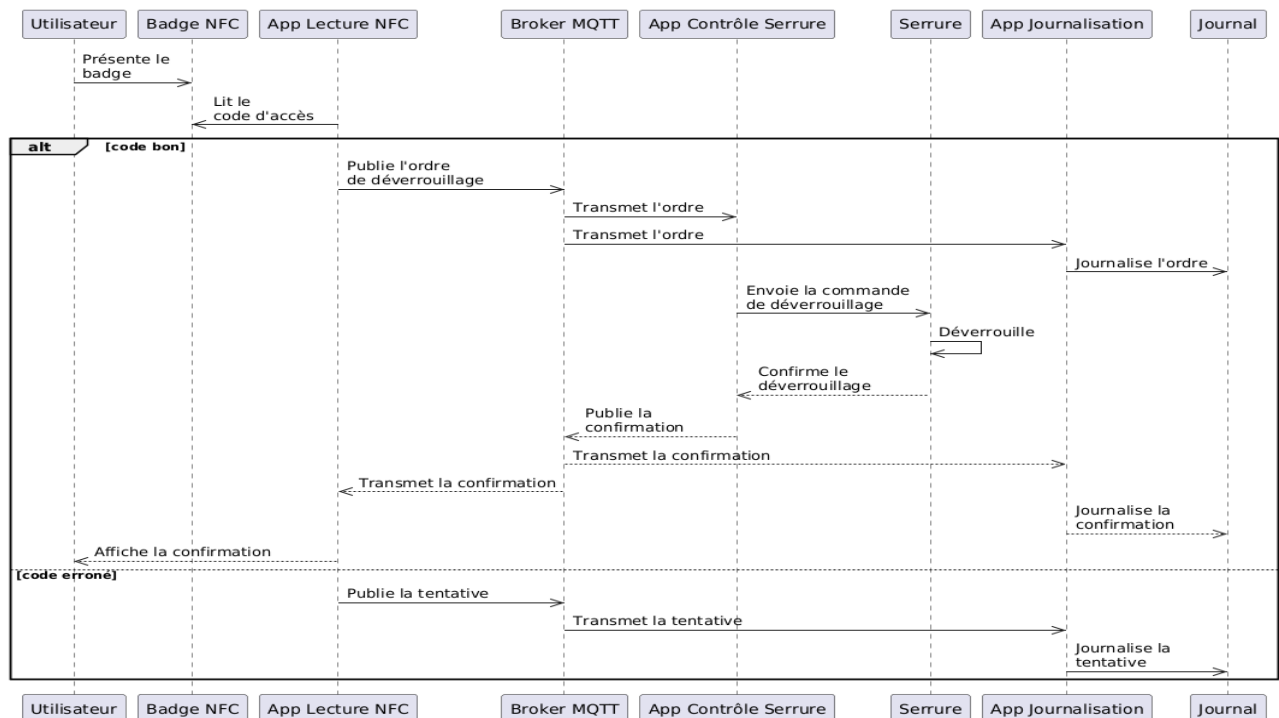
Réalisation :

- 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
- Dossier de développement

14.3 Scénario.



15. La serrure.

15.1 Test de la serrure sur Arduino.

Dans un premier temps je me suis renseigné sur le fonctionnement de la serrure et de ces possibilités utilisations avec la raspberry pi, j'ai trouvé les premières informations intéressantes sur le constructeur de la serrure.

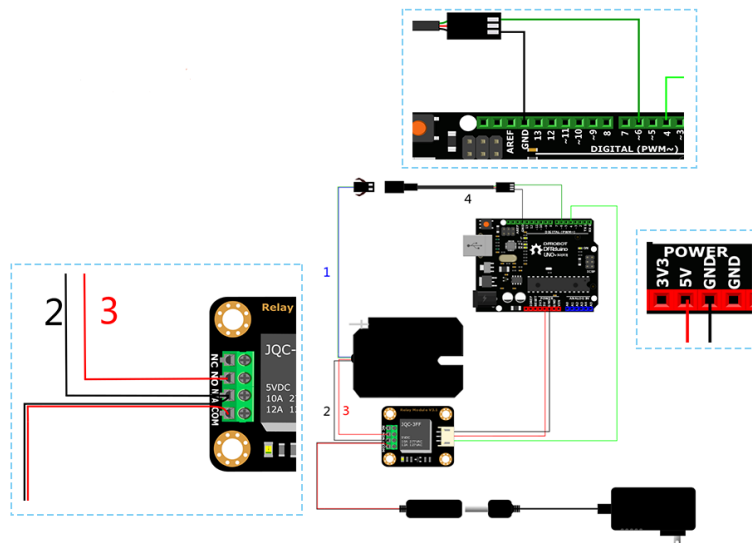
<https://wiki.dfrobot.com/Electric%20Solenoid%20Lock%20SKU:%20FIT0620>

Nous avons comme information :

- la tension de fonctionnement 9V en alternatif et 12 V en courant continue
- le courant de fonctionnement 1,5 A

⚠ Un temps d'activation de plus de 5 sec peut endommagé l'électrovanne

Dans un second temps nous avons tester la serrure grâce à une arduino.



On peut voir sur le schema :

- Une arduino
- La serrure
- Le relai
- Le câble d'alimentation

Nous avons mis dans l'arduino un code, le code ressemble a ceci :

```
* *****
* @brief Electric Solenoid Lock

* @copyright [DFRobot](https://www.dfrobot.com), 2016
* @copyright GNU Lesser General Public License

* @author [Xiaoyu](Xiaoyu.zhang@dfrobot.com)
* @version V1.0
* @date 2019-03-11

* GNU Lesser General Public License.
* All above must be included in any redistribution
* *****/
int Relay = 4;

void setup() {
  Serial.begin(57600);
  pinMode(6, INPUT_PULLUP);      //Set Pin6 as output
  pinMode(Relay, OUTPUT);
}
void loop() {
  int singal=digitalRead(6);
  Serial.println(singal);
  digitalWrite(Relay, HIGH);
  delay(2000);
  digitalWrite(Relay, LOW);
  delay(2000);
}
```

```
void setup() {
  Serial.begin(57600);
  pinMode(6, INPUT_PULLUP);      //Set Pin6 as output
  pinMode(Relay, OUTPUT);
}
```

Ceci initialisé les parametre de connexion, du pin brancher, et du relai

```
void loop() {
  int singal=digitalRead(6);
  Serial.println(singal);
  digitalWrite(Relay, HIGH);
  delay(2000);
  digitalWrite(Relay, LOW);
  delay(2000);
}
```

Et ceci pour crée un signal en continue de 2 sec pour activé et désactiver la serrure

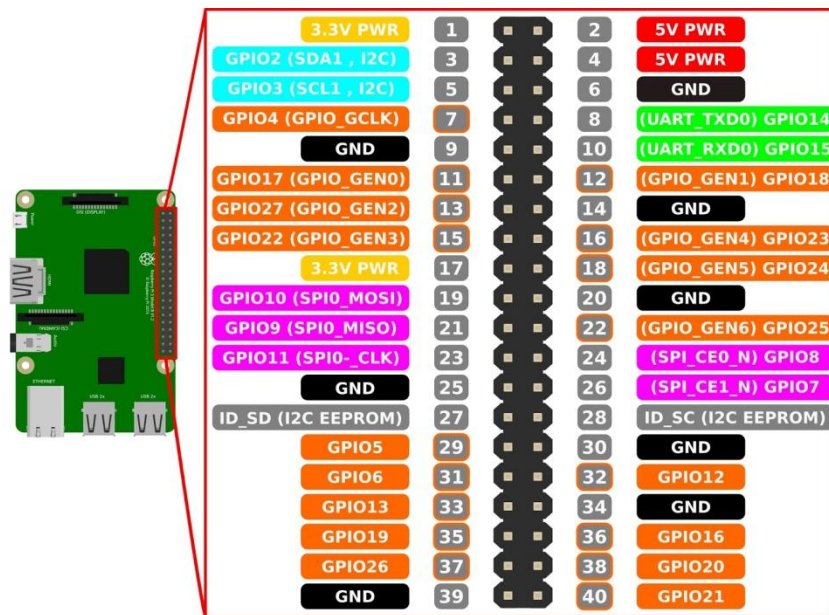
Ce test a marché parfaitement

15.2 Test de la serrure sur Raspberry Pi 4.

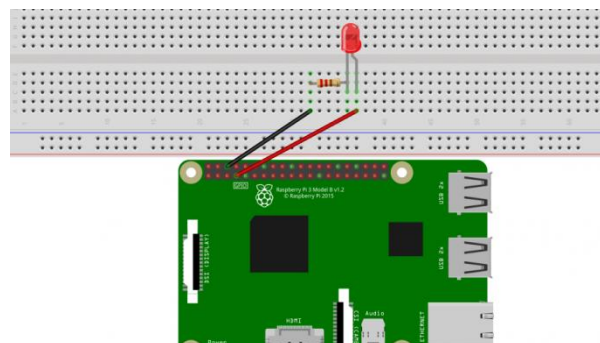
Par la suite, nous avons fait nos tests sur la raspberry pi. La serrure a été donné à mon collègue de travail pour qu'il continue les recherches de son côté, donc pour simuler la serrure nous l'avons remplacé par une LED

Il y a sur la RPI (RasPberry pi) des PIN de connexion qui ont chacune leur utilisation spécifique.

Nous avons branché cette LED sur la raspberry pi de la manière suivante :



La LED est brancher sur la raspberry pi sur le pin 6 (GND) et pin 7 (GPIO 4) avec une résistance relié sur la partie (–) de la LED



Nous avons effectué sur la Raspberry un code de test pour la LED, le code est le suivant

```

import RPi.GPIO as GPIO
import time

# Définition du numéro du GPIO
LED_PIN = 4

# Configuration du mode des broches
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)

try:
    # Allumer la LED
    GPIO.output(LED_PIN, GPIO.HIGH)
    print("LED allumée")
    time.sleep(2) # Attendre 2 secondes

    # Éteindre la LED
    GPIO.output(LED_PIN, GPIO.LOW)
    print("LED éteinte")
    time.sleep(2)
finally:

    # Nettoyer les GPIO avant de quitter
    GPIO.cleanup()

```

Ce code va configurer le pin utilisé, paramétrer le GPIO, puis allumé pendant 2 secs et l'éteindre pendant 2 secs. Le programme va s'arrêter par la suite avec un nettoyage du GPIO.

Le code fonctionnera parfaitement donc la simulation de la serrure est un succès.

16. Connexion avec le broker MQTT.

Sur la même Raspberry pi j'ai configuré un Broker MQTT

Un broker MQTT permet aux appareils de communiquer efficacement en envoyant des messages via un système de topics, (par exemple topic/serrure) sans connexion directe entre eux. On peut aussi grâce au topic lancer un programme à distance.

Grâce à cela nous allons pouvoir lancer un programme à partir d'une API par exemple, et c'est ce que nous recherchons.

Le code qui va nous permettre de nous connecter au broker pour allumer la LED par la suite va être le suivant :

```
import RPi.GPIO as GPIO
import time
import paho.mqtt.client as mqtt

# Définition du numéro du GPIO
LED_PIN = 4

# Configuration du mode des broches
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)

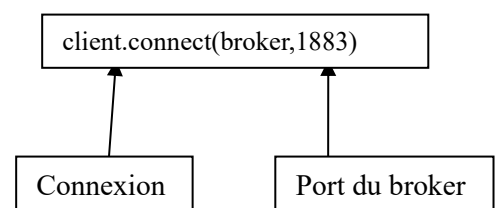
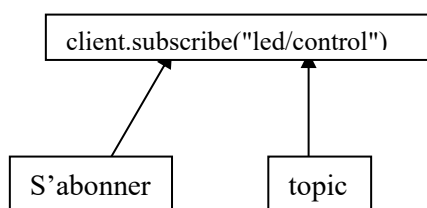
# Callback quand un message est reçu
def on_message(client, userdata, message):
    msg = message.payload.decode()
    if msg == "ON":
        GPIO.output(LED_PIN, GPIO.HIGH)
        print("LED allumée")
    elif msg == "OFF":
        GPIO.output(LED_PIN, GPIO.LOW)
        print("LED éteinte")

# Configuration du client MQTT
broker = "broker.hivemq.com" #
Adresse du broker MQTT

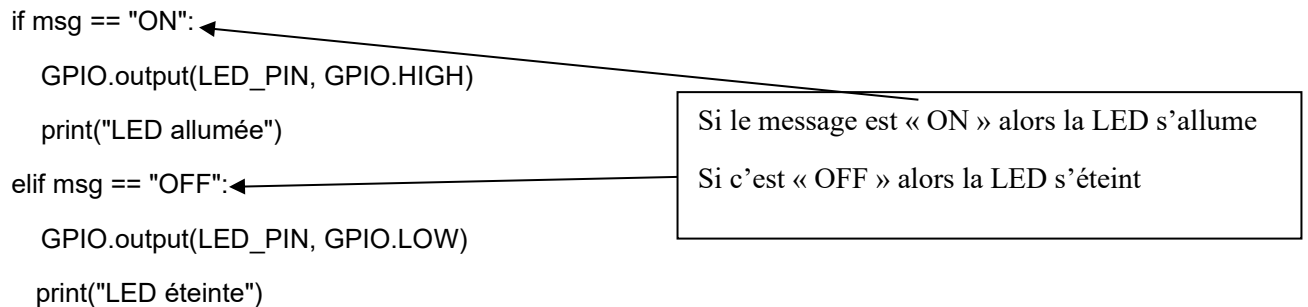
client = mqtt.Client()
client.on_message = on_message

client.connect(broker, 1883)
client.subscribe("led/control")

try:
    client.loop_forever() # Boucle pour
écouter les messages
finally:
    GPIO.cleanup() # Nettoyer les GPIO
avant de quitter
```



```
if msg == "ON":
    GPIO.output(LED_PIN, GPIO.HIGH)
    print("LED allumée")
elif msg == "OFF":
    GPIO.output(LED_PIN, GPIO.LOW)
    print("LED éteinte")
```



Si le message est « ON » alors la LED s'allume
Si c'est « OFF » alors la LED s'éteint

Je me suis servi pour la connexion avec le broker, d'une application nommé MQTT.fx qui a comme utilité de pouvoir s'abonner à mon topic et d'envoyer un message dessus.

J'ai par la suite lancé mon programme sur la raspberry pi et envoyé le message « ON » sur le topic (« led/control »).

La LED c'est allumé, puis je les éteint avec le message « OFF ».

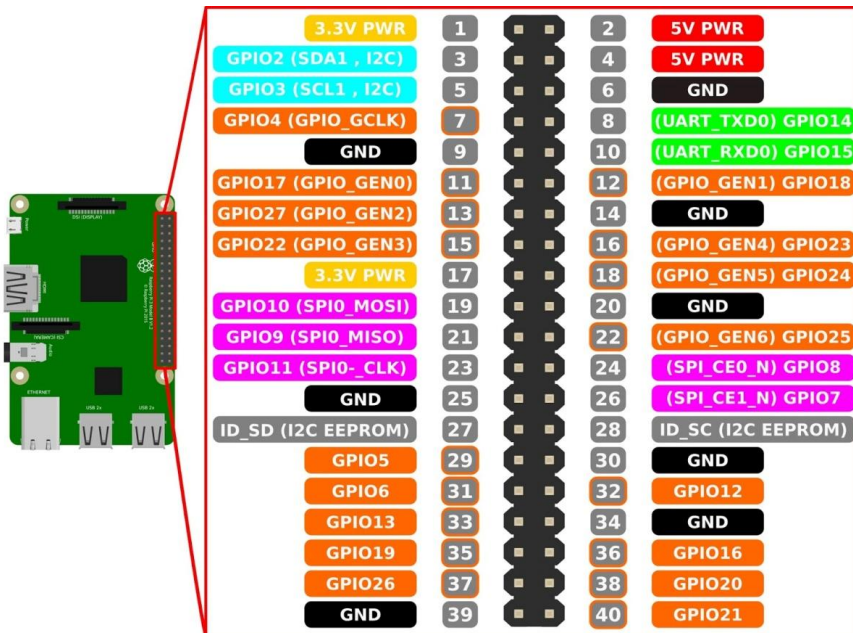
17. Le module NFC.

La seconde partie consiste à utiliser un module NFC (c'est un système de badge connecté) qui nous servira à ouvrir la serrure.

Dans un premier temps on va voir comment brancher le module NFC sur la raspberry pi. Il y a sur la RPI (Raspberry pi) des PIN de connection qui ont chacune leur utilisation spécifique.

Voici un schéma :

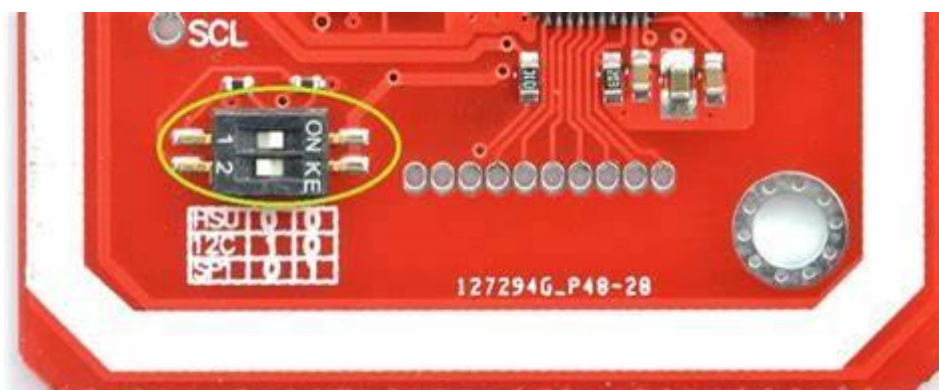
(A savoir que la connection entre le module NFC et la RPI doit être reliée par un bus SPI (Serial Peripheral Interface))



Les PIN à utilisé et qui doit être relié au module NFC :

Pin de la RPI	Module NFC
Pin 2 (5V)	VCC
Pin 6 (GND)	GND
Pin 23 (GPIO 11)	SCK
Pin 19 (GPIO 10)	MOSI
Pin 21(GPIO 9)	MISO
Pin 24 (GPIO 8)	SS

Il y a aussi sur le module NFC un paramètre à changer manuellement sur la carte.



Les deux interrupteurs définissent le bus de communication du module NFC. Pour activer le SPI, mettez l'interrupteur 1 à gauche et le 2 à droite. Ainsi, le module fonctionnera uniquement en SPI.

Il y a 2 sortes de badge qui on la même utilité :



Ils sont utilisés comme clé pour déverrouiller la serrure avec le module NFC

Par la suite du branchement, j'ai utilisé un petit code pour tester le module NFC, le code est le suivant :

```
import board
import busio # Assurez-vous que busio est bien importé
from digitalio import DigitalInOut
from adafruit_pn532.spi import PN532_SPI

# Configuration SPI
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
cs_pin = DigitalInOut(board.D8) # Pin GPIO8 (CE0)

# Initialisation du module NFC
pn532 = PN532_SPI(spi, cs_pin, debug=True)

# Vérification de la connexion
ic, ver, rev, support = pn532.firmware_version
print(f"PN532 firmware version: {ver}.{rev}")

# Configurer le module pour lire les cartes RFID
pn532.SAM_configuration()

print("Approchez une carte NFC/RFID...")
while True:
    uid = pn532.read_passive_target(timeout=0.5)
    if uid is not None:
        print("Carte détectée ! UID:", [hex(i) for i in uid])
```

Ce code va afficher les informations du badge approché du module NFC, comme l'ID du badge.

Si les informations de la carte apparaissent cela veut dire que le module NFC fonctionne correctement.

Aucun souci de ce côté-là.

Puis je me suis connecté au broker MQTT connecter au préalable et crée un nouveau topic

```
import time
import board
import busio
import paho.mqtt.client as mqtt
from adafruit_pn532.i2c import PN532_I2C

# Configuration du PN532 (en mode I2C)
i2c = busio.I2C(board.SCL, board.SDA)
pn532 = PN532_I2C(i2c, debug=False)

# Vérification du PN532
version = pn532.firmware_version
if not version:
    raise RuntimeError("Impossible de détecter le PN532")
else:
    print(f"PN532 détecté ! Firmware Version: {version[1]}.{version[2]}")

# Configuration du broker MQTT
BROKER_ADDRESS = "192.168.4.57" # Remplace par l'adresse IP de ton broker
BROKER_PORT = 1883 # Port par défaut pour MQTT non sécurisé
MQTT_TOPIC = "DVIL/nfc" # Topic où publier les UID des cartes

# Initialisation du client MQTT
client = mqtt.Client()
client.connect(BROKER_ADDRESS, BROKER_PORT, 60)

def publish_nfc_data():
    """Détection des cartes NFC et publie les UID sur MQTT."""
    print("Approchez une carte NFC...")
    while True:
        uid = pn532.read_passive_target(timeout=1) # Détection passive
        if uid:
            print(f"UID détecté : {[hex(i) for i in uid]}")
            # Publier l'UID sous forme JSON sur MQTT
            client.publish(MQTT_TOPIC, f'{{"uid": {[hex(i) for i in uid]}}}')
            time.sleep(1) # Pause pour éviter les publications en boucle
        else:
            time.sleep(0.5)
```

```
try:
    print("En attente de cartes NFC...")
    publish_nfc_data()
except KeyboardInterrupt:
    print("\nArrêt du programme.")
finally:
    client.disconnect()

BROKER_ADDRESS = "192.168.4.57" # Remplace par l'adresse IP de ton broker
Ceci est l'adresse du Broker MQTT donc l'adresse de la RPI
BROKER_PORT = 1883 # Port par défaut pour MQTT non sécurisé
Le port du Broker est 1883
MQTT_TOPIC = "DVIL/nfc" # Topic où publier les UID des cartes
Le nouveau topic est DVIL/nfc
```

Après le badge scanné par le module NFC les informations du badge sont envoyées sur le broker MQTT, et donc peut être récupéré

J'ai ensuite crée un schéma de fritzing sur l'application Fritzing pour le branchement entre le Module NFC (PN532) et la Raspberry Pi Zero.

Voici à quoi il ressemble:

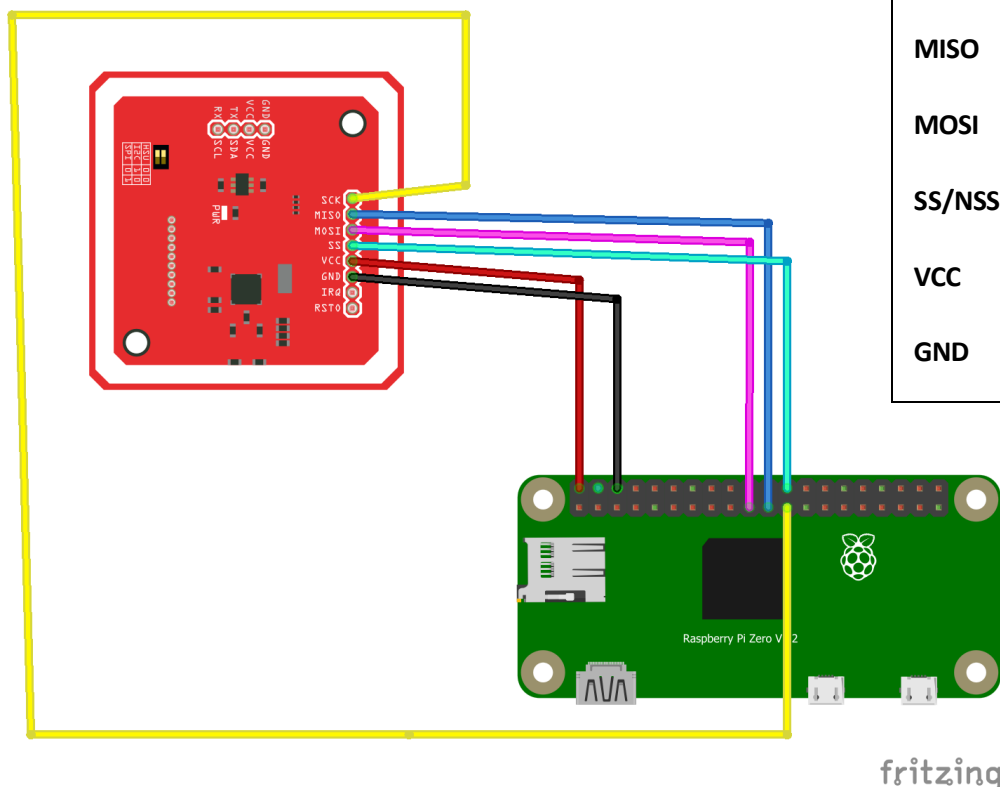


Schéma de cablage	
PN532 Pin	Raspberry Pi GPIO Pin
SCK	GPIO11 (SPI_CLK, Pin 23)
MISO	GPIO9 (SPI_MISO, Pin 21)
MOSI	GPIO10 (SPI_MOSI, Pin 19)
SS/NSS	GPIO8 (SPI_CE0, Pin 24)
VCC	3.3V (Pin 1 ou 17)
GND	GND (Pin 6, 9, 14, etc.)

Partie Personnelle
RIVAREL
Marius(ER)

18. Introduction :

Mon but au sein du projet est l'étude et la conception de carte électronique, ainsi que le raccordement de ces dernières avec les différents composants qui composent ce projet. Je dois m'occuper du schéma structurel ainsi que du routage final de la carte. Ainsi que l'intégration des deux bus de données, le bus SPI pour le NFC, ainsi que le bus I2C pour le clavier matriciel.

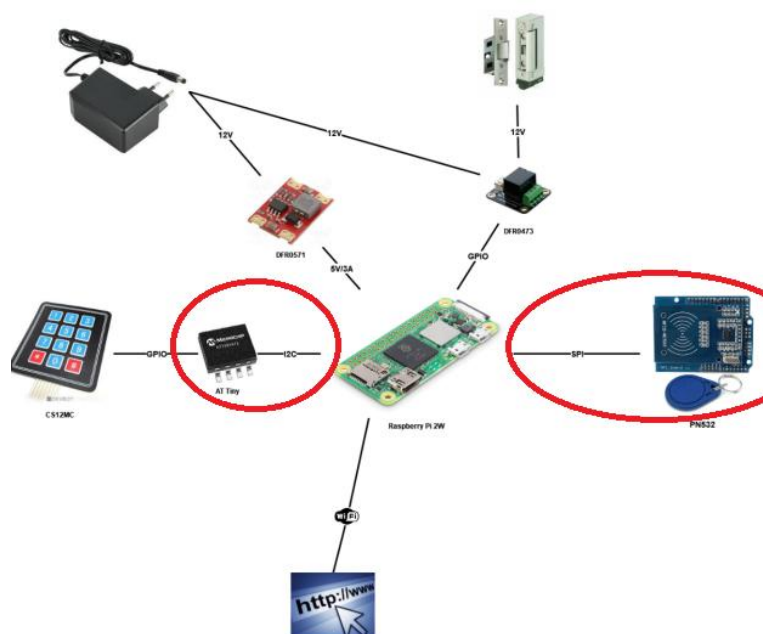
Pour ce faire, je dispose d'une serrure que je dois relier à un transistor qui fait office d'un relais, d'un ATtiny 1616 qu'on utilise à la place d'un Arduino Uno.

Je dois aussi m'occuper de l'alimentation de toutes les parties du projet, afin d'éviter les pertes et un courant trop faible envoyés dans les composants.

Et pour ce faire il faut que les composants soient compatibles entre eux.

19. Analyse du fonctionnement des bus de transmission :

Le projet DVIL utilise deux bus différents, l'I2C et le SPI, cependant comme je n'ai pas pu mettre en œuvre le SPI pour le moment, je ne présenterai ici que le bus I2C.



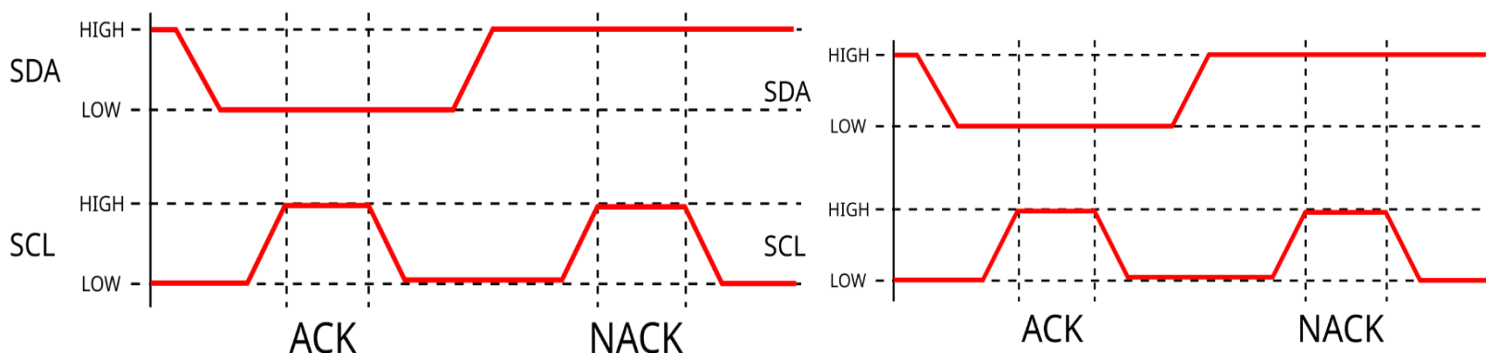
20. Le bus I2C :

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).

20.1 Utilisation de l'I2C :

L'I2C a l'avantage d'être un bus simple à utiliser car il peut communiquer avec n'importe quel microcontrôleur. Et du fait de sa simplicité, il est utilisé sur de nombreux circuits intégrés. Il dispose aussi d'une vitesse de transmission allant de 100 Kbit/s (Standard Mode) à 5 Mbit/s (Ultra-fast mode)

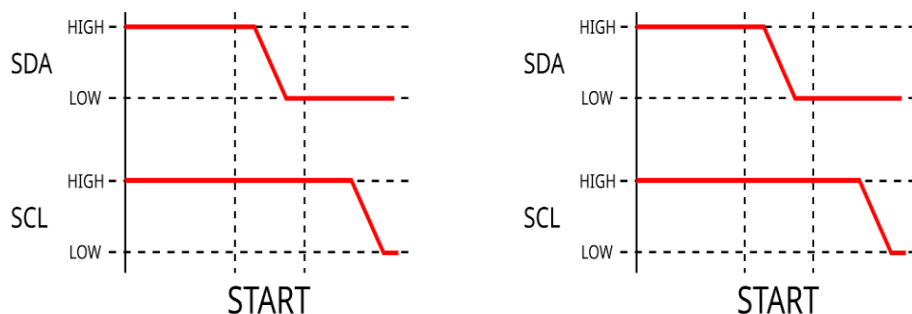


20.2 Structure d'une trame I2C :

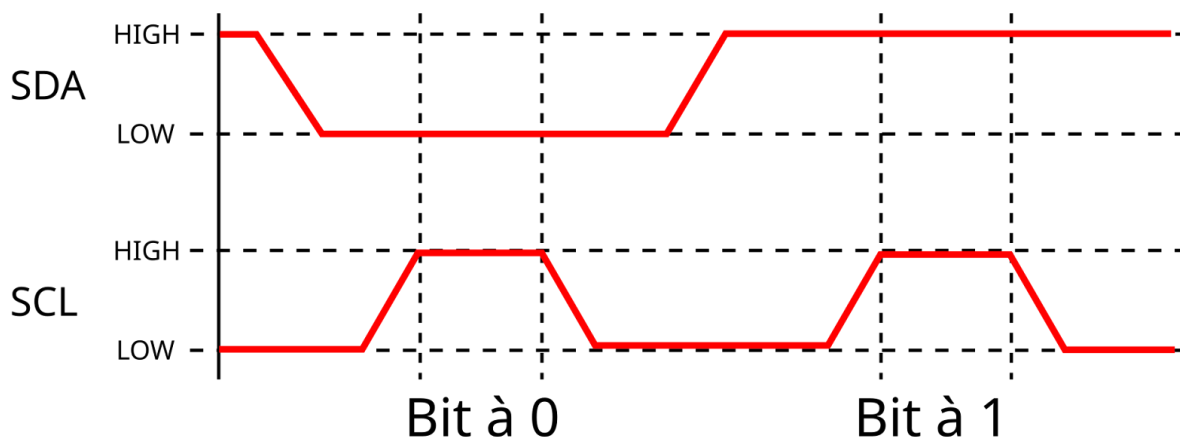
La trame I2C utilise un fonctionnement plutôt simple, avec un niveau haut à 1, et un niveau bas à 0.

Le système de validation d'un signal se fait sur les niveaux haut de l'horloge (SCL) et le niveau de (SDA) doit rester stable jusqu'à ce que l'horloge retourne au niveau bas.

Cependant, le START et le STOP ne sont pas affectés par cette règle, au contraire, par conséquent le bit de START et de STOP transite à un autre niveau pendant qu'ils sont synchronisés avec SCL.



L'acquittement est réalisé afin d'éviter une erreur de transmission et s'assurer que les données ne soient pas envoyées dans le vide. C'est pourquoi il y a deux types d'acquittements, le ACK (Acknowledge) et le NACK (Non-Acknowledge). Le ACK signifie à signaler la bonne réception de l'octet, et le NACK signifie qu'il y a soit une erreur, soit à indiquer la fin de la transmission si c'est le maître qui est en récepteur.

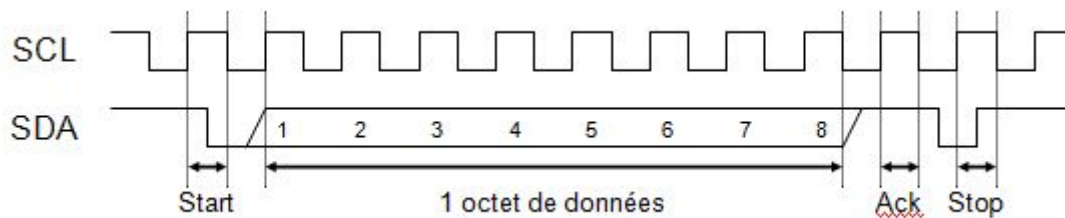


L'adressage :

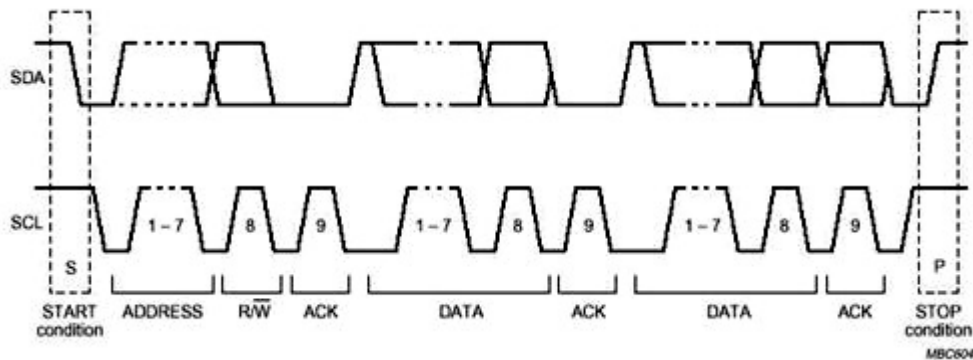
Avant d'envoyer les données dans une trame, le maître doit annoncer à qui il demande la communication. L'adressage se fait sur 7 bits, suivi du bit R/W, si le bit est à 0, le maître signale qu'il va envoyer des octets, et que l'esclave doit les lire. Et le 1 signifie que le maître veut recevoir des octets, et que l'esclave doit lui fournir.

Les données :

La transmission de données se fait en paquet de 8 bits, soit un octet, et pour transmettre des données plus importantes, il faut alors les segmenter.



Et si on assemble tout ceci, on obtient une trame I2C complète.



21. Début du projet :

Mes premiers objectifs sont tout d'abord de me documenter sur les différents composants à ma disposition, il y a :

- Une serrure à solénoïde FIT0620, une serrure qui utilise le magnétisme généré via l'électricité.



- Un module relais DFR0473



- Un clavier souple à 12 touches (4x3) matriciel



- Une raspberry PI Zero 2W



- Et un ATtiny1614 qui ne sera pas dès le début mais qui est à ce moment remplacé par une Arduino Uno qui a les mêmes propriétés.

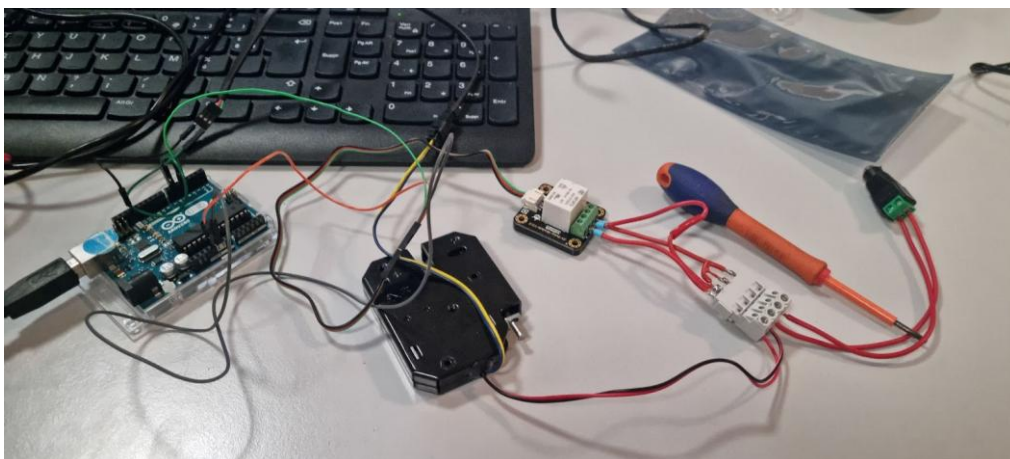


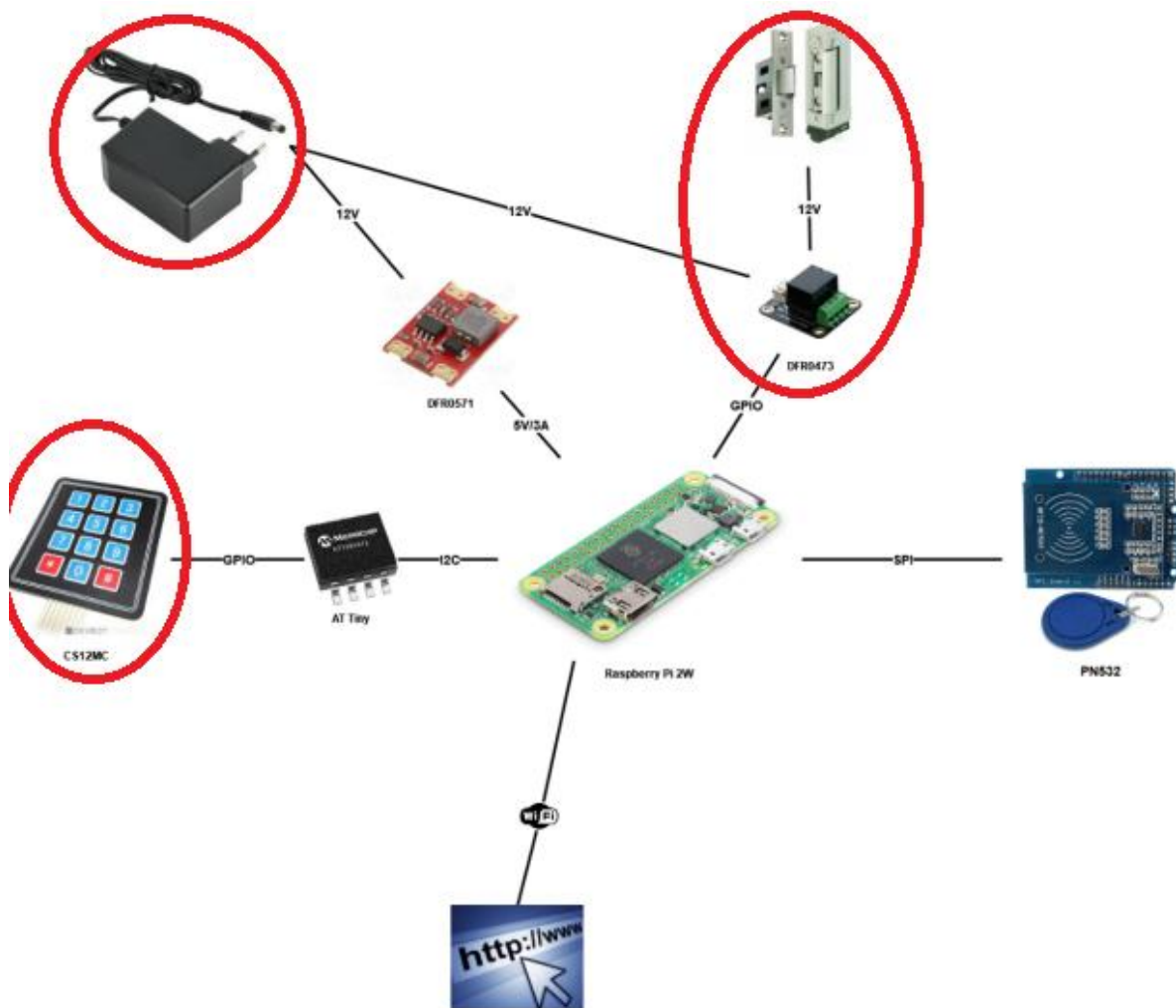
1er câblage :

Ce premier câblage a pour but de mettre en lien l'arduino, le relais, ainsi que la serrure afin de voir si ces composants parvenaient à communiquer entre eux, mais aussi à comprendre le fonctionnement de la serrure, qui à le même principe de fonctionnement qu'une LED.

Un niveau haut, et elle s'ouvre, un niveau bas et elle se ferme, cependant l'état de la serrure (0 ou 1) ne dépendant pas de son ouverture, mais si l'embout en métal sur lequel elle se raccroche est présent ou non.

À noter qu'il faut extrêmement faire attention à la serrure car elle ne peut supporter d'être ouverte plus de 5 secondes, au risque d'une surchauffe et qu'elle devienne inutilisable.





22. 1ère réunion de projet :

Avec les deux IR qui sont avec moi sur le projet : Lysandre VIALET et Abel BRUSSET, nous avons effectué la première réunion de projet afin de prendre quelques décisions au sujet du placement des différents composants sur le coffre.

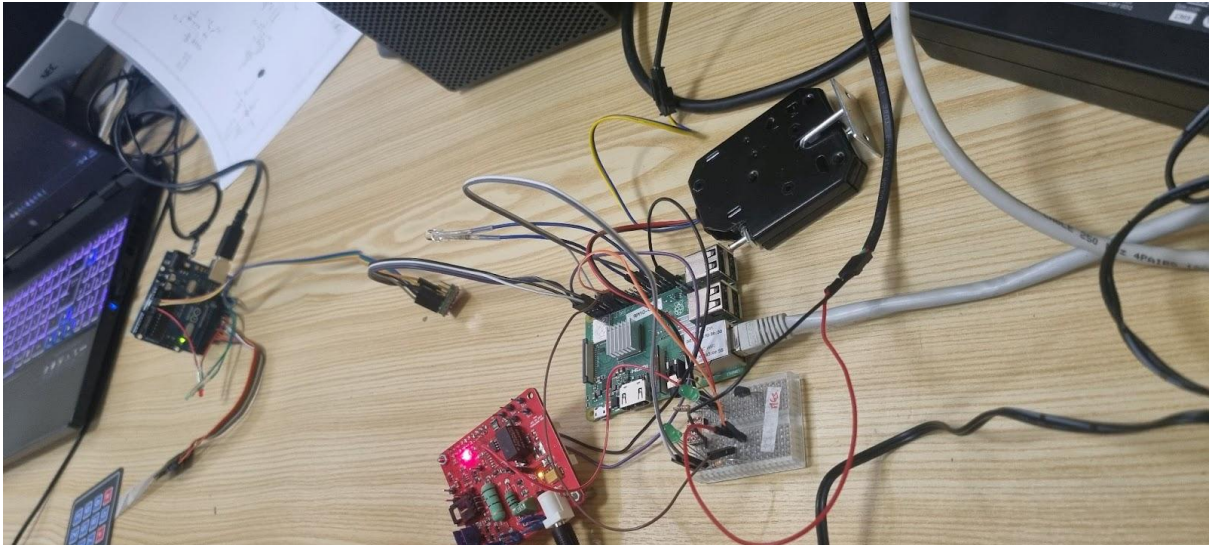
Nous avons convenu que le clavier serait sur le devant de la boîte afin d'éviter de tirer sur les fils qui risqueraient de s'user avec le temps avant de céder.

Nous avons également convenu que les composants devraient être cachés et le moins accessible possible pour ne pas être modifiés et que la boîte ne soit pas ouverte trop simplement.

C'est pourquoi le choix a été de faire un double fond car cela correspond parfaitement à nos objectifs.

On en a également profité pour faire le point sur les tâches de chacun.

2nd câblage :



Le relais se fait remplacer par un transistor MOSFET, l'utilisation est la même mais plus pratique et cela prend bien moins de place.

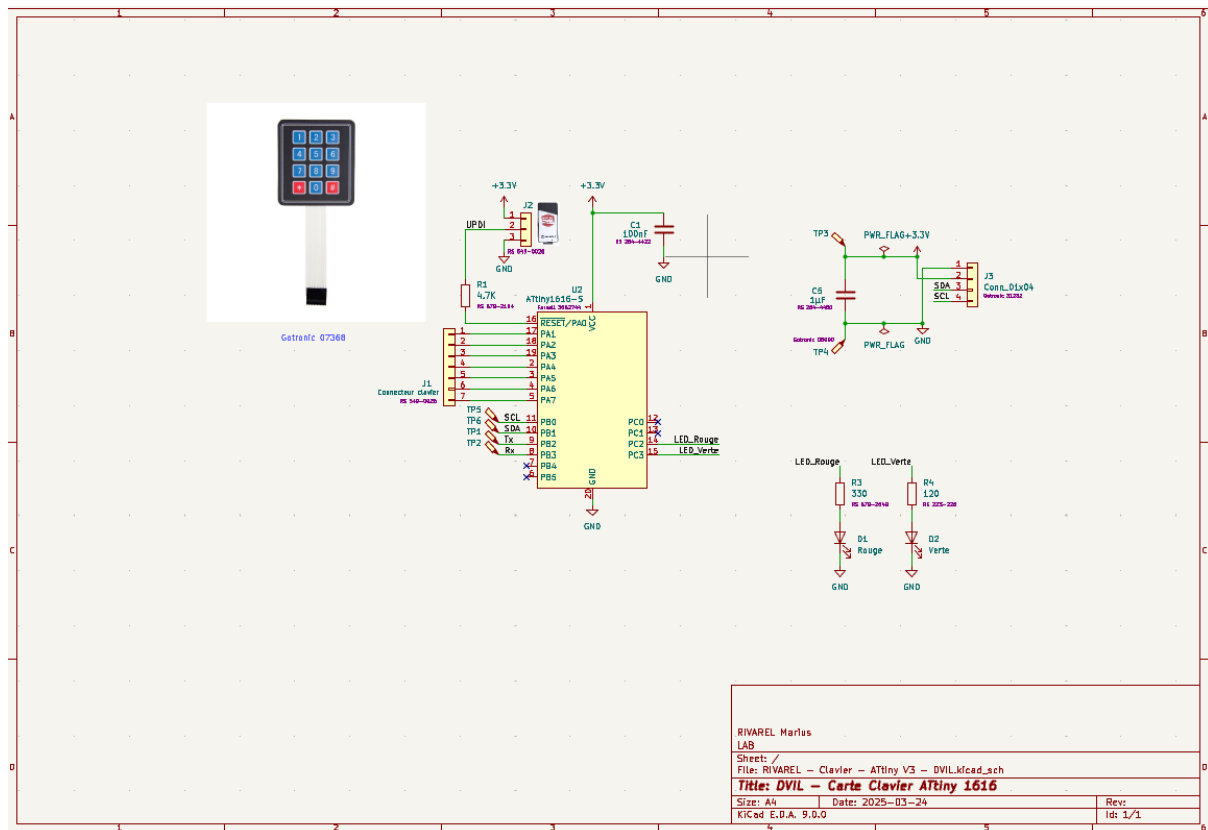
Et surtout il y est intégré la raspberry pi et l'API de Lysandre.

Désormais le code ouvre la serrure parfaitement, elle se referme peu après pour les problèmes de surchauffe.

Et via l'API, il est possible de l'ouvrir juste avec un clic sur l'application.

23. Schéma KICAD :

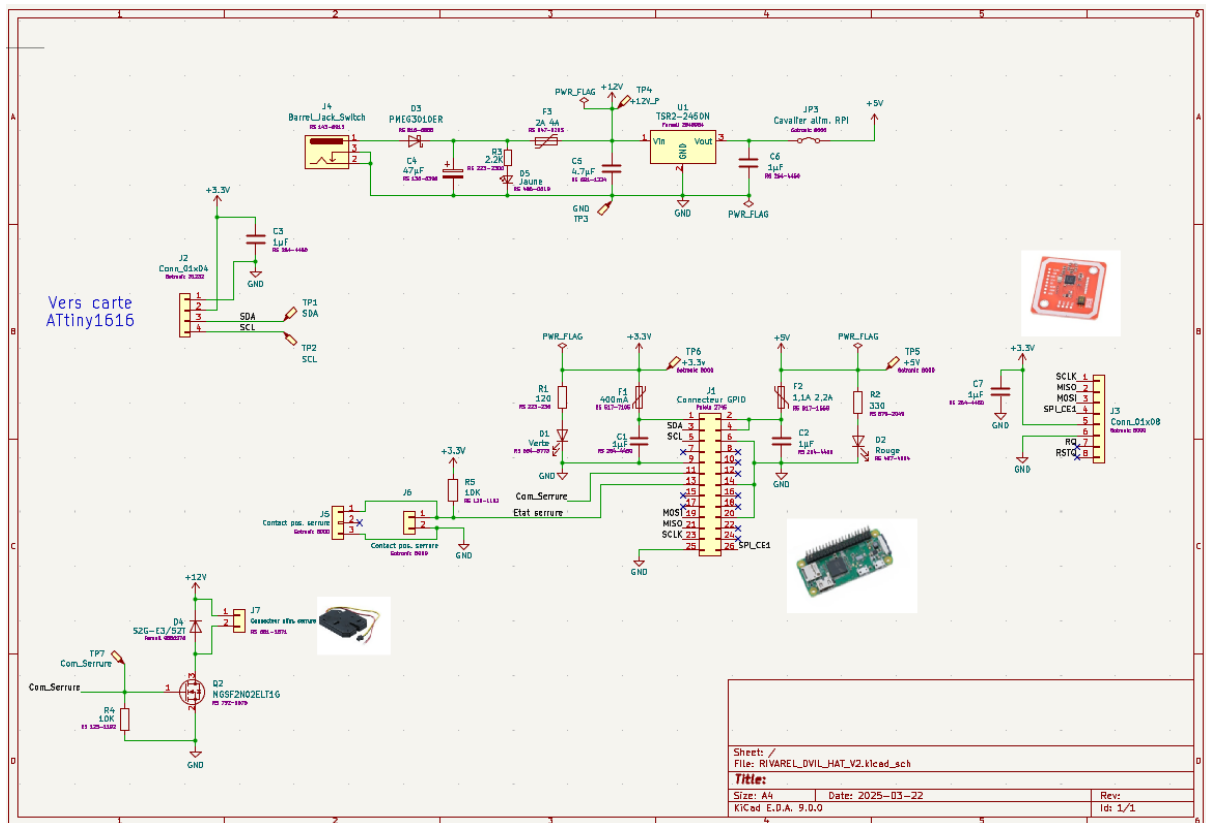
Les schémas ont été réalisé peu après le 1er câblage, et ont été amenés à évoluer au fil de l'avancement du projet, et vont peut-être encore évoluer.



Voici le premier schéma qui correspond à l'ATtiny 1616, il y a eu ce changement à cause du manque de broche du 1614, mais le principe de fonctionnement reste exactement le même.

Il y a l'ajout des deux LED qui sont présentes pour indiquer à l'utilisateur si le mot de passe fonctionne ou non.

Le clavier matricielle quant à lui se branche simplement sur les ports de l'ATtiny.



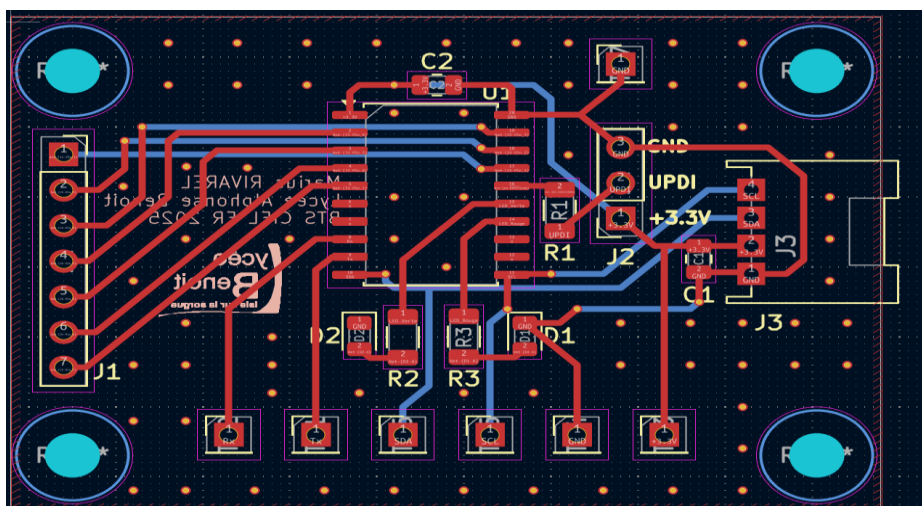
Sur ce schéma on peut y voir en bas à gauche, le transistor MOSFET remplaçant le relais, qui est ensuite directement relié à la serrure.

Au milieu, la raspberry pi zéro avec toutes les connexions aux GPIO.

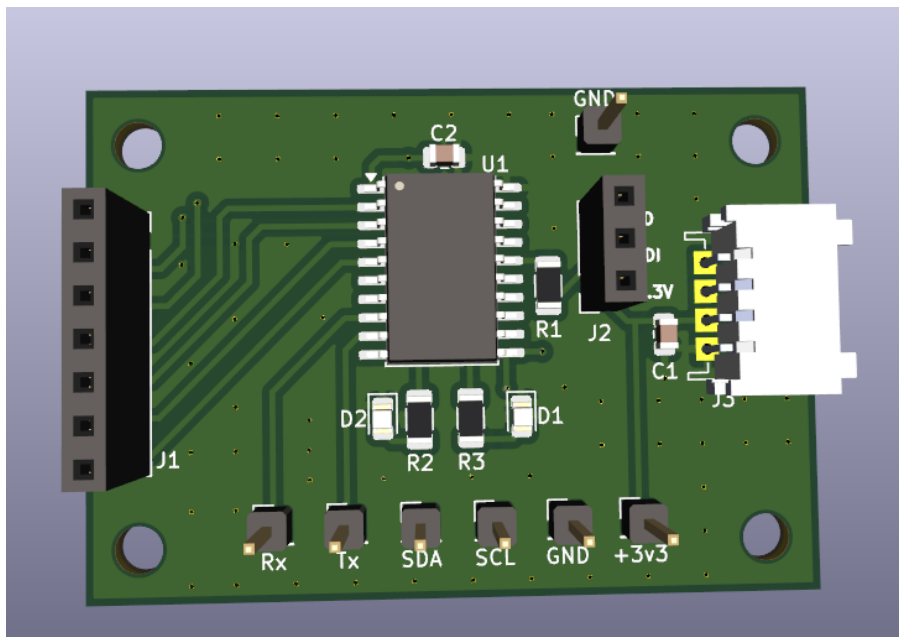
En haut l'alimentation de la serrure ainsi qu'un convertisseur CC-CC (Un convertisseur de tension vers un autre niveau de tension)

Route et plan 3D :

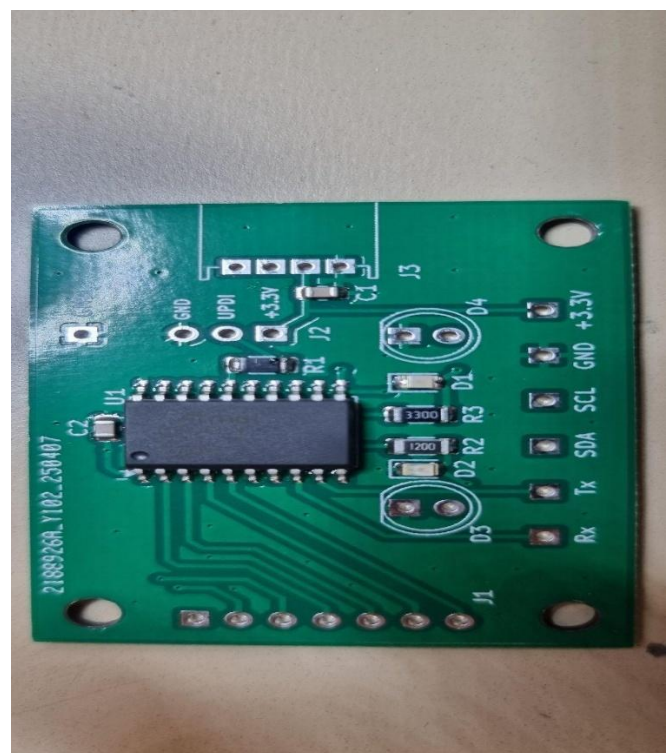
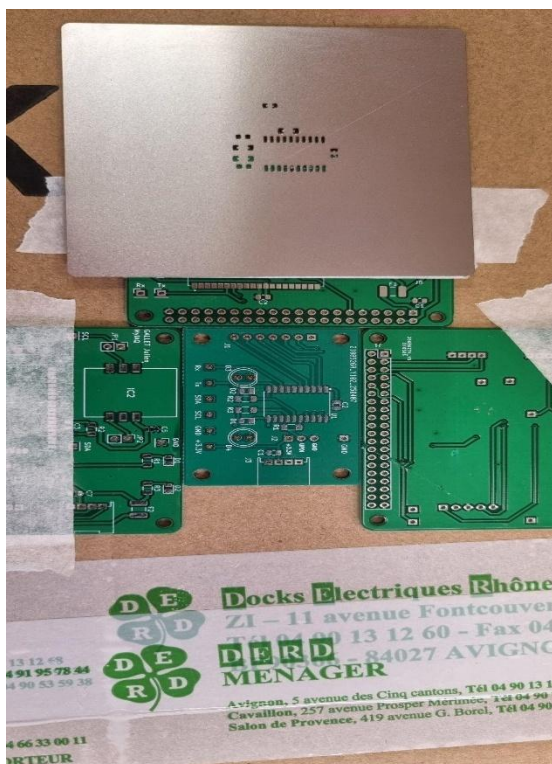
Voici le routage de la carte avec l'Attiny ainsi que le clavier 4x3



Ainsi que la prévisualisation avec l’affichage 3D intégré dans KICAD.

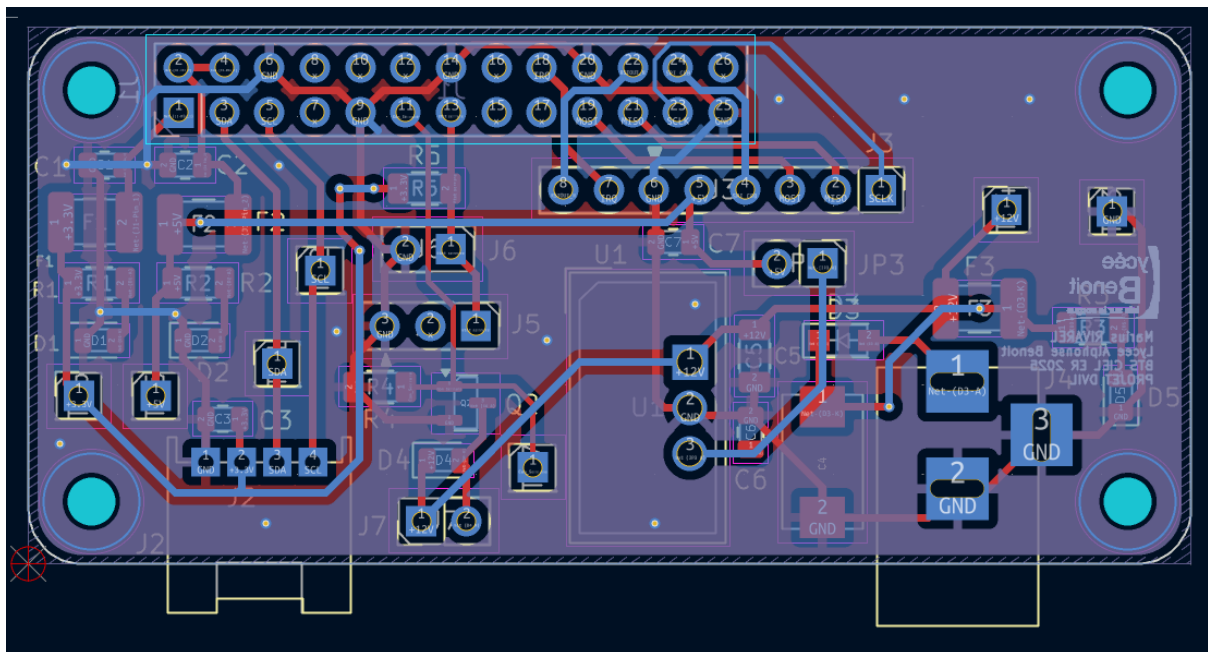
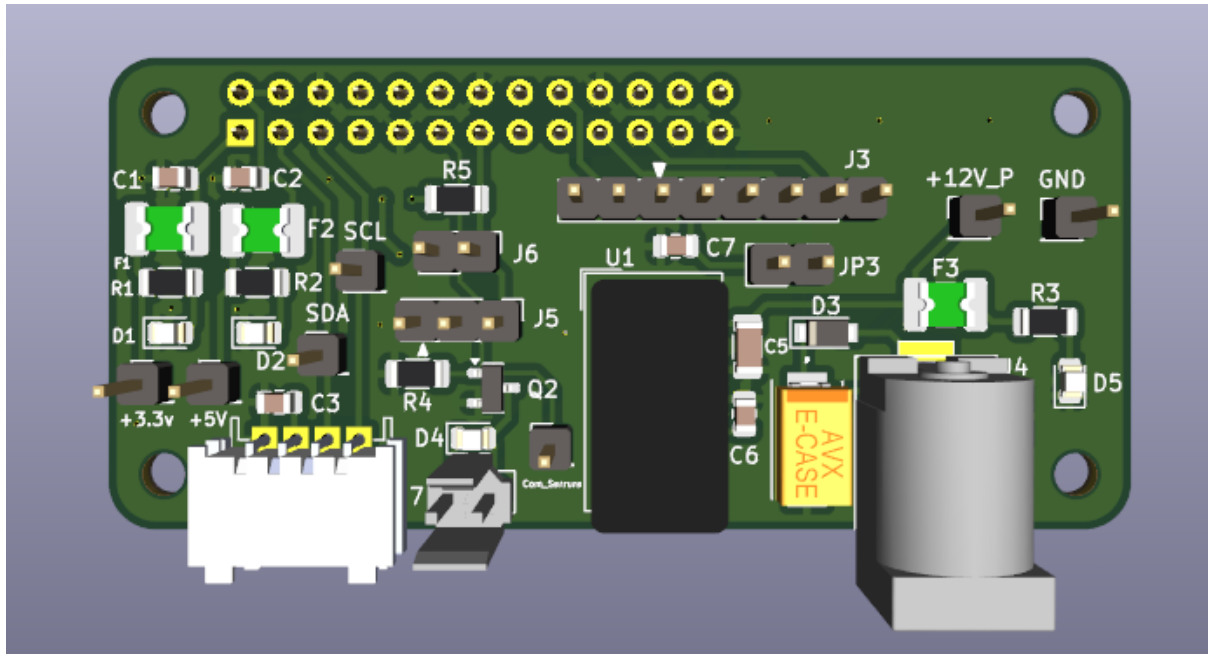


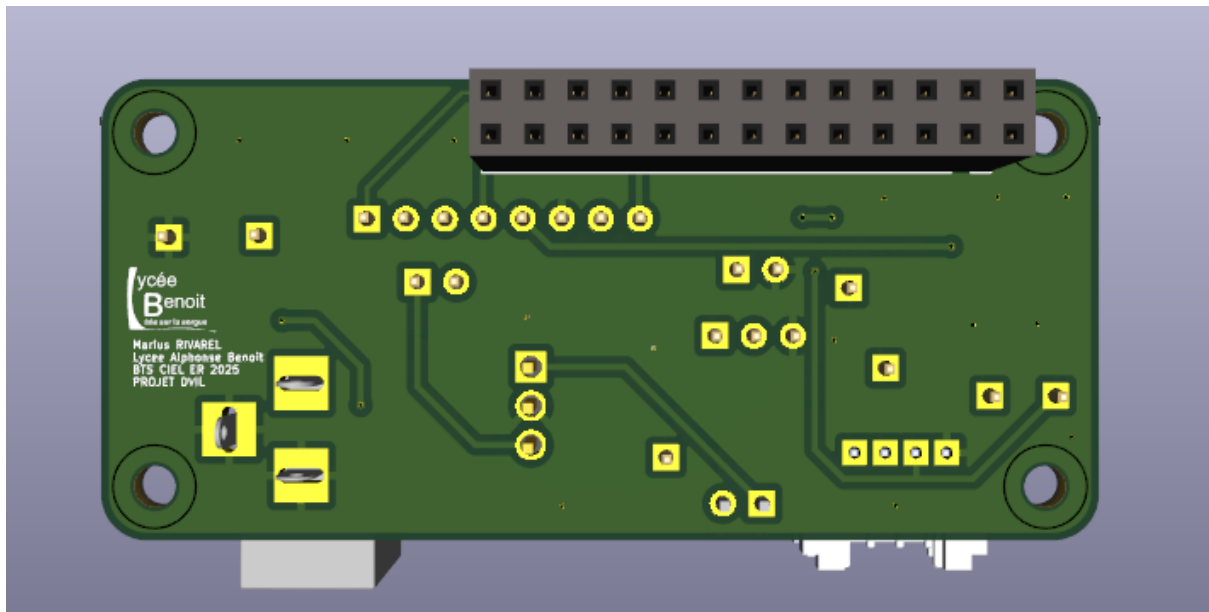
Une fois le résultat final obtenu, il reste plus qu’à commander et attendre que les pièces arrivent et que la carte soit fabriquée.



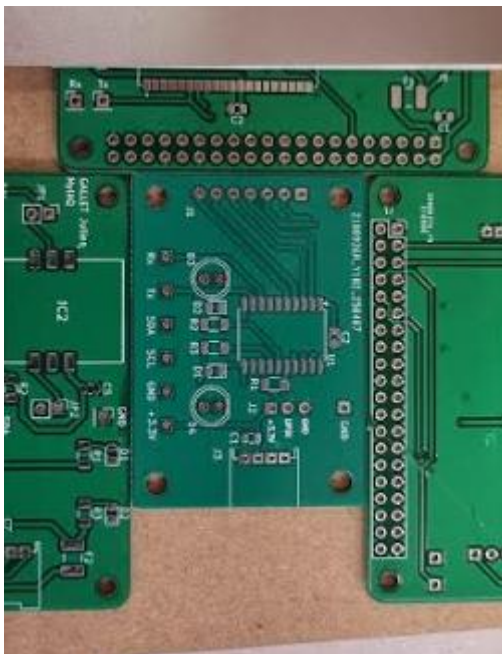
Il reste plus qu'à souder les composants CMS et traversants sur la carte, et on parvient au résultat final.

Pour ce qui est du HAT que l'on va mettre par-dessus la Raspberry PI Zero, voici le routage, ainsi que le résultat via l'affichage 3D :





Suite à cela, on peut sortir ce résultat en réel auprès d'une entreprise spécialisée dans la fabrication de ces cartes.



24. Compte rendu.

13/01 :

- Création du drive et d'un espace en ligne pour communiquer.
- Création d'un fichier Project personnel et commun.
- Réunion d'une heure avec M.Ortholand et M.Defrance, voir les notes de la réunion du 13/01.
- Mis au propre des notes.
- Lecture du dossier.

21/01 :

Après avoir réalisé les branchements basiques entre le relais et la serrure, je vais relier l'arduino tout en ayant vérifié au préalable que les connectiques que je veux relier le soit, et qu'elles ne soient pas en contact avec d'autres non voulues.

Ensuite, je vais aller changer le code fourni dans l'exemple, le même que l'on s'est servi pour faire le branchement, pour le moment on va utiliser un arduino, mais plus tard il faudra adapter ce schéma pour s'en servir avec un ATTINY.

Pour le moment le code comporte une problématique, il ne fait la lecture de l'état qu'une fois par boucle, or, il change deux fois d'état par boucle et toujours dans le même ordre, ce qui va nous renvoyer toujours le même résultat et c'est pour ça qu'il faut que j'intègre une double lecture pour obtenir la lecture de l'état à chaque changement d'état.

Le code fonctionne, cependant l'état lue est le même à chaque fois donc c'est à corriger pour le 27/01.

27/01 :

L'état lu est désormais le bon, car je lisais simplement le mauvais pin, il va maintenant falloir créer/adapter un code pour le clavier numérique.

J'ai donc suivi une video pour un clavier 4x4, avec explication du fonctionnement du clavier matriciel, il suffit alors d'adapter son code pour un clavier 4x3, ce qui pose certains problèmes, quand bien même juste le copier coller fonctionnerait car la disposition des touches et le fonctionnement reste le même, néanmoins cela peut permettre d'alléger légèrement le code.

28/01 :

Mise en place du code vu en vidéo, cependant le résultat n'est pas concluant pour des raisons que j'ignore, je vais effectuer des test lors de la prochaine séance.

Pour le moment, j'ai essayé avec le code exact de la vidéo en 4x4, et cela fonctionne très bien, il faut juste adapter le code en fonction de nos besoins.

03/02 :

Avec l'aide de Lysandre et d'Abel, le code pour le clavier a été réalisé, par conséquent, il faut assembler le tout désormais, et surtout le mettre en lien avec un raspberry pi, car pour le moment tout était sur l'Arduino.

J'ai également dû souder une led avec une résistance et deux cables d'un bout à l'autre pour Abel pour qu'il puisse voir s'il peut piloter la carte à distance.

04/02 :

Documentation sur le fonctionnement des pins de la raspberry via des vidéos, des documents en ligne, ainsi que le TP vu en classe, globalement aucun souci car les pins sont déjà fonctionnels, et il ne vaut pas risquer de bloquer définitivement ces derniers.

25/02 :

Ajout d'un level shifter pour le faire le pont entre l'arduino (5v) et le raspberry pi (3,3v), ainsi que travail en duo avec Lysandre afin de commencer à lier tous les composants afin de voir les potentiels problèmes et que l'on puisse avancer sans craindre une erreur de base qui pourrait créer d'autres problèmes par la suite.

03/03 :

- Réunion avec les STI2D ITEC pour avoir une idée des dimensions de la boîte pour le projet, ainsi que de la matière de celle-ci, le plexiglass a été choisi.
- Fin du schéma sur KICAD car pour le moment, on ne sait pas si l'on doit garder certaines parties, et il ne vaut mieux pas avoir des composants inutiles sur le projet.
- Solution peut-être trouvée, en remplaçant au passage l'arduino uno, par un Attiny 1614, qui peut exécuter le même code avec les mêmes résultats, seuls les ports changent, ainsi que la taille, ce qui est à notre avantage.

04 / 03 :

Test avec la serrure de codes, ainsi que programmation d'un ATtiny, avec un outil spécialisé pour cela, (Mplab) ainsi que révision de quelques soucis rencontrés.

10/03 :

Code définitif pour l'ATtiny

11/03 :

Continuation sur les schémas KICAD. Et choix d'un ATtiny 1616 à la place de l'ATtiny 1614 car il a plus de port.

23/03

Les derniers composants encore non-sûrs ont été validés, et ainsi le routage peut désormais commencer.

24/03 :

Essais de routage de la carte du clavier.

30/03 :

Correction des erreurs et progression en vue d'une étape finale.

01/04 :

Dernières modifications de la carte avec le clavier.

22/04

Reprise des projets avec soudure de la carte avec clavier car les composants ont été commandés pendant les vacances.

28/04 et 29/04 :

La carte du HAT de la Raspberry Pi Zero est routée et validé.

05/05 et 06/05

Création des fiches de mise en service pour les cartes ainsi qu'une fiche de dépannage.

13/05 :

Soudure des composants CMS du HAT.

19/05 :

Soudure des travers pour le HAT et assemblage de toute les parties, le projet est presque terminé, voir terminé sur le plan matériel.

Nomenclature

[illegible]

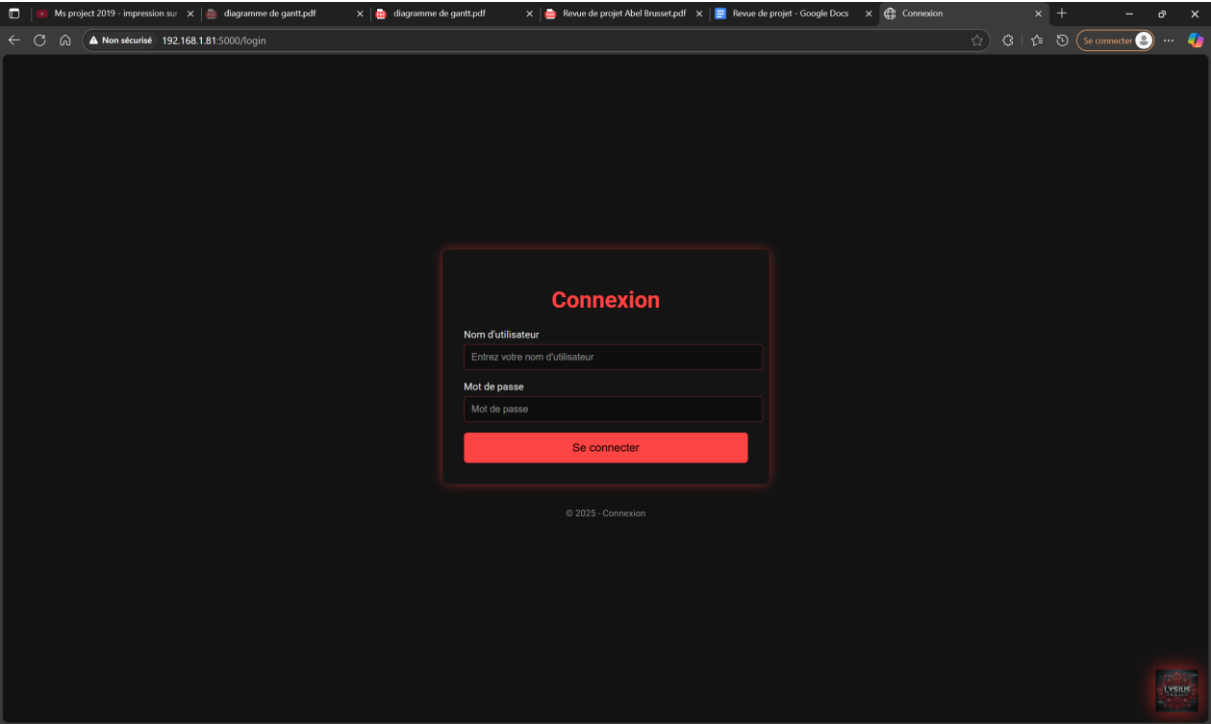
Nomenclature

Carte formPhotoCcp 1.2										
Requis	Description du matériel	Valeur (Référence)	Taux	Eggs aux	Fournisseur	Boîtier	Remarque	Qty	Catégorie	réf. article
R1	Relais CMS 1206 0.20W 4100µm/C	120 Dinars	41%		Radarguys	CMS 1206		1	Par	679-2049
R2	Relais CMS 1206 0.20W 4100µm/C	320 Dinars	41%		Radarguys	CMS 1206		1	Par 50	0.036
R3	Relais CMS 1206 0.20W 4100µm/C	2.2K Dinars	41%		Radarguys	CMS 1206		1	Par 50	0.036
R4, R5	Relais CMS 1206 0.20W 4100µm/C	10K Dinars	41%		Radarguys	CMS 1206		2	Par 120	0.027
C1, C3, C4, C7	Condensateur CMS ML CC 16V	1µF	-20 > +80%		Radarguys	CMS 0005		5	Par 25	0.04
C4	Condensateur tamble, CMS, 20V	27µF	±10%		Radarguys	CMS 734331		1	Par 20	0.054
C4	Condensateur CMS ML CC 50V	4.7µF	±10%		Radarguys	CMS 1206		1	Par 10	0.274
U1	TSR 22450				Panel			1	Par 1	12.130
B1	LED V4L, CMS, 2.2V				Radarguys	CMS 2012		1	Par 50	0.151
B2	LED Rouge, CMS, 2V				Radarguys	CMS 2012		1	Par 50	0.151
B3	Diode CMS Neopex, 1A, 30V				Radarguys	800-123	Shimdy	1	Par 5	0.194
B4	62CE327				Panel			1	Par 5	0.146
B5	LED Jaune, CMS, 2.1V				Radarguys	CMS 2012		1	Par 25	0.146
B6	Resistor Neopex 0.540m 12413pm, Resipm				Radarguys			1	Par 35	0.214
B7	Condensateur CMS ML CC 16V				Radarguys			1	Par 1	0.214
B8	Condensateur CMS ML CC 16V				Radarguys			1	Par 1	2.280
A3, A4, A5, A9	Compteur CMS Digi				Radarguys			1	Par 1	0.150
A4	Prise d'alimentation CC 1A 12V				Radarguys			1	Par 1	0.150
A7	Embase, AIRPHODU/HEU, 2.550m, 2.65mm, 60H				Radarguys			1	Par 5	0.600
F1	Fusible thermique 400mA, 0.20V, 4C				Radarguys			1	Par 10	0.300
F2	Fusible thermique 22A, 11A, 24V, 1812				Radarguys			1	Par 10	1.009
F3	Fusible thermique 4A, 2A, 8V, 1812				Radarguys			1	Par 10	0.348
TOTAL HT FUSIBLE									22.562	
TOTAL HT C.C.M.									22.562	
COEFFICIENT TVA									1.2	
Mars 2018 :									27.074	
Mars 2019 :										
Carte formPhotoCcp 1.2										
Requis	Description du matériel	Valeur (Référence)	Taux	Eggs aux	Fournisseur	Boîtier	Remarque	Qty	Catégorie	réf. article
O2	Transistor MOSFET canal N, 2A 20V, 3 broches				Radarguys	SOT-23		1	20	792-6675
										0.273
										0.273

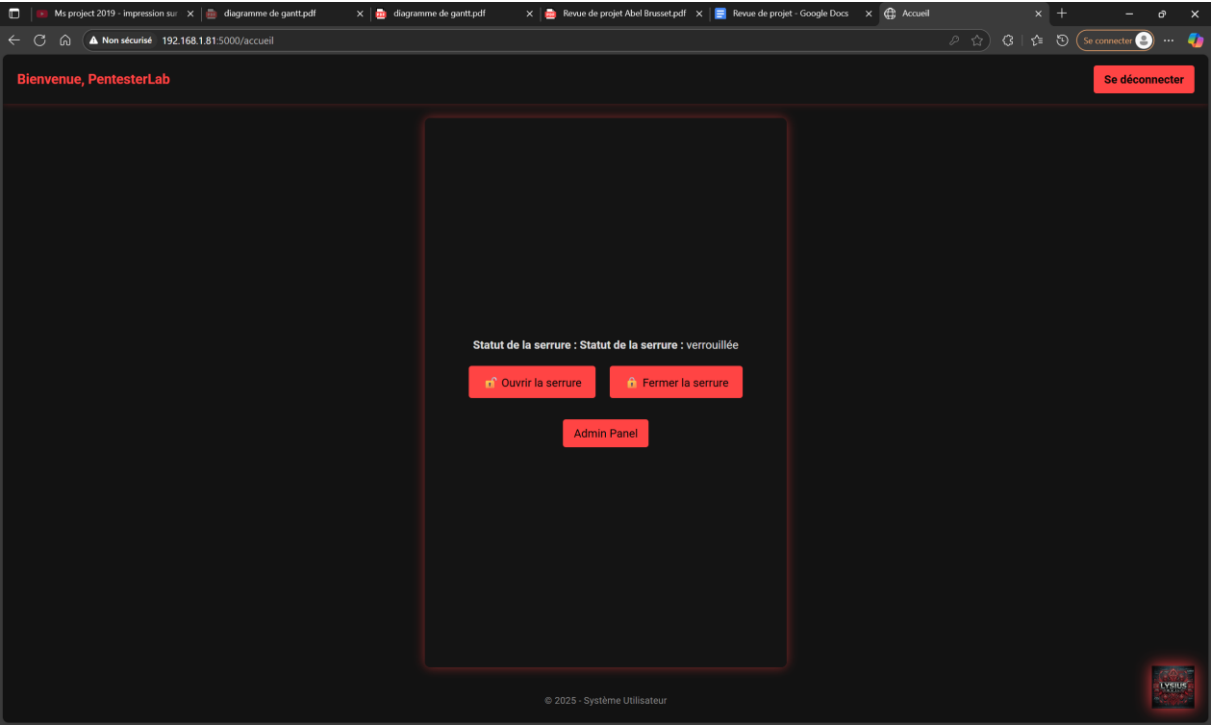
ANNEXES

25. L'interface Web (API) Python | ANNEXE [1].

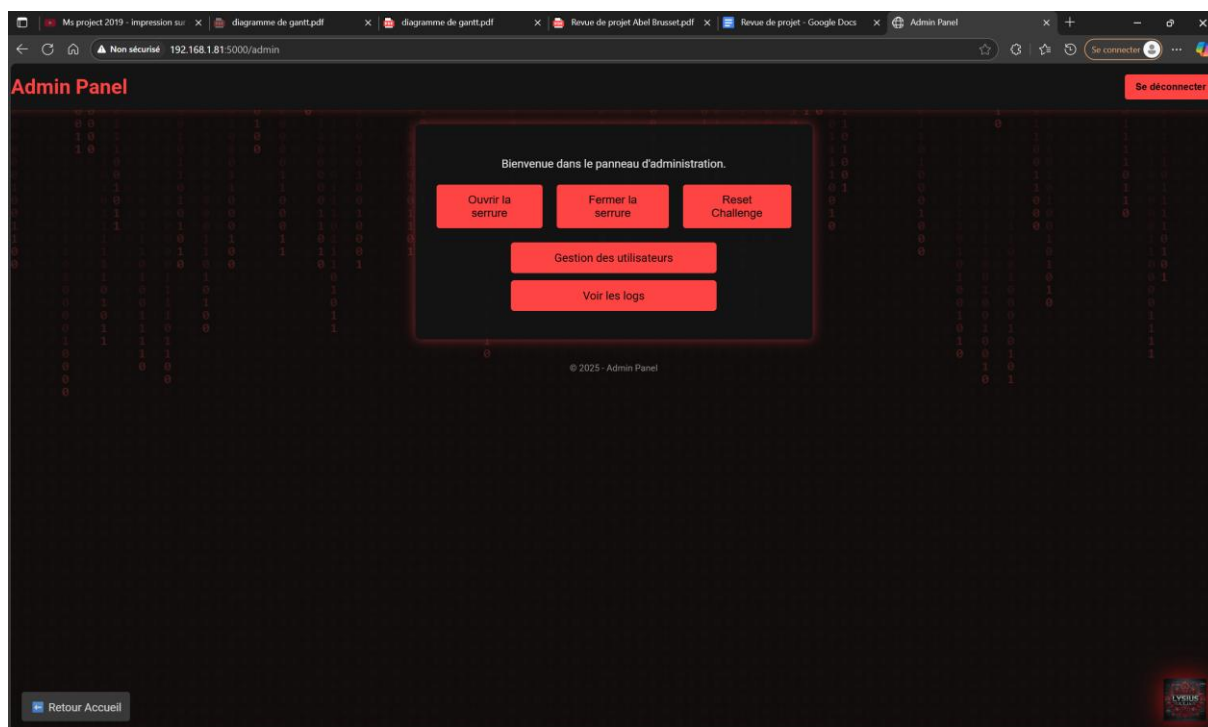
Page de login :



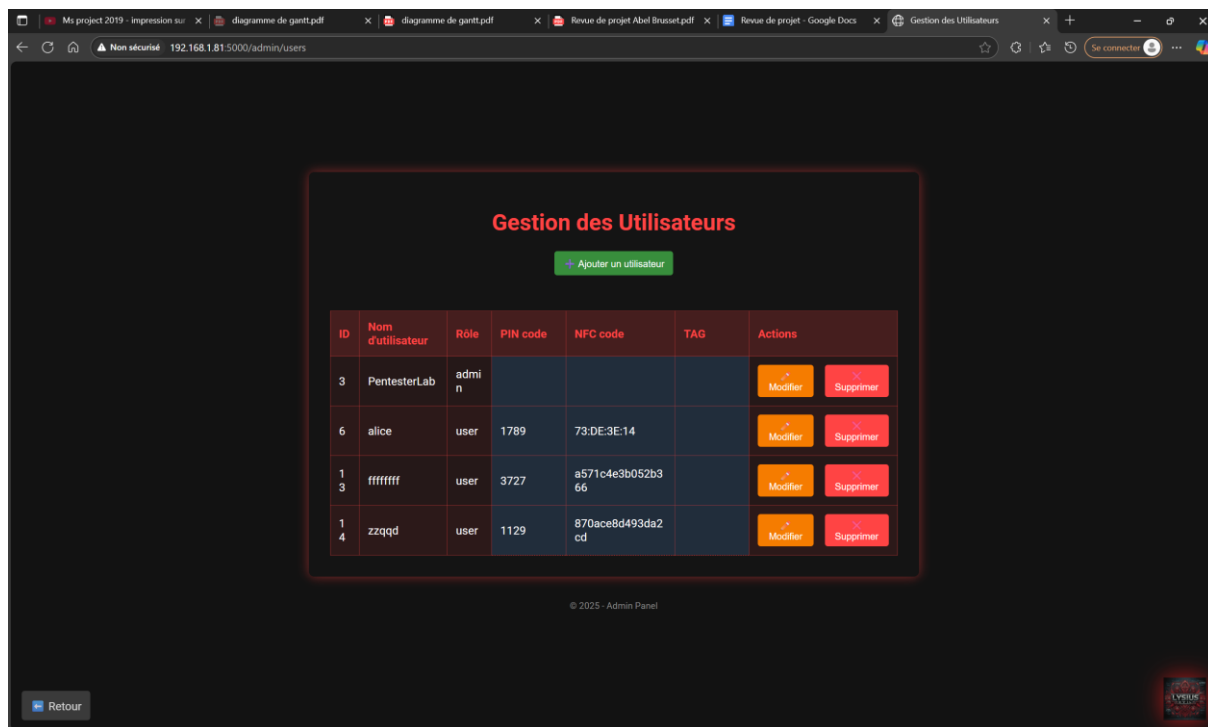
Page d'utilisateur :



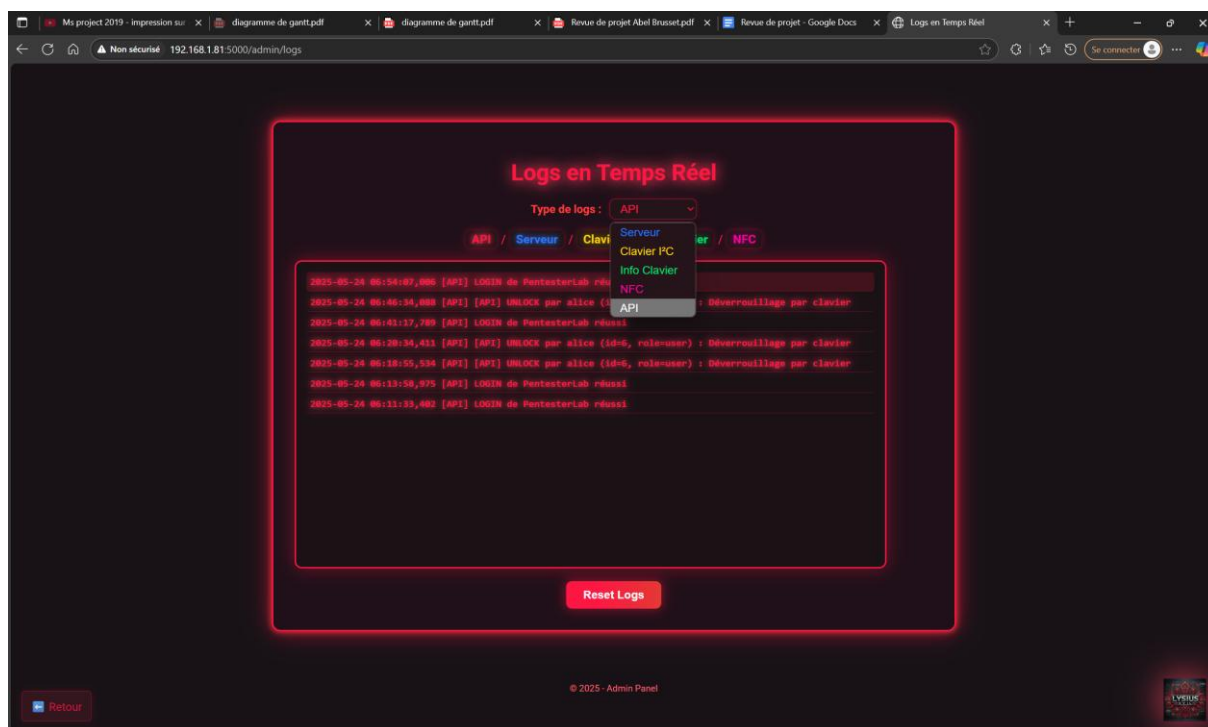
Page admin panel :



Page de la gestion des utilisateurs :



Page de la gestion des logs



26. Dictionnaire des Endpoint | ANNEXE [2].

URL	Méthode(s)	Description courte	Authentification
/	GET	Page d'accueil (index.html)	Non
/accueil	GET	Accueil utilisateur connecté	Oui (require_auth)
/login	GET, POST	Page de connexion / Authentification	Non
/logout	POST	Déconnexion (suppression cookie JWT)	Non
/verify_pin	POST	Déverrouillage par code PIN (clavier)	Non
/verify_nfc	POST	Déverrouillage par badge NFC	Non
/user/status	GET	Statut de l'utilisateur connecté	Oui (require_auth)
/status	GET	État actuel de la serrure	Oui (require_auth)
/unlock	POST	Déverrouille la serrure (API)	Oui (require_auth, user/admin)
/lock	POST	Verrouille la serrure (API)	Oui (require_auth, user/admin)
/admin	GET	Panneau d'administration	Oui (require_auth, admin)
/admin/users	GET	Gestion des utilisateurs	Oui (require_auth, admin)

/admin/logs/filter	GET	Filtre de logs (panel admin)	Oui (require_auth, admin)
/admin/logs	GET	Visualisation logs multi-type (panel admin)	Oui (require_auth, admin)
/admin/log_filter	POST	Récupération logs filtrés (panel admin)	Oui (require_auth, admin)
/admin/update_field	POST	MAJ direct PIN/NFC/tag user	Oui (require_auth, admin)
/admin/reset_challenge	POST	Reset des failles challenges niveau 2	Oui (require_auth, admin)
/api/log_keypad_raw	POST	Log raw clavier pour monitor (embarqué)	Non
/api/log_event	POST	Journalisation événement API (ex : tentative déverrouillage)	Oui (require_api_key)
/api/mqtt_challenge	POST	Simulation challenge MQTT	Non
/v2/unlock_challenge	POST	Déverrouille via exploitation XSS+Path Traversal (challenge)	Non si XSS/path exploité
/v2/xss	GET, POST	Formulaire vulnérable XSS (challenge)	Non
/v2/traversal	GET	Lecture fichier (faille Path Traversal)	Non

```

# Import des bibliothèques système et tierces nécessaires
import os                # Gestion des chemins, dossiers et variables
d'environnement
import logging           # Gestion des logs et fichiers de logs
import json              # Encodage/décodage JSON
import datetime          # Gestion des dates et heures
import threading         # Exécution simultanée (logs, MQTT)
import time              # Fonctions temporelles (sleep, pause)
import jwt               # Gestion des tokens JWT (authentification)
import paho.mqtt.client as mqtt  # Client MQTT pour messages IoT
import redis             # Client Redis (stockage temporaire)
import secrets           # Génération sécurisée de secrets (PIN, etc)
from functools import wraps  # Pour créer des décorateurs

# Import des modules Flask et extensions
from flask import (
    Flask, request, jsonify, render_template,
    render_template_string, make_response, redirect,
    url_for, session
)
from flask_sqlalchemy import SQLAlchemy  # ORM SQL pour les modèles BDD
from flask_migrate import Migrate      # Migration BDD
from flask_bcrypt import Bcrypt        # Hash sécurisé des mots de passe
from flask_socketio import SocketIO    # WebSocket pour logs temps réel
from dotenv import load_dotenv          # Variables d'environnement .env
from flask_limiter import Limiter       # Limitation du débit API (rate
limit)
from flask_limiter.util import get_remote_address
from flask_login import LoginManager, UserMixin, current_user  # Auth
utilisateur
from flask_talisman import Talisman    # Sécurité HTTP (headers CSP, ...)
from sqlalchemy import text            # Requêtes SQL brutes

# ----- Initialisation des dossiers de logs -----
-
load_dotenv()  # Charge les variables d'environnement depuis .env

BASE_DIR = os.path.dirname(os.path.abspath(__file__))  # Chemin de base du
projet
LOGS_DIR = os.path.join(os.path.dirname(os.path.abspath(__file__)), "logs")  #
Dossier logs

# Crée le dossier logs s'il n'existe pas
if not os.path.exists(LOGS_DIR):
    os.makedirs(LOGS_DIR)

```

```

# Dictionnaire des fichiers de logs utilisés dans l'appli
LOG_FILES = {
    "SERVEUR": "serveur.log",
    "Clavier-I²C": "clavier.log",
    "INFO-CLAVIER": "keypad_raw.log",
    "NFC": "nfc.log",
    "API": "api.log",
    "API-challenge": os.path.join(LOGS_DIR, "challenge.log")
}

os.makedirs(LOGS_DIR, exist_ok=True) # Crée (ou recrée) le dossier logs si
jamais absent

def create_logger(name, file, level=logging.INFO, fmt=None):
    # Crée un logger dédié à un fichier spécifique avec un format personnalisé
    logger = logging.getLogger(name)
    if not logger.handlers():
        fhandler = logging.FileHandler(file)
        formatter = logging.Formatter(fmt or '%(asctime)s [%(levelname)s]
%(message)s')
        fhandler.setFormatter(formatter)
        logger.addHandler(fhandler)
        logger.setLevel(level)
    return logger

# Instancie tous les loggers nécessaires avec leurs formats
serveur_logger = create_logger("serveur", os.path.join(LOGS_DIR,
"serveur.log"), fmt='% (asctime)s [SERVEUR] %(message)s')
clavier_logger = create_logger("clavier", os.path.join(LOGS_DIR,
"clavier.log"), fmt='% (asctime)s [Clavier-I²C] %(message)s')
info_clavier_logger = create_logger("info_clavier", os.path.join(LOGS_DIR,
"keypad_raw.log"), fmt='% (asctime)s [INFO-CLAVIER] %(message)s')
nfc_logger = create_logger("nfc", os.path.join(LOGS_DIR,
"nfc.log"), fmt='% (asctime)s [NFC] %(message)s')
api_logger = create_logger("api", os.path.join(LOGS_DIR,
"api.log"), fmt='% (asctime)s [API] %(message)s')
main_logger =
create_logger("main", os.path.join(BASE_DIR, "serrure.log"))

def ensure_logs_exist():
    # Vérifie et crée les fichiers de logs s'ils sont absents
    if not os.path.exists(LOGS_DIR):
        os.makedirs(LOGS_DIR, exist_ok=True)
    for fname in LOG_FILES.values():
        fpath = os.path.join(LOGS_DIR, fname)
        if not os.path.exists(fpath):
            open(fpath, "a", encoding="utf-8").close()

def ensure_log_file(log_path):

```

```

    # Créé le fichier de log si absent (chemin complet)
    os.makedirs(os.path.dirname(log_path), exist_ok=True)
    if not os.path.exists(log_path):
        with open(log_path, "a", encoding="utf-8") as f:
            f.write("")

def log_all(message, logger_list):
    # Logge un message dans tous les loggers fournis
    for lg in logger_list:
        lg.info(message)

# ----- Flask & Extensions -----
app = Flask(
    __name__,
    template_folder=os.path.join(BASE_DIR, "templates"),
    static_folder=os.path.join(BASE_DIR, "static")
)

# Active les protections HTTP en production
if os.getenv("FLASK_ENV") == "production":
    Talisman(app)

# Configuration des clés et de la BDD
SECRET_KEY = os.getenv("SECRET_KEY", "secretkey")
app.config['SECRET_KEY'] = SECRET_KEY

DB_HOST      = os.getenv("DB_HOST", "localhost")
DB_USER      = os.getenv("DB_USER", "Ad_@min")
DB_PASSWORD  = os.getenv("DB_PASSWORD", "Admin")
DB_NAME      = os.getenv("DB_NAME", "dvil_db")
app.config['SQLALCHEMY_DATABASE_URI'] =
f"mysql+pymysql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}/{DB_NAME}"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Initialisation des modules Flask extensions
db = SQLAlchemy(app)          # ORM SQLAlchemy
migrate = Migrate(app, db)    # Outil de migration de schéma
bcrypt = Bcrypt(app)          # Pour hash des mots de passe
socketio = SocketIO(app, cors_allowed_origins="*") # Websocket pour logs
temps réel

# ----- Limiter -----
try:
    redis_client = redis.StrictRedis(host=os.getenv("REDIS_HOST",
"localhost"), port=6379, db=0)
    redis_client.ping()
    limiter_storage = "redis://localhost:6379"
except redis.ConnectionError:
    serveur_logger.warning("⚠ Redis non accessible, fallback mémoire.")

```

```

    redis_client = None
    limiter_storage = "memory://"

# Limiteur d'accès API (rate limit)
limiter = Limiter(
    key_func=get_remote_address,
    app=app,
    storage_uri=limiter_storage,
    default_limits=["10000 per hour"]
)

# ----- Flask-Login -----
login_manager = LoginManager(app)
login_manager.login_view = "login_page"

@login_manager.user_loader
def load_user(user_id):
    # Charge un utilisateur depuis la BDD (par id) pour la session Flask-Login
    return User.query.get(int(user_id))

# ----- Modèles DB -----
class User(UserMixin, db.Model):
    # Table des utilisateurs, héritée de Flask-Login UserMixin pour la gestion
    # de session
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False,
index=True)
    password = db.Column(db.String(255), nullable=False)
    pin_code = db.Column(db.String(10), nullable=True)
    nfc_code = db.Column(db.String(32), nullable=True)
    tag = db.Column(db.String(64), nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.datetime.utcnow)
    role = db.Column(db.String(20), default="user")
    mfa_secret = db.Column(db.String(16), unique=True, nullable=True)

class SerrureState(db.Model):
    # Table qui enregistre l'état de la serrure (verrouillée/déverrouillée)
    __tablename__ = 'serrure_state'
    id = db.Column(db.Integer, primary_key=True)
    state = db.Column(db.String(10), nullable=False)

class Comment(db.Model):
    # Table des commentaires utilisateurs (non utilisée ici mais structure
    # prête)
    __tablename__ = 'comment'
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.Text, nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.datetime.utcnow)

```

```

with app.app_context():
    db.create_all() # Créé les tables si besoin (développement)

# ----- Authentification et décorateurs de protection -----
def verify_token_cookie():
    # Vérifie et décode le token JWT stocké dans le cookie utilisateur
    token = request.cookies.get("jwt_token")
    if not token:
        return None
    try:
        decoded = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
        return decoded
    except (jwt.ExpiredSignatureError, jwt.InvalidTokenError):
        return None

def require_auth(f):
    # Décorateur : oblige l'utilisateur à être authentifié (JWT valide)
    @wraps(f)
    def decorated(*args, **kwargs):
        payload = verify_token_cookie()
        if not payload:
            return redirect(url_for('login_page'))
        return f(*args, **kwargs)
    return decorated

def require_admin(f):
    # Décorateur : oblige à être admin (vérifie le rôle dans le token puis la BDD)
    @wraps(f)
    def decorated(*args, **kwargs):
        payload = verify_token_cookie()
        if not payload:
            return redirect(url_for('login_page'))
        user = User.query.filter_by(username=payload.get("username")).first()
        if not user or user.role != "admin":
            return jsonify({"error": "Accès refusé (admin seulement)."}), 403
        return f(*args, **kwargs)
    return decorated

def require_user_or_admin(f):
    # Décorateur : accès à tout utilisateur authentifié OU admin
    @wraps(f)
    def decorated(*args, **kwargs):
        payload = verify_token_cookie()
        if not payload:
            return redirect(url_for('login_page'))
        user = User.query.filter_by(username=payload.get("username")).first()
        if not user:

```

```

        return jsonify({"error": "Utilisateur introuvable"}), 404
    if user.role not in ["user", "admin"]:
        return jsonify({"error": "Accès refusé (user ou admin uniquement)"}), 403
    return f(*args, **kwargs)
    return decorated

API_KEY = os.getenv("API_KEY", "secret")
def require_api_key(f):
    # Décorateur : accès restreint via clé API dans l'en-tête HTTP (sécurité inter-app)
    @wraps(f)
    def decorated(*args, **kwargs):
        if request.headers.get("X-API-Key") != API_KEY:
            return jsonify({"error": "Clé API invalide"}), 401
        return f(*args, **kwargs)
    return decorated

# ----- Configuration MQTT -----
MQTT_BROKER = os.getenv("MQTT_BROKER", "localhost")
MQTT_PORT = int(os.getenv("MQTT_PORT", 1883))
MQTT_TOPIC_LOCK = "DVIL/Serrure" # Topic pour verrouillage
MQTT_TOPIC_UNLOCK = "DVIL/Serrure" # Topic pour déverrouillage
MQTT_TOPIC_STATUS = "DVIL/Status" # Topic statut de la serrure

mqtt_client = mqtt.Client() # Création du client MQTT
mqtt_client.username_pw_set("admin", "admin") # Authentification broker (ici valeurs par défaut)
lock_state = {"locked": True} # Dictionnaire qui maintient l'état actuel de la serrure
last_status_logged = None # Pour logger les changements de statut uniquement

def on_connect(client, userdata, flags, rc):
    # Callback lors de la connexion au broker MQTT
    if rc == 0:
        serveur_logger.info("[MQTT] Connecté au broker")
        client.subscribe(MQTT_TOPIC_UNLOCK)
    else:
        serveur_logger.error(f"[MQTT] Échec de connexion, code={rc}")

def on_message(client, userdata, msg):
    # Callback lors de la réception d'un message sur les topics MQTT suivis
    global lock_state
    if msg.topic == MQTT_TOPIC_UNLOCK:
        payload_str = msg.payload.decode()
        if payload_str == "deverrouillage":
            lock_state["locked"] = False
        elif payload_str == "fermeture":

```

```

        lock_state["locked"] = True
        serveur_logger.info(f"[MQTT] Serrure =>
locked={lock_state['locked']}")

# Association des callbacks à l'instance du client MQTT
mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message

def mqtt_loop():
    # Lance la boucle MQTT dans un thread séparé pour écouter en permanence
    mqtt_client.connect(MQTT_BROKER, MQTT_PORT, 60)
    mqtt_client.loop_forever()

# Démarre le thread MQTT (daemon = fond)
threading.Thread(target=mqtt_loop, daemon=True).start()

# ----- Socket.IO logs en temps réel -----
def tail_logs():
    # Stream les logs serrure.log en temps réel via WebSocket pour affichage
    dynamique
    try:
        with open(os.path.join(BASE_DIR, "serrure.log"), "r") as f:
            f.seek(0, 2) # Va à la fin du fichier
            while True:
                line = f.readline()
                if line:
                    socketio.emit("log_update", {"log": line.strip()})
                else:
                    time.sleep(0.5)
    except Exception as e:
        print(f"[TAIL_LOGS] ERREUR: {e}")

# Lance le thread pour le streaming de logs en temps réel (daemon)
threading.Thread(target=tail_logs, daemon=True).start()

@app.after_request
def security_headers(response):
    # Ajoute des headers HTTP de sécurité à chaque réponse (anti XSS et
    sniffing)
    response.headers["X-Frame-Options"] = "DENY"
    response.headers["X-Content-Type-Options"] = "nosniff"
    return response

# ===== ROUTES PRINCIPALES (Accueil, login, logout, etc.) =====

@app.route("/")
def home():
    # Affiche la page d'accueil (index.html)
    return render_template("index.html")

```

```

@app.route("/accueil")
@require_auth
def accueil():
    # Page d'accueil utilisateur après connexion (protégée)
    payload = verify_token_cookie()
    return render_template("accueil.html", username=payload["username"])

@app.route("/login", methods=["GET"])
def login_page():
    # Affiche la page de connexion
    return render_template("login.html")

@app.route("/login", methods=["POST"])
@limiter.limit("5 per minute") # Limite les tentatives à 5 par minute
(protection brute force)
def login():
    # Authentifie l'utilisateur (JSON ou formulaire)
    auth = request.json if request.is_json else request.form
    username = auth.get("username")
    password = auth.get("password")
    user = User.query.filter_by(username=username).first()
    if user and bcrypt.check_password_hash(user.password, password):
        api_logger.info(f"LOGIN de {username} réussi")
        exp = datetime.datetime.utcnow() + datetime.timedelta(hours=2)
        token = jwt.encode({"username": username, "exp": int(exp.timestamp())
+ 7200}, SECRET_KEY, algorithm="HS256")
        response = jsonify({"redirect": "/accueil"})
        response.set_cookie("jwt_token", token, httponly=True, secure=False,
samesite="Lax")
        return response
    else:
        api_logger.warning(f"Tentative login échouée pour {username}")
        return jsonify({"error": "Identifiants incorrects."}), 401

@app.route("/logout", methods=["POST"])
def logout():
    # Déconnecte l'utilisateur (supprime le cookie JWT)
    resp = make_response(redirect(url_for("login_page")))
    resp.set_cookie("jwt_token", "", expires=0)
    return resp

@app.route("/verify_pin", methods=["POST"])
def verify_pin():
    # Vérifie le code PIN du clavier pour déverrouiller la serrure
    data = request.get_json(force=True)
    pin = data.get("pin_code", "")
    user = User.query.filter_by(pin_code=pin).first()

```

```

    info_clavier_logger.info(f"[KEYPAD_RAW] {pin}")

    if user:
        # Si le code PIN est correct, publie l'événement MQTT pour
        déverrouiller
        mqtt_client.publish("DVIL/Serrure", "deverouillage")
        mqtt_client.publish("DVIL/Clavier", "deverouillage")
        clavier_logger.info(f"[OUVERTURE] par {user.username} (id={user.id},
pin_code={pin})")
        api_logger.info(f"[API] UNLOCK par {user.username} (id={user.id},
role={user.role}) : Déverrouillage par clavier")
        return jsonify({
            "authorized": True,
            "username": user.username,
            "user_id": user.id,
            "role": user.role
        })
    else:
        info_clavier_logger.warning(f"[KEYPAD_RAW] PIN incorrect : {pin}")
        api_logger.warning(f"[API] Echec ouverture clavier avec code: {pin}")
        return jsonify({"authorized": False})

@app.route("/verify_nfc", methods=["POST"])
def verify_nfc():
    # Vérifie le badge NFC pour déverrouiller la serrure
    data = request.get_json(force=True)
    nfc_uid = data.get("nfc_code", "").strip()
    user = User.query.filter_by(nfc_code=nfc_uid).first()
    nfc_logger.info(f"{nfc_uid}")
    if user:
        nfc_logger.info(f"Déverrouillage badge par {user.username}
(id={user.id}, nfc_code={nfc_uid})")
        mqtt_client.publish("DVIL/Serrure", "deverouillage")
        return jsonify({
            "authorized": True,
            "username": user.username,
            "user_id": user.id,
            "role": user.role
        })
    else:
        nfc_logger.warning(f"Echec badge inconnu : {nfc_uid}")
        return jsonify({"authorized": False})

@app.route("/user/status", methods=["GET"])
@require_auth
def user_status():
    # Renvoie le statut de l'utilisateur connecté (nom et rôle)
    payload = verify_token_cookie()
    user = User.query.filter_by(username=payload["username"]).first()

```

```

    if not user:
        return jsonify({"error": "Utilisateur introuvable"}), 404
    return jsonify({"username": user.username, "role": user.role}), 200

@app.route("/status", methods=["GET"])
@require_auth
def get_status():
    # Renvoie l'état actuel de la serrure (verrouillée/déverrouillée)
    global last_status_logged
    status = "verrouillée" if lock_state["locked"] else "déverrouillée"

    if last_status_logged != status:
        serveur_logger.info(f"Status consulté ({status})")
        last_status_logged = status

    return jsonify({"status": status})

@app.route("/unlock", methods=["POST"])
@require_auth
@require_user_or_admin
def unlock():
    # Déverrouille la serrure via API, MQTT et met à jour l'état global
    try:
        mqtt_client.publish(MQTT_TOPIC_UNLOCK, "deverouillage")
        lock_state["locked"] = False
        serveur_logger.info("Déverrouillage par API")
        return jsonify({"status": "success", "message": "Serrure
déverrouillée."})
    except Exception as e:
        serveur_logger.error(f"Erreur lors de la publication MQTT (unlock):
{e}")
        return jsonify({"status": "fail", "message": "Erreur lors de l'envoi
de la commande MQTT."}), 500

@app.route("/lock", methods=["POST"])
@require_auth
@require_user_or_admin
def lock():
    # Verrouille la serrure via API, MQTT et met à jour l'état global
    try:
        mqtt_client.publish(MQTT_TOPIC_LOCK, "fermeture")
        lock_state["locked"] = True
        serveur_logger.info("Verrouillage par API")
        return jsonify({"status": "success", "message": "Serrure
verrouillée."})
    except Exception as e:
        serveur_logger.error(f"Erreur lors de la publication MQTT (lock):
{e}")

```

```

        return jsonify({"status": "fail", "message": "Erreur lors de l'envoi
de la commande MQTT."}), 500

# --- ADMIN PANEL et LOGS ---

@app.route("/admin")
@require_auth
@require_admin
def admin_panel():
    # Affiche le panneau d'administration principal
    return render_template("admin.html")

@app.route("/admin/users")
@require_auth
@require_admin
def admin_users():
    # Affiche la gestion des utilisateurs pour l'admin
    users = User.query.all()
    return render_template("admin_users.html", users=users)

@app.route("/admin/log_filter", methods=["POST"])
@require_auth
@require_admin
def admin_log_filter():
    """
    Endpoint pour filtrer et retourner les logs demandés.
    Attendu : JSON { "log_type": "api" } (ou "serveur", "clavier", etc.)
    """
    data = request.get_json()
    log_type = data.get("log_type", "").lower()

    # Mapping des types de logs vers leur chemin
    log_paths = {
        "api": "logs/api.log",
        "serveur": "logs/serveur.log",
        "clavier": "logs/clavier.log",
        "nfc": "logs/nfc.log",
        "keypad": "logs/keypad_raw.log",
        "challenge": "logs/challenge.log",
    }
    # Par défaut : log API si rien ou inconnu
    log_file = log_paths.get(log_type, "logs/api.log")

    # Lecture du log
    try:
        if not os.path.exists(log_file):
            raise FileNotFoundError(f"No such file or directory:
'{log_file}'")
        with open(log_file, "r", encoding="utf-8") as f:

```

```

        lines = f.readlines()
        return jsonify({"lines": lines, "error": None})
    except FileNotFoundError as e:
        return jsonify({"lines": [], "error": f"Erreur lecture log : {str(e)}"})
    except Exception as e:
        return jsonify({"lines": [], "error": f"Erreur inattendue lors de la lecture du log : {str(e)}"})

@app.route("/admin/logs")
@require_auth
@require_admin
def admin_uloogs():
    # Affiche les logs principaux dans le panneau d'admin
    import os

    LOGS_DIR = os.path.join(os.path.dirname(__file__), "logs")
    os.makedirs(LOGS_DIR, exist_ok=True)

    log_files = {
        "SERVEUR": os.path.join(LOGS_DIR, "serveur.log"),
        "Clavier-I²C": os.path.join(LOGS_DIR, "clavier.log"),
        "INFO-CLAVIER": os.path.join(LOGS_DIR, "keypad_raw.log"),
        "NFC": os.path.join(LOGS_DIR, "nfc.log"),
        "API": os.path.join(LOGS_DIR, "api.log"),
        "API-challenge": os.path.join(LOGS_DIR, "challenge.log"),
    }
    logs_by_type = {}
    for logtype, path in log_files.items():
        if not os.path.exists(path):
            with open(path, "w", encoding="utf-8") as f:
                pass
        try:
            with open(path, "r", encoding="utf-8") as f:
                lines = f.readlines()[-300:]
                logs_by_type[logtype] = lines
        except Exception as e:
            logs_by_type[logtype] = [f"[ERROR] Impossible de lire ce log : {e}\n"]
    return render_template("admin_logs.html", logs_by_type=logs_by_type)

@app.route("/admin/log_raw")
@require_auth
@require_admin
def admin_log_raw():
    # Renvoie les logs bruts sous forme de liste pour affichage dynamique
    import os
    LOGS_DIR = os.path.join(os.path.dirname(__file__), "logs")
    log_files = [

```

```

        ("SERVEUR", os.path.join(LOGS_DIR, "serveur.log")),
        ("Clavier-I²C", os.path.join(LOGS_DIR, "clavier.log")),
        ("INFO-CLAVIER", os.path.join(LOGS_DIR, "keypad_raw.log")),
        ("NFC", os.path.join(LOGS_DIR, "nfc.log")),
        ("API", os.path.join(LOGS_DIR, "api.log")),
    ]
    logs = []
    for cat, path in log_files:
        if not os.path.exists(path):
            with open(path, "w", encoding="utf-8") as f:
                pass
        try:
            with open(path, "r", encoding="utf-8") as f:
                lines = f.readlines()[-200:]
                logs += [[cat, l.strip()] for l in lines]
        except Exception as e:
            logs.append([cat, f"[ERROR] Erreur lecture log: {e}"])
    return {"logs": logs}

@app.route("/admin/reset_logs", methods=["POST"])
@require_auth
@require_admin
def admin_reset_logs():
    # Vide tous les fichiers de logs depuis l'admin
    import os
    LOGS_DIR = os.path.join(os.path.dirname(__file__), "logs")
    log_files = [
        "serveur.log",
        "clavier.log",
        "keypad_raw.log",
        "nfc.log",
        "api.log",
        "challenge.log",
    ]
    errors = []
    for fname in log_files:
        path = os.path.join(LOGS_DIR, fname)
        try:
            with open(path, "w", encoding="utf-8"):
                pass # Vide le fichier
        except Exception as e:
            errors.append(f"{fname}: {e}")
    if errors:
        return {"success": False, "errors": errors}, 500
    return {"success": True, "message": "Tous les logs ont été réinitialisés."}

@app.route("/admin/users/add", methods=["POST"])
@require_auth

```

```

@require_admin
def add_user():
    # Ajoute un nouvel utilisateur (génère pin et nfc par défaut)
    data = request.json
    username = data.get('username')
    password = data.get('password')
    role = data.get('role', 'user')
    if not username or not password:
        return jsonify({"error": "username et password requis"}), 400
    if User.query.filter_by(username=username).first():
        return jsonify({"error": "Utilisateur déjà existant"}), 409
    hashed = bcrypt.generate_password_hash(password).decode('utf-8')
    default_pin = "{:04d}".format(secrets.randbelow(10000))
    default_nfc = secrets.token_hex(8)
    new_user = User(
        username=username,
        password=hashed,
        role=role,
        pin_code=default_pin,
        nfc_code=default_nfc
    )
    db.session.add(new_user)
    db.session.commit()
    serveur_logger.info(f"[ADMIN] Ajout user={username} role={role}
pin={default_pin} nfc={default_nfc}")
    return jsonify({
        "message": "OK",
        "pin_code": default_pin,
        "nfc_code": default_nfc
    }), 201

@app.route("/admin/users/update", methods=["PUT"])
@require_auth
@require_admin
def update_user():
    # Met à jour le rôle ou le mot de passe d'un utilisateur existant
    data = request.json
    user_id = data.get('user_id')
    new_role = data.get('role')
    new_password = data.get('password')
    user = User.query.get(user_id)
    if not user:
        return jsonify({"error": "Inexistent"}), 404
    if new_role:
        user.role = new_role
    if new_password:
        user.password =
bcrypt.generate_password_hash(new_password).decode("utf-8")
    db.session.commit()

```

```

    serveur_logger.info(f"[ADMIN] Update user_id={user_id}")
    return jsonify({"message": "Mise à jour OK"}), 200

@app.route("/admin/users/delete", methods=["DELETE"])
@require_auth
@require_admin
def delete_user():
    # Supprime un utilisateur de la base
    data = request.json
    user_id = data.get('user_id')
    user = User.query.get(user_id)
    if not user:
        return jsonify({"error": "Inexistant"}), 404
    db.session.delete(user)
    db.session.commit()
    serveur_logger.info(f"[ADMIN] Delete user_id={user_id}")
    return jsonify({"message": "Utilisateur supprimé"}), 200

@app.route("/admin/update_field", methods=["POST"])
def admin_update_field():
    # Permet de modifier en direct un champ utilisateur (PIN, NFC ou tag)
    if not request.is_json:
        return jsonify({"status": "error", "error": "Requête invalide"}), 400
    data = request.get_json()
    user_id = data.get("user_id")
    field = data.get("field")
    value = data.get("value")
    if not user_id or field not in ("pin_code", "nfc_code", "tag"):
        return jsonify({"status": "error", "error": "Champs non valides"}),
400
    user = User.query.get(user_id)
    if not user:
        return jsonify({"status": "error", "error": "Utilisateur
introuvable"}), 404
    setattr(user, field, value)
    try:
        db.session.commit()
        serveur_logger.info(f"[ADMIN] Modification {field} pour
user_id={user_id} : {value}")
        return jsonify({"status": "ok"})
    except Exception as e:
        db.session.rollback()
        serveur_logger.error(f"[ADMIN] Erreur modification {field} pour
user_id={user_id} : {e}")
        return jsonify({"status": "error", "error": str(e)}), 500

# --- LOGS RAW CLAVIER ET NFC (pour apps embarquées) ---

@app.route("/api/log_keypad_raw", methods=["POST"])

```

```

def log_keypad_raw():
    # Ajoute une entrée brute dans le log clavier (pour monitoring)
    data = request.get_json(force=True)
    code = data.get("raw", "")
    info_clavier_logger.info(f"{code}")
    return jsonify({"status": "ok"})

@app.route("/api/log_event", methods=["POST"])
@require_api_key
def log_event():
    # Journalise un évènement spécifique côté API avec tous les détails
    data = request.get_json(force=True)
    event = data.get("event")
    pin_code = data.get("pin_code")
    details = data.get("details", "")
    username = "anonyme"
    user_id = None
    user_role = None
    status = "fail"
    if event == "KEYPAD_UNLOCK" and pin_code and len(pin_code) == 4 and
pin_code.isdigit():
        user = User.query.filter_by(pin_code=pin_code).first()
        if user:
            username = user.username
            user_id = user.id
            user_role = user.role
            details = f"Déverrouillage via clavier (PIN: {pin_code})"
            status = "success"
        else:
            details = f"Échec déverrouillage clavier : PIN inconnu
({pin_code})"
        log_all(
            f"{event} par {username} (id={user_id}, role={user_role}) :
{details}",
            [api_logger, clavier_logger, main_logger]
        )
    return jsonify({
        "status": status,
        "user": username,
        "user_id": user_id,
        "role": user_role,
        "details": details
    }), 200 if status == "success" else 401

# -----
# Lancement de l'application (uniquement en mode debug ; avec Gunicorn/WSGI
pour prod)

```

```
# -----  
-----  
# if __name__ == "__main__":  
#     socketio.run(app, host="0.0.0.0", port=5000, debug=True)
```

27. Base de données.

Table	Champ	Type	Clé	Null	Valeur par défaut	Usage/Remarque
alembic_version	version_num	varchar(32)	-	Non	-	Version migration de la base (SQLAlchemy)
comment	id	int(11)	PK	Non	auto_increment	Identifiant unique du commentaire
	content	text	-	Non	-	Texte du commentaire
	created_at	datetime	-	Oui	current_timestamp()	Date/heure de création
serrure_state	id	int(11)	PK	Non	auto_increment	Identifiant unique
	state	varchar(10)	-	Non	-	État actuel de la serrure (locked/unlocked)
users	id	int(11)	PK	Non	auto_increment	Identifiant utilisateur
	username	varchar(50)	UNIQUE	Non	-	Nom d'utilisateur
	password	varchar(255)	-	Non	-	Mot de passe haché (Bcrypt)
	pin_code	varchar(10)	-	Oui	NULL	Code PIN pour accès physique
	nfc_code	varchar(32)	-	Oui	NULL	UID du badge NFC
	created_at	datetime	-	Oui	current_timestamp()	Date/heure de création
	role	varchar(20)	-	Oui	'user'	Rôle : user/admin
	mfa_secret	varchar(16)	UNIQUE	Oui	NULL	Secret MFA (authentification double facteur)
	tag	varchar(64)	-	Oui	NULL	Tag libre utilisateur

28. Documentation Utilisateur/ Administrateur | ANNEXE [3].

DVIL SECURE — Documentation Administrateur

• **1. Introduction & Contexte**

But : Présenter le système, ses usages et la notion de “challenges” par niveaux.

Le projet DVIL Secure est une plateforme pédagogique dédiée à l'expérimentation de la sécurité IoT et web via des API RESTful. Elle permet à la fois un usage réel (contrôle d'une serrure connectée sécurisée) et la mise en évidence de failles courantes (challenges par niveaux), dans le but de former les administrateurs à la détection, l'exploitation et la remédiation de vulnérabilités typiques.

• **2. Authentification et JWT**

But : Décrire les mécanismes de connexion, rôles, gestion du JWT.

Exemple de rédaction :

L'API repose sur une authentification forte basée sur JWT (JSON Web Token).

- Connexion : L'utilisateur soumet ses identifiants via /login. Un token JWT est généré, signé et stocké en cookie HTTPOnly.
- Rôles :
 - User : Peut consulter l'état de la serrure et la contrôler.
 - Admin : Accès complet (gestion utilisateurs, logs, panel admin).
- Déconnexion : Le token JWT est supprimé via /logout.

3. Routes Principales

- Authentification
 - GET /login : Affiche la page de connexion.
 - POST /login : Authentifie et délivre le JWT.
 - POST /logout : Déconnecte l'utilisateur.
 - GET /user/status : Retourne les infos de l'utilisateur connecté.
- Contrôle de la Serrure
 - GET /status : Récupère l'état actuel de la serrure.
 - POST /unlock : Déverrouille la serrure via MQTT.
 - POST /lock : Verrouille la serrure via MQTT.
- c) Interface Administrateur
 - GET /admin : Accès au panneau d'administration.
 - GET /admin/users : Liste les utilisateurs.
 - GET /admin/logs : Affichage des logs temps réel.
 - POST /admin/users/add : Ajoute un utilisateur.
 - PUT /admin/users/update : Modifie un utilisateur.
 - DELETE /admin/users/delete : Supprime un utilisateur.
 - POST /admin/reset_challenge : Réinitialise la serrure.

4. Base de Données

But : Montrer la structure, usage, sécurité.

Exemple de rédaction :

La base MySQL/MariaDB gère :

- Les utilisateurs (authentification, rôle, PIN, badge NFC).
- L'état de la serrure (locked/unlocked).

Les migrations sont automatisées (Flask-Migrate) pour garantir la cohérence et la maintenabilité du schéma.

(Insère ici le tableau complet récapitulatif des tables/colonnes fourni plus haut.)

• **5. Communication via MQTT**

But : Expliquer l'interfaçage temps réel avec le hardware.

Exemple :

L'API communique avec le matériel (serrure connectée) via le protocole MQTT :

- Topic : DVIL/Serrure
- Utilisation : Publier les ordres de verrouillage/déverrouillage et recevoir le statut matériel.

• **6. Logs et Surveillance**

But : Montrer le monitoring, la traçabilité.

Exemple :

Toutes les actions critiques sont loguées (serrure.log, etc).

Un thread Python diffuse ces logs en temps réel via Socket.IO, permettant à l'admin de surveiller l'état du système et de détecter tout comportement anormal.

• **7. Utilisation & Maintenance**

But : Donner les consignes opérationnelles, bonnes pratiques de gestion.

Exemple :

- Utiliser l'interface admin pour gérer les utilisateurs et les logs.
- Effectuer des sauvegardes régulières de la base de données.
- Surveiller le système via le panel pour réagir rapidement en cas d'incident.

• **8. Mesures de Sécurité & Bonnes Pratiques**

But : Lister ce qui protège le système, et donner les réflexes à adopter.

Exemple :

- Authentification JWT & cookies HTTPOnly.
- Validation stricte de toutes les entrées utilisateur.
- Utilisation de requêtes paramétrées (SQLAlchemy).
- Sécurisation HTTP (Flask-Talisman), limitation du débit (Flask-Limiter).

- Audits réguliers & formation continue des équipes.

Section dédiée Challenges (Niveau 1, 2, 3)

- **Challenge Niveau 1 — Route non protégée**
 - Contexte : Présence d'un endpoint critique accessible sans authentification.
 - Vulnérabilité : "Broken Authentication" — absence de contrôle d'accès sur /v1/unlock_public.
 - Exploitation : Toute personne peut envoyer une requête POST et déverrouiller la serrure.
 - Impact : Prise de contrôle totale, faille critique.
 - Contremesures : Mettre un contrôle d'accès systématique (JWT, clé API...).
 - Bonnes pratiques : Audit de toutes les routes, protection par défaut, monitoring.
- **Challenge Niveau 2 — XSS & Path Traversal**
 - Contexte : Endpoints vulnérables à l'injection XSS (ex : /v2/xss) et à l'accès fichiers (/v2/traversal).
 - Exploitation XSS : Injection de code JS dans les commentaires, exécution dans le navigateur d'un admin.
 - Exploitation Path Traversal : Lecture de fichiers arbitraires via manipulation du paramètre file.
 - Impact : Vol de session, accès à des fichiers confidentiels, escalade d'attaque.
 - Contremesures : Validation/échappement systématique, CSP, restriction des accès fichiers.
- **Challenge Niveau 3 — SSTI, Pickle, SQLi**
 - Contexte : Routes exposant plusieurs failles enchaînables.
 - Failles :
 - SSTI (ex. /vuln/template): Injection de code Jinja2, extraction de la clé secrète.
 - Désérialisation Pickle : Modification de variables globales (ex : token d'admin).
 - SQL Injection : Altération ou extraction directe de données sensibles (/vuln/serrure).
 - Impact : Exécution de code arbitraire, prise de contrôle serveur, fuite de données.
 - Contremesures : Utiliser JSON au lieu de Pickle, requêtes paramétrées, validation rigoureuse.

Conclusion

Cette documentation doit te permettre d'exploiter, de maintenir et de sécuriser le système DVIL Secure en toute autonomie. Les exemples de challenges illustrent

l'importance de l'audit régulier, de la défense en profondeur et de la vigilance dans la gestion des accès et des entrées utilisateur.

DVIL SECURE — Documentation Utilisateur

Introduction Générale

Bienvenue sur la plateforme DVIL Secure !

Ici, tu vas te mettre dans la peau d'un **pentester** pour apprendre à repérer, exploiter et documenter des failles classiques dans une API web connectée à une serrure intelligente.

Chaque "Niveau" correspond à une ou plusieurs vulnérabilités : à toi de les découvrir, de comprendre le fonctionnement de l'API, et d'atteindre l'objectif final (déverrouillage).

Approche Générale d'un Challenge

1. Reconnaissance

- I. Explore l'API (Postman, curl, Burp, navigateur...)
- II. Identifie les endpoints accessibles sans login

2. Test des comportements

- I. Essaie plusieurs méthodes HTTP (GET, POST, etc.)
- II. Observe les réponses (codes HTTP, messages, JSON...)
- III. Cherche des comportements "anormaux" ou des endpoints sensibles non protégés

3. Exploitation

- Tente l'exploitation de la faille repérée
- Valide l'effet obtenu (exemple : serrure déverrouillée)

Conseil : Garde trace de tes tests et captures pour le rapport !

Niveau 1 — Route Non Protégée

• a) Introduction

Dans ce premier challenge, ton but est d'exploiter une **route d'API critique** laissée sans authentification, permettant de déverrouiller la serrure **sans identifiants**.

• b) Approche

1. Recherche les endpoints (ex : /v1/unlock_public)
2. Tente une requête POST sur cette route (curl ou Postman)
3. Observe la réponse de l'API et le changement d'état de la serrure

• c) Vulnérabilité

- **Type** : Absence totale de contrôle d'accès (Broken Authentication)
- **Objectif** : Déclencher une action critique sans être authentifié

• d) Exemple

```
• curl -X POST http://<ip>:<port>/v1/unlock_public
```

La serrure doit s'ouvrir même sans compte : c'est la faille !

Niveau 2 — XSS & Path Traversal

• a) Introduction

Ce challenge contient **deux failles à exploiter dans l'ordre** pour déverrouiller la serrure :

- **XSS (Cross-Site Scripting)**
- **Path Traversal**

• b) Approche

1. XSS

- Cherche un endpoint où tu peux injecter du texte (ex : /v2/xss)
- Tente d'injecter un script (<script>alert('XSS');</script>)
- Vérifie l'exécution du code dans la page (alerte affichée)

2. Path Traversal

- Identifie une route vulnérable (ex : /v2/traversal?file=...)
- Essaie de lire un fichier (test.txt) dans le dossier /uploads/ ou de sortir du dossier via ../
 - o L'API doit te retourner le contenu du fichier

- **c) Enchaînement et Objectif**

- Pour que /unlock fonctionne, il faut réussir XSS **et** Path Traversal (sessions "done")
- Appelle /unlock en POST pour publier le message MQTT de déverrouillage

- **d) Exemple de script d'exploitation**

```
• import requests
• BASE_URL = "http://<ip>:<port>"
• # 1) XSS
• r1 = requests.post(f"{BASE_URL}/v2/xss",
• data={"comment":"<script>alert('XSS');</script>"})
• # 2) Path Traversal
• r2 = requests.get(f"{BASE_URL}/v2/traversal",
• params={"file":"test.txt"})
• # 3) Unlock
• r3 = requests.post(f"{BASE_URL}/unlock")
```

Niveau 3 — SSTI, Pickle, SQL Injection

- **a) Introduction**

Dans ce niveau, il faut **enchaîner trois failles** pour déverrouiller la serrure :

1. **SSTI** (Server-Side Template Injection)
2. **Désérialisation Pickle non sécurisée**
3. **Injection SQL**

- **b) Approche par étape**

1. **Reconnaissance** : Liste tous les endpoints vulnérables (/vuln/template, /vuln/deserialize, /vuln/serrure)
2. **SSTI (Jinja2)** :
 - I. Envoie une chaîne malicieuse via le paramètre q sur /vuln/template
 - II. Objectif : Récupérer le SECRET_KEY ou toute autre variable sensible
 - III. Ex : {{ config.__class__.__init__.__globals__['os'].popen('cat .env').read() }}

3. Désérialisation Pickle

- I. Crée un objet Python malicieux (Pickle) pour définir une variable globale (OVERRIDE_TOKEN)
- II. Envoie-le sur /vuln/deserialize

4. Injection SQL

- I. Utilise le token récupéré pour exploiter /vuln/serrure?token=...&update=...
- II. Injecte une valeur pour forcer state = 'unlock' (ex : unlock' --)

• c) Objectif final

- Le but est de faire passer la serrure à l'état "unlock" via ces trois failles, prouvant ainsi l'exploitation en chaîne

• 6. Conseils & Bonnes Pratiques

- ✓ **Garde une trace de tes tests** : captures, scripts, retours d'API
- ✓ **Reste dans le cadre pédagogique** : n'attaque pas l'infra ou les comptes hors scope
- ✓ **Analyse les logs** pour vérifier l'état de la serrure et l'impact de tes actions
- ✓ **Prends le temps d'analyser** chaque étape avant d'automatiser tout le processus

• 7. Objectif & Remarques

Ton objectif : **exploiter chaque niveau en respectant la démarche offensive et en documentant tes résultats** (payloads utilisés, étapes, captures d'écran).

Chaque challenge te prépare à des situations réelles de pentest sur API/IoT.

Bonne chance, et pense à réinitialiser l'environnement après exploitation pour les autres !

29. Fiche d'audit de sécurité | ANNEXE [4].

Fiche d'audit de sécurité.		Réf. FAS867469
Projet	Auteur	Date
Projet DVIL	VIALETTE Lysandre	01/06/2025
Partie de projet		
Ex : Sous-système XXX		
Désignation	Risque	Solution
Accès WEB	Intrusion, injection de requête SQL, lecture de trame ; Path traversal ; Injection XSS	Passer le site en HTTPS Sécuriser les formulaires (Dans un but pédagogique)
Trame I²C/SPI	Attaque par force brut ; Sniffer I²C ; SPI ; Reverse	Code pin ou nfc à ne pas mettre en dur ; Solution avec des variables SECRET_KEY
Synthèse		
La plateforme DVIL Secure propose un environnement riche pour la formation offensive. Attention : les failles présentes sont volontaires pour l'apprentissage mais doivent être supprimées ou désactivées en production. L'application respecte de nombreux standards de sécurité mais doit être renforcée sur les points démonstratifs.		

30. Branchement du projet.

