**INGINIERIE INFORMATIQUE ET SYSTEMES EMBARQUES**

**RAPPORT**

Effectué du : 01/04/2025

contrôle D’ACCES A UNE PORTE VIA UNE APPLICATION MOBILE

**Réalisé par :**

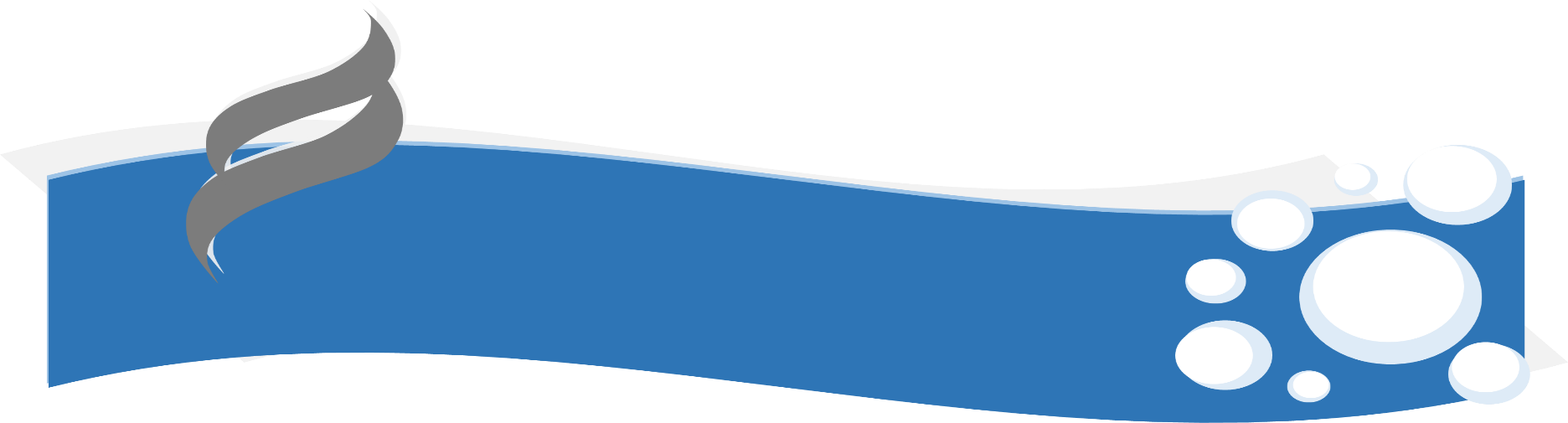
**-***SARMA FATIMA ZAHRA*

*-MAICHNI HAJAR*

*-SEBBAR MOHAMED*

*-SAGO YOUNES*

|  |
| --- |
| **Encadrant :** |
| * *Prof. JAAFAR IDRAIS* |



**Année Universitaire :**

2024/2025

**Table des matières**

[I. Introduction : 5](#_Toc198665251)

[II. Architecture logicielle : 6](#_Toc198665252)

[1. Description générale de l’architecture du système : 6](#_Toc198665253)

[i. Composants principaux : 6](#_Toc198665254)

[ii. Fonctionnement général du système 6](#_Toc198665255)

[iii. Technologies utilisées : 7](#_Toc198665256)

[2. Déploiement distribué : 7](#_Toc198665257)

[i. Topologie du réseau : 7](#_Toc198665258)

[ii. Sécurité et sessions : 7](#_Toc198665259)

[III. Diagramme uml : 8](#_Toc198665260)

[1. Diagramme des Cas d’utilisation : 8](#_Toc198665261)

[2. Diagramme de classes : 9](#_Toc198665262)

[3. Diagramme de Séquence : 9](#_Toc198665263)

[4. Diagramme de déploiement : 11](#_Toc198665264)

[5. Diagramme de composants : 12](#_Toc198665265)

[IV. Détails Techniques et implémentation : 12](#_Toc198665266)

[1. Explication des techniques utilisées : 12](#_Toc198665267)

[2. Implémentation de l’ESP32 dans Wokwi : 13](#_Toc198665268)

[i. Configuration matérielle : 13](#_Toc198665269)

[ii. Logique principale : 14](#_Toc198665270)

[iii. Détail du code : 14](#_Toc198665271)

[3. Serveur Flask : 15](#_Toc198665272)

[i. Point d’Entrée Principal : 15](#_Toc198665273)

[ii. Logique de validation : 15](#_Toc198665274)

[4. Intégration de Ngrok: 15](#_Toc198665275)

[i. Commande de lancement : 15](#_Toc198665276)

[ii. Conditions de fonctionnement : 16](#_Toc198665277)

[5. Serveur gRPC : 16](#_Toc198665278)

[i. Code Serveur gRPC : 16](#_Toc198665279)

[ii. Configuration de la Connexion (SQL Server) : 17](#_Toc198665280)

[V. Tests et validation : 17](#_Toc198665281)

[1. Tests Unitaires : 17](#_Toc198665282)

[2. Tests d’integration complète : 18](#_Toc198665283)

[i. Accès Autorisé (UID Valide) 18](#_Toc198665284)

[ii. Accès refusé (UID Invalide/Inconnu) : 20](#_Toc198665285)

[iii. Problèmes réseau/serveur : 20](#_Toc198665286)

[iv. Configuration Système Critique : 21](#_Toc198665287)

[3. Journal des Incidents et Résolutions : 22](#_Toc198665288)

[4. Documentation Technique : 23](#_Toc198665289)

[i. Logs Serveur (Extraits) : 23](#_Toc198665290)

[ii. Base de données (Extrait) : 23](#_Toc198665291)

[iii. Flux Ngrok : 24](#_Toc198665292)

[5. Perspectives d’amélioration : 24](#_Toc198665293)

[i. Monitoring Temps Réel : 24](#_Toc198665294)

[ii. Gestion des Erreurs : 24](#_Toc198665295)

[iii. Sécurité : 24](#_Toc198665296)

***Tables des figures***

[Figure 1: Diagramme des cas d’utilisation. 8](#_Toc198233493)

[Figure 2: Diagramme de classes. 9](#_Toc198233494)

[Figure 3: Diagramme de séquence. 10](#_Toc198233495)

[Figure 4: Diagramme de déploiement. 11](#_Toc198233496)

[Figure 5: Diagramme de composants. 12](#_Toc198233497)

[Figure 6: Le montage de matérielles. 13](#_Toc198233498)

[Figure 7: Tunnel Ngrok actif redirigeant les requêtes vers le serveur Flask local. 17](#_Toc198233499)

[Figure 8: Traitement réussi d'un UID (HTTP 200) dans le serveur de développement Flask. 17](#_Toc198233500)

[Figure 9: Journaux du serveur Flask montrant le traitement des UID et les réponses HTTP. 17](#_Toc198233501)

[Figure 10: Enregistrements en base de données des tentatives d'accès avec statut de validation des UID. 18](#_Toc198233502)

[Figure 11: Journal des accès ESP32 - Réponses du système. 18](#_Toc198233503)

[Figure 12 : Journal d'erreur Flask montrant un échec de traitement de l'UID (HTTP 500). 18](#_Toc198233504)

[Figure 13: Accès refusé en raison du rejet par le serveur Flask. 19](#_Toc198233505)

[Figure 14: Extrait de la base de données - Ligne 4. 19](#_Toc198233506)

[Figure 15: imeout réseau (-11) dû à une instabilité de Ngrok. 19](#_Toc198233507)

[Figure 16: Erreur HTTP 307 indiquant une mauvaise configuration de l'URL Ngrok. 19](#_Toc198233508)

[Figure 17: Erreur de connexion à SQL Server due à des problèmes réseau/firewall. 20](#_Toc198233509)

[Figure 18:Vérification du statut du service SQL Server via le Gestionnaire des services Windows. 20](#_Toc198233510)

[Figure 19: Service SQL Server (MSSQLSERVER) confirmé comme actif dans les services Windows. 21](#_Toc198233511)

[Figure 20: Configuration de la règle firewall pour autoriser les connexions SQL Server 21](#_Toc198233512)

[Figure 21: Journal d'erreur Flask montrant un échec de traitement de l'UID (HTTP 500). 22](#_Toc198233513)

[Figure 22: Accès refusé. 22](#_Toc198233514)

1. Introduction :

Dans un contexte où la sécurité des biens et des personnes constitue un enjeu primordial, le contrôle d’accès physique devient un domaine stratégique. Les systèmes traditionnels, tels que les clés mécaniques, les badges passifs ou les verrous manuels, montrent aujourd’hui leurs limites : absence de traçabilité, impossibilité de supervision à distance, vulnérabilité face à la duplication ou la perte de clés, et manque de réactivité en cas d'intrusion.

En parallèle, l’essor de l’Internet des Objets (IoT), la démocratisation des réseaux sans fil, ainsi que l’usage généralisé des smartphones ouvrent la voie à de nouvelles approches en matière de sécurité. Les utilisateurs recherchent désormais des solutions intelligentes, connectées et accessibles à distance, capables de gérer les accès en temps réel, de les tracer, de les superviser, et de réagir automatiquement en cas d’anomalie.

C’est dans cette optique que s’inscrit notre projet, qui vise à concevoir un système de contrôle d’accès distribué à une porte via une application mobile. Ce système repose sur un lecteur RFID connecté et permet :

* à un utilisateur autorisé d’ouvrir la porte en scannant une carte RFID,
* au serveur central de vérifier les identifiants, d’autoriser ou de refuser l’accès,
* au propriétaire de consulter en temps réel les événements liés aux accès via une application mobile, et de recevoir immédiatement des notifications en cas de tentative non autorisée.

Le projet s’appuie sur une architecture distribuée composée de plusieurs éléments interconnectés : un microcontrôleur ESP32 simulé dans Wokwi, un serveur distant implémenté en PythonavecgRPC et accessible via Ngrok, et une application mobile développée avec ReactNative. Ces composants collaborent à distance pour garantir une gestion sécurisée, réactive et moderne des accès.

**Problématique :**

Comment concevoir une solution de contrôle d’accès basée sur RFID, fiable et distribuée, permettant une gestion centralisée, une supervision en temps réel, et une réaction immédiate en cas de tentative d’accès non autorisé, tout en garantissant une communication rapide, sécurisée et évolutive entre les composants du système ?

Ce projet vise à répondre à cette problématique en exploitant les technologies modernes de l’IoT et du développement mobile, tout en s’appuyant sur une architecture logicielle modulaire, sécurisée et performante, adaptée aux environnements domestiques ou professionnels.

1. A**rchitecture logicielle**:
2. Description générale de l’architecture du système :

Le projet **«** Contrôle d’accès à une porte via une application mobile **»** s’appuie sur une architecture distribuée intelligente, conçue pour permettre à un propriétaire de maison de gérer à distance l’accès à sa porte d’entrée grâce à un système RFID connecté via WiFi à une plateforme centrale de gestion.

Le dispositif autorise les membres de la famille, préalablement enregistrés, à déverrouiller la porte en scannant leur carte RFID. En parallèle, le propriétaire peut suivre en temps réel toutes les tentatives d’accès via une application mobile dédiée. Toute tentative non autorisée est automatiquement enregistrée et signalée.

Le système repose sur gRPC (Google Remote Procedure Call) comme protocole de communication distante, offrant des performances élevées, une faible latence, et une sécurité renforcée, idéales pour un environnement distribué et en temps réel.

1. Composants principaux :

* ESP32 + lecteur RFID : Lit les cartes RFID et envoie l'UID via WiFi.
* Serveur gRPC (Python) : Authentifie les cartes, prend des décisions d'accès, envoie les commandes à l'ESP32 et journalise tous les événements.
* Tunnel Ngrok : Assure une connexion sécurisée entre l’ESP32 et le serveur gRPC local, en exposant une URL publique accessible depuis Internet.
* Application mobile du propriétaire (React Native) : Permet au propriétaire de consulter les accès, ajouter/supprimer des cartes autorisées, et recevoir des notifications d'accès refusé.

1. Fonctionnement général du système
2. Un membre de la famille scanne sa carte RFID sur le lecteur connecté à l’ESP32.
3. L’ESP32 envoie l’UID de la carte au serveur via WiFi (par gRPC ou requête HTTP).
4. Le serveur vérifie si l’UID est autorisé :

* Si oui : il envoie un signal d’ouverture à l’ESP32 et enregistre l’accès.
* Si non : l’accès est refusé, l’événement est enregistré, et une alerte peut être envoyée au propriétaire.

1. Le propriétaire peut, via l’application mobile :

* Gérer les cartes autorisées.
* Consulter les journaux d’accès.
* Être alerté des tentatives non autorisées.

1. Technologies utilisées :

|  |  |
| --- | --- |
| **Composant** | **Technologies** |
| ESP32 + RFID | Simulé via Wokwi avec WiFi + lecteur RC522 |
| Communication | gRPC (Python) |
| Backend | Python (serveur gRPC avec vérification des droits) |
| Application mobile | React Native |
| Protocoles | gRPC pour communication distante, HTTP/2, WiFi pour transmission. |

Tableau 1: technologies utilisées

1. Déploiement distribué :

Le système est réparti entre plusieurs entités qui communiquent entre elles via le réseau :

* ESP32 + RFID : déployé sur un microcontrôleur simulé (Wokwi), connecté au WiFi.
* Serveur gRPC : hébergé sur une machine locale ou cloud. Il est l’unité de traitement central.
* Application mobile du propriétaire : installée sur son smartphone, utilisée pour la supervision.

1. Topologie du réseau :

Carte RFID──► ESP32 ──► gRPC ──► Ngrok──► Backend +SQL DB──► Signal retour (Y/N)

│

▼

Application Mobile (visualisation et gestion)

1. Sécurité et sessions :
   * Chaque carte RFID est liée à un utilisateur spécifique dans la base de données.
   * Les tentatives d’accès sont journalisées avec :

* UID de la carte
* Heure et date
* Statut (autorise / refuse)
  + L’application mobile utilise une authentification propriétaire.

1. D**iagramme** uml:
2. Diagramme des Cas d’utilisation :

Figure 1: Ce diagramme décrit les principales interactions entre les utilisateurs et le système. Il inclut des cas d'utilisation comme l'Ajout d'une carte autorisée, la Consultation des logs d'accès, et la Réception de notifications sur l'application mobile. Les acteurs, comme l'Utilisateur ou l'Administrateur, interagissent avec le système pour gérer les accès à la porte, configurer des cartes autorisées, et recevoir des informations en temps réel.

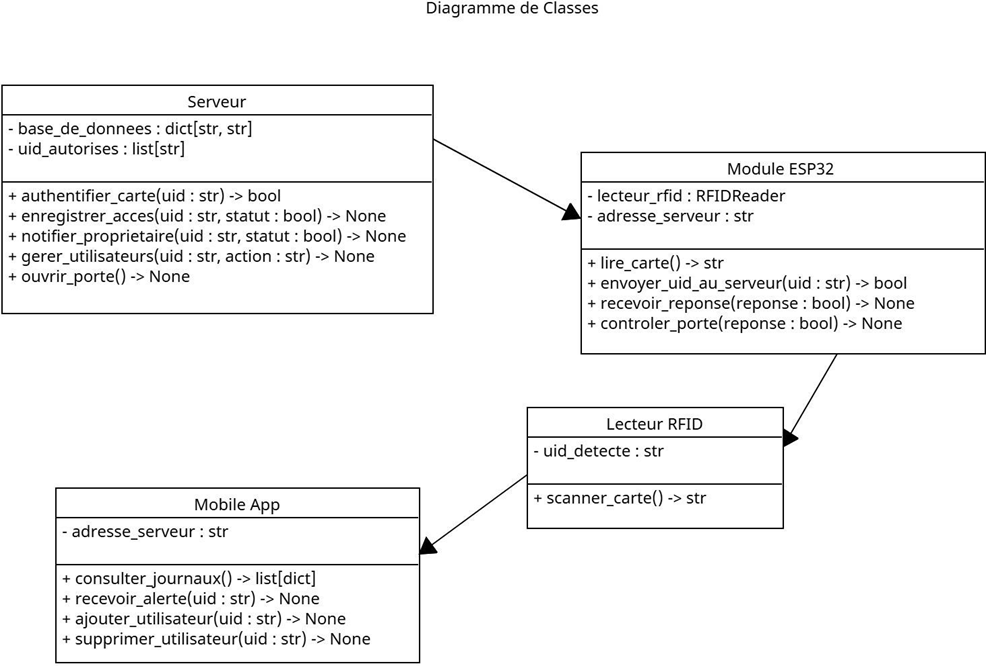
1. Diagramme de classes :

Figure 2: Ce diagramme définit la structure détaillée des classes du système, ainsi que leurs attributs et méthodes. Il inclut les classes utilisées. Chaque classe possède des variables et des méthodes qui définissent le comportement du système. Le diagramme montre également les relations entre les classes, telles que l’envoi de commandes entre le Serveur et l’ESP32, ou l'ajout de cartes autorisées dans la Base de données du Serveur.

1. Diagramme de Séquence :

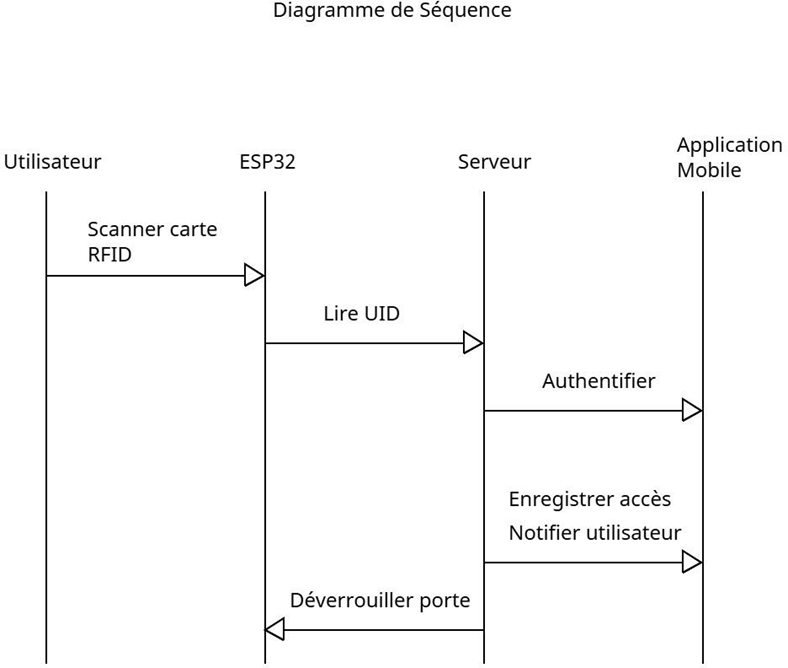
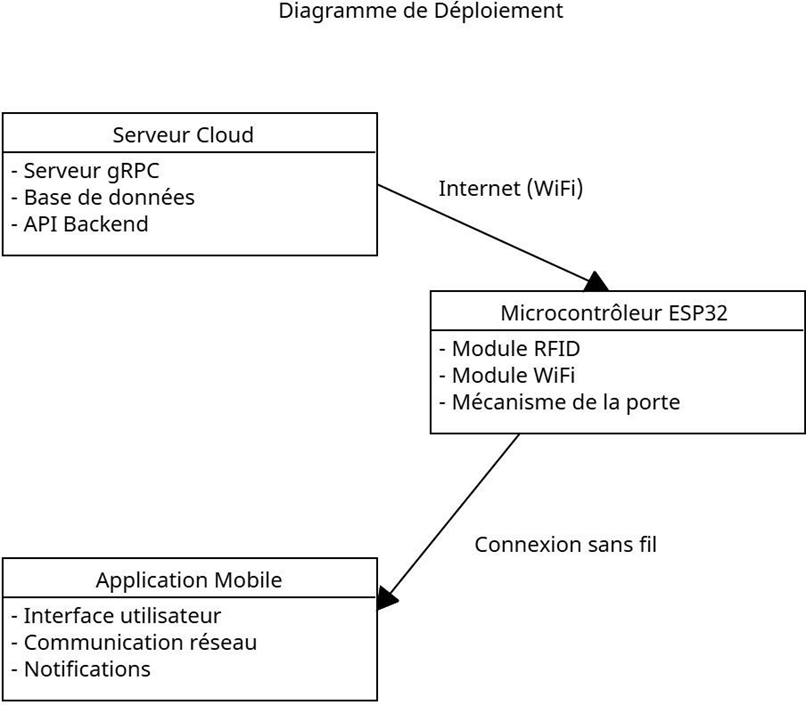
****

Figure 3  : Ce diagramme montre l'ordre chronologique des interactions entre les composants du système lors de l'exécution d'une fonctionnalité spécifique. Il illustre comment les objets (comme Application Mobile, Serveur, et ESP32) échangent des messages pour accomplir une tâche. Par exemple, lorsqu'un membre de la famille scanne une carte RFID, l'Application Mobile envoie une requête au Serveur pour vérifier l'UID, et le Serveur envoie ensuite une commande au ESP32 pour ouvrir la porte, ou renvoie un message de refus si l'UID est non autorisé. Ce diagramme aide à visualiser le flux d'exécution et l'interdépendance entre les composants du système au cours de l'interaction.



1. Diagramme de déploiement :

*Figure 4: Ce diagramme montre comment les composants du système sont déployés dans l'environnement physique. Il inclut la répartition des serveurs, des ESP32, et des appareils mobiles sur le réseau. Il illustre comment ces composants communiquent entre eux à travers un réseau, que ce soit localement ou via le cloud, et comment chaque appareil (Serveur, ESP32, Application Mobile) est connecté pour assurer le bon fonctionnement du système.*

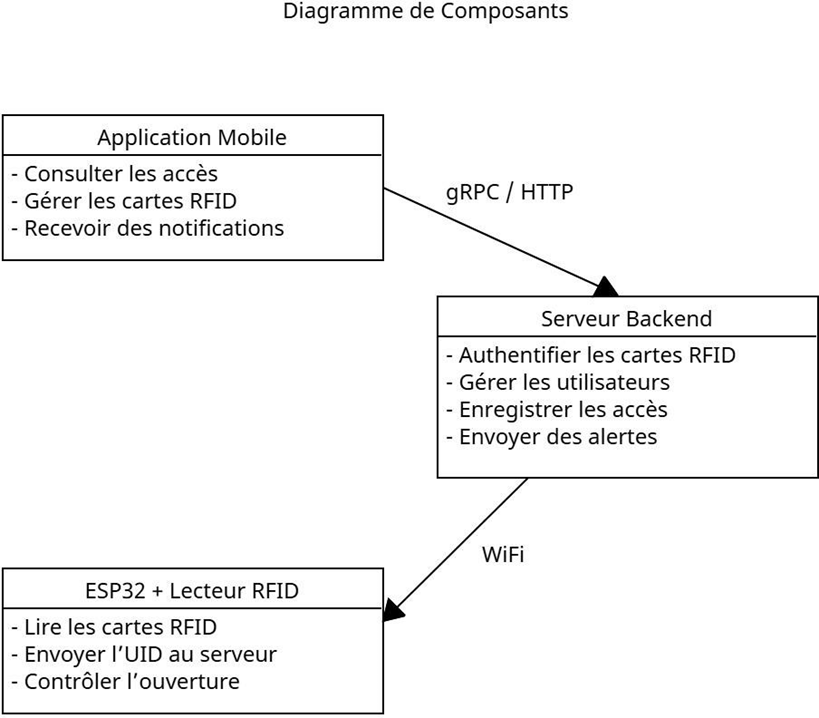
1. Diagramme de composants :

Figure 5: Ce diagramme représente les composants principaux du système et leur interaction. Il inclut des éléments comme le Serveur, l'ESP32, et l'Application Mobile, illustrant comment ils communiquent entre eux. Le Serveur vérifie l’accès des cartes RFID et envoie des commandes au ESP32 pour ouvrir ou fermer la porte. L'Application Mobile permet à l'utilisateur de gérer l'accès et de recevoir des notifications.

1. Détails Techniques et implémentation :
2. Explication des techniques utilisées :

* **Wokwi** est une plateforme de simulation en ligne pour microcontrôleurs comme l’ESP32. Elle permet de tester du code, des capteurs et composants sans matériel physique. Dans notre cas, nous utilisons Wokwi pour simuler le comportement d’un dispositif de contrôle d’accès avec clavier, bouton et LED.
* **Flask** est un micro-framework Python léger, parfait pour créer rapidement des serveurs web simples. Dans ce projet, Flask reçoit les identifiants saisis par l’utilisateur et décide si l’accès est accordé ou refusé.
* **Ngrok** est un outil qui crée un tunnel sécurisé entre votre serveur local et Internet. Il fournit une URL publique temporaire, redirigeant vers votre serveur Flask tournant en local. Cela permet à l’ESP32 simulée d’accéder à votre serveur même s’il n’est pas hébergé en ligne.
* **Sql server** SQL Server stocke les UIDs autorisés et enregistre chaque tentative d’accès. Il permet de gérer les droits d’accès et de conserver un historique des passages.
* **Flask front end** L’interface web Flask permet à l’administrateur de voir, ajouter ou supprimer des UIDs autorisés. Elle affiche aussi un journal des tentatives d’accès.

1. Implémentation de l’ESP32 dans Wokwi :
2. Configuration matérielle :

L’ESP32 simulée utilise les composants suivants :

● Broche 26 : LED (pour signaler l’accès)

● Broche 25 : Bouton poussoir (pour valider)

● Clavier matriciel 4x4 connecté aux broches :

* Lignes : 19, 18, 5, 17
* Colonnes : 16, 4, 0, 2

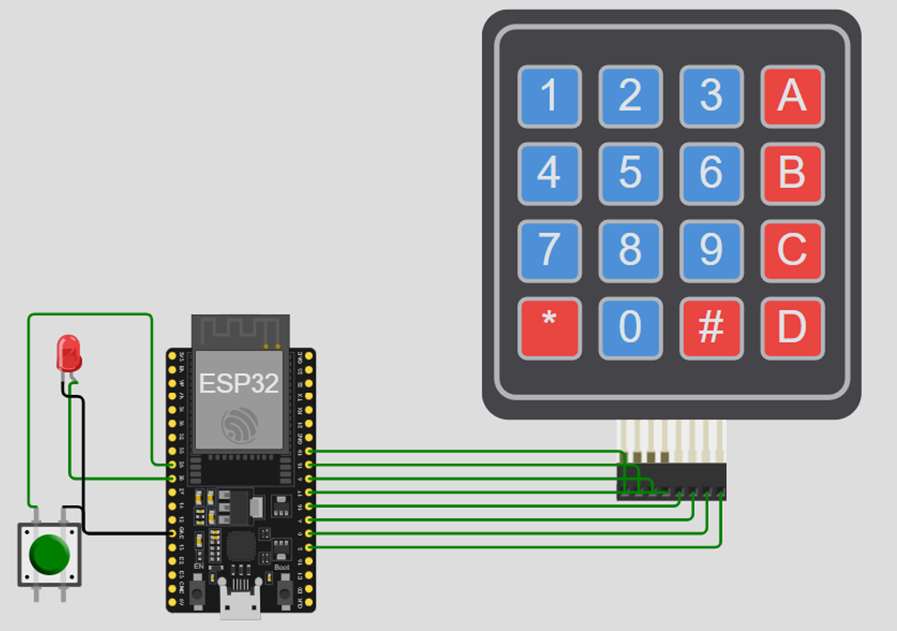


Figure 6: Le montage de matérielles.

1. Logique principale :

Le programme suit cette séquence :

1. Connexion au WiFi “Wokwi-GUEST” (intégré)

2. Initialisation des entrées clavier et bouton

3. Boucle principale :

* Lit les touches saisies
* En cas d’appui sur le bouton :
  + Prépare l’UID au format JSON
  + Envoie au serveur Flask
  + Allume ou éteint la LED selon la réponse

1. Détail du code :

**Configuration réseau :** const char\* ssid = "Wokwi-GUEST";  
 const char\* password = "";  
 const char\* server\_address = "http://[ngrok-url]/uid";

**Définition du clavier :** char keys[4][4] = {  
 {'1','2','3','A'},  
 {'4','5','6','B'},  
 {'7','8','9','C'},  
 {'\*','0','#','D'}  
 };  
 byte rowPins[4] = {19,18,5,17};  
 byte colPins[4] = {16,4,0,2};

**Lecture UID et bouton :** String uid = "";  
 void loop() {  
 char key = keypad.getKey();  
 if(key) {  
 if(key == 'C') uid = "";  
 else uid += key;  
 }  
 if(digitalRead(BUTTON\_PIN) == LOW) {  
 send\_uid\_to\_server(uid);  
 uid = "";  
 }  
 }

**Envoi HTTP :** bool send\_uid\_to\_server(String uid) {  
 HTTPClient http;  
 http.begin(server\_address);  
 http.addHeader("Content-Type", "application/json");  
 String json = "{\"uid\":\"" + uid + "\"}";  
 int httpCode = http.POST(json);  
 }

1. Serveur Flask :
2. Point d’Entrée Principal :

Un seul endpoint : /uid

* + - Accepte uniquement les requêtes POST.
    - Reçoit une chaîne JSON du type : {"uid":"123456"}
    - Renvoie "True" ou "False" selon la validité.

1. Logique de validation :

Validation simple (à remplacer par gRPC dans le futur) :

@app.route('/uid', methods=['POST'])  
 def handle\_uid():  
 uid = request.json.get('uid')  
 access\_granted = uid == "123456"  
 return str(access\_granted), 200

1. Intégration de Ngrok:
2. Commande de lancement :

Lancer le tunnel avec :

ngrok http 5000

Exemple de sortie :

Forwarding https://xyz123.ngrok.io -> http://localhost:5000

1. Conditions de fonctionnement :

* Le serveur Flask doit tourner sur le port 5000.
* L’adresse **Ngrok** doit être copiée dans le code ESP32 (server\_address).
* Aucune règle de pare-feu ne doit bloquer le port 5000.

1. Serveur gRPC :
2. Code Serveur gRPC :

**Importation des bibliothèques nécessaires :**

import grpc

from concurrent import futures

import access\_control\_pb2

import access\_control\_pb2\_grpc

import db\_config

from datetime import datetime

**Dictionnaire uid\_mapping :** liste locale d’UID autorisés

uid\_mapping = {

"123456": "Father",

"123": "Son",

"1234": "Mother"

}

**Classe AccessControlServicer :**

C’est la classe principale qui implémente le service gRPC défini dans le fichier. proto

Class AccessControlServicer(access\_control\_pb2\_grpc.

AccessControlServicer):

**Méthode CheckUID :**

Reçoit une requête contenant un UID depuis un client (dans notre cas c’est l’ESP32).

def CheckUID(self, request, context):

**Fonction serve() :**

Démarre le serveur gRPC sur le port 50051, prêt à recevoir des requêtes.

def serve():

**Exécution du serveur :**

if \_\_name\_\_ == '\_\_main\_\_':

serve()

1. Configuration de la Connexion (SQL Server) :

**Importation de la bibliothèque pyodbc :**

Permet à Python de se connecter à une base de données SQL Server via ODBC (Open Database Connectivity).

import pyodbc

**Fonction get\_connection() :**

def get\_connection():

conn = pyodbc.connect(

"Driver={ODBC Driver 17 for SQL Server};"

"Server=DESKTOP-A37BJM4\\SQLEXPRESS;"

"Database=RFID;"

"UID=ema;"

"PWD=emaema889;"

)

**Retour de la connexion :**

return conn

1. Tests et validation :
2. Tests Unitaires :

**ESP32 (environnement simulé) :**

* + Simulation de lecture d’une carte RFID valide et invalide.
  + Réactions attendues du microcontrôleur selon les signaux reçus (ouvrir / refuser l’accès).

**Backend (gRPC) :**

* + Vérification de la validité des identifiants de carte RFID.
  + Tests des fonctions de vérification des accès : retour « autorisé » ou « refusé » selon les droits.
  + Tests des fonctions de journalisation : insertion correcte des logs dans la base de données.

**Base des données (SQL Server) :**

* Vérification de la structure et de la conformité du schéma des tables.
* Tests des opérations d’insertion, de mise à jour et de suppression (cartes, logs, utilisateurs).
* Validation de la cohérence et de l’intégrité des données (pas de doublons, suppression en cascade correcte).
* Vérification des performances lors des requêtes courantes (authentification, lecture des logs, etc.).

**Application mobile :**

* + Vérification de l’authentification du propriétaire.
  + Tests des interfaces de consultation des accès.
  + Ajout/suppression de cartes autorisées (test des entrées et validations de formulaire).

1. Tests d’integration complète :

**Scénarios Validés**

1. Accès Autorisé (UID Valide)
   * Test : Envoi d’UIDs valides (12345678, 123, 1234) via ESP32 → Flask → SQL Server.
   * Résultat :

Réponse HTTP 200

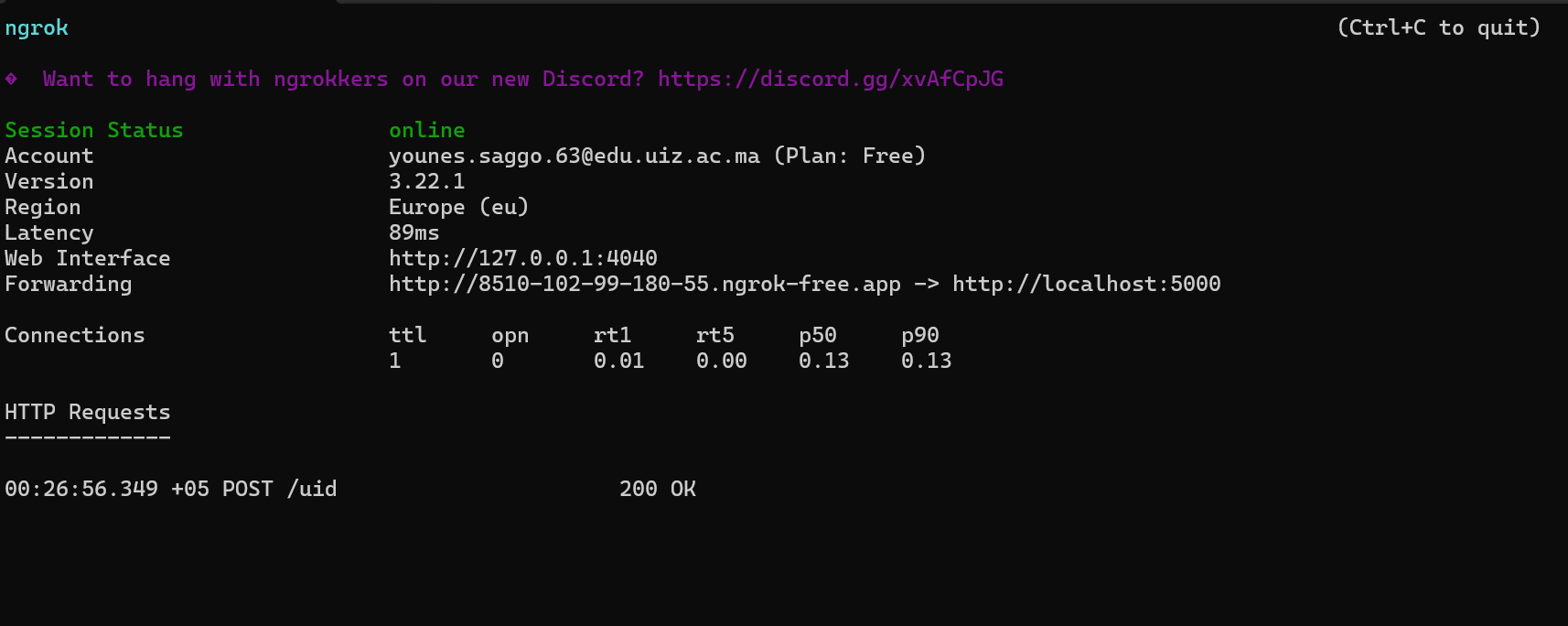


Figure 7: Tunnel Ngrok actif redirigeant les requêtes vers le serveur Flask local.



Figure 8: Traitement réussi d'un UID (HTTP 200) dans le serveur de développement Flask.

* Entrée DB avec access\_granted=1.

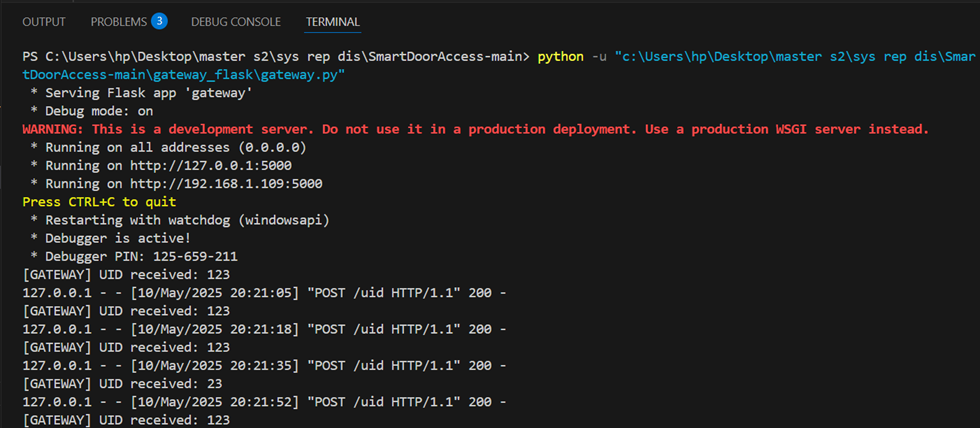


Figure 9: Journaux du serveur Flask montrant le traitement des UID et les réponses HTTP.

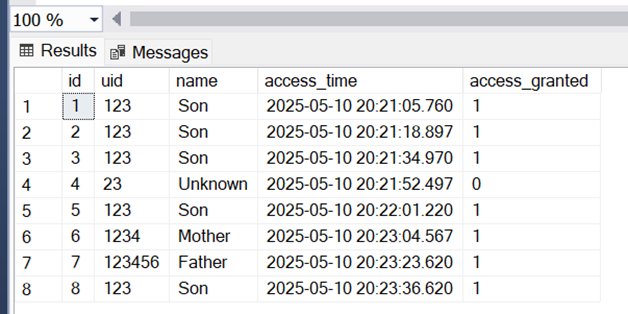


Figure 10: Enregistrements en base de données des tentatives d'accès avec statut de validation des UID.

* Logs ESP32 confirmant l’ouverture de la porte



Figure 11: Journal des accès ESP32 - Réponses du système.

1. Accès refusé (UID Invalide/Inconnu) :
   * + **Test** : UID 23 ou erreur de saisie (123456 vs 12345678).
     + **Résultat** :

* Réponse 500 ou Accès refusé

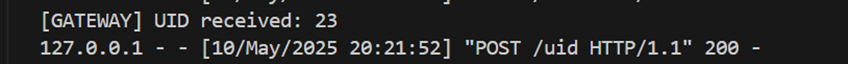


Figure 12 : Journal d'erreur Flask montrant un échec de traitement de l'UID (HTTP 500).

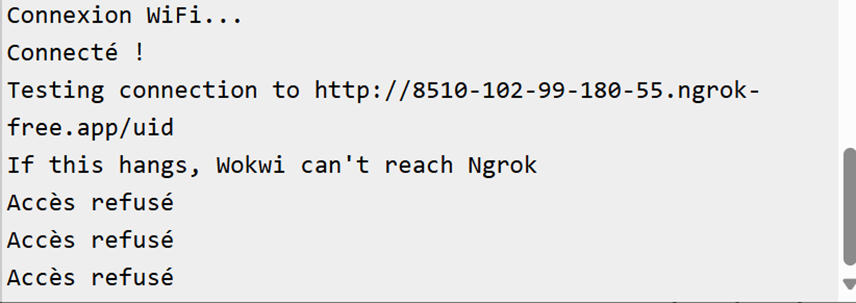


Figure 13: Accès refusé en raison du rejet par le serveur Flask.

* Entrée en base de données avec access\_granted=0.



Figure 14: Extrait de la base de données - Ligne 4.

1. Problèmes réseau/serveur :

* **Timeout/URL Incorrecte :**
* Erreur HTTP-11

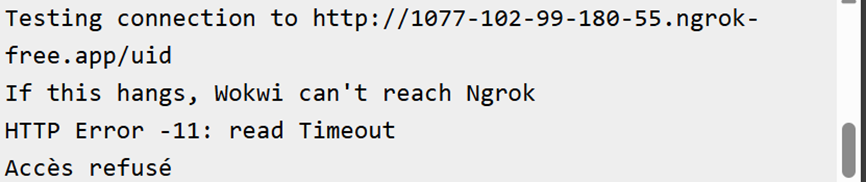


Figure 15: imeout réseau (-11) dû à une instabilité de Ngrok.

* Erreur 307

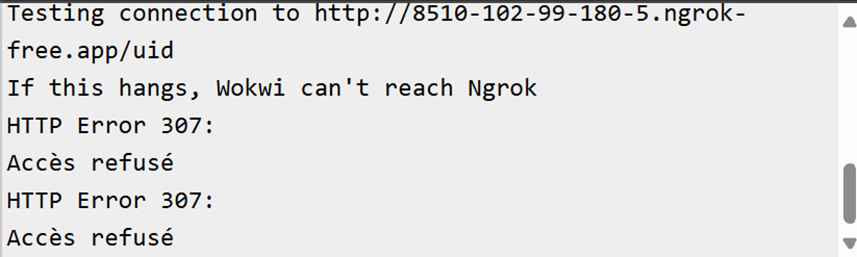


Figure 16: Erreur HTTP 307 indiquant une mauvaise configuration de l'URL Ngrok.

* **Serveur SQL Inaccessible :**
* Erreur 500 avec logs détaillés.

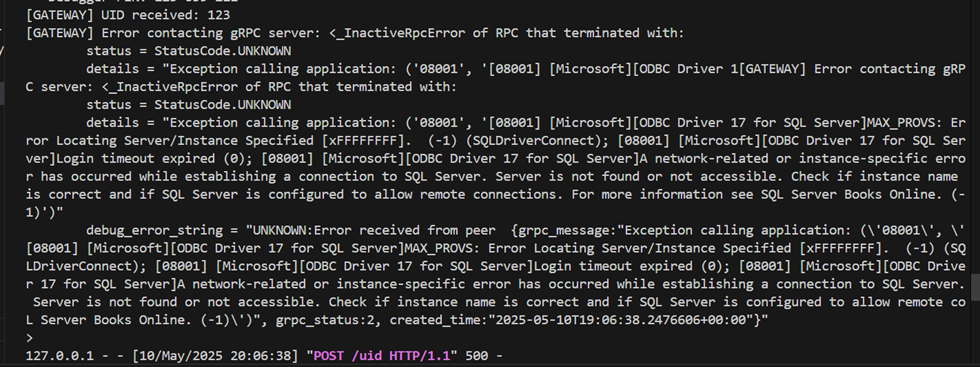


Figure 17: Erreur de connexion à SQL Server due à des problèmes réseau/firewall.

1. Configuration Système Critique :

* **SQL Server Non Démarré :**
* Vérification du service.

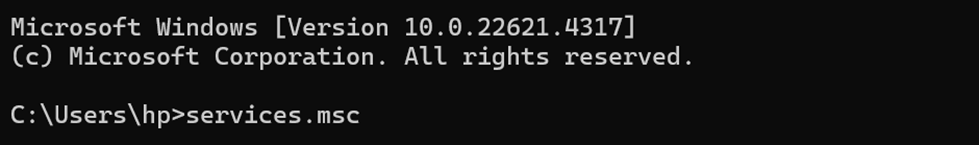


Figure 18:Vérification du statut du service SQL Server via le Gestionnaire des services Windows.

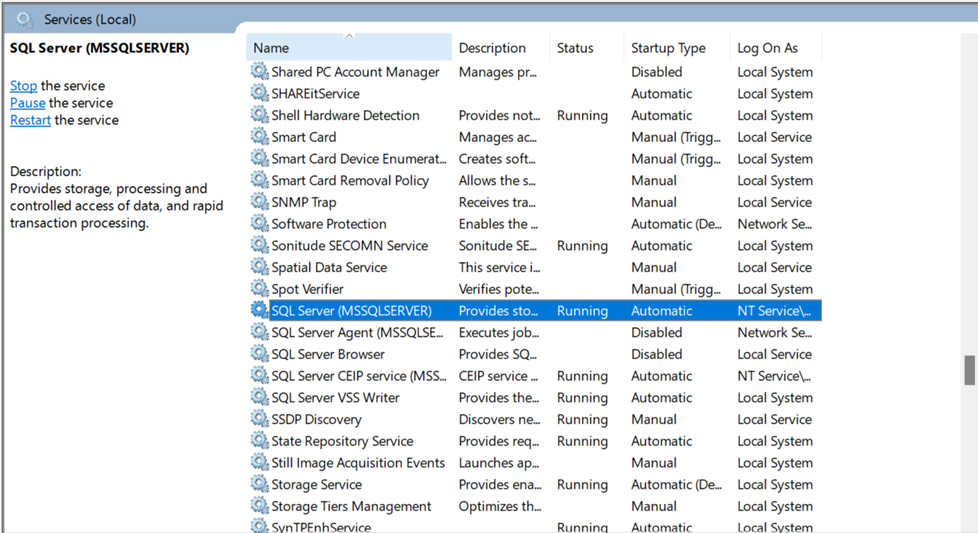


Figure 19: Service SQL Server (MSSQLSERVER) confirmé comme actif dans les services Windows.

* **Firewall Bloquant :**

Autorisation du port 1433.



Figure 20: Configuration de la règle firewall pour autoriser les connexions SQL Server sur le port 1433.

1. Journal des Incidents et Résolutions :

| **Problème** | **Cause Racine** | **Solution** | **Preuve** |
| --- | --- | --- | --- |
| Erreur 500 avec UID valide | Connexion DB bloquée (firewall/instance) | Vérification SQL Server + règles firewall. | Figure 11, 13-14 |
| Timeout (HTTP -11) | Ngrok instable ou URL incorrecte | Redémarrage Ngrok + vérification URL. | Figure 9 |
| Erreur 307 | URL Ngrok mal configurée | Mise à jour de l’URL dans le code ESP32. | Figure 10 |
| UID partiel refusé | Comparaison côté serveur incomplète | Correction de la vérification | Figure 5,7 |

1. Documentation Technique :
2. Logs Serveur (Extraits) :

* **Succès :**

[GATEWAY] UID received: 12345678 → 200 OK (**Figure 7**)

* **Echec :**

[GATEWAY] UID received: 12345678 → 500 (Erreur DB) (**Figures 12, 21-22**)

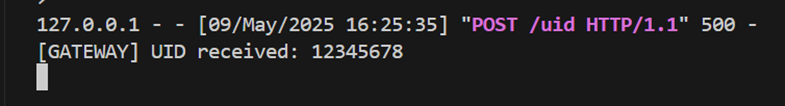


Figure 21: Journal d'erreur Flask montrant un échec de traitement de l'UID (HTTP 500).

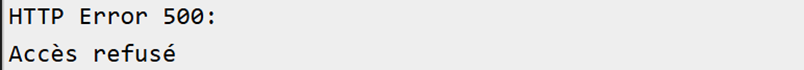


Figure 22: Accès refusé.

1. Base de données (Extrait) :

| **UID** | **Nom** | **Statut** | **Heure** |
| --- | --- | --- | --- |
| 12345678 | Father | Autorisé | 2025-05-08 00:26:56 |
| 23 | Unknown | Refusé | 2025-05-10 20:21:52 |

1. Flux Ngrok :

* Session active avec forwarding vers localhost:5000 (**Figure 6**)

1. Perspectives d’amélioration :
2. Monitoring Temps Réel :

* Ajouter un dashboard pour surveiller l’état du serveur/ESP32.

1. Gestion des Erreurs :

* Implémenter des reprises automatiques côté ESP32 après timeout.

1. Sécurité :

* Chiffrer les UIDs en transit (HTTPS/TLS).