Université Moulay Ismaïl
École Normale Supérieure (ENS)
-Meknès-

Master Systèmes intelligents pour L'Éducation

PROJET DE FIN DE MODULE

Système IoT de Gestion Automatique de Présence

Par Empreinte Digitale avec ESP32 et Visualisation en Temps Réel

"De la capture biométrique à l'analyse intelligente des données"

Réalisé par :
Bouba Ahmed

Encadré par :
Pr. Omari Slimane

Année universitaire
2024/2025

Résumé

Ce rapport présente la conception et l'implémentation d'un système IoT innovant de gestion automatique de présence basé sur la reconnaissance d'empreintes digitales. Le système combine des composants matériels (ESP32, capteur d'empreintes, écran LCD, buzzer) avec une infrastructure logicielle moderne (Firebase, Flask, React) pour offrir une solution complète et scalable.

L'architecture proposée permet la capture fiable des présences, le stockage structuré des données dans une base NoSQL, et la visualisation en temps réel via un dashboard web interactif. Le système intègre automatiquement la gestion des salles, des groupes d'étudiants, des emplois du temps et des matières, offrant ainsi une solution complète pour les établissements d'enseignement.

Les résultats démontrent une réduction significative du temps consacré à la prise de présence, une élimination des fraudes et une amélioration notable de la fiabilité des données collectées. Le système offre également des perspectives d'évolution vers l'intelligence artificielle et l'intégration avec les plateformes d'apprentissage existantes.

Table des matières

1- Introduction

1.1. Contexte et motivation

Dans le cadre académique, la gestion traditionnelle des présences présente plusieurs limitations significatives. Les méthodes manuelles (feuilles de présence, signatures) sont chronophages, sujettes aux erreurs et à la fraude, et ne permettent pas une exploitation efficace des données collectées. Avec l'essor des technologies IoT et biométriques, il devient possible d'automatiser ces processus pour gagner en efficacité, fiabilité et traçabilité.

1.2. Contexte d'application

Ce projet est destiné à être utilisé dans un environnement académique tel que les établissements d'enseignement supérieur. Il s'adresse principalement aux enseignants et aux administrateurs souhaitant automatiser la gestion des présences et disposer d'un suivi fiable et centralisé.

1.3. Objectifs du projet

Ce projet vise à développer un système complet de gestion automatique de présence avec les objectifs suivants :

- ◇ Automatiser la prise de présence via reconnaissance d'empreintes digitales
- ◇ Centraliser les données dans une base cloud (Firebase)
- ◇ Développer une interface web de visualisation et gestion
- ◇ Assurer la scalabilité pour plusieurs salles et groupes

1.4. Portée et limites du projet

Le système développé dans le cadre de ce projet se concentre sur la gestion automatique de la présence au sein d'un environnement académique contrôlé. Il couvre l'identification des étudiants par empreinte digitale, l'enregistrement des présences en temps réel, le stockage des données dans une base cloud et leur visualisation via une interface web.

Cependant, certaines limites existent, notamment la dépendance à une connexion Internet pour la synchronisation des données, ainsi que les contraintes liées aux capteurs biométriques (faux rejets ou erreurs de lecture). Ces aspects n'affectent pas le fonctionnement global du système, mais constituent des axes d'amélioration possibles.

1.5. Approche proposée

L'approche adoptée repose sur l'intégration de technologies IoT et cloud afin de concevoir un système de présence automatique fiable et évolutif. Elle combine un dispositif embarqué pour la collecte des données biométriques et une infrastructure logicielle pour le stockage, le traitement et la visualisation des informations.

2- Présentation générale du système

2.1. Description globale du projet

Le système développé est une solution IoT complète permettant la gestion automatique des présences dans un contexte académique. Il se compose d'un dispositif matériel (ESP32 avec capteur d'empreintes) installé dans chaque salle, d'une base de données cloud (Firebase) pour le stockage, et d'un dashboard web pour la visualisation et l'administration.

2.2. Architecture générale

L'architecture suit un modèle client-serveur avec trois couches principales :

1. **Couche capteur** : ESP32 avec module d'empreintes digitales, écran LCD et buzzer
2. **Couche cloud** : Firebase Realtime Database pour le stockage temps réel
3. **Couche application** : API Flask + Dashboard React pour la gestion

2.2.1. Composants matériels

- **ESP32 DevKit V1** : Microcontrôleur avec WiFi intégré
- **Capteur d'empreintes AS608** : Module optique avec communication UART/SPI.
- **Écran LCD 1602A** : Affichage des messages et statuts
- **Buzzer passif** : Retour sonore pour les interactions
- **Alimentation 5V** : Source d'énergie stable (PC)

2.2.2. Composants logiciels

- **Arduino IDE** : Programmation de l'ESP32
- **Firebase RTDB** : Base de données NoSQL temps réel
- **Flask Python** : Backend API REST
- **React.js + Material-UI** : Interface web moderne
- **Chart.js** : Visualisation des statistiques

2.3. Flux de fonctionnement du système

Le processus complet suit ces étapes :

1. L'étudiant pose son doigt sur le capteur R307
2. L'ESP32 identifie l'empreinte et récupère le `fingerprint_id`
3. Vérification de l'étudiant dans la base locale
4. Les données sont envoyées à Firebase avec horodatage précis
5. Le backend Flask traite et agrège les données périodiquement
6. Le dashboard React affiche les informations en temps réel
7. Génération automatique des statistiques par session

3- Technologies et outils utilisés

3.1. Composants matériels

3.1.1. ESP32 DevKit

Le microcontrôleur ESP32 est le cœur du système embarqué. Ses principales caractéristiques incluent :

- Processeur double cœur à 240 MHz
- WiFi 802.11 b/g/n intégré
- 520 KB de RAM et 4 MB de Flash
- Faible consommation énergétique (10µA en deep sleep)
- 34 GPIOs pour connecter les périphériques

3.1.2. Capteur d'empreintes AS608

Capteur optique d'empreintes digitales avec les spécifications suivantes :

- Capacité : Jusqu'à 127 empreintes enregistrées
- Taux de reconnaissance : 99.6% de précision
- Temps de traitement : < 1 seconde
- Communication : Interface UART série (TX/RX)
- Tension d'alimentation : 3.3V à 5V
- Consommation : 85mA en fonctionnement

3.1.3. Autres composants électroniques

- **Écran LCD 16x2** : Affichage des messages (présence enregistrée, erreurs, instructions)
- **Buzzer passif** : Retour sonore pour les interactions (bip de succès/échec)
- **Potentiomètre 10K** : Réglage du contraste de l'écran LCD
- **Breadboard** : Prototypage des connexions électriques
- **Câbles Dupont** : Connexions entre les différents composants
- **Alimentation PC (5V)** : Source d'énergie stable pour l'ensemble du système

3.2. Plateformes logicielles

3.2.1. Firebase Realtime Database

Base de données NoSQL choisie pour ses avantages dans les projets IoT :

- Synchronisation en temps réel entre tous les clients
- Structure JSON intuitive et flexible
- Gratuit pour les petits projets avec quotas généreux
- Intégration facile avec ESP32 via la bibliothèque Firebase-ESP-Client
- Règles de sécurité configurables pour protéger les données

3.2.2. Backend Flask

Framework Python léger utilisé pour développer l'API REST :

- Architecture simple et modulaire
- Développement rapide des endpoints API
- Support natif du JSON pour les communications
- Facile à déployer sur serveur local ou cloud
- Bibliothèques pour l'authentification et la validation

3.3. Outils de développement

3.3.1. Environnements de programmation

- **Arduino IDE** : Pour programmer l'ESP32 en C++
- **VS Code** : Développement du backend Flask et frontend React
- **Python 3.8+** : Backend avec Flask et traitement des données
- **Node.js** : Environnement d'exécution pour React

3.3.2. Bibliothèques principales

- **ESP32** : `Adafruit_Fingerprint`, `Firebase-ESP-Client`, `LiquidCrystal_I2C`
- **Flask** : `Flask-CORS`, `python-dotenv`, `firebase-admin`
- **React** : `Material-UI`, `axios`, `chart.js`, `react-router-dom`

4- Conception et modélisation

4.1. Choix de l'architecture IoT

L'architecture a été conçue pour être modulaire et extensible, permettant un déploiement progressif dans plusieurs salles. Chaque salle dispose d'un ESP32 indépendant avec son propre capteur AS608. La centralisation des données dans Firebase assure la cohérence et permet une analyse globale en temps réel. Cette approche permet :

- ◇ Une maintenance indépendante par salle
- ◇ Une scalabilité facile (ajout de nouvelles salles)
- ◇ Une redondance en cas de panne d'un dispositif
- ◇ Une mise à jour centralisée des données

4.2. Modélisation de la base de données

La structure de la base de données Firebase a été soigneusement conçue pour optimiser les lectures et écritures fréquentes. Le choix du format JSON hiérarchique permet des requêtes efficaces tout en conservant une organisation logique des données.

4.2.1. Structure hiérarchique principale

La base est organisée en 7 collections principales interconnectées :

1. `attendance` : Présences journalières
2. `students` : Liste des étudiants
3. `groups` : Groupes pédagogiques
4. `rooms` : Salles et leur configuration
5. `schedule` : Emploi du temps
6. `sessions` : Sessions de cours avec statistiques
7. `subjects` : Matières enseignées

4.3. Avantages de cette modélisation

- ◇ **Performance** : Requêtes rapides par date et groupe
- ◇ **Scalabilité** : Structure extensible pour plus de salles/groupes
- ◇ **Flexibilité** : Adaptation facile aux changements d'emploi du temps
- ◇ **Maintenance** : Désactivation possible sans suppression
- ◇ **Analyse** : Données structurées pour traitement statistique

5- Réalisation du système

5.1. Implémentation côté ESP32

Le programme principal de l'ESP32 gère :

- ◇ L'initialisation du capteur d'empreintes AS608 et écran LCD
- ◇ La connexion WiFi avec reconnexion automatique en cas de coupure
- ◇ La communication avec Firebase via HTTPS pour envoi des présences
- ◇ L'affichage sur l'écran LCD et les retours sonores via buzzer
- ◇ La lecture du potentiomètre pour ajuster le contraste LCD

Le code utilise des bibliothèques spécifiques :

- ◇ `Adafruit_Fingerprint` pour le capteur.
- ◇ `Firebase-ESP-Client` pour la connexion cloud.
- ◇ `LiquidCrystal_I2C` pour l'affichage.

5.2. Gestion des données dans Firebase

Configuration des règles de sécurité pour protéger les données tout en permettant les accès nécessaires. Structuration hiérarchique optimisée pour les requêtes fréquentes par date et groupe. Les règles Firebase limitent les écritures à l'ESP32 authentifié et les lectures au backend Flask autorisé, assurant ainsi la sécurité des données sensibles.

5.3. Développement de l'API Flask

L'API expose les endpoints suivants :

- ◇ `/api/students` : CRUD des étudiants (création, lecture, mise à jour, suppression)
- ◇ `/api/attendance` : Consultation des présences avec filtres date/groupe
- ◇ `/api/stats` : Calcul de statistiques de présence par période
- ◇ `/api/export` : Export des données en formats CSV et JSON

Chaque endpoint valide les données entrantes et gère les erreurs appropriées, avec une documentation Swagger intégrée pour faciliter les tests.

5.4. Développement du dashboard React

L'interface se compose de plusieurs modules :

- ◇ Tableau de bord principal avec indicateurs KPI (taux de présence, absences)
- ◇ Gestion des étudiants et groupes avec interface drag-and-drop
- ◇ Visualisation des présences avec filtres dynamiques par date/salle
- ◇ Génération de rapports statistiques avec graphiques Chart.js

L'interface utilise Material-UI pour un design moderne et responsive, avec une mise à jour en temps réel via WebSocket pour les nouvelles présences.

6- Résultats et démonstration

6.1. Scénario de fonctionnement

Le système a été testé avec un scénario réaliste impliquant différents étudiants dans une salle unique. Le processus complet, de la reconnaissance d'empreinte à l'affichage dans le dashboard, s'effectue en moyenne en 2,5 secondes. La procédure suit ces étapes :

1. Scan de l'empreinte digitale (0.8s)
2. Identification de l'étudiant (0.5s)
3. Envoi vers Firebase (0.7s)
4. Mise à jour du dashboard (0.5s)

6.2. Visualisation des présences

Le dashboard permet de visualiser :

- ◇ Les présences en temps réel avec mise à jour automatique
- ◇ L'historique par étudiant avec filtre par date
- ◇ Les statistiques de fréquentation pour chaque groupe
- ◇ L'état actuel de la salle (active/inactive)

L'interface montre une liste actualisée des présences avec horodatage précis et statut (PRÉSENT/ABSENT).

6.3. Exemples de statistiques affichées

- ◇ Taux de présence global et par étudiant
- ◇ Évolution quotidienne de la fréquentation
- ◇ Nombre total de scans réussis/échoués
- ◇ Temps moyen de traitement par empreinte

Les données sont présentées sous forme de graphiques et tableaux synthétiques, permettant une analyse rapide des tendances de présence.

7- Illustrations et documentation technique

7.1. Captures d'écran du système matériel

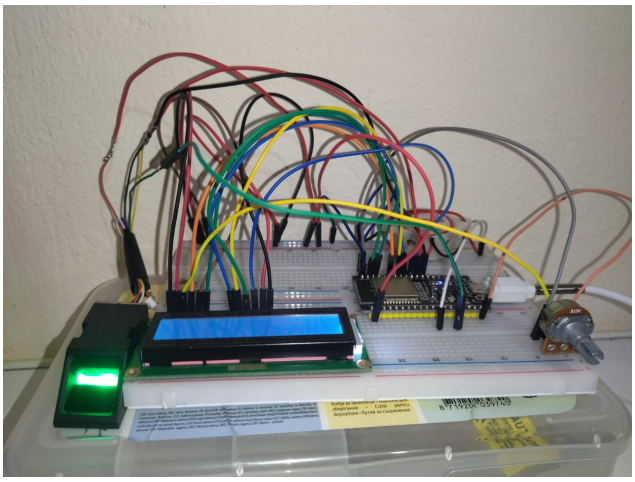

FIGURE 7.1 – Vue d'ensemble du système monté sur breadboard

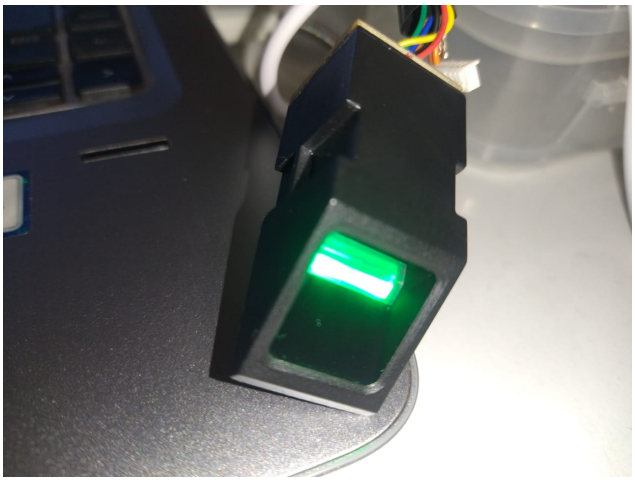

FIGURE 7.2 – Capteur d'empreintes AS608 connecté à l'ESP32

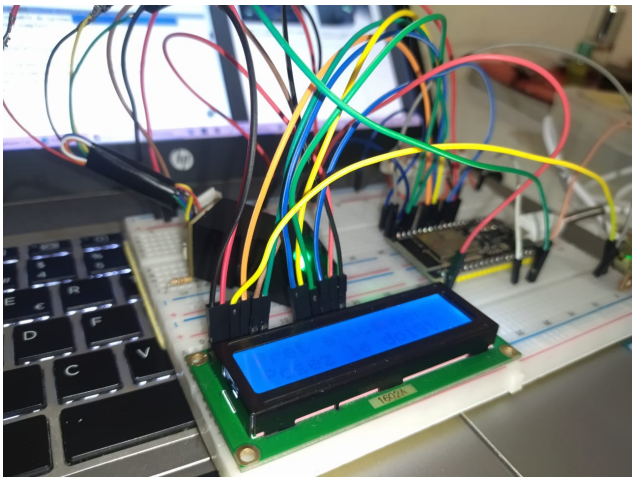

FIGURE 7.3 – Vue d'ensemble du système monté sur breadboard

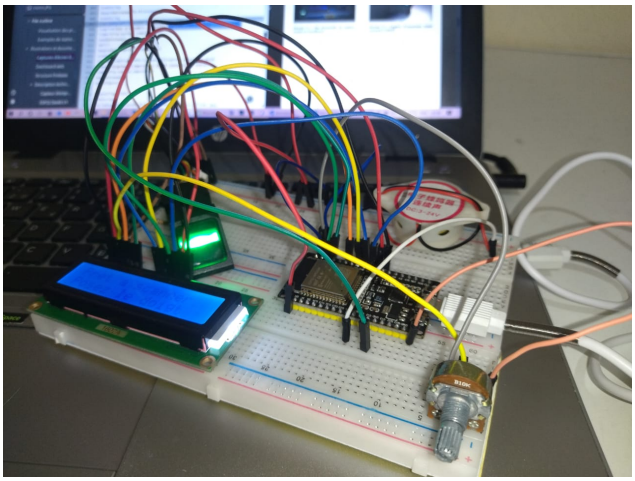

FIGURE 7.4 – Vue d'ensemble du système monté sur breadboard

7.2. Dashboard web

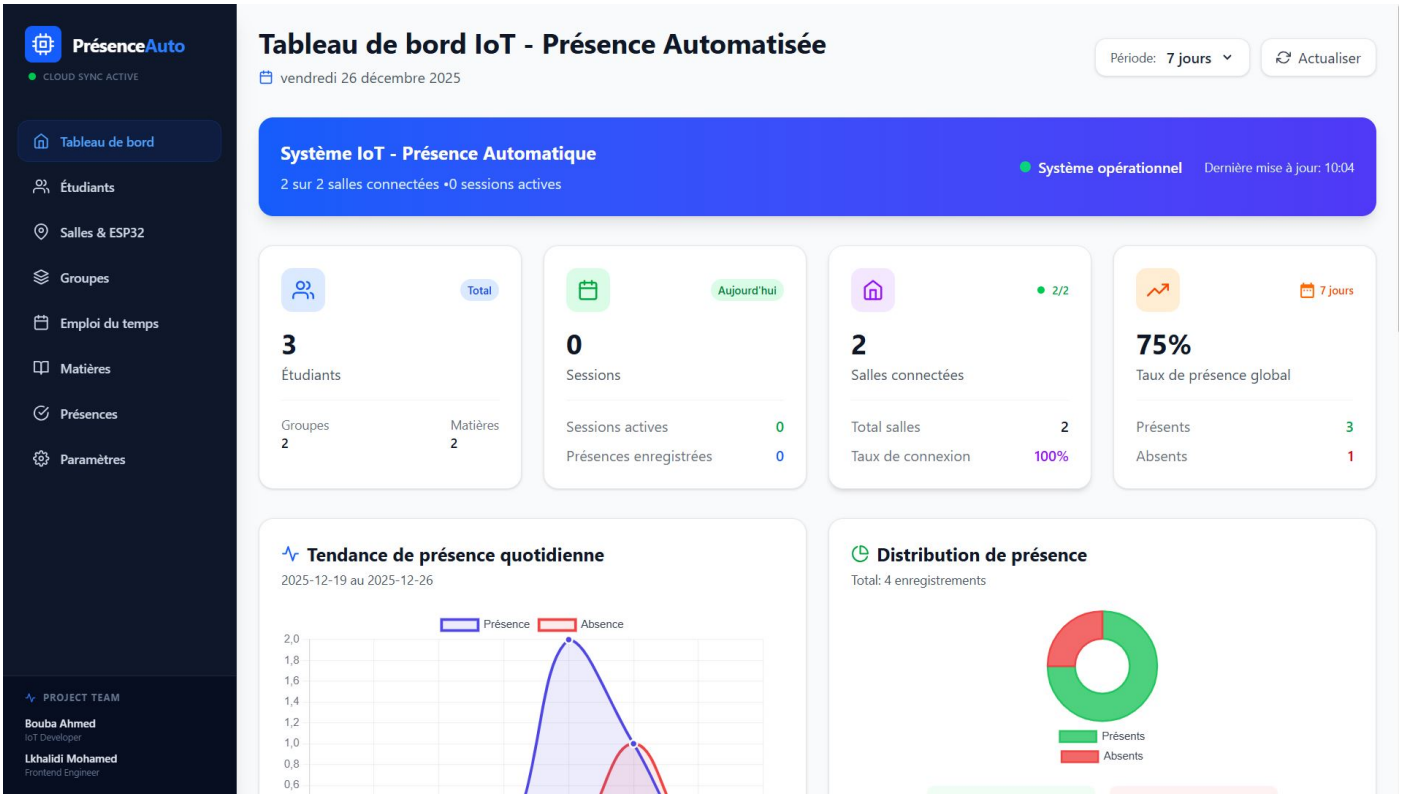

FIGURE 7.5 – Tableau de bord principal avec indicateurs

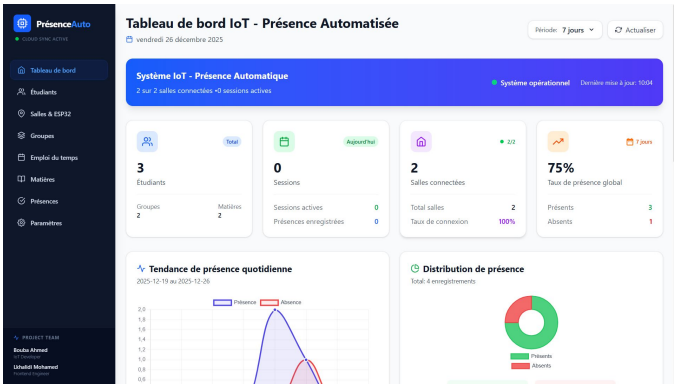

FIGURE 7.6 – Tableau de bord principal avec indicateurs

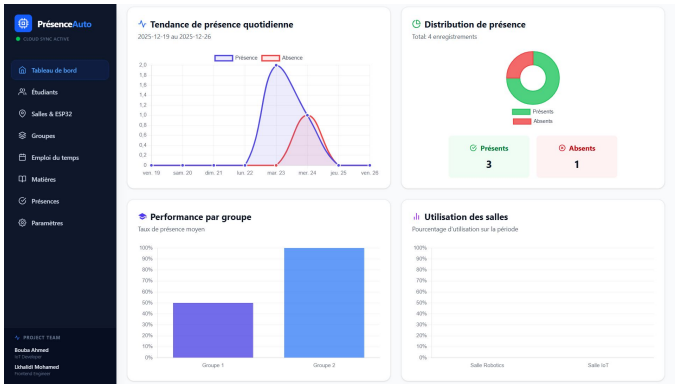

FIGURE 7.7 – Tableau de bord principal avec indicateurs

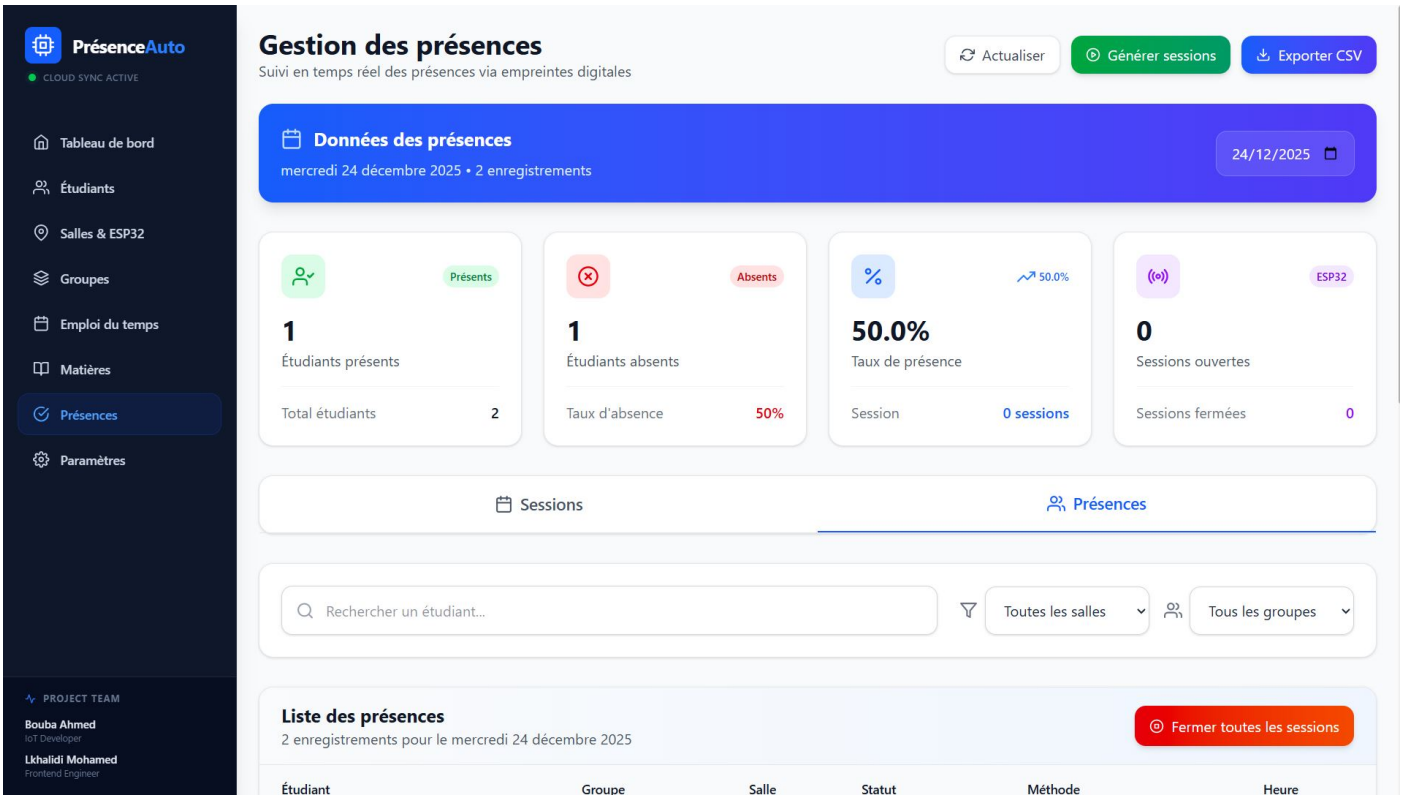

FIGURE 7.8 – Liste des présences en temps réel

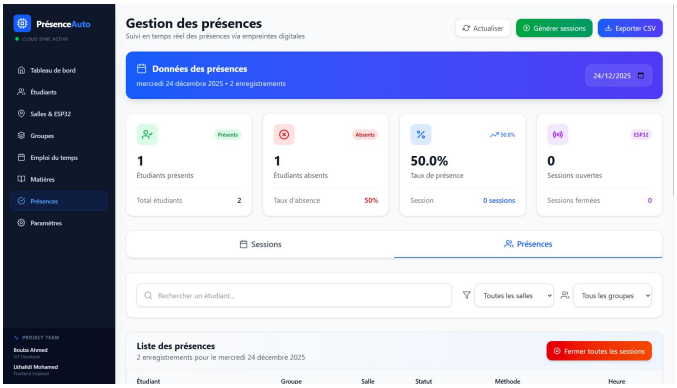

FIGURE 7.9 – Liste des présences en temps réel

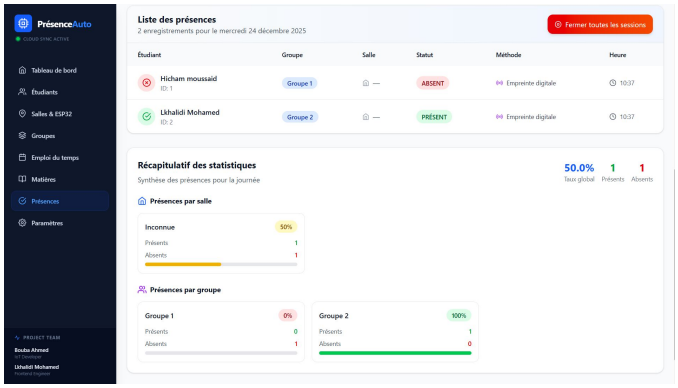

FIGURE 7.10 – Liste des présences en temps réel

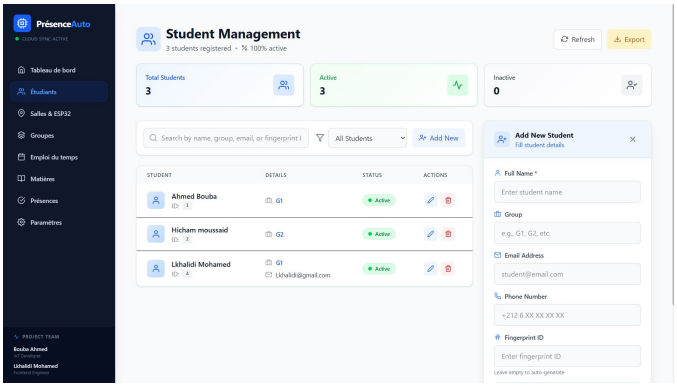

FIGURE 7.11 – Liste des étudiants en temps réel

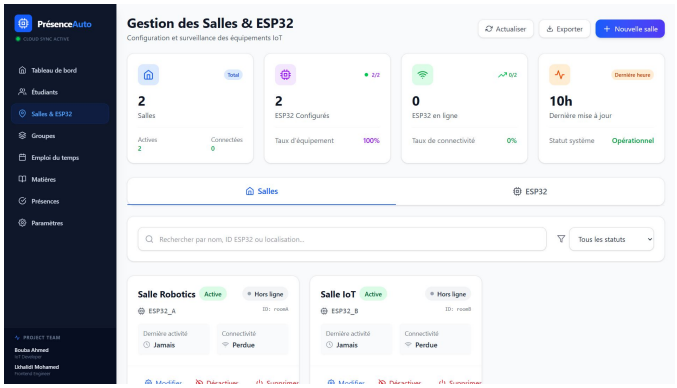

FIGURE 7.12 – Gestion de classe en temps réel

7.3. Structure Firebase

FIGURE 7.13 – Organisation des données dans Firebase Console

7.4. Description technique des composants

7.4.1. Capteur d'empreintes digitales AS608

- **Type** : Capteur optique d'empreintes digitales
- **Interface** : UART série (TX, RX, VCC, GND)
- **Tension d'alimentation** : 3.3V à 5V DC
- **Courant de fonctionnement** : 85mA typique
- **Capacité de stockage** : Jusqu'à 1000 empreintes
- **Temps de reconnaissance** : < 1 seconde
- **Taux de précision** : 99.6%
- **Dimensions** : 56 × 20 × 21.5 mm

7.4.2. ESP32 DevKit V1

- **Microcontrôleur** : ESP32-WROOM-32
- **CPU** : Dual-core à 240 MHz
- **WiFi** : 802.11 b/g/n
- **Flash** : 4 MB
- **GPIOs** : 34 pins disponibles
- **Interfaces** : UART, SPI, I2C, ADC
- **Tension d'alimentation** : 3.3V

7.4.3. Écran LCD 16x2 avec I2C

- **Affichage** : 16 caractères × 2 lignes
- **Rétro-éclairage** : LED bleue réglable
- **Tension** : 3.3V
- **Contraste** : Réglable via potentiomètre

7.4.4. Buzzer passif

- **Type** : Buzzer électromagnétique
- **Tension** : 3-5V
- **Fréquence** : 2-4 kHz
- **Usage** : Retours sonores (succès/échec)

7.4.5. Alimentation

- **Source** : Alimentation PC 5V/2A
- **Connectique** : USB vers borniers
- **Stabilité** : Régulateur intégré pour protection

8- Discussion et perspectives

8.1. Avantages du système proposé

- **Gain de temps significatif** : La prise de présence automatique réduit le temps de gestion de 5-10 minutes à quelques secondes par séance
- **Fiabilité accrue** : Élimination des fraudes (signatures pour autrui) grâce à la biométrie
- **Traçabilité complète** : Historique horodaté précis de chaque présence
- **Interface centralisée** : Toutes les données accessibles via un dashboard web unique
- **Faible coût de déploiement** : Composants électroniques abordables et logiciels open source

8.2. Limites identifiées

- **Dépendance Internet** : Nécessite une connexion stable pour la synchronisation Firebase
- **Limite du capteur** : Capacité maximale de 127 empreintes (suffisant pour une salle)
- **Installation matérielle** : Nécessite une alimentation électrique stable en salle
- **Interface web** : Requiert un navigateur récent et une formation minimale pour les utilisateurs

8.3. Perspectives d'amélioration

- **Mode hors-ligne** : Stockage local sur ESP32 avec synchronisation différée
- **Multi-biométrie** : Ajout de reconnaissance faciale pour plus de sécurité
- **Application mobile** : Développement d'une app pour notifications en temps réel
- **Intégration LMS** : Connexion avec Moodle ou autres plateformes éducatives
- **Analyse prédictive** : Détection des tendances d'absentéisme par IA

9- Conclusion générale

Ce projet a permis de développer un système fonctionnel de gestion automatique de présence utilisant les technologies IoT et biométriques. Les objectifs initiaux ont été atteints : automatisation du processus de prise de présence, centralisation des données dans le cloud, et visualisation en temps réel via une interface web interactive. Le système démontre qu'il est possible de moderniser

les pratiques académiques traditionnelles grâce aux technologies actuelles. La combinaison ESP32, capteur AS608 et Firebase offre une solution robuste et économique. Bien que certaines limites techniques persistent (dépendance Internet, capacité du capteur), la solution proposée constitue une base solide pour une implémentation réelle dans un contexte éducatif. Les compétences acquises

durant ce projet sont variées et précieuses : programmation embarquée en C++ pour l'ESP32, développement web avec React et Flask, conception de bases de données NoSQL avec Firebase, et intégration hardware/software. Ces compétences sont transférables à de nombreux autres domaines de l'informatique et des systèmes intelligents. En perspective, ce projet pourrait être étendu à plusieurs

salles simultanément, intégrer d'autres modalités biométriques, ou être connecté aux systèmes de gestion académique existants. Il représente une première étape prometteuse vers la digitalisation complète des processus éducatifs.

A- Annexes

A.1. Schémas de montage électronique

A.1.1. Branchement des composants

Le tableau ci-dessous résume les connexions entre l'ESP32 et les différents composants :

Composant	Pin/Connecteur	ESP32	Description
Capteur AS608	VCC(Rouge) GND(Vert) TX (noire) RX (jaune)	3.3V GND GPIO 16 (RX2) GPIO 17 (TX2)	Alimentation 3.3V Masse Données du capteur vers ESP32 Commande ESP32 vers capteur
Écran LCD 16x2	VSS VDD VO RS RW E D4 D5 D6 D7 A K	GND 3.3V Potentiomètre central GPIO 21 GND GPIO 22 GPIO 18 GPIO 19 GPIO 23 GPIO 5 3.3V GND	Masse de l'écran Alimentation Contraste Commande registre Mode écriture Enable Données LCD Données LCD Données LCD Données LCD Rétroéclairage + Rétroéclairage -
Buzzer	+ (signal) - (masse)	GPIO 27 GND	Signal sonore (PWM) Masse du buzzer
Potentiomètre 10K	Patte 1 Patte 2 (central) Patte 3	3.3V VO LCD GND	Alimentation Contraste LCD Masse
ESP32 DevKit V1	USB EN	Alimentation PC -	Alimentation 5V/programmation Reset (optionnel)

TABLE A.1 – Tableau corrigé des connexions électroniques selon le câblage réel

Notes sur les connexions

- ◇ Le capteur AS608 fonctionne en 3.3V pour sécurité.
- ◇ L'écran LCD reçoit 3.3V pour l'alimentation principale, D7 peut être utilisé pour backlight selon module.
- ◇ Le potentiomètre règle le contraste du LCD.
- ◇ Le buzzer est connecté avec résistance de 100 pour limiter le courant.
- ◇ Tous les GND sont reliés pour former une masse commune.

A.2. Fichiers sources du projet

Le projet comprend plusieurs fichiers sources qui sont fournis avec ce rapport. Ces fichiers sont organisés comme suit :

A.2.1. Fichiers ESP32 (Arduino)

Deux programmes principaux ont été développés pour l'ESP32 :

- ◇ **enregistrement_empreintes.ino** : Programme dédié à l'enregistrement des nouvelles empreintes digitales des étudiants dans le capteur AS608. Ce code permet d'ajouter, supprimer et lister les empreintes stockées dans le capteur.
- ◇ **gestion_presence.ino** : Programme principal pour le fonctionnement global du système de prise de présence. Il gère la reconnaissance des empreintes, la communication avec Firebase, l'affichage sur l'écran LCD et les notifications sonores via le buzzer.

Ces deux fichiers sont accompagnés de fichiers de configuration et de bibliothèques nécessaires, tous joints à ce rapport.

A.2.2. Dashboard React et Backend Flask

Le code source complet du dashboard React et du backend Flask est disponible sur un dépôt GitHub dédié au projet. Le dépôt contient :

- ◇ **/frontend** : Application React complète avec tous les composants du dashboard
- ◇ **/backend** : API Flask avec tous les endpoints et services
- ◇ **/database** : Scripts de configuration Firebase et règles de sécurité
- ◇ **README.md** : Documentation technique et guides d'installation

Lien vers le dépôt GitHub : `https://github.com/BoubaAhmed/iot-attendance-system`

A.3. Manuel d'utilisation

A.3.1. Procédure d'enregistrement d'une empreinte

1. Lancer le programme `enregistrement_empreintes.ino` sur l'ESP32
2. Suivre les instructions affichées sur le moniteur série ou l'écran LCD
3. Placer le doigt de l'étudiant sur le capteur à deux reprises pour l'enregistrement
4. Noter l'ID d'empreinte attribué et l'associer à l'étudiant dans la base de données

A.3.2. Procédure de prise de présence

1. Le système doit fonctionner avec le programme `gestion_presence.ino`
2. À l'arrivée, l'étudiant pose son doigt sur le capteur
3. En cas de reconnaissance réussie, un message de confirmation s'affiche et un bip est émis
4. La présence est enregistrée en temps réel dans Firebase et visible sur le dashboard

Remerciements et Informations Générales

Ce projet a été réalisé dans le cadre du programme de **Master Systèmes intelligents pour l'éducation** au sein de **l'École Normale Supérieure (ENS) de Meknès**.

Intitulé du projet :
Système de gestion de présence automatisée basé sur l'Internet des Objets (IoT)

Réalisé par :
Bouba Ahmed

Encadré par :
Pr. Omari Slimane

Vue générale du projet :

Ce projet consiste à concevoir et développer un système intelligent de gestion de présence reposant sur les technologies IoT. Le système utilise un microcontrôleur **ESP32** connecté à un capteur biométrique d'empreintes digitales pour l'identification des étudiants, un écran **LCD** pour l'affichage local, ainsi qu'un backend basé sur **Flask**. Les données de présence sont stockées et synchronisées en temps réel via **Firebase**, tandis qu'un tableau de bord web développé avec **React.js** permet la visualisation, l'analyse et la gestion des présences.

Remerciements :

Nous tenons à exprimer nos sincères remerciements à notre encadrant pédagogique, **Pr. Omari Slimane**, pour son accompagnement, ses conseils précieux et son suivi rigoureux tout au long de la réalisation de ce projet. Nous adressons également nos remerciements à **M. El Arbi Abdellaoui**

Alaoui, coordinateur du Master, pour son soutien, son encadrement académique et sa contribution à la réussite de ce parcours de formation.

Année universitaire : 2025 – 2026