Université Ibn Zohr
Faculté des sciences
Centre d'Excellence
- Agadir -

Master d'excellence
Analytique des Données et Intelligence Artificielle
&
Ingénierie Informatique et Systèmes Embarqués

Module :

Bracelet connecté

Réalisé par :
CHAOU Abdelhamid
Yasser Daoud
Maichni hajar
Atigui yassine
EL AIMARI Youssef
Sarma fatima zahra

Encadrer par :
Prof. Rghioui
Prof. Boughrous

Année Universitaire : **2025-2026**

TABLE DES MATIÈRES

TABLE DES FIGURES

CHAPITRE 1

INTRODUCTION

Dans un monde où les technologies intelligentes occupent une place de plus en plus importante, les objets connectés (IoT) jouent aujourd'hui un rôle central dans l'amélioration du bien-être, de la sécurité et de la qualité de vie. Les dispositifs portables, en particulier les bracelets connectés, connaissent une croissance significative grâce à leur capacité à collecter, analyser et transmettre des données en temps réel. Ces outils deviennent indispensables dans plusieurs domaines, notamment la santé, le sport, la surveillance et l'assistance personnalisée.

Le projet Bracelet Connecté s'inscrit dans cette dynamique d'innovation. Il vise à concevoir et développer un dispositif intelligent capable de mesurer et d'envoyer différentes données vers une application mobile dédiée, tout en s'appuyant sur une plateforme backend sécurisée pour le stockage et le traitement. Pour assurer une communication fluide et fiable, nous avons utilisé un bracelet IoT couplé à une application mobile moderne, ainsi qu'un backend implémenté avec FastAPI et déployé sur la plateforme cloud Microsoft Azure.

Ce projet répond à un besoin croissant : disposer d'un système complet, accessible et évolutif permettant le suivi en temps réel d'informations utiles, avec une interface simple d'utilisation et une architecture performante. La réalisation de ce système nous a permis d'explorer plusieurs aspects techniques : conception électronique, communication IoT, développement mobile, création d'API, cloud computing, sécurité et déploiement. Dans ce rapport, nous présentons toutes les étapes de conception et de réalisation du projet, depuis l'analyse du besoin et la formulation de la problématique, jusqu'à la mise en œuvre technique du prototype final. Nous décrivons également les choix technologiques adoptés, les difficultés rencontrées, les solutions proposées ainsi que les perspectives d'amélioration futures.

CHAPITRE 2

PROBLÉMATIQUE

2.1 Problématique

Avec l'évolution rapide des objets connectés, les dispositifs portables tels que les bracelets intelligents jouent un rôle essentiel dans le suivi continu des indicateurs physiologiques. Cependant, la majorité des solutions existantes restent soit coûteuses, soit peu flexibles, ou encore limitées en termes de personnalisation et d'intégration avec des plateformes cloud modernes. Dans ce contexte, se pose la question de la conception d'un bracelet connecté simple, fiable et entièrement personnalisable, capable de mesurer des données physiologiques sensibles en temps réel tout en assurant leur transmission sécurisée vers une application mobile et un environnement cloud.

Le projet vise à concevoir un bracelet basé sur un ESP32, équipé d'un capteur de fréquence cardiaque, d'un capteur de température corporelle et d'un écran OLED permettant l'affichage instantané des mesures. Une difficulté importante réside dans la capacité à garantir une collecte précise et continue de ces données, tout en assurant une consommation énergétique réduite pour optimiser l'autonomie du bracelet.

Un autre enjeu majeur est la transmission des données via Bluetooth Low Energy (BLE) vers l'application mobile. Cette étape nécessite une gestion efficace de la communication sans fil, de la synchronisation des données, et de la stabilité de la connexion, particulièrement dans un contexte mobile où les interruptions sont fréquentes.

Enfin, le stockage et la centralisation des informations dans le cloud soulèvent des défis supplémentaires liés à la fiabilité, à la scalabilité et à la sécurité. L'intégration du backend développé avec FastAPI et déployé sur Azure doit permettre un traitement efficace des données, leur conservation et leur disponibilité pour d'éventuelles analyses futures.

Ainsi, la problématique générale à laquelle répond ce projet peut être formulée comme suit :

Comment concevoir et mettre en œuvre un bracelet connecté capable de mesurer des données physiologiques en temps réel, de les afficher localement, de les transmettre de manière fiable via BLE à une application mobile, puis de les stocker et les gérer de manière sécurisée sur une plateforme cloud ?

Cette problématique regroupe plusieurs défis : l'intégration matérielle des capteurs, la communication IoT, la conception logicielle multiplateforme, la gestion du backend cloud, et l'assurance de la qualité des données transmises.

2.2 Solutions proposée

Pour répondre à la problématique identifiée, la solution envisagée consiste à développer un système complet de suivi physiologique en temps réel, composé de trois éléments principaux : un bracelet IoT intelligent, une application mobile dédiée et une plateforme cloud assurant le traitement et le stockage des données. L'objectif est d'offrir une architecture cohérente, fiable et évolutive couvrant l'ensemble des besoins fonctionnels et techniques identifiés.

1. Bracelet connecté basé sur ESP32

La première composante de la solution est un bracelet équipé d'un ESP32, choisi pour sa faible consommation énergétique, son support natif de Bluetooth Low Energy (BLE) et sa capacité à gérer plusieurs capteurs simultanément. Le bracelet intègre :

— un capteur de fréquence cardiaque pour la mesure des battements du cœur,
— un capteur de température corporelle,
— un écran OLED permettant l'affichage instantané des données collectées,
— un module d'alimentation optimisé pour maximiser l'autonomie.

Le rôle du bracelet est de capturer, traiter localement si nécessaire, puis envoyer les données vers l'application mobile. L'ESP32 agit également comme serveur BLE, permettant une communication fluide et peu énergivore avec le smartphone.

2. Application mobile avec communication BLE

La seconde composante est une application mobile assurant la réception des données issues du bracelet via BLE. Elle permet :

— la synchronisation en temps réel des données du bracelet,
— leur visualisation sous forme de graphiques ou valeurs instantanées,
— l'envoi automatique des mesures vers le backend cloud,
— la gestion de l'utilisateur et des sessions de suivi.

L'application joue un rôle essentiel d'intermédiaire entre l'IoT et le cloud, garantissant une transmission fiable même en cas de variations de connectivité.

3. Backend FastAPI déployé sur Azure

La dernière partie de la solution est une infrastructure cloud basée sur :

— une API REST développée en FastAPI,
— une base de données hébergée sur Azure,
— un déploiement sur les services cloud d'Azure pour assurer disponibilité, scalabilité et sécurité.

Le cloud permet de centraliser les données physiologiques, de les stocker durablement, et de les rendre accessibles pour un suivi à long terme ou pour des traitements plus avancés.

4. Avantages de la solution

— Architecture modulaire (IoT + Mobile + Cloud) permettant une maintenance et une évolution aisées.
— Transmission fiable et en temps réel grâce à BLE.
— Sécurité renforcée via Azure et les mécanismes d'authentification fournis par FastAPI.
— Possibilité d'ajouter de nouveaux capteurs ou fonctionnalités à long terme.

CHAPITRE 3

CONCEPTION

3.1 Introduction

La phase de conception constitue une étape essentielle dans la réalisation du projet Bracelet Connecté. Elle permet de transformer les besoins identifiés lors de l'analyse en une architecture technique cohérente, détaillée et capable d'être implémentée efficacement. Cette étape vise à définir l'organisation globale du système, les interactions entre ses différentes composantes, ainsi que les choix technologiques nécessaires pour assurer sa performance, sa fiabilité et son évolutivité.

Dans notre projet, la conception doit prendre en compte plusieurs contraintes spécifiques : la gestion des capteurs connectés au microcontrôleur ESP32, la communication Bluetooth Low Energy (BLE) avec l'application mobile, la transmission sécurisée vers le backend FastAPI, ainsi que l'intégration avec les services cloud d'Azure pour l'hébergement et le stockage des données. Une attention particulière est également portée à l'optimisation énergétique, à la sécurité des données physiologiques et à la fluidité des échanges en temps réel.

3.2 Architecture générale du système

L'architecture globale adoptée pour le projet repose sur un modèle en trois couches :

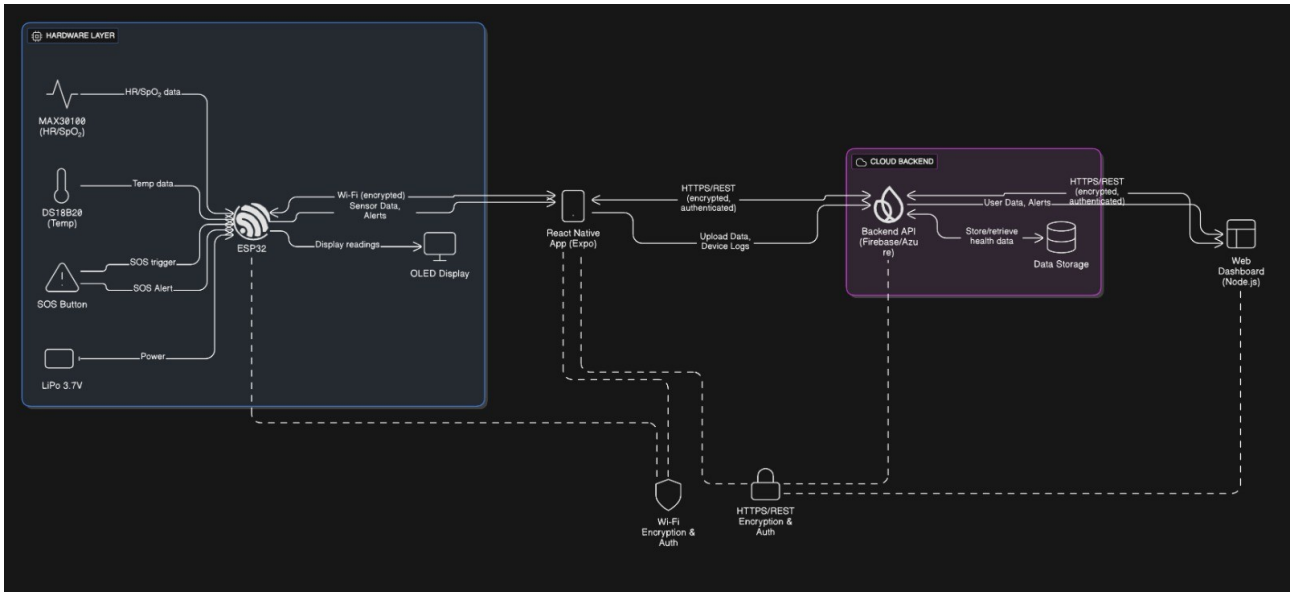


FIGURE 3.1 – l'architecture générale

Cette architecture illustre un système IoT complet de télésurveillance médicale structuré en quatre étapes clés, allant de la captation physique à la visualisation web. Au cœur du dispositif matériel, un microcontrôleur ESP32, alimenté par une batterie LiPo, collecte les constantes vitales via des capteurs spécifiques (MAX30100 pour le rythme cardiaque et la SpO, DS18B20 pour la température) et gère les alertes d'un bouton SOS tout en affichant les données localement sur un écran OLED. Ces informations sont transmises via Bluetooth Low Energy (BLE) vers une application mobile développée sous React Native, qui sert de passerelle intermédiaire. Pour assurer la sécurité des données sensibles, l'application relaie ensuite les informations de manière chiffrée et authentifiée via le protocole HTTPS/REST vers une infrastructure Cloud Backend (type Firebase ou Azure), où une API se charge de stocker l'historique dans une base de données dédiée.

3.3 Diagramme de cas d'utilisation générale

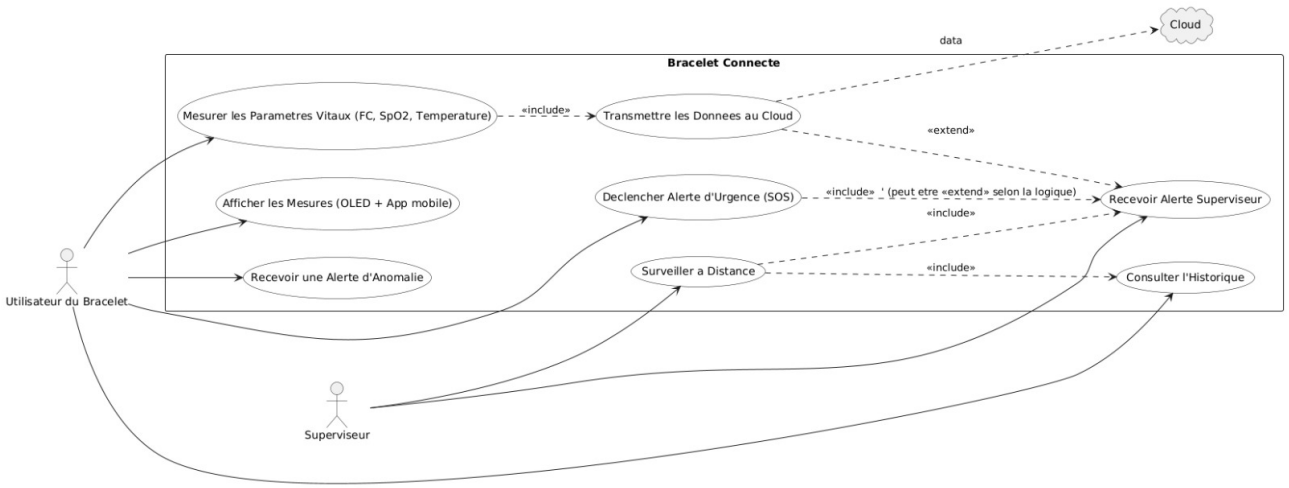


FIGURE 3.2 – Diagramme de cas d'utilisation générale

Ce diagramme modélise les interactions fonctionnelles du système "Bracelet Connecté", orchestrées autour de deux acteurs principaux : l'utilisateur du bracelet et un superviseur distant. L'utilisateur interagit avec le dispositif pour mesurer ses paramètres vitaux (fréquence cardiaque, SpO, température), un processus qui inclut systématiquement la transmission des données vers le Cloud et permet la consultation des résultats via un écran OLED ou une application mobile. En termes de sécurité, l'utilisateur peut déclencher manuellement une alerte SOS ou recevoir des notifications en cas d'anomalie détectée ; ces événements, tout comme la transmission de données critiques, entraînent l'envoi d'une alerte vers le superviseur. Ce dernier assure une surveillance à distance complète, ce qui englobe la réception des alertes de sécurité ainsi que la consultation de l'historique des données de santé enregistrées.

3.4 Conception de l'Application Mobile

L'application est un compagnon pour un bracelet intelligent qui permet de suivre en temps réel les données de santé de l'utilisateur. Elle sert d'interface principale pour visualiser les métriques vitales, gérer la connexion avec le dispositif via **Bluetooth Low Energy (BLE)**, et analyser l'historique des données de santé.

L'objectif est de fournir une interface intuitive et réactive ("Premium Design") pour surveiller la santé (cœur, sommeil, activité, température, oxygène) et alerter l'utilisateur en cas d'anomalie.

3.4.1 Fonctionnalités Principales

Authentification & Utilisateur

— **Inscription (Sign Up)** : Création de compte avec email et mot de passe.

— **Connexion (Sign In)** : Accès sécurisé au compte.
— **Profil Utilisateur** : Gestion des informations personnelles et des objectifs de santé.

3.4.2 Fonctionnalités Critiques

Affichage Temps Réel (Real-time Monitoring)

— **Tableau de Bord** : Affichage instantané des métriques vitales avec rafraîchissement automatique.
— **Animation** : Cœur pulsant synchronisé avec les BPM pour un feedback visuel immédiat.
— **Indicateurs** : Codes couleurs (Vert/Orange/Rouge) pour indiquer l'état de chaque métrique.

Système d'Alertes

— **Détection d'Anomalies** : Modèle ML pour détecter :
  — **Bradycardie / Tachycardie** (BPM < 60 ou > 100).
  — **Hypoxie** ($SpO_2 < 95\%$).
  — **Fièvre** (Temp $> 37.5°C$).
— **Notifications** : Alertes visuelles (Bannières, Badges) et haptiques (Vibrations) pour avertir l'utilisateur immédiatement.

Historique et Analyse

— **Stockage Local** : Sauvegarde des données même hors ligne.
— **Graphiques Interactifs** : Visualisation de l'évolution sur 24h, 7 jours ou 30 jours.
— **Tendances** : Indicateurs de progression (Amélioration/Détérioration) par rapport à la période précédente.

Gestion du Dispositif (Device Management)

— **Connexion Bluetooth** : Scan et appairage avec le bracelet intelligent.
— **État de connexion** : Indicateur visuel (connecté/déconnecté) et gestion des permissions (Bluetooth/Localisation).
— **Synchronisation** : Récupération des données en temps réel et synchronisation hors ligne (*Offline Storage*).

3.4.3 Flux Utilisateur (User Flows)

Flux de Connexion (Connection Flow)

1. **Lancement** : L'utilisateur ouvre l'application.
2. **Vérification** : L'app vérifie l'état du Bluetooth et les permissions.
3. **Scan** : Si pas de dispositif connu, l'utilisateur ouvre le modal "Connect Device".
4. **Sélection** : L'utilisateur choisit son bracelet dans la liste des appareils détectés.

5. **Appairage** : L'app établit la connexion BLE sécurisée.
6. **Succès** : Le statut passe au vert, le cœur commence à battre, les données s'affichent.

Flux de Configuration (Configuration Flow)

1. **Accès** : L'utilisateur navigue vers l'onglet "Profil" ou "Settings".
2. **Modification** : L'utilisateur change ses objectifs (ex : nombre de pas) ou ses informations personnelles.
3. **Validation** : L'utilisateur sauvegarde les changements.
4. **Feedback** : Confirmation visuelle ("Profil mis à jour").
5. **Impact** : Les seuils d'alerte et les graphiques s'adaptent aux nouveaux paramètres.

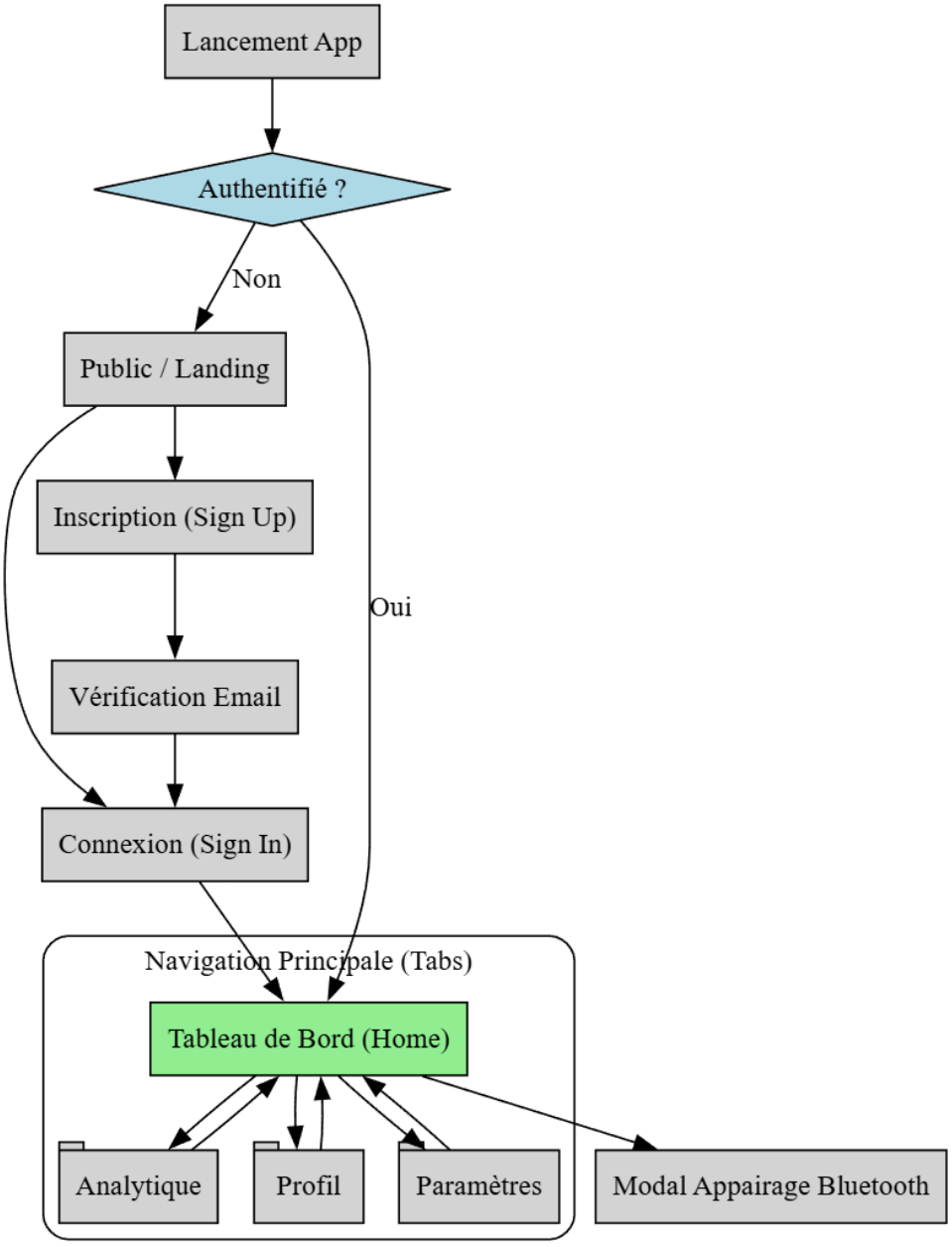


FIGURE 3.3 – Diagramme d'État/d'Activité

3.4.4 Navigation (Sitemap)

L'application utilise une navigation hybride : une partie **publique** pour l'authentification et une partie **privée** avec une barre d'onglets (*Tab Bar*) pour l'application principale.

3.5 Conception Backend FastAPI

Le backend du projet "Bracelet Connecté" est développé sur le framework **FastAPI** (Python 3.x), choisi pour ses performances exceptionnelles, notamment grâce à sa gestion asynchrone native via `async/await`, ce qui est crucial pour gérer le flux constant et les pics de charge des données de capteurs IoT. Sa conception, basée sur les standards Python modernes (*type hints*), permet non seulement une simplicité de développement mais aussi une validation de schéma et une documentation automatique rigoureuse.

L'architecture est de type **RESTful**, utilisant le préfixe `/api/v1` pour l'ensemble des routes afin de garantir une gestion des versions future.

Structure de l'API

L'API est organisée autour de modules fonctionnels principaux, chacun gérant un domaine métier spécifique et utilisant le design pattern MVC (Modèle-Vue-Contrôleur) implicite à FastAPI et SQLAlchemy.

1. **Auth** : Gestion de l'authentification et de l'autorisation (jetons JWT).
2. **Users** : Gestion des profils utilisateurs (enregistrement, mise à jour, vérification, réinitialisation de mot de passe).
3. **Devices** : Enregistrement, gestion (CRUD) et consultation des bracelets connectés.
4. **Metrics** : Réception et stockage des données de signes vitaux en temps réel.
5. **Issues** : Suivi et gestion (CRUD) des incidents et des alertes.
6. **Summary** : Fourniture de données agrégées et brutes filtrées pour les tableaux de bord de l'application mobile.

Endpoints Clés

L'API utilise les méthodes HTTP standards (GET, POST, PUT, DELETE) pour les opérations CRUD (Create, Read, Update, Delete). La sécurité est gérée par un double mécanisme : **OAuth2** (Password Flow) pour les utilisateurs finaux et une **API Key** dédiée pour la soumission des données par les appareils (bracelet).

Sécurité et Authentification

Le système repose sur un modèle de sécurité à deux niveaux pour séparer l'accès de l'utilisateur de celui du périphérique IoT :

1. **Utilisateurs (OAuth2)** : L'application mobile interagit via le standard OAuth2 (Flow Password). Le serveur génère des jetons **JWT** (JSON Web Tokens) après une connexion

M.HTTP	Endpoint	Description	Sécurité
Authentification (Auth)			
POST	/token	Génération du jeton JWT (Login).	Public
Gestion des Utilisateurs (Users)			
POST	/users	Création d'un nouveau compte (Register).	Public
GET	/users/me	Récupération du profil de l'utilisateur connecté.	OAuth2
GET	/users/{user_id}	Récupération par ID.	OAuth2
PUT	/users/{user_id}	Mise à jour du profil.	OAuth2
DELETE	/users/{user_id}	Suppression logique (Soft Delete).	OAuth2
POST	/users/ verify-email	Vérification de l'email avec un code.	Public
POST	/users/ reset-password	Réinitialisation du mot de passe avec un code.	Public
Gestion des Appareils (Devices)			
POST	/devices/register	Enregistrement d'un nouveau bracelet.	OAuth2
GET	/devices/	Liste des appareils de l'utilisateur.	OAuth2
PUT	/devices/ {device_id	Mise à jour d'un appareil.	OAuth2
DELETE	/devices/ {device_id	suppression d'un appareil.	OAuth2
Métriques (Metrics)			
POST	/metrics/batch/	**Point critique** : Insertion en masse des signes vitaux.	API Key
GET	/devices/ {device_id}/metrics	Récupération des données brutes pour un appareil.	OAuth2
Synthèse des Données (Summary)			
GET	/users/{user_id} /metrics/summary	Données agrégées (jour, semaine, mois).	OAuth2
GET	/users/{user_id} /metrics/data	Données brutes filtrées par type de métrique.	OAuth2
Incidents (Issues)			
POST	/issues/	Création d'une nouvelle alerte/incident (e.g., chute, anomalie).	OAuth2
PUT	/issues/{issue_id}	Mise à jour du statut d'une alerte (e.g., résolue).	OAuth2
GET	/issues/	Consultation des alertes.	OAuth2

TABLE 3.1 – Liste détaillée des Endpoints principaux de l'API

réussie. Ces jetons sont requis dans l'en-tête `Authorization` de toutes les requêtes protégées pour accéder aux données personnelles (profil, historique des métriques, alertes).

2. **Appareils (API Key)** : Le bracelet lui-même est authentifié par un mécanisme simple de **Clé API (X-API-KEY)** transmise dans l'en-tête de la requête. Ce mécanisme est optimisé pour les microcontrôleurs et assure que seul l'appareil enregistré peut soumettre des données au point critique `/metrics/batch/`.

Pour garantir la sécurité des comptes utilisateurs, les mots de passe sont hachés de manière irréversible à l'aide de l'algorithme **bcrypt** avant d'être stockés en base de données.

Validation avec Pydantic

Pydantic est utilisé de manière intensive pour garantir l'intégrité et la conformité des données échangées, en se basant sur la validation de schéma :

— **Validation des requêtes (Input)** : Les modèles Pydantic, tels que `UserCreate` ou le crucial `MetricBatch` (qui gère l'insertion massive de points de mesure), définissent la structure et le type de données attendues. FastAPI rejette automatiquement toute requête non conforme (e.g., format d'email invalide, valeur de métrique manquante) avec une erreur 422, assurant ainsi l'intégrité des données avant le traitement métier.
— **Sérialisation des réponses (Output)** : Pydantic est également utilisé pour structurer les données envoyées au client (e.g., le modèle `User` sans le hachage du mot de passe), garantissant que seules les informations autorisées sont exposées.
— **Gestion de la configuration** : Pydantic gère également la configuration de l'application (variables d'environnement, clés secrètes) de manière sécurisée et typée.

Schéma de la Base de Données

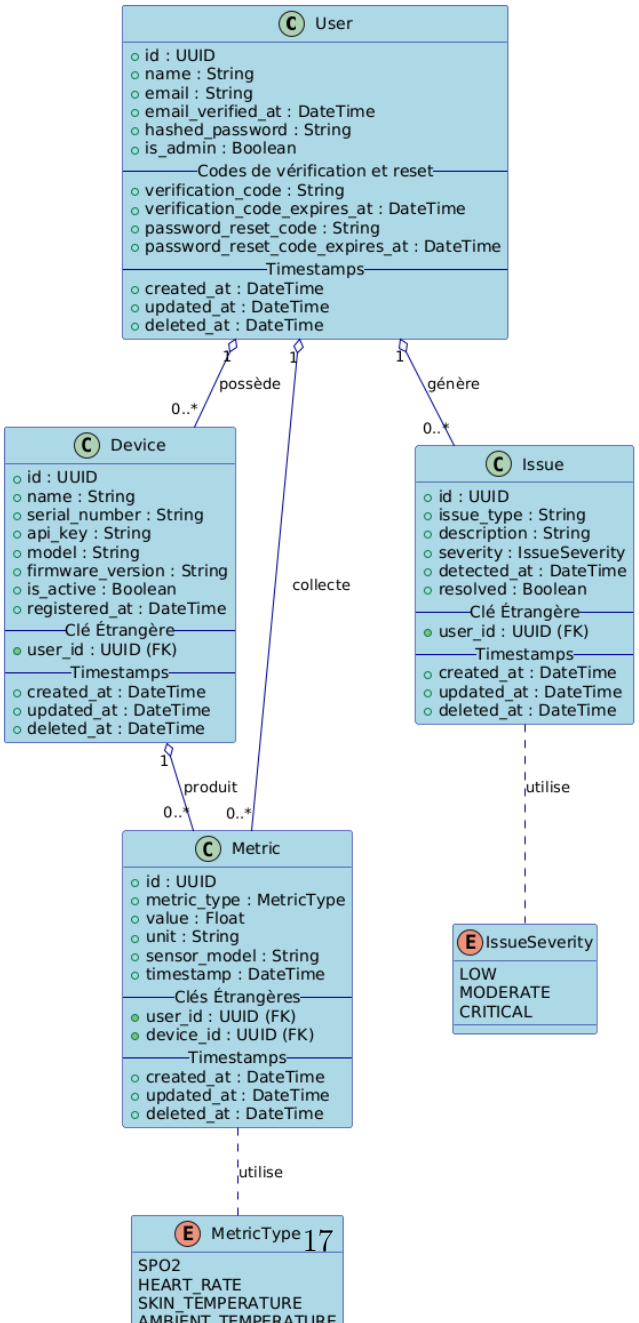
La persistance des données est assurée par une base de données relationnelle **PostgreSQL** (Azure SQL), interfacée par l'ORM **SQLAlchemy** pour une gestion simplifiée et sécurisée des interactions (connexions, requêtes SQL).

Le schéma repose sur les quatre modèles clés suivants, structurant les relations (schéma entité-association) :

Cette structure relationnelle garantit que chaque donnée de signe vital (*Metric*) est liée à un appareil spécifique (*Device*) et à l'utilisateur propriétaire (*User*), permettant un historique précis, des requêtes optimisées pour l'agrégation (*Summary*) et une récupération efficace des données par utilisateur.

Modèle	Description	Relations Clés
User	Informations d'identification et de profil, statut de vérification (email, mot de passe haché).	Un-à-plusieurs (avec Device, Metric, Issue)
Device	Informations matérielles du bracelet (numéro de série, version du firmware, clé API utilisée pour la soumission des données).	Plusieurs-à-un avec User, Un-à-plusieurs avec Metric
Metric	**Table de séries chronologiques**. Stocke chaque point de mesure (`spo2`, `heart_rate`, `skin_temperature`, `ambient_temperature`) avec son horodatage précis. C'est la table à plus haut volume.	Plusieurs-à-un avec User et Device
Issue	Enregistre les alertes et incidents détectés (type, gravité : `low`, `moderate`, `critical`, statut de résolution).	Plusieurs-à-un avec User

TABLE 3.2 – Modèles Clés du Schéma de Base de Données



3.6 conception de la partie IoT

3.6.1 Composants matériels utilisés

Le bracelet connecté est constitué des éléments suivants :

— **ESP32-C3 Super Mini** : microcontrôleur principal assurant la gestion des capteurs et la communication BLE ;
— **MAX30102** : capteur de fréquence cardiaque et SpO2 ;
— **MPU6050** : accéléromètre et gyroscope pour la détection de mouvement ;
— **MLX90614** : capteur de température corporelle infrarouge sans contact ;
— **Écran OLED 0.96"** : affichage instantané des données collectées ;
— **Batterie Li-Po 3.7V** : alimentation du bracelet.

3.6.2 Choix du microcontrôleur

Le Wemos D1 Mini, initialement prévu, ne possédant pas de module Bluetooth, il a été remplacé par l'ESP32-C3 Super Mini. Ce dernier offre plusieurs avantages majeurs :
— Support natif du BLE 5.0 ;
— 13 GPIO disponibles ;
— 6 canaux ADC ;
— Bus I2C flexible ;
— Faible consommation énergétique ;
— Format compact adapté à un bracelet.

3.6.3 Tests préliminaires

Avant l'intégration complète des capteurs, des tests ont été réalisés sur le microcontrôleur :
— Vérification des GPIO via une LED ;
— Tension stable de 3.3V confirmée ;
— Configuration du bus I2C sur GPIO8 (SDA) et GPIO9 (SCL).

3.6.4 Intégration et calibration des capteurs

Capteur MAX30102

Des valeurs instables ont nécessité une calibration :
— Valeur IR optimale : 80 000 – 100 000 ;
— Ajustement du ratio RED/IR ;
— Temps de réponse inférieur à 3 s ;
— Précision de la fréquence cardiaque : ±2 bpm.

Capteur de température MLX90614

Le LM35 ayant montré des limites, il a été remplacé par le MLX90614 :

— Mesure sans contact ;
— Précision de ±0.5°C ;
— Temps de réponse inférieur à 1 s ;
— Adresse I2C : 0x5A.

Accéléromètre/Gyroscope MPU6050

Utilisé pour la détection de mouvement et l'analyse d'activité :
— Adresse I2C : 0x68 ;
— Plage : $\pm 2g$ et ±250°/s ;
— Fréquence d'échantillonnage : 100 Hz.

Écran OLED 0.96"

— Résolution : 128x64 ;
— Interface I2C (0x3C) ;
— Affichage : BPM, SpO2, température, niveau de batterie, statut BLE.

3.6.5 Communication Bluetooth Low Energy (BLE)

Configuration générale

— Service UUID : 6E400001-B5A3-F393-E0A9-E50E24DCCA9E ;
— Caractéristique TX : 6E400003 ;
— Caractéristique RX : 6E400002.

Problème du JSON tronqué

Le BLE limite la charge utile à 20 octets, ce qui tronquait les données JSON. **Solution :** augmentation du MTU à 512 octets permettant une transmission complète et stable des données.

3.6.6 Assemblage final et optimisation énergétique

— Bus I2C partagé entre les capteurs (GPIO8/9) ;
— Mise en veille automatique de l'écran après 10 s ;
— Réduction de la fréquence d'échantillonnage en mode veille ;
— Déconnexion BLE en cas d'inactivité prolongée ;
— Autonomie estimée : 8–12 heures.

3.6.7 Tests de validation

— MAX30102 : précision ±2 bpm ;
— MLX90614 : précision ±0.5°C ;

— Portée BLE : environ 10 m en intérieur ;
— Stabilité : aucune déconnexion pendant 2 heures ;
— Validation complète du flux : capteurs → mobile → backend Azure.

CHAPITRE 4

RÉALISATION

4.1 Introduction

La phase de réalisation constitue l'étape où le projet prend forme concrètement, en transformant les modèles conceptuels en un système fonctionnel. Elle inclut la mise en place du bracelet connecté, le développement de l'application mobile en React Native, ainsi que la construction et le déploiement du backend. Dans cette phase, une attention particulière a été portée à l'architecture serveur : le backend FastAPI a été conteneurisé avec Docker afin de garantir une portabilité optimale, une installation simplifiée et un déploiement cohérent sur Microsoft Azure. L'objectif principal est d'assurer une communication fluide entre le bracelet IoT, l'application mobile et le backend conteneurisé, tout en garantissant un stockage fiable des données dans une base PostgreSQL hébergée sur Azure.

4.2 Technologies et Outils Utilisés

4.2.1 Backend et API

— **FastAPI** : framework Python moderne utilisé pour développer l'API REST permettant la réception, la gestion et la mise à disposition des données physiologiques.

FIGURE 4.1 – FastAPI

— **Python 3.10** : langage principal utilisé pour l'implémentation du backend.

FIGURE 4.2 – Python

— **PostgreSQL** : système de gestion de base de données relationnelle utilisé pour stocker les données envoyées par l'application mobile.

FIGURE 4.3 – PostgreSQL

— **SQLAlchemy** : ORM permettant la communication entre FastAPI et PostgreSQL.

FIGURE 4.4 – SQLAlchemy

— **Docker** : outil de conteneurisation utilisé pour exécuter le backend FastAPI dans un environnement isolé, portable et facilement déployable sur Azure.

FIGURE 4.5 – Docker

4.2.2 Déploiement et Cloud

Dans le cadre de ce projet, plusieurs services Azure ont été déployés afin d'assurer l'hébergement du backend, le stockage des données, la supervision de l'activité et la gestion sécurisée des secrets. L'ensemble des ressources a été regroupé dans un même groupe de ressources pour faciliter la gestion et le suivi du projet.

- **Azure App Service** : service d'hébergement utilisé pour exécuter l'image Docker contenant l'API FastAPI. Ce service assure la disponibilité continue du backend.
- **Azure Database for PostgreSQL (Flexible Server)** : base de données relationnelle managée, utilisée pour stocker les données envoyées par l'application mobile et le bracelet connecté.
- **Azure Container Registry (ACR)** : registre privé permettant de stocker l'image Docker du backend avant son déploiement sur App Service.
- **Azure Storage Account** : espace de stockage utilisé pour conserver certains fichiers générés par l'API ou nécessaires au fonctionnement du système.
- **Azure Key Vault** : service de gestion sécurisée utilisé pour stocker les secrets sensibles tels que les chaînes de connexion à la base PostgreSQL.
- **Azure Application Insights** : outil de monitoring intégré au backend, permettant d'analyser les performances, la charge et les erreurs de l'API.
- **Azure Log Analytics Workspace** : espace de travail pour centraliser les logs et métriques provenant d'App Service et d'Application Insights.
- **Resource Group "cloud"** : ensemble logique regroupant toutes les ressources du projet afin de simplifier leur gestion sur Azure.

Ressources

Récent Favori

Nom	Type	Dernier affichage
stockagebracelet123	Compte de stockage	il y a quelques secondes
DefaultWorkspace-38cbd89a-d842-437f-942e-442a5483d78d-PLC	Espace de travail Log Analytics	il y a quelques secondes
Azure for Students	Abonnement	il y a quelques secondes
cloud	Groupe de ressources	il y a 21 heures
braceletiot	Application Insights	il y a 21 heures
braceletiot	App Service	il y a 21 heures
bracelet-iot	Coffre de clés	il y a 3 jours
braceletbd	Serveur flexible Azure Database pour PostgreSQL	il y a 3 jours
braceletiot	Container registry	il y a une semaine
ASP-cloud-ac10	Plan App Service	il y a 2 semaines

FIGURE 4.6 – Ressources du projet sur Microsoft Azure

4.2.3 Application Mobile

— **React Native** : framework mobile cross-platform utilisé pour développer l'application permettant de recevoir les données via BLE et les envoyer au backend.

FIGURE 4.7 – React Native

— **Bluetooth Low Energy (BLE)** : protocole utilisé pour la communication entre le bracelet ESP32 et l'application mobile.

FIGURE 4.8 – React Native

— **Axios / Fetch API** : bibliothèques utilisées pour communiquer efficacement avec l'API FastAPI.

FIGURE 4.9 – Axios / Fetch API

FIGURE 4.10 – Technologies utilisées pour l'application mobile (image à insérer)

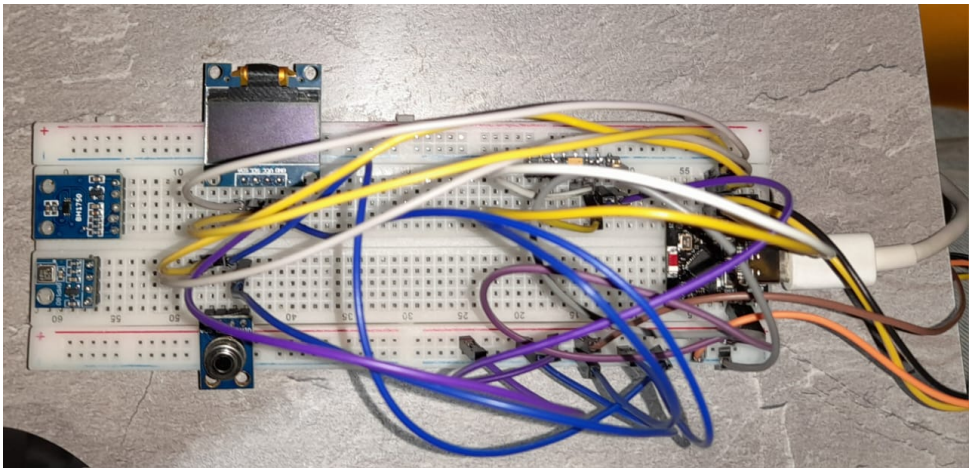
4.3 Réalisation (Implémentation)

4.3.1 Implémentation du Bracelet (Capteurs et Firmware)

La réalisation du bracelet a impliqué l'intégration des capteurs (SpO2, FC, température) avec le microcontrôleur et l'implémentation du protocole de communication Wi-Fi. Le firmware embarqué est chargé de :

1. L'acquisition des données des capteurs à une fréquence définie.
2. Le formatage des données en lots (*batch*) au format JSON.
3. L'authentification auprès du Backend via la **Clé API** dédiée.
4. L'envoi des lots de données à l'endpoint `/metrics/batch/` du serveur cloud Azure.

L'optimisation du code C++ a été essentielle pour garantir une faible consommation d'énergie et la fiabilité de la transmission des données, même en cas de perte temporaire de connexion réseau.



4.3.2 Implémentation de l'Application Mobile

L'application a été développée en [Nommer la technologie, ex : React Native] en se concentrant sur une expérience utilisateur fluide et la visualisation immédiate des données critiques.

Tableaux de Bord et Visualisation

La page d'accueil affiche les indicateurs vitaux en temps réel. La clarté des données est assurée par l'utilisation de graphiques et de codes couleur basés sur les seuils de santé.

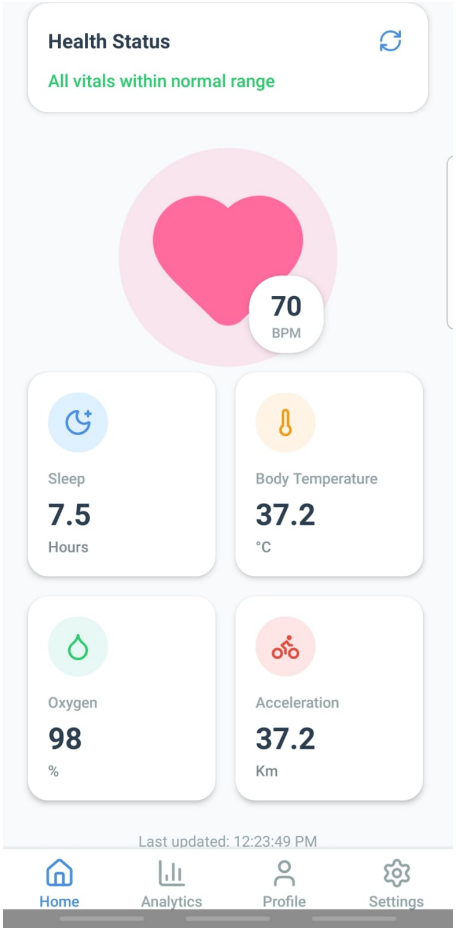


FIGURE 4.11 – Capture de l'écran principal affichant les données vitales en direct.

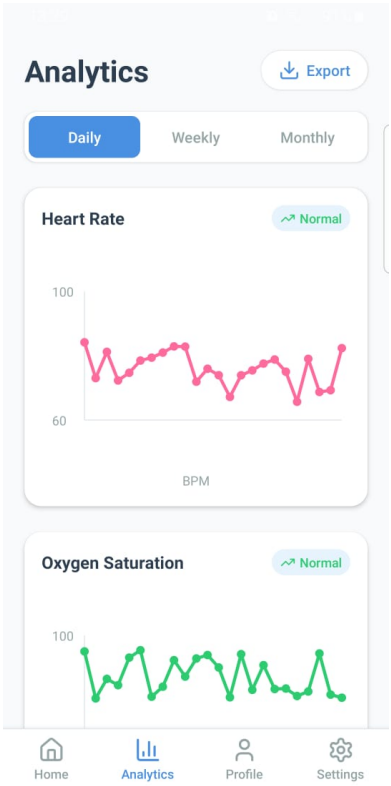


FIGURE 4.12 – Capture de l'écran d'analyse des tendances avec filtres temporels.

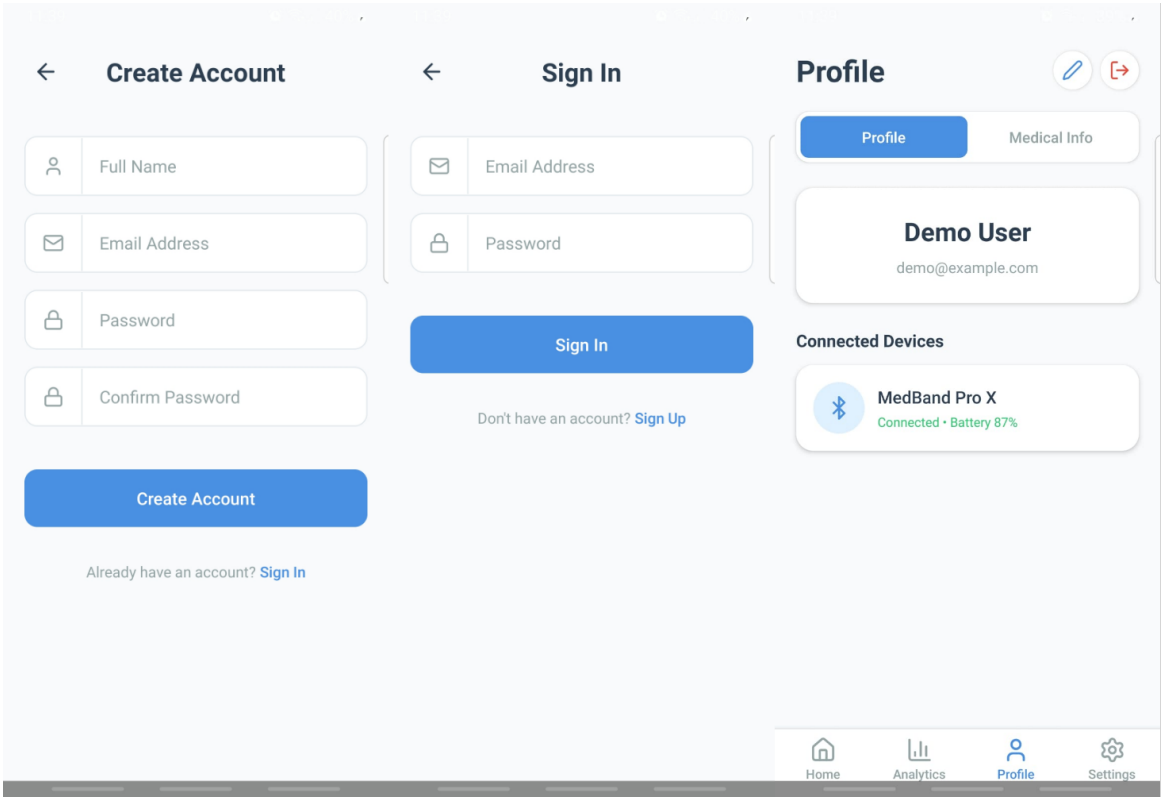


FIGURE 4.13 – Capture de l'écran de sign up ,login , profile

Gestion de la Sécurité et des Alertes

L'application gère de manière transparente la connexion de l'utilisateur via le jeton JWT. Le module d'alerte est constamment à l'écoute des nouveaux incidents créés sur l'API (e.g., une chute détectée par le bracelet ou une anomalie de SpO2), et affiche des notifications push pour les événements critiques.

4.3.3 Implémentation du Backend (FastAPI) et Base de Données

Le Backend a été réalisé en tant que service RESTful haute performance, hébergé dans un conteneur Docker.

Structure RESTful et Documentation

Toutes les routes de l'API sont accessibles sous le préfixe `/api/v1`. L'utilisation de FastAPI a permis de générer automatiquement la documentation interactive (OpenAPI), qui a été utilisée comme source de vérité pour l'intégration avec le firmware du bracelet et l'application mobile.

FIGURE 4.14 – Capture de l'interface démontrant les routes et la validation Pydantic.

Gestion des Données et Performances

L'ORM SQLAlchemy a permis une implémentation robuste des modèles de données. Le mécanisme d'insertion en masse (*bulk insert*) a été validé pour insérer jusqu'à 1000 points de données par seconde, confirmant la capacité du backend à absorber un volume élevé de métriques IoT.

4.3.4 Mise en Œuvre du Déploiement Azure

Le déploiement assure la portabilité et l'évolutivité de l'application via les services Azure, en s'appuyant sur une architecture conteneurisée.

Infrastructure et Services

Le Backend FastAPI est conteneurisé à l'aide de **Docker**. Les images conteneurs sont stockées dans un registre privé **Azure Container Registry (ACR)**, nommé `braceletiot.azurecr.io`. Le service est ensuite hébergé sur **Azure App Service** (ou Azure Container Apps) qui tire l'image depuis ce registre. La base de données relationnelle est fournie par **Azure SQL Database** (PostgreSQL).

Processus de Construction et de Déploiement

Le déploiement des mises à jour de l'application est géré via l'interface en ligne de commande (CLI) de Docker. Ce processus garantit que la dernière version du code est encapsulée et disponible pour le cloud. Les étapes réalisées sont les suivantes :

1. **Construction de l'image (Build) :** Création de l'image Docker locale en utilisant le tag correspondant au registre Azure.

   ```
   docker build -t braceletiot.azurecr.io/mon-api:latest .
   ```

2. **Publication de l'image (Push) :** Transfert de l'image construite vers le registre Azure Container Registry afin qu'elle soit accessible par le service d'hébergement.

   ```
   docker push braceletiot.azurecr.io/mon-api:latest
   ```

Une fois l'image *pushée* avec succès, l'instance Azure App Service est configurée pour récupérer (pull) cette nouvelle version, assurant ainsi la mise à jour de l'application en production.

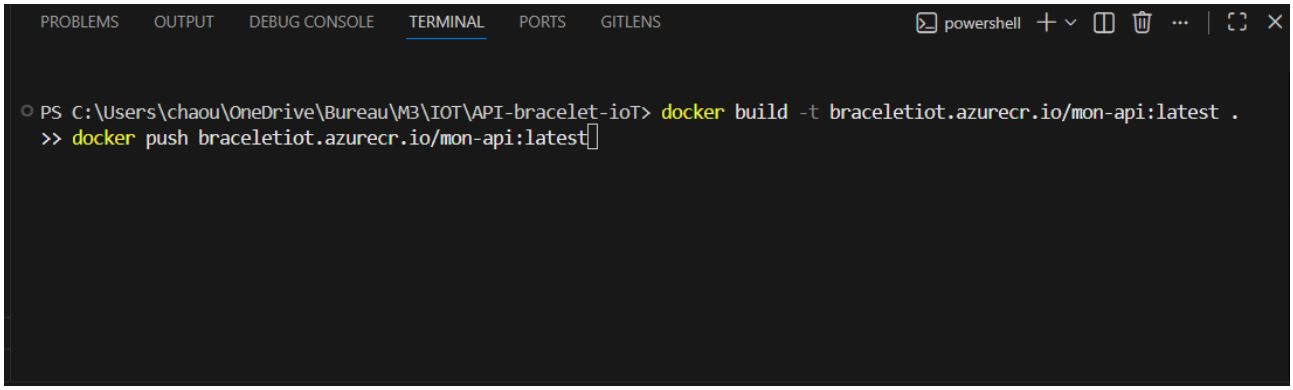


FIGURE 4.15 – Succès de la construction et du transfert de l'image Docker vers Azure ACR.

Configuration et Sécurité de l'Environnement

Une fois l'image disponible dans le registre, le service **Azure App Service** est configuré pour l'exécuter. Cependant, pour des raisons de sécurité, les informations sensibles (comme

l'URL de connexion à la base de données PostgreSQL ou les clés secrètes) ne sont jamais stockées en dur dans le code ou l'image Docker.

Nous utilisons les **Variables d'Environnement** (Application Settings) directement dans le portail Azure. Ces variables sont injectées dynamiquement dans le conteneur au démarrage.

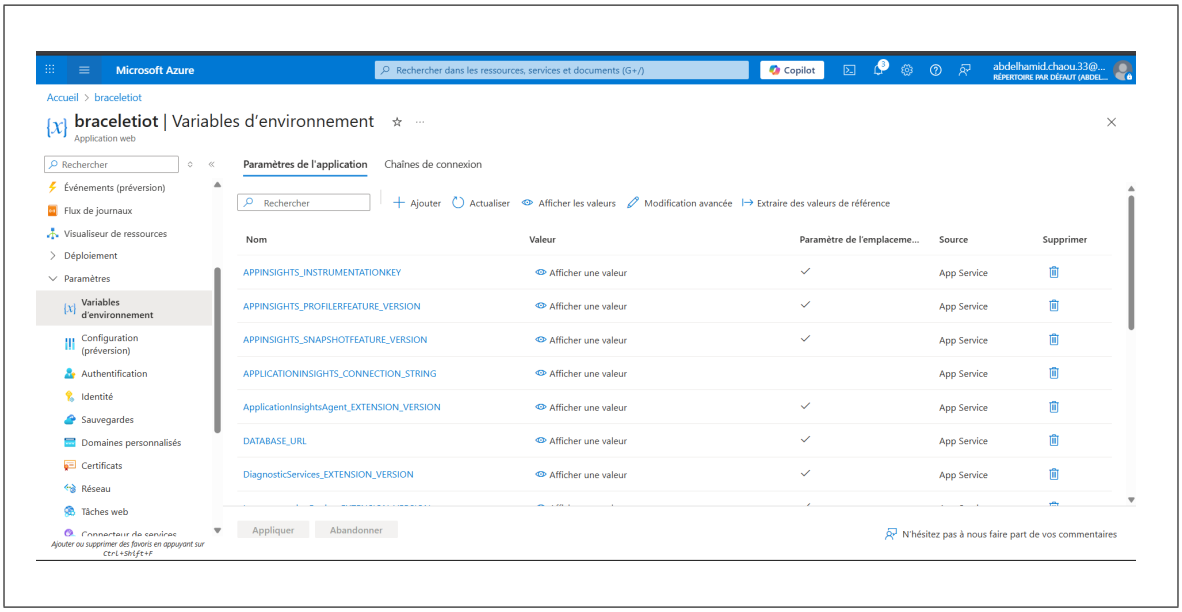


FIGURE 4.16 – Configuration des variables d'environnement (Database URL, API Key) dans Azure App Service pour sécuriser les accès.

Cette approche permet de modifier la configuration (par exemple, changer de base de données) sans avoir à recompiler ou redéployer l'image Docker, respectant ainsi les bonnes pratiques du *Twelve-Factor App*.

Validation du Déploiement et Tests

Après la configuration, le conteneur démarre automatiquement. La validation du déploiement s'effectue en accédant à l'URL publique fournie par Azure (ex : `https://braceletiot.azurewebsite`

L'interface de documentation interactive **Swagger UI** (générée automatiquement par FastAPI) permet de confirmer que l'API est bien en ligne et fonctionnelle.

Surveillance et Monitoring

Pour garantir la maintenabilité de la solution en production, nous avons intégré **Azure Application Insights**. Cet outil nous permet de surveiller :

— La disponibilité de l'application (Live Metrics).
— Les temps de réponse des requêtes HTTP.
— Les journaux d'erreurs (Logs) en cas de dysfonctionnement du conteneur.

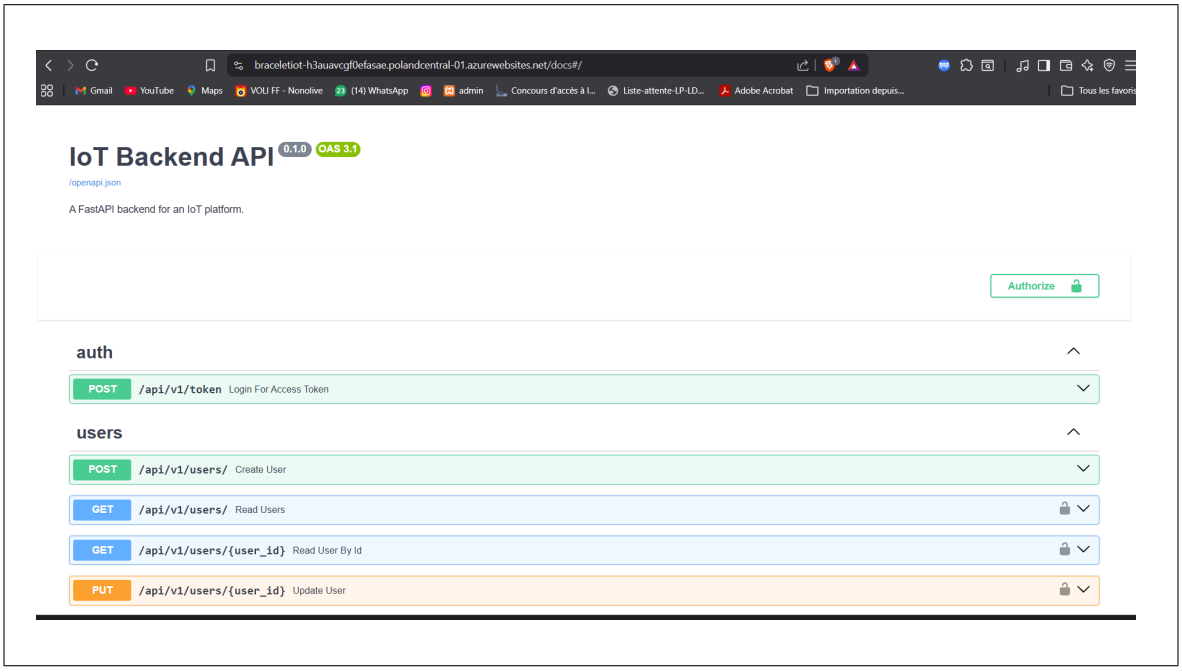


FIGURE 4.17 – Interface Swagger accessible via l'URL publique Azure, prouvant le succès du déploiement.

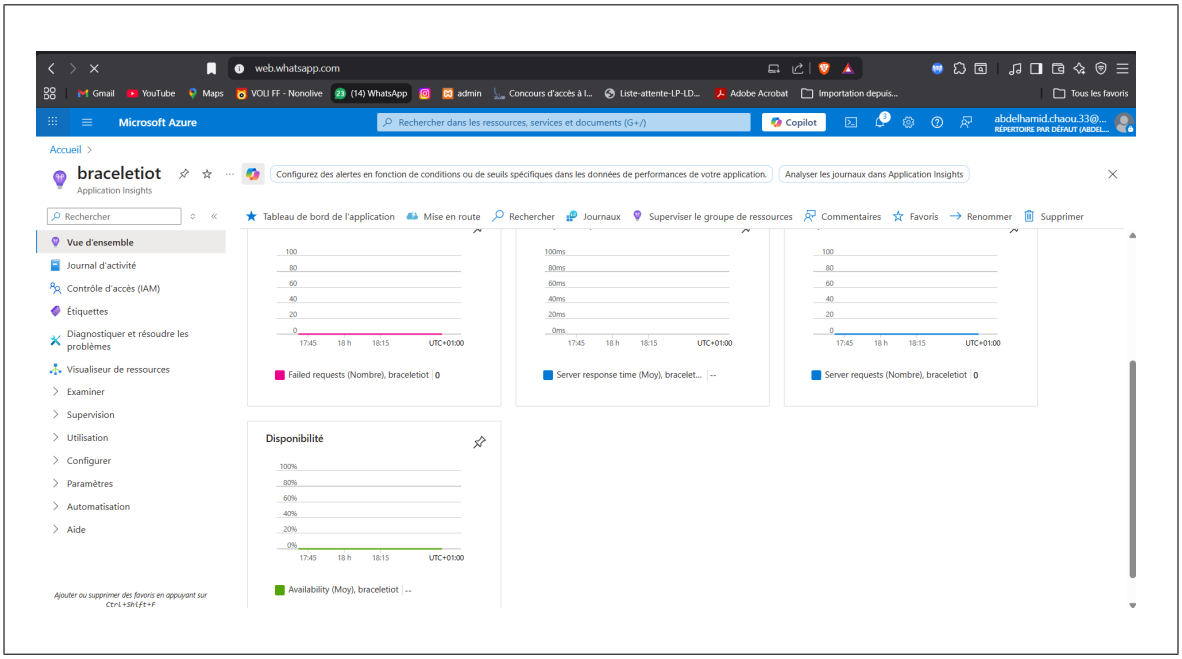


FIGURE 4.18 – Tableau de bord de surveillance montrant l'activité de l'API en temps réel.

CHAPITRE 5

CONCLUSION

Ce projet a permis de concrétiser une solution complète et performante, répondant à la problématique initiale par le développement d'une **application de messagerie et de suivi associée à un bracelet connecté**. Le succès de cette réalisation repose sur l'intégration harmonieuse de trois piliers technologiques : l'**IoT**, le **Backend** et l'**Application Mobile**.

5.0.1 I. Atteinte des Objectifs et Validation Technique

La phase de Conception a abouti à une architecture robuste, définissant des flux utilisateurs clairs et une séparation nette des responsabilités entre les composants.

- **Côté IoT** : L'intégration des composants matériels et la maîtrise du **Bluetooth Low Energy (BLE)** ont permis d'établir un canal de communication fiable et optimisé en énergie entre le bracelet et l'application mobile. Tous les tests de validation ont confirmé la capacité du bracelet à opérer de manière autonome et à transmettre les données ou messages requis.
- **Côté Backend (FastAPI)** : L'implémentation a mis l'accent sur la performance et la sécurité. L'utilisation de **SQLAlchemy** et des index composites a optimisé la persistance des données massives issues de l'IoT, tandis que la mise en œuvre rigoureuse d'**OAuth2/JWT** garantit une sécurité stricte pour les données utilisateurs et les endpoints critiques.
- **Côté Application Mobile** : La réalisation a permis de valider l'ensemble des fonctionnalités principales et critiques, offrant aux utilisateurs une interface intuitive pour l'interaction avec le bracelet (messagerie, appairage) et la visualisation des données.

5.0.2 II. Bilan et Apports

Ce projet représente une expérience d'intégration système très formatrice, soulignant l'importance de la synergie entre les disciplines :

- **Maîtrise de l'Asynchrone** : L'approche asynchrone du backend FastAPI s'est révélée essentielle pour gérer efficacement l'ingestion de données en masse sans dégrader la performance globale du serveur.
- **Sécurité et Intégrité** : Le choix de l'authentification par **API Key** pour le bracelet et

par **JWT** pour l'application mobile a permis de mettre en place un modèle de sécurité multicouche adapté à l'écosystème IoT.
— **Déploiement Professionnel** : La mise en œuvre du déploiement continu sur **Azure** a assuré un environnement de production stable et évolutif, préparant la solution à un passage à l'échelle potentiel.

5.0.3 III. Perspectives d'Amélioration

Pour aller plus loin, plusieurs pistes d'évolution pourraient être explorées :

1. **Analyse de Données Avancée** : Développer des endpoints d'agrégation plus complexes pour offrir des analyses prédictives.
2. **Gestion du Temps Réel** : Intégrer des WebSockets (ou similaire) pour permettre une communication en temps réel entre le bracelet, le backend et l'application mobile.
3. **Optimisation BLE** : Poursuivre l'optimisation du firmware pour minimiser davantage la consommation d'énergie du bracelet et prolonger son autonomie.

En conclusion, le projet a non seulement atteint ses objectifs fonctionnels et techniques, mais a également démontré la faisabilité d'intégrer des composants matériels propriétaires dans une architecture Cloud moderne, créant ainsi une base solide pour de futurs développements dans le domaine de la santé ou de la communication connectée.