Rapport Projet Architecture Logicielle

Projet E-santé

Equipe :

- Yasmine Moussaoui
- Bilal Mesaouri Deboun
- Emmanuelle Adote
- Lamya Baidouri
- Dassou Ablam Kaleb Sika

Encadré par:

Nicolas Ferry & Gérald Rocher

Sommaire

1. Introduction

Le présent rapport présente E-santé, une solution de santé connectée que nous avons développée pour optimiser la gestion des soins et permettre une surveillance continue de l'état de santé des patients. Cette solution repose sur l'utilisation de montres connectées **Scan Watch Nova Brillant** qui mesurent en continu divers signes vitaux des patients, tels que la fréquence cardiaque, la saturation en oxygène, la température corporelle, et plus encore. Ces données sont transmises périodiquement à l'application, permettant ainsi aux professionnels de santé (médecins et infirmières) et aux proches d'accéder rapidement aux informations médicales essentielles.
Grâce à E-santé, les intervenants peuvent suivre l'évolution de la santé des patients à distance, être alertés en cas de valeurs critiques, et intervenir si nécessaire. Cette approche de la télésurveillance vise à améliorer la prise en charge des patients en offrant une vue d'ensemble de leur état de santé, tout en favorisant une réactivité accrue pour les situations urgentes.
.

2. Analyse des besoins

Lors de l'analyse des besoins, nous avons identifié cinq personas représentant des utilisateurs potentiels, chacun ayant des besoins spécifiques et des attentes précises par rapport à notre application.

Persona 1 : Médecin (Dr. Martin)

- **Profil** : Médecin généraliste dans un centre hospitalier
- **Besoins** :
    - En tant que médecin, je veux pouvoir consulter la liste des patients ordonnée selon la sévérité des cas, afin de prioriser les soins.
    - En tant que médecin, je veux pouvoir afficher le dossier de mon patient et voir l'évolution des mesures en se basant sur un graphe, pour suivre son état de santé.
    - En tant que médecin, je veux pouvoir transférer mon patient à un autre médecin, afin d'assurer une continuité dans la prise en charge.
    - En tant que médecin, je veux pouvoir avoir accès à une messagerie pour obtenir plus d'informations, afin de communiquer efficacement avec mes patients et collègues.
    - En tant que médecin, je veux pouvoir envoyer un message à un patient pour obtenir plus d'informations, afin d'adapter son traitement rapidement.
    - En tant que médecin, je veux pouvoir assigner une infirmière à un patient pour assurer sa prise en charge, afin de mieux coordonner les soins.
    - En tant que médecin, je veux pouvoir fixer un rendez-vous avec les cas les plus urgents, pour optimiser mon emploi du temps et répondre aux priorités.
    - En tant que médecin, je veux recevoir des alertes automatiques lorsqu'un patient présente des valeurs critiques dans ses mesures, afin de réagir rapidement et ajuster les soins si nécessaire.

Persona 2 : Patient (Marie)

- **Profil** : Patiente souffrant d'une maladie chronique
- **Besoins** :
    - En tant que patient, je veux pouvoir envoyer un message à mon médecin traitant en cas de besoin, pour rester en contact et obtenir des conseils médicaux.
    - En tant que patient, je veux pouvoir transmettre mon ressenti en remplissant un formulaire personnalisé selon ma maladie, afin d'informer mon médecin de mon état de santé actuel.
    - En tant que patient, je veux pouvoir consulter l'historique de mon état de santé, pour suivre ma progression et comprendre les variations dans mon état.
    - En tant que patient, je veux pouvoir envoyer une alerte urgente à mon médecin ou à un proche infirmier, en cas de besoin immédiat d'assistance.
    - En tant que patient, je veux pouvoir créer des rappels pour me souvenir de prendre mes médicaments ou d'un rendez-vous médical, afin de gérer efficacement mes traitements.
    - En tant que patient, je veux pouvoir consulter mon dossier médical contenant toutes les informations (médecin, infirmière, rendez-vous, proches), pour avoir une vue complète de ma situation.
    - En tant que patient, je veux pouvoir choisir la liste des proches qui peuvent suivre mon état de santé, afin de partager les informations avec ceux en qui j'ai confiance.
    - En tant que patient, je veux pouvoir définir le niveau de confidentialité des informations à partager avec mes proches, afin de contrôler quelles données de santé ils peuvent consulter.

Persona 3 : Proche du patient (Pierre)

- **Profil** : Proche d'un patient souffrant d'une maladie chronique (fils d'un patient)
- **Besoins** :
    - En tant que proche du patient X, je veux pouvoir recevoir un bilan hebdomadaire sur l'état de santé de mon père, avec son consentement, afin d'être à jour sur l'évolution de son état.
    - En tant que proche du patient X, je veux pouvoir être alerté en cas de transfert à l'hôpital, afin de pouvoir le rejoindre rapidement et organiser mon soutien.
    - En tant que proche du patient X, je veux pouvoir recevoir des recommandations d'actions à entreprendre pour favoriser la guérison de mon père (par exemple, modifier son alimentation), afin de contribuer à son rétablissement.

Persona 4 : Infirmière (Laura)

- **Profil** : Infirmière responsable des soins à domicile
- **Besoins** :
    - En tant qu'infirmière, je veux consulter les tickets qui me sont adressés par les docteurs, pour savoir quels patients je dois suivre.
    - En tant qu'infirmière, je veux pouvoir planifier des visites à domicile ou des consultations virtuelles avec les patients qui me sont affectés par les docteurs, afin de gérer efficacement mes interventions.

- En tant qu'infirmière, je veux pouvoir rendre compte au médecin traitant des diverses interventions réalisées chez un patient, pour assurer un suivi médical complet.
- En tant qu'infirmière, je veux pouvoir accéder à l'historique des données du patient afin de prendre les décisions adéquates pour les soins à apporter.
- En tant qu'infirmière, je veux pouvoir communiquer avec les proches des patients et partager les informations pertinentes sur leur santé, pour faciliter la coordination des soins.
- En tant qu'infirmière, je souhaite être notifiée en temps réel des incidents ou anomalies concernant les patients qui me sont affectés, afin de pouvoir réagir rapidement.

Persona 5 : Administrateur (Thomas)

- **Profil** : Administrateur du système de gestion des utilisateurs et des capteurs connectés
- **Besoins** :
  - En tant qu'administrateur, je veux pouvoir créer, modifier et supprimer des comptes utilisateurs (médecins, infirmières, patients), afin d'assurer une gestion efficace des accès au système.
  - En tant qu'administrateur, je veux pouvoir définir et gérer les rôles et permissions des utilisateurs, afin que chaque utilisateur (médecin, infirmière, proche, etc.) ait accès uniquement aux informations et fonctionnalités qui le concernent.
  - En tant qu'administrateur, je veux pouvoir ajouter, associer ou supprimer des capteurs pour les patients, afin de m'assurer que chaque patient dispose des dispositifs de suivi adaptés à son état de santé.
  - En tant qu'administrateur, je veux recevoir des notifications en cas de dysfonctionnement technique (ex. capteur défectueux, panne serveur), afin de pouvoir intervenir rapidement et garantir la continuité du service.

3. Choix de l'architecture

L'architecture mise en place est orientée vers la sécurité, la fiabilité et l'efficacité. Elle contient des technologies modernes comme MQTT pour le transfert de données IoT, Kafka pour la gestion d'événements inter-service, et Grafana pour la visualisation en temps réel. Les choix technologiques et la segmentation en microservices assurent une scalabilité et facilitent la maintenance de l'application, tandis que l'ajout de traitements en bordure sur la Raspberry Pi réduit la charge sur le backend et améliore la résilience de l'application.

1- Traitement en Bordure (FOG) sur la Raspberry Pi

Pour la première collecte des données, nous avons choisi d'implémenter un dispositif de traitement en bordure. Ce dispositif permet non seulement de traiter les données avant leur envoi, mais aussi de les stocker temporairement en cas de problèmes de connexion Internet. Le choix de la **Raspberry Pi** s'avère pertinent pour répondre à ces besoins, notamment grâce à sa flexibilité et à ses faibles besoins en énergie. Contrairement aux appareils mobiles, elle ne dépend pas de la batterie de l'utilisateur, ce qui élimine les problèmes d'autonomie. En cas de mobilité, elle peut même être alimentée par une batterie externe (power bank).

Les traitements effectués en "edge computing" ou en **Fog computing** sur cet appareil comprennent plusieurs étapes :

- **Réduction de bruit** : pour éliminer les interférences dans les données brutes, en particulier dans des scénarios où les mouvements du patient (par exemple, lors de la mesure de la pression artérielle) peuvent introduire des erreurs. Des algorithmes de filtrage, tels que le **filtre de Kalman**, seront utilisés pour atténuer ces bruits et garantir la précision des mesures.
- **Agrégation** : pour regrouper les données sur une période donnée (par exemple, la moyenne de la fréquence cardiaque sur 5 minutes) afin de réduire la fréquence de transmission, tout en maintenant la pertinence des données.
- **Compression** : pour diminuer la quantité de données à transmettre, optimisant ainsi la bande passante.
- **Stockage local en cas de perte de connexion** : en utilisant une carte microSD, les données sont stockées temporairement jusqu'à ce qu'une connexion soit rétablie.

Ces traitements permettent de minimiser la quantité de données envoyées au serveur central, ce qui réduit les coûts en bande passante et améliore la rapidité de traitement. En compressant et en agrégeant les données, la Raspberry Pi diminue la latence et les besoins en stockage sur le serveur. Enfin, le blocage local en cas de déconnexion garantit que les données ne sont pas perdues, ce qui est crucial dans une application de santé où la continuité de l'information est essentielle pour le suivi des patients par les médecins.

Nous tenons à préciser que la réduction du bruit et l'agrégation des données sont actuellement appliquées pour la mesure de la pression artérielle. En effet, lors de cette mesure ponctuelle, des mouvements du bras du patient peuvent générer du bruit, ce qui pourrait fausser les résultats. Grâce à des algorithmes de réduction du bruit, nous sommes en mesure d'assurer des mesures précises même dans des conditions non idéales.

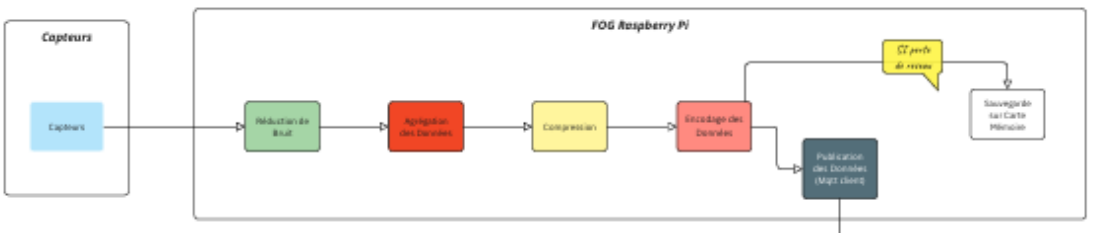

Figure 1.

2- Transmission Sécurisée par MQTT et TCP/TLS

Dans notre architecture, la transmission des données de santé entre la **Raspberry Pi** et le **backend** est réalisée via le protocole **MQTT** sécurisé par **TCP/TLS** pour les données critiques.

Contrairement à HTTP, qui fonctionne sur un modèle de requête-réponse, MQTT utilise un modèle de publication/souscription. Cela permet une transmission continue et réactive des données, sans avoir besoin de rétablir une connexion à chaque envoi, ce qui est idéal pour les dispositifs comme la Raspberry Pi.

En plus, MQTT utilise moins de bande passante par rapport à HTTP, car il envoie **uniquement** les données essentielles dans des messages de petite taille. Cela est particulièrement important dans les zones rurales où la connexion peut être instable ou limitée.

Un autre point positif de cette technologie, le modèle **Publish/Subscribe**, Dans MQTT, les données sont publiées une seule fois par la Raspberry Pi sur un topic spécifique, et tous les services qui sont abonnés à ce topic reçoivent automatiquement le message. Cela signifie que la Raspberry Pi envoie ses données **une seule fois**, et le broker s'occupe de distribuer ces données aux abonnés concernés.

Avec MQTT et grâce à notre choix d'EMQX, nous pouvons configurer des règles pour détecter facilement l'inactivité des Raspberry Pi. Si un Raspberry Pi ne transmet pas de données pendant un certain temps défini (delta), une alerte sera automatiquement envoyée aux mainteneurs de notre système via un topic approprié..

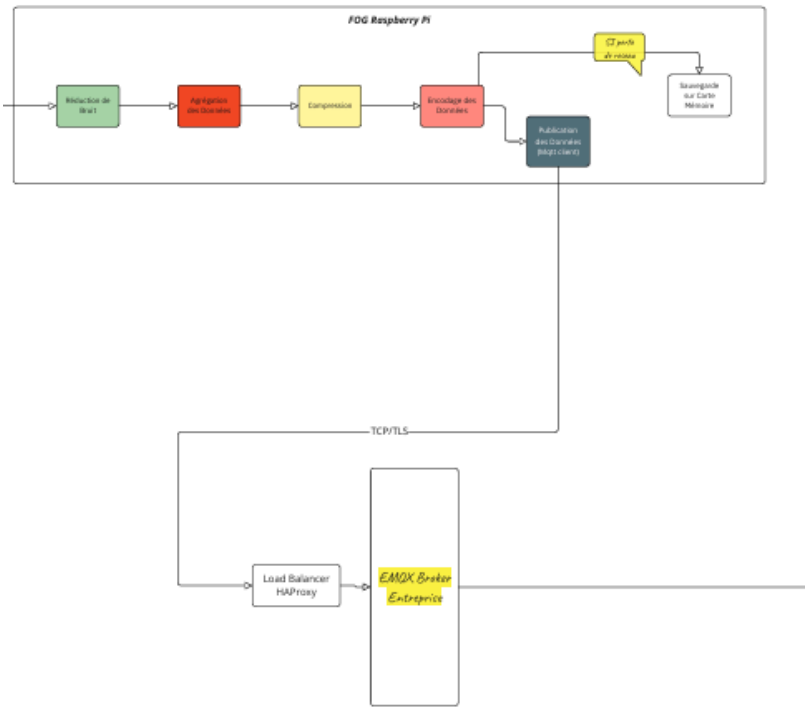

Figure 2.

3- Backend Centralisé avec Kafka pour la Communication Inter-Service

Le backend est organisé en plusieurs services spécialisés, notamment :

- **PatientService** : pour gérer les informations des patients.
- **MeasurementService** : pour traiter les mesures des capteurs.
- **NotificationService** : pour envoyer des notifications, des alertes ....
- **UserService** : pour gérer les utilisateurs, l'authentification …

Vous trouverez plus d'informations sur les fonctionnalités de chaque service dans les user stories.

Les services communiquent via Kafka pour la publication/souscription d'événements, et les données sont stockées dans une base de données MongoDB, adaptée pour les séries temporelles.

MongoDB est aussi conçu pour être **hautement scalable**, ce qui est essentiel dans une application où la quantité de données collectées peut croître rapidement,

Pour la communication inter-service, Kafka est un choix robuste pour gérer le flux de données continu des capteurs. Il permet un traitement asynchrone et une scalabilité, indispensable dans un environnement où des centaines voire des milliers de capteurs peuvent émettre des données simultanément. La spécialisation des services améliore la modularité et facilite la maintenance et l'évolution de l'application.

L'architecture choisie repose sur des microservices conteneurisés avec **Docker**, orchestrés par **Kubernetes** pour assurer la **scalabilité** et la **résilience** de l'application e-santé. Chaque microservice (par exemple, gestion des patients, mesures, notifications) fonctionne de manière indépendante, facilitant la maintenance et le déploiement continu (via un serveur Jenkins, où une pipeline construit les conteneurs, les pousse sur Docker Hub ou GHCR, puis utilise un chart Helm pour redéployer le service avec une simple commande helm upgrade). Kubernetes prend en charge l'**auto-scaling**, permettant d'ajuster dynamiquement le nombre de répliques des conteneurs en fonction de la charge, assurant ainsi une gestion optimale des ressources.

Un autre point de kubernetes est le **load balancing**, Kubernetes utilise un équilibrage interne pour répartir le trafic réseau de manière uniforme entre les instances de chaque microservice. Cela garantit que les requêtes sont traitées efficacement, réduisant les temps de réponse et évitant les surcharges sur des instances spécifiques.

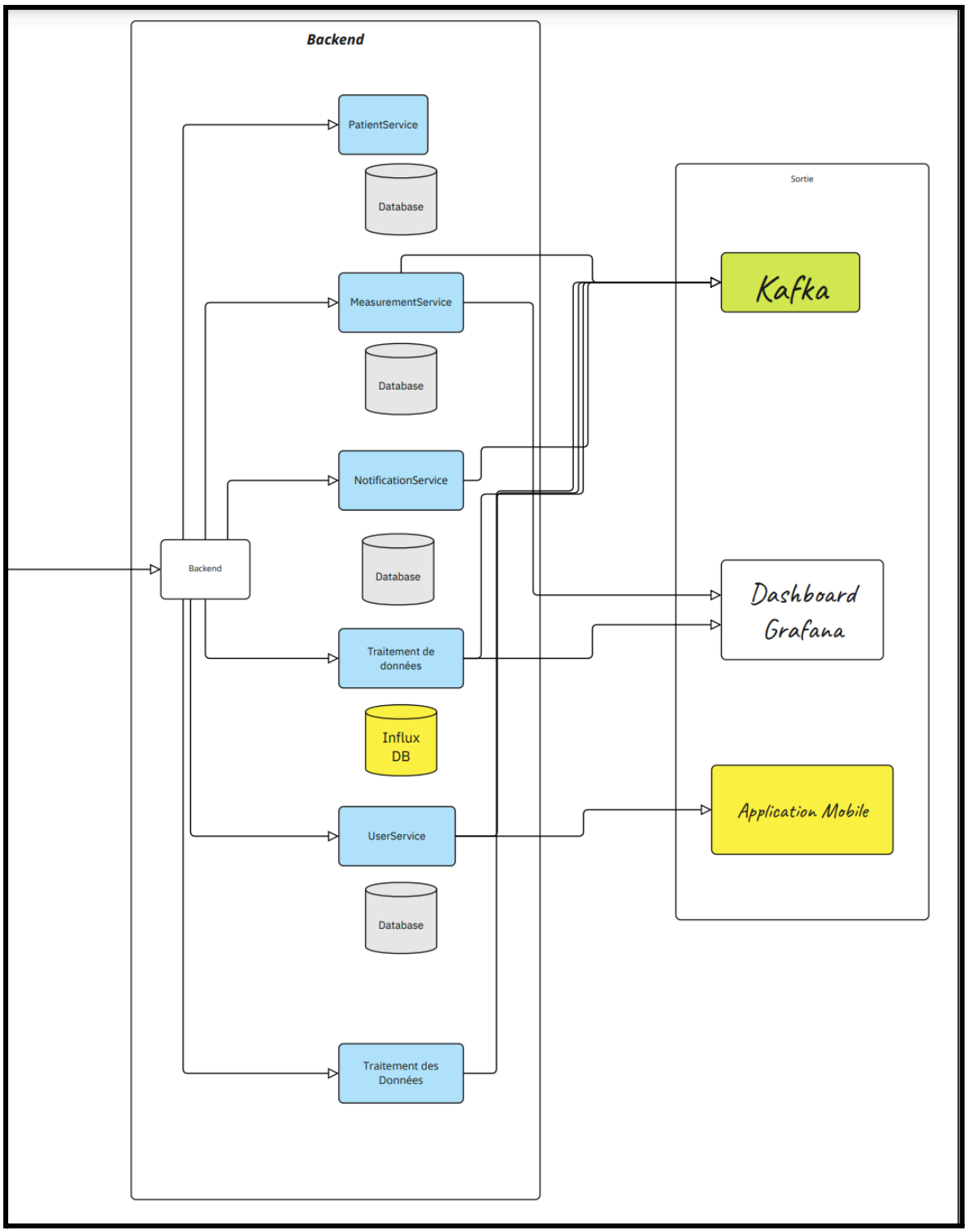

Figure 3. Architecture backend

4- Visualisation des Données

Les utilisateurs de notre application sont répartis en trois catégories :

1. **Les patients**
2. **Le personnel médical**

3. **L’administration**

Le choix de l'appareil dépend des besoins de chaque catégorie d'utilisateur.

- **Patients** : Pour les patients, nous avons opté pour l'utilisation du smartphone. Cet appareil est accessible à tout moment, disponible pour la plupart des utilisateurs, et idéal pour recevoir des notifications en temps réel sans coûts additionnels.
- **Personnel médical et administration** : Pour le personnel médical et l'administration, nous avons choisi **Grafana** comme outil principal. Grafana permet une analyse approfondie des données des patients et des statistiques globales, offrant une interface claire et adaptée pour le suivi et la prise de décision en milieu médical.

Cette distinction d’appareil par catégorie assure une expérience utilisateur optimisée pour chaque groupe et permet d’adapter les fonctionnalités aux besoins spécifiques de chacun

4. Les micros-services

L'architecture microservices a été choisie pour ce projet en raison de sa capacité à répondre aux exigences de modularité, scalabilité et résilience. Chaque service est indépendant et focalisé sur une responsabilité unique, ce qui permet une évolution rapide et un déploiement autonome sans affecter les autres services. Dans un contexte où les données médicales sont sensibles et critiques, cette approche garantit une meilleure sécurité, avec des bases de données dédiées par service et une séparation stricte des responsabilités. De plus, la communication asynchrone entre les services assure une réponse en temps réel pour des événements critiques, tout en maintenant une faible latence et une tolérance aux pannes.

1.Service Patient

Responsabilité :
Gérer les informations des patients, les relations avec le personnel médical et les contacts proches.

Méthode	**Description**	**Utilisateurs**
getAllPatientsOrderedBySeverity()	Liste les patients par gravité des cas.	Médecin
getPatientDetails(Long patientId)	Afficher les informations complètes du patient	Médecin,Infirmière
transferPatient(Long patientId, Long newDoctorId)	Transférer un patient à un autre médecin	Médecin
manageTrustedContacts(Long patientId, List<Long> contactIds,	Gère les autorisations des accès des Gérer les accès aux proches proches aux informations médicales	Patient

Méthode	Description	Utilisateurs
getAllPatientsOrderedBySeverity()	Liste les patients par gravité des cas.	Médecin
getPatientDetails(Long patientId)	Afficher les informations complètes du patient	Médecin,Infirmière
transferPatient(Long patientId, Long newDoctorId)	Transférer un patient à un autre médecin	Médecin
PrivacySettingsDTO privacySettings)		
scheduleAppointment(Long patientId, LocalDateTime dateTime)	Planifier un rendez-vous	Médecin
submitFeedback(Long patientId, FeedbackDTO feedback)	Permettre au patient de soumettre des retours	Patient
sendMessageToDoctor(Long DoctorId, String message)	Envoyer message au médecin	Patient
createPatient(PatientDTO patientDTO)	Créer un utilisateur avec le rôle Patient.	Administrateur, Patient

Table 1.

2.Service Data

Persister les données reçues des capteurs via le broker EMAX et exposer des endpoints pour accéder à ces données.

Méthode	Description	Utilisateurs
persistSensorData(SensorDataDTO sensorData)	Enregistrer les données reçues des capteurs.	MeasurementService
getSensorDataByPatientId(Long patientId)	Récupérer les données des capteurs pour un patient.	Médecin, Infirmière, Patient, ReportingService
getSensorDataByType(Long patientId, String type)	Récupérer les données d'un patient filtrées par type	Médecin, Infirmière, ReportingService

deleteSensorData(Long patientId)	Supprimer les données des capteurs pour un patient.	Administrateur
getLatestSensorData(Long patientId)	Récupérer les dernières données pour un patient.	Médecin, Infirmière

Table 2.

3. Service measurements

Responsabilité :

Gérer les données brutes des capteurs et détecter les anomalies critiques.

Méthodes :

Méthode	**Description**	**Utilisateurs**
receiveSensorData(SensorDataDTO sensorData)	Reçoit les données des capteurs via MQTT	DataService
detectCriticalValues(Long patientId, Measurement measurement)	Détecter valeurs critiques et envoie des alertes	NotifcationService
manageSensors(Long patientId, List<SensorDTO> sensors)	Ajouter ou supprimer un capteur pour un patient	Administrateur

Table 3.

4. Service Notification

Responsabilité :

Gérer l'envoi de notifications aux utilisateurs et la messagerie .

Méthode	**Description**	**Utilisateurs**
sendCriticalAlert(Long patientId, String message)	Envoyer une alerte critique à un médecin ou une infirmière.	MeasurementService

sendTransferAlert(Long patientId, String message)	Notifier qu'un patient a été transféré.	PatientService
sendMessage(Long senderId, Long receiverId, String message)	Permettre l'échange de messages entre utilisateurs.	Médecin, Patient
getNotificationsForUser(Long userId)	Récupèrer toutes les notifications d'un utilisateur	Tous les rôles

Table 4.

5. Service User

Responsabilité :

Gérer l'authentification et la création d'utilisateurs.

Méthode	**Description**	**Utilisateurs**
createUser(UserDTO userDTO)	Créer un utilisateur générique avec un rôle spécifique.	Administrateur
authenticateUser(String username, String password)	Authentifier un utilisateur et génère un token JWT	Tous les rôles
assignRole(Long userId, String role)	Attribuer un rôle spécifique à un utilisateur	Administrateur
resetPassword(Long userId, String newPassword)	Réinitialiser le mot de passe d'un utilisateur.)	Tous les rôles

Table 5.

6 . Service Doctor

Responsabilité : Gérer les opérations spécifiques aux médecins.

Méthode	**Description**	**Utilisateurs**
createDoctor(DoctorDTO doctorDTO)	Créer un nouveau médecin	Administrateur

getPatientsUnderDoctor(Long doctorId)	Liste les patients du médecin.	Médecin
transferPatient(Long patientId, Long newDoctorId)	Transfèrer un patient.	Médecin
sendMessageToPatient(Long patientId, String message)	Envoyer un message au patient	Médecin
getNursesUnderDoctor	Liste des infirmières avec qui il collabore	Médecin

Table 6.

7.Service Nurse

Responsabilité : Gérer les opérations spécifiques aux infirmières.

Méthode	**Description**	**Utilisateurs**
createNurse(NurseDTO nurseDTO)	Créer une nouvelle infirmière.	Administrateur , infirmière
getAssignedPatients(Long nurseId)	Liste des patients liée à cette infirmière	Infirmière
communicateWithDoctor(Long doctorId, MessageDTO message)	Communiquer avec le médecin.	Infirmière
recordIncident(Long patientId, IncidentDTO incident)	Enregistrer un incident.	Infirmière

Table 7.

8.Service Proche

Responsabilité : Gérer les opérations des proches.

Méthode	**Description**	**Utilisateurs**

createPatientContact(CloseContactDTO closeContactDTO)	Créer un nouveau proche.	Administrateur, Proche
viewAuthorizedPatientData(Long closeContactId, Long patientId)	Consulter les données autorisées du patient.	Proche
receiveNotifications(Long closeContactId)	Récupérer les notifications.	Proche

Table 8.

5. Modèles de Base de Données pour E-santé avec Architecture Microservices

Dans le cadre de la solution E-santé, une approche recommandée consiste à maintenir des tables séparées pour chaque type de relation (proches, médecins, infirmières,médecins) associée aux patients . on s'est basée sur certains principe :

Séparation des Préoccupations

La séparation des préoccupations est un principe fondamental en ingénierie logicielle qui préconise de diviser un système en sections distinctes, chacune étant responsable d'un aspect spécifique du fonctionnement global. Appliqué à la conception de bases de données, cela signifie que chaque type de relation (proches, médecins, infirmières) devrait être géré par des tables dédiées. Cette approche présente de nombreux avantages, notamment la clarté et l'organisation, car chaque table, ayant une responsabilité unique, rend la structure de la base de données plus compréhensible et plus facile à naviguer. De plus, elle permet de réduire la complexité en évitant la surcharge d'une seule table avec de multiples types de relations, limitant ainsi la complexité des requêtes et des opérations de gestion des données.

Normalisation des Données

La normalisation est un processus visant à structurer une base de données de manière à minimiser la redondance et à améliorer l'intégrité des données. En maintenant des tables séparées pour chaque type de relation, on adhère plus strictement aux premières formes normales (1NF, 2NF, 3NF), ce qui évite les anomalies de mise à jour, d'insertion et de suppression

Intégration Harmonisée avec l'Architecture Microservices

Dans une architecture microservices, chaque service gère une partie spécifique des fonctionnalités de l'application, et maintenir des tables séparées pour chaque type de relation s'aligne parfaitement avec ce paradigme, favorisant l'indépendance et la modularité des services. Cela apporte plusieurs avantages, tels que l'indépendance des services, permettant à chaque microservice de gérer sa propre

table sans dépendre directement des autres services, ce qui facilite le déploiement et la scalabilité indépendants. De plus, la communication inter-services est facilitée, les services pouvant échanger via des API bien définies sans avoir besoin de partager directement les données internes des tables des autres services

1. Service Patient

Table : Patients

Nom de Colonne	Type de Donnée	Clé	Description
patient_id	BIGINT	PK	Identifiant unique du patient
username	VARCHAR	UNIQUE	Nom d'utilisateur du patient
Password	VARCHAR		mot de passe
first_name	VARCHAR		Prénom
last_name	VARCHAR		Nom de famille
date_of_birth	DATE		Date de naissance
gender	VARCHAR		Sexe
address	VARCHAR		Adresse
phone_number	VARCHAR		Numéro de téléphone
email	VARCHAR		Adresse e-mail
privacy_settings	JSON		Paramètres de confidentialité

Figure 4.

Table : Trusted_Contacts

Nom de Colonne	Type de Donnée	Clé	Description
proche_id	BIGINT	PK	Identifiant unique du proche
patient_id	BIGINT	FK vers Patients	Référence au patient associé
contact_id	BIGINT	Référence au Service Proche	Identifiant du proche
access_permissions	JSON		Permissions d'accès accordées

Figure 5.

Table : Patient_Doctors

Nom de Colonne	Type de Donnée	Clé	Description
patient_id	BIGINT	PK (Composite)	Identifiant du patient
doctor_id	BIGINT	PK (Composite)	Identifiant du médecin (référence au Service Doctor)

Figure 6.

Table : Appointments

Nom de Colonne	Type de Donnée	Clé	Description
appointment_id	BIGINT	PK	Identifiant unique du rendez-vous
patient_id	BIGINT	FK vers Patients	Référence au patient
doctor_id	BIGINT	Référence au Service Doctor	Référence au médecin
date_time	DATETIME		Date et heure du rendez-vous
status	VARCHAR		Statut du rendez-vous

Figure 7.

Table : Feedbacks

Nom de Colonne	Type de Donnée	Clé	Description
feedback_id	BIGINT	PK	Identifiant unique du feedback
patient_id	BIGINT	FK vers Patients	Référence au patient
feedback_text	TEXT		Contenu du feedback
submitted_at	DATETIME		Date et heure de soumission

Figure 8.

2. Service Doctor

Table : Doctors

Nom de Colonne	Type de Donnée	Clé	Description
doctor_id	BIGINT	PK	Identifiant unique du médecin
username	VARCHAR	UNIQUE	Nom d'utilisateur du médecin
password	VARCHAR		mot de passe
first_name	VARCHAR		Prénom
last_name	VARCHAR		Nom de famille
specialization	VARCHAR		Spécialisation
phone_number	VARCHAR		Numéro de téléphone
email	VARCHAR		Adresse e-mail

Figure 9.

Table : Doctor_Nurses

Nom de Colonne	Type de Donnée	Clé	Description
doctor_id	BIGINT	PK	Identifiant du médecin
nurse_id	BIGINT	PK	Identifiant de l'infirmière (référence au Service Nurse)

Figure 10.

3. Service Nurse

Table : Nurses

Nom de Colonne	Type de Donnée	Clé	Description
nurse_id	BIGINT	PK	Identifiant unique de l'infirmière
username	VARCHAR	UNIQUE	Nom d'utilisateur de l'infirmière
password	VARCHAR		mot de passe
first_name	VARCHAR		Prénom
last_name	VARCHAR		Nom de famille
department	VARCHAR		Département
phone_number	VARCHAR		Numéro de téléphone
email	VARCHAR		Adresse e-mail

Figure 11.

Table : Nurse_Patients

Nom de Colonne	Type de Donnée	Clé	Description
nurse_id	BIGINT	PK	Identifiant de l'infirmière
patient_id	BIGINT	PK	Identifiant du patient (référence au Service Patient)

Figure 12.

4. Service Proche

Table : Proches

Nom de Colonne	Type de Donnée	Clé	Description
contact_id	BIGINT	PK	Identifiant unique du proche
username	VARCHAR	UNIQUE	Nom d'utilisateur du proche
password	VARCHAR		mot de passe
first_name	VARCHAR		Prénom
last_name	VARCHAR		Nom de famille
relationship	VARCHAR		Relation avec le patient
phone_number	VARCHAR		Numéro de téléphone
email	VARCHAR		Adresse e-mail

Figure 13.

Table : Patient_Proche

Nom de Colonne	Type de Donnée	Clé	Description
patient_id	BIGINT	PK	Identifiant du patient (référence au Service Patient)
contact_id	BIGINT	PK	Identifiant du proche
access_permissions	JSON		Permissions d'accès accordées

Figure 14.

5. Service Data

Table : Sensor_Data

Nom de Colonne	Type de Donnée	Clé	Description
data_id	BIGINT	PK	Identifiant unique de la donnée
patient_id	BIGINT	Référence au Service Patient	Référence au patient
sensor_type	VARCHAR		Type de capteur
measurement_value	DECIMAL		Valeur mesurée
measurement_unit	VARCHAR		Unité de mesure
timestamp	DATETIME		Date et heure de la mesure

Figure 15.

Table : Sensor_Types

Nom de Colonne	Type de Donnée	Clé	Description
sensor_type	VARCHAR	PK	Type de capteur
description	VARCHAR		Description du type de capteur

Figure 16.

6. Service Measurements

Table : Sensors

Nom de Colonne	Type de Donnée	Clé	Description
sensor_id	BIGINT	PK	Identifiant unique du capteur
patient_id	BIGINT	Référence au Service Patient	Référence au patient
sensor_type	VARCHAR	Référence au Service Data	Type de capteur
status	VARCHAR		Statut du capteur (actif/inactif)
installed_at	DATETIME		Date d'installation

Figure 17.

Table : Critical_Alerts

Nom de Colonne	Type de Donnée	Clé	Description
alert_id	BIGINT	PK	Identifiant unique de l'alerte
patient_id	BIGINT	Référence au Service Patient	Référence au patient
measurement_id	BIGINT	Référence au Service Data	Référence à la mesure
alert_type	VARCHAR		Type d'alerte
alert_message	TEXT		Message de l'alerte
triggered_at	DATETIME		Date et heure du déclenchement

Figure 18.

7. Service Notification

Table : Notifications

Nom de Colonne	Type de Donnée	Clé	Description
notification_id	BIGINT	PK	Identifiant unique de la notification
user_type	VARCHAR		Type d'utilisateur ('Patient', 'Doctor', 'Nurse', 'CloseContact')
user_id	BIGINT		Identifiant de l'utilisateur dans le service correspondant
message	TEXT		Contenu de la notification
created_at	DATETIME		Date et heure de création
is_read	BOOLEAN		Indique si la notification a été lue

Figure 19.

Table : Messages

Nom de Colonne	Type de Donnée	Clé	Description
message_id	BIGINT	PK	Identifiant unique du message
sender_type	VARCHAR		Type de l'expéditeur
sender_id	BIGINT		Identifiant de l'expéditeur
receiver_type	VARCHAR		Type du destinataire
receiver_id	BIGINT		Identifiant du destinataire
content	TEXT		Contenu du message
sent_at	DATETIME		Date et heure d'envoi

Relations Données Entre les Services :

Figure 20.

Service Source	Table Source	Clé Source	Service Cible	Table Cible	Clé Cible	Type de Relation
Service Patient	Patients	patient_id	Service Doctor	Doctors	doctor_id	N-N via Patient_Doctors
Service Patient	Trusted_Contacts	contact_id	Service CloseContact	Close_Contacts	contact_id	N-1
Service Doctor	Doctor_Nurses	nurse_id	Service Nurse	Nurses	nurse_id	N-N via Doctor_Nurses
Service Nurse	Nurse_Patients	patient_id	Service Patient	Patients	patient_id	N-N via Nurse_Patients
Service Measurements	Sensors	patient_id	Service Patient	Patients	patient_id	N-1
Service Measurements	Sensors	sensor_type	Service Data	Sensor_Types	sensor_type	N-1
Service Data	Sensor_Data	patient_id	Service Patient	Patients	patient_id	N-1
Service Notification	Notifications	user_id, user_type	Services Respectifs	-	user_id	N-1 (Basé sur user_type)
Service Notification	Messages	sender_id, sender_type	Services Respectifs	-	user_id	N-1 (Basé sur sender_type)
Service Notification	Messages	receiver_id, receiver_type	Services Respectifs	-	user_id	N-1 (Basé sur receiver_type)

Figure 21.

6. Choix des capteurs

Pour enregistrer les données de santé des patients, nous avons opté pour le choix d'une montre afin d'assurer le confort des patients :

Il s'agit de la montre **Scan Watch Nova Brillant**, qui permet de mesurer les paramètres suivants :

- la température
- la fréquence cardiaque
- les paramètres de sommeil
- la fréquence respiratoire
- le SpO2
- l'ECG

Les capteurs intégrés sont :

- Temp Tech24/7 Module
- High Dynamic Range Accelerometer
- Multi-wavelength PPG 16 channels
- Altimeter

Notre choix a été motivé par son autonomie de 30 jours, la variété de paramètres mesurés, la connectivité BLE et la documentation riche et accessible.

Pour des raisons de simplicité, nous partons de l'hypothèse selon laquelle la **ScanWatch Nova Brilliant** est connectée directement à notre **gateway** en local, sans passer par le cloud du fabricant, et envoie les données en continu à ce dernier. Bien que cette montre nous permette de mesurer un certain nombre de paramètres, nous souhaitons également enregistrer certaines données, comme les chutes, que le patient soit chez lui ou à l'extérieur, ainsi que la pression artérielle.

Pour détecter les chutes, nous avons choisi un système d'alerte fictif fonctionnant de manière similaire à celui de **MobileHelp**. Il permet au patient, en cas de malaise, de presser sur un bouton pour recevoir de l'aide. En cas de chute, une alerte est envoyée automatiquement avec la position GPS du patient et les données sur la gravité de l'incident, mesurées par **l'accéléromètre** intégré. Une carte SIM intégrée permet au dispositif de rester toujours connecté. Les données ne sont envoyées qu'en cas de chute. Le dispositif garde les données jusqu'à une future connexion au **gateway**, au cas où le patient serait éloigné du dispositif.

Pour compléter la surveillance des paramètres de santé, nous avons opté pour le **tensiomètre Omron Blood Pressure Sensor**. Ce tensiomètre, qui peut se connecter via **Bluetooth** à notre Raspberry Pi, permet de mesurer la **pression systolique** et **diastolique**, ainsi que le **rythme cardiaque**, offrant ainsi une surveillance complète de la pression artérielle du patient.

En prenant en compte les capteurs intégrés,une entête MQTT de 3 octets, le format des données transmises, et une fréquence d'envoi toutes les 5 minutes, nous pouvons estimer le volume de données suivant par jour pour les paramètres envoyés en continus :

Paramètre	Taille par envoi (en octets)	Volume par jour (en Ko)
Température (celsius)	7	2,02
Fréquence cardiaque(2 octets)	5	1,44
Fréquence respiratoire (5 octets)	4	1,15
ECG (10 octets)	13	3,74
SpO2 (%)	4	1,15
Total	-	9,79

Table 9.

Soit 100 Mo pour 10000 maisons ce qui est relativement faible.

7. Choix des technologies

1. Scan Watch Nova Brillant

- **Utilisation** : Montre connectée pour la collecte des données de santé, comme la fréquence cardiaque, la saturation en oxygène, et la température.
- **motivation** : Ces montres permettent une collecte continue et non invasive des données de santé essentielles. Leur précision et leur capacité à se connecter facilement aux réseaux facilitent l'intégration dans un système de monitoring de santé, surtout pour une surveillance à distance et en temps réel.

2. Raspberry Pi

- **Utilisation** : Traitement en bordure (edge computing) des données capturées par les capteurs, avec des fonctionnalités de réduction de bruit, compression, et stockage temporaire.
- **motivation** : Le Raspberry Pi est un ordinateur compact et peu coûteux, idéal pour le cas de notre projet. L'edge computing permet de traiter les données localement, réduisant la latence et la bande passante nécessaire pour la transmission des données. De plus, la réduction de bruit et la compression des données diminuent la charge de traitement sur le backend et augmentent la rapidité des réponses.

3. Protocole MQTT avec TCP/TLS

- **Utilisation** : Transmission sécurisée et en temps réel des données de santé des appareils connectés vers le backend.
- **motivation** : MQTT est un protocole léger de publication/souscription, très adapté pour les environnements à faible bande passante et les applications IoT. Le support de TCP/TLS garantit la sécurité des données en transit, essentielle pour protéger les informations de santé des utilisateurs.

4. Kafka

- **Utilisation** : Gestion des événements inter-services dans le backend, permettant un flux de données continu et asynchrone.
- **Motivation** : Kafka est conçu pour gérer de grands volumes de données en temps réel avec une latence faible. Il est idéal pour les architectures microservices, car il permet aux services de communiquer de manière asynchrone, assurant la résilience du système en cas d'augmentation de la charge ou de pannes de certains services.

5. MongoDB

- **Utilisation** : Base de données pour stocker les données temporelles des patients.
- **Motivation** : MongoDB est une base de données NoSQL qui gère efficacement les données non structurées et volumineuses, typiques des applications de santé qui collectent des données

continues à haute fréquence. Elle est également hautement scalable et capable de stocker les données des capteurs en format JSON, facilitant la manipulation de données complexes et non standardisées.

6. Architecture Micro-services avec Spring Boot

- **Utilisation** : Architecture avec des micro-services spécialisés, comme le Patient Service et le Measurement Service.
- **Motivation** : L'architecture microservices permet de diviser l'application en modules indépendants, chacun gérant une fonction spécifique. Spring Boot simplifie le développement des microservices grâce à une gestion facile des dépendances, permettant de déployer rapidement des services modulaires et de les faire évoluer indépendamment. Cette architecture améliore également la résilience, car un service peut être mis à jour sans impacter le reste du système.

7. Docker et Kubernetes

- **Utilisation** : Docker pour la conteneurisation des microservices et Kubernetes pour l'orchestration, auto-scaling, load balancing, et déploiement.
- **Motivation** : Docker garantit la portabilité et la cohérence des applications en encapsulant les microservices avec leurs dépendances. Kubernetes gère efficacement le déploiement de ces conteneurs sur un cluster, assurant l'auto-scaling et la gestion de la charge en fonction des besoins, une fonctionnalité cruciale pour une plateforme de santé en temps réel où la scalabilité et la disponibilité sont essentielles.

8. Grafana

- **Utilisation** : Visualisation et analyse en temps réel des données de santé.
- **Motivation** : Grafana est une solution populaire pour la visualisation de données en temps réel. Elle permet de créer des tableaux de bord adaptés aux besoins des différents utilisateurs (patients, personnel médical, administration). La visualisation immédiate des données de santé permet de réagir rapidement aux alertes et de surveiller les tendances de santé de manière proactive.

9. Jenkins et Helm

- **Utilisation** : Jenkins pour les pipelines CI/CD, et Helm pour le déploiement et la gestion des configurations Kubernetes.
- **Motivation** : Jenkins automatise les pipelines d'intégration et de déploiement continus (CI/CD), réduisant les risques d'erreurs lors des mises à jour et déploiements. Helm simplifie la gestion des configurations Kubernetes, rendant le déploiement plus rapide et moins sujet aux erreurs. Cela est essentiel pour garantir que les mises à jour de l'application se déroulent sans interruption de service.

10. Réseau GSM avec Carte SIM

- **Utilisation** : Connectivité pour le dispositif de détection de chutes, permettant l'envoi d’alertes même en extérieur.
- **Motivation** : Le réseau GSM permet au dispositif de détection de chutes d’envoyer des alertes même sans connexion Wi-Fi ou hors de portée de la passerelle locale. Cette connectivité est cruciale pour les utilisateurs en mobilité, assurant qu’ils peuvent toujours signaler une alerte en cas d’urgence.

Chaque technologie est donc choisie pour maximiser la performance, la scalabilité et la fiabilité du système, en répondant aux besoins critiques d'une solution de santé en temps réel.

8. Comment contribuer à ce projet

Pour faciliter la création de nos microservices, nous avons mis en place un template réutilisable. Ce template est un projet Spring Boot accompagné d'une configuration Docker et Kubernetes, pensé pour être cloné à chaque fois qu'un nouveau microservice doit être créé. Voici comment nous procédons :

Clonage du Template

Lors de la création d'un nouveau micro service, nous clonons le template depuis notre dépôt GitHub, puis nous ajustons les références nécessaires, notamment en modifiant les liens vers les dépôts GitHub, les identifiants de conteneur, et les variables spécifiques au micro service en question.

Contenu du Template

Projet Spring Boot : Ce projet constitue le cœur du micro service, incluant la structure de base, les dépendances, et les configurations par défaut.

Dockerisation : Le template contient un Dockerfile pour créer l'image du microservice ainsi qu'un fichier docker-compose.yml qui met en place l'environnement de développement. Ce docker-compose associe le conteneur du micro service à un conteneur MongoDB, simplifiant ainsi la mise en place des bases de données pour les environnements de développement et de test.

Déploiement Kubernetes avec Helm : Le template inclut également un chart Helm qui permet de déployer le microservice sur un cluster Kubernetes. Ce chart gère la configuration des pods, la liaison avec MongoDB, et les paramètres de déploiement afin de standardiser l'intégration du service dans notre architecture cloud native.

Création d'un Nouveau Projet

Pour créer un nouveau micro service, nous pouvons soit cloner ce template en entier, soit utiliser ce dernier comme modèle de base pour initier un projet. Cela garantit que tous les microservices partagent une structure et une configuration cohérentes, ce qui facilite la maintenance et le déploiement sur un cluster Kubernetes.

9. Matrice d’analyse de risque

Risque	Description	Catégorie	Impact	Probabilité	Niveau de risque	Plan de Mitigation	Actions de suivi
Défaillance de la montre ScanWatch	Les capteurs Hexiwear peuvent cesser de fonctionner ou envoyer des données erronées.	Opérationnel	Élevé	Moyenne	Élevé	Implémenter une détection automatique des erreurs de capteurs et notifier les utilisateurs. Prévoir un processus de remplacement rapide des capteurs défectueux.	Vérifier régulièrement la qualité des données provenant des capteurs.

Perte de connexion FOG-Backend	Problèmes de connexion empêchant le transfert des données en temps réel vers le backend.	Technique	Élevé	Moyenne	Élevé	Utiliser une carte microSD pour stocker temporairement les données en cas de coupure. Reconnecter et synchroniser les données dès que la connexion est rétablie.	Contrôler la stabilité du réseau et planifier des sauvegardes locales.
Latence élevée dans les alertes critiques	Retard dans la réception des alertes critiques pour les médecins/infirmières.	Sécurité des patients	Élevé	Moyenne	Élevé	Prioriser les alertes critiques via Kafka, en utilisant des messages de priorité élevée. Optimiser le réseau pour réduire la latence.	Surveiller les délais d'alerte avec un rapport hebdomadaire.
Vulnérabilité des données de santé	Les données de santé peuvent être compromises en cas d'accès non autorisé.	Sécurité	Très Élevé	Moyenne	Très Élevé	Chiffrement de bout en bout des données (en transit et au repos). Imposer une authentification multifactorielle (MFA) pour	Effectuer des tests de sécurité trimestriels pour identifier les failles.

						les utilisateurs sensibles.	
Panne de l'EMQX Broker Entreprise	Si le broker MQTT tombe en panne, la transmission des données de capteurs est interrompue.	Technique	Élevé	Faible	Moyen	Configurer un cluster EMQX redondant pour assurer la continuité. Mettre en place une solution de basculement automatique.	Superviser l'état des clusters et vérifier la redondance des messages.
Scalabilité insuffisante du backend	Difficulté à traiter un volume élevé de données en temps réel avec l'augmentation du nombre d'utilisateurs.	Technique	Élevé	Moyenne	Élevé	Utiliser Kubernetes pour l'auto-scaling et migrer vers une architecture cloud-native élastique. Prévoir une répartition de charge dynamique.	Effectuer des tests de charge mensuels pour garantir la scalabilité.
Erreur humaine dans la gestion des utilisateurs	Mauvaise configuration ou suppression accidentelle d'utilisateurs pouvant affecter les accès.	Organisationnel	Moyen	Moyenne	Moyen	Limiter les droits d'accès pour les actions sensibles et intégrer des confirmations pour les actions critiques. Prévoir des sauvegardes	Former régulièrement les administrateurs sur les bonnes pratiques.

						automatiques des configurations.	
Non-conformité aux réglementations de santé (GDPR, HIPAA)	Risque de non-conformité aux lois de protection des données de santé, avec des amendes potentielles.	Réglementaire	Très Élevé	Faible	Élevé	Mettre en place une équipe dédiée à la conformité et des audits de conformité réguliers. Utiliser des outils de monitoring pour suivre les accès aux données sensibles.	Auditer chaque trimestre la conformité aux régulations en vigueur.
Problème de fiabilité du stockage dans InfluxDB	Risque de perte de données si InfluxDB tombe en panne ou atteint sa capacité maximale.	Technique	Élevé	Moyenne	Élevé	Configurer des sauvegardes régulières et des répliques pour assurer la redondance. Migrer vers une base de données scalable pour les séries temporelles si besoin.	Vérifier régulièrement la taille des sauvegardes et la redondance.

Manque de formation des utilisateurs finaux	Les utilisateurs (patients, médecins, infirmières) peuvent avoir des difficultés à utiliser l'application.	Utilisateur	Moyen	Moyenne	Moyen	Organiser des sessions de formation et fournir un manuel d'utilisation détaillé. Mettre en place un support client réactif pour répondre aux questions des utilisateurs.	Évaluer les retours des utilisateurs pour améliorer la formation ou le manuel
Défaillance du système de sauvegarde	Risque de perte des données sensibles et historiques en cas de panne de sauvegarde.	Technique	Élevé	Faible	Moyen	Mettre en place un système de sauvegarde automatique et vérifier périodiquement l'intégrité des sauvegardes. Prévoir une restauration de secours en cas de panne.	Effectuer des tests de restauration de sauvegardes chaque trimestre.
Perte de connexion avec le cloud pour les données en temps réel	Si la connexion au cloud est perdue, les données ne seront pas disponibles en temps réel.	Technique	Élevé	Moyenne	Élevé	Permettre un stockage local temporaire sur la Raspberry Pi avec synchronisation automatique	Contrôler la connectivité cloud et ajouter des alertes en cas de perte.

						dès la reprise de connexion.	

Matrice de risques

10. Vue globale de l'architecture

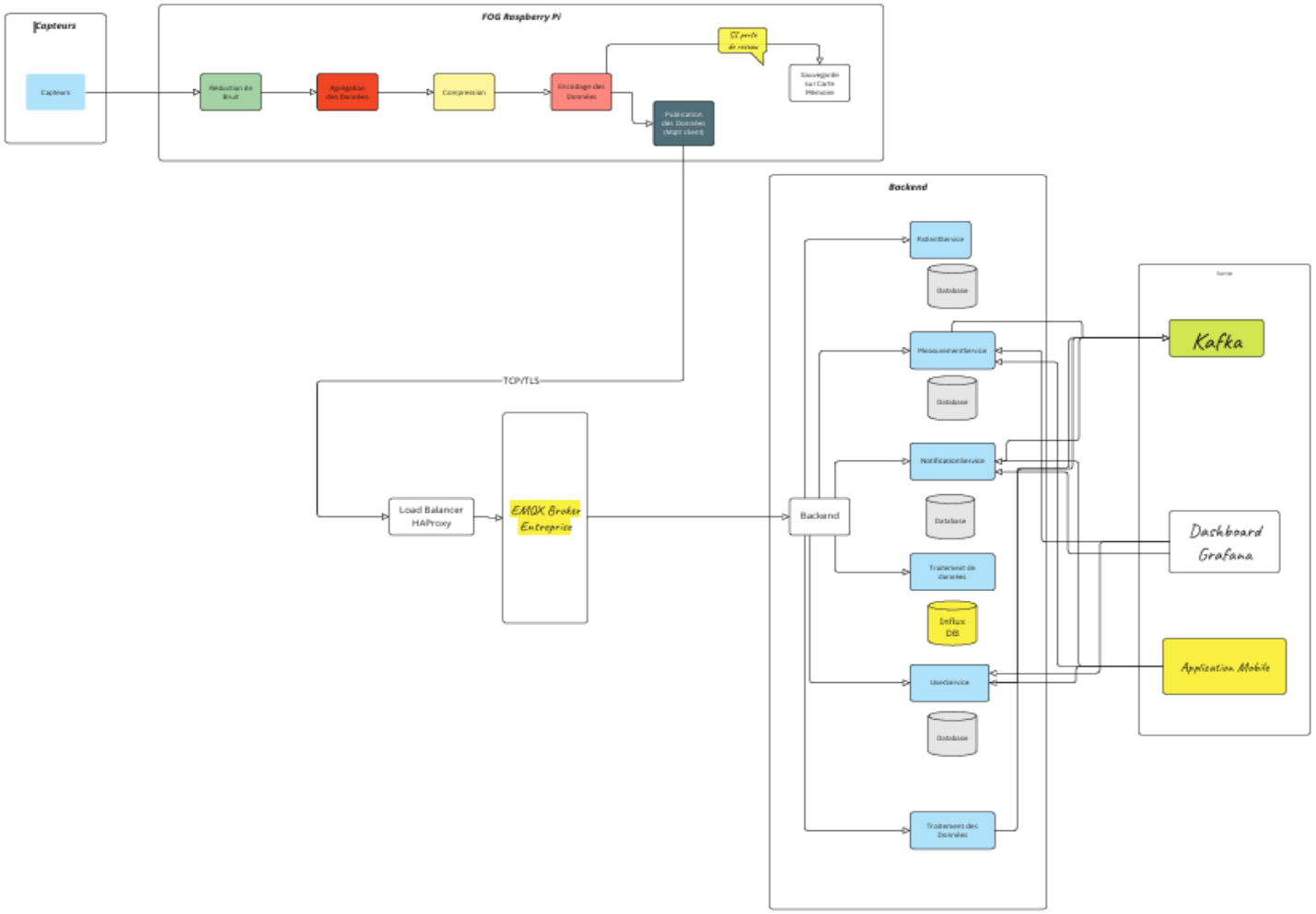

Figure 22.