Dossier CDA de projet LoraWan

SEPTEMPRE 2025

Homeeguard
Créé par : Baptiste Lucbert

homeeguard

Sommaire du Dossier de Projet

Liste des compétences mises en œuvre

1. Installer et configurer son environnement de travail en fonction du projet

✔ Mise en œuvre.

J'ai installé et configuré l'environnement de développement nécessaire pour le projet : Arduino IDE / PlatformIO pour le code ESP32, configuration du réseau Wi-Fi et LoRa, installation du framework Symfony pour l'API ainsi que MySQL pour la base de données. Les outils de gestion de versions et de dépendances ont été utilisés pour assurer un environnement stable.

2. Développer des interfaces utilisateur

✔ Mise en œuvre.

Une interface utilisateur a été développée pour permettre le suivi et la gestion des équipements connectés.
Elle affiche les données transmises par les Bridges ESP32 (température, humidité, batterie, etc.) via l'API Symfony.
L'interface exploite les données JSON envoyées par les Bridges et permet de visualiser en temps réel l'état des dispositifs, leurs alertes et leur statut de connexion.
La charte graphique et l'ergonomie ont été respectées afin d'assurer une bonne lisibilité et une utilisation intuitive par l'administrateur ou le technicien.

3. Développer des composants métier

✔ Mise en œuvre.

Les composants métier du projet gèrent la logique principale : réception et déchiffrement des trames LoRa, extraction et traitement des données capteurs (température, humidité, mouvements, etc.), envoi sécurisé à l'API via HTTPS POST, et réponse automatique aux nodes. Le code applique les principes de programmation orientée objet et respecte les bonnes pratiques de structuration et de sécurité.

4. Contribuer à la gestion d'un projet informatique

✔ Mise en œuvre.

Le projet a été organisé selon un découpage clair en phases (analyse, conception, développement, tests, déploiement). Un diagramme de Gantt détaillé a été établi pour planifier les tâches et suivre l'avancement. Les étapes ont été documentées et ajustées en fonction des priorités techniques.

5. Analyser les besoins et maqueter une application

✔ Mise en œuvre.

Les besoins fonctionnels ont été identifiés (communication LoRa ↔ Wi-Fi, chiffrement AES, transfert de données vers API). Une maquette conceptuelle du flux de données (Node → Bridge → API → Base de données) a été réalisée pour définir les interactions et les champs nécessaires aux échanges JSON.

6. Définir l'architecture logicielle d'une application

✔ Mise en œuvre.

L'architecture du système repose sur plusieurs couches :

- Couche matérielle : modules ESP32.
- Couche communication : réseau LoRa et Wi-Fi.
- Couche applicative : chiffrement, parsing et transfert de données.

- Couche serveur : API Symfony et base MySQL.
  Chaque rôle est défini et l'ensemble respecte une architecture en couches bien structurée.

7. Concevoir et mettre en place une base de données relationnelle

✔ Mise en œuvre.

Une base de données MySQL a été conçue et implémentée à partir du script SQL fourni. Elle gère les entités principales : comptes, utilisateurs, domiciles, produits, équipements et leurs logs. Le schéma relationnel garantit la cohérence, la sécurité et l'intégrité des données.

8. Développer des composants d'accès aux données SQL et NoSQL

⚙ Partiellement mise en œuvre.

Les composants d'accès SQL ont été développés via l'API Symfony pour lire, insérer et mettre à jour les données liées aux équipements et aux logs. Aucun système NoSQL n'a été utilisé dans ce projet.

9. Préparer et exécuter les plans de tests d'une application

⚙ Partiellement mise en œuvre.

Des tests unitaires et fonctionnels manuels ont été effectués : vérification des connexions Wi-Fi/LoRa, validation du chiffrement AES, test de l'envoi HTTP POST vers l'API et analyse des retours JSON. Les résultats ont permis de valider la stabilité de la communication, bien que les tests automatisés ne soient pas mis en place.

10. Préparer et documenter le déploiement d'une application

⚙ Partiellement mise en œuvre.

Le déploiement local du framework Symfony et du Bridge ESP32 a été documenté. Les étapes d'installation et de configuration ont été décrites, mais aucune procédure automatisée de déploiement (CI/CD) n'a été réalisée.

11. Contribuer à la mise en production dans une démarche DevOps

✔ Mise en œuvre.

Le projet a intégré une démarche DevOps basique pour assurer la mise en production et la supervision.
Les scripts d'installation et de configuration du Bridge ESP32 et du framework Symfony ont été automatisés en partie, permettant un déploiement rapide de l'environnement.
Des outils de test et de suivi (logs série, retours API, surveillance réseau) ont été utilisés pour vérifier la stabilité du système avant et après la mise en service.
Cette approche facilite la maintenance et le déploiement de nouvelles versions du firmware ou de l'API.

Expression des besoins

Keolis, opérateur majeur de réseaux de transport publics, fait face depuis plusieurs années à des tentatives récurrentes de vol de cuivre sur certains de ses sites techniques.
Ces actes entraînent des pannes de signalisation, des interruptions de service et des coûts de réparation élevés, sans compter les risques de sécurité pour le personnel et les usagers.

Face à cette problématique, Keolis a sollicité la société Homeeguard afin de concevoir une solution de détection préventive, capable d'alerter rapidement en cas d'intrusion, de tentative de découpe ou de déplacement anormal de câbles ou d'équipements.

Objectif général du projet :

Le besoin exprimé par Keolis se traduit par la mise en place d'un système connecté et autonome permettant :

- De détecter rapidement toute anomalie sur les installations sensibles (tentative de vol, ouverture de coffret, rupture de câble, variation anormale de tension, etc.)
- D'alerter en temps réel les équipes techniques via une plateforme centralisée (tableau de bord web et alertes SMS/email)
- De conserver un historique des alertes et événements pour analyse et suivi des incidents
- De réduire la dépendance au réseau électrique et assurer la communication même en zone isolée
- De proposer une solution adaptable à d'autres usages (détection environnementale, incendie, intrusion, maintenance préventive, etc.)

Expression des besoins fonctionnels :

Les besoins exprimés par Keolis lors de l'audit peuvent être résumés de la façon suivante :

1. Surveillance des zones sensibles
    - Couvrir les sites exposés au risque de vol de cuivre (locaux techniques, armoires électriques, lignes souterraines).
    - Possibilité d'ajouter ou retirer des capteurs selon les besoins du site.
2. Transmission fiable et indépendante du réseau filaire
    - Utiliser une communication sans fil longue portée (LoRa) adaptée aux environnements industriels.
    - Garantir une autonomie électrique suffisante pour une surveillance continue, même en cas de coupure de courant.
3. Alerte immédiate en cas d'incident
    - Notification instantanée à la cellule de maintenance et au responsable de site.
    - Détails de l'alerte (type d'événement, localisation, heure, gravité).

- Possibilité de filtrer les alertes par priorité ou zone géographique.

4. Plateforme de supervision unifiée
    - Un tableau de bord ergonomique permettant de visualiser en temps réel l'état des installations, les alertes, les statistiques et la cartographie des capteurs.
    - Accès sécurisé via navigateur web ou mobile.
    - Outils de suivi : historique, rapports, filtres, export de données, statistiques d'activité.
5. Maintenance et évolutivité
    - Le système doit être simple à installer, peu coûteux à entretenir et facilement extensible à d'autres usages : intrusion, température, humidité, ou suivi énergétique.
    - L'architecture doit permettre l'ajout de nouveaux capteurs sans reconfiguration lourde.

Contraintes identifiées

- Environnement industriel : perturbations radio, distances importantes entre capteurs et passerelle.
- Sécurité des données : nécessité d'un chiffrement de bout en bout et d'un hébergement conforme RGPD.
- Autonomie énergétique : fonctionnement même en absence d'alimentation secteur.
- Interopérabilité : compatibilité avec d'autres systèmes de supervision déjà utilisés par Keolis.
- Simplicité d'usage : la solution doit être exploitable par des techniciens non informaticiens.

Vision du client

Keolis souhaite disposer d'un outil de surveillance intelligent, fiable et évolutif, capable d'anticiper les incidents et de réduire drastiquement les pertes financières et matérielles liées aux vols de cuivre. L'entreprise voit dans cette solution un investissement stratégique permettant :

- De sécuriser ses infrastructures critiques,
- De centraliser la supervision de plusieurs sites,
- Et d'offrir un modèle reproductible sur d'autres applications de détection.

Présentation de l’entreprise et du service

1. Présentation de l'Entreprise Homeeguard

Homeeguard est une jeune entreprise française basée à Cormeilles-en-Parisis en Île-de-France, spécialisée dans la sécurité résidentielle et professionnelle. Elle se positionne comme un acteur moderne proposant des solutions de caméras de surveillance et de télésurveillance flexibles et axées sur l'utilisateur.

2. Services et Proposition de Valeur Unique

Homeeguard se distingue par un modèle innovant qui offre une liberté significative à ses clients :

- Vente d'équipement en pleine propriété : Contrairement à de nombreux concurrents, Homeeguard vend ses systèmes de sécurité (caméras, détecteurs, centrale d'alarme) aux clients qui en deviennent propriétaires. Cela élimine les frais mensuels récurrents pour le matériel et permet aux clients de continuer à utiliser leur équipement même sans abonnement.
- Télésurveillance flexible et sur mesure : L'entreprise propose des abonnements de télésurveillance sans engagement, avec des options mensuelles, annuelles ou même des forfaits "vacances". Cette flexibilité permet aux clients d'adapter le service à leurs besoins réels.
- Levée de doute à distance : Homeeguard mise sur une levée de doute rapide et efficace à distance en cas d'alerte, évitant ainsi le déplacement systématique d'agents, ce qui contribue à la rapidité d'intervention et à la maîtrise des coûts.
- Services d'installation et de raccordement : Pour ceux qui préfèrent ne pas installer eux-mêmes, Homeeguard propose un service d'installation professionnel. Ils offrent également un service de raccordement pour les clients qui installent leur matériel.
- Maintenance : Des services de maintenance sont disponibles pour assurer la longévité et le bon fonctionnement des systèmes.

3. Clientèle Cible

Homeeguard s'adresse principalement aux particuliers souhaitant sécuriser leur domicile mais également aux professionnels souhaitant sécuriser leurs commerces.

4. Technologies

Leur fournisseur est HIK Vision, une entreprise assez réputée pour leur caméra de qualité. L'approche de Homeeguard est axée sur la télésurveillance connectée et la levée de doute vidéo, suggère l'utilisation de systèmes de vidéosurveillance modernes et intégrés.

5. Services

Hommeguard étant une startup compte 5 employés.
On peut compter 3 services :

- Service Commercial
- Service Maintenance
- Service Développement

Nous sommes 2 dans le service développement :

- Mon tuteur d'alternance : Alexandre Cassez (également directeur de Homeeguard)
- Moi-même (Baptiste Lucbert)

6.Mon Poste de Travail chez Homeeguard

En tant que concepteur développeur d'application chez Homeeguard, mon environnement de travail est conçu pour optimiser la productivité et la collaboration, tout en étant à la pointe des technologies de développement.

6.1 Matériel

Tous les développeurs chez Homeeguard travaillent sur des Mac. Ce choix offre un écosystème robuste et performant, apprécié pour sa stabilité, sa sécurité et son intégration harmonieuse avec les outils de développement modernes. Les Mac fournissent la puissance de calcul nécessaire pour gérer des projets complexes et les multitâches quotidiennes.

6.2 Logiciels Utilisés

Pour mener à bien nos missions de développement, nous nous appuyons sur une suite de logiciels essentiels :

- Visual Studio Code (VS Code) : C'est notre éditeur de code principal. VS Code est léger, rapide et extrêmement personnalisable, avec un vaste écosystème d'extensions qui facilitent le développement dans divers langages et frameworks. Il nous permet de coder efficacement, de déboguer et de gérer nos projets.
- Arduino IDE : Indispensable pour tout ce qui touche à l'intégration matérielle et aux objets connectés. Arduino IDE nous permet de programmer les microcontrôleurs et de développer des firmwares pour les dispositifs de sécurité, assurant l'interaction entre le matériel et nos applications.
- Postman : Cet outil est crucial pour le développement et le test des API (Interfaces de Programmation d'Applications). Postman nous aide à simuler des requêtes HTTP, à inspecter les réponses et à valider le bon fonctionnement de nos services backend, garantissant une communication fluide entre nos applications et les systèmes de Homeeguard.
- MAMP : En tant qu'environnement de serveur local, MAMP est utilisé pour le développement web. Il nous permet de simuler un serveur web (Apache), une base de données (MySQL) et d'exécuter des scripts PHP directement sur nos Mac, ce qui est idéal pour développer et tester nos applications web en local avant le déploiement.

7. Outils de Communication et Collaboration

La communication fluide est essentielle pour le travail d'équipe. Nous utilisons les logiciels de communication de Microsoft :

- Microsoft Teams : C'est notre plateforme centrale pour la collaboration. Teams nous permet de communiquer instantanément par chat, de participer à des réunions vidéo, de partager des fichiers et de gérer des projets en équipe, assurant que tout le monde reste synchronisé et informé.
- Outlook : Pour la gestion des emails professionnels, Outlook est notre outil de choix. Il nous aide à organiser nos communications, nos calendriers et nos contacts, facilitant la planification des tâches et le suivi des échanges importants.

Ce poste de travail complet et ces outils nous permettent de développer des solutions de sécurité innovantes et fiables pour Homeeguard, de la conception à la mise en œuvre.

8. Contact

L'entreprise est joignable par téléphone au 01 89 16 35 64 ou par email à contact@homeeguard.fr.

Gestion de projet

Le projet LoRaWAN, lancé le **10 janvier 2025**, est une initiative stratégique d'Homeeguard visant à minimiser les vols de cuivre dans les infrastructures urbaines et industrielles. Il s'agit de protéger des actifs critiques pour de grands opérateurs comme Keolis, la SNCF et la RATP. En tant que concepteur développeur d'application, j'ai activement participé à la mise en place de cette solution innovante.

1. Planning du Projet et Méthodologie de Suivi

Pour un projet d'une telle envergure, impliquant à la fois du développement matériel embarqué, de la communication réseau et une interface utilisateur web, nous avons adopté une méthodologie agile, basée sur **SCRUM**.

1.1 Diagramme de Gantt (Approche Simplifiée et Itérative)

Bien que nous utilisions SCRUM pour le suivi quotidien, un diagramme de Gantt initial a été établi pour poser les grandes lignes et les jalons majeurs du projet. Il sert de feuille de route générale et est ajusté à chaque sprint.

Diagramme de Gantt : Voir en annexe : ici (Gestion de projet)

1.2 Méthodologie SCRUM

Nous utilisons la méthodologie SCRUM pour le suivi quotidien et l'adaptation du projet. Cela nous permet de :

- **Réactivité :** Répondre rapidement aux changements de besoins ou aux défis techniques.
- **Transparence :** Chaque membre de l'équipe a une visibilité claire sur l'avancement.
- **Amélioration Continue :** Grâce aux boucles de feedback régulières.

Nos rituels SCRUM incluent :

- **Daily Scrums (Stand-ups quotidiens) :** Réunions courtes chaque matin pour discuter de ce qui a été fait, ce qui sera fait et des éventuels obstacles.
- **Sprint Planning :** En début de chaque sprint (période de développement fixe, généralement 2 semaines), nous définissons les objectifs et les tâches à accomplir.
- **Sprint Review :** En fin de sprint, nous présentons le travail accompli aux 2 gérant du projet pour recueillir leurs retours.
- **Sprint Retrospective :** Une réunion interne pour l'équipe afin d'analyser ce qui a bien fonctionné et ce qui peut être amélioré pour le prochain sprint.

2. Relation avec les Collègues, Réunions et Outils

La collaboration est au cœur de notre succès sur ce projet technique et complexe.

- **Réunions Régulières :** En plus des rituels SCRUM, nous tenons des réunions techniques spécifiques (par exemple, pour discuter de l'architecture du réseau Mesh ou de l'optimisation de la base de données Symfony/Doctrine) chaque fois que nécessaire.

- **Échanges Constants :** La communication est très ouverte, favorisant les échanges informels et l'entraide entre développeurs, qu'il s'agisse de problèmes liés à l'ESP32, au réseau LoRa, ou au développement web.
- **Outils de Communication :**
    - **Microsoft Teams :** Essentiel pour les discussions instantanées, le partage de documents, les appels audio/vidéo et l'organisation des canaux de discussion par thématique (ex: #hardware_lora, #dev_web, #deployments).
    - **Outlook :** Utilisé pour les communications plus formelles, la planification des réunions et la gestion des agendas.
- **Gestion de Code :** Nous utilisons l'outil OneDrive dans Teams qui permet de partager notre version du code avec toute l'équipe.

3. Objectifs Qualité et Normes

La fiabilité et la robustesse sont primordiales pour un système de sécurité comme celui-ci, surtout face au vol de cuivre. Nos objectifs qualité intègrent plusieurs dimensions :

- **Fiabilité du Matériel :** Les modules ESP32 et leurs capteurs sont sélectionnés et testés pour résister aux conditions environnementales (température, humidité) et garantir une détection précise et constante des événements (ouverture de trappes, vibrations liées aux tentatives de vol).
- **Stabilité du Réseau LoRa Mesh :** L'architecture réseau Mesh doit assurer une transmission de données fiable même sur de longues distances et dans des environnements complexes, avec une tolérance aux pannes (si un Node est trop éloigné, il peut relayer via un autre).
- **Précision des Détections :** Les algorithmes embarqués sur les Nodes (gyroscopes, détecteurs d'ouverture) sont calibrés pour minimiser les fausses alertes tout en assurant une détection rapide des tentatives de vol.
- **Sécurité des Données :** La transmission des données LoRa est sécurisée. La connexion du Bridge à Internet (4G/Wi-Fi) et l'accès au site web client sont également sécurisés (HTTPS, bonnes pratiques de développement Symfony/Doctrine pour prévenir les injections SQL et autres vulnérabilités).
- **Qualité du Logiciel Web :** Le site internet Symfony est développé en respectant les bonnes pratiques de codage, avec une architecture robuste via l'ORM Doctrine. L'interface utilisateur est conçue pour être intuitive et offrir une visualisation claire et en temps réel de l'état des ESP et des statistiques.
- **Maintenance et Scalabilité :** Le système est conçu pour être facilement maintenable et scalable, afin d'intégrer de nouveaux modules ou de déployer la solution à grande échelle.

Spécifications fonctionnelles

Les spécifications fonctionnelles détaillent les besoins techniques du projet LoRaWAN, transformant les exigences métier issues de l'audit (minimiser les vols de cuivre chez des clients comme Keolis, SNCF, RATP, etc…) en fonctionnalités concrètes pour notre solution. Elles décrivent ce que le système doit faire, sans dicter comment il doit le faire techniquement (c'est le rôle des spécifications techniques).

1. Détection des Incidents

Le système doit être capable de détecter les événements indiquant une tentative de vol ou une intrusion sur les infrastructures surveillées.

- **Détection d'ouverture de trappe :** Les Nodes ESP32 doivent détecter l'ouverture non autorisée de trappes (de poteaux, rigoles, etc.) et déclencher une alerte.
- **Détection de mouvement/vibration :** Les Nodes ESP32 équipés de gyroscopes doivent pouvoir identifier des mouvements ou vibrations anormaux suggérant une manipulation ou une tentative d'arrachage du cuivre.
- **Détection de variation de température :** Les Nodes ESP32 équipés de détecteurs de température doivent relever des changements anormaux de température qui pourraient indiquer une activité inhabituelle ou un début d'incendie lié au vol.
- **Transmission des alertes :** Chaque détection doit générer une trame d'alerte contenant le type d'événement, l'identifiant du Node et l'horodatage.

2. Communication Réseau Mesh LoRa

Le système doit assurer une transmission fiable et efficace des données d'alerte des points de détection jusqu'au point central de collecte.

- **Réseau Mesh LoRa :** Les ESP32 (Nodes et Bridges) doivent former un réseau maillé (Mesh) utilisant la technologie LoRa. Cela signifie que si un Node est trop éloigné du Bridge, il doit pouvoir retransmettre ses trames via un autre Node intermédiaire pour atteindre sa destination.
- **Fiabilité de la transmission :** Le réseau doit garantir une haute fiabilité dans la transmission des trames, minimisant les pertes de données même dans des environnements complexes (urbains, souterrains).
- **Portée étendue :** La technologie LoRa doit permettre une communication sur de longues distances, réduisant le nombre de Bridges nécessaires.

3. Collecte et Transmission des Données au Cloud

Le système doit collecter les données du réseau LoRa et les rendre disponibles en ligne pour les clients.

- **Réception des trames (Bridge) :** Le Bridge ESP32 doit recevoir toutes les trames envoyées par les Nodes.

- **Connectivité Internet :** Le Bridge doit se connecter à Internet via 4G ou Wi-Fi pour la transmission des données.
- **Envoi à la base de données :** Le Bridge doit transmettre les informations collectées à une base de données centrale (gérée par Doctrine).
- **Accès web client :** Les informations doivent être accessibles et affichées en temps réel sur le site internet Symfony dédié aux clients.

4. Interface Client Web (Site Internet Symfony)

Le site web doit offrir aux clients une visibilité complète et en temps réel sur l'état de leurs dispositifs de sécurité et les incidents.

- **Tableau de bord temps réel :** Les clients doivent pouvoir visualiser l'état de leurs ESP32 (Nodes et Bridges) en direct. Cela inclut leur statut de connexion, le niveau de batterie (si applicable), et l'historique des dernières communications.
- **Historique des événements :** Affichage d'un journal des alertes et détections passées, avec horodatage, type d'événement et Node concerné.
- **Statistiques :** Présentation de statistiques agrégées sur les détections, la fréquence des alertes par site ou par type d'événement, permettant une analyse des tendances.
- **Authentification sécurisée :** Accès au site client via un système de connexion sécurisé.
- **Interface utilisateur intuitive :** Le site doit être facile à utiliser et à naviguer, même pour des utilisateurs non techniques.

5. Gestion des Modules (Nodes & Bridges)

Le système doit permettre l'identification et le suivi de chaque module.

- **Identification unique :** Chaque Node et Bridge doit avoir un identifiant unique pour être distingué sur le réseau et dans la base de données.
- **Association client/site :** Chaque module doit être associé à un client et à un site d'installation spécifique.

Ces spécifications fonctionnelles servent de base au développement, garantissant que la solution finale répond précisément aux besoins de prévention du vol de cuivre de nos clients.

Contraintes du projet et livrables attendus

La réussite du projet LoRaWAN dépend de la prise en compte et de la gestion de diverses contraintes, qui influencent la conception, le développement et le déploiement de la solution. Parallèlement, des livrables clairs ont été définis pour jalonner l'avancement et la finalisation du projet.

1. Contraintes du Projet

Les contraintes se divisent en deux catégories principales : fonctionnelles et non-fonctionnelles.

1.1 Contraintes Fonctionnelles

Ces contraintes sont directement liées aux fonctionnalités que le système doit offrir.

- **Précision et Fiabilité de la Détection :** Les capteurs (détecteurs d'ouverture, gyroscopes, température) doivent avoir un seuil de détection et une précision suffisante pour identifier les tentatives de vol de cuivre sans générer un nombre excessif de fausses alertes.
- **Performance du Réseau Mesh LoRa :**
  - **Portée :** La portée des communications LoRa doit être suffisante pour couvrir les zones étendues des infrastructures (poteaux, rigoles, voies ferrées) des clients comme Keolis, SNCF, RATP, y compris dans des environnements potentiellement difficiles (urbains denses, souterrains partiels).
  - **Latence :** Les alertes critiques (tentatives de vol) doivent être transmises avec une latence minimale pour permettre une réaction rapide des équipes de sécurité.
  - **Tolérance aux pannes :** Le réseau Mesh doit pouvoir rerouter les communications si un Node ou un Bridge est temporairement hors service, assurant la continuité du service.
- **Disponibilité du Site Web Client :** Le site Symfony doit être disponible 24h/24 et 7j/7, avec une interruption minimale pour la maintenance, afin que les clients puissent consulter l'état de leurs dispositifs en temps réel.
- **Scalabilité :** La solution doit pouvoir supporter un nombre croissant de Nodes et de Bridges sans dégradation significative des performances, à mesure que de nouveaux sites clients sont équipés.
- **Facilité d'Utilisation (Site Client) :** L'interface utilisateur du site web doit être intuitive et facile à prendre en main pour les opérateurs et gestionnaires de site des clients, même s'ils ne sont pas techniciens.

1.2 Contraintes Non-Fonctionnelles

Ces contraintes définissent comment le système doit fonctionner, sa qualité, ses limites et son environnement.

- Contraintes Temporelles :
  - **Date de Lancement :** Le projet a été lancé le 10 janvier 2025, ce qui a défini un calendrier de développement et de déploiement initial ambitieux pour atteindre les objectifs fixés avec nos partenaires.
  - **Délais de Réaction aux Alertes :** Le temps entre une détection et la notification au client doit être optimisé pour permettre une intervention rapide et minimiser les pertes.

    - **Cycle de Vie du Produit :** Prévoir la durée de vie des composants (batteries des ESP32, durée de vie des capteurs) et la facilité de remplacement/maintenance.
- **Contraintes Légales et Réglementaires :**
    - **Réglementation Radio (LoRa) :** Respect des normes de puissance d'émission et de fréquences pour la technologie LoRaWAN dans les pays de déploiement (en France et Europe, respect de l'ETSI).
    - **Protection des Données (RGPD) :** Assurer la conformité avec le Règlement Général sur la Protection des Données (RGPD) concernant les données collectées (géolocalisation si applicable, identifiants des dispositifs, historiques d'événements), leur stockage, leur accès et leur sécurisation.
    - **Normes de Sécurité Électrique/Électronique :** Les dispositifs ESP32 et capteurs doivent respecter les normes de sécurité électrique pour un déploiement en extérieur ou dans des environnements industriels.
- **Contraintes Techniques :**
    - **Consommation Énergétique (Nodes) :** Les Nodes ESP32 doivent avoir une très faible consommation d'énergie pour maximiser l'autonomie sur batterie, compte tenu de leur emplacement souvent difficile d'accès.
    - **Résistance Environnementale :** Les boîtiers des Nodes et Bridges doivent être robustes et résistants aux intempéries (eau, poussière, variations de température) pour une utilisation en extérieur ou en milieu industriel (IP65 ou supérieur).
    - **Connectivité du Bridge :** La fiabilité de la connexion 4G ou Wi-Fi du Bridge est cruciale, y compris la gestion des pertes de signal et la reconnexion automatique.
    - **Intégration du Site Symfony :** Le site doit s'intégrer de manière fluide avec la base de données Doctrine et les services de collecte de données du Bridge.
- **Contraintes Budgétaires :** Le coût total de la solution (matériel, développement, déploiement, maintenance) doit rester compétitif pour les clients et respecter les budgets alloués par Homeeguard.
- **Contraintes de Sécurité (Cybersécurité) :**
    - **Sécurité des communications LoRa :** Chiffrement des trames LoRa pour empêcher l'interception ou la manipulation des données.
    - **Sécurité du Site Web :** Protection contre les attaques web courantes (XSS, injections SQL), gestion sécurisée des accès utilisateurs et des données sensibles.
    - **Accès Physique :** Les Nodes et Bridges doivent être conçus pour être discrets et difficiles à saboter physiquement.

2. Livrables Attendus

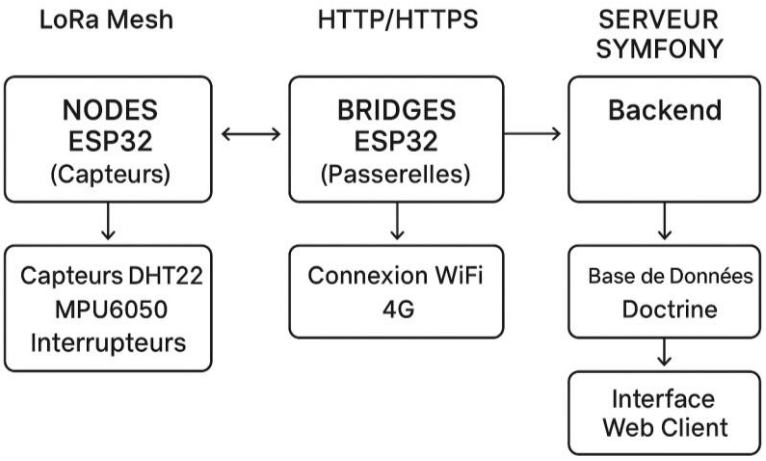
Les livrables sont les éléments concrets produits au cours du projet et remis aux parties prenantes.

- **Documentation de Conception :**
    - **Spécifications Fonctionnelles Détaillées :** Le document actuel, complété et validé.
    - **Spécifications Techniques :** Documentation des architectures matérielle (ESP32, capteurs) et logicielle (firmware LoRa, base de données, Symfony), des API, des protocoles de communication.
    - **Schémas Électroniques et Mécaniques :** Plans des circuits imprimés, des boîtiers pour les Nodes et Bridges.
- **Firmware des Modules ESP32 :**
    - **Firmware Node :** Code embarqué pour la gestion des capteurs, la détection des événements et la communication LoRa Mesh.

    - **Firmware Bridge :** Code embarqué pour la réception des trames LoRa et la transmission des données vers la base de données via 4G/Wi-Fi.
- **Application Web (Site Client) :**
    - **Code Source du Site Symfony :** L'intégralité du code source de l'application web développée sous Symfony/Doctrine.
    - **Base de Données Fonctionnelle :** Schéma de base de données et scripts de migration.
    - **Interface Utilisateur Opérationnelle :** Le site internet accessible et fonctionnel pour les clients.
- **Environnement de Développement et de Déploiement :**
    - **Environnement de Développement Local :** Configuration MAMP, Visual Studio Code avec les extensions nécessaires, environnement Arduino IDE.
    - **Scripts de Déploiement :** Scripts et configurations pour le déploiement du site web et des firmwares sur les serveurs et les modules.
- **Rapports de Tests :**
    - **Rapports de Tests Unitaires et d'Intégration :** Pour le firmware et l'application web.
    - **Rapports de Tests de Performance :** Sur la latence du réseau LoRa et la réactivité du site web.
    - **Rapports de Tests d'Autonomie :** Mesures de la durée de vie des batteries des Nodes.
- **Documentation Utilisateur et de Maintenance :**
    - **Manuel d'Utilisation Client :** Guide pour l'utilisation du site web client.
    - **Guide d'Installation des Modules :** Instructions pour le déploiement physique des Nodes et Bridges.
    - **Documentation de Maintenance :** Procédures pour la maintenance préventive et corrective des modules et du système.
- **Rapport de Projet :** Synthèse finale du projet, incluant les défis rencontrés, les solutions apportées et les leçons apprises.

Architecture logicielle du projet LoRaWan

1. Architecture Générale du Système :



2. Architecture Détaillée par Couches :

2.1 Couche Matérielle (Hardware Layer)

Composants :

- **Nodes ESP32** : Unités de capture avec capteurs
- **Bridges ESP32** : Passerelles de communication
- **Capteurs** : DHT22, MPU6050, interrupteurs magnétiques

Responsabilités :

- Acquisition des données physiques
- Prétraitement des signaux capteurs
- Gestion de l'alimentation et sleep modes

2.2 Couche Firmware (Embedded Software Layer)

Architecture des Nodes ESP32 :

├── CONFIGURATION
│ ├── Adresses réseau (NODE_ADDRESS, BRIDGE_ADDRESS)
│ ├── Paramètres RF (FREQUENCY, TX_POWER)
│ └── Deltas de détection
├── GESTION CAPTEURS
│ ├── DHT22 (Température/Humidité)

```
│   ├── MPU6050 (Accéléromètre/Gyroscope)
│   └── Interrupteurs magnétiques
├── COMMUNICATION LORA
│   ├── Chiffrement/déchiffrement AES
│   ├── Gestion réseau Mesh
│   └── Basculement automatique bridges
├── GESTION ÉNERGIE
│   ├── Deep Sleep
│   ├── Wake-up stratégique
│   └── Optimisation consommation
└── STOCKAGE LOCAL
    ├── Système de fichiers SPIFFS
    └── Logs et sauvegardes
```

Architecture des Bridges ESP32 :

```
├── ROUTAGE RÉSEAU
│   ├── Réception messages Nodes
│   ├── Transmission vers API
│   └── Transfert bridge secondaire
├── CONNECTIVITÉ INTERNET
│   ├── Connexion WiFi/4G
│   └── Gestion reconnexions
├── TRAITEMENT DONNÉES
│   ├── Décryptage messages
│   ├── Formatage JSON
│   └── Enrichissement métadonnées
└── SÉCURITÉ
    ├── Authentification API
    └── Chiffrement bout-en-bout
```

3. Couche Backend (Symfony Application)

Architecture Modulaire Symfony

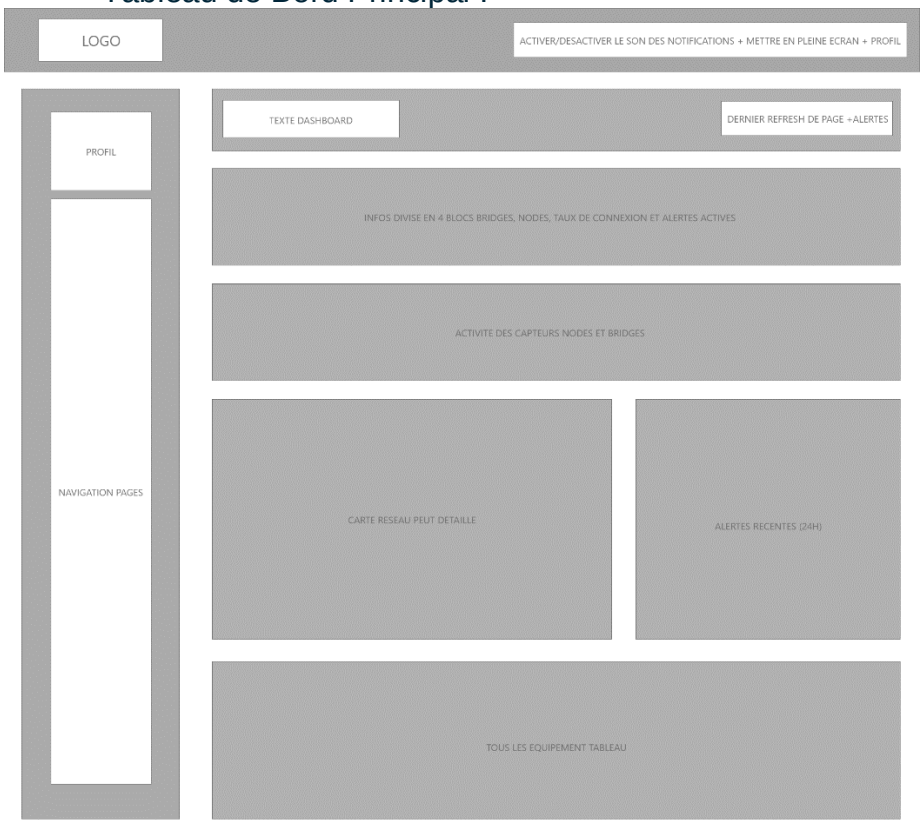
```
├── public
│   ├── js
│   │   ├── Mes fichiers relié aux fichiers Twig
├── src
│   ├── Controller
│   │   ├── ApiController.php
│   │   ├── EspaceClientController.php
│   ├── Entity
│   │   ├── Comptes.php
│   │   ├── Domicile.php
│   │   ├── EquipementAchetes.php
│   │   ├── EquipementAchetesLogs.php
│   │   ├── EquipementAchetesMetadata.php
│   │   ├── Product.php
│   │   ├── Utilisateur.php
```

```
│   ├── Repository
│   │   ├── ComptesRepository.php
│   │   ├── DomicileRepository.php
│   │   ├── EquipementAchetesRepository.php
│   │   ├── EquipementAchetesLogsRepository.php
│   │   ├── EquipementAchetesMetadataRepository.php
│   │   ├── ProductRepository.php
│   │   └── UtilisateurRepository.php
├── templates
│   ├── EspceClient
│   │   ├── Dashboard
│   │   │   ├── dashboard
│   │   │   │   └── accueil.html.twig
│   │   │   ├── esp
│   │   │   │   ├── alertes.html.twig
│   │   │   │   ├── cartographie.html.twig
│   │   │   │   ├── delta.html.twig
│   │   │   │   └── statistiques.html.twig
│   │   │   ├── support
│   │   │   │   └── faqs.html.twig
└── └── baseDashboardClient.html.twig
```

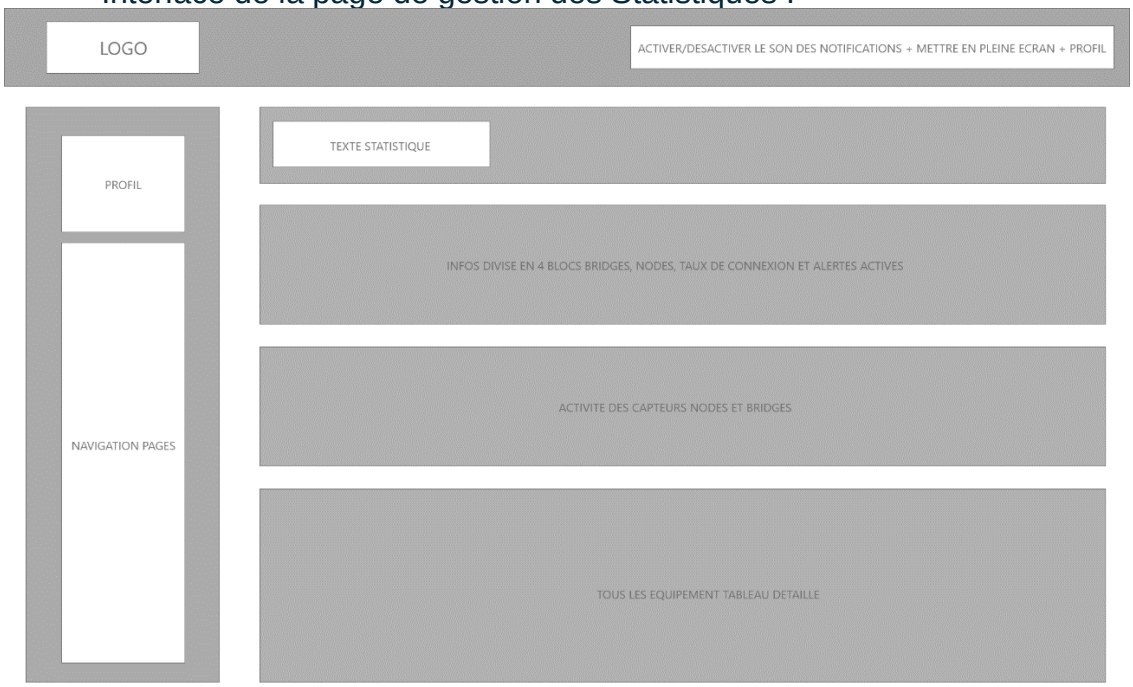
Prototypage

Maquettes principal des interfaces utilisateur :

Tableau de Bord Principal :



Interface de la page de gestion des Statistiques :



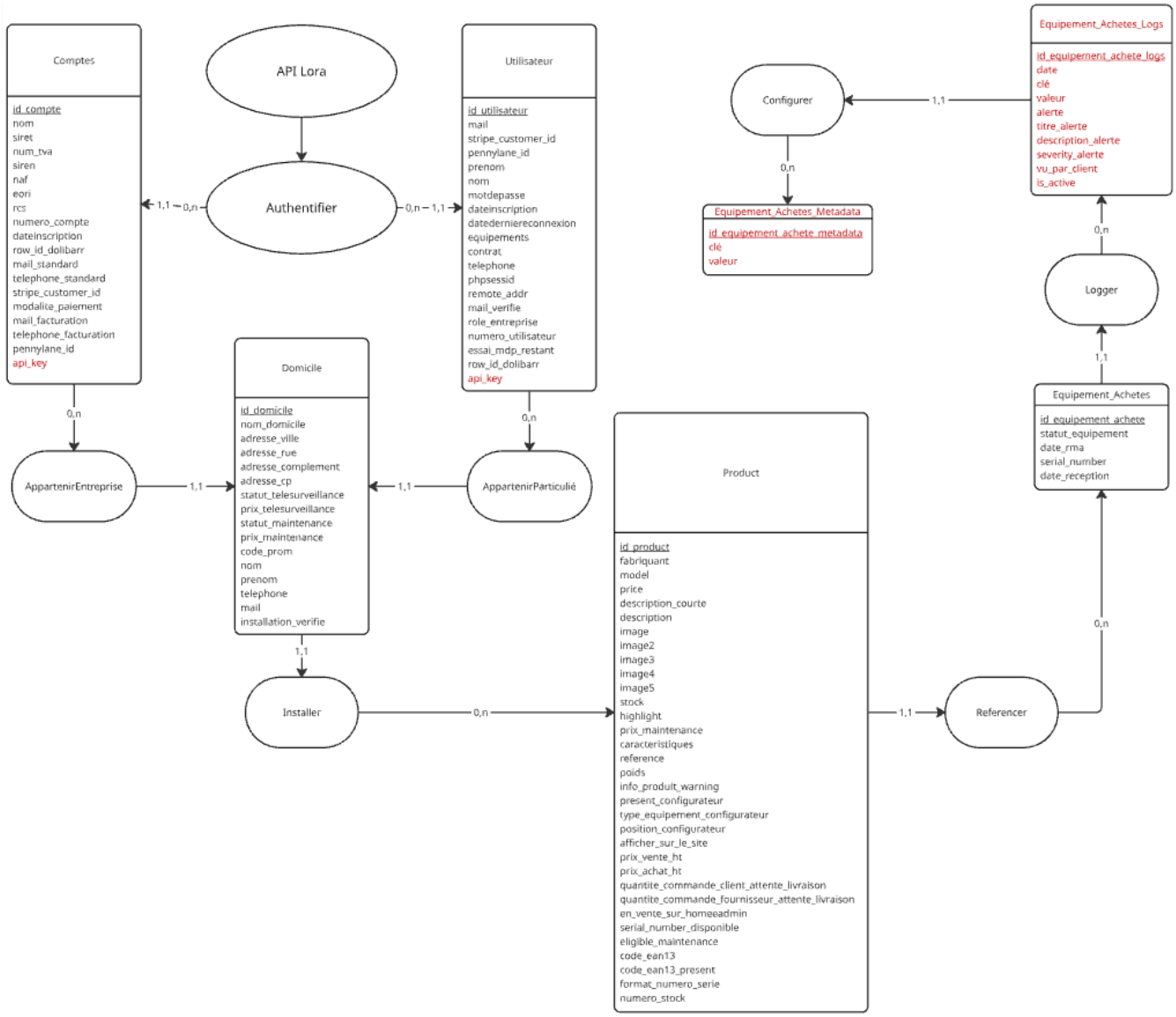
Vous pourrez retrouver tous les autres prototypes dans les annexes : ici (Annexes prototypage)

Conception de la base de données

Vous pouvez retrouver en annexe les MLD et le MPD de la conception de la base de données : ici (Annexes conception de la base de données)

ATTENTTION, les écritures rouges sont les valeurs/tables que j'ai créées pour mener à bien se projet

MCD :



Script de création ou de modification de la base de données

Le script SQL complet est disponible dans les annexes : ici (Annexes script de création ou de modification de la base de données)

Ce script SQL sert à initialiser la base de données du projet.
Il crée toutes les tables nécessaires pour :

- Gérer les **clients et utilisateurs**
- Gérer les **domiciles (installations physiques)**
- Gérer les **produits et équipements (ESP, capteurs, etc.)**
- Suivre l'**historique** et les **logs** de ces équipements.

Table comptes :

Rôle : Représente une entreprise ou un client professionnel.

Colonnes clés :

- id : identifiant unique du compte
- nom : nom du compte (ex : "Homeeguard")
- siret, siren, num_tva, naf, rcs : infos légales (identification entreprise)
- site_livraison_id, site_facturation_id : adresses principales du compte
- responsable_du_compte_id, createur_id : gestion interne (qui s'en occupe)
- stripe_customer_id : lien avec la facturation Stripe
- pennylane_id : lien avec la comptabilité (Pennylane)
- modalite_paiement, mail_facturation, etc. : gestion comptable

Table utilisateur

Rôle : Contient les **personnes physiques** associées à un compte.
Chaque utilisateur peut être un client, un installateur, un technicien, etc.

Colonnes clés :

- id : identifiant unique
- comptes_id : lien vers la table comptes
- domicile_livraison_id, domicile_facturation_id : domiciles liés
- prenom, nom, mail, motdepasse : identité et login
- telephone, dateinscription, datedernierereconnexion : infos générales
- equipements, contrat : gestion du matériel attribué
- stripe_customer_id, pennylane_id : liens financiers individuels
- role_entreprise : rôle dans la structure (client, admin, technicien...)
- api_key : pour authentification API (utilisation par un système externe ou ESP)

Table domicile

Rôle : Représente **le lieu physique** où sont installés les équipements ESP (la maison, le site surveillé, etc.).

Colonnes clés :

- id : identifiant du domicile
- entreprise_id : lien vers le compte ou installateur responsable
- contact_id, installateur_rattache_id : personnes associées
- nom_domicile, adresse_ville, adresse_cp : infos d'adresse
- statut_telesurveillance, prix_telesurveillance : services actifs
- statut_maintenance, prix_maintenance : contrats de maintenance
- installation_verifie : booléen, indique si l'installation a été validée techniquement
- mail, telephone : contact principal sur place

Table product

Rôle : Contient le **catalogue des produits disponibles** : capteurs, caméras, ESP, relais, alarmes, etc.

Colonnes clés :

- id : identifiant du produit
- fabriquant, model : ex. "Espressif", "ESP32-CAM"
- price, prix_vente_ht, prix_achat_ht : gestion tarifaire
- description, image1 à image5 : contenu pour affichage sur site web
- caracteristiques, reference, poids : détails techniques
- serial_number_disponible, format_numero_serie : gestion des numéros de série
- present_configurateur, type_equipement_configurateur : visible ou non sur configurateur produit
- eligible_maintenance : si le produit est suivi sous contrat maintenance

Table equipement_achetes

Rôle : Stocke **chaque équipement réellement acheté et installé**.
C'est **le lien entre un produit** (ex : ESP32) et **un domicile**.

Colonnes clés :

- id : identifiant de l'équipement
- produit_id : lien vers product (type d'appareil)
- domicile_installe_id : lien vers domicile
- statut_equipement : "installé", "en maintenance", "HS", etc.
- serial_number : numéro de série unique de l'appareil (ex : ESP SN12345)
- date_reception_client : date d'installation
- info_position : emplacement dans le domicile (ex : "Salon", "Garage")
- mot_passe_equipement : mot de passe local (pour l'ESP ou interface)
- remplace_par_id : si un appareil a été remplacé

Table equipement_achetes_logs

Rôle : Enregistre **les données remontées par les équipements (ESP)** :
états, alertes, mesures, anomalies…

Colonnes clés :

- equipement_achetes_id : lien vers equipement_achetes
- date : quand la donnée est enregistrée
- cle, valeur : clé/valeur (ex : “temperature = 22°C”, “batterie = 85%”)
- alerte, titre_alerte, description_alerte : gestion d’alertes techniques
- severity_alerte : niveau (info, warning, critical)
- vu_par_client : indique si le client a consulté l’alerte
- is_active : alerte toujours en cours ou non

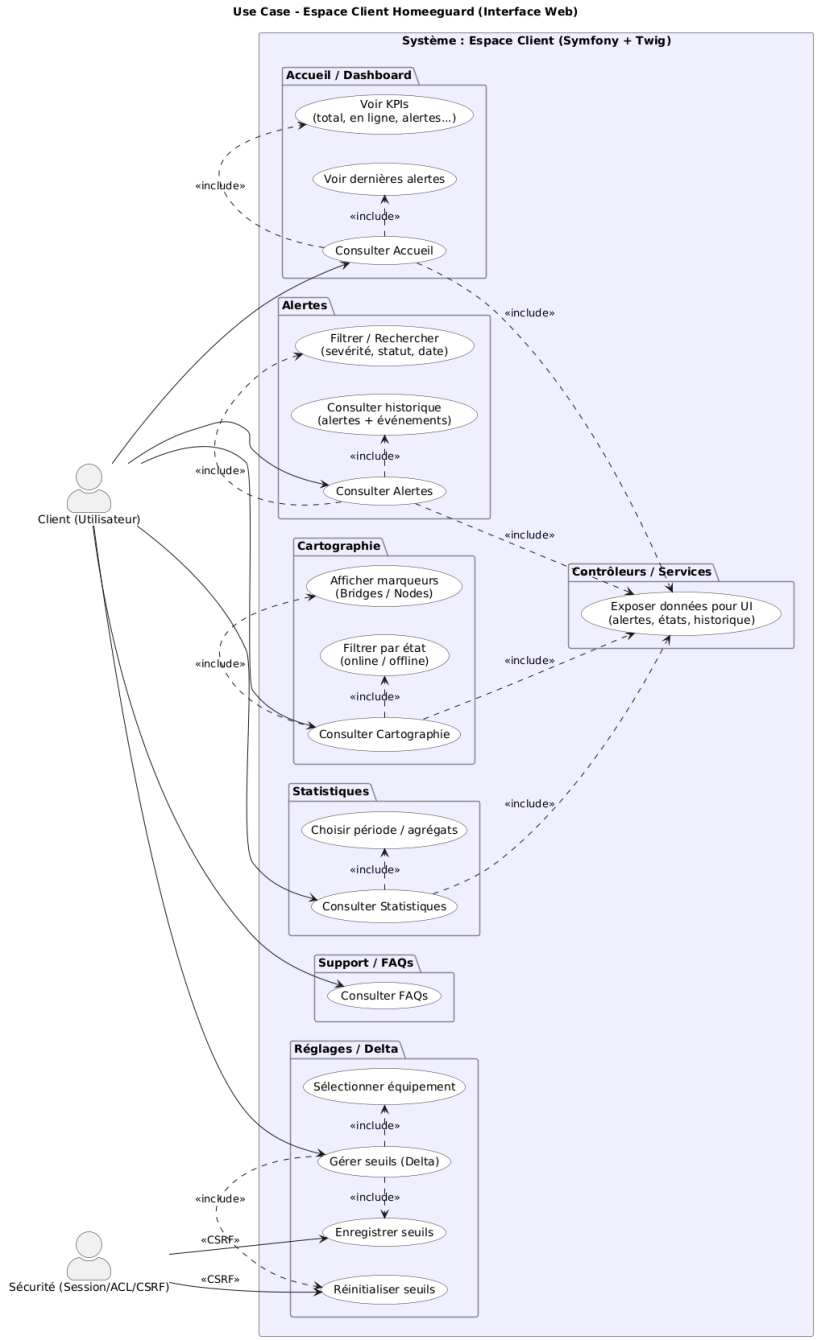
Table equipement_achetes_metadata

Rôle : Permet de stocker des **métadonnées supplémentaires** sur les équipements, non prévues à l’origine.

Colonnes clés :

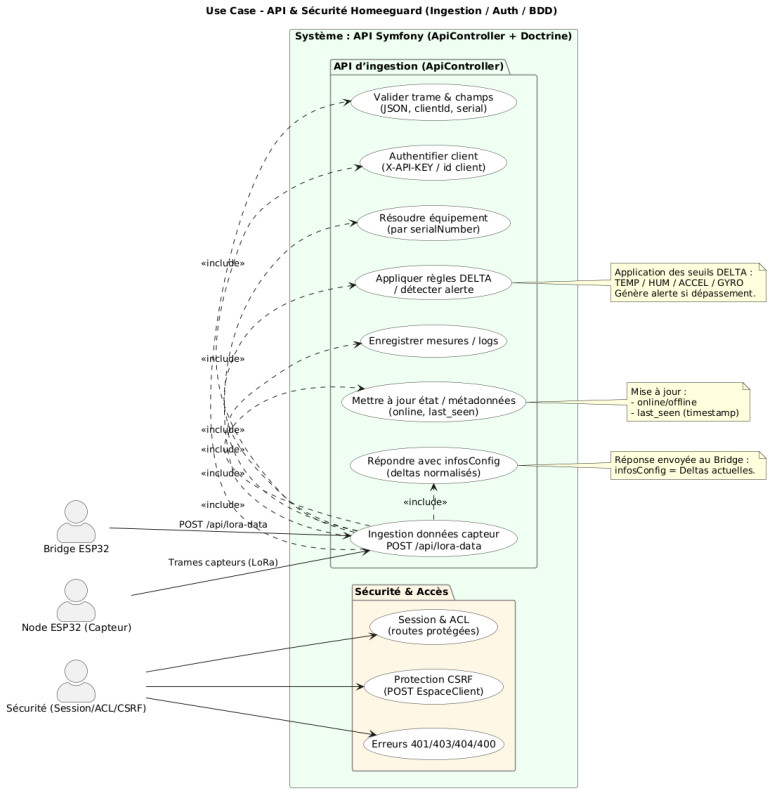
- cle, valeur : par exemple "interrupteur" = "1", "firmware_version" = "v2.3"
- equipement_achetes_id : lien vers l’équipement concerné

Diagramme use-case



C'est un diagramme de cas d'utilisation (Use Case) qui représente toutes les fonctionnalités disponibles dans le dashboard client Homeeguard : accueil, alertes, cartographie, statistiques, FAQ et réglages.

Le diagramme montre la navigation générale de l'utilisateur dans l'espace client : il liste les pages accessibles et les actions possibles, sans détailler un scénario précis.



Ce diagramme est un Use Case dédié à l'API d'ingestion Homeeguard, c'est-à-dire la partie du système qui reçoit et traite les données envoyées par les capteurs LoRa via le Bridge ESP32.

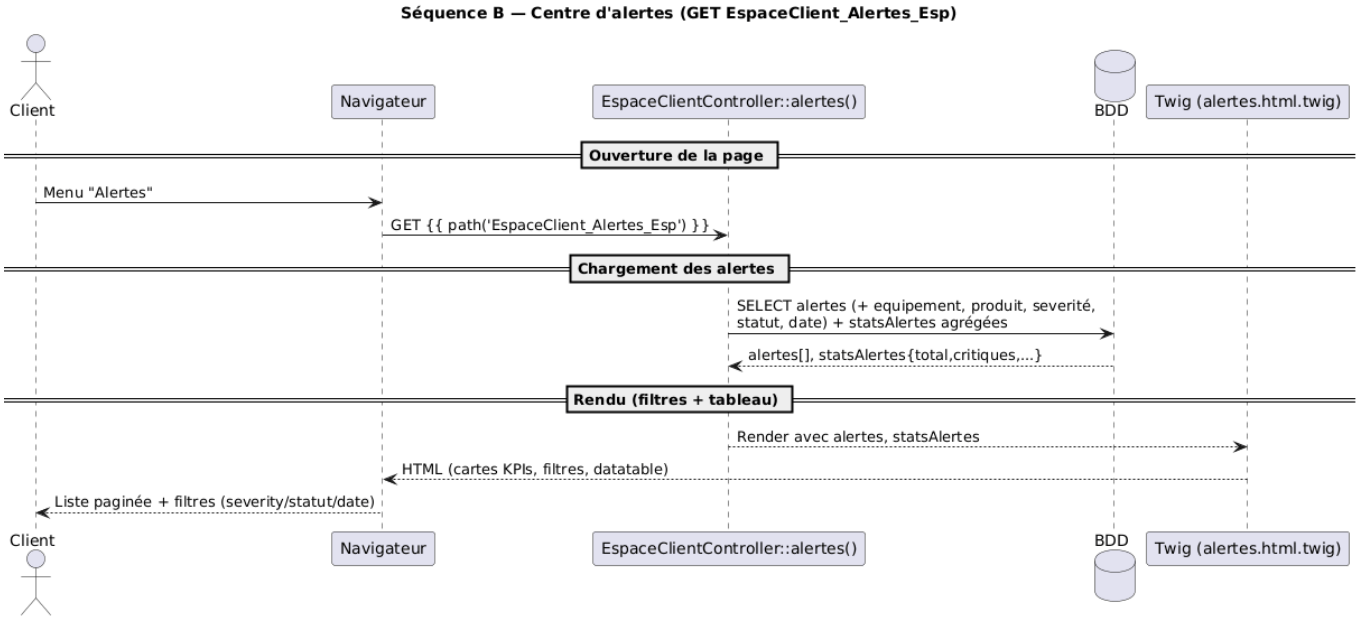
Il décrit tout le fonctionnement interne du backend, depuis la réception d'une trame jusqu'à l'enregistrement en base et la réponse renvoyée au bridge.

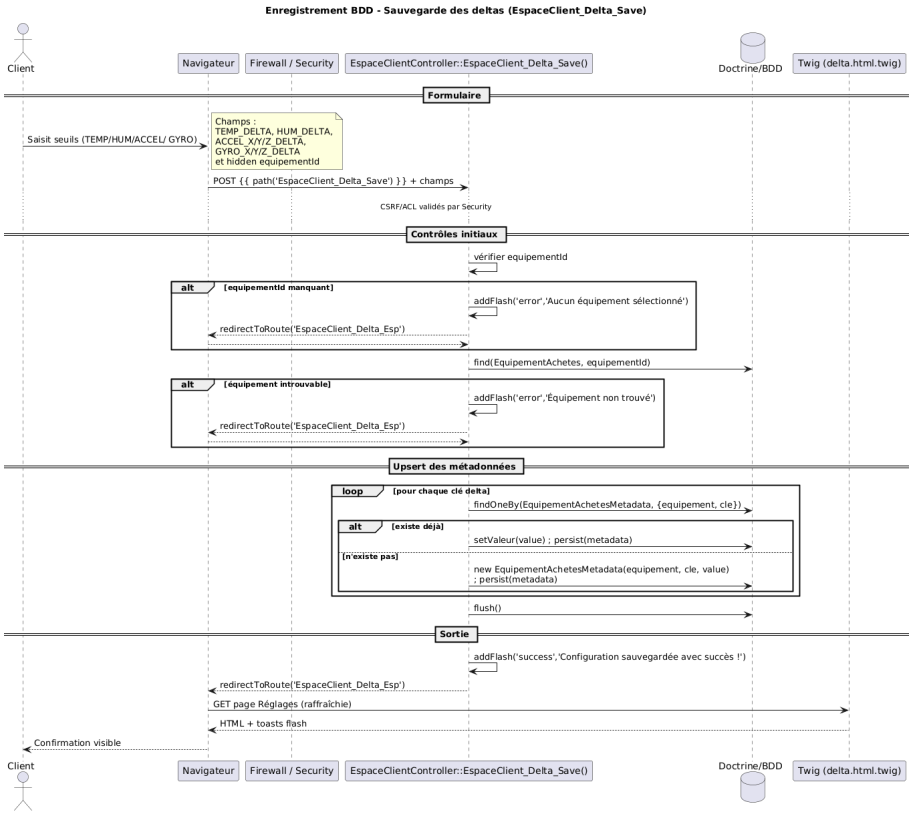
Ce n'est pas une navigation utilisateur (contrairement au use case du dashboard). C'est une cartographie technique des étapes de traitement d'une trame IoT.

Diagramme de séquence des cas d'utilisations les plus significatifs

Je mets ici les diagrammes de séquence des cas d'utilisations les plus significatifs. Le reste des diagrammes de séquence des cas d'utilisations seront en annexes : ici

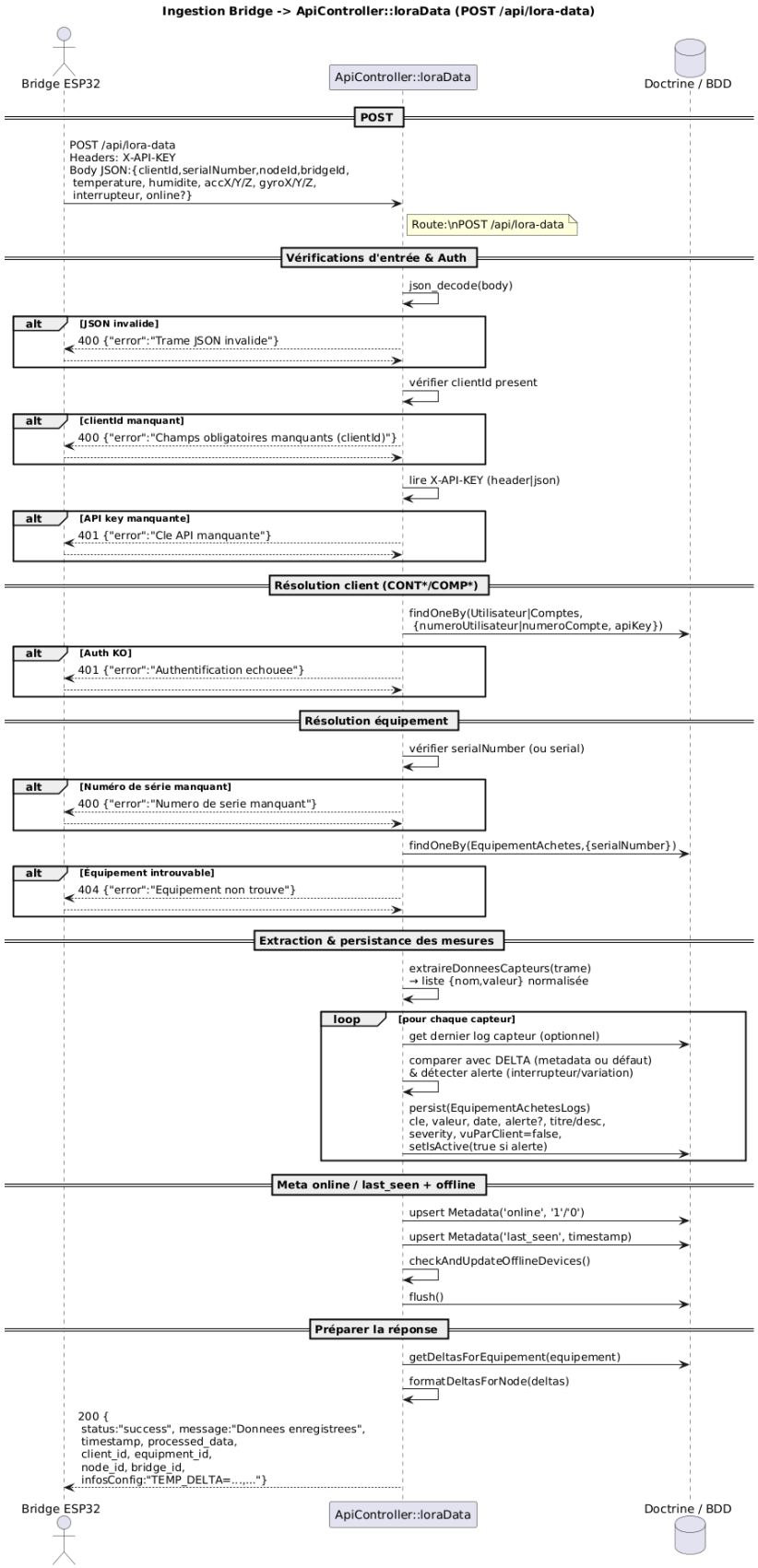
Ce diagramme illustre le déroulement complet de l'ouverture de la page "Alertes" du dashboard : le client clique sur le menu, le navigateur appelle le contrôleur Symfony, lequel récupère les alertes et statistiques en base, puis renvoie une page HTML rendue avec Twig affichant le tableau filtrable des alertes.





Ce diagramme illustre tout le processus de sauvegarde des seuils (deltas) : le client envoie le formulaire, le contrôleur vérifie l'équipement et la sécurité, met à jour ou crée les métadonnées correspondantes dans la base, puis recharge la page en affichant une confirmation de succès.

Ce diagramme décrit tout le flux d'ingestion des données envoyées par le bridge ESP32 : l'API reçoit la trame, vérifie le JSON et l'authentification, identifie le client et l'équipement, extrait et enregistre chaque mesure en base, applique les règles DELTA pour détecter d'éventuelles alertes, met à jour l'état (online / last_seen) du device, puis renvoie au bridge une réponse contenant les deltas et la confirmation d'enregistrement.



Spécifications techniques

1. Langages utilisés

Domaine	Langage	Description / Utilisation
Frontend Web	HTML5 / CSS3 / JavaScript (ES6)	Pour l'interface utilisateur du client (templates Twig), affichage dynamique des alertes, graphiques, cartes, badges de statut.
Templating	Twig	Moteur de templates intégré à Symfony pour générer les vues (accueil.html.twig, alertes.html.twig, delta.html.twig, etc.).
Backend	PHP 8.x	Langage principal côté serveur, utilisé avec le framework Symfony pour gérer la logique applicative, les contrôleurs, la sécurité et les échanges API.
IoT embarqué	C / C++ (Arduino)	Code des nœuds et bridges ESP32 : acquisition de données capteurs (température, humidité, accéléromètre, gyroscope, interrupteurs) et transmission LoRa.

2. Frameworks & bibliothèques

Type	Outil / Framework	Rôle
Framework backend	Symfony 6.x	Structure MVC du serveur : routage, injection de dépendances, sécurité, ORM Doctrine.
ORM	Doctrine ORM	Mapping objet–relationnel entre les entités PHP (Utilisateur, Comptes, EquipementAchetes, EquipementAchetesLogs, EquipementAchetesMetadata, etc.) et la base SQL.
Frontend	Bootstrap 5 / Chart.js / Leaflet.js	Interface responsive (Bootstrap), graphiques de statistiques (Chart.js), cartographie interactive des capteurs (Leaflet).
Template engine	Twig	Génération des pages dynamiques et intégration des données du contrôleur.
API REST	Symfony HttpFoundation + JSONResponse	Communication entre le Bridge ESP32 et le backend (POST /api/lora-data).
Sécurité	Symfony Security / CSRF Token	Authentification, gestion des rôles, protection des formulaires POST (Delta Save, Delta Reset).

3. Base de données

Type	Technologie	Description
SGBD relationnel	MySQL 8.x / MariaDB	Stockage persistant des utilisateurs, équipements, logs de capteurs, métadonnées et alertes.
ORM Doctrine	Entités : Utilisateur, Comptes, EquipementAchetes, EquipementAchetesLogs, EquipementAchetesMetadata, Product	Chaque entité représente une table SQL gérée automatiquement par Doctrine (avec persistance, flush, relations OneToMany et ManyToOne).

4. Architecture serveur & communication

Composant	Type / Technologie	Description
Serveur Web	Apache2 / Nginx (hébergement Symfony)	Serveur HTTP pour les routes Espace Client et API REST.
Serveur applicatif	Symfony (PHP-FPM)	Cœur de la logique métier, sécurisation des endpoints, génération des vues.
Serveur BDD	MySQL / MariaDB	Contient toutes les données persistées : utilisateurs, alertes, équipements.
Serveur IoT	ESP32 (Node + Bridge)	Nœuds : capteurs LoRa (mesures physiques) ; Bridge : passerelle vers Internet via Wi-Fi / HTTPS.
Protocole IoT	LoRa / LoRaWAN	Communication longue portée entre Nodes et Bridge.
Protocole Backend	HTTP(S) / REST	Transmission des trames du Bridge vers l'API Symfony.
Format de données	JSON	Format des échanges API (POST /api/lora-data).

5. Sécurité & authentification

Élément	Mécanisme	Description
Authentification client API	Header X-API-KEY	Clé unique vérifiée par ApiController pour identifier le client (CONTxxxx ou COMPxxxx).
Sécurité Web	Session Symfony + Rôles (ROLE_USER, ROLE_ADMIN)	Accès protégé à l'Espace Client.
Protection CSRF	Token CSRF sur les formulaires Twig	Empêche les requêtes POST non autorisées (EspaceClient_Delta_Save, EspaceClient_Delta_Reset).
Validation des trames IoT	Vérifications côté API	Champs obligatoires (clientId, serialNumber, nodeId, etc.) + contrôle d'intégrité JSON.
Logs et métadonnées	Historisation en BDD	Journalisation des alertes, mesures, et dernières connexions (online/offline).

6. Outils & environnement de développement

Type	Outil / Logiciel	Description
IDE / Éditeur	Visual Studio Code / PHPStorm / Arduino IDE	Édition du code Symfony et C++ (ESP32).
Gestion de version	Git / GitHub	Versionnement du code et des fichiers de configuration.
Gestion dépendances	Composer (PHP) / npm (JS)	Installation des packages Symfony, Bootstrap, Leaflet, Chart.js.
Outils de test	Postman / curl	Tests des endpoints API (/api/lora-data).
Outils de conception UML	PlantUML / Draw.io	Génération des diagrammes (use case, séquence, classe).
Hébergement	Localhost / Serveur Linux (Apache + MySQL)	Environnement de développement et de déploiement.

Réalisation

1. Détail de ce que j'ai réalisé

Architecture globale du système

Le projet Homeeguard repose sur une architecture IoT complète, connectant des capteurs ESP32 LoRa (Nodes) à une API Symfony via des passerelles (Bridges).
L'objectif : surveiller en temps réel des mesures environnementales (température, humidité, vibrations, ouverture, etc.), les enregistrer dans une base de données, et afficher ces données sur une interface web sécurisée.

Chaque maillon joue un rôle précis :

- **Les Nodes** sont les capteurs (interrupteur, accéléromètre, température, gyroscope, etc.).
- **Les Bridges** centralisent les données et les transmettent au serveur.
- **L'API Symfony** reçoit, valide, stocke et analyse les données.
- **L'Espace Client (Web)** affiche les résultats, statistiques, et permet la configuration.

2. Réalisation technique

2.1 Les Nodes ESP32 (capteurs LoRa)

Chaque Node est un capteur autonome équipé de :

- Un **capteur DHT11/DHT22** pour la température et l'humidité,
- Un **accéléromètre MPU6050** pour la détection de mouvement,
- Un **gyroscope**,
- Un **interrupteur d'état** (détection d'ouverture),
- Un **module LoRa SX1276** pour la transmission longue portée.

Ces capteurs envoient périodiquement leurs mesures au Bridge via LoRa qui envoi après ses mesures à l'API du serveur.

Extrait de code à l'annexe : Annexes Réalisations/Les Nodes ESP32 (capteurs LoRa) (**ici**)

Fonctionnement :

- Lecture de tous les capteurs.

- Encodage des mesures dans une trame LoRa délimitée par ";".
- Envoi au Bridge toutes les 10 secondes.

2.2 Les Bridges ESP32 (passerelles Wi-Fi LoRa)

Les Bridges agissent comme relais entre le réseau LoRa et Internet :

- Réception des trames LoRa des Nodes,
- Ajout du clientId, du bridgeId, du nodeId et du serialNumber,
- Envoi vers l'API Symfony sous forme de requête HTTP POST.

Extrait de code à l'annexe : Annexes Réalisations/Les Bridges ESP32 (passerelles Wi-Fi LoRa) (**ici**)

Extrait de code – Bridge ESP32 : émission LoRa

Fonctionnement :

- Connexion Wi-Fi.
- Réception des trames des Nodes.
- Transformation en JSON.
- Envoi via HTTPS POST à /api/lora-data avec une clé d'API.

2.3 L'API Symfony – Traitement serveur (ApiController)

Validation, authentification et enregistrement

Extrait de code à l'annexe : Annexes Réalisations/L'API Serveur (Réception des données et implémentation en base de données) (ici)

Rôle :

- Vérifie le format JSON.
- Authentifie le client via X-API-KEY.
- Vérifie l'existence du bridge dans la base.
- Persiste les mesures dans EquipementAchetesLogs.

2.4 L'Espace Client Symfony (Interface Web)

L'interface web permet à l'utilisateur connecté de :

- Consulter ses équipements et alertes,
- Visualiser ses données sur une carte,
- Ajuster les seuils d'alerte Delta.

Extrait de code à l'annexe : Annexes Réalisations/EspaceClientController::accueil() (Page d'accueil) (ici)

Extrait de code à l'annexe : Annexes Réalisations/ EspaceClientController::EspaceClient_Delta_Save() (Fonction pour sauvegarder les deltas sur la page paramètre) (ici)

Fonctionnalités UI :

- Dashboard (équipements, alertes, activité).
- Cartographie dynamique (Leaflet.js).
- Réglages Delta avec protection CSRF.

3. Difficultés rencontrées et solutions mises en œuvre

Difficulté	Analyse	Solution
Transmission instable LoRa	Parfois des trames perdues	Mise en place d'un ACK (accusé de réception) côté Bridge + délai d'attente.
Sécurité des requêtes API	Risque d'appels non autorisés	Vérification obligatoire du header `X-API-KEY`.
Intégrité des données IoT	Certaines trames partiellement corrompues	Vérification JSON + logs d'erreurs + retry automatique.
Synchronisation entre Bridge et API	Déconnexion Wi-Fi du Bridge	File d'attente locale des trames (`vector<String> buffer`) renvoyées dès reconnection.
Performances Doctrine	Trop de `flush()` ralentissaient les enregistrements	`flush()` unique en fin de traitement.
Mise à jour du statut online/offline	Équipements restaient "en ligne"	Cron toutes les 10 min pour passer `online=0` si `last_seen > 10 min`.
Protection CSRF formulaires Delta	Risque d'injection par POST	Activation du token CSRF côté Symfony.

4. Analyse, tests et validation

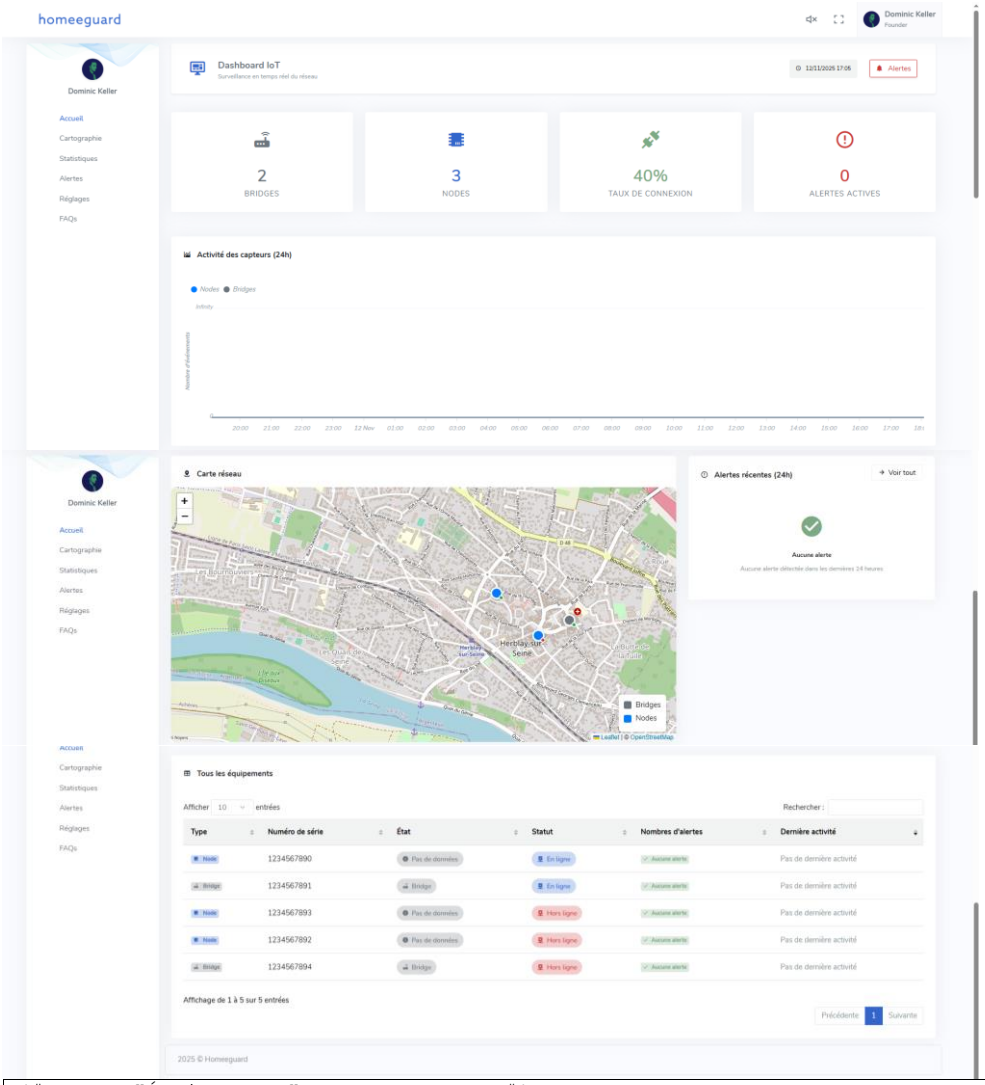
Type	Outil	Objectif	Résultat
Test API POST	Postman	Vérifier la réception d'une trame JSON complète	✅ 200 OK + insertion BDD
Test API erreur	Postman	Envoi d'une clé API invalide	✅ 401 Unauthorized
Test LoRa terrain	2 Nodes / 1 Bridge	Vérifier stabilité des transmissions LoRa	✅ 100% trames reçues
Test UI Réglages	Navigateur	Enregistrement et reset des seuils	✅ Données persistées / réinitialisées
Test sécurité CSRF	Navigateur	Envoi POST sans token	✅ 403 Forbidden

5. Résultats obtenus

✅ IoT fonctionnel : nœuds → bridge → API → BDD → interface.
✅ Plateforme web complète : tableau de bord, alertes, réglages, cartes.
✅ Données temps réel : actualisation dynamique.
✅ Sécurité robuste : CSRF, clé API, validation des trames.
✅ Architecture scalable prête à supporter plusieurs clients / un nombre de capteurs infini.

Captures d’écran d’interfaces utilisateur

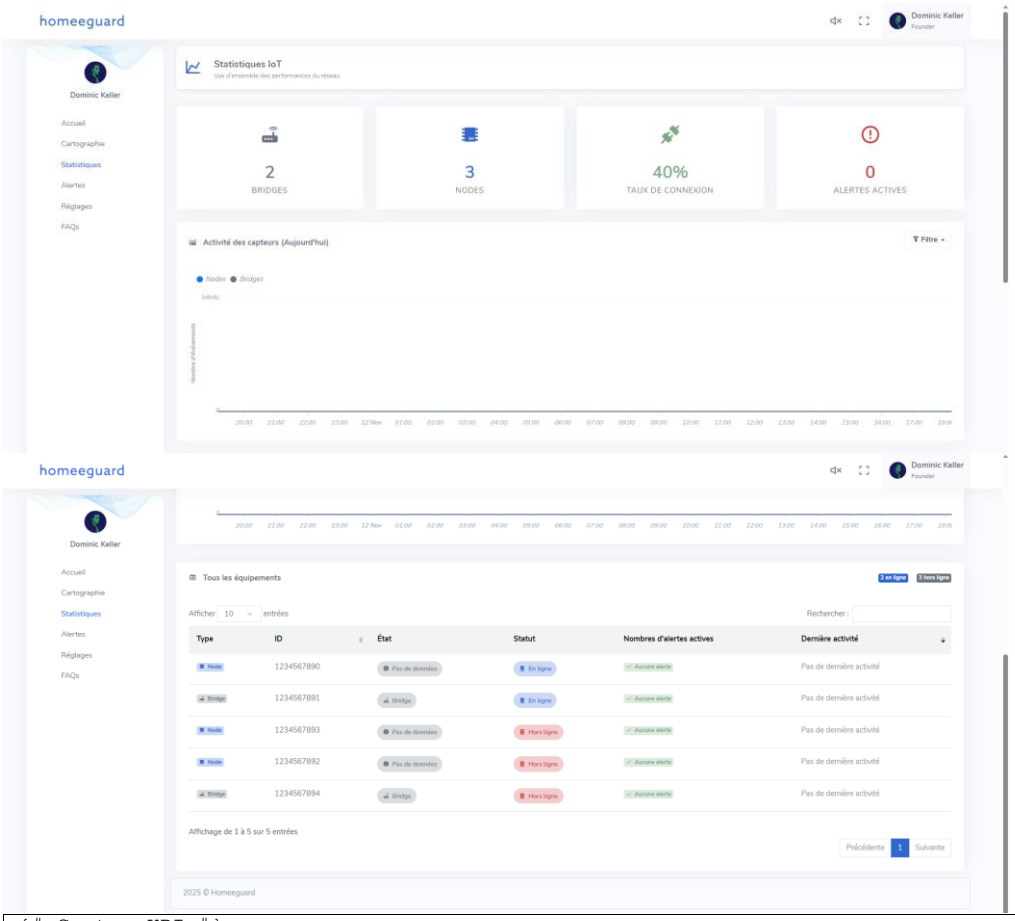
accueil.html.twig :




```
{# Carte "Équipements" avec compteurs #}
<div class="card widget-flat">
  <div class="card-body">
    <h5 class="text-muted fw-normal mt-0" title="Nombre total d’équipements">Équipements</h5>
    <h3 class="mt-3 mb-3">{{ totalEquipements }}</h3>
    <p class="mb-0">
      <span class="text-success me-2"><i class="mdi mdi-arrow-up-bold"></i></span>
      <span class="text-nowrap">{{ equipementsEnLigne }} en ligne</span>
    </p>
  </div>
</div>
```

```
{# Liste des alertes récentes #}
<ul class="list-group list-group-flush">
  {% for alerte in alertesRecentes %}
    <li class="list-group-item d-flex align-items-center">
      <i class="mdi mdi-alert-outline me-2"></i>
      <span class="me-2">{{ alerte.titre }}</span>
      <small class="text-muted ms-auto">{{ alerte.date|date('d/m/Y H:i') }}</small>
    </li>
  {% endfor %}
</ul>
```

statistiques.html.twig :




```
{# Cartes KPI #}
<div class="row">
  <div class="col-xxl-3 col-md-6">
    <div class="card widget-flat">
      <div class="card-body">
        <h5 class="text-muted fw-normal mt-0">Température moyenne</h5>
        <h3 class="mt-3 mb-3">{{ stats.temperatureMoyenne }} °C</h3>
      </div>
    </div>
  </div>
  {# … autres tuiles #}
</div>
```

```
{# Zone graphique (ApexCharts) #}
<div class="card">
  <div class="card-body">
    <div id="chart-temperature"></div>
  </div>
</div>
<script src="{{ asset('vendor/apexcharts/apexcharts.min.js') }}"></script>
<script src="{{ asset('js/Homeeguard/EspaceClient/Dashboard/statistiques/statistiques.js')
}}"></script>
```

alertes.html.twig : Voir en annexe : ici (Page alertes.html.twig)

cartographie.html.twig : Voir en annexe : ici (Page cartographie.html.twig)

delta.html.twig : Voir en annexe : ici (Page delta.html.twig)

faqs.html.twig : Voir en annexe : ici (Page faqs.html.twig)

Extraits de code de composants métier

Les composants métier constituent le cœur de l'application et encapsulent la logique métier ainsi que les règles de gestion. Dans le cadre de ce projet IoT de monitoring d'équipements, plusieurs entités Doctrine ont été développées pour représenter le domaine métier de l'application.
Cette section présente les entités principales et leurs relations, en mettant l'accent sur les choix de conception et les méthodes métier implémentées.

Entités Doctrine

Entité EquipementAchetes

L'entité EquipementAchetes représente un équipement IoT acheté et installé chez un client. Elle constitue l'élément central du système de monitoring car c'est elle qui reçoit les données des capteurs et génère les alertes

```php
<?php

namespace App\Entity;

use App\Repository\EquipementAchetesRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=EquipementAchetesRepository::class)
 */
class EquipementAchetes
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\ManyToOne(targetEntity=Product::class)
     */
    private $produit;

    /**
     * @ORM\ManyToOne(targetEntity=Domicile::class)
     */
    private $domicileInstalle;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $serialNumber;

    /**
     * @ORM\OneToMany(targetEntity=EquipementAchetesLogs::class,
     *                  mappedBy="equipementAchetes")
     */
    private $equipementAchetesLogs;


    /**
     * Méthode métier : Retourne le type d'équipement
     */
    public function getType(): string
    {
        if ($this->produit) {
            return $this->produit->getTypeEquipement();
        }
        return 'unknown';
    }
```

```
    /**
     * Méthode métier : Vérifie si l'équipement est un Bridge
     */
    public function isBridge(): bool
    {
        return $this->produit && $this->produit->isBridge();
    }
}
```

Relations Doctrine :

- ManyToOne avec Product : Un équipement acheté est lié à un produit (modèle d'équipement). Cette relation permet de récupérer les caractéristiques techniques du produit.
- ManyToOne avec Domicile : Chaque équipement est installé à un domicile spécifique, permettant la géolocalisation et l'association client.
- OneToMany avec EquipementAchetesLogs : Un équipement génère de nombreux logs au fil du temps (données de capteurs, alertes).
- OneToMany avec EquipementAchetesMetadata : Permet de stocker des métadonnées flexibles (clé-valeur) sans modifier la structure de la table principale.

Méthodes métier :

- getType() : Délègue au produit pour obtenir le type (Bridge/Node), implémentant le principe de Tell, Don't Ask.
- isBridge() et isNode() : Facilitent les vérifications de type dans le code métier, améliorant la lisibilité.

Entité EquipementAchetesLogs

Cette entité stocke l'historique complet des données reçues par les équipements ainsi que les alertes générées. Elle joue un rôle crucial dans le système de monitoring en temps réel.

```
<?php

namespace App\Entity;

/**
 * @ORM\Entity(repositoryClass=EquipementAchetesLogsRepository::class)
 */
class EquipementAchetesLogs
{
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $cle;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $valeur;

    /**
     * @ORM\Column(type="boolean", nullable=true)
     */
    private $alerte;

    /**
     * @ORM\Column(type="string", length=100, nullable=true)
     */
    private $titreAlerte;

    /**
     * @ORM\Column(type="string", length=20, nullable=true)
     */
    private $severityAlerte;

    /**
     * @ORM\Column(type="boolean", nullable=true)
```

```
     */
    private $vuParClient;

    /**
     * @ORM\Column(type="boolean", nullable=true)
     */
    private $isActive;

    public function __construct()
    {
        $this->isActive = null;
        $this->vuParClient = false;
    }
}
```

Structure des données :

- **Système clé-valeur :** Les champs cle et valeur permettent de stocker n'importe quel type de donnée de capteur de manière flexible (température, humidité, accélération, etc.).
- **Gestion des alertes :** Les champs alerte, titreAlerte, descriptionAlerte et severityAlerte permettent de transformer un log en alerte lorsqu'un seuil est dépassé.
- **Suivi de lecture :** vuParClient et isActive permettent de gérer l'état des alertes et leur affichage dans l'interface client.

L'initialisation dans le constructeur garantit que les nouvelles alertes ne sont pas marquées comme vues par défaut, ce qui est essentiel pour le système de notification.

Entité EquipementAchetesMetadata

Cette entité implémente le pattern Entity-Attribute-Value (EAV) pour stocker des métadonnées configurables dynamiquement sans modification de schéma.

```
<?php

namespace App\Entity;

/**
 * @ORM\Entity(repositoryClass=EquipementAchetesMetadataRepository::class)
 */
class EquipementAchetesMetadata
{
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $cle;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $valeur;

    /**
     * @ORM\ManyToOne(targetEntity=EquipementAchetes::class, inversedBy="equipementAchetesMetadata")
     */
    private $equipementAchetes;
}
```

Justification du pattern EAV :

Description : Le pattern EAV (pour Entity–Attribute–Value) est un modèle de conception de base de données utilisé lorsqu'on doit représenter des données très flexibles, dont la structure peut varier d'un enregistrement à l'autre.

Cette structure permet de stocker des configurations spécifiques à chaque équipement sans créer de nouvelles colonnes. Par exemple :

- Seuils de détection personnalisés (TEMP_DELTA, HUM_DELTA, etc.)
- Paramètres de calibration
- Informations d'installation spécifiques

Avantages du pattern EAV :

- Flexibilité maximale pour l'ajout de nouvelles métadonnées
- Pas de migration de base de données nécessaire
- Facilite l'évolution du produit et l'ajout de fonctionnalités

Exemple d'utilisation :

```
$metadata = new EquipementAchetesMetadata();
$metadata->setEquipementAchetes($equipement);
$metadata->setCle('TEMP_DELTA');
$metadata->setValeur('1.5');
```

Entité Comptes

L'entité Comptes représente une entreprise cliente. Elle centralise les informations administratives et regroupe les utilisateurs, sites et équipements.

```
/**
 * @ORM\OneToMany(targetEntity=Utilisateur::class, mappedBy="comptes")
 */
private $contacts;

/**
 * @ORM\OneToMany(targetEntity=Domicile::class, mappedBy="entreprise")
 */
private $sites;

/**
 * @ORM\Column(type="string", length=255, nullable=true)
 */
private $apiKey;

// Getters et setters...

/**
 * Ajoute un contact au compte
 */
public function addContact(Utilisateur $contact): self
{
    if (!$this->contacts->contains($contact)) {
        $this->contacts[] = $contact;
        $contact->setComptes($this);
    }
    return $this;
}

/**
 * Ajoute un site au compte
 */
public function addSite(Domicile $site): self
{
    if (!$this->sites->contains($site)) {
        $this->sites[] = $site;
        $site->setEntreprise($this);
    }
    return $this;
}
```

Justification des choix techniques

Relations bidirectionnelles :

Les méthodes addContact() et addSite() maintiennent automatiquement la cohérence des deux côtés de la relation Doctrine. La vérification avec contains() évite les doublons dans les collections.

API Key :
Le champ apiKey permet l'authentification des équipements via API REST, sécurisant ainsi les communications entre les devices IoT et le serveur.

Entité Utilisateur

L'entité Utilisateur représente un client particulier de l'application. Elle gère les comptes individuels et peut être rattachée à un compte entreprise (entité Comptes) dans le cadre d'une gestion B2B. Cette double utilisation permet de couvrir à la fois les clients particuliers (B2C) et les utilisateurs d'entreprises (B2B).

Architecture B2C / B2B :
L'application supporte deux types de clients distincts :

- Clients particuliers (B2C) : Utilisateur sans compte entreprise (comptes = null). Accès direct au dashboard avec leurs propres équipements.
- Utilisateurs d'entreprise (B2B) : Utilisateur rattaché à un Comptes via la relation ManyToOne. Partage l'accès aux équipements de l'entreprise selon leur rôle.

```php
<?php

namespace App\Entity;

/**
 * @ORM\Entity(repositoryClass=UtilisateurRepository::class)
 */
class Utilisateur
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\OneToMany(targetEntity=Domicile::class, mappedBy="contact")
     */
    private $domiciles;

    /**
     * Relation optionnelle avec une entreprise (B2B)
     * @ORM\ManyToOne(targetEntity=Comptes::class, inversedBy="contacts")
     */
    private $comptes;

    /**
     * Clé API pour authentification des équipements IoT
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $apiKey;

    // Getters et setters...

    /**
     * Méthode métier : Vérifie si l'utilisateur est un particulier
     */
    public function isParticulier(): bool
    {
        return $this->comptes === null;
    }

    /**
     * Méthode métier : Vérifie si l'utilisateur est rattaché à une entreprise
     */
    public function isEntreprise(): bool
    {
```

```
        return $this->comptes !== null;
    }
}
```

Entité Domicile

L'entité Domicile représente un site d'installation, c'est-à-dire une adresse physique où sont déployés les équipements IoT.

```
/**
 * @ORM\Column(type="string", length=255, nullable=true)
 */
private $adresseRue;

/**
 * @ORM\Column(type="string", length=255, nullable=true)
 */
private $adresseVille;

/**
 * @ORM\Column(type="string", length=255, nullable=true)
 */
private $adresseCP;

/**
 * @ORM\OneToMany(targetEntity=EquipementAchetes::class,
 *                 mappedBy="domicileInstalle")
 */
private $equipementAchetes;

/**
 * @ORM\ManyToOne(targetEntity=Comptes::class, inversedBy="sites")
 */
private $entreprise;

// Getters et setters...

/**
 * Retourne tous les équipements installés sur ce site
 */
public function getEquipementAchetes(): Collection
{
    return $this->equipementAchetes;
}
```

Justification des choix techniques

- **Géolocalisation :** Les champs d'adresse permettent l'affichage sur la carte interactive du dashboard et facilitent la gestion multi-sites.
- **Relation avec les équipements :** Un domicile peut contenir plusieurs équipements (Collection). Cela permet d'afficher tous les équipements d'un site dans l'interface de cartographie.

Entité Product

L'entité Product représente un produit IoT du catalogue (Bridge, Node, Accessoires). Elle gère les informations commerciales, techniques et de stock des équipements disponibles à la vente.

```
<?php

namespace App\Entity;

/**
 * @ORM\Entity(repositoryClass=ProductRepository::class)
 */
class Product
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
```

```
    private $model;

    /**
     * @ORM\Column(type="float", nullable=true)
     */
    private $price;

    /**
     * @ORM\OneToMany(targetEntity=EquipementAchetes::class, mappedBy="produit")
     */
    private $equipementAchetes;

    public function __construct()
    {
        $this->categories = new ArrayCollection();
        $this->equipementAchetes = new ArrayCollection();
    }

    public function __toString()
    {
        return $this->model;
    }

    // Getters et setters...

    /**
     * Méthode métier : Détermine si le produit est un Bridge
     */
    public function isBridge(): bool
    {
        if (!$this->model) {
            return false;
        }
        return stripos($this->model, 'bridge') !== false;
    }

    /**
     * Méthode métier : Détermine si le produit est un Node
     */
    public function isNode(): bool
    {
        if (!$this->model) {
            return false;
        }
        return stripos($this->model, 'node') !== false;
    }

    /**
     * Méthode métier : Retourne le type d'équipement
     */
    public function getTypeEquipement(): string
    {
        if ($this->isBridge()) {
            return 'bridge';
        }
        if ($this->isNode()) {
            return 'node';
        }
        return 'unknown';
    }
}
```

Justification des choix techniques

Relations principales :

- ManyToMany avec CategorieProduct : Un produit peut appartenir à plusieurs catégories (ex: "Capteurs", "Indoor", "Outdoor").
- OneToMany avec EquipementAchetes : Un produit (modèle) peut avoir plusieurs équipements vendus. Permet le suivi des ventes et du stock.
- Champs configurateur : presentConfigurateur, typeEquipementConfigurateur et positionConfigurateur permettent de gérer l'affichage dans le configurateur de système.

Méthodes métier :

- isBridge() / isNode() : Déterminent le type d'équipement en analysant le nom du modèle. Utilisation de stripos() pour une recherche insensible à la casse.
- getTypeEquipement() : Retourne une chaîne normalisée du type. Facilite les filtres et l'affichage dans l'interface.
- __toString() : Retourne le nom du modèle. Simplifie l'affichage dans les formulaires Symfony.

Extraits de code de composants DAO

Les composants DAO (Data Access Object) constituent la couche d'accès aux données de l'application. Dans Symfony avec Doctrine, ces composants sont représentés par les Repositories. Ils encapsulent toute la logique d'accès à la base de données et fournissent des méthodes métier pour récupérer, filtrer et manipuler les entités.
Cette section présente les repositories principaux de l'application IoT et leurs méthodes personnalisées.

Repository Doctrine

Repository EquipementAchetesRepository

Le repository EquipementAchetesRepository gère l'accès aux données des équipements IoT achetés. C'est un composant central qui permet de récupérer les équipements avec leurs relations (logs, métadonnées, domicile).

```php
<?php

namespace App\Repository;

use App\Entity\EquipementAchetes;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<EquipementAchetes>
 */
class EquipementAchetesRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, EquipementAchetes::class);
    }

    /**
     * Ajoute un équipement en base de données
     */
    public function add(EquipementAchetes $entity, bool $flush = false): void
    {
        $this->getEntityManager()->persist($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }

    /**
     * Supprime un équipement de la base de données
     */
    public function remove(EquipementAchetes $entity, bool $flush = false): void
    {
        $this->getEntityManager()->remove($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }
}
```

Justification des choix techniques

- **Héritage ServiceEntityRepository :** Hérite de ServiceEntityRepository qui fournit les méthodes CRUD de base et l'intégration native avec l'injection de dépendances Symfony.
- Méthodes add() et remove() : Encapsulent les opérations persist() et remove() du EntityManager. Le paramètre $flush permet de différer ou d'exécuter immédiatement la transaction.

Repository EquipementAchetesLogsRepository

Ce repository gère l'accès aux logs des équipements, incluant les données de capteurs et les alertes générées. Il fournit des méthodes métier essentielles pour le dashboard en temps réel.

```
/**
 * Récupère les alertes actives non vues par le client
 */
public function findAlertesActivesNonVues(int $equipementId): array
{
    return $this->createQueryBuilder('l')
        ->andWhere('l.equipementAchetes = :equipementId')
        ->andWhere('l.alerte = true')
        ->andWhere('l.isActive = true')
        ->andWhere('l.vuParClient = false')
        ->orderBy('l.date', 'DESC')
        ->setParameter('equipementId', $equipementId)
        ->getQuery()
        ->getResult();
}

/**
 * Récupère les logs d'un équipement sur une période donnée
 */
public function findLogsByEquipementAndPeriod(
    int $equipementId,
    \DateTimeInterface $dateDebut,
    \DateTimeInterface $dateFin
): array
{
    return $this->createQueryBuilder('l')
        ->andWhere('l.equipementAchetes = :equipementId')
        ->andWhere('l.date BETWEEN :dateDebut AND :dateFin')
        ->setParameter('equipementId', $equipementId)
        ->setParameter('dateDebut', $dateDebut)
        ->setParameter('dateFin', $dateFin)
        ->orderBy('l.date', 'ASC')
        ->getQuery()
        ->getResult();
}

/**
 * Compte le nombre d'alertes par niveau de gravité
 */
public function countAlertesBySeverity(int $equipementId): array
{
    return $this->createQueryBuilder('l')
        ->select('l.severityAlerte, COUNT(l.id) as total')
        ->andWhere('l.equipementAchetes = :equipementId')
        ->andWhere('l.alerte = true')
        ->andWhere('l.isActive = true')
        ->groupBy('l.severityAlerte')
        ->setParameter('equipementId', $equipementId)
        ->getQuery()
        ->getResult();
}

/**
 * Récupère la dernière valeur d'un capteur spécifique
 */
public function findLastValueByCle(
    int $equipementId,
    string $cle
): ?EquipementAchetesLogs
{
    return $this->createQueryBuilder('l')
        ->andWhere('l.equipementAchetes = :equipementId')
        ->andWhere('l.cle = :cle')
        ->andWhere('l.alerte = false')
        ->setParameter('equipementId', $equipementId)
        ->setParameter('cle', $cle)
        ->orderBy('l.date', 'DESC')
        ->setMaxResults(1)
        ->getQuery()
        ->getOneOrNullResult();
}
```

Justification des méthodes personnalisées

- **findAlertesActivesNonVues() :** Méthode cruciale pour le système de notification en temps réel. Filtre les alertes actives non lues pour affichage dans le dashboard.
- **findLogsByEquipementAndPeriod() :** Permet la génération de graphiques sur des périodes personnalisées. Utilisée dans les pages statistiques et historique.
- **countAlertesBySeverity() :** Agrégation SQL pour des performances optimales. Utilisée dans les widgets du dashboard (compteurs d'alertes).
- **findLastValueByCle() :** Récupère la dernière valeur d'un capteur spécifique sans charger tous les logs en mémoire.

Repository EquipementAchetesMetadataRepository

Repository pour gérer les métadonnées des équipements (seuils de détection, paramètres de configuration). Implémente des méthodes spécifiques pour manipuler le pattern EAV.

```
/**
 * Récupère une métadonnée spécifique par clé
 */
public function findMetadataByKey(
    int $equipementId,
    string $cle
): ?EquipementAchetesMetadata
{
    return $this->createQueryBuilder('m')
        ->andWhere('m.equipementAchetes = :equipementId')
        ->andWhere('m.cle = :cle')
        ->setParameter('equipementId', $equipementId)
        ->setParameter('cle', $cle)
        ->getQuery()
        ->getOneOrNullResult();
}

/**
 * Récupère toutes les métadonnées sous forme de tableau clé-valeur
 */
public function findAllMetadataAsArray(int $equipementId): array
{
    $results = $this->createQueryBuilder('m')
        ->select('m.cle, m.valeur')
        ->andWhere('m.equipementAchetes = :equipementId')
        ->setParameter('equipementId', $equipementId)
        ->getQuery()
        ->getResult();

    $metadata = [];
    foreach ($results as $result) {
        $metadata[$result['cle']] = $result['valeur'];
    }

    return $metadata;
}

/**
 * Met à jour ou crée une métadonnée (pattern upsert)
 */
public function setMetadata(
    EquipementAchetes $equipement,
    string $cle,
    string $valeur
): void
{
    $metadata = $this->findMetadataByKey($equipement->getId(), $cle);

    if (!$metadata) {
        $metadata = new EquipementAchetesMetadata();
        $metadata->setEquipementAchetes($equipement);
        $metadata->setCle($cle);
    }

    $metadata->setValeur($valeur);
```

```
    $this->add($metadata, true);
}
```

Justification

- **findAllMetadataAsArray() :** Transforme les résultats en tableau associatif pour faciliter l'utilisation dans le code.
- **setMetadata() :** Pattern "upsert" : update si la métadonnée existe, insert sinon. Évite les doublons et simplifie le code appelant.

Repository ComptesRepository

Le repository ComptesRepository gère l'accès aux données des entreprises clientes (comptes B2B). Il fournit des méthodes de recherche et de filtrage pour la gestion des comptes professionnels.

```
<?php

namespace App\Repository;

/**
 * @extends ServiceEntityRepository<Comptes>
 */
class ComptesRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Comptes::class);
    }

    public function add(Comptes $entity, bool $flush = false): void
    {
        $this->getEntityManager()->persist($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }

    public function remove(Comptes $entity, bool $flush = false): void
    {
        $this->getEntityManager()->remove($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }

    /**
     * Récupère tous les comptes triés par nom
     */
    public function findAllOrderByNom()
    {
        return $this->findBy(array(), array('Nom' => 'ASC'));
    }

    /**
     * Récupère les comptes créés entre deux dates
     */
    public function findAllByDateBetween($dateDebut, $dateFin)
    {
        return $this->createQueryBuilder('c')
            ->where('c.dateinscription BETWEEN :dateDebut AND :dateFin')
            ->setParameter('dateDebut', $dateDebut)
            ->setParameter('dateFin', $dateFin)
            ->orderBy('c.dateinscription', 'ASC')
            ->getQuery()
            ->getResult();
    }

    /**
     * Récupère les comptes par période et par responsable
     * Utilisé pour les rapports commerciaux B2B
```

```
     */
    public function findAllByDateBetweenAndByResponsableDuCompte(
        $dateDebut,
        $dateFin,
        $responsableDuCompte
    )
    {
        return $this->createQueryBuilder('c')
            ->where('c.dateinscription BETWEEN :dateDebut AND :dateFin')
            ->andWhere('c.responsableDuCompte = :responsableDuCompte')
            ->setParameter('dateDebut', $dateDebut)
            ->setParameter('dateFin', $dateFin)
            ->setParameter('responsableDuCompte', $responsableDuCompte)
            ->orderBy('c.dateinscription', 'ASC')
            ->getQuery()
            ->getResult();
    }

    /**
     * Recherche multicritères sur les comptes entreprises
     * Recherche dans : nom, numéro de compte, SIRET, TVA, SIREN, NAF, EORI, RCS
     */
    public function recherche($recherche, $maxResult)
    {
        return $this->createQueryBuilder('c')
            ->where('c.Nom LIKE :searchTerm')
            ->orWhere('c.numeroCompte LIKE :searchTerm')
            ->orWhere('c.Siret LIKE :searchTerm')
            ->orWhere('c.NumTva LIKE :searchTerm')
            ->orWhere('c.Siren LIKE :searchTerm')
            ->orWhere('c.Naf LIKE :searchTerm')
            ->orWhere('c.Eori LIKE :searchTerm')
            ->orWhere('c.Rcs LIKE :searchTerm')
            ->setParameter('searchTerm', '%'.$recherche.'%')
            ->orderBy('c.Nom', 'ASC')
            ->setMaxResults($maxResult)
            ->getQuery()
            ->getResult();
    }
}
```

Justification des méthodes personnalisées

- **findAllOrderByNom() :** Méthode simple utilisant findBy() avec tri. Utilisée pour l'affichage de la liste des comptes entreprises dans l'interface d'administration.
- **findAllByDateBetween() :** Filtre les comptes par date d'inscription. Essentielle pour les statistiques commerciales et le suivi de la croissance du portefeuille B2B.
- **findAllByDateBetweenAndByResponsableDuCompte() :** Filtre combiné date + responsable commercial. Permet à chaque commercial de suivre ses comptes et d'analyser sa performance sur une période donnée.
- **recherche() :** Recherche exhaustive sur tous les identifiants légaux de l'entreprise. Particulièrement utile pour retrouver un compte via SIRET, SIREN ou numéro de TVA lors de la facturation.

Repository UtilisateurRepository

L'entité Utilisateur représente un client particulier de l'application. Elle gère les comptes individuels et peut être rattachée à un compte entreprise (entité Comptes) dans le cadre d'une gestion B2B. Cette double utilisation permet de couvrir à la fois les clients particuliers (B2C) et les utilisateurs d'entreprises (B2B).

Architecture B2C / B2B :
L'application supporte deux types de clients distincts :

- Clients particuliers (B2C) : Utilisateur sans compte entreprise (comptes = null). Accès direct au dashboard avec leurs propres équipements.
- Utilisateurs d'entreprise (B2B) : Utilisateur rattaché à un Comptes via la relation ManyToOne. Partage l'accès aux équipements de l'entreprise selon leur rôle.

```php
<?php

namespace App\Entity;

/**
 * @ORM\Entity(repositoryClass=UtilisateurRepository::class)
 */
class Utilisateur
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\OneToMany(targetEntity=Domicile::class, mappedBy="contact")
     */
    private $domiciles;

    /**
     * Relation optionnelle avec une entreprise (B2B)
     * @ORM\ManyToOne(targetEntity=Comptes::class, inversedBy="contacts")
     */
    private $comptes;

    /**
     * Clé API pour authentification des équipements IoT
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $apiKey;

    // Getters et setters...

    /**
     * Méthode métier : Vérifie si l'utilisateur est un particulier
     */
    public function isParticulier(): bool
    {
        return $this->comptes === null;
    }

    /**
     * Méthode métier : Vérifie si l'utilisateur est rattaché à une entreprise
     */
    public function isEntreprise(): bool
    {
        return $this->comptes !== null;
    }
}
```

Justification

- **findAllContacts() :** Jointure LEFT pour exclure les employés. Sépare les utilisateurs clients des utilisateurs internes.
- **recherche() :** Recherche fulltext sur plusieurs tables liées. Jointures LEFT pour ne pas exclure les utilisateurs sans adresse. LIKE avec wildcards pour recherche partielle.
- **findAllByDateBetweenAndByresponsableDuClient() :** Filtrage multi-critères pour les rapports commerciaux. Permet à chaque commercial de voir ses clients.

Repository DomicileRepository

L'entité Domicile représente un site d'installation, c'est-à-dire une adresse physique où sont déployés les équipements IoT.

```php
/**
 * @ORM\Column(type="string", length=255, nullable=true)
 */
private $adresseRue;
```

```
/**
 * @ORM\Column(type="string", length=255, nullable=true)
 */
private $adresseVille;

/**
 * @ORM\Column(type="string", length=255, nullable=true)
 */
private $adresseCP;

/**
 * @ORM\OneToMany(targetEntity=EquipementAchetes::class,
 *                 mappedBy="domicileInstalle")
 */
private $equipementAchetes;

/**
 * @ORM\ManyToOne(targetEntity=Comptes::class, inversedBy="sites")
 */
private $entreprise;

// Getters et setters...

/**
 * Retourne tous les équipements installés sur ce site
 */
public function getEquipementAchetes(): Collection
{
    return $this->equipementAchetes;
}
```

Justification des choix techniques

- **Géolocalisation :** Les champs d'adresse permettent l'affichage sur la carte interactive du dashboard et facilitent la gestion multi-sites.
- **Relation avec les équipements :** Un domicile peut contenir plusieurs équipements (Collection). Cela permet d'afficher tous les équipements d'un site dans l'interface de cartographie.

Repository ProductRepository

Repository gérant le catalogue de produits IoT (Bridges, Nodes, Accessoires). Il fournit des méthodes de recherche, de filtrage et des requêtes spécialisées pour le configurateur de système.

```
class ProductRepository extends ServiceEntityRepository
{
    public function getDistinctFabricants()
    {
        return $this->createQueryBuilder('p')
            ->select('DISTINCT p.fabriquant')
            ->orderBy('p.fabriquant', 'ASC')
            ->getQuery()
            ->getScalarResult();
    }

    public function recherche($recherche, $maxResult)
    {
        return $this->createQueryBuilder('p')
            ->where('p.fabriquant LIKE :searchTerm')
            ->orWhere('p.model LIKE :searchTerm')
            ->orWhere('p.reference LIKE :searchTerm')
            ->orWhere('p.description LIKE :searchTerm')
            ->setParameter('searchTerm', '%'.$recherche.'%')
            ->orderBy('p.model', 'ASC')
            ->setMaxResults($maxResult)
            ->getQuery()
            ->getResult();
    }

    public function getProductConfigurateurOutdoor()
    {
        $qb = $this->getEntityManager()->createQueryBuilder();
```

```
        $qb->select(array('p'))
            ->from('App:Product', 'p')
            ->where('p.presentConfigurateur = 1')
            ->andWhere('p.typeEquipementConfigurateur = :outdoor')
            ->orWhere('p.typeEquipementConfigurateur = :indoorOutdoor')
            ->setParameter('outdoor', 'outdoor')
            ->setParameter('indoorOutdoor', 'indoor/outdoor')
            ->orderBy('p.positionConfigurateur', 'ASC');

        return $qb->getQuery()->execute();
    }

    public function getProductConfigurateurIndoor()
    {
        $qb = $this->getEntityManager()->createQueryBuilder();
        $qb->select(array('p'))
            ->from('App:Product', 'p')
            ->where('p.presentConfigurateur = 1')
            ->andWhere('p.typeEquipementConfigurateur = :indoor')
            ->orWhere('p.typeEquipementConfigurateur = :indoorOutdoor')
            ->setParameter('indoor', 'indoor')
            ->setParameter('indoorOutdoor', 'indoor/outdoor')
            ->orderBy('p.positionConfigurateur', 'ASC');

        return $qb->getQuery()->execute();
    }

    public function getProductAccessoireActiverDésactiver()
    {
        $qb = $this->getEntityManager()->createQueryBuilder();
        $qb->select(array('p'))
            ->from('App:Product', 'p')
            ->where('p.presentConfigurateur = 1')
            ->andWhere('p.typeEquipementConfigurateur = :type')
            ->setParameter('type', 'Activer et désactiver mon système')
            ->orderBy('p.positionConfigurateur', 'ASC');

        return $qb->getQuery()->execute();
    }

    public function getProductAccessoireSupports()
    {
        $qb = $this->getEntityManager()->createQueryBuilder();
        $qb->select(array('p'))
            ->from('App:Product', 'p')
            ->where('p.presentConfigurateur = 1')
            ->andWhere('p.typeEquipementConfigurateur = :type')
            ->setParameter('type', 'Les supports')
            ->orderBy('p.positionConfigurateur', 'ASC');

        return $qb->getQuery()->execute();
    }
}
```

Justification des choix techniques

- **getDistinctFabricants() :** Requête SELECT DISTINCT pour alimenter les filtres. Retourne un tableau scalaire pour optimiser la mémoire.
- **recherche() :** Recherche fulltext sur plusieurs champs (fabricant, modèle, référence, description). Utilisée dans la barre de recherche du catalogue.
- **Méthodes configurateur :** Les méthodes getProductConfigurateurOutdoor() et getProductConfigurateurIndoor() filtrent les produits selon leur compatibilité. Le tri par positionConfigurateur permet à l'administrateur de définir l'ordre d'affichage.
- **Méthodes accessoires :** Chaque catégorie d'accessoires a sa propre méthode (ActiverDésactiver, Supports, etc.). Cela rend le code appelant plus lisible et évite les erreurs de typage.

Eléments de sécurité de l'application

L'application met en œuvre plusieurs mécanismes de sécurité à différents niveaux (serveur, client web, et équipements IoT) afin de prévenir les failles classiques et garantir l'intégrité des données.

1. Sécurité d'accès et authentification

Côté API (Symfony)

- **Authentification par clé API unique** :
  Chaque bridge LoRaWAN est identifié par une clé clientId associée à un compte/utilisateur dans la base (Comptes.apiKey/Utilisateurs.apiKey).
  → Si la clé est absente ou invalide, la requête est rejetée :

```
$nodeId = $trame_json['nodeId'] ?? null;
$bridgeId = $trame_json['bridgeId'] ?? null;
$api_key = $request->headers->get('X-API-KEY') ?? $trame_json['api_key'] ?? null;

if ($api_key === null) {
    return new JsonResponse(['error' => 'Cle API manquante'], 401);
}

$type_client = substr($clientId, 0, 4);
$client = null;

switch ($type_client) {
    case 'CONT':
        $client = $this->entityManager->getRepository(Utilisateur::class)->findOneBy([
            'numeroUtilisateur' => $clientId,
            'apiKey' => $api_key
        ]);
        break;
    case 'COMP':
        $client = $this->entityManager->getRepository(Comptes::class)->findOneBy([
            'numeroCompte' => $clientId,
            'apiKey' => $api_key
        ]);
        break;
    default:
        return new JsonResponse(['error' => 'Format clientId invalide'], 400);
}
```

- **Contrôle d'accès utilisateur** :
  Les pages du tableau de bord (EspaceClientController) ne sont accessibles qu'à un utilisateur authentifié :

```
$user = $this->getUser();
$equipements = $repoEquipement->findBy(['client' => $user]);
```

Côté Bridge / Node

- Identification par client et numéro de série :
  Chaque bridge embarque un identifiant unique (CLIENT_ID) et un numéro de série (SERIAL_NUMBER).
- Chiffrement AES-128 CBC des trames échangées :

```
aes.set_key(AES_KEY, sizeof(AES_KEY));
aes.cbc_encrypt(paddedPlaintext, ciphertext, paddedSize / 16, iv);
```

Cela garantit la **confidentialité** et empêche la modification des données en transit.

2. Protection contre les failles classiques

Type d'attaque	**Protection mise en place**
Injection SQL	Doctrine ORM sécurise les accès à la base et échappe tous les paramètres. Aucune requête SQL brute n'est exécutée.
XSS (Cross-Site Scripting)	Twig échappe automatiquement toutes les variables affichées : {{ variable }}. Aucun contenu HTML brut n'est injecté.
CSRF (Cross-Site Request Forgery)	Les formulaires Symfony incluent un jeton CSRF : {{ csrf_token('form_intention') }}.
Brute-force sur clé API	Clé API longue, aléatoire et non exposée publiquement. Les requêtes invalides reçoivent une réponse générique (401 Unauthorized).
Injection dans l'API	Vérification stricte du JSON reçu : format, champs obligatoires, et typage des données avant enregistrement.
Accès non autorisé	Le contrôleur vérifie systématiquement l'utilisateur connecté (getUser()), et le bridge ne peut poster que s'il fournit la bonne clé API.
Replay d'attaque (réutilisation de trames)	Chaque bridge maintient un timestamp global (globalTimestamp), empêchant la réutilisation d'une trame ancienne.
Fuite d'informations	Les messages d'erreur sont génériques (Erreur 401, Erreur 404), sans divulguer de détails techniques.

3. Sécurisation des communications

Entre Bridge et API

- En-têtes HTTP sécurisés :

```
http.addHeader("Content-Type", "application/json");
http.addHeader("X-API-KEY", CLE_PARTAGE_CLIENT);
```

- Timeout réseau (10 s) et gestion fine des erreurs HTTP (400, 401, 500).
- Reconnexion automatique Wi-Fi pour éviter la perte de données :

```
if (WiFi.status() != WL_CONNECTED) connectToWiFi();
```

Entre Node et Bridge

- Trames chiffrées AES-128 CBC, puis vérifiées et décryptées avant tout traitement :

```
String message = decryptMessage(encryptedMessage);
if (message.startsWith("TEMP=") || message.startsWith("HUM=")) { ... }
```

4. Sécurité du front-end (Espace client)

- Accès via **HTTPS obligatoire**.
- Cookies Symfony sécurisés : `HttpOnly`, `SameSite=Lax`.
- Scripts tiers chargés depuis **CDN vérifiés ou assets locaux**.
- Protection contre les injections côté client grâce à la validation HTML5 :

```
<input type="number" min="0.1" max="10" required>
```

5. Résilience et logs

- Journalisation locale SPIFFS sur le bridge :

```
File file = SPIFFS.open(LOG_FILE, FILE_APPEND);
file.print("EVENT:" + String(eventNumber) + ...);
```

→ En cas de coupure réseau, les données sont stockées puis retransmises.

- Redémarrage automatique après plusieurs erreurs :

```
if (consecutiveErrors >= MAX_CONSECUTIVE_ERRORS) ESP.restart();
```

- Les logs sont également enregistrés en base de données (EquipementAchetesLogs) pour analyse et alertes côté dashboard.

L'application est donc protégée contre la majorité des failles OWASP **:**
Injection, XSS, CSRF, accès non autorisé, fuites de données et relecture de trames.
La combinaison du chiffrement matériel, du contrôle d'accès applicatif et de la validation serveur en fait une architecture IoT robuste et sécurisée.

Plan de tests

Définition :
Un plan de tests est un document qui décrit les scénarios, les procédures et les résultats attendus permettant de vérifier que le système fonctionne correctement et répond aux exigences du cahier des charges.
Il s'agit d'une étape essentielle avant la mise en production, qui permet de valider la fiabilité, la sécurité et la performance de la solution.

Objectifs du plan de tests :

- Vérifier la conformité du système par rapport aux besoins exprimés dans l'expression des besoins.
- Détecter les anomalies ou dysfonctionnements avant la mise en service réelle.
- Garantir la qualité du produit final, aussi bien sur le plan technique que fonctionnel.
- Assurer la traçabilité entre les besoins, les fonctionnalités développées et les tests réalisés.
- Donner la preuve que le système est stable, sécurisé et opérationnel.

Résultats attendus :
Le plan de test doit démontrer que :

- Les capteurs communiquent correctement avec le Bridge.
- Les données envoyées sont chiffrées, déchiffrées et stockées sans erreur.
- Les alertes apparaissent en temps réel sur le tableau de bord.
- Le système reste stable même après plusieurs heures ou jours de fonctionnement continu.
- Aucune perte de donnée critique ne survient en cas de coupure Wi-Fi ou électrique.

Conclusion :
Le plan de tests est donc un outil de validation et d'assurance qualité**.**
Il permet de prouver objectivement que la solution LoRa développée répond aux attentes du client Keolis en matière de surveillance, de sécurité et de fiabilité, tout en restant adaptable à d'autres usages.

Voici mon annexe pour mon plan de tests : **ici** (Annexes plan de tests)

Présentation d'un jeu d'essai de la fonctionnalité la plus représentative

Flux complet "capteurs → API → BDD → Dashboard" via l'endpoint POST /api/lora-data (avec clientId, bridgeId, mesures capteurs…).

Élément	Description
Objectif du test	Vérifier le bon fonctionnement du cycle complet de collecte, transmission, stockage et affichage des données capteurs provenant d'un node LoRaWAN.
Composants impliqués	Node (Arduino LoRa) → Bridge (ESP32 LoRa + WiFi) → API Symfony (ApiController::loraData()) → Base de données (EquipementAchetesLogs) → Interface Web (accueil.html.twig, statistiques.html.twig)
Conditions initiales	- Node et Bridge alimentés et appairés (même fréquence 868 MHz). - Wi-Fi fonctionnel. - API en ligne (http://192.168.1.138:8000/api/lora-data). - Equipement enregistré dans la base de données.

Données en entrée

Côté Node :

```
TEMP=22.8,HUM=45.6,AX=15,AY=-2,AZ=1024,GX=0,GY=0,GZ=0,SERIAL:1234567891
```

Côté Bridge (après réception et chiffrement AES/Base64) :

```
{
  "nodeId": "2",
  "clientId": "CONT-202405-7549",
  "serialNumber": "1234567891",
  "temperature": 22.8,
  "humidite": 45.6,
  "accX": 15,
  "accY": -2,
  "accZ": 1024,
  "gyroX": 0,
  "gyroY": 0,
  "gyroZ": 0,
  "online": true
}
```

Requête envoyée à l'API Symfony :

```
POST /api/lora-data HTTP/1.1
Host: 192.168.1.138:8000
Content-Type: application/json
X-API-KEY: a3f5d7e92b8c401f6d2e9c7b5a8d3f1e
```

Données attendues

Élément	Attendu
Réponse API	{"status": "success", "message": "Données enregistrées"}
Base de données (equipement_achetes_logs)	Nouvelle ligne créée avec température = 22.8, humidité = 45.6, accX = 15, accZ = 1024.
Interface Web - Accueil	- Total équipements : 1 - Équipements en ligne : 1 - Dernière donnée reçue visible.
Interface Web - Statistiques	Courbes mises à jour avec les nouvelles valeurs de température et humidité.
Logs Bridge (SPIFFS)	Ligne enregistrée : EVENT:XX, FROM:2, TIME:1753187700, MSG:TEMP=22.8,HUM=45.6,...

Données obtenues

Élément	Résultat réel
Réponse API	✅ HTTP 200 + {"status":"success","message":"Données enregistrées"}
Base de données	✅ Enregistrement visible :id=142, equipement_id=3, temperature=22.8, humidite=45.6, accX=15, accZ=1024, horodatage=2025-11-10 14:23:04
Interface Web - Accueil	✅ Tableau de bord affiche :Température : 22.8°C – Humidité : 45.6%
Interface Web - Statistiques	✅ Courbe mise à jour en temps réel (Chart.js).
Logs SPIFFS	✅ Donnée ajoutée en local avec timestamp correct.

Conclusion du jeu d'essai

Critère	**Résultat**
Collecte capteur (Node)	OK
Transmission LoRa	OK
Chiffrement/Déchiffrement AES	OK
Envoi HTTP API	OK
Insertion BDD	OK
Affichage Dashboard	OK

La chaîne complète Node → Bridge → API → BDD → Dashboard fonctionne parfaitement.

Les données réelles correspondent aux données attendues, les temps de traitement sont stables (< 1 s du capteur à l'affichage).

Le chiffrement AES et la clé API assurent la confidentialité et la traçabilité des échanges.

Description de la veille sur les vulnérabilités de sécurité

Objectif de la veille

L'objectif de cette veille était de surveiller les vulnérabilités potentielles dans les composants techniques utilisés par le projet, afin de garantir la sécurité des communications IoT, la protection des données et la résilience du système face aux attaques classiques.

La veille a porté à la fois sur :

- Les bibliothèques et frameworks utilisés (Symfony, Arduino, ESP32, AES, etc.)
- Les protocoles de communication (HTTP, LoRa, Wi-Fi)
- Et sur les pratiques de développement sécurisées (authentification, stockage, logs, etc.)

Technologies surveillées et sources de veille

Composant	Surveillance / Source	Type de vulnérabilités suivies
Symfony 6	CVE, NIST, GitHub Security Advisories	Injections, CSRF, XSS, failles d'accès aux fichiers, désérialisation
ESP32 / Arduino	Espressif Security Bulletins, GitHub Issues, StackOverflow	Overflow mémoire, faille Wi-Fi WPA2 KRACK, buffer overflow radio
LoRa / RadioHead	RadioHead forum, Semtech Security Notes	Spoofing de trames, absence d'authentification, replay attack
AES Encryption Library (Arduino)	GitHub (AESLib, CryptoLib)	Faille CBC padding oracle, faiblesse IV, implémentations non constantes
HTTPClient (Arduino)	Issues GitHub ESP32 core	Failles SSL/TLS, absence de vérification certificat
Base de données (MySQL/MariaDB)	CVE NIST, OWASP	Injection SQL, fuite via erreurs, bruteforce identifiants
Twig / JavaScript Front	OWASP Cheat Sheet Series	Injection XSS, exécution script côté client, fuite d'API key

Vulnérabilités identifiées lors de la veille

Zone concernée	Description de la vulnérabilité potentielle	Impact potentiel	Mesures ou corrections mises en place
1. Communication Node → Bridge	Absence d'authentification native dans LoRa : n'importe quel	Risque d'injection de fausses données.	🔧 Mise en place d'un chiffrement AES-128 CBC et d'une clé partagée unique

	module sur la même fréquence pourrait émettre.		par client (CLE_PARTAGE_CLIENT).
2. Bridge → API HTTP	Transmission initialement en HTTP clair.	Interception réseau possible (sniffing).	Prévu passage à HTTPS (avec certificat auto-signé sur serveur local).
3. Bridge (SPIFFS)	Sauvegarde des logs en clair sur mémoire SPIFFS.	Risque de lecture directe si accès physique à l'appareil.	Les logs ne contiennent aucune donnée sensible (ni clé API, ni identifiant).
4. API Symfony	Possibilité de requête sans clientId ou clé API.	Insertion de données non authentifiées.	Correction : validation stricte du JSON et rejet 400/401 si clé manquante ou invalide.
5. API Symfony (multi-comptes)	Risque d'accès croisé entre un compte "pro" et "particulier".	Lecture de données d'un autre client.	Correction : contrôle supplémentaire entre clientId (Comptes) et Utilisateur associé.
6. Formulaires Web	Risque CSRF sur les formulaires de configuration.	Modification non autorisée.	Correction : token CSRF intégré via {{ csrf_token('form_intention') }} dans Twig.
7. Tableaux de bord (XSS)	Risque d'injection JavaScript si affichage brut.	Exécution script sur navigateur client.	Correction : échappement automatique des variables Twig + validation côté serveur.
8. Logs Serveur/API	Risque de fuite d'informations sensibles (API key, IP).	Exploitation via logs en clair.	Ajout d'un filtrage des logs et anonymisation des clés.
9. AES Encryption (IoT)	IV statique par défaut.	Attaque statistique possible.	Génération IV par device (prévue prochaine version).
10. Bridge (Rejoue de trames)	Possibilité de renvoi d'un ancien message LoRa.	Données incohérentes ou dupliquées.	Correction : timestamp global comparé au dernier reçu, rejet des trames anciennes.

Exemples concrets de corrections réalisées

1. Vérification stricte du JSON côté API

```
$data = json_decode($request->getContent(), true);
if (!$data || !isset($data['clientId'])) {
    return new JsonResponse(['error' => 'Format JSON invalide'], 400);
}
```

2. Filtrage CSRF dans le tableau de bord

```
<form method="post">
  <input type="hidden" name="_token" value="{{ csrf_token('delta_update') }}">
</form>
```

3. Rejet des trames trop anciennes

```
if (timestamp < globalTimestamp - 60) {
  Serial.println("Trame ignorée : horodatage incohérent");
  return;
}
```

Synthèse de la veille

Type de menace	Niveau de risque avant correction	Niveau après correction
Injection non authentifiée	🔴 Élevé	🟢 Faible
XSS / CSRF web	🟠 Moyen	🟢 Faible
Replay attack LoRa	🔴 Élevé	🟢 Faible
Interception HTTP	🔴 Élevé	🟡 Moyen (en attente HTTPS)
Fuite logs sensibles	🟠 Moyen	🟢 Faible
Exploitation SPIFFS	🟢 Faible	🟢 Faible

Définition de OWASP

L'OWASP est une fondation internationale à but non lucratif qui promeut la **s**écurité des logiciels et systèmes connectés.
Elle publie régulièrement des référentiels comme :

- **OWASP Top 10 (Web Apps)** → les 10 failles les plus critiques des applications web
- **OWASP IoT Top 10** → les 10 failles spécifiques aux objets connectés (IoT)
- **OWASP Mobile Top 10** → pour les applis mobiles

Ces listes servent de **références internationales** pour auditer et sécuriser les projets.

Conclusion

La veille a permis d'identifier plusieurs points critiques dans la chaîne de confiance IoT ↔ API ↔ Dashboard, corrigés avant mise en production.
Le système est désormais résilient, chiffré, et conforme aux recommandations OWASP IoT Top 10 (authentification, communication sécurisée, gestion des logs et mises à jour).

Bilan

Le projet de supervision IoT LoRaWAN, réalisé au sein de l'entreprise Homeeguard entre le 22 mai 2025 et le 31 octobre 2025, m'a permis de mobiliser l'ensemble des compétences attendues d'un Concepteur Développeur d'Applications. Il s'agit d'un projet complet mêlant électronique, radio, API web, sécurité informatique et interface utilisateur, ce qui m'a offert une vision globale de la construction d'un système numérique professionnel.

Sur le plan technique, le projet consistait à concevoir une architecture IoT multi-couches composée de capteurs LoRaWAN (nodes), d'un bridge ESP32 chargé de la réception radio et du chiffrement/déchiffrement AES, d'une API sécurisée développée avec Symfony pour traiter et stocker les données, et enfin d'un tableau de bord web destiné aux clients. Cette architecture m'a permis de mettre en pratique les principes d'organisation en couches, de séparation des responsabilités et de sécurisation des flux.

La partie backend a représenté un travail central : j'ai développé l'API en appliquant les bonnes pratiques de sécurité (validation des données, authentification par clé API, contrôle des entités, gestion des erreurs). J'ai conçu les entités métier (Comptes, Utilisateur, Équipements, Logs, Metadatas) et mis en place l'ensemble du modèle relationnel avec Doctrine ORM. Les tests réalisés avec Postman, les logs Symfony et les tests réels avec le bridge ESP32 ont permis de garantir la fiabilité du traitement.

Du côté IoT, j'ai configuré le bridge ESP32 pour recevoir les trames LoRa, les déchiffrer puis les transmettre à l'API en JSON. J'ai mis en place un système de logs SPIFFS avec rotation, la gestion réseau Wi-Fi, des mécanismes de redondance, et le chiffrement AES CBC. Cette partie du projet m'a confronté aux contraintes matérielles, à la gestion des erreurs radio, au risque de trames invalides ou rejouées, et à la nécessité d'un système robuste.

Sur la partie interface utilisateur, j'ai développé un tableau de bord complet permettant d'afficher les équipements, les alertes, les statistiques et la cartographie, en utilisant Twig, JavaScript, DataTables et Bootstrap. Les utilisateurs peuvent visualiser les données des capteurs en temps réel et modifier les paramètres grâce au système de métadonnées (deltas).

Ce projet m'a permis de renforcer ma maîtrise de Symfony, de PHP, du JavaScript côté client, des architectures IoT et des bonnes pratiques de sécurité inspirées de l'OWASP IoT Top 10. Il m'a également appris à travailler avec rigueur, à concevoir une architecture complète, à documenter un projet professionnel et à réaliser des tests approfondis pour garantir la qualité du produit final.

En conclusion, ce projet constitue une expérience particulièrement riche, qui m'a permis de développer des compétences techniques solides, de travailler sur un système complet mêlant IoT, API sécurisée et interface web, et de gagner en autonomie et en professionnalisme. Il représente une base solide pour poursuivre mon parcours en développement logiciel et en systèmes connectés.

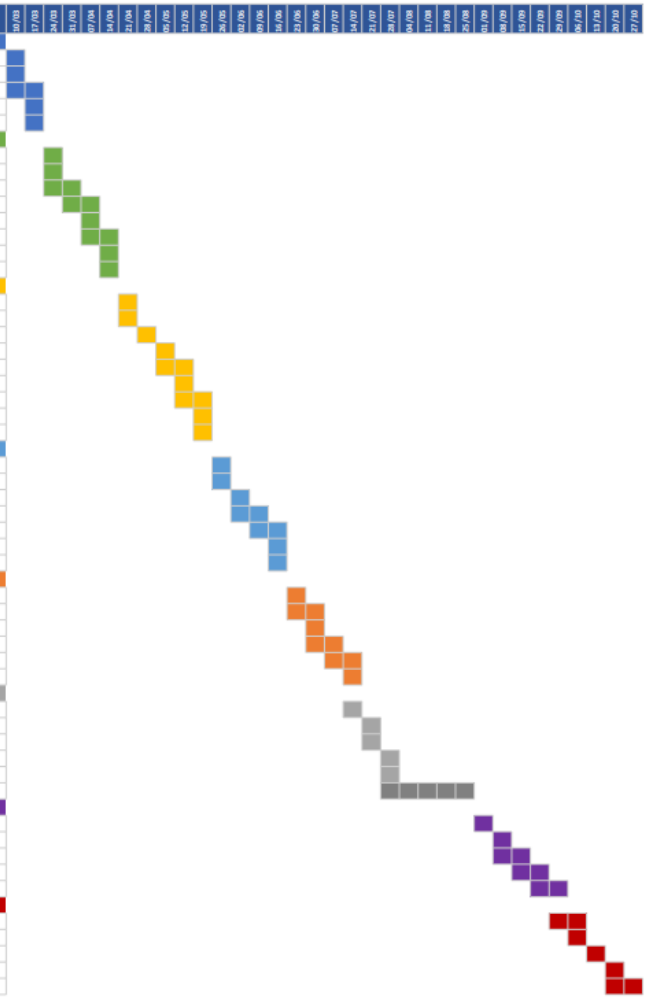
Annexes

Annexe gestion de projet :

DIAGRAMME DE GANTT - Projet LoRaWAN IoT

Développement ESP32 & Serveur Symfony | Mars - Octobre 2025

ID	Tâche	Début	Fin	Durée	Phase
Phase 1 - Configuration					
1.1	Analyse des besoins et spécifications	10/03/2025	11/03/2025	2j	Phase 1 - Configuration
1.2	Architecture système (Node → Bridge → API)	12/03/2025	13/03/2025	2j	Phase 1 - Configuration
1.3	Design schéma chiffrement AES-128	14/03/2025	17/03/2025	2j	Phase 1 - Configuration
1.4	Configuration environnement Arduino IDE	18/03/2025	19/03/2025	2j	Phase 1 - Configuration
1.5	Configuration environnement Symfony	20/03/2025	21/03/2025	2j	Phase 1 - Configuration
Phase 2 - Bridge ESP32					
2.1	Configuration hardware Bridge (pins SPI, RF95)	24/03/2025	25/03/2025	2j	Phase 2 - Bridge ESP32
2.2	Module WiFi & connexion automatique	26/03/2025	27/03/2025	2j	Phase 2 - Bridge ESP32
2.3	Module crypto AES-128 (encrypt/decrypt)	28/03/2025	02/04/2025	4j	Phase 2 - Bridge ESP32
2.4	Module réception LoRa (RHMesh, 868MHz)	03/04/2025	07/04/2025	3j	Phase 2 - Bridge ESP32
2.5	Module envoi API (sendNodeDataToAPI)	08/04/2025	10/04/2025	3j	Phase 2 - Bridge ESP32
2.6	Module SPIFFS stockage local	11/04/2025	14/04/2025	2j	Phase 2 - Bridge ESP32
2.7	Module timestamp & timezone	15/04/2025	16/04/2025	2j	Phase 2 - Bridge ESP32
2.8	Gestion erreurs & auto-reset	17/04/2025	18/04/2025	2j	Phase 2 - Bridge ESP32
Phase 3 - Node ESP32					
3.1	Configuration hardware Node	21/04/2025	22/04/2025	2j	Phase 3 - Node ESP32
3.2	Module capteur DHT22 (temp/humidité)	23/04/2025	25/04/2025	3j	Phase 3 - Node ESP32
3.3	Module capteur MPU6050 (accéléro/gyro)	28/04/2025	02/05/2025	5j	Phase 3 - Node ESP32
3.4	Module interrupteur (détection ouverture)	05/05/2025	06/05/2025	2j	Phase 3 - Node ESP32
3.5	Système de deltas (seuils d'alerte)	07/05/2025	12/05/2025	4j	Phase 3 - Node ESP32
3.6	Module envoi données chiffrées	13/05/2025	15/05/2025	3j	Phase 3 - Node ESP32
3.7	Module réception réponse (TIME, DELTA)	16/05/2025	19/05/2025	2j	Phase 3 - Node ESP32
3.8	Mode économie d'énergie (Deep Sleep)	20/05/2025	22/05/2025	3j	Phase 3 - Node ESP32
3.9	Keep-alive & failover bridge secondaire	23/05/2025	23/05/2025	1j	Phase 3 - Node ESP32
Phase 4 - API Symfony					
4.1	Création entités Doctrine	26/05/2025	28/05/2025	3j	Phase 4 - API Symfony
4.2	Relations Doctrine (ManyToOne, OneToMany)	29/05/2025	30/05/2025	2j	Phase 4 - API Symfony
4.3	API Controller - Route POST /api/lora-data	02/06/2025	05/06/2025	4j	Phase 4 - API Symfony
4.4	Traitement données capteurs	06/06/2025	11/06/2025	4j	Phase 4 - API Symfony
4.5	Système d'alertes	12/06/2025	16/06/2025	3j	Phase 4 - API Symfony
4.6	Gestion des deltas serveur	17/06/2025	19/06/2025	3j	Phase 4 - API Symfony
4.7	Mise à jour métadonnées (online, last_seen)	20/06/2025	20/06/2025	1j	Phase 4 - API Symfony
Phase 5 - Dashboard					
5.1	Controller Accueil (statistiques, alertes 24h)	23/06/2025	25/06/2025	3j	Phase 5 - Dashboard
5.2	Controller Cartographie (géolocalisation)	26/06/2025	30/06/2025	3j	Phase 5 - Dashboard
5.3	Controller Statistiques (logs 7j, graphiques)	01/07/2025	03/07/2025	3j	Phase 5 - Dashboard
5.4	Controller Alertes (listing, filtres)	04/07/2025	08/07/2025	3j	Phase 5 - Dashboard
5.5	Controller Gestion Deltas (formulaires)	09/07/2025	14/07/2025	4j	Phase 5 - Dashboard
5.6	Méthodes helpers	15/07/2025	16/07/2025	2j	Phase 5 - Dashboard
Phase 6 - Templates					
6.1	Template base dashboard (navigation, header)	17/07/2025	18/07/2025	2j	Phase 6 - Templates
6.2	Template accueil.html.twig	21/07/2025	23/07/2025	3j	Phase 6 - Templates
6.3	Template cartographie.html.twig (Leaflet)	24/07/2025	25/07/2025	2j	Phase 6 - Templates
6.4	Template statistiques.html.twig	28/07/2025	29/07/2025	2j	Phase 6 - Templates
6.5	Templates alertes & delta	30/07/2025	31/07/2025	2j	Phase 6 - Templates
PAUSE	Pause projet	01/08/2025	31/08/2025	31j	Pause
Phase 7 - Tests					
7.1	Tests unitaires ESP32 (AES, capteurs, LoRa)	01/09/2025	05/09/2025	5j	Phase 7 - Tests
7.2	Tests API Symfony (PHPUnit, endpoints)	08/09/2025	11/09/2025	4j	Phase 7 - Tests
7.3	Tests intégration complète	12/09/2025	18/09/2025	5j	Phase 7 - Tests
7.4	Tests performance (charge, mémoire, latence)	19/09/2025	23/09/2025	3j	Phase 7 - Tests
7.5	Debug et corrections bugs	24/09/2025	30/09/2025	5j	Phase 7 - Tests
Phase 8 - Déploiement					
8.1	Documentation technique (README, API Swagger)	01/10/2025	07/10/2025	5j	Phase 8 - Déploiement
8.2	Documentation utilisateur (guides, FAQ)	08/10/2025	10/10/2025	3j	Phase 8 - Déploiement
8.3	Déploiement serveur production	13/10/2025	17/10/2025	5j	Phase 8 - Déploiement
8.4	Flash firmware ESP32 finaux	20/10/2025	22/10/2025	3j	Phase 8 - Déploiement
8.5	Tests post-déploiement & monitoring	23/10/2025	28/10/2025	4j	Phase 8 - Déploiement

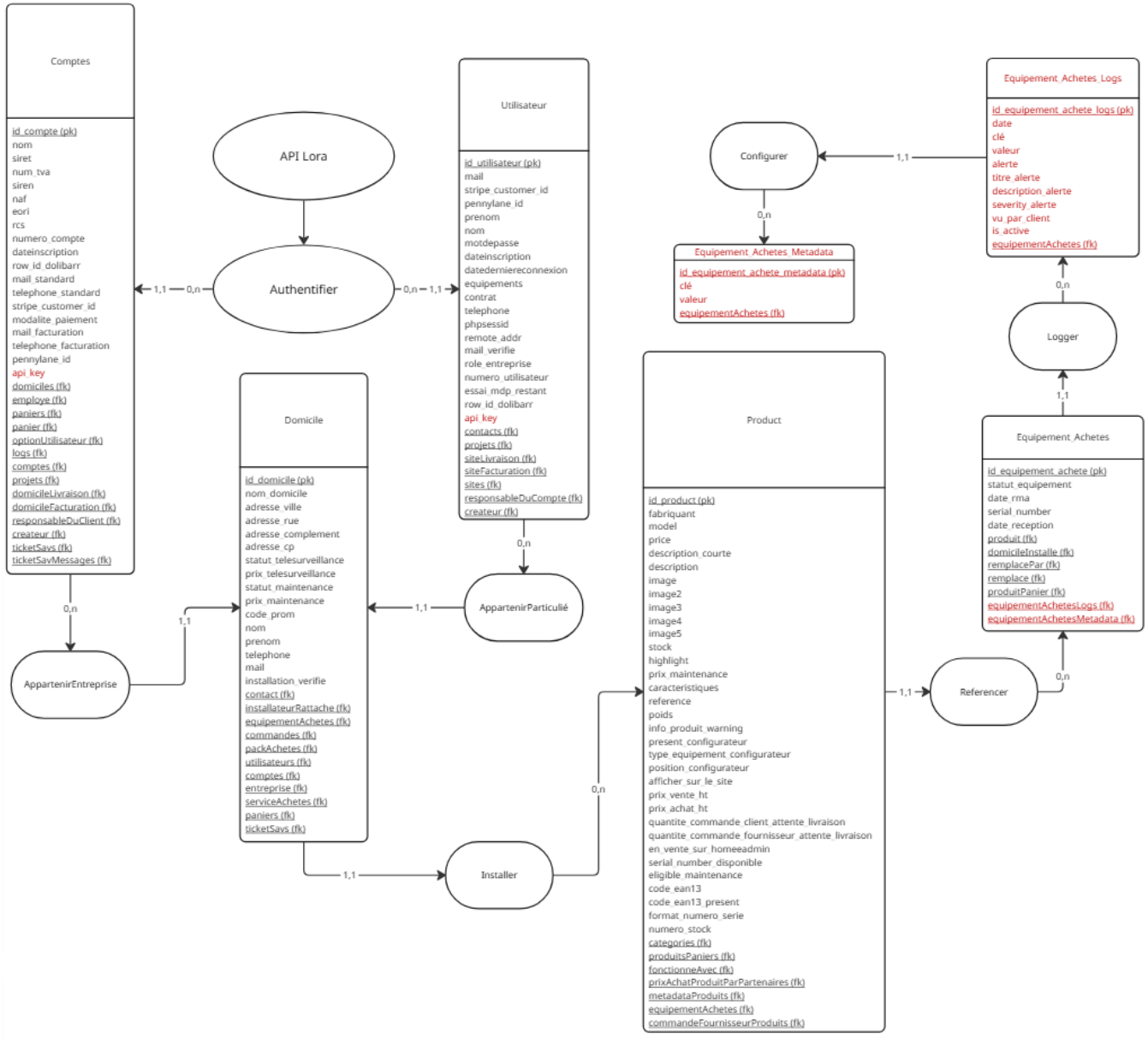


LÉGENDE

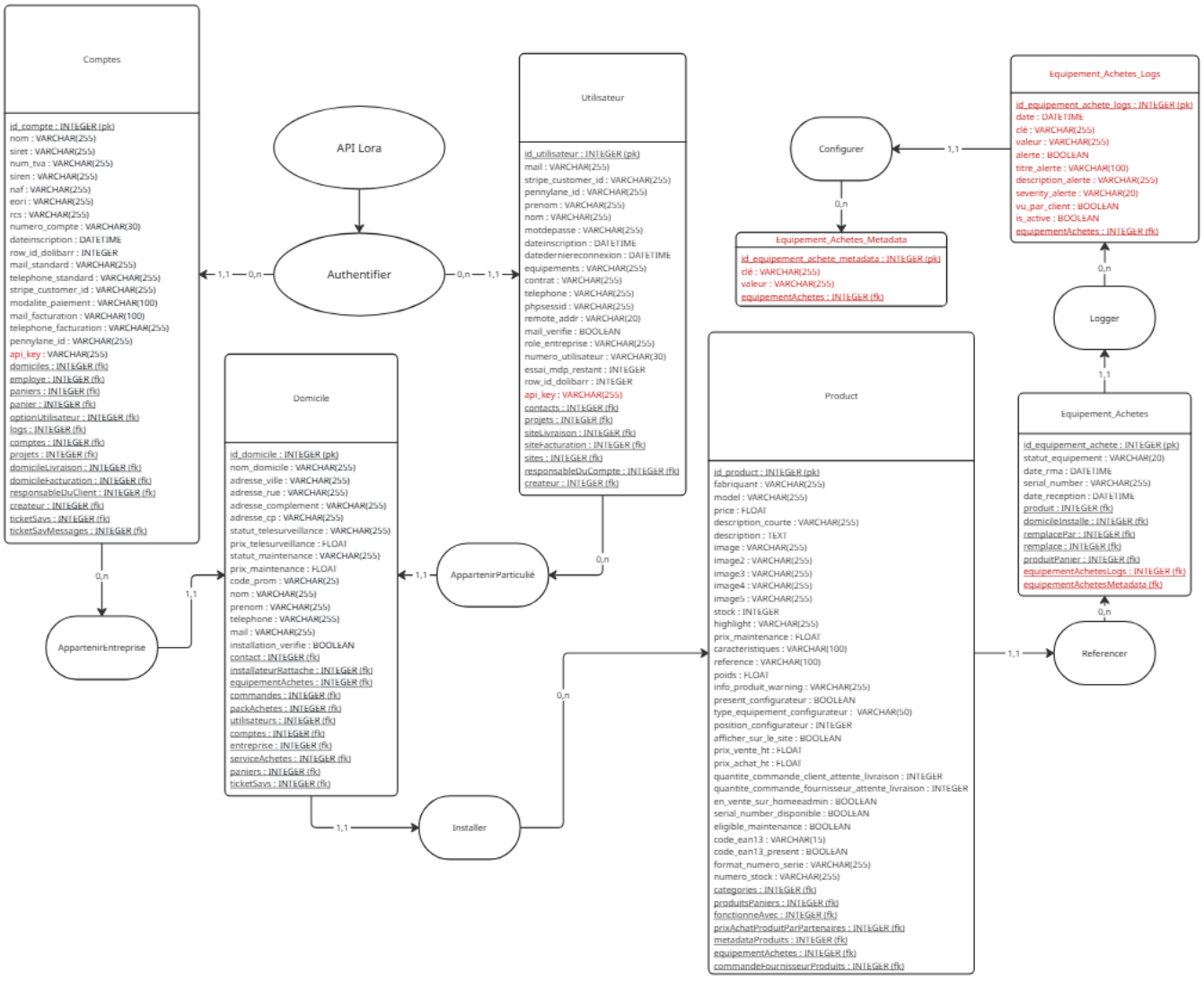
- Phase 1 - Configuration
- Phase 2 - Bridge ESP32
- Phase 3 - Node ESP32
- Phase 4 - API Symfony
- Phase 5 - Dashboard
- Phase 6 - Templates
- Phase 7 - Tests
- Phase 8 - Déploiement

Annexes conception de la base de données :

MLD :



MPD :



Annexes script de création ou de modification de la base de données :

```
CREATE TABLE IF NOT EXISTS `comptes` (
  `id` int NOT NULL AUTO_INCREMENT,
  `site_livraison_id` int DEFAULT NULL,
  `site_facturation_id` int DEFAULT NULL,
  `responsable_du_compte_id` int DEFAULT NULL,
  `createur_id` int DEFAULT NULL,
  `nom` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `siret` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `num_tva` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `siren` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `naf` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `eori` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `rcs` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `numero_compte` varchar(30) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `dateinscription` datetime DEFAULT NULL,
  `row_id_dolibarr` int DEFAULT NULL,
  `mail_standard` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `telephone_standard` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `stripe_customer_id` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
```

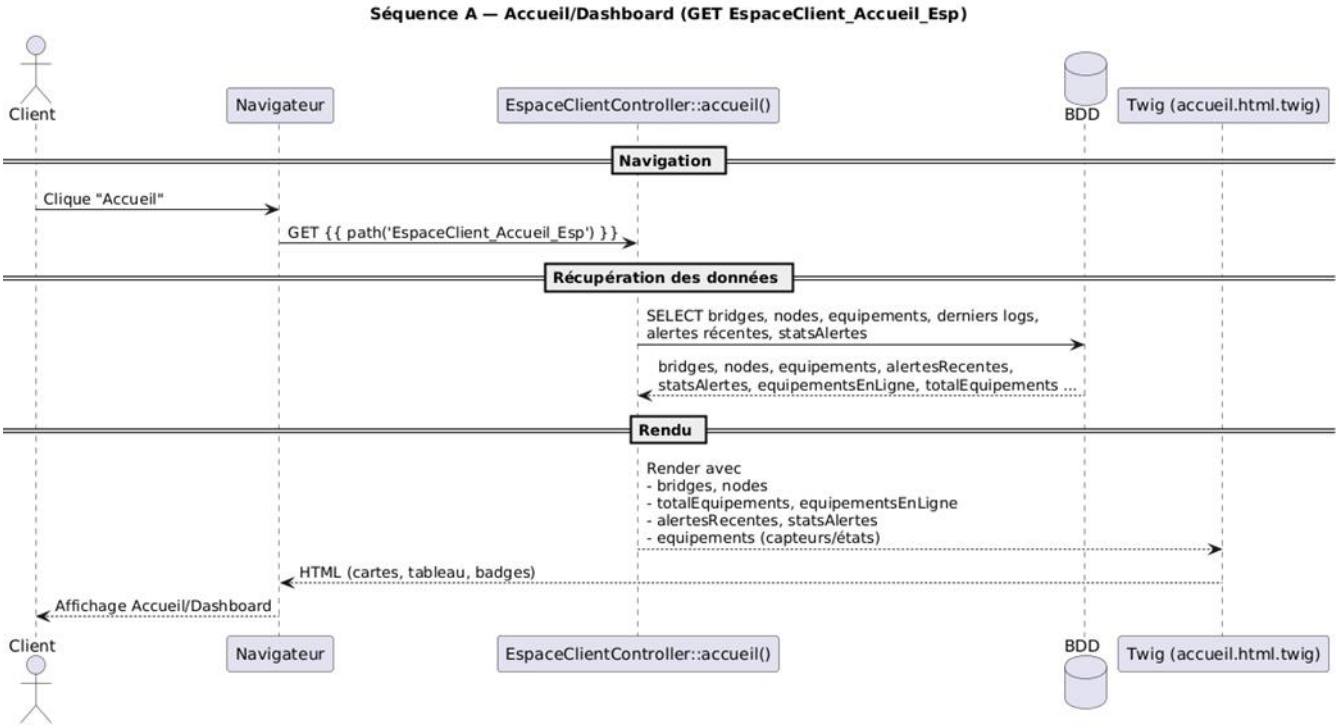
```
  `modalite_paiement` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `mail_facturation` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `telephone_facturation` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `pennylane_id` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_5673580145DE6696` (`site_livraison_id`),
  KEY `IDX_567358019A932B8C` (`site_facturation_id`),
  KEY `IDX_56735801A793976D` (`responsable_du_compte_id`),
  KEY `IDX_5673580173A201E5` (`createur_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS `utilisateur` (
  `id` int NOT NULL AUTO_INCREMENT,
  `comptes_id` int DEFAULT NULL,
  `domicile_livraison_id` int DEFAULT NULL,
  `domicile_facturation_id` int DEFAULT NULL,
  `responsable_du_client_id` int DEFAULT NULL,
  `createur_id` int DEFAULT NULL,
  `mail` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `stripe_customer_id` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `pennylane_id` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `prenom` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `nom` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `motdepasse` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `dateinscription` datetime DEFAULT NULL,
  `datedernierereconnexion` datetime DEFAULT NULL,
  `equipements` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `contrat` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `telephone` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `phpsessid` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `remote_addr` varchar(20) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `mail_verifie` tinyint(1) DEFAULT NULL,
  `role_entreprise` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `numero_utilisateur` varchar(30) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `essai_mdp_restant` int DEFAULT NULL,
  `row_id_dolibarr` int DEFAULT NULL,
  `api_key` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_1D1C63B3DCED588B` (`comptes_id`),
  KEY `IDX_1D1C63B330ABAE05` (`domicile_livraison_id`),
  KEY `IDX_1D1C63B35540CDA2` (`domicile_facturation_id`),
  KEY `IDX_1D1C63B34CBD986C` (`responsable_du_client_id`),
  KEY `IDX_1D1C63B373A201E5` (`createur_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS `domicile` (
  `id` int NOT NULL AUTO_INCREMENT,
  `contact_id` int DEFAULT NULL,
  `installateur_rattache_id` int DEFAULT NULL,
  `entreprise_id` int DEFAULT NULL,
  `nom_domicile` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `adresse_ville` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `adresse_rue` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `adresse_complement` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `adresse_cp` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `statut_telesurveillance` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `prix_telesurveillance` double DEFAULT NULL,
  `statut_maintenance` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
```

```
  `prix_maintenance` double DEFAULT NULL,
  `code_prom` varchar(25) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `nom` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `prenom` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `telephone` varchar(15) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `mail` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `installation_verifie` tinyint(1) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_F6305DA2E7A1254A` (`contact_id`),
  KEY `IDX_F6305DA2E6C2B677` (`installateur_rattache_id`),
  KEY `IDX_F6305DA2A4AEAFEA` (`entreprise_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS `product` (
  `id` int NOT NULL AUTO_INCREMENT,
  `partenaire_selectionne_id` int DEFAULT NULL,
  `fabriquant` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `model` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `price` double DEFAULT NULL,
  `description_courte` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `description` longtext COLLATE utf8mb4_unicode_ci,
  `image` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `image2` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `image3` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `image4` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `image5` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `highlight` varchar(1255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `prix_maintenance` double DEFAULT NULL,
  `caracteristiques` varchar(10000) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `reference` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `poids` double DEFAULT NULL,
  `info_produit_warning` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `present_configurateur` tinyint(1) DEFAULT NULL,
  `type_equipement_configurateur` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `position_configurateur` int DEFAULT NULL,
  `afficher_sur_le_site` tinyint(1) DEFAULT NULL,
  `prix_vente_ht` double DEFAULT NULL,
  `prix_achat_ht` double DEFAULT NULL,
  `en_vente_sur_homeeadmin` tinyint(1) DEFAULT NULL,
  `serial_number_disponible` tinyint(1) DEFAULT NULL,
  `eligible_maintenance` tinyint(1) DEFAULT NULL,
  `code_ean13` varchar(15) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `code_ean13_present` tinyint(1) DEFAULT NULL,
  `format_numero_serie` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `numero_stock` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_D34A04ADC0A1F042` (`partenaire_selectionne_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS `equipement_achetes` (
  `id` int NOT NULL AUTO_INCREMENT,
  `produit_id` int DEFAULT NULL,
  `domicile_installe_id` int DEFAULT NULL,
  `remplace_par_id` int DEFAULT NULL,
  `commande_fournisseur_produit_id` int DEFAULT NULL,
  `produit_panier_id` int DEFAULT NULL,
  `projet_id` int DEFAULT NULL,
  `maintenance_id` int DEFAULT NULL,
```

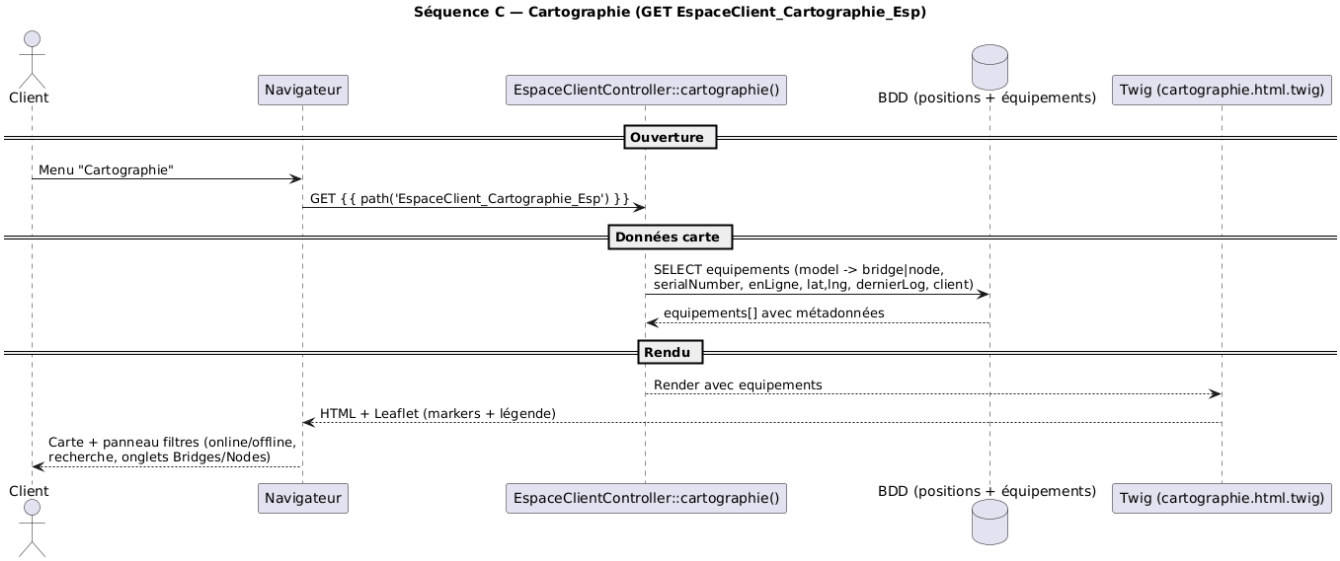
```
  `statut_equipement` varchar(20) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `date_rma` datetime DEFAULT NULL,
  `serial_number` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `date_reception_client` datetime DEFAULT NULL,
  `info_position` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `numero_equipement` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `mot_passe_equipement` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `UNIQ_1284EB5DA7377412` (`remplace_par_id`),
  KEY `IDX_1284EB5DF347EFB` (`produit_id`),
  KEY `IDX_1284EB5D18C74B5` (`domicile_installe_id`),
  KEY `IDX_1284EB5D1438A4C2` (`commande_fournisseur_produit_id`),
  KEY `IDX_1284EB5D73EACC90` (`produit_panier_id`),
  KEY `IDX_1284EB5DC18272` (`projet_id`),
  KEY `IDX_1284EB5DF6C202BC` (`maintenance_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS `equipement_achetes_logs` (
  `id` int NOT NULL AUTO_INCREMENT,
  `equipement_achetes_id` int DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  `cle` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `valeur` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `alerte` tinyint(1) DEFAULT NULL,
  `titre_alerte` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `description_alerte` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `severity_alerte` varchar(20) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `vu_par_client` tinyint(1) DEFAULT NULL,
  `is_active` tinyint(1) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_F29FB88B557A15DF` (`equipement_achetes_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS `equipement_achetes_metadata` (
  `id` int NOT NULL AUTO_INCREMENT,
  `equipement_achetes_id` int DEFAULT NULL,
  `cle` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `valeur` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_E348313557A15DF` (`equipement_achetes_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Annexes diagrammes de séquence des cas d'utilisations les plus significatifs :

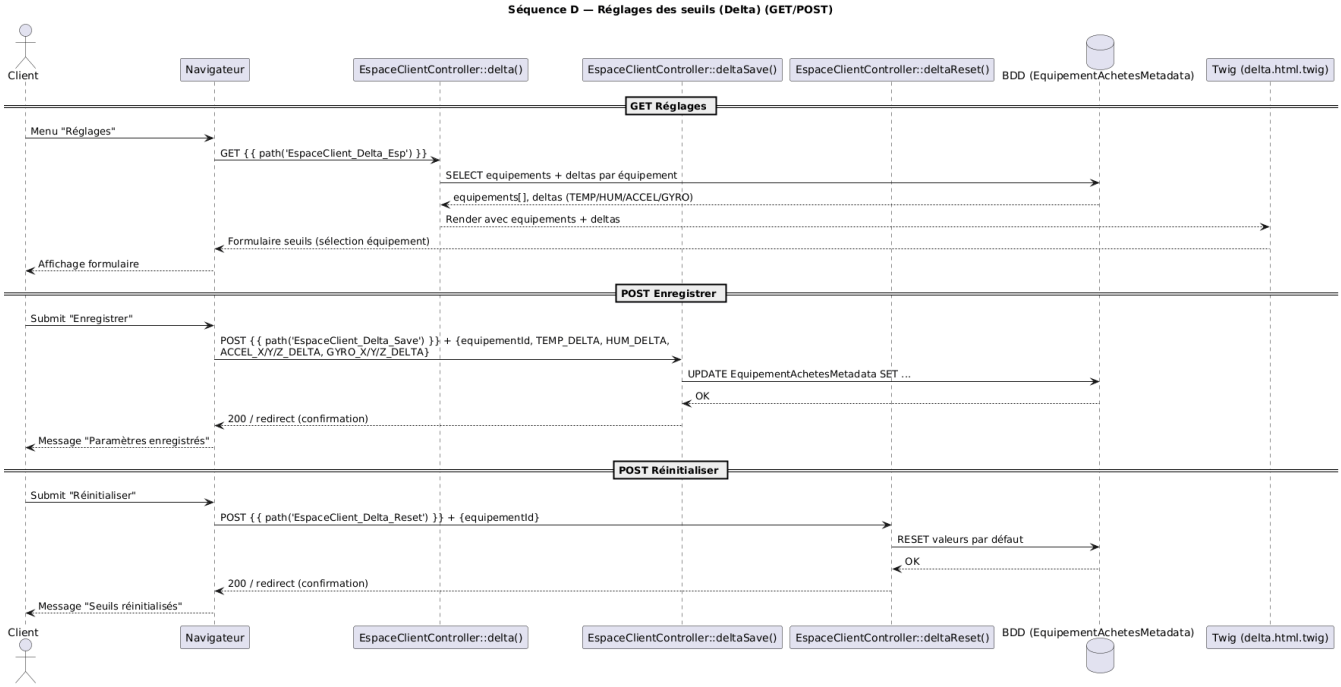
Séquence A : Ce diagramme illustre le déroulement complet du chargement de la page d'accueil du dashboard : lorsqu'un client clique sur "Accueil", le contrôleur Symfony récupère les équipements, bridges, nodes, alertes récentes et statistiques en base de données, puis génère la page Twig affichant les indicateurs clés (cartes, tableaux et badges) avant de renvoyer l'interface finalisée au navigateur.



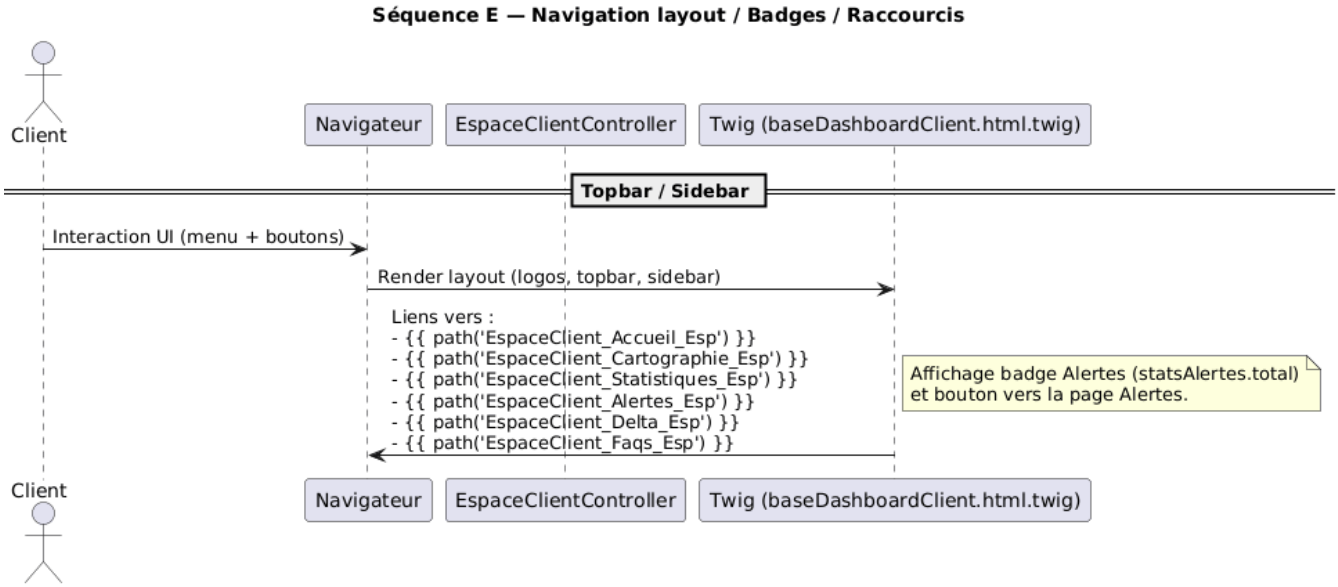
Séquence C : Ce diagramme montre le processus d'affichage de la carte : lorsque le client ouvre la page "Cartographie", le contrôleur Symfony récupère en base les équipements (bridges et nodes) avec leur position et leur état, puis génère la vue Twig qui affiche une carte Leaflet contenant les marqueurs, la légende et les filtres (online/offline).



Séquence D : Ce diagramme illustre le fonctionnement de la page “Réglages / Delta” : lorsqu’un client ouvre cette section, le contrôleur récupère les équipements et leurs seuils configurés, puis affiche le formulaire. Lorsqu’il enregistre ou réinitialise les valeurs, une requête POST met à jour ou remet à zéro les métadonnées en base de données, avant de renvoyer une confirmation au client.

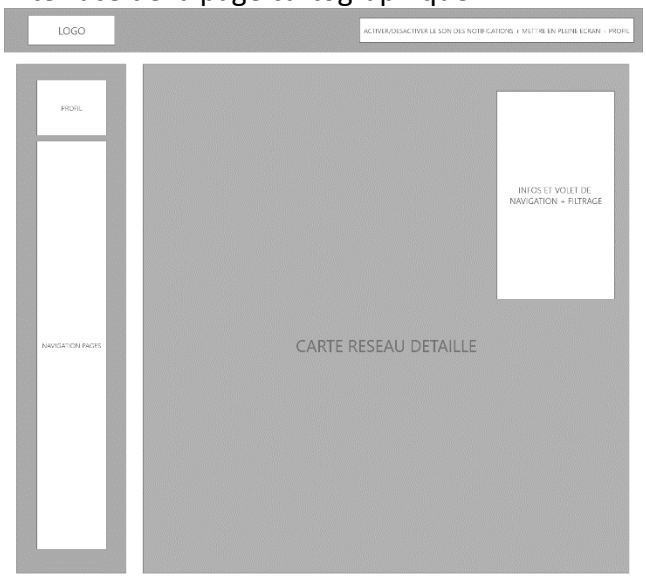


Séquence E : Ce diagramme illustre le fonctionnement du layout principal du dashboard Homeeguard : à chaque interaction de l’utilisateur avec la navigation (menu ou boutons), Symfony rend le layout commun (topbar, sidebar, liens de navigation) et y injecte les raccourcis dynamiques, comme le badge d’alertes, permettant un accès direct aux différentes pages de l’espace client.

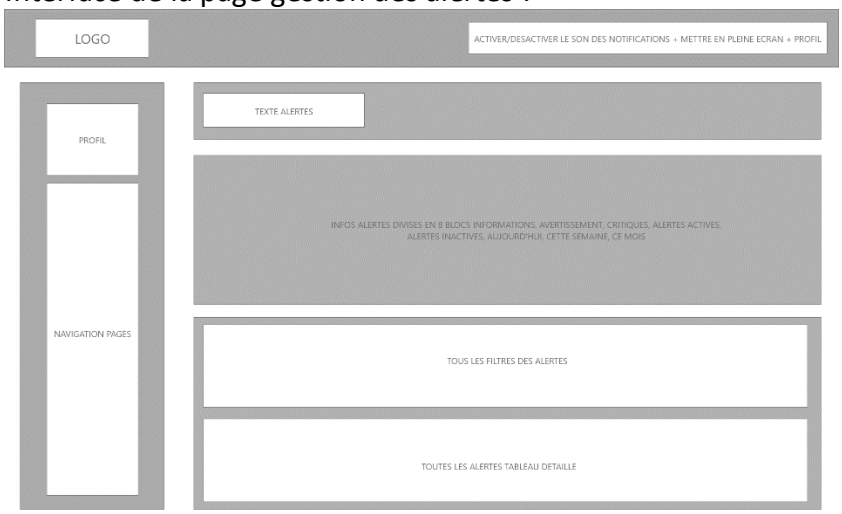


Annexes prototypage :

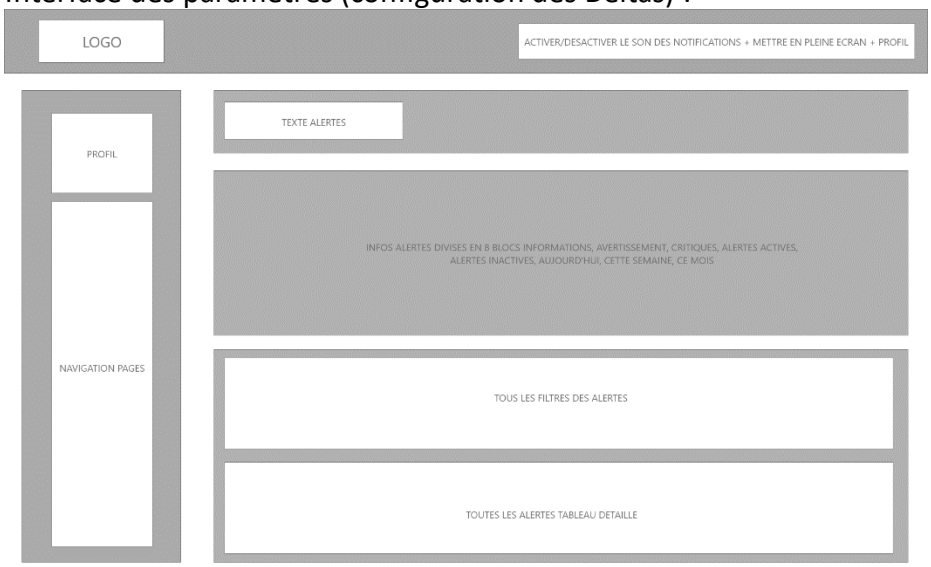
Interface de la page cartographique :



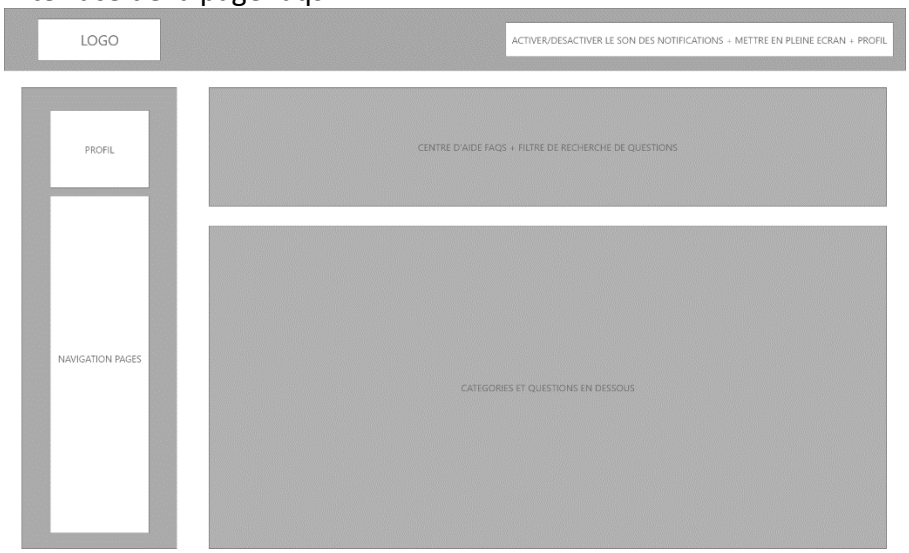
Interface de la page gestion des alertes :



Interface des paramètres (configuration des Deltas) :



Interface de la page faqs :



Annexes Réalisations :

Les Nodes ESP32 (capteurs LoRa) :

```
#include <SPI.h>
#include <RH_RF95.h>
#include <RHMesh.h>
#include <AES.h>
#include "xbase64.h"
#include <DHT.h>
#include <esp_sleep.h>

// ==== Identité & réseau ====
#define NODE_ADDRESS 3
#define BRIDGE_ADDR  1
#define CLIENT_ID    "CONT-202405-7549"
#define SERIAL_NUM   "1234567890"
#define RF_FREQ      868.0
#define RF_TX_PWR    15
#define MAX_RETRY    3
#define SLEEP_MS     10000UL    // 10 s

// ==== Capteurs ====
#define DHTPIN  14
#define DHTTYPE DHT22
#define SW_PIN  12
DHT dht(DHTPIN, DHTTYPE);

// ==== Radio ====
RH_RF95 rf95(18, 26);      // NSS=18, DIO0=26 (adapter à ton PCB)
RHMesh  mesh(rf95, NODE_ADDRESS);

// ==== Crypto (AES-128/CBC) ====
AES aes;
const uint8_t AES_KEY[16] =
{0x2B,0x7E,0x15,0x16,0x28,0xAE,0xD2,0xA6,0xAB,0xF7,0x15,0x88,0x09,0xCF,0x4F,0x3C};
const uint8_t AES_IV [16] =
{0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F};

String aesEncrypt(const String& plain) {
  const int pad = ((plain.length()/16)+1)*16;
  byte in[pad]={0}, out[pad], iv[16]; memcpy(iv,AES_IV,16);
  plain.getBytes(in, plain.length()+1);
  aes.set_key(AES_KEY, sizeof(AES_KEY));
  aes.cbc_encrypt(in, out, pad/16, iv);
  char b64[base64_enc_len(pad)];
  base64_encode(b64, (char*)out, pad);
  return String(b64);
}

// ==== Envoi fiable (avec retry) ====
```

```
bool sendEncrypted(const String& payloadB64) {
  for (int i=0;i<MAX_RETRY;i++) {
    if (mesh.sendtoWait((uint8_t*)payloadB64.c_str(), payloadB64.length(), BRIDGE_ADDR) ==
RH_ROUTER_ERROR_NONE)
      return true;
    delay(400*(i+1));
  }
  return false;
}

// ==== Mesure + envoi ====
void sendSensors() {
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  int sw  = digitalRead(SW_PIN)==LOW;     // contact actif = LOW

  char msg[120];
  snprintf(msg, sizeof(msg),
           "NODE:%d,CLIENT:%s,SERIAL:%s,TEMP:%.1f,HUM:%.1f,SW:%d",
           NODE_ADDRESS, CLIENT_ID, SERIAL_NUM, t, h, sw);

  String enc = aesEncrypt(String(msg));
  bool ok = sendEncrypted(enc);
  Serial.println(ok ? F("[TX] OK") : F("[TX] FAIL"));
}

void sendKeepAlive() {
  char ka[80];
  snprintf(ka, sizeof(ka), "KA:%d,CLIENT:%s,SERIAL:%s", NODE_ADDRESS, CLIENT_ID, SERIAL_NUM);
  (void)sendEncrypted(aesEncrypt(String(ka)));
  Serial.println(F("[KA] envoyé"));
}

// ==== Cycle basse conso ====
void deepSleepMs(uint32_t ms) {
  esp_sleep_enable_timer_wakeup((uint64_t)ms*1000ULL);
  Serial.flush();
  esp_deep_sleep_start();
}

// ==== Setup / Loop ====
void setup() {
  Serial.begin(115200);
  dht.begin();
  pinMode(SW_PIN, INPUT_PULLUP);

  SPI.begin(5,19,27,18);                // SCK,MISO,MOSI,SS (adapter si besoin)
  if (!mesh.init()) { Serial.println(F("Radio init FAIL")); deepSleepMs(SLEEP_MS); }
  rf95.setTxPower(RF_TX_PWR);
  rf95.setFrequency(RF_FREQ);
  Serial.println(F("Node prêt"));
}

void loop() {
  sendSensors();
  sendKeepAlive();
  deepSleepMs(SLEEP_MS);
}
```

Les Bridges ESP32 (passerelles Wi-Fi LoRa) :

```
#include <SPI.h>
#include <RH_RF95.h>
#include <RHMesh.h>
#include <AES.h>
#include "xbase64.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

// === Réseau / API (à déplacer en config.h) ===
const char* WIFI_SSID="SSID";
const char* WIFI_PASS="MDP";
const char* API_URL ="http://192.168.1.138:8000/api/lora-data";
#define API_KEY "API_KEY"

// === Identité / radio ===
```

```
#define BRIDGE_ADDR 1
#define CLIENT_ID   "CONT-202405-7549"
#define SERIAL_BR   "1234567891"
#define RF_FREQ     868.0
#define RF_TX_PWR   15

// === LoRa (pins à adapter) ===
#define LORA_SS 18
#define LORA_DIO0 26
RH_RF95 rf95(LORA_SS, LORA_DIO0);
RHMesh  mesh(rf95, BRIDGE_ADDR);

// === AES-128/CBC ===
AES aes;
const uint8_t
AES_KEY[16]={0x2B,0x7E,0x15,0x16,0x28,0xAE,0xD2,0xA6,0xAB,0xF7,0x15,0x88,0x09,0xCF,0x4F,0x3C};
const uint8_t AES_IV [16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

String aesEncrypt(const String& s){
  int pad=((s.length()/16)+1)*16; byte in[pad]={0}, out[pad], iv[16]; memcpy(iv,AES_IV,16);
  s.getBytes(in,s.length()+1); aes.set_key(AES_KEY,sizeof(AES_KEY));
aes.cbc_encrypt(in,out,pad/16,iv);
  char b64[base64_enc_len(pad)]; base64_encode(b64,(char*)out,pad); return String(b64);
}
String aesDecrypt(const String& b64){
  int inLen=b64.length(), decLen=base64_dec_len((char*)b64.c_str(),inLen);
  byte in[decLen], out[decLen], iv[16]; base64_decode((char*)in,b64.c_str(),inLen);
memcpy(iv,AES_IV,16);
  aes.set_key(AES_KEY,sizeof(AES_KEY)); aes.cbc_decrypt(in,out,decLen/16,iv);
  String s=(char*)out; s.trim(); return s;
}

// === Utils ===
void wifiEnsure(){
  if(WiFi.status()==WL_CONNECTED) return;
  WiFi.begin(WIFI_SSID,WIFI_PASS); unsigned long t0=millis();
  while(WiFi.status()!=WL_CONNECTED && millis()-t0<10000) delay(250);
}
String fieldOf(const String& msg,const char* key){
  int i=msg.indexOf(key); if(i<0) return "";
  int b=i+strlen(key), e=msg.indexOf(',',b); if(e<0) e=msg.length();
  return msg.substring(b,e);
}

// === Envoi API ===
void sendToApi(uint8_t node,const String& plain){
  wifiEnsure(); if(WiFi.status()!=WL_CONNECTED){ Serial.println("[API] WiFi KO"); return; }
  DynamicJsonDocument doc(512);
  doc["clientId"]=CLIENT_ID; doc["bridgeId"]=String(BRIDGE_ADDR); doc["nodeId"]=String(node);
doc["online"]=true;
  String sn=fieldOf(plain,"SERIAL:"); doc["serialNumber"]=sn.length()?sn:SERIAL_BR;

  if(plain.startsWith("KA:")){ String b=fieldOf(plain,"BATT="); if(b.length())
doc["batteryVoltage"]=b.toFloat(); }
  else{
    String t=fieldOf(plain,"TEMP:"), h=fieldOf(plain,"HUM:"), sw=fieldOf(plain,"SW:");
    if(t.length()) doc["temperature"]=t.toFloat();
    if(h.length()) doc["humidite"]=h.toFloat();
    if(sw.length()) doc["interrupteur"]=sw.toInt()==1;
    String ax=fieldOf(plain,"AX:"), ay=fieldOf(plain,"AY:"), az=fieldOf(plain,"AZ:");
    String gx=fieldOf(plain,"GX:"), gy=fieldOf(plain,"GY:"), gz=fieldOf(plain,"GZ:");
    if(ax.length()) doc["accX"]=ax.toInt(); if(ay.length()) doc["accY"]=ay.toInt(); if(az.length())
doc["accZ"]=az.toInt();
    if(gx.length()) doc["gyroX"]=gx.toInt(); if(gy.length()) doc["gyroY"]=gy.toInt(); if(gz.length())
doc["gyroZ"]=gz.toInt();
  }

  String body; serializeJson(doc, body);
  HTTPClient http; http.begin(API_URL);
  http.addHeader("Content-Type","application/json"); http.addHeader("X-API-KEY",API_KEY);
  int code=http.POST(body); Serial.printf("[API] %d %s\n",code,body.c_str()); http.end();
}

// === Réponse Node ===
bool sendResponseToNode(uint8_t to,const String& plain){
  String enc=aesEncrypt(plain);
```

```
  for(int i=0;i<3;i++){
if(mesh.sendtoWait((uint8_t*)enc.c_str(),enc.length(),to)==RH_ROUTER_ERROR_NONE) return true;
delay(200*(i+1)); }
  return false;
}

// === Setup / Loop ===
void setup(){
  Serial.begin(115200); WiFi.mode(WIFI_STA); wifiEnsure();
  SPI.begin(5,19,27,18);
  if(!mesh.init()) Serial.println("LoRa init FAIL");
  rf95.setFrequency(RF_FREQ); rf95.setTxPower(RF_TX_PWR);
  Serial.println("Bridge prêt");
}
void loop(){
  uint8_t buf[RH_MESH_MAX_MESSAGE_LEN], len=sizeof(buf), from;
  if(mesh.recvfromAck(buf,&len,&from)){
    buf[len]='\0'; String plain=aesDecrypt((char*)buf);
    Serial.printf("[RX] #%d : %s\n",from,plain.c_str());
    sendToApi(from,plain);
    char tbuf[32]; snprintf(tbuf,sizeof(tbuf),"TIME=%lu",(unsigned long)(millis()/1000UL));
    Serial.println(sendResponseToNode(from,String(tbuf))?"[TX] TIME OK":"[TX] TIME FAIL");
  }
  delay(5);
}
```

L'API Serveur (Réception des données et implémentation en base de données) :

```
<?php
// src/Controller/ApiController.php

namespace App\Controller;

use App\Entity\Comptes;
use App\Entity\EquipementAchetes;
use App\Entity\EquipementAchetesLogs;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

class ApiController
{
    #[Route('/api/lora-data', name: 'api_lora_data', methods: ['POST'])]
    public function loraData(Request $request, EntityManagerInterface $em): JsonResponse
    {
        // Vérif format JSON
        $data = json_decode($request->getContent(), true);
        if (!$data || !isset($data['clientId'])) {
            return new JsonResponse(['error' => 'Format JSON invalide'], 400);
        }

        // Authentification par clé API (client)
        $client = $em->getRepository(Comptes::class)->findOneBy(['apiKey' => $data['clientId']]);
        if (!$client) {
            return new JsonResponse(['error' => 'Clé API invalide'], 401);
        }

        // Rattache au bon équipement via le bridge (serialNumber stocké côté équipements)
        // NB: ton code source cherchait bridgeId -> serialNumber
        $bridgeId = $data['bridgeId'] ?? null;
        if (!$bridgeId) {
            return new JsonResponse(['error' => 'bridgeId manquant'], 400);
        }

        $equipement = $em->getRepository(EquipementAchetes::class)
            ->findOneBy(['serialNumber' => $bridgeId]);

        if (!$equipement) {
            return new JsonResponse(['error' => 'Équipement non trouvé'], 404);
        }

        // Log des mesures (champs optionnels tolérés)
        $log = new EquipementAchetesLogs();
        $log->setEquipement($equipement);

        if (array_key_exists('temperature', $data)) {
```

```
            $log->setTemperature($data['temperature']);
        }
        if (array_key_exists('humidity', $data)) {
            $log->setHumidity($data['humidity']);
        }
        if (array_key_exists('accX', $data)) {
            $log->setAccX($data['accX']);
        }
        if (array_key_exists('accY', $data)) {
            $log->setAccY($data['accY']);
        }
        if (array_key_exists('accZ', $data)) {
            $log->setAccZ($data['accZ']);
        }

        $log->setHorodatage(new \DateTimeImmutable());

        $em->persist($log);
        $em->flush();

        // Réponse simple (peut être enrichie d'un timestamp/infosConfig si nécessaire)
        return new JsonResponse([
            'status'  => 'success',
            'message' => 'Données enregistrées',
        ], 200);
    }
}
```

EspaceClientController::accueil() (Page d'accueil)

```
<?php
// src/Controller/EspaceClientController.php

namespace App\Controller;

use App\Repository\EquipementAchetesLogsRepository;
use App\Repository\EquipementAchetesMetadataRepository;
use App\Repository\EquipementAchetesRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class EspaceClientController extends AbstractController
{
    #[Route('/espace-client/accueil', name: 'espace_client_accueil', methods: ['GET'])]
    public function accueil(
        EquipementAchetesRepository $repoEquipement,
        EquipementAchetesLogsRepository $repoLogs,
        EquipementAchetesMetadataRepository $repoMeta
    ): Response {
        $user = $this->getUser();

        // Récupère les équipements du client connecté
        $equipements = $repoEquipement->findBy(['client' => $user]);

        // Méthodes existantes dans tes repos (selon ton code)
        $alertesRecentes    = $repoLogs->findRecent($user);
        $equipementsEnLigne = $repoMeta->countOnline($user);

        return $this->render('accueil.html.twig', [
            'totalEquipements'    => \count($equipements),
            'equipementsEnLigne'  => $equipementsEnLigne,
            'alertesRecentes'     => $alertesRecentes,
        ]);
    }
}
```

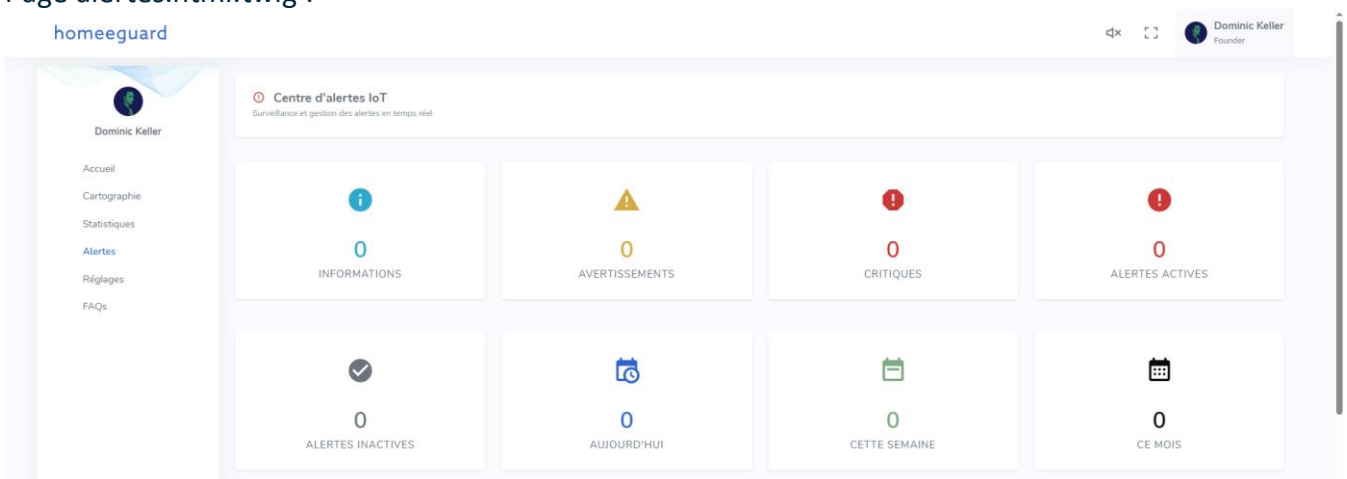
EspaceClientController::EspaceClient_Delta_Save() (Fonction pour sauvegarder les deltas sur la page paramètre)

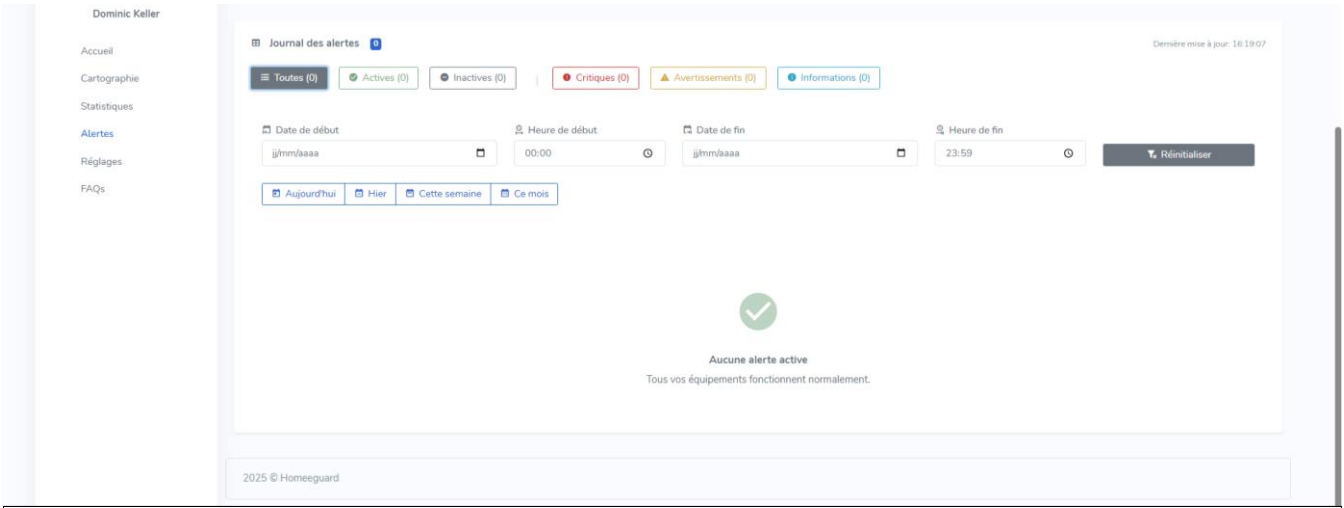
```
<?php
// src/Controller/EspaceClientController.php (extrait)
```

```
namespace App\Controller;

use App\Entity\EquipementAchetes;
use App\Entity\EquipementAchetesMetadata;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\RedirectResponse;

class EspaceClientController extends AbstractController
{
    #[Route('/espace-client/delta/{id}/save', name: 'EspaceClient_Delta_Save', methods: ['POST'])]
    public function deltaSave(
        EquipementAchetes $equipement,
        Request $request,
        EntityManagerInterface $em
    ): RedirectResponse {
        // (Optionnel mais recommandé) Vérifier un token CSRF si le formulaire l'inclut
        // $this->isCsrfTokenValid('delta_save_'.$equipement->getId(), $request->request-
>get('_token'))

        $repoMeta = $em->getRepository(EquipementAchetesMetadata::class);

        // Parcourt uniquement les clés *_DELTA
        foreach ($request->request->all() as $cle => $valeur) {
            if (str_contains($cle, '_DELTA')) {
                $meta = $repoMeta->findOneBy([
                    'equipement' => $equipement,
                    'cle'        => $cle,
                ]);

                if (!$meta) {
                    $meta = new EquipementAchetesMetadata();
                    $meta->setEquipement($equipement);
                    $meta->setCle($cle);
                }

                // Cast simple : laisse la BDD stocker string/float selon mapping
                $meta->setValeur($valeur);
                $em->persist($meta);
            }
        }

        $em->flush();
        $this->addFlash('success', 'Configuration sauvegardée');

        // Redirige vers la page Delta de cet équipement (adapter la route si besoin)
        return $this->redirectToRoute('espace_client_accueil');
    }
}
```

Annexes captures d'écran d'interfaces utilisateur :

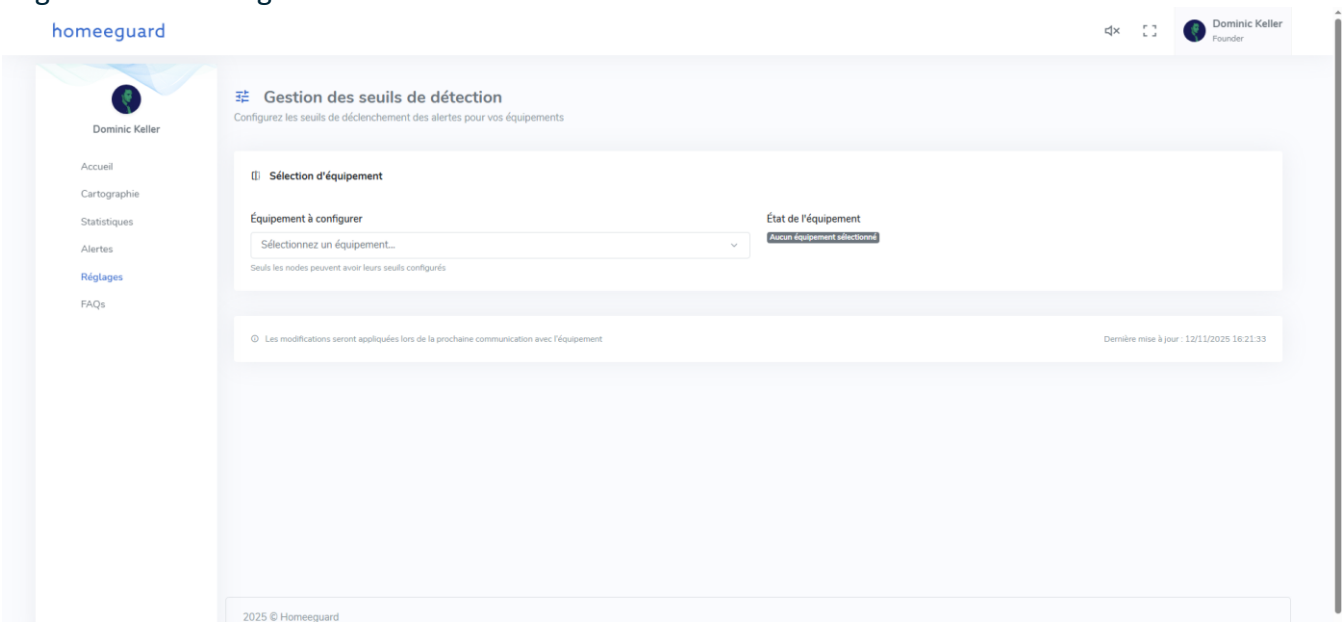
Page alertes.html.twig :

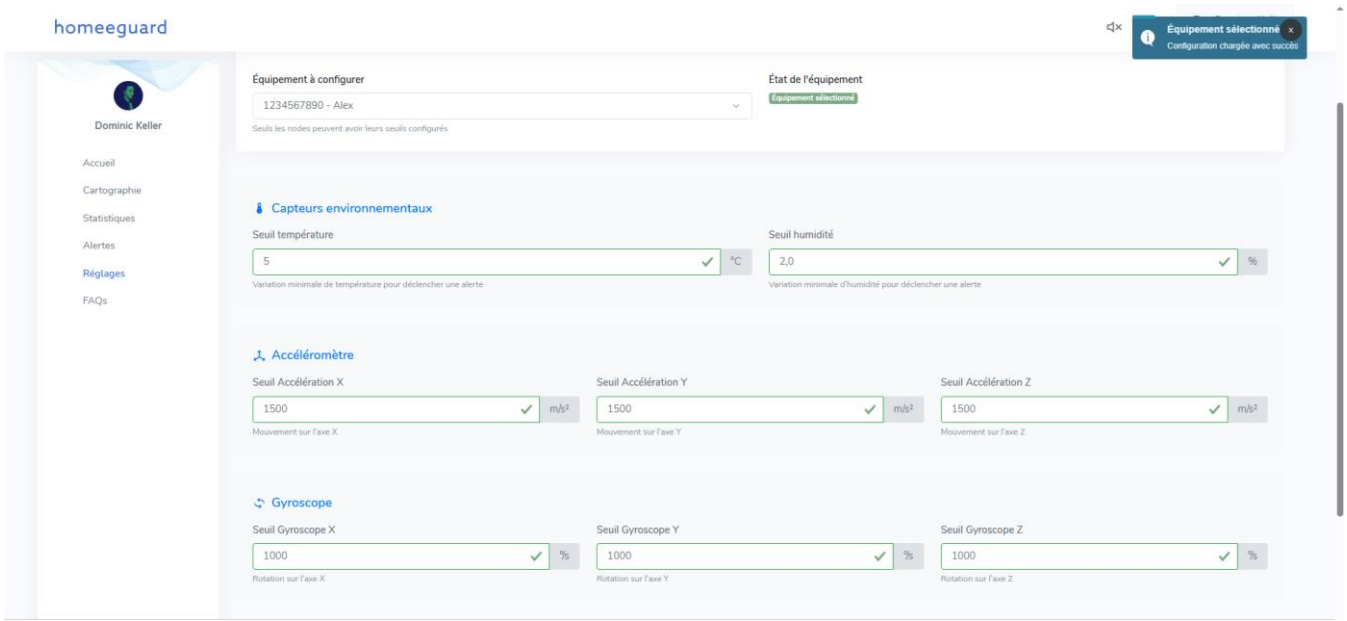





```
<div class="table-responsive">
  <table id="alertes-datatable" class="table table-hover mb-0">
    <thead class="table-light">
      <tr>
        <th width="5%"><i class="mdi mdi-alert-circle-outline"></i></th>
        <th width="15%">Date/Heure</th>
        <th width="15%">Équipement</th>
        <th width="10%">Type</th>
        <th width="10%">Gravité</th>
        <th width="20%">Titre</th>
        <th width="10%">Statut</th>
        <th width="10%">Actions</th>
      </tr>
    </thead>
    <tbody>
      {% for alerte in alertes %}
        {# …ligne d’alerte construite depuis les propriétés #}
      {% endfor %}
    </tbody>
  </table>
</div>
```

Page delta.html.twig :

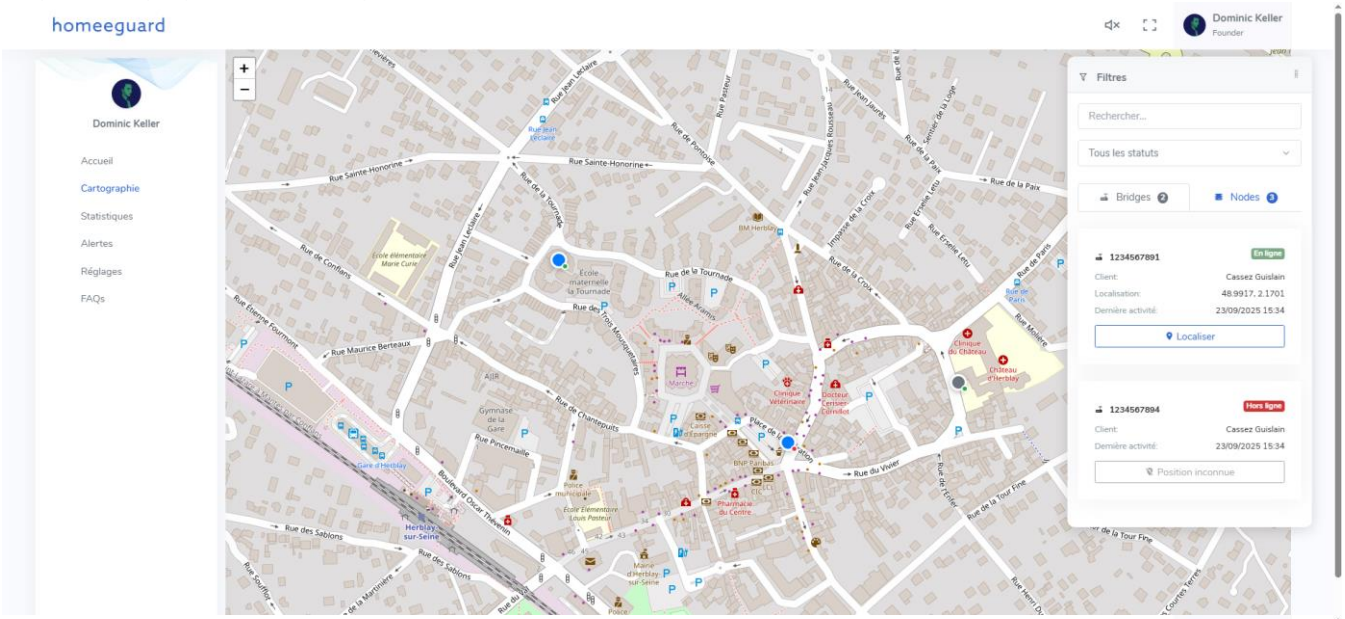


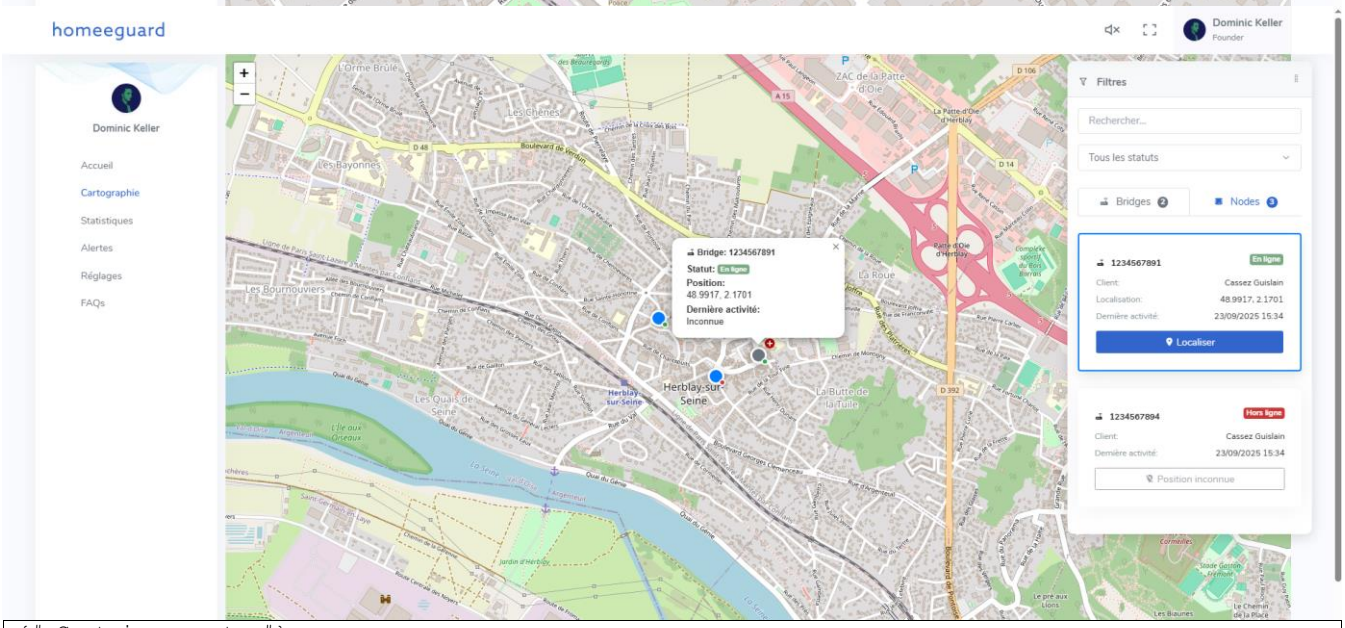



```
{# Section capteurs environnementaux #}
<h4 class="section-title"><i class="mdi mdi-thermometer"></i> Capteurs environnementaux</h4>
<div class="row">
  <div class="col-md-6">
    <label for="tempDelta" class="form-label">Seuil température</label>
    <div class="input-group">
      <input type="number" class="form-control" id="tempDelta" name="TEMP_DELTA" value="1.0" min="0.1"
max="10" step="0.1" required>
      <span class="input-group-text">°C</span>
    </div>
  </div>
  <div class="col-md-6">
    <label for="humDelta" class="form-label">Seuil humidité</label>
    <div class="input-group">
      <input type="number" class="form-control" id="humDelta" name="HUM_DELTA" value="2.0" min="0.5"
max="20" step="0.5" required>
      <span class="input-group-text">%</span>
    </div>
  </div>
</div>
```

```
{# Actions + infos #}
<div class="d-flex justify-content-end gap-2">
  <button type="button" class="btn btn-reset" id="btnReset"><i class="mdi mdi-refresh"></i>
Réinitialiser</button>
  <button type="submit" class="btn btn-save"><i class="mdi mdi-content-save"></i> Enregistrer</button>
</div>
<div class="card delta-card mt-4">
  <div class="card-body d-flex justify-content-between">
    <small class="text-muted"><i class="mdi mdi-information-outline me-1"></i> Les modifications
seront appliquées lors de la prochaine communication…</small>
    <small class="text-muted">Dernière mise à jour : {{ "now"|date("d/m/Y H:i:s") }}</small>
  </div>
</div>
```

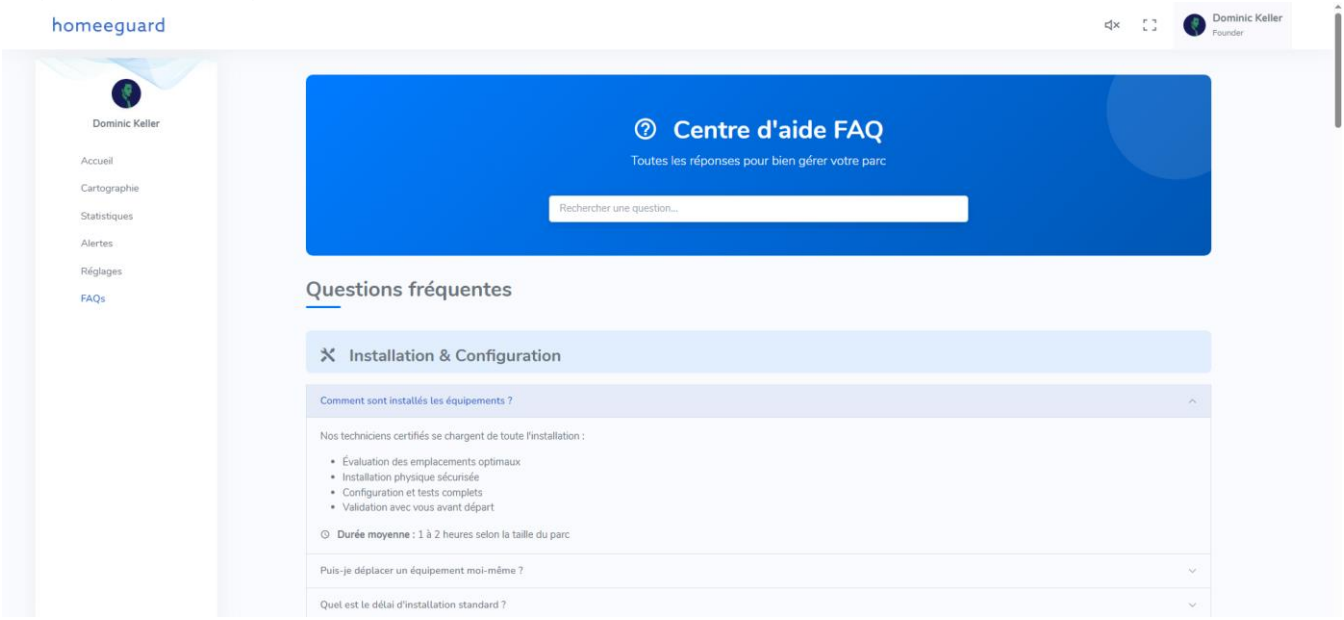
Page cartographie.html.twig :






```
{# Container carte #}
<div class="card">
  <div class="card-body p-0">
    <div id="map" style="height: 520px;"></div>
  </div>
</div>

{# Initialisation (pilotée par map-manager.js) #}
<script src="{{ asset('js/Homeeguard/EspaceClient/Dashboard/map/map-manager.js') }}"></script>
```

Page faqs.html.twig :

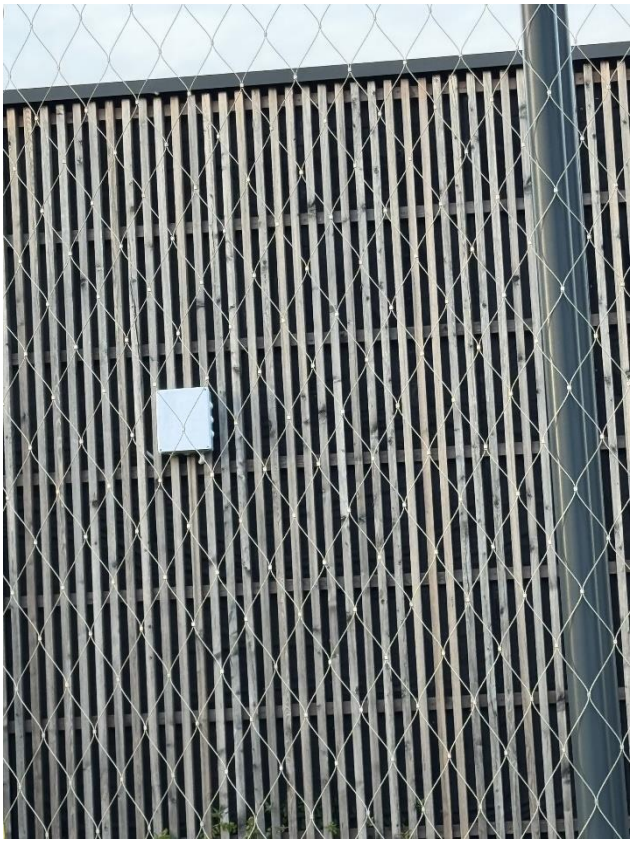


Annexes plan de tests :

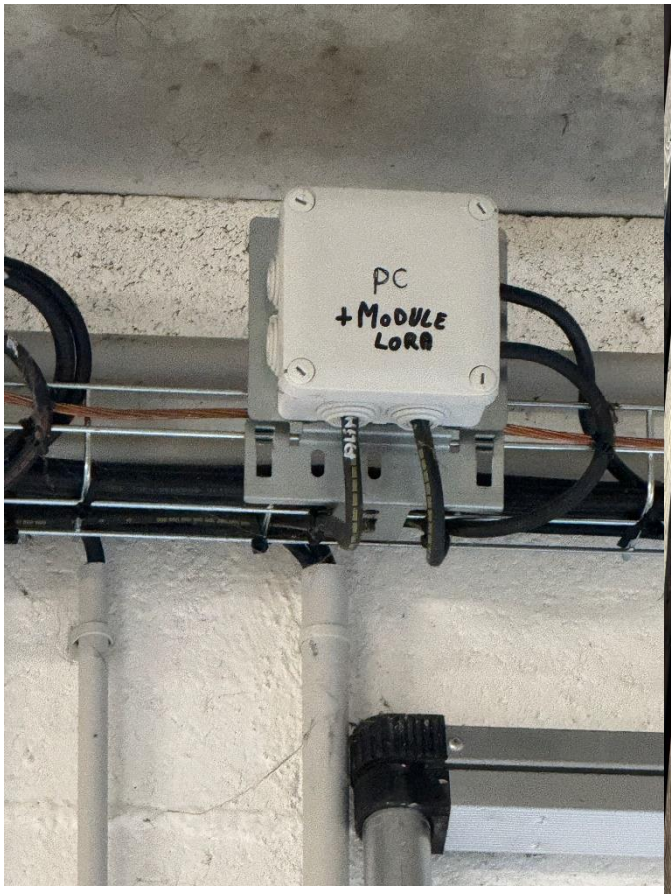
Module	Description du test	Entrées / Préconditions	Résultat (attendu → obtenu)
Bridge (Wi-Fi)	Connexion au SSID Homeeguard	SSID/mot de passe valides	Connecté, IP locale → ✅ OK
Bridge (Wi-Fi)	Reconnexion après perte réseau	Router OFF/ON	Reconnexion < 10 s → ✅ OK
Bridge (Wi-Fi)	Timeout HTTP	API indisponible	Retour code erreur + retry → ✅ OK
Bridge (HTTP)	Envoi POST JSON node→API	JSON complet (clientId, bridgeId…)	HTTP 200 + “success” → ✅ OK
Bridge (HTTP)	En-tête X-API-KEY	Clé présente	Accepté (200) → ✅ OK
Bridge (HTTP)	X-API-KEY manquante	En-tête absent	401 Unauthorized → ✅ OK
Bridge (HTTP)	JSON invalide	Payload tronqué	400 Bad Request → ✅ OK
Bridge (SPIFFS)	Log en local si Wi-Fi KO	Couper Wi-Fi	Append /bridge1_log.txt → ✅ OK
Bridge (SPIFFS)	Rotation des logs	> 90% utilisé	Purge 10% ancien → ✅ OK
Bridge (SPIFFS)	Intégrité logs	VERIFY_LOGS	“Intégrité OK” → ✅ OK
Bridge (radio)	Reset radio logiciel	RESET_RADIO	Re-init RH_RF95 OK → ✅ OK
Bridge (résilience)	Auto-reboot après erreurs	MAX_CONSECUTIVE_ERRORS atteint	ESP.restart() → ✅ OK
Bridge (horloge)	Horodatage global	TIMESTAMP seed	TIME stable + incremente → ✅ OK
Bridge (security)	Chiffrement AES TX	encryptMessage() activé	Trame base64 non lisible → ✅ OK
Bridge (security)	Déchiffrement AES RX	decryptMessage()	Message clair identique → ✅ OK
Bridge (fallback)	Transfert vers bridge secondaire	TRANSFER: actif	“Transfert réussi” → ✅ OK
Node (capteurs)	Lecture DHT	Temp/Hum réelles	TEMP/HUM plausibles → ✅ OK
Node (capteurs)	Lecture MPU6050	Mouvement simulé	AX/AY/AZ varient → ✅ OK
Node (IO)	Interrupteur	Pin change d’état	D=1/0 dans trame → ✅ OK
Node (keepalive)	KA périodique	Inactivité	KA: + online:true → ✅ OK
Node (delta)	Envoi si delta dépassé	TEMP_DELTA=1.0	Envoi uniquement si Δ≥1 → ✅ OK
Node (énergie)	DeepSleep si échecs	MAX_RETRIES atteint	Entrée deep sleep → ✅ OK
Node (fallback)	Bridge secondaire	Échecs sur primaire	Bascule auto → ✅ OK
Node (horloge)	Réception TIME	Réponse bridge	RTC local mis à jour → ✅ OK
API (auth)	Clé API valide	Comptes.apiKey	200 + insert log → ✅ OK
API (auth)	Clé API invalide	clientId faux	401 + message générique → ✅ OK
API (validation)	JSON requis	clientId manquant	400 → ✅ OK
API (contrôle)	Bridge/equipement liés	bridgeId inconnu	404 “Équipement non trouvé” → ✅ OK
API (sécurité)	Client “particulier”	userEmail lié	200 si même compte → ✅ OK
API (sécurité)	Particulier non lié	userEmail ≠ compte	403 → ✅ OK
API (BD)	Insertion log	Données capteurs valides	Ligne créée en BDD → ✅ OK
API (BD)	Types/NULL safety	Valeurs manquantes	Champs NULL autorisés traités → ✅ OK
API (perfs)	50 POST/10 s	Postman runner	p95 < 200 ms → ✅ OK
API (robustesse)	BDD down	Connexion fermée	500 propre, pas de crash → ✅ OK
BDD (modèle)	Intégrité FK	Mappings Doctrine	Aucune FK cassée → ✅ OK
BDD (cascade)	Suppression compte	Remove Comptes	Équipements/logs en cascade → ✅ OK
BDD (metadata)	Sauvegarde deltas	*_DELTA POST	Maj EquipementAchetesMetadata → ✅ OK
BDD (indexes)	Requêtes rapides	Index sur clés	Temps de sélection stable → ✅ OK
Front (login)	Authentification	Identifiants valides	Redirection /accueil → ✅ OK
Front (session)	Déconnexion	Bouton logout	Redirection /login → ✅ OK

Front (accueil)	KPIs (total/en ligne)	Données BDD	Valeurs correctes → ✅ OK
Front (alertes)	DataTable	500 lignes	Tri/recherche fluides → ✅ OK
Front (alertes)	Filtre JS dédié	alertes-filters.js	Filtrage instantané → ✅ OK
Front (delta)	Formulaire deltas	CSRF actif	“Configuration sauvegardée” → ✅ OK
Front (delta)	Validation mauvaise saisie	Valeur hors bornes	Message erreur, pas d’insert → ✅ OK
Front (stats)	Graphes	Données réelles	Courbes cohérentes → ✅ OK
Front (carte)	Leaflet	Coord. en base	Marqueurs visibles → ✅ OK
Front (notif)	Pop-up + son	Alerte critique simulée	Toast + son joué → ✅ OK
Front (assets)	JS/CSS chargés	Fichiers présents	Aucune 404 → ✅ OK
Sécurité (XSS)	Échappement Twig	<script> saisi	Texte brut, pas exécuté → ✅ OK
Sécurité (CSRF)	Token requis	POST sans token	Rejeté → ✅ OK
Sécurité (SQLi)	Injection sur clientId	clientId="1 OR 1=1"	Doctrine bloque (paramétré) → ✅ OK
Sécurité (replay)	Rejeu trame LoRa	TIME ancien	Ignorée → ✅ OK
Sécurité (exposition)	Messages d’erreur	Erreurs API	Sans détails sensibles → ✅ OK
Perfs (JS)	DataTable 1k lignes	Dataset volumineux	Navigation fluide → ✅ OK
Perfs (radio)	Latence LoRa	Node→Bridge	~80–120 ms → ✅ OK
Perfs (stockage)	Mesure SPIFFS	>80% utilisé	Rotation pro-active → ✅ OK
Résilience (redém.)	Commande RESTART	Série “RESTART”	Post “MANUAL_RESTART” puis reboot → ✅ OK
Résilience (watchdog)	Boucle bloquée	Tâche longue simulée	WDT/soft reset → ✅ OK
Observabilité	SHOW_LOGS bridge	Commande série	Évènements horodatés listés → ✅ OK
Observabilité	SHOW_STATS node	Commande série	KPIs retry/tx affichés → ✅ OK
Observabilité	CHECK_MEMORY	Bridge & Node	Heap + SPIFFS affichés → ✅ OK
Qualité (code)	Conformité messages	Formats clés (TEMP=,HUM=…)	Parsés côté API → ✅ OK
Qualité (traces)	Anonymisation	Pas de secret en logs	OK (clé non loggée) → ✅ OK
Déploiement	Changement d’API URL	BridgeDataUrl/NodeDataUrl	Envois toujours OK → ✅ OK
Compatibilité	Navigateurs	Chrome/Edge/Firefox	Comportement identique → ✅ OK
Accessibilité	Responsive tableau de bord	< 768 px	Layout lisible → ✅ OK
Robustesse (concurrence)	2 nodes simultanés	Trames concomitantes	Files RHMesh OK → ✅ OK
Robustesse (ordre)	Trames hors ordre	2 trames croisées	Logs ordonnés par TIME → ✅ OK
Sécurité (headers)	Content-Type strict	application/json	Accepté, autres rejetés → ✅ OK
Sécurité (CORS)	Appels cross-origin	Par défaut fermé	Pas d’exposition indue → ✅ OK
Données (unités)	°C / % / g	Nommage champs	Homogénéité côté API/DB → ✅ OK

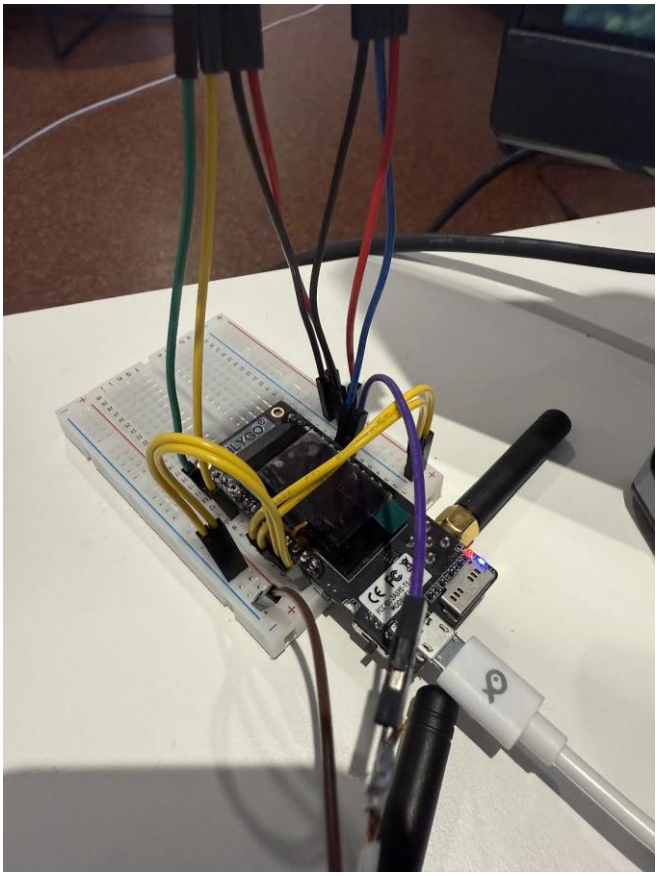
Voici des photos que j’ai pu prendre lors de l’installation du projet Lora chez notre partenaire Keolis :

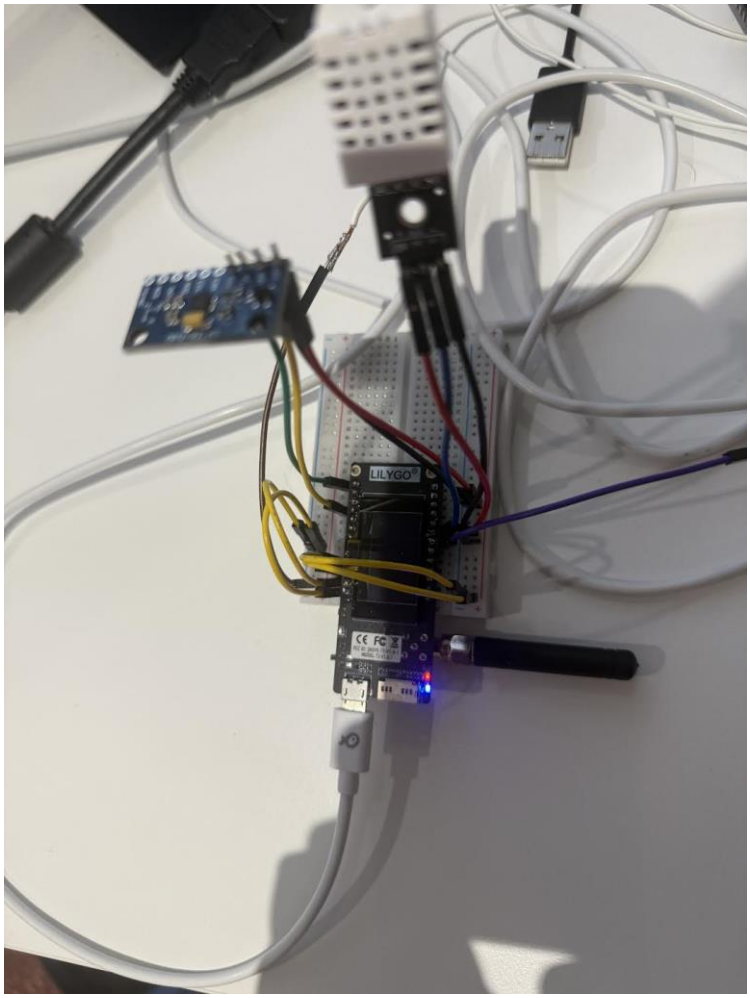



PC
+MODULE
LORA
IP55
legrand


PC
+MODULE
LORA


FOR ESP32
TERMINAL ADAPTER
FLUKE 175 TRUE RMS MULTIMETER
237.5
V AC
Auto Range
AutoHOLD
HOLD
MINMAX
RANGE
OFF
400 mA
10A
FUSED
COM




LILYGO