Documentation Officielle - UBBEE Smart Building (V1)

Bienvenue dans la documentation officielle du projet **UBBEE**, la plateforme SaaS B2B de Gestion Technique de Bâtiment (GTB) intelligente.

Ce répertoire contient tous les documents de référence pour comprendre, déployer et maintenir la plateforme.

Sommaire de la documentation

Généré pour la livraison de la Version 1 (V1) du produit.

1. Documentation Fonctionnelle UBBEE (V1)

Vision du Produit

UBBEE est une plateforme SaaS (Software as a Service) B2B permettant la supervision, le pilotage et l'analyse intelligente des bâtiments d'entreprise (Smart Building). Elle s'adresse aux *Facility Managers*, aux responsables d'exploitation et aux gestionnaires d'énergie pour optimiser la consommation, réduire l'empreinte carbone et assurer le confort des occupants grâce à des capteurs IoT et des règles domotiques.

Multi-Tenancy et Rôles

Le système est pensé pour gérer de multiples sociétés (Tenants). Il supporte deux grands contextes :

1. Contexte Opérateur (Super Admin / UBBEE)

- **Rôle :** `SUPER_ADMIN`
- **Périmètre :** A la vision de **tous les clients** de la plateforme.
- **Actions :** Peut créer/supprimer des clients, ajouter des sites, forcer des configurations réseau. C'est l'interface de "Gestion de Parc" globale.

2. Contexte Client (B2B)

- **Rôles :** `ENERGY_MANAGER` (Admin Client), `TECHNICIAN`, `CLIENT` (Lecteur/Invité).
- **Périmètre :** Restreint aux **sites et utilisateurs de sa propre organisation** (Ex: Decathlon ne voit que les magasins Decathlon).
- **Actions :** Pilotage des sites, lecture des consommations, paramétrage des règles de ses bâtiments.

Modules Fonctionnels (Fonctions Clés)

A. Tableau de Bord Global

- **Supervision temps réel :** KPIs dynamiques de santé des bâtiments, coût énergétique estimé, émissions CO2 évitées.

- **Répartition :** Graphiques de split consommation CVC (Chauffage, Ventilation, Clim) vs le reste. Modulable Temporellement (Jour/Mois).
- **Mises à jour asynchrones :** Les métriques fluctuent en direct via WebSockets reflétant la réalité des capteurs.

B. Gestion de Parc (Sites & Zones)

- **Typologie :** Organisation hiérarchique : Client -> Sites (Bâtiments géolocalisés) -> Zones (Pièces / Étages).
- **Carte Globale :** Interface cartographique mondiale listant les sites actifs avec une vue filtrable.

C. Jumeau Numérique (Digital Twin)

- **Vue 3D Dynamique :** Modélisation isométrique 3D du plancher d'un site.
- **Couches DataViz :** Possibilité de voir la donnée colorisée sur le plan 3D :
  - *Thermique* (Bleu/Vert/Rouge selon la T°).
  - *Qualité d'Air* (Orange si > 800ppm CO2).
  - *Occupation* (Mise en évidence des salles occupées).

D. Moteur de Règles & Alertes

- **Logique SI/ALORS (IFTTT) :** Interfaces visuelle (drag & drop) de conditionnement IoT (Ex: "Si T° > 25°C ALORS allumer Climatisation").
- **Tableau des Alertes :** Journalisation des erreurs de communication réseau ou des dépassements de seuils critiques.

E. Paramètres & Marque Blanche

- **SSO / Utilisateurs :** Gestion interne des collaborateurs du client.
- **Plages Horaires :** Définition des modes "Ouvert / Fermé" pour le pilotage automatique ÉCO.
- **Marque Blanche :** Personnalisation des couleurs de l'espace au logo de l'entreprise B2B (Mise en place de tokens UI).
- **Intégrations :** Webhooks sortants et lecture de Clés API pour intégration avec PowerBI ou un ERP tiers.

F. Appairage & Hub IoT (No-Code)

- **Provisionnement Rapide :** Auto-découverte et association (Plug & Play) d'équipements certifiés UBBEE sur les ondes via identifiant matériel.
- **Interopérabilité Avancée :** Interface visuelle (Glisser-Déposer) permettant aux intégrateurs de "mapper" n'importe quelle trame de données brutes d'un capteur ancien ou tiers vers les modèles de données standards UBBEE.

- **Positionnement Physique :** Placement du nouveau capteur dans l'arborescence (Client > Site > Zone) pour activation des jumeaux numériques.

2. Documentation Technique

Stack Technologique Globale

L'application est construite autour du framework Javascript **React** coté Frontend et **Node.js / Express** coté Backend.

Frontend

- **Framework :** Next.js 14+ (App Router) en React (TypeScript).
- **Style CSS :** Tailwind CSS avec plugin "Tailwind Animate". Prise en charge stricte du Dark/Light mode via la classe `dark:` et design ultra-moderne (*Glassmorphism*).
- **Librairies Graphiques (DataViz) :**
  - **Recharts :** Génération des courbes temporelles, des graphiques multi-axes (ComposedChart avec Line/Area).
  - **React-Leaflet :** Intégration de OpenStreetMap pour la cartographie des sites. Composants chargés dynamiquement (`next/dynamic`) pour éviter les conflits SSR (Server-Side Rendering).
  - **Three.js / React Three Fiber :** Rendu du Jumeau Numérique 3D hardware-accelerated. Utilisation de `@react-three/drei` pour les contrôles orbitaux de caméra et l'incrustation HTML de la domotique.
- **Temps Réel :** `socket.io-client` pour le maintien d'une connexion persistante avec le Backend permettant la mise à jour des jauges sans refresh.
- **Icônes :** Lucide React.
- **UI Builders :** DnD-Kit pour le Drag&Drop des widgets du dashboard. Customisation poussée avec `clsx` et `tailwind-merge` (`cn` utility).

Backend

- **Serveur :** Node.js avec Express.
- **ORM & Base de données :** TypeORM avec SQLite (Moteur local, prêt à être switché sur PostgreSQL en production par simple modification de la chaîne de connexion).
- **Authentification :** JSON Web Tokens (JWT) générés au login.

- **Temps Réel :** `socket.io` branché sur le serveur HTTP Express. Un worker émet des payloads aléatoires ou réels vers les clients abonnés pour simuler le trafic IoT.
- **Scripts Utilitaires :** Utilisation intensive de scripts CommonJS (`.cjs`) pour le seeding de base de données, la génération massive de fausses alertes, et les déploiements (SSH scripts).

Structure des Fichiers (Frontend)

```
src/
├── app/                    # Routeur Next.js
│   ├── page.tsx            # Tableau de bord Global
│   ├── clients/            # Annuaire et Dashboard clients B2B spécifiques
│   ├── map/                # Cartographie plein écran
│   ├── energy/             # Rapports de consommations
│   ├── rules/              # Moteur IFTTT domotique
│   └── settings/           # Configuration Marque Blanche et Utilisateurs
├── components/
│   ├── dashboard/          # Widgets et visualisations (Charts, Three.js Building
│   ├── layout/             # Sidebar, Header avec le 'Tenant Switcher'
│   └── ui/                 # Éléments réutilisables (Select, Inputs, etc.)
├── lib/
│   ├── TenantContext.tsx # Cerveau de la Multi-tenancy (Rôle + Organisation cou
│   └── utils.ts            # Helpers Tailwind (cn)
└── globals.css             # Variables CSS, Classes personnalisées (Glassmorphism
```

Gestion de l'Etat de Session (TenantContext)

L'authentification et le multi-tenant repose fortement sur `TenantContext` . A chaque appel API (`authFetch`), le frontend injecte automatiquement l'identifiant de l'organisation courante et le jeton sécurisé. Si l'utilisateur est un `SUPER_ADMIN` , il peut manipuler la variable `currentTenant` pour se faire passer temporairement pour une autre entreprise et visionner la plateforme telle que le client la voit.

3. Architecture Logicielle & Modèle Conceptuel de Données

Modèle de Données (MCD - ORM)

L'architecture de base repose sur un schéma entité-relation rigide permettant la multi-tenancy :

1. **Organization (Client)** : Entité Racine (ex: "UBBEE" ou "Decathlon").
2. **User (Utilisateurs)** : Les employés d'une Organisation (Rôle, Email, Mot de Passe haché).
3. **Site (Bâtiment)** : Les infrastructures physiques liées à une Organisation (ex: "Station-Service Genève").
4. **Zone (Pièce / Étages)** : Les différents espaces cartographiés d'un site (utilisés par le Jumeau Numérique pour positionner les capteurs).
5. **Rule (Règles Domotiques IFTTT)** : Liées à une organisation pour conditionnaliser les événements.

Architecture Matérielle / Flux de Données (IoT)

1. Ingestion de Données (Acquisition)

Les *gateways* ou concentrateurs IoT situés dans les bâtiments des clients émettent des données agrégées (Température, Consommation CVC, CO2).

- **Protocole Nominal:** MQTT ou HTTP POST vers le Backend UBBEE (Node.js/Express `api/energy` etc.).
- **Worker de Simulation:** Actuellement, un service de simulation interne (voir `/backend/index.js` `simulateData()`) crache aléatoirement de la donnée sur le Socket IO.

2. Traitement (Processing & Rule Engine)

Le Backend reçoit la trame IoT:

1. Enregistrement en base de données SQL (TSDB native / Historisation).
2. Évaluation face au **Moteur de Règles** (`Rules`). Si une condition est remplie (Ex: *Consommation* > *X kWh*), une **Alerte** est générée en base.
3. Le Backend "pousse" instantanément la nouvelle donnée vers le réseau de serveurs Frontend connectés via WebSocket (`socket.emit('refresh_data')`).

3. Restitution (Dashboard Client)

1. Le Frontend abonné au WebSockets reçoit les signaux de rafraîchissement au fil de l'eau.
2. Les graphes React (Recharts / Three.js) se mettent à jour réactivement sans recharger le DOM.
3. Les Notifications In-App remontent visuellement l'alerte traitée par le moteur.

```
graph TD
    A[Capteurs IoT / Ubots] -->|MQTT / API REST| B(Backend UBBEE - Node.js)
```

```
    B -->|TypeORM / Insert| C[(Base TypeDB SQLite/Postgres)]
    B -->|Check Sécurité / IFTTT| D{Moteur de Règles}
    D -->|Si Dépassement| E[Génération Alertes / Webhooks Externes]
    B -->|WebSocket Pub/Sub| F[Frontend Next.js React]
    F <-->|Demande Info Client (JWT)| B
    F --> G[Render Dashboard HTML/3D Jumeau]
```

Concept de Jumeau Numérique B2B (DataViz 3D)

Afin de ne pas surcharger la page avec 50 sites simultanés :

- Chaque maquette 3D est rattaché à Un Seul Bâtiment (`Site`).
- Les zones sont modélisées dynamiquement (Box3D) selon la table `Zone` de la base SQL (les tailles `size` et `position` des zones SQL définissent où elles se trouvent en 3D).
- Le Frontend applique un "matériau" de couleur transparente par-dessus (Layer) selon ce qu'il écoute (Thermique, CO2, Occupation) transformant de la Data Textuelle en une carte de chaleur immersive (Heatmap spatiale).

4. Dossier d'Exploitation (RUN / MCO)

Ce dossier décrit le Maintien en Condition Opérationnelle (MCO) et la gestion du cycle de vie des applicatifs du Front et du Back sur la machine Linux Cible.

Écosystème & Hébergement

- **Serveur Cible:** `76.13.59.115` (Droplet DigitalOcean / VPS Ubuntu).
- **Compte Utilisateur:** `root` (Pour la V1, déploiements effectués via clés asymétriques SSH).
- **Dossier Racine:** `/opt/gravity-lab/smart-building`
- **Process Manager:** PM2 (Gestionnaire de daemon pour applications Node.js en continu).

Commandes Fréquentes d'Exploitation

Backend (API & WebSockets Express)

Le Backend gère les API et héberge la base de donnée SQLite locale.

- **Redémarrer le Backend :**

```
pm2 restart gtb-backend
```

- **Lire les Logs Backend (Erreurs / Pushs WebSocket) :**

```
pm2 logs gtb-backend --lines 200
```

Frontend (Next.js Application B2B)

L'application Front tourne soit en mode Serveur (Next.js Node) soit sur des exports statiques.

- **Redémarrer le Frontend :**

```
pm2 restart gtb-frontend
```

- **Lire les Logs Frontend :**

```
pm2 logs gtb-frontend --lines 200
```

Scripts Automatisés de Déploiement (Pipeline SSH)

Le projet maintient une suite de scripts CommonJS (`.cjs`) agissant comme des runners simplifiés (Simili-Ansible) qui s'exécutent en local pour ordonner la compilation distante.

- **`ssh_build_front.cjs`** : Outil de déploiement à chaud du Frontend.
  i. Se connecte en SSH au VPS.
  ii. Procède à un `git pull` sur le repository.
  iii. Réinstalle les dépendances Frontend (`npm i`).
  iv. Compile le Frontend (`npm run build`).
  v. Relance PM2 silencieusement et remonte les métriques de succès.

- **`fetch_alerts.cjs / ssh_restore_users.cjs`** : Scripts utilitaires d'injection de logs métiers/erreurs dans la base de données de production pour des besoins de démonstration commerciale sans attendre l'IoT réel.

Surveillance de la machine (Serveur)

Utilisation générique des outils de performance Ubuntu.

- `htop` : Surconsommation éventuelle de la RAM par PM2 ou Next.js Build Processes.
- `pm2 monit` : Tableau de bord visuel Node.js affichant le CPU/Memory des Workers.

Procédures d'Urgence

1. **Perte de mot de passe "Super Admin" UBBEE :** Exécuter les scripts de "Seed" depuis le serveur distant. La base de données SQLite écrasera et reconfigura le plan de compte à son état de distribution.
2. **Crash Silencieux du WebSockets Frontend :** Si les données temps réel ne bougent plus sur la plateforme sans indiquer d'erreur, c'est que le worker PM2 Backend du Socket a été déconnecté (Timeout). Exécuter un `pm2 restart gtb-backend` force la reconnexion automatique des clients Front.

5. Hébergement, Réseau & Infrastructure

Ce document détaille l'architecture matérielle (IaaS), la configuration réseau attendue et les choix liés à l'hébergement de la plateforme Smart Building sur notre environnement cible.

1. Infrastructure Cloud (IaaS)

Pour assurer une flexibilité maximale et garder la maîtrise complète de l'application et de la base de données, l'application est hébergée sur un Serveur Privé Virtuel (VPS).

- **Fournisseur Cloud :** DigitalOcean
- **Type d'instance :** Droplet
- **Système d'Exploitation :** Ubuntu Linux (LTS)
- **IP Publique Cible :** `76.13.59.115`
- **Répertoire de déploiement :** `/opt/gravity-lab/smart-building/`

2. Topologie Réseau & Reverse Proxy (NGINX)

Dans une configuration de production aboutie, l'accès direct aux ports applicatifs Node.js est prohibé depuis l'extérieur. Un serveur web frontral (**NGINX**) est utilisé comme *Reverse Proxy*.

Routage NGINX

- **Trafic App Web (Frontend) :**
  - L'application Next.js écoute sur le port `3000` (`localhost:3000`).

    - NGINX capture tout le trafic entrant public sur HTTP/HTTPS (racine `/`) et le transfère au port 3000.
- **Trafic API & WebSockets (Backend) :**
    - L'API Express ainsi que le serveur d'événements Socket.IO écoutent sur le port `3001` (`localhost:3001`).
    - NGINX capture les routes spécifiques (ex: `/api/` ou `/socket.io/`) et transfère ce trafic. Il doit être paramétré pour accepter l'en-tête `Connection: Upgrade` vitale pour le fonctionnement persistant des WebSockets.

Alerte de Sécurité V1 : Tant que le nom de domaine B2B officiel n'est pas déployé et chiffré par un certificat SSL (Certbot / Let's Encrypt), l'application répond directement via l'IP publique sur ses ports ouverts.

3. Gestionnaire de Daemons (PM2)

La robustesse du cluster Node.js est assurée par l'utilitaire **PM2**, installé de manière globale sur la machine.

Il garantit :

- Le redémarrage automatique en cas de crash applicatif.
- L'équilibrage de charge si l'on décide un jour de le passer en mode *Cluster* multi-coeurs.
- Le redémarrage au boot (*script pm2 startup*) si la machine physique DigitalOcean venait à redémarrer.

Les processus enregistrés sont :

- `gtb-frontend` (Le Next.js App Router).
- `gtb-backend` (L'API Express, Moteur de Règles et Socket.IO).

4. Politique de Sécurité Serveur

La sécurité de l'hébergement est construite en couches :

1. **Authentification SSH :** Fermée aux connexions par mots de passe (`PasswordAuthentication no`). Sécurisée par un jeu de clés asymétriques RSA/Ed25519 gérées sur les postes développeurs. User : `root`.
2. **Pare-feu (UFW) :** Recommandé d'activer l'UFW Ubuntu pour ne laisser ouvert que :
    - `Port 22 (SSH)` : Idéalement sur-restreint aux IP fixes de l'agence.
    - `Port 80 / 443 (HTTP/S)` : Pour les clients finaux et l'IoT qui remonte ses flux POST de télémétrie.
3. Les ports `3000`, `3001` seront verrouillés en interne (loopback `127.0.0.1`).

5. Stockage des Données (Database)

Dans ce premier jalon logiciel :

- **SQLite :** Le Backend TypeORM génère et gère intimement la base entière sous la forme d'un simple fichier binaire (`.sqlite`).
- **Avantage en Hébergement :** Sauvegarder la plateforme est aussi trivial que de copier ce fichier (par exemple via une cron-tape journalière qui exécute un `rsync` ou un `scp` vers un serveur de stockage S3/froid).
- **Prochaine étape (Scale-up) :** Lorsque la plateforme embarquera des dizaines de milliers de points de données journaliers IoT, ce format limite la concurrence/verrouillage d'écriture. L'environnement sera basculé sur le service managé *PostgreSQL de DigitalOcean*, où la variable d'environement `DATABASE_URL` du démon PM2 du Backend pointera vers l'URL managée.

6. Procédure d'Appairage IoT (Guide Energy Manager)

L'appairage d'équipements sur la plateforme UBBEE se réalise via le module **Appairage & Hub IoT** (Interopérabilité No-Code). Ce module offre deux approches distinctes selon le niveau technique du matériel à intégrer.

Pré-requis communs

Avant toute manipulation sur la plateforme, l'Energy Manager doit s'assurer de :

1. **Couverture réseau** : Une Gateway UBBEE (ou LoraWan/Zigbee tierce) doit être active et couvrir physiquement la zone d'installation prévue du capteur.
2. **Alimentation** : Le capteur doit être sous tension (sur pile ou raccordé au secteur) et en mode appairage (généralement indiqué par une LED clignotante selon la notice du fabricant).
3. **Hiérarchie Spatiale** : Le Client (Organisation), le Bâtiment (Site) et la Zone (Espace) cible doivent préalablement exister dans l'annuaire UBBEE.

Méthode 1 : Provisionnement Rapide (Capteurs Standardisés / Plug & Play)

Cas d'usage : Déploiement de capteurs modernes, certifiés UBBEE ou reconnus nativement par la plateforme.

Cette méthode est conçue pour être extrêmement rapide, idéale pour les déploiements massifs.

1. **Accès :** Se rendre sur l'onglet `Nouveaux Équipements (Parc Simple)` du module d'appairage.
2. **Déclaration du Matériel :**
   - **Type / Modèle :** Sélectionner la catégorie de l'équipement dans le menu déroulant (ex: Sonde Multimédia UBBEE, Détecteur de Présence PIR, Compteur d'Énergie Modbus/IP).
   - **Identifiant Réseau :** Saisir ou scanner l'identifiant unique de l'équipement (Adresse MAC, DevEUI pour le LoRaWAN, etc.). Ce code est généralement inscrit sur une étiquette située sur, ou à l'intérieur du capteur.
3. **Positionnement Physique :**
   - Utiliser la section "Positionnement du Capteur" en bas de page.
   - Sélectionner successivement le **Client (Organisation)** ciblé, puis le **Bâtiment (Site)**, et enfin préciser la **Zone (Espace)** exacte où le capteur est physiquement installé (au mur, au plafond, etc.).
4. **Validation :**
   - Cliquer sur le bouton "Lancer le Provisionning".
   - La plateforme initie alors une procédure d'auto-découverte sur le réseau (Handshake) via la Gateway. Une fois le capteur détecté, il est immédiatement associé à la zone configurée et remontera ses premières valeurs sur les tableaux de bord.

Méthode 2 : Interopérabilité Avancée (Capteurs Existants / Custom MQTT)

Cas d'usage : Récupération d'un parc de capteurs existants, intégration de matériels non standards, ou gestion de flux de données JSON spécifiques (ex: via Zigbee2MQTT).

Cette méthode permet à un technicien ou un intégrateur d'adapter n'importe quel payload de données aux formats standardisés d'UBBEE, sans écrire une seule ligne de code.

1. **Accès :** Se rendre sur l'onglet `Équipements Existants (Interop. Avancée)` du module d'appairage.
2. **Positionnement Physique :**
   - Même procédure que la méthode 1 : Définir le positionnement du capteur (Client > Site > Zone).
3. **Configuration du Flux Source :**
   - **Nom du Flux :** Donner un nom explicite à cette configuration (ex: "Anciennes sondes Chaufferie").

- **Topic MQTT :** Renseigner le topic MQTT exact sur lequel le capteur externe publie ses données brutes (ex: `zigbee2mqtt/0x00158d00045ab`). La plateforme se met à l'écoute de ce topic.

4. **Le Mapping Visuel (Drag & Drop) :**
   - **Colonne Source (Gauche) :** Une fois le Topic connecté, la plateforme intercepte un message test ("LIVE Payload") et extrait automatiquement toutes les clés (propriétés) du fichier JSON envoyé par le capteur (ex: `temperature`, `humidity`, `battery`, `linkquality`). Ces clés apparaissent sous forme de blocs déplaçables.
   - **Colonne Cible (Droite) :** Cette colonne affiche la liste des "Champs Standards UBBEE" attendus par notre base de données (ex: `Température (°C)`, `Humidité (%)`, `Batterie Équipement (%)`).
   - **Action :** Avec la souris, l'Energy Manager doit "glisser-déposer" chaque bloc de la colonne Source vers son équivalent dans la colonne Cible. (Ex: Glisser la clé `{ temperature }` vers le champ "Température (°C)").
5. **Validation :**
   - Un bloc correctement lié affiche un statut vert "Connecté : [nom_de_la_cle]".
   - Enregistrer la configuration. Désormais, chaque donnée brute arrivant sur ce Topic sera traduite à la volée dans le format universel UBBEE et liée à la Zone définie.

Documentation du Copilote IA UBBEE

Le Copilote IA UBBEE est un assistant conversationnel intelligent intégré à la plateforme GTB (Gestion Technique du Bâtiment). Il permet aux utilisateurs d'interagir avec les données de leur parc immobilier et de contrôler des équipements en langage naturel.

1. Fonctionnalités Principales

- **Contexte Dynamique :** L'Agent connaît en temps réel l'utilisateur connecté, son rôle (Client, Technicien, Energy Manager) et son tenant (`tenantId`).
- **Compréhension du Langage Naturel :** Traduit une phrase humaine (ex: "Coupe l'éclairage de Casa Rivoli") en une action technique structurée sur le bon bâtiment.
- **Récupération de Données :** Capacité à récupérer l'historique et l'état des capteurs/compteurs.
- **Moteur de Recherche Intégrée :** Grâce à des algorithmes de filtrage en base de données, l'IA scrute d'abord les équipements associés au domaine du client avant de proposer une action.

2. Architecture Technique

- **Frontend (Widget Client) :**
  - Composant React `CopilotWidget.tsx` intégré dans le layout général.
  - Fenêtre de chat flottante (Glassmorphism UI).
  - Gestion des requêtes utilisateur, état de chargement et persistance temporaire des messages.
  - **Human-in-the-Loop (HITL) :** Affichage d'une carte d'alerte pour les actions de contrôle (Modification de consigne, allumage/extinction). L'IA ne lance **jamais** de commande électrique en autonomie totale.
  - Communication asynchrone avec l'API Backend `/api/copilot/chat` .
- **Backend (Service & Orchestration) :**
  - `CopilotController` & `CopilotService` (NestJS).
  - Sécurité des Endpoints (Bloqué par `JwtAuthGuard`).
  - Filtrage RBAC (Role-Based Access Control) : Un utilisateur "Viewer" ne sera pas autorisé à muter l'état d'un équipement, le backend rejettera l'exécution.
  - Interaction avec **DeepSeek API** (LLM externe, remplaçable par Llama ou Qwen local).

3. Mécanisme de Function Calling (Outils LLM)

Le Copilote n'hallucine pas l'état du bâtiment, il exécute des requêtes de découverte (Tools) via le backend :

1. **`list_my_available_devices`** : Permet au LLM de chercher l'ID technique d'un équipement en fonction d'un champ sémantique saisi par l'humain (`searchText` = Nom, Pièce, Zone, Site).
2. **`get_sensor_history`** : Permet au modèle d'analyser l'historique récent d'une série chronologique de températures ou consommations énergétiques.
3. **`set_device_state`** : Prépare la structure d'une commande (ON/OFF/Consigne). L'agent a l'instruction **de ne pas le demander en texte**, mais de simplement appeler cette fonction de manière "silencieuse" pour déclencher la Pop-up UI côté client (HITL).

4. Sécurité & Bonnes Pratiques

- **Aucun accès BDD en direct** : Le LLM n'écrit aucune requête SQL. Il ne peut utiliser que les 3 fonctions strictement définies et contrôlées décrites ci-dessus.
- **Human-in-the-Loop garanti :** Même si le LLM est trompé par un prompt malveillant, toute exécution de `set_device_state` est suspendue par l'orchestrateur Backend (Status : `pending_human_confirmation`) ce qui délègue la validation à un bouton clické par l'humain.

- **Variable d'Environnement :** La `DEEPSEEK_API_KEY` est injectée silencieusement au niveau OS et non dans le code source pour éviter toute fuite sur GitHub.

5. Déploiement

Le démarrage du service nécessite l'injection stricte de la variable d'environnement au lancement du processus Node/PM2.

```bash

# Injection propre de la variable d'environnement (Exemple VPS)

pm2 start ecosystem.config.cjs --only gtb-backend --env production --update-env ```

# 7. Sécurité & Conformité de la Plateforme (RGPD)

La sécurité, la confidentialité des données et la conformité légale sont des piliers de l'architecture SaaS d'UBBEE. Ce document liste les mesures techniques et organisationnelles mises en place pour assurer la protection du système et des données (Security by Design).

## 🔒 1. Mesures de Sécurité Applicative

### Authentification & Accès (IAM)

- **Authentification forte :** Implémentation de jetons **JWT (JSON Web Tokens)** pour les sessions utilisateur, signés cryptographiquement. Aucune session de longue durée n'est stockée en clair côté client.
- **Mots de passe protégés :** Les mots de passe utilisateurs sont systématiquement hachés et salés (*salted hash* via bcrypt) avant insertion en base de données. Il est mathématiquement impossible de les retrouver en clair en cas de fuite de la base.
- **Rôle et Isolation (Multi-Tenancy) :** La plateforme intègre un strict RBAC (*Role-Based Access Control*). Un Energy Manager de l'Organisation 'A' ne peut techniquement pas requêter les
```

données, l'annuaire ou les sites de l'Organisation 'B' via l'API, car le backend filtre systématiquement l'accès par le `Tenant ID` sécurisé du JWT injecté.

Tolérance aux Vulnérabilités Web

- **Injections SQL (CWE-89) :** L'utilisation stricte de l'ORM (TypeORM) empêche l'injection SQL classique, l'ORM échappant lui-même les paramètres de requête.
- **Failles XSS (CWE-79) :** Le frontend développé en React échappe nativement le contenu textuel rendu dans le DOM.
- **Contrôle du trafic API :** Les requêtes entrantes sont soumises à la vérification globale des Headers d'authentification avant de rentrer dans la couche métier.

📡 2. Sécurité Réseaux et Infrastructures

- **Chiffrement des Flux (En Transit) :** Dans l'environnement de production standardisé, toutes les connexions entre le navigateur client et les serveurs UBBEE (Frontend HTML, API REST, WebSockets temps réel) se font exclusivement via le protocole chiffré **HTTPS (TLS 1.2/1.3)**.
- **Chiffrement IoT :** Les concentrateurs et capteurs transmettent leurs trames vers la plateforme en utilisant les couches sécurisées (ex: MQTT over TLS, Webhooks M2M authentifiés).
- **Surface d'attaque minimale :** Le serveur d'hébergement s'appuie sur une politique de "Zero Trust" (voir `05_hebergement_et_reseau.md`). Seuls les ports 80/443 (HTTP/s) et 22 (SSH sur clé asymétrique uniquement) sont accessibles depuis l'extérieur. Le pare-feu système (UFW) bloque toute autre tentative.

⚖️ 3. Conformité RGPD & Données à caractère personnel

La plateforme **UBBEE** a été pensée pour minimiser l'impact sur la vie privée : sa fonction principale est de mesurer l'activité technique des bâtiments, et non de surveiller des individus.

La minimisation des données (Data Minimization)

- Aucun traceur ni cookie de ciblage publicitaire n'est injecté. Seul un identifiant de session technique (Strictement nécessaire) est exploité.
- Les seules données à Caractère Personnel (PII) conservées en base concernent les administrateurs et collaborateurs enregistrés de la plateforme (Prénom, Nom, Courriel professionnel, Rôle, Mot de passe haché).
- **Aucune donnée personnelle de grand public ou de "visiteurs" du bâtiment n'est collectée.**

L'anonymisation des données de comptage (IoT)

- Les caméras thermiques, les capteurs de CO2 ou les détecteurs infrarouges (PIR) remonteront via UBBEE des **états binaires ou agglomérés** (ex: *Présence active dans la salle A* ou *Pourcentage d'occupation*).
- UBBEE refuse par définition technique la captation et l'ingestion d'images, de données biométriques ou d'enregistrements vocaux dans son modèle spatial interne.

Exercice des droits

- En accord avec le Chapitre 3 du RGPD européen, tout utilisateur, via son compte, peut :
  - Procéder à la modification de ses données.
  - Demander la désactivation ou la suppression définitive de son compte et des traces nominatives qui y sont associées ("Droit à l'oubli").

🚨 4. Maintien en Condition de Sécurité (Sécurité du Copilote IA)

L'intégration de modèles d'Intelligence Artificielle de type LLM (Large Language Models) dans la GTB fait l'objet d'un "sandboxing" (voir document `06_copilot_ia.md`) :

- **Human-in-the-Middle (HIM) :** L'IA ne possède aucun accès direct aux relais de puissance ou aux déclencheurs CVC de vos bâtiments. Chaque action demandée par le Copilote déclenche un composant visuel de confirmation (Carte d'autorisation), exigeant une action physique et intentionnelle d'un opérateur humain (Energy Manager) authentifié.
- Les requêtes générées vers les serveurs LLM extérieurs ne transmettent jamais l'identité nominative du requérant ni d'informations confidentielles du client, mais uniquement le contexte technique immédiat de la salle (Température, Nom matériel, ID Anonymisé).