Dossier technique de référence

Architecture, stack logicielle, protocoles et intégrations — Frutera · FruteraScan · LR Connect · Éclairage

Ce document est la référence technique consolidée de l'écosystème LR Systems. Il regroupe l'ensemble des choix d'architecture, des stacks logicielles, des protocoles de communication, des conventions de code et des intégrations matérielles déployées sur les trois projets actifs : Frutera (automate de serre), FruteraScan V4 (logiciel métier d'exploitation) et LR Connect (plateforme SaaS LoRaWAN multi-clients). À usage interne — base de travail et de traçabilité pour les évolutions et la maintenance.

SOMMAIRE

01 Architecture d'ensemble LR Systems p. 2
02 Frutera — Stack et boucle de régulation p. 3
03 Frutera — 8 moteurs et 21 routes API p. 4
04 Frutera — Drivers I/O et chantiers p. 5
05 FruteraScan V4 — Stack et workflow p. 6
06 FruteraScan V4 — Base MySQL ~79 tables p. 7
07 FruteraScan V4 — 11 collecteurs et intégrations p. 8
08 FruteraScan V4 — Refonte pilotage 5 phases p. 9
09 ESP32 — Firmware, capteurs et OTA p. 10
10 LR Connect — Stack et multi-tenant p. 11
11 Protocoles & intégrations transverses p. 12
12 Infrastructure, déploiement & conventions p. 13

REPÈRES CLÉS

Frutera

- Python 3.10+ / FastAPI / SQLAlchemy 2.0 async / SQLite
- React 19 / TypeScript 5.9 / Vite / Recharts
- Boucle régulation native ~2 s, WebSocket
- 180 tests pytest, 11 fichiers
- ~167 endpoints API préfixe /api

FruteraScan V4

- Node.js / Express 4 / mysql2 pool 50 conn
- JavaScript pur (pas de TypeScript)
- MySQL ~79 tables, charset utf8mb4
- 11 collecteurs externes (Jeedom, IPX800…)
- Firmware ESP32 PlatformIO / C++

LR Connect

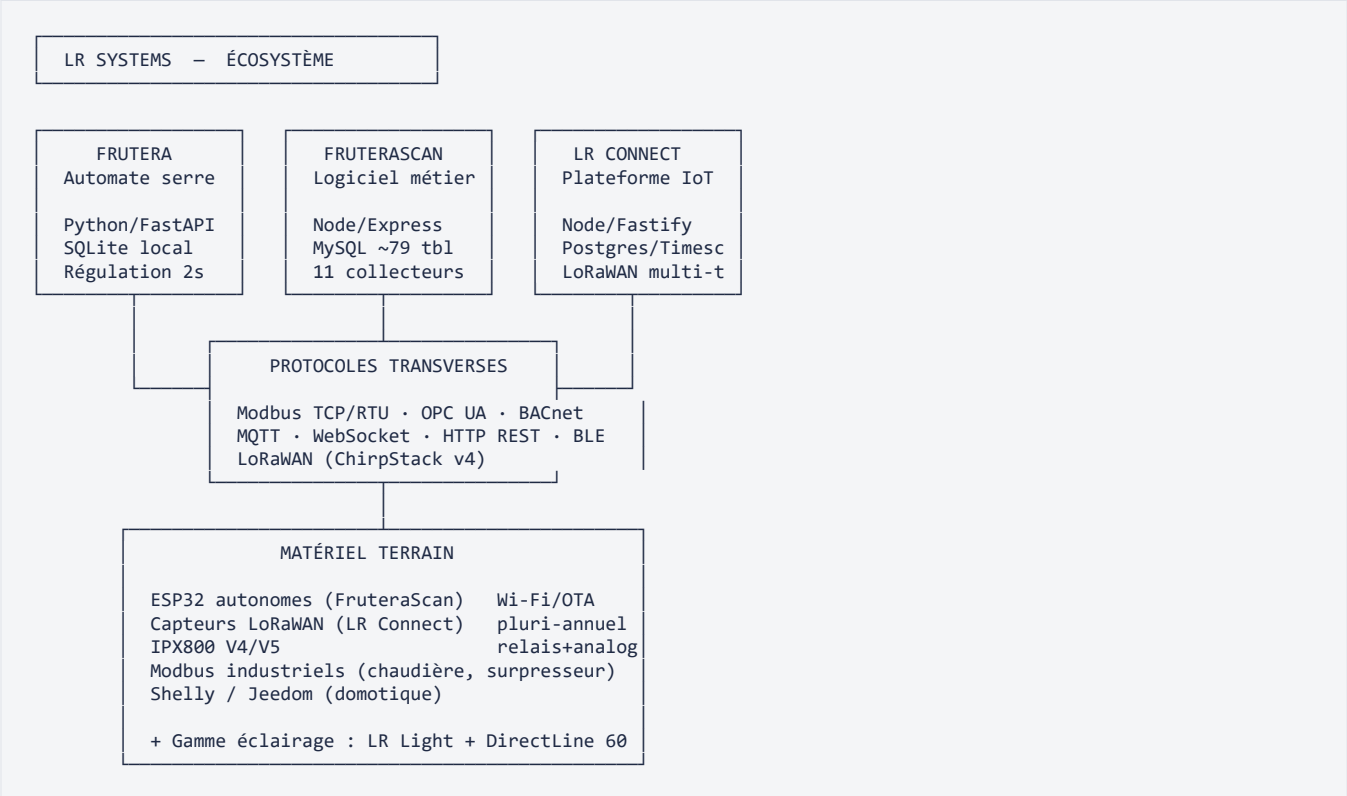
- Node 22 LTS / Fastify 5 / TypeScript strict
- Prisma 6 / Postgres 16 + TimescaleDB
- Redis 7 / BullMQ / WebSocket pub/sub
- React 19 / Vite 6 / Tailwind 3
- LoRaWAN via ChirpStack v4 (Phase B)

Philosophie d'architecture commune aux 3 projets

Décisions explicables (journal d'action lisible) · Sécurités en cascade (hiérarchie de protections par module) · Modèle prédictif (anticipation par météo et rayonnement) · Économies automatiques (intégration 24 h) · Interface 100 % française (code anglais, UI française) · Multi-tenant strict (isolation par `customerId`) · WebSocket pour temps réel (jamais de polling DB) · Standards ouverts (Modbus, MQTT, ChirpStack, IPX800).

01 Architecture d'ensemble LR Systems

L'écosystème LR Systems s'organise autour de trois logiciels métier et d'une gamme matérielle d'éclairage. Chaque logiciel adresse un périmètre fonctionnel distinct, et tous peuvent coexister sur une même exploitation. Les flux de données sont conçus pour circuler dans les deux sens : un capteur LR Connect peut alimenter FruteraScan en mesures pour la traçabilité, tout en influençant les régulations de Frutera en temps réel.



PÉRIMÈTRES FONCTIONNELS PAR PRODUIT

PRODUIT	CIBLE	FONCTION PRINCIPALE	TEMPS RÉEL	MULTI-TENANT
Frutera	1 exploitation	Régulation climat / eau / lumière / ferti	WebSocket 2 s	Non (mono-tenant)
FruteraScan V4	1 exploitation	ERP métier, traçabilité, capteurs, pilotage	WebSocket événementiel	Non (mono-tenant)
LR Connect	N exploitations	Plateforme SaaS LoRaWAN, supervision	WebSocket pub/sub	Oui (isolation stricte)
LR Light / DirectLine	Matériel passif	Éclairage photopériodique fraisier	—	—

INFRASTRUCTURE D'HÉBERGEMENT ET D'ACCÈS DISTANT

Existant (Frutera production)

› Cloudflare — zone `lrsystems.org`, DNS + tunnels
› Tunnel `admin.lrsystems.org` → PC dev :9000
› Tunnel `<client>.lrsystems.org` → PC client :8000
› Tailscale — tailnet `lrat002@`, IP `100.123.213.16`
› RDP via Tailscale, compte Windows `Administrator`
› Service Windows : NSSM pour auto-boot

Cible LR Connect

› Phase 0 : ChirpStack + plateforme sur PC Debian chez Ludo
› Phase 1 : VPS Hetzner Helsinki (Debian)
› Sous-domaines prévus : `connect.lrsystems.org`, `scan.lrsystems.org`
› Docker Compose source unique en dev + prod
› WSL Ubuntu côté dev Windows, Docker natif côté Debian
› Auteur Git unifié : `lrat002@gmail.com`

02 Frutera — Stack et boucle de régulation

Backend

Langage	**Python 3.10+**
Framework web	**FastAPI 0.115**
ORM	**SQLAlchemy 2.0 async**
Base de données	**SQLite** (`data/frutera.db`)
Migrations	**Auto-création tables, pas de manuel**
Validation	**Pydantic Settings** (`.env`)
Serveur ASGI	**uvicorn (auto-reload dev)**
Port backend	**8000**
Tests	**pytest — 180 tests / 11 fichiers**

Frontend

Framework	**React 19**
Langage	**TypeScript 5.9**
Bundler	**Vite 8 (HMR)**
Charts	**Recharts 3.8**
Icônes	**lucide-react**
Port frontend	**3000**
Proxy	`/api` → **:8000**
Langue UI	**100 % français**
Langue code	**Anglais (variables, fonctions)**

ARCHITECTURE BACKEND

`main.py` orchestre le démarrage via lifespan : `init_db` → `load_drivers` → `start_realtime_loop`. `config.py` charge les paramètres via Pydantic Settings depuis `.env`. L'application expose ~167 endpoints API préfixe `/api` répartis sur 21 modules de routes.

```
C:/Frutera/
├── main.py                ← lifespan, app FastAPI, WebSocket /ws
├── config.py              ← Pydantic Settings, .env
├── data/frutera.db        ← SQLite, auto-créé au démarrage
├── models/                ← 12 modèles SQLAlchemy 2.0 async
├── routes/                ← 21 modules ~167 endpoints /api/*
├── engine/                ← 8 moteurs régulation + realtime_loop
├── drivers/               ← BaseDriver + IPX800, Modbus, Shelly, MQTT, Simulation
├── frontend/
│   ├── App.tsx            ← Providers Theme/Navigation/Chart/Compartment
│   ├── pages/             ← 16 pages
│   ├── components/        ← PeriodGraph, Sidebar, Synoptiques
│   ├── services/          ← API client
│   └── hooks/             ← hooks custom
└── tests/                 ← 180 tests pytest / 11 fichiers
```

BOUCLE TEMPS RÉEL

Principe

La `realtime_loop` orchestre l'ensemble des 8 moteurs de régulation avec une cadence de ~2 secondes. À chaque tick :

- Lecture de l'état physique via les drivers
- Évaluation des consignes (périodes, influences)
- Calcul des commandes par chaque moteur
- Envoi physique aux équipements (relais, modulants)
- Broadcast WebSocket aux clients connectés
- Écriture des historiques (asynchrone, non bloquant)

Règle stricte de dev

Les relais doivent claquer physiquement quand on commande. Tout développement se valide avec l'IPX800 réel branché, jamais uniquement en simulation. La couche driver est abstraite via `BaseDriver` pour pouvoir basculer Modbus / IPX800 / MQTT sans changer la logique métier.

WebSocket natif

Pas de Socket.IO ni de polling — WebSocket FastAPI direct avec broadcast manuel, reconnect côté client.

03 Frutera — 8 moteurs de régulation

FICHIER	MODULE	RÔLE	LIGNES
climate_regulation.py	Climat	Orchestration chauffage / ventilation, gestion des 5 périodes, intégration 24 h	—
vent_regulation.py	Ouvrants	Position côté abri / au vent, influences (radiation, HR, écart T°, Δp), sécurités météo en cascade	—
heating_regulation.py	Chauffage	T° eau, vanne 3 voies, pompe, courbe de chauffe, protection gel, couplage écran thermique	—
irrigation_regulation.py	Irrigation	EC/pH avec régulation PID, 14+ types de démarrage, phases, vannes, substrat	998
co2_light_regulation.py	CO_2 + Éclairage	CO_2 (4 sources), éclairage (5 modes), cible DLI, anti-scintillement, tolérance nuages	766
fog_regulation.py	Brumisation	Groupes pompes HP, VPD, secteurs, pression, comparaison cmd/état	—
screen_regulation.py	Écrans	Énergie, ombrage, occultation, fente anti-condensation, paliers lents	—
boiler_regulation.py	Chaudière	Brûleur modulant, cascade, courbe de chauffe extérieure	—
realtime_loop.py	Orchestrateur	Boucle ~2 s sur tous moteurs, broadcast WebSocket, ticks parallélisés	—
irrigation_loop.py	Orch. irrigation	Multi-station, file d’attente, séquencement des cycles	—

MODÈLES SQLALCHEMY

12 modèles principaux couvrent les entités métier : Compartment, Sensor, Actuator, IrrigationStation, IrrigationGroup, Tank, Valve, Period, Setpoint, Alarm, EventLog, Config. Tous les modèles utilisent l’API asynchrone de SQLAlchemy 2.0 avec sessions async. L’écriture historique passe par une queue non bloquante pour ne pas pénaliser la boucle 2 s.

FRONTEND — PROVIDERS ET PAGES

Architecture React

App.tsx instancie 4 providers globaux dans cet ordre : ThemeProvider, NavigationProvider, ChartProvider, CompartmentProvider. Le ChartProvider mutualise la logique de visualisation Recharts entre toutes les pages qui affichent des séries temporelles. Le CompartmentProvider injecte le contexte du compartiment sélectionné dans tout l’arbre.

Composants clés

› PeriodGraph — ligne continue avec segments plats par période et rampes en dégradé
› Sidebar — navigation par module, sélecteur de compartiment
› Synoptiques — circuits hydrauliques SVG animés
› JaugeArc — affichage temps réel des consignes
› Comparateur — vue multi-compartiments côte à côte

Règle d’or de visualisation Frutera

PeriodGraph : une seule ligne continue, des segments plats colorés pour chaque période, des rampes en dégradé entre périodes, jamais de chevauchement. Cette règle est appliquée systématiquement sur toutes les visualisations de consignes temporelles (T°, HR, CO_2, EC, pH, ouverture).

04 Frutera — Drivers I/O et chantiers en cours

DRIVERS D'ENTRÉES / SORTIES

DRIVER	PROTOCOLE	USAGE	ÉTAT
IPX800	HTTP REST	Relais TOR, entrées analogiques 0-10 V / 4-20 mA	Production
Modbus TCP	Modbus/TCP 502	Automates supervisés, chaudière modulante, surpresseurs	Production
Modbus RTU	RS-485	Sondes industrielles, débitmètres, sondes EC/pH	Production
Shelly	HTTP REST	Relais domotique, mesures de consommation	Production
MQTT	MQTT v5	Bus IoT générique, intégration tierces	Production
Simulation	—	Tests, démos, développement hors-site	Production

CHANTIERS PRIORITAIRES

① Migration UI multi-station irrigation

- Modèle de données : FAIT
- Boucle `irrigation_loop.py` : FAIT
- UI `IrrigationPage.tsx` mono-station : À MIGRER
- Sélecteur station, filtrage groupes / vannes / tanks par station
- Vue résumé multi-station type tableau croisé

② Fiabilité production

- Service Windows auto-boot (NSSM ou Task Scheduler)
- Build frontend statique servi par FastAPI (pas de Vite dev en prod)
- Logs persistants `RotatingFileHandler`
- Health endpoint `GET /api/health` avec uptime, clients WS, état boucle

FONCTIONNALITÉS INCOMPLÈTES IDENTIFIÉES

FONCTION	LOCALISATION	ÉTAT ACTUEL
Suivi poids substrat post-irrigation	`irrigation_regulation.py L672`	TODO en commentaire
Monitoring niveau tanks	`UI`	Affiche "--", modèle existe
Détection défaut capteur	`Moteurs`	Enum SENSOR_FAULT jamais set
Enforcement ORP / DO	`Moteur eau de process`	Limites modèle non vérifiées
Timer countdown injection manuelle	`UI ferti`	État existe, pas affiché
UI Advanced Calc	`Frontend`	Moteur OK, UI minimale
UI Event-Based irrigation	`Frontend`	Moteur OK, pas de config UI
Enforcement pression brumisation	`Moteur fog`	Affichée SCADA, pas d'alarme moteur

05 FruteraScan V4 — Stack et workflow

Backend

Runtime	**Node.js**
Framework	**Express 4.18**
Driver BDD	**mysql2/promise (pool 50 conn)**
WebSocket	`ws`
Modbus	`modbus-serial`
HTTP client	`axios`
Imagerie / PDF / Excel	**sharp · pdfkit · exceljs**
Dates	**dayjs, suncalc**
Langage	**JavaScript pur (PAS de TypeScript)**
Tests	**Aucun automatisé (tests manuels)**

Frontend + BDD

Framework front	**Aucun — JS / HTML / CSS pur**
Build	**Aucun (pas de bundler)**
Style	**CSS direct, design slate-900**
Charts	**SVG natif + Chart.js ponctuel**
Langue UI	**100 % français**
SGBD	**MySQL · port 3306**
Tables	**~79, charset utf8mb4**
Pool	**50 connexions max**

WORKFLOW DEV / PROD STRICT

Le projet utilise un workflow à deux environnements parallèles sur la même machine : `dev/` (port 3000, BDD `fruterascan_dev`) et `prod/` (port 3001, BDD `fruterascan_prod`). Toute modification se fait dans `dev/` uniquement, puis se déploie via le script `scripts/deploy-to-prod.bat` qui copie le code, synchronise la BDD et préserve le fichier `config.json` de production.

```
C:\fruterascan\
├── dev\                        ← Développement (port 3000, BDD fruterascan_dev)
│   ├── server.js               ← ~2000 L
│   ├── config.json
│   ├── routes\                 ← endpoints HTTP
│   ├── lib\
│   │   ├── database.js         ← pool MySQL, helpers
│   │   ├── alarm-hub.js        ← centralisation alarmes
│   │   ├── sensor-filter.js    ← lissage / filtrage
│   │   ├── irrigation-analyzer.js
│   │   ├── epidemiological-models.js  ← Botrytis / Oïdium
│   │   ├── formula-evaluator.js       ← mathjs whitelist (refonte)
│   │   ├── collectors\         ← 11 collecteurs (voir p. 8)
│   │   └── automation\         ← engines pilotage (refonte)
│   ├── config\
│   │   ├── location.json       ← GPS site
│   │   └── protocol-config.json
│   ├── migrations\             ← scripts SQL ordonnés
│   ├── backend\database.sql    ← schéma initial
│   └── esp32\firmware\         ← code C++ PlatformIO
├── prod\                       ← Production (port 3001, BDD fruterascan_prod)
├── scripts\                    ← deploy-to-prod, start-dev, backup, NSSM, aggregate-nightly
├── backups\                    ← Dumps SQL horodatés
└── nssm.exe                    ← Service manager Windows
```

DOCUMENTATION API AUTO-GÉNÉRÉE

FruteraScan V4 expose une documentation technique auto-générée à `http://localhost:3000/api/docs` (dev) ou `:3001` (prod). Le système scanne automatiquement les routes API, le schéma de base de données et l'état des collecteurs (régénération auto au démarrage, cache 5 min). Endpoints : `/api/docs` (HTML interactif), `/api/docs/data` (JSON complet), `/api/docs/refresh` (forcer régénération), `/api/docs/search?q=` (recherche full-text), `/api/docs/export` et `/api/docs/markdown` (exports).

06 FruteraScan V4 — Base MySQL (~79 tables)

RÉFÉRENTIEL D'EXPLOITATION

TABLE	CONTENU
societes	Sociétés / clients
sites	Sites physiques (ex : Sablot, Pigousset)
zones_climatiques	Zones / compartiments par site
varietes	Variétés fraises (ex : Gariguette)
substrats	Substrats hors-sol (fibre coco, etc.)
parcelles	Parcelles avec variété + substrat + zone

CAPTEURS ET DONNÉES TEMPS RÉEL · MÉTÉO

CAPTEURS	CONTENU
capteurs	Référentiel capteurs
capteur_data	Mesures brutes (timestamp Unix secondes)
capteur_data_archive	Archive auto à > 180 jours
capteur_data_hourly	Agrégation horaire > 1 an
climat_metriques	VPD, DLI, GDD calculés

MÉTÉO (OPEN-METEO)	CONTENU
meteo_current	Conditions instantanées
meteo_forecast	Prévisions 4 jours
meteo_daily	Agrégats journaliers

CAPTEURS ESP32 AUTONOMES

TABLE	CONTENU
esp32_devices	MAC, GPS, niveau batterie, état
esp32_firmwares	Versions disponibles pour OTA
esp32_spectral_data	Mesures spectromètre AS7341 (11 canaux)

IRRIGATION ET PILOTAGE

IRRIGATION	RÔLE
irrigation_parcelle_config	Programmation par parcelle
irrigation_cycles_log	Historique cycles déclenchés
irrigation_drainage_daily	% drainage quotidien
irrigation_calibrations	Calibration vannes/débits

PILOTAGE TECHNIQUE	RÔLE
chauffage_config	Config chauffage par zone
chauffage_equipements	Inventaire chaudières / pompes
filtration_config	Config filtres à sable
filtration_lavages_log	Historique des lavages

PERSONNEL, TRAÇABILITÉ, ALARMES

PERSONNEL / TRAÇA.	RÔLE
personnel	Salariés et saisonniers
pointage_heures	Heures travaillées
taches	Affectation des tâches
lots_tracabilite	Lots production traçables
registre_phyto	Registre traitements (légal)

ALARMES	RÔLE
alertes	Alarmes actives en cours
alarms_history	Historique horodaté
alarms_notif_config	Cfg Pushover / Telegram

07 FruteraScan V4 — 11 collecteurs et intégrations

Chaque collecteur est une classe Node.js dans `lib/collectors/` implémentant l'interface standard `start() / stop() / poll() / getStatus()`. Un `setInterval` avec fréquence configurable déclenche le cycle : fetch externe → transformation → insert DB → log. Tous les collecteurs sont démarrés en parallèle au boot du serveur et reportent leur état dans la doc auto-générée.

COLLECTEUR	PROTOCOLE	SOURCE	RÔLE
`jeedom-collector`	HTTP JSON-RPC	`192.168.0.20:80`	Domotique / capteurs Jeedom
`ipx800-collector`	HTTP REST	`192.168.0.204:80`	Relais GCE + entrées analogiques
`modbus-collector`	Modbus RTU / TCP	`RS-485 / 502`	Capteurs industriels (100+)
`esp32-collector`	HTTP (POST capteurs)	`Devices LAN`	Gestion ESP32 + diffusion OTA
`openmeteo-collector`	HTTP API	`api.open-meteo.com`	Météo + prévisions 4 j (Marmande 44.4981 / 0.1653)
`shelly-collector`	HTTP REST	`Devices LAN`	Relais Shelly, mesures conso
`pilotage-collector`	Interne	`Engines automation`	Orchestration équipements pilotés
`collector-risks-v2`	Interne (modèles)	`capteur_data + meteo`	Modèles épidémiologiques Botrytis / Oïdium
`collector-rappelsculture`	Interne (logique)	`parcelles + tasks`	Rappels parcelles inactives
`ephy-collector`	HTTP / CSV	`data.gouv.fr E-Phy`	Référentiel phytos officiel (DAR, ZNT, mélanges)

INTÉGRATIONS EXTERNES DÉTAILLÉES

Jeedom

Protocole : HTTP JSON-RPC
Endpoint : `http://192.168.0.20:80/core/api/jeeApi.php`
Auth : API key dans la requête
Usage : remontée capteurs Z-Wave / Zigbee, ordres relais

IPX800 V4 / V5

Protocole : HTTP REST
Endpoint : `http://192.168.0.204/api/xdevices.json`
Auth : clé API
Usage : jusqu'à 32 relais TOR + 16 entrées analogiques par module

Open-Meteo

Protocole : HTTP API publique
Endpoint : `https://api.open-meteo.com/v1/forecast`
Auth : aucune (gratuit)
Usage : météo courante + prévisions 4 jours, coords défaut Marmande 44.4981, 0.1653

E-Phy (data.gouv.fr)

Protocole : téléchargement CSV
Endpoint : `data.gouv.fr — jeu de données ANSES`
Auth : aucune (donnée publique)
Usage : référentiel produits phytopharmaceutiques homologués

08 FruteraScan V4 — Refonte pilotage (5 phases)

La refonte du pilotage technique (chauffage, filtration, ozonation) repose sur 6 principes architecturaux qui visent la robustesse, l'auditabilité et la performance. Aucun de ces principes n'est négociable.

PRINCIPES FONDAMENTAUX DE LA REFONTE

1 Commande = retour état réel

- Cycle COMMANDE → attente → LECTURE ÉTAT RÉEL → confirmation / alarme
- Timeout par défaut : 5 s
- Retry max : 3
- Échec confirmation = alarme automatique

2 Temps réel WebSocket

- Aucun polling BDD pour l'UI
- Channels : `chauffage-state`, `filtration-state`
- État mémoire Map + sync batch BDD toutes les 30 s
- Frontend : DOM diff, jamais re-render complet

3 Zéro Consolaseval()

- Remplacé par `lib/formula-evaluator.js`
- Basé sur mathjs avec whitelist stricte
- Opérateurs autorisés : `+ - * / % **`
- Fonctions : `Math.min / max / abs`

4 Error handling strict

- INTERDIT : `catch` vide
- OBLIGATOIRE : `catch` avec log contexte
- Erreur critique → arrêt automation + alarme
- Erreur transitoire → retry backoff exponentiel

5 Modulaire & testable

- 1 fichier = 1 responsabilité, max 600 lignes
- Classe mère `AutomationBase`
- Config en BDD, pas hardcodée
- Chaque engine isolé, instanciable seul

6 Performance

- État en mémoire — pas de SELECT à chaque tick
- Écritures batch toutes les 30 s
- INSERT historique async non bloquant
- Jamais de requête N+1

ARCHITECTURE CIBLE — LIB/AUTOMATION/

MODULE	RÔLE	LIGNES CIBLE
`automation-base.js`	Classe mère : lifecycle, tick safe, sendCommand + vérif, broadcastState, shutdown propre	~200 L
`chauffage-engine.js`	regulatePression, regulateZones jour / nuit / périodes, checkAlarms, getState	~400 L
`filtration-engine.js`	evaluateLavages (horaire), evaluateDeltaP, executeLavage, manageOzone (horaire + ORP)	~500 L
`formula-evaluator.js`	Évaluateur mathjs whitelist	~50 L
`automation-manager.js`	Orchestrateur : initAll, stopAll, getStatus	~150 L

FRONTEND CIBLE — PILOTAGE REFONDU

Le frontend pilotage s'organise autour d'un orchestrateur d'onglets `pilotage-app.js` (~200 L) qui maintient une connexion WebSocket unique avec reconnect, un `eventBus` interne et un `switchTab` qui ne recharge jamais les vues. Les modules partagés se trouvent dans `shared/` : `ws-client.js` (~80 L, backoff 1s/2s/4s/max 30s), `event-bus.js` (~30 L), `equipment-card.js` (LED + sparkline + commande), `alarm-panel.js`, `mode-selector.js` (OFF / MANUEL / AUTO), `synoptique-base.js` (SVG).

09 ESP32 — Firmware, capteurs et OTA

Plateforme de développement

IDE	**PlatformIO**
Langage	**C++ Arduino**
Libs clés	**ArduinoJson, WiFiManager, Update**
Provisioning	**AP Wi-Fi (WiFiManager)**
OTA	**HTTP Update conditionnel**
Audio	**I2S + FFT (détection abeilles)**
Persistance	`RTC_DATA_ATTR` **entre sleeps**
Allocation	`new / delete` **pour gros objets**

Cycle de vie typique

1 · Boot	**Démarrage processeur**
2 · Init périphériques	**I2C / I2S / ADC**
3 · Wi-Fi + NTP	**Connexion + synchronisation horloge**
4 · Lecture	**Capteurs I2C, audio FFT si activé**
5 · JSON + POST	**Construction payload, HTTP vers serveur**
6 · OTA	**Check version cible, MAJ conditionnelle**
7 · Calcul réveil	**Détermination prochain timer**
8 · Deep sleep	**Mise en veille jusqu'au prochain cycle**

CAPTEURS I2C (SDA = GPIO 21, SCL = GPIO 22)

CAPTEUR	ADRESSE I2C	MESURE
SHT31	`0x44 / 0x45`	Température + humidité relative
BH1750	`0x23 / 0x5C`	Luminosité (lux)
SCD40 / SCD41	`0x62`	CO_2 + T° + HR
AS7341	`0x39`	Spectromètre 11 canaux
MLX90614	`0x5A`	Température infrarouge
ADS1115	`0x48 / 0x49`	ADC 16 bits 4 canaux

CAPTEURS SPÉCIALISÉS

CAPTEUR	BROCHES	MESURE
HX711	GPIO 32 / 15	Cellule de charge (poids substrat)
JSN-SR04T	GPIO 13 / 27	Ultrason niveau eau cuves
Modbus RS-485	TX 17 / RX 16 / DE 4	Capteurs industriels
I2S micro audio	GPIO 26 / 25 / 33	Détection abeilles (FFT)
LED état	GPIO 2	Indicateur visuel
Bouton config	GPIO 0	Reset / mode AP
Tension batterie	ADC GPIO 35	Niveau pile
Tension solaire	ADC GPIO 34	Niveau panneau

Sécurité OTA : double condition obligatoire

La mise à jour OTA n'est tentée que si les deux conditions sont remplies simultanément : batterie > 30 % ET signal Wi-Fi > -75 dBm. Cette règle protège contre un brick en cours de flash si la pile chute ou si le Wi-Fi décroche en plein milieu de la mise à jour. La version cible est annoncée par le serveur ; le capteur compare et ne télécharge que si elle diffère de sa version courante.

Commandes PlatformIO usuelles : `pio run` (compile) · `pio run -t upload --upload-port COM3` (flash USB) · `pio device monitor -b 115200` (moniteur série).

10 LR Connect — Stack et multi-tenant

Backend

Runtime	**Node 22 LTS**
Framework	**Fastify 5**
Langage	**TypeScript strict**
Exécution dev	`tsx watch`
ORM	**Prisma 6**
SGBD	**Postgres 16 + TimescaleDB**
Cache / queues	**Redis 7 + BullMQ**
Validation	**Zod sur env + entrées API**
Port backend	**4000**
LoRaWAN	**ChirpStack v4 (Phase B)**

Frontend

Framework	**React 19**
Bundler	**Vite 6**
Style	**Tailwind 3 (migration 4 prévue)**
Accent	`sky-600` **(#0284c7)**
Polices	**Lobster (titres) · Inter (UI) · JetBrains Mono (data)**
Cibles	**Web + Tauri desktop + CLI + PWA mobile**

CONTRAINTE CRITIQUE — PORTABILITÉ WINDOWS DEV / DEBIAN PROD

Code développé sur Windows mais déployé sur Debian (Phase 0 PC chez Ludo, puis Phase 1 VPS Hetzner Helsinki). Toutes les décisions techniques doivent préserver cette portabilité.

› Docker Compose = unique source de BDD en dev ET en prod (jamais Postgres natif Windows)
› `.gitattributes` force LF sur tous les fichiers source (sinon scripts shell cassent sur Debian)
› `schema.prisma` génère le client pour native ET debian-openssl-3.0.x
› Imports case-sensitive obligatoires : Linux distingue `myFile.ts` de `MyFile.ts`
› Pas de chemin absolu Windows (`C:\…`) dans le code source
› Backend dockerisé en Phase D : migration Win → Debian = `git pull && docker compose up -d`

MODÈLE MULTI-TENANT STRICT

RÈGLE	MISE EN ŒUVRE
Isolation données	customerId obligatoire sur chaque ligne de chaque table
Filtrage	Middleware Fastify injecte le customerId depuis le JWT
Tests d'isolation	OBLIGATOIRES : un user du client A ne doit jamais voir les données du client B
Sessions	JWT access 15 min + refresh 30 j, stockés en Redis
Hypertable	SQL create_hypertable() post-migrate (Prisma ne le gère pas)

ROADMAP PAR PHASES

PHASE	OBJECTIF	ÉTAT
Phase A	Squelette Fastify + Prisma + Postgres + React	EN COURS
Phase B	LoRaWAN branché (ChirpStack v4, multi-schéma)	À VENIR
Phase C	Multi-tenant complet, console admin LR	À VENIR
Phase D	Déploiement Hetzner Debian (backend dockerisé)	À VENIR
Phase E	Polish, site vitrine + boutique, lancement	À VENIR

Décision 15 mai 2026

LR Garden codé en premier (BLE Mi Flora, grand public) parce que le matériel LoRaWAN n'est pas encore en main. ~80 % de code partagé entre LR Garden et LR Connect Pro, qui prendra le relais quand les capteurs LoRa arriveront.

11 Protocoles & intégrations transverses

PROTOCOLE	PORT	COUCHE	UTILISÉ PAR	USAGE
Modbus TCP	502	TCP/IP	Frutera, FruteraScan	Automates supervisés, chaudière modulante, surpresseurs
Modbus RTU	—	RS-485 9600 8N1	Frutera, FruteraScan, ESP32	Sondes industrielles, débitmètres, EC / pH
OPC UA	4840	TCP/IP	Frutera	Automates industriels supervisés
BACnet	47808	UDP	Frutera	Gestion technique de bâtiment
MQTT	1883 / 8883	TCP/IP	Frutera, FruteraScan	Bus IoT générique, intégration tierces
WebSocket	HTTP/S	TCP/IP	3 projets	Temps réel UI ↔ backend (jamais polling)
HTTP REST	80 / 443	TCP/IP	3 projets	API métier, intégrations Jeedom / IPX800 / Shelly
JSON-RPC	80	HTTP	FruteraScan	Domotique Jeedom
LoRaWAN	868 MHz	Radio	LR Connect	Capteurs sans-fil longue portée multi-sites
BLE	2,4 GHz	Radio	LR Garden (LR Connect)	Capteurs Mi Flora grand public
I2C	—	Bus série	ESP32	SHT31, BH1750, SCD40, AS7341, MLX90614, ADS1115
I2S	—	Bus audio	ESP32	Micro pour FFT détection abeilles

STANDARDS D'INTEROPÉRABILITÉ FRUTERA

Engagement « aucune marque imposée »

Frutera est volontairement compatible avec l'ensemble des protocoles industriels et IoT du marché. Le client n'est jamais prisonnier d'un fournisseur :

Industriel : Modbus TCP/RTU, OPC UA, BACnet, sondes 4-20 mA / 0-10 V, compteurs impulsionnels.
IoT & domotique : MQTT, IPX800 V4/V5 (GCE Electronics), Shelly, Tasmota, Sonoff.
Stations météo : Davis, Vantage, Météo France.

CONVENTIONS DE TIMESTAMPS

Tous les timestamps de FruteraScan et LR Connect sont stockés en Unix epoch secondes (`Math.floor(Date.now()/1000)`), jamais en millisecondes. Cette convention est non-négociable et appliquée systématiquement côté capteurs ESP32, collecteurs et BDD. Frutera utilise des timestamps SQLAlchemy `DateTime` avec timezone UTC.

12 Infrastructure, déploiement & conventions

COMPTES, IDENTIFIANTS ET ACCÈS

RUBRIQUE	DÉTAIL
Git / GitHub	lrat002@gmail.com · repo privé lrat002-lab/lr-connect
Tailscale	Tailnet lrat002@ · PC dev Frutera 100.123.213.16 (win-qjji6531ejh)
Cloudflare	Zone lrsystems.org · tunnels admin.lrsystems.org, <client>.lrsystems.org · prévus connect.lrsystems.org, scan.lrsystems.org
RDP	Via Tailscale, compte Windows Administrator
Notion	Workspace LR Systems — pages : Hub, Spec, Roadmap, Boîte à idées, Journal

SERVICES WINDOWS ET AUTO-DÉMARRAGE

Sur les machines en production Windows (Frutera, FruteraScan), les services sont installés via NSSM (Non-Sucking Service Manager) qui transforme un exécutable Node.js ou un script en service Windows démarrable au boot, avec relance automatique en cas de crash. Côté LR Connect (Phase A), un script `infra/start-all.cmd` est exécuté via `shell:startup` au login Windows.

CONVENTIONS DE CODE TRANSVERSES

RÈGLE	DÉTAIL
Langue UI	100 % français — pas de jargon informatique
Langue code	Anglais — variables, fonctions, classes
Commentaires	Français côté FruteraScan, anglais côté Frutera et LR Connect
Taille fichier	≤ 600 lignes (refonte FruteraScan), ≤ 1000 lignes acceptable Frutera
Tests	pytest 180 tests sur Frutera · tests manuels sur FruteraScan · à venir LR Connect
eval()	INTERDIT — remplacé par formula-evaluator.js
catch vide	INTERDIT — toujours log + contexte
Validation entrées	Zod systématique côté LR Connect — validation manuelle côté FruteraScan
WebSocket	Toujours pour temps réel — jamais de polling BDD
API keys	Ne jamais committer — toujours via .env
Pool MySQL	Toujours release() en finally

PIÈGES CONNUS ET SOLUTIONS

SYMPTÔME	SOLUTION
`TypeError: Do not know how to serialize a BigInt`	BigInt.prototype.toJSON = ...toString() au boot Fastify
`Mesures lentes à 10M lignes (LR Connect)`	SQL create_hypertable() TimescaleDB après prisma migrate
`Port 5432 occupé (LR Connect)`	Postgres natif Windows installé — changer port Docker
`WSL Ubuntu en veille`	Containers Docker redémarrent au réveil (10-15 s)
`Web Bluetooth iOS (LR Garden)`	Safari < 2026 non supporté — workaround Capacitor (Phase C+)

Glossaire express

VPD Vapor Pressure Deficit · DLI Daily Light Integral · GDD Growing Degree Days · PPFD Photosynthetic Photon Flux Density · PAR Photosynthetically Active Radiation · R:FR ratio Rouge / Rouge lointain · OCAP Oogendam CO_2 Aanvoerleiding Project · OTA Over-The-Air update · NSSM Non-Sucking Service Manager · WSL Windows Subsystem for Linux · ChirpStack serveur LoRaWAN open-source · TimescaleDB extension Postgres séries temporelles · BullMQ queue Redis pour Node · NTP Network Time Protocol.