Cahier des charges — Smart Greenhouse Manager

Projet individuel (un seul développeur)

Version: 1.1
Date: 01/09/2025

Table des matières

1 Contexte et objectifs (SMART)

Smart Greenhouse Manager est une application web de monitoring d'une serre connectée.

Objectifs SMART :

— **Spécifique** : Suivre température, humidité, luminosité ; gérer seuils ; générer alertes.

— **Mesurable** : <200 ms de latence moyenne API, 99% de disponibilité en démo, 1 mesure/min injectée.

— **Atteignable** : Stack simple (React/Express/MySQL/Mongo), Dockerisée.

— **Réaliste** : Environnement de démo avec simulateur.

— **Temporel** : v1 livrée en 4 semaines.

2 Indicateurs (KPIs)

— Taux de disponibilité API (% uptime).

— Latence P95 des endpoints clés (auth, mesures).

— Nombre d'alertes générées/jour.

— Temps moyen d'affichage du dashboard.

— Taux d'erreurs 5xx backend.

3 Périmètre (In/Out)

In scope : Frontend, API, DBs, simulateur, Docker.
Out scope : Notifications externes, temps réel WS (v2), capteurs physiques, multilingue.

4 Parties prenantes et RACI

Projet réalisé en solo : une seule personne assume l'ensemble des rôles.
RACI simplifié :

— Product Owner (R/A) : priorisation, vision, validation.

— Développement Front/Back (R/A) : conception et implémentation.

— DevOps (R/A) : conteneurisation, déploiement Docker, scripts.

— QA (R/A) : tests manuels, vérifications et validation.

— Parties prenantes externes (I) : lecteurs/évaluateurs, informés ponctuellement.

5 Personas et parcours utilisateur

— Responsable serre : consulte dashboard, ajuste seuils.

— Technicien : analyse historiques d'alertes.

— Admin : gère utilisateurs.

Parcours : Onboarding (register/login) → Dashboard → Config seuils → Suivi alertes.

6 Exigences fonctionnelles détaillées

6.1 Authentification/Autorisation

— Register/Login JWT, rôle user/admin.

— Accès protégés par Bearer token.

6.2 Mesures

— GET /api/mesures ?limit=N renvoie dernières N mesures.

— POST /api/mesures (simulateur) valide payload et stocke.

6.3 Seuils

— POST /api/seuils crée/maj par (utilisateur_id,type), unique.

— GET /api/seuils/ :id récupère par utilisateur.

6.4 Alertes

— Génération auto lors des dépassements au POST /api/mesures.

— GET /api/alertes liste filtrable (à minima par user).

6.5 Administration

— GET /api/users (admin) liste des comptes.

7 Exigences non-fonctionnelles (NFR)

— Sécurité : bcrypt, JWT, CORS restreint.

— Performance : P95 $<$ 300 ms.

— Robustesse : backoff DB, healthcheck.

— Maintenabilité : code structuré, services séparés.

— Portabilité : Docker Compose.

8 Architecture cible

Frontend (5173) → proxy → Backend (4000). Backend → MySQL (utilisateurs, seuils, alertes). Backend → MongoDB (mesures). Simulateur → Backend (POST /api/mesures). Réseau Docker par noms de service.

9 Modèle de données et index

MySQL :

— Utilisateur(email UNIQUE).

— Seuil UNIQUE(utilisateur_id,type) ; FK → Utilisateur(id).

— Alerte FK → Utilisateur(id) ; index(date, type).

MongoDB :

— Mesure : {temperature, humidite, luminosite, date}.

— Index recommandé : date desc ; TTL optionnel (v2) pour purge.

10 Spécification API (JSON)

Exemples :

— POST /api/auth/register Request : { nom, email, mot_de_passe } Response : 201 { id, nom, email }

— POST /api/auth/login Request : { email, mot_de_passe } Response : 200 { token }

(etc. pour toutes les routes décrites)

11 Règles de gestion

— Unicité des seuils par (utilisateur,type).

— Les mesures hors [min,max] génèrent des alertes.

— L'admin seul liste tous les utilisateurs.

12 Sécurité

— Hash bcrypt (cost 10-12).

— JWT signé (secret .env), expiration 1j typique.

— CORS : origine frontend.

— Pas de secrets en clair dans le dépôt.

13 Environnements et configuration

Variables principales (.env) :

```
PORT=4000
JWT_SECRET=...
MYSQL_HOST=mysql, MYSQL_USER=root, MYSQL_PASSWORD=root, MYSQL_DB=greenhouse
MONGO_URI=mongodb://mongo:27017/greenhouse
API_URL=http://backend:4000/api/mesures (simulateur)
VITE_API_BASE=http://backend:4000 (frontend, via docker-compose)
```

14 Déploiement et runbook

Docker Compose : services backend, frontend, simulator, mysql, mongo.
Commandes :

```
cd infrastructure && docker compose up --build
```

Runbook incidents :

— Port 3307 déjà utilisé : arrêter l'ancien MySQL docker/local ou changer le port.
— Frontend 500 proxy : vérifier VITE_API_BASE et service backend.
— Simulateur ECONNREFUSED : vérifier API_URL → backend :4000, état backend.

15 Tests et validation

— Parcours : register → login → dashboard → définir seuils → vérifier alertes.
— API health : GET /api/health.
— JWT requis sur routes protégées.
— Données simulées visibles sous 1-2 minutes.

16 Monitoring et observabilité

— Logs applicatifs (backend, simulateur).
— Healthcheck MySQL et readiness Mongo.
— Metrics (v2) : requêtes/min, erreurs 5xx, latences.

17 Risques et mitigations

— Conflits de ports → ports host configurables.
— Pannes DB → backoff + messages d'erreur clairs.
— Mauvaise config réseau Docker → utiliser noms de service, env cohérents.

18 Planning

— S1 (solo) : Backend + MySQL/Mongo, auth.
— S2 (solo) : Frontend auth + dashboard.
— S3 (solo) : Simulateur + alertes + Docker.
— S4 (solo) : QA, docs, polishing.

19 Critères d'acceptation

Auth/roles OK, dashboard affiche mesures, seuils/alertes opérationnels, compose OK.

20 Roadmap (évolutions)

WebSocket temps réel, charts, notifications, RBAC avancé, purge/TTL, CI/CD.

21 Glossaire

Mesure, Seuil, Alerte, JWT, TTL.
Document vivant : évolue selon retours et priorités.