Documentation Technique

Projet Métier

Réalisé par :
Mohamed Amine Darraj
Adam Khald

Encadré par :
Mr Tawfik Masrour
Mme Ibtissam Elhassani

15 janvier 2026

Version 2.0.0 – Build 15 janvier 2026

Table des matières

1

Introduction Générale

1.1 Contexte et Enjeux

La maintenance industrielle constitue un pilier fondamental de la performance opérationnelle des entreprises manufacturières. Dans un contexte de compétitivité accrue et de digitalisation croissante, les systèmes de **Gestion de Maintenance Assistée par Ordinateur** (GMAO) deviennent des outils stratégiques incontournables.

Définition GMAO

Un système GMAO (ou CMMS – *Computerized Maintenance Management System*) est une solution logicielle permettant de centraliser, planifier, suivre et optimiser l'ensemble des opérations de maintenance d'une organisation.

Le projet **ProAct** s'inscrit dans cette dynamique en proposant une plateforme GMAO moderne, enrichie de capacités d'**Intelligence Artificielle** pour accompagner les équipes de maintenance dans leurs prises de décision.

1.2 Objectifs du Projet

Le système ProAct a été conçu pour répondre aux objectifs suivants :

1. **Centralisation des données** : Regrouper l'ensemble des informations relatives aux équipements, interventions, pièces de rechange et techniciens dans une base de données unifiée.
2. **Suivi des interventions** : Permettre la création, l'assignation et le suivi complet des ordres de travail avec traçabilité des coûts et temps d'arrêt.
3. **Analyse des performances** : Calculer et visualiser les KPIs clés de maintenance (MTBF, MTTR, Disponibilité, OEE).
4. **Aide à la décision par IA** : Intégrer des modules d'intelligence artificielle pour :
   - La prédiction des pannes (maintenance prédictive)
   - L'analyse documentaire via RAG (Retrieval-Augmented Generation)
   - L'assistance conversationnelle (Copilot)
   - L'extraction automatique de données (OCR)
5. **Gestion des compétences** : Suivre les compétences des techniciens et identifier les besoins de formation.
6. **Analyse AMDEC/RPN** : Évaluer la criticité des modes de défaillance via l'analyse RPN (Risk Priority Number).

1.3 Périmètre Fonctionnel

Le tableau 1.1 synthétise les principaux modules fonctionnels du système ProAct.

TABLE 1.1 – Modules fonctionnels du système ProAct

Module	Fonctionnalités	Utilisateurs
Équipements	CRUD, historique, statistiques	Admin, Supervisor
Interventions	Gestion ordres de travail	Tous
Pièces de rechange	Stock, alertes seuil	Admin, Supervisor
Techniciens	Profils, compétences, assignations	Admin
KPI Dashboard	MTBF, MTTR, Disponibilité	Tous
AMDEC/RPN	Modes de défaillance, criticité	Supervisor, Admin
RAG Assistant	Requêtes documentaires IA	Tous
Copilot	Assistant conversationnel	Tous
OCR	Extraction données images	Admin, Supervisor
AI Forecast	Prédiction pannes	Admin, Supervisor

1.4 Structure de ce Document

Ce document technique est organisé en plusieurs chapitres couvrant l'ensemble des aspects du système :

- **Chapitres 2-3** : Vue d'ensemble et architecture frontend
- **Chapitre 4** : Système d'authentification et autorisation
- **Chapitres 5-6** : Architecture backend et modèle de données
- **Chapitres 7-8** : Gestion des interventions et soumissions
- **Chapitre 9** : KPIs et tableaux de bord analytiques
- **Chapitre 10** : Modules d'Intelligence Artificielle
- **Chapitres 11-12** : Sécurité et expérience utilisateur
- **Chapitres 13-14** : Performances et tests
- **Chapitres 15-17** : Déploiement, limites et perspectives

1.5 Technologies Utilisées

Stack Technologique

Frontend :
- Next.js 14 avec App Router
- React 18 + TypeScript
- Makerkit SaaS Boilerplate
- Tailwind CSS + shadcn/ui

Backend :
- FastAPI (Python 3.11+)
- SQLAlchemy ORM
- Pydantic pour la validation

Base de données :
- PostgreSQL (Supabase)
- SQLite (développement local)

Intelligence Artificielle :
- Ollama (LLM local)
- FAISS (recherche vectorielle)
- Prophet (séries temporelles)
- Scikit-learn, XGBoost

Vue d'Ensemble du Système

2.1 Architecture Globale

Le système ProAct adopte une architecture **découplée** (decoupled) avec une séparation nette entre le frontend et le backend, communiquant via une API RESTful sécurisée.

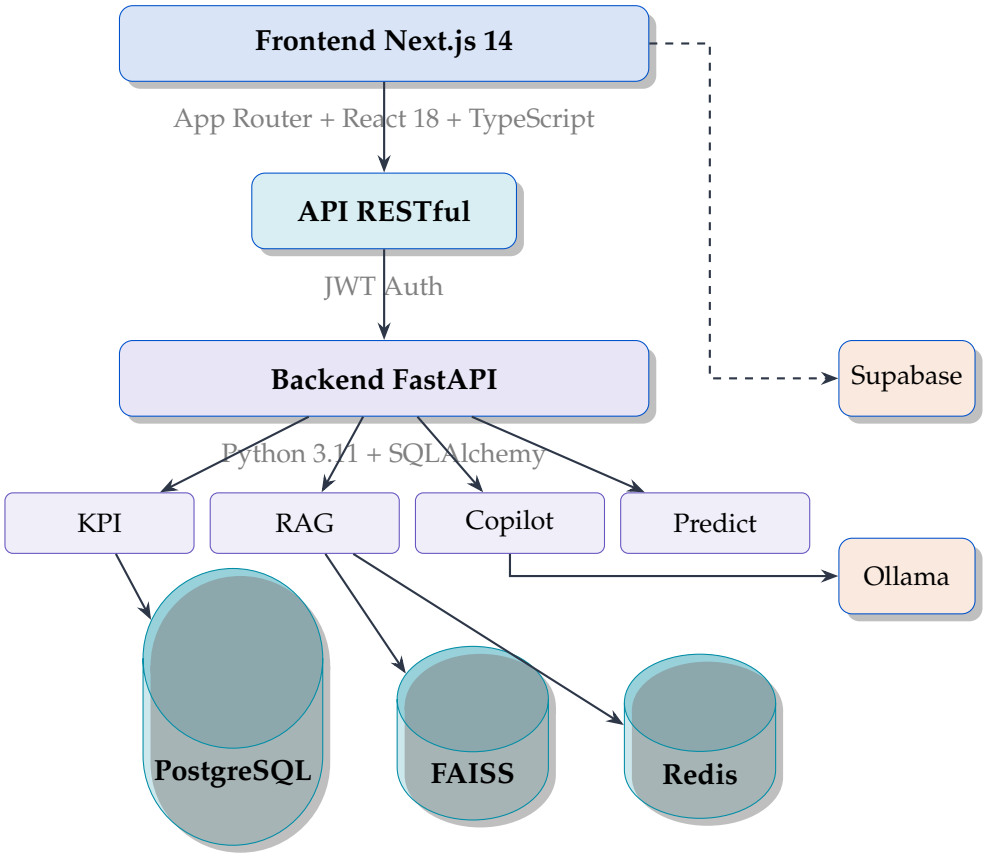

FIGURE 2.1 – Architecture globale du système ProAct

2.2 Flux de Données Principal

Le système gère plusieurs flux de données interconnectés :

1. **Flux d'authentification** : L'utilisateur s'authentifie via Supabase Auth, qui émet un token JWT contenant les claims de rôle.
2. **Flux CRUD** : Les opérations sur les entités (équipements, interventions, etc.) transitent par l'API REST protégée.
3. **Flux analytique** : Les calculs de KPIs sont effectués côté backend et retournés au frontend pour visualisation.
4. **Flux IA** : Les requêtes RAG et Copilot sont traitées par les services spécialisés utilisant Ollama.

2.3 Composants Principaux

2.3.1 Frontend (Next.js)

Le frontend est construit sur le boilerplate **Makerkit SaaS Lite** qui fournit :

- Authentification intégrée avec Supabase
- Structure de navigation multi-tenant
- Composants UI modernes (shadcn/ui)
- Gestion des thèmes (clair/sombre)
- Internationalisation (i18n)

Structure des Pages Dashboard

Les pages du dashboard sont organisées dans `apps/web/app/home/` avec des sous-dossiers dédiés : equipment, interventions, spare-parts, technicians, kpi, amdec, assistant, copilot, ocr, ai-forecast.

2.3.2 Backend (FastAPI)

Le backend expose une API RESTful complète :

TABLE 2.1 – Endpoints API par module

Préfixe	Tag	Description
/api/equipment	Equipment	CRUD équipements, stats
/api/interventions	Interventions	Ordres de travail
/api/spare-parts	Spare Parts	Inventaire pièces
/api/technicians	Technicians	Profils, compétences
/api/kpi	KPIs	MTBF, MTTR, Disponibilité
/api/amdec	AMDEC	Modes de défaillance
/api/rag	RAG System	Upload docs, requêtes
/api/copilot	Copilot	Assistant conversationnel
/api/ocr	OCR Vision	Extraction texte images
/api/predict	AI Forecast	Prédiction RUL, MTBF

2.3.3 Base de Données

Le modèle de données relationnelles comprend les entités principales suivantes :

- **Equipment** : Équipements/machines avec métadonnées
- **Intervention** : Ordres de travail avec coûts et durées
- **SparePart** : Pièces de rechange avec niveaux de stock
- **Technician** : Personnel de maintenance avec compétences

- **FailureMode** : Modes de défaillance pour analyse AMDEC
- **RPNAnalysis** : Évaluations RPN (Gravité × Occurrence × Détection)

2.4 Rôles Utilisateurs

Le système implémente un modèle RBAC avec quatre niveaux :

TABLE 2.2 – Matrice des rôles et permissions

Fonctionnalité	Adm	Sup	Tech	View
Créer équipements	✓	✓	–	–
Modifier interventions	✓	✓	✓	–
Consulter KPIs	✓	✓	✓	✓
Gérer techniciens	✓	–	–	–
Import/Export données	✓	✓	–	–
Utiliser Copilot/RAG	✓	✓	✓	✓

2.5 Points d'Intégration

2.5.1 Supabase

L'intégration Supabase fournit :

- Authentification (email/mot de passe, OAuth)
- Stockage des rôles dans `app_metadata`
- Base de données PostgreSQL managée

2.5.2 Ollama

Le serveur Ollama local héberge les modèles LLM pour la génération de réponses RAG, le traitement des requêtes Copilot, et l'extraction OCR avec modèles vision.

Configuration Requise

Le serveur Ollama doit être accessible à l'URL configurée dans `OLLAMA_BASE_URL` (par défaut `http://ollama:11434` en Docker).

3

Architecture Frontend

3.1 Vue d'Ensemble

Le frontend de ProAct est développé avec **Next.js 14** utilisant le nouvel **App Router**, basé sur le boilerplate **Makerkit SaaS Kit Lite**. Cette architecture moderne permet le rendu côté serveur (SSR), la génération statique (SSG) et les Server Components de React.

Caractéristiques Techniques

- **Framework** : Next.js 14.x avec App Router
- **UI Library** : React 18.x
- **Langage** : TypeScript (strict mode)
- **Styling** : Tailwind CSS v3
- **Composants** : shadcn/ui + Radix UI
- **État** : React Query (TanStack Query)
- **Formulaires** : React Hook Form + Zod

3.2 Structure du Projet

3.2.1 Organisation des Dossiers

Le projet suit une structure monorepo avec `apps/web/` contenant l'application principale. Les dossiers clés sont :

- `app/` : Pages et layouts (App Router)
- `app/(marketing)/` : Pages publiques (landing, pricing)
- `app/auth/` : Authentification (login, signup, callback)
- `app/home/` : Dashboard protégé avec tous les modules GMAO
- `components/` : Composants réutilisables
- `lib/` : Utilitaires, client API, hooks personnalisés

3.2.2 Pages du Dashboard

Le dashboard (`app/home/`) contient les modules fonctionnels :

TABLE 3.1 – Structure des pages dashboard

Dossier	Fonctionnalité
`equipment/`	Gestion des équipements
`interventions/`	Ordres de travail
`spare-parts/`	Inventaire pièces de rechange
`technicians/`	Gestion des techniciens
`kpi/`	Dashboard KPIs avec graphiques
`amdec/`	Analyse AMDEC et RPN
`assistant/`	Interface chat RAG
`copilot/`	Assistant Copilot
`ocr/`	Upload et extraction OCR
`knowledge-base/`	Base de Connaissances (Docs)
`priority-training/`	Priorisation Formations IA
`ai-forecast/`	Prédictions maintenance
`import-export/`	Import/Export CSV

3.3 Système de Layout

3.3.1 Layout Principal

Le fichier `app/home/layout.tsx` définit le layout du dashboard avec support pour deux modes de navigation : **sidebar** (navigation latérale) et **header** (navigation horizontale). Le mode est déterminé par un cookie utilisateur.

3.3.2 Composants de Navigation

- **HomeSidebar** : Navigation latérale avec icônes et labels
- **HomeMenuNavigation** : Navigation horizontale (mode header)
- **HomeMobileNavigation** : Menu hamburger pour mobile
- **GuidanceWidget** : Widget d'aide IA flottant

3.4 Client API GMAO

Le fichier `lib/gmao-api.ts` centralise toutes les communications avec le backend. La classe **`GmaoApiClient`** fournit :

- Gestion automatique des tokens JWT via Supabase
- Méthodes typées pour chaque entité (Equipment, Intervention, etc.)
- Gestion des erreurs avec types personnalisés
- Support des paramètres de requête (pagination, filtres)

Pattern Singleton

Le client API est exporté comme instance globale `gmaoApi` pour assurer une gestion cohérente des tokens d'authentification à travers l'application.

3.5 Composants UI Réutilisables

3.5.1 Data Tables

Les tableaux de données utilisent **TanStack Table** avec les fonctionnalités suivantes :

- Tri multi-colonnes avec indicateurs visuels (flèches asc/desc)
- Pagination côté client configurable
- Filtrage par colonnes avec autocomplétion
- Sparklines intégrés pour visualisation rapide des tendances
- Actions contextuelles (édition, suppression, détails)

3.5.2 Formulaires

Les formulaires sont construits avec **React Hook Form** pour la gestion d'état et **Zod** pour la validation. Chaque schéma de validation définit les contraintes qui génèrent automatiquement les messages d'erreur.

3.5.3 Graphiques

Les visualisations sont réalisées avec **Recharts** : BarChart, LineChart, PieChart, et AreaChart.

3.6 Gestion des Thèmes

Le système supporte les modes clair et sombre via l'attribut `data-theme`. La préférence utilisateur est persistée dans un cookie.

3.7 Optimisations de Performance

Optimisations Implémentées

- **Server Components** : Réduction du bundle JavaScript client
- **Streaming SSR** : Affichage progressif des composants
- **Image Optimization** : Next/Image avec lazy loading
- **Code Splitting** : Routes dynamiques à la demande
- **React Query Caching** : Mise en cache des requêtes API

4

Authentification et Autorisation

4.1 Architecture de Sécurité

ProAct implémente une architecture d'authentification moderne basée sur **JWT** (JSON Web Tokens) avec **Supabase Auth** comme fournisseur d'identité. Le système adopte une approche de **single source of truth** où les rôles utilisateurs sont stockés dans les métadonnées Supabase.

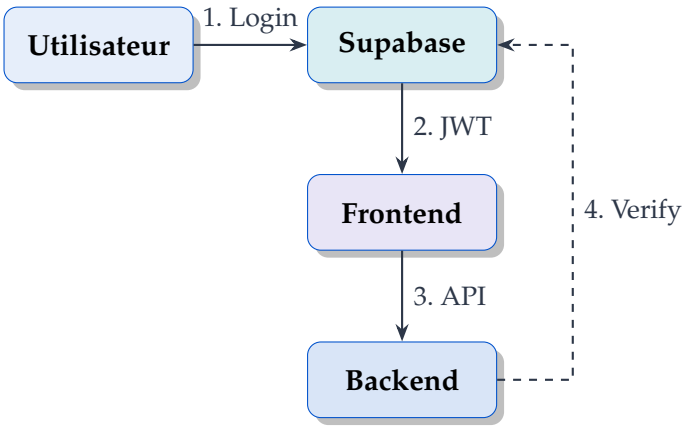

FIGURE 4.1 – Flux d'authentification

4.2 Intégration Makerkit

Le boilerplate Makerkit fournit une intégration Supabase clé en main incluant :

- Configuration client Supabase (browser et server)
- Pages d'authentification pré-construites
- Middleware de protection des routes
- Gestion des sessions avec refresh automatique

4.3 Modèle de Données Utilisateur

Le backend définit une dataclass `AuthUser` avec :

- **id** : Identifiant Supabase unique (claim `sub`)
- **email** : Adresse email de l'utilisateur
- **role** : Rôle extrait de `app_metadata`
- **raw_claims** : Claims JWT bruts

Le système définit quatre rôles hiérarchiques :

TABLE 4.1 – Hiérarchie des rôles

Rôle	Valeur	Description
Admin	admin	Accès total, gestion système
Supervisor	supervisor	Gestion opérationnelle
Technician	technician	Exécution interventions
Viewer	viewer	Consultation seule

4.4 Vérification JWT

Le processus de vérification comprend :

1. Extraction du token depuis l'en-tête `Authorization: Bearer <token>`
2. Vérification de la signature avec `SUPABASE_JWT_SECRET`
3. Validation de l'expiration et des claims requis
4. Extraction du rôle depuis `app_metadata.role`

4.5 Contrôle d'Accès (RBAC)

Le système fournit une factory `require_role()` qui crée des dépendances FastAPI pour le contrôle d'accès. Des raccourcis sont disponibles : `require_admin()`, `require_supervisor_or_ad` `require_technician_or_above()`.

4.6 Matrice des Permissions

TABLE 4.2 – Matrice des permissions par endpoint

	Endpoint	Adm	Sup	Tech	View
Équipements					
	GET /equipment	✓	✓	✓	✓
	POST /equipment	✓	✓	–	–
	DELETE /equipment/ :id	✓	–	–	–
Interventions					
	GET /interventions	✓	✓	✓	✓
	POST /interventions	✓	✓	✓	–
IA					
	POST /rag/query	✓	✓	✓	✓
	POST /ocr/extract	✓	✓	–	–

4.7 Sécurité des Sessions

Bonnes Pratiques Implémentées

- Tokens JWT signés avec secret cryptographique (256 bits min)
- Expiration des tokens configurée (par défaut 1 heure)
- Refresh tokens pour renouvellement transparent
- Validation stricte de tous les claims JWT requis
- Logging des tentatives d'accès non autorisées

4.8 Gestion des Erreurs

TABLE 4.3 – Codes d'erreur d'authentification

Code	Situation	Action
401 Unauthorized	Token manquant/invalide	Redirection vers login
403 Forbidden	Rôle insuffisant	Affichage erreur
500 Server Error	JWT_SECRET non configuré	Alerte admin

5

Architecture Backend

5.1 Vue d'Ensemble

Le backend ProAct est développé avec **FastAPI**, un framework Python moderne offrant des performances élevées et une documentation automatique. L'architecture suit les principes de **Clean Architecture**.

Stack Backend

- **Framework** : FastAPI 0.100+
- **ORM** : SQLAlchemy 2.0
- **Validation** : Pydantic v2
- **Auth** : python-jose (JWT)
- **ML** : Scikit-learn, XGBoost, Prophet
- **LLM** : Ollama
- **Vector Store** : FAISS

5.2 Structure du Projet

Le backend suit une organisation modulaire :

- `app/main.py` : Point d'entrée FastAPI, configuration CORS
- `app/database.py` : Configuration SQLAlchemy
- `app/models.py` : Modèles ORM
- `app/schemas.py` : Schémas Pydantic
- `app/security.py` : Auth JWT et guards RBAC
- `app/routers/` : 15 routers par domaine
- `app/services/` : Logique métier

5.3 Couche Routers

Les routers définissent les endpoints REST :

TABLE 5.1 – Routers principaux

Router	Préfixe	Fonctionnalités
equipment	/api/equipment	CRUD équipements, stats
interventions	/api/interventions	Ordres de travail
spare_parts	/api/spare-parts	Inventaire, alertes
technicians	/api/technicians	Profils, compétences
kpi	/api/kpi	MTBF, MTTR, Dashboard
amdec	/api/amdec	Modes défaillance, RPN
rag	/api/rag	Upload docs, requêtes
copilot	/api/copilot	Assistant IA
ocr	/api/ocr	Extraction texte
prediction	/api/predict	Prévision pannes

5.4 Couche Services

Les services encapsulent la logique métier :

- **KPIService** : Calculs MTBF, MTTR, Disponibilité
- **CopilotService** : Orchestration des requêtes IA
- **PredictionService** : Modèles ML pour RUL
- **OCRService** : Extraction via modèles VLM
- **RAGService** : Pipeline de recherche documentaire

5.5 Gestion des Erreurs

TABLE 5.2 – Codes d'erreur HTTP

Code	Signification	Exemple
400	Bad Request	Validation Pydantic échouée
401	Unauthorized	JWT expiré ou invalide
403	Forbidden	Rôle non autorisé
404	Not Found	Équipement ID inconnu
422	Unprocessable	Champ requis manquant
500	Server Error	Exception non gérée

5.6 Endpoints de Santé

Le backend expose des endpoints de monitoring :

- `GET /health` : Vérification DB et RAG
- `GET /api/stats` : Compteurs des entités
- `GET /` : Informations version

5.7 Documentation API

FastAPI génère automatiquement la documentation :

- **Swagger UI** : `/docs`
- **ReDoc** : `/redoc`
- **OpenAPI** : `/openapi.json`

Tags API

Les endpoints sont organisés par tags : Equipment, Interventions, Spare Parts, Technicians, KPIs, AMDEC, RAG, Copilot, OCR, Forecast.

6 Modèle de Données

6.1 Vue d'Ensemble

Le modèle de données de ProAct est implémenté avec **SQLAlchemy ORM** et conçu pour capturer l'ensemble des entités et relations d'un système GMAO complet. La base de données supporte PostgreSQL (production) et SQLite (développement).

6.2 Schéma Entités-Relations

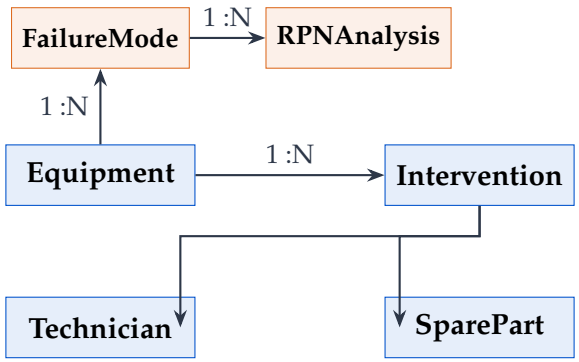

FIGURE 6.1 – Diagramme entités-relations

6.3 Entités Principales

6.3.1 Equipment

L'entité **Equipment** représente les actifs physiques à maintenir.

TABLE 6.1 – Attributs de l'entité Equipment

Attribut	Type	Description
id	Integer (PK)	Identifiant unique
designation	String(200)	Nom de l'équipement
type	String(100)	Type ou catégorie
location	String(200)	Emplacement physique
status	Enum	Active, Inactive, Maintenance
acquisition_date	Date	Date d'acquisition
manufacturer	String(100)	Fabricant
serial_number	String(100)	Numéro de série (unique)

6.3.2 Intervention

L'entité **Intervention** capture les ordres de travail :

- **Classification** : Type de panne, catégorie, cause
- **Timing** : Date demande, réalisation, durée
- **Coûts** : Coût matériel, main d'œuvre, total
- **Statut** : Open, In Progress, Completed, Closed

6.3.3 SparePart

Gère l'inventaire des pièces de rechange : désignation, référence unique, coût unitaire, stock actuel, seuil d'alerte, fournisseur.

6.3.4 Technician

Représente le personnel : nom, email unique, spécialité, taux horaire, statut (Active, Inactive, On Leave).

6.4 Tables d'Association

- **InterventionPart** : N :M interventions-pièces avec quantité
- **TechnicianAssignment** : N :M interventions-techniciens avec heures
- **TechnicianSkill** : N :M techniciens-compétences avec niveau

6.5 Modèles AMDEC

- **FailureMode** : Mode de défaillance avec effet et cause
- **RPNAnalysis** : Scores Gravité, Occurrence, Détection (1-10)

6.6 Modèles RAG

- **RAGDocument** : Métadonnées des documents indexés
- **RAGDocumentChunk** : Segments de texte pour recherche
- **RAGQuery** : Historique des requêtes

6.7 Schémas Pydantic

Les schémas Pydantic assurent la validation :

- ***Base** : Attributs communs
- ***Create** : Schéma de création
- ***Update** : Schéma mise à jour (optionnels)
- ***Response** : Réponse avec id et timestamps

Validation Automatique

Pydantic v2 génère la validation des types, les contraintes de champs, et la documentation OpenAPI automatiquement.

7

Système d'Interventions

7.1 Vue d'Ensemble

Le module d'interventions constitue le cœur opérationnel du système GMAO. Il gère l'ensemble du cycle de vie des ordres de travail de maintenance.

Définition - Intervention

Une **intervention** représente une action de maintenance : **corrective** (suite à panne), **préventive** (planifiée), ou **améliorative** (upgrade).

7.2 Cycle de Vie

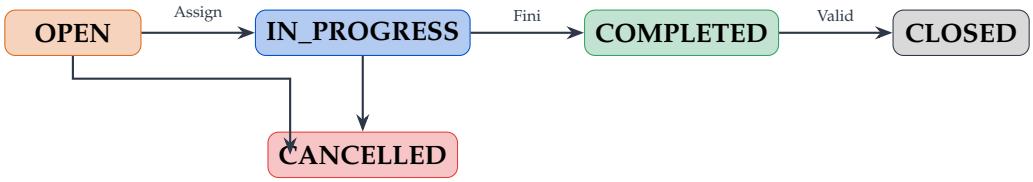

FIGURE 7.1 – États du cycle de vie

7.3 Création d'une Intervention

Via `POST /api/interventions/` avec :

- **equipment_id** (requis) : Identifiant équipement
- **type_panne** : Classification
- **categorie_panne** : Catégorie
- **cause** : Cause identifiée
- **date_demande** : Date de la demande

7.4 Assignation et Pièces

L'assignation crée un **`TechnicianAssignment`** (technicien, heures, taux). L'ajout de pièces via `POST /api/interventions/{id}/parts` vérifie le stock et décrémente automatiquement.

7.5 Requêtes et Filtrage

TABLE 7.1 – Paramètres de filtrage

Paramètre	Type	Description
skip/limit	int	Pagination
status	enum	Filtrer par statut
equipment_id	int	Filtrer par équipement
start_date	date	Début de période
end_date	date	Fin de période
type_panne	string	Recherche par type

7.6 Données Capturées

TABLE 7.2 – Données par intervention

Donnée	Type	Utilisation
Durée indisponibilité	Float (heures)	Calcul MTTR
Coût matériel	Float (€)	Analyse coûts
Coût main d'œuvre	Float (€)	Analyse coûts
Type de panne	String	Distribution
Date intervention	Date	Tendances

Traçabilité

Chaque intervention maintient un historique complet : horodatages, transitions de statut, assignations.

8

Système de Soumission Technicien

8.1 Vue d'Ensemble

Le module de soumission permet aux techniciens de reporter leurs activités et soumettre leurs heures de travail.

8.2 Profil Technicien

TABLE 8.1 – Attributs du profil technicien

Attribut	Type	Description
Nom / Prénom	String	Identité
Email	String (unique)	Contact et auth
Téléphone	String	Contact mobile
Spécialité	String	Domaine d'expertise
Taux horaire	Float	Coût horaire
Statut	Enum	Active, Inactive, On Leave
Date embauche	Date	Ancienneté

Les compétences sont associées avec niveau de maîtrise (1-5) et statut de certification.

8.3 Workflow

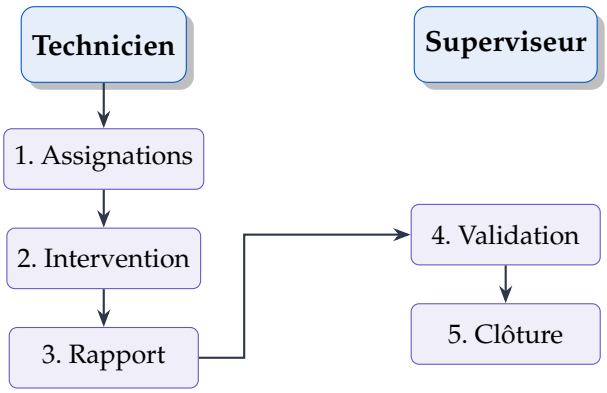

FIGURE 8.1 – Workflow de soumission

8.4 Fonctionnalités API

- `GET /api/technicians/me/assignments` : Interventions assignées
- `PUT /api/assignments/{id}/submit-hours` : Soumission heures
- `POST /api/interventions/{id}/submit-report` : Rapport complet

8.5 Contenu du Rapport

Description des travaux, cause identifiée, actions correctives, pièces utilisées, heures travaillées, date de réalisation.

8.6 Statistiques

TABLE 8.2 – Statistiques technicien

Métrique	Description
Total interventions	Nombre réalisées
Total heures	Cumul heures travaillées
Coût MO	Somme des coûts générés
Moyenne heures/interv.	Indicateur efficacité
Nombre compétences	Polyvalence

Intégration Future

L'intégration notification (email, SMS, push) est prévue pour les versions futures.

9

KPI et Analytics

9.1 Introduction aux KPIs de Maintenance

Les **Key Performance Indicators** (KPIs) sont des métriques essentielles pour évaluer l'efficacité des opérations de maintenance. ProAct implémente les indicateurs standards de l'industrie, calculés automatiquement à partir des données d'intervention.

9.2 Indicateurs Principaux

9.2.1 MTBF – Mean Time Between Failures

Définition MTBF

Le **MTBF** (Temps Moyen Entre Pannes) mesure la fiabilité d'un équipement. Plus le MTBF est élevé, plus l'équipement est fiable.

Formule :

$$MTBF = \frac{\sum_{i=1}^{n-1}(t_{i+1} - t_i)}{n-1} \tag{9.1}$$

Où t_i représente la date de la i-ème panne et n le nombre total de pannes.

Le calcul du MTBF dans ProAct :

1. Récupère les interventions correctives triées par date
2. Calcule les intervalles entre pannes consécutives (en heures)
3. Retourne la moyenne des intervalles

Conditions : Nécessite au minimum 2 pannes pour calculer un intervalle. Retourne `null` si données insuffisantes.

9.2.2 MTTR – Mean Time To Repair

Définition MTTR

Le **MTTR** (Temps Moyen de Réparation) mesure l'efficacité des réparations. Un MTTR bas indique des réparations rapides.

Formule :

$$MTTR = \frac{\sum_{i=1}^{n} D_i}{n} \tag{9.2}$$

Où D_i représente la durée d'indisponibilité de la i-ème intervention.

Le MTTR utilise le champ `duree_indisponibilite` des interventions, exprimé en heures.

9.2.3 Disponibilité

Définition Disponibilité

La **Disponibilité** mesure le pourcentage de temps où l'équipement est opérationnel.
Formule :

$$A = \frac{MTBF}{MTBF + MTTR} \times 100 = \frac{T_{op} - T_{down}}{T_{op}} \times 100 \tag{9.3}$$

Le calcul prend en compte :

- La période d'analyse (start_date à end_date)
- Les heures opérationnelles par jour (par défaut 24h)
- Les jours opérationnels par semaine (par défaut 7)
- Le cumul des durées d'indisponibilité

La valeur est bornée entre 0% et 100%.

9.3 Dashboard KPIs

L'endpoint `GET /api/kpi/dashboard` retourne un ensemble consolidé de métriques en un seul appel :

TABLE 9.1 – Structure du Dashboard KPIs

Catégorie	Métriques
Core Metrics	MTBF, MTTR, Disponibilité
Counts	Total interventions, équipements actifs
Costs	Coût total, matériel, main d'œuvre
Distributions	Par type de panne, par statut
Trends	Données mensuelles pour graphiques

9.3.1 Paramètres de Filtrage

- **start_date / end_date** : Période d'analyse
- **equipment_id** : Filtrer par équipement spécifique
- **operational_hours_per_day** : Heures de fonctionnement (1-24)

9.4 Distributions et Tendances

9.4.1 Distribution des Pannes

L'endpoint `GET /api/kpi/failure-distribution` retourne le comptage par type de panne avec pourcentages. Permet d'identifier les types de pannes les plus fréquents pour prioriser les actions préventives.

9.4.2 Répartition des Coûts

L'endpoint `GET /api/kpi/cost-breakdown` fournit :

- Coût matériel total et pourcentage
- Coût main d'œuvre total et pourcentage
- Coût total de maintenance

9.5 Visualisation Frontend

Le dashboard KPI du frontend affiche :

- **Cards métriques** : MTBF, MTTR, Disponibilité avec indicateurs de tendance
- **Graphique en barres** : Distribution des types de pannes
- **Graphique en ligne** : Évolution temporelle des interventions
- **Pie chart** : Répartition des coûts (matériel vs MO)
- **Tableau récapitulatif** : KPIs par équipement

9.6 Interprétation des KPIs

TABLE 9.2 – Interprétation des KPIs

KPI	Bon	Acceptable	Critique
MTBF	> 500h	200-500h	< 200h
MTTR	< 2h	2-8h	> 8h
Disponibilité	> 95%	85-95%	< 85%

Bonnes Pratiques

Les seuils ci-dessus sont indicatifs. Il est recommandé de les adapter selon le type d'industrie, la criticité des équipements, et les objectifs de performance définis.

10

Intelligence Artificielle

10.1 Vue d'Ensemble

ProAct intègre plusieurs modules IA pour augmenter les capacités analytiques du système GMAO.

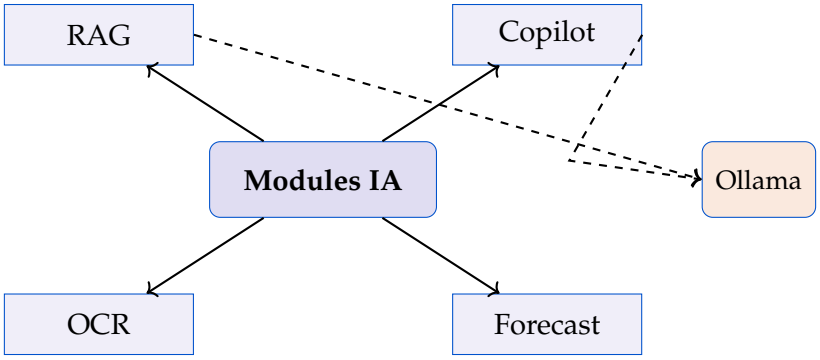

FIGURE 10.1 – Modules IA and their connections to services and Ollama

10.2 Système RAG

RAG (Retrieval-Augmented Generation) permet d'interroger une base documentaire en langage naturel :

1. **Retrieval** : Recherche vectorielle de passages pertinents
2. **Augmentation** : Enrichissement du contexte LLM
3. **Generation** : Production de réponses contextualisées

Modules : RAGService, DocumentProcessor, EmbeddingService, VectorStore (FAISS), LLM-Service, CacheService (Redis).

10.3 Maintenance Copilot

Le **Copilot** est un assistant conversationnel qui peut expliquer les KPIs, générer des résumés de santé équipement, produire des rapports, et recommander des actions.

TABLE 10.1 – Intentions Copilot

Intent	Description
KPI_EXPLANATION	Questions métriques
EQUIPMENT_HEALTH	État équipement
INTERVENTION_REPORT	Demande rapport
GENERAL_QUESTION	Questions générales

10.4 OCR Vision

Extraction de texte via modèles Vision-Language (qwen2.5vl :3b). Formats : Markdown, HTML, JSON, Texte brut. Images optimisées avant envoi.

10.5 AI Forecast

Prédiction de la durée de vie restante (RUL) et prévision MTBF sur 90 jours.

TABLE 10.2 – Algorithmes de prédiction

Algorithme	Type	Usage
Random Forest	ML supervisé	Prédiction RUL
XGBoost	ML supervisé	RUL améliorée
Prophet	Séries temp.	Tendance MTBF

10.6 Fournisseurs LLM

Le système supporte désormais une architecture multi-providers permettant de basculer entre l'inférence locale (Ollama) et cloud (Groq).

10.6.1 Configuration

TABLE 10.3 – Variables d'environnement AI

Variable	Provider	Description
`LLM_PROVIDER`	–	`ollama` ou `groq`
`OLLAMA_BASE_URL`	Ollama	URL serveur local
`GROQ_API_KEY`	Groq	Clé API Cloud
`AI_MODEL_NAME`	Commun	ex : `llama3-70b-8192`

10.6.2 Comparatif

- **Ollama** : Confidentialité totale (local), pas de coût API, dépend du hardware serveur.
- **Groq** : Inférence ultra-rapide (LPUs), idéal pour production, coût à l'usage.

Prérequis Ollama

En mode local, les modèles doivent être téléchargés via `ollama pull <model>`.

11 Sécurité

11.1 Vue d'Ensemble

La sécurité de ProAct est conçue selon le principe de **défense en profondeur**.

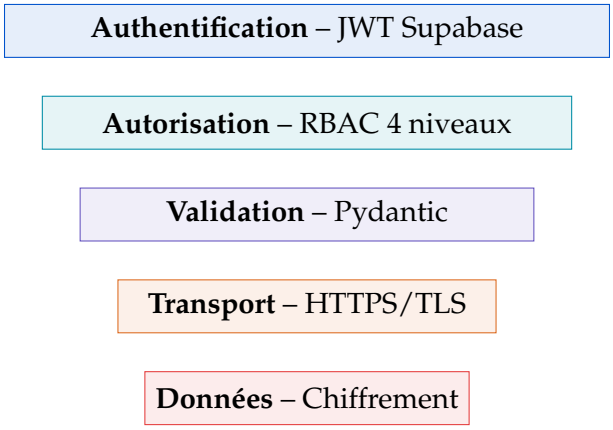

FIGURE 11.1 – Couches de sécurité

11.2 Authentification JWT

- **Algorithme** : HS256
- **Secret** : `SUPABASE_JWT_SECRET` (256 bits min)
- **Expiration** : 1 heure par défaut
- **Refresh** : Automatique par Supabase

11.3 Autorisation RBAC

TABLE 11.1 – Hiérarchie des rôles

Rôle	Niveau	Privilèges
Admin	4	Accès total, gestion utilisateurs
Supervisor	3	Gestion opérationnelle
Technician	2	Interventions, soumissions
Viewer	1	Consultation seule

11.4 Validation des Entrées

Pydantic valide types et contraintes. SQLAlchemy ORM génère des requêtes paramétrées pour prévenir les injections SQL.

11.5 Sécurité des Communications

Configuration CORS : origines autorisées, méthodes (GET, POST, PUT, DELETE), headers (Authorization, Content-Type).

Configuration Production

Remplacer `allow_origins=["*"]` par la liste explicite des domaines autorisés.

11.6 Protection des Données

TABLE 11.2 – Classification des données

Donnée	Sensibilité	Protection
Mots de passe	Critique	Hash bcrypt (Supabase)
Tokens JWT	Haute	HttpOnly cookies
Emails	Moyenne	Accès restreint
Coûts	Moyenne	Visible selon rôle
Données tech.	Basse	Accès authentifié

11.7 Checklist Sécurité

TABLE 11.3 – Checklist de sécurité

Mesure	Impl.	Prod
Authentification JWT	✓	✓
RBAC 4 niveaux	✓	✓
Validation Pydantic	✓	✓
Protection SQL Injection	✓	✓
Configuration CORS	✓	Configurer
HTTPS/TLS	–	Requis
Rate Limiting	–	Recommandé

12

UX et Rôles Utilisateurs

12.1 Principes de Design

L'interface est conçue selon les principes de clarté, efficacité, cohérence, adaptabilité et accessibilité (WCAG niveau AA).

12.2 Parcours Utilisateurs

12.2.1 Administrateur

Configuration système, gestion utilisateurs, import/export, analyse KPIs globaux, configuration AMDEC.

12.2.2 Superviseur

Planification interventions, assignation techniciens, validation rapports, suivi KPIs équipe, gestion stock.

12.2.3 Technicien

Consultation interventions assignées, réalisation et documentation, soumission rapports, assistant IA.

12.2.4 Viewer

Consultation tableaux de bord, visualisation KPIs, historique interventions, RAG en lecture seule.

12.3 Structure de Navigation

TABLE 12.1 – Menu de navigation

Section	Icône	Adm	Sup	Tech	View
Dashboard	Home	✓	✓	✓	✓
Équipements	Settings	✓	✓	✓	✓
Interventions	Wrench	✓	✓	✓	✓
Pièces	Package	✓	✓	–	–
Techniciens	Users	✓	✓	–	–
KPI	BarChart	✓	✓	✓	✓
AMDEC	Alert	✓	✓	–	–
IA & Analytics					
RAG	Message	✓	✓	✓	✓
Copilot	Bot	✓	✓	✓	✓
OCR	Scan	✓	✓	–	–
Forecast	Trend	✓	✓	–	–
Import/Export	Upload	✓	✓	–	–

12.4 Composants d'Interface

12.4.1 Dashboard

Cards KPI avec tendances, graphiques distribution pannes, alertes stock bas, actions rapides.

12.4.2 Tableaux

Tri par colonnes, pagination configurable, filtrage multi-critères, recherche globale, export CSV, actions inline.

12.4.3 Formulaires

Validation temps réel, autocomplétion, confirmation actions destructives.

12.5 Responsive Design

TABLE 12.2 – Breakpoints

Device	Largeur	Adaptation
Mobile	< 640px	Menu hamburger, cards empilées
Tablette	640-1024px	Sidebar réduite, grille 2 colonnes
Desktop	> 1024px	Sidebar complète, grille 3+ colonnes

12.6 Thèmes et Accessibilité

Modes clair/sombre via `data-theme`. Conformité WCAG : contraste 4.5 :1, navigation clavier, labels ARIA, skip links.

12.7 Widget d'Aide IA

Le **GuidanceWidget** est un assistant flottant offrant des suggestions contextuelles et intégration Copilot.

13

Performances et Scalabilité

13.1 Objectifs de Performance

TABLE 13.1 – SLOs (Service Level Objectives)

Métrique	Objectif	Critique
Latence API (P95)	< 200ms	< 500ms
Latence RAG Query	< 3s	< 5s
Time to First Byte	< 100ms	< 300ms
Disponibilité	> 99.5%	> 99%

13.2 Optimisations Backend

13.2.1 Indexation Base de Données

Index SQL sur colonnes fréquemment filtrées : intervention_date_type, intervention_equipment_date, intervention_status, equipment_serial.

13.2.2 Pagination

Paramètres `skip` et `limit` (max 500), comptage total optimisé, indicateur `has_more`.

13.2.3 Caching Redis

Cache RAG : clé = hash MD5 requête, TTL = 1 heure, invalidation automatique.

13.2.4 Async I/O

FastAPI utilise le modèle asynchrone pour les opérations I/O non-bloquantes.

13.3 Optimisations Frontend

- **Server Components** : Réduction bundle JS
- **React Query** : staleTime 5min, cacheTime 30min
- **Code Splitting** : Chargement dynamique composants lourds

13.4 Scalabilité Horizontale

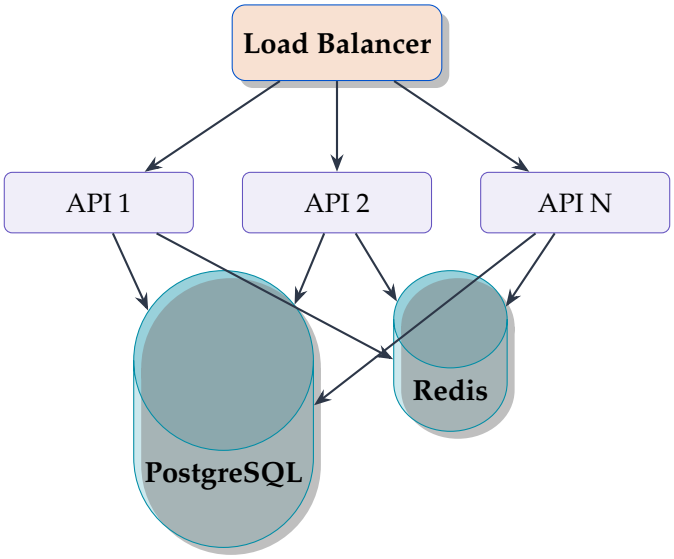

FIGURE 13.1 – Architecture horizontale

Architecture stateless : pas d'état en mémoire, sessions par Supabase, cache partagé Redis.

13.5 Monitoring

Health checks vérifient : PostgreSQL, Redis, Ollama, FAISS.

Métriques recommandées : Request Rate, Error Rate, Latency (P50/P95/P99), DB connections, Cache hit ratio.

Outils Recommandés

Prometheus, Grafana, Sentry, Vercel Analytics.

14

Tests et Validation

14.1 Stratégie de Test

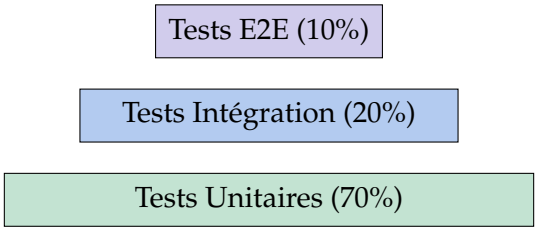

FIGURE 14.1 – Pyramide des tests

14.2 Tests Backend

14.2.1 Structure

Tests dans `backend/tests/` : conftest.py (fixtures), test_equipment.py, test_interventions.py, test_kpi.py, test_security.py.

14.2.2 Fixtures Pytest

- **db_session** : SQLite en mémoire
- **client** : TestClient FastAPI
- **auth_headers** : Headers JWT de test

14.2.3 Tests Unitaires

Calcul MTBF/MTTR, Disponibilité bornée 0-100%, retour null si données insuffisantes.

14.2.4 Tests Intégration API

Création POST, lecture GET avec pagination/filtres, mise à jour PUT, suppression DELETE, codes d'erreur (401, 403, 404).

14.3 Tests Frontend E2E

Playwright simule les parcours : navigation, authentification, formulaires, assertions.

14.4 Couverture de Code

TABLE 14.1 – Couverture cible

Module	Actuel	Cible
Services (KPI, Copilot)	75%	80%
Routers	70%	75%
Models/Schemas	90%	90%
Security	85%	90%

14.5 CI/CD Testing

Pipeline GitHub Actions : checkout, setup Python, tests avec couverture, upload Codecov, seuil 70% minimum.

Bonnes Pratiques

Chaque nouvelle fonctionnalité doit avoir des tests. Les bugs corrigés doivent inclure un test de non-régression.

15 Déploiement

15.1 Vue d'Ensemble

TABLE 15.1 – Options de déploiement

Config	Usage	Caractéristiques
Dev local	Dev/Test	Docker Compose, SQLite, hot-reload
Staging	Pré-prod	Docker, PostgreSQL, Supabase test
Production	Live	K8s/Docker, Supabase prod, CDN

15.2 Déploiement Local

15.2.1 Prérequis

Docker Desktop 4.x+, Node.js 18.x+/pnpm, Python 3.11+, Git.

15.2.2 Services Docker Compose

- **api** : Backend FastAPI (port 8000)
- **ollama** : Serveur LLM (port 11434)
- **redis** : Cache (port 6379)

15.2.3 Lancement

Backend : `docker-compose up -d` depuis `backend/`. Modèles : `ollama pull llama3.2:3b`.
Frontend : `pnpm install` puis `pnpm run dev`.

15.3 Configuration Supabase

Variables frontend (.env.local) : NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_
NEXT_PUBLIC_API_BASE_URL.

Variables backend (.env) : DATABASE_URL, SUPABASE_JWT_SECRET, OLLAMA_BASE_URL,
REDIS_HOST.

15.4 Déploiement Production

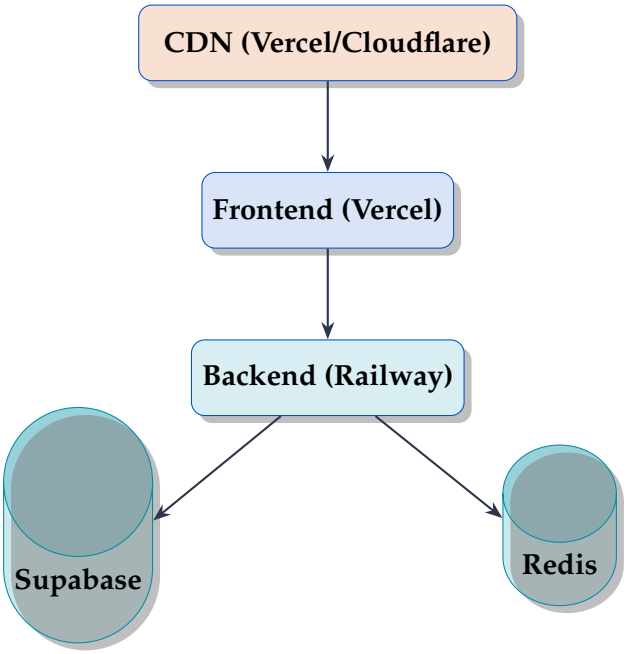

FIGURE 15.1 – Architecture production

15.5 Migrations Alembic

`alembic revision --autogenerate -m "desc"` : créer migration. `alembic upgrade head` : appliquer. `alembic downgrade -1` : rollback.

15.6 Checklist Déploiement

TABLE 15.2 – Checklist pré-déploiement

Tâche	Dev	Prod
Variables d'environnement	✓	✓
Base de données migrée	✓	✓
Modèles Ollama téléchargés	✓	✓
CORS configuré	–	✓
HTTPS activé	–	✓
Monitoring configuré	–	✓
Backup automatique	–	✓

Important

Tester en staging, vérifier sécurité, configurer backups et monitoring avant production.

16

Limites et Contraintes

16.1 Limites Techniques

16.1.1 Dépendance à Ollama

- **GPU** : Performances dégradées sans GPU
- **Démarrage** : 30-60 secondes pour charger modèles
- **VRAM** : 4-8 GB minimum pour VLM
- **Latence** : 2-5 secondes par requête IA

Impact

Sans GPU, temps de réponse 10-30 secondes ou timeout sur requêtes complexes.

16.1.2 Scalabilité RAG

TABLE 16.1 – Limites du système RAG

Contrainte	Limite Pratique
Documents indexés	~1000 documents
Taille par document	~50 pages / 100 KB
Chunks par document	~200 chunks
Requêtes concurrentes	~10/seconde

16.2 Limites Fonctionnelles

- Modèle RBAC simple avec 4 rôles fixes
- Pas de permissions granulaires par ressource
- Pas de multi-tenancy natif
- Workflow intervention linéaire (pas de branches)
- Pas de notifications push temps réel
- Pas de générateur de rapports personnalisés

16.3 Limites IA

- **Données requises** : Min. 20-30 interventions par équipement
- **Précision RUL** : MAE typique 15-30 jours
- **Hallucinations** : Réponses parfois incorrectes

- **Contexte** : ∼4-8K tokens max

16.4 Contraintes Intégration

Pas d'intégration ERP native, pas de connecteurs IoT/SCADA, API REST uniquement, authentification Supabase uniquement.

16.5 Contraintes Opérationnelles

TABLE 16.2 – Contraintes opérationnelles

Aspect	Contrainte
Utilisateurs concurrents	∼50-100 selon infrastructure
Volume de données	∼1M interventions max recommandé
Taille uploads	10 MB par fichier
Backup	Non automatisé

Évolutions Prévues

Ces limitations sont priorisées pour les versions futures. Voir chapitre 17.

17

Perspectives d'Évolution

17.1 Feuille de Route

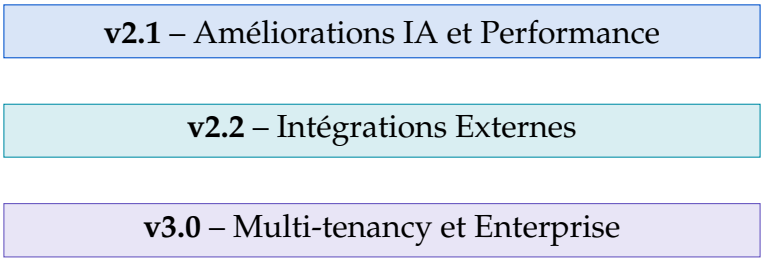

FIGURE 17.1 – Feuille de route

17.2 Version 2.1 – Court Terme

17.2.1 Intelligence Artificielle

- **RAG** : Support PDF natif, recherche hybride, historique multi-tour
- **Prédiction** : Modèles LSTM, saisonnalité, fleet-level analytics
- **OCR** : Support multi-pages, extraction formulaires structurés

17.2.2 Performance

Agrégation KPIs pré-calculée, pagination curseurs, compression gzip, intégration Prometheus/Grafana.

17.3 Version 2.2 – Intégrations

TABLE 17.1 – Intégrations prévues

Système	Type	Fonctionnalité
SAP PM	ERP	Import/sync équipements et OT
SCADA/OPC-UA	IoT	Données temps réel machines
Microsoft 365	Productivité	Notifications, calendrier
Power BI	Analytics	Export reporting avancé

17.3.1 Notifications et API

Push PWA, alertes email, Slack/Teams, SMS critiques. Endpoint GraphQL, WebSocket temps réel, webhooks sortants, API versioning.

17.4 Version 3.0 – Enterprise

17.4.1 Multi-tenancy

Isolation données par organisation, personnalisation par tenant, quotas et limites configurables.

17.4.2 Sécurité Enterprise

SSO SAML 2.0, LDAP/AD, MFA obligatoire, audit trail immutable, conformité RGPD.

17.4.3 Fonctionnalités Avancées

- **Workflows** : Designer graphique, approbations multi-niveaux, escalations
- **Reporting** : Générateur personnalisé, dashboards configurables
- **Mobile** : App iOS/Android native, mode hors-ligne, QR codes

17.5 Évolutions Techniques

Microservices optionnel, Kubernetes, event sourcing, fine-tuning LLM sur données GMAO, vision AI inspection.

17.6 Conclusion

ProAct constitue une base solide pour une GMAO moderne intégrant l'IA. Les évolutions visent à renforcer les capacités prédictives, faciliter l'intégration, répondre aux exigences enterprise, et améliorer l'expérience utilisateur.