

ProAct

PLATEFORME GMAO INTELLIGENTE

Documentation Technique

Système de Gestion de Maintenance
Assistée par Ordinateur

Stack Technologique

Next.js 14 · React 18 · TypeScript

FastAPI · SQLAlchemy · PostgreSQL

Supabase Auth · Makerkit SaaS

RAG · Copilot IA · OCR Vision · ML Forecasting

Mohamed Amine Darraj

Projet Académique – Génie Industriel

14 janvier 2026

Version 2.0.0

Table des matières

1	Introduction Générale	2
1.1	Contexte et Enjeux	2
1.2	Objectifs du Projet	2
1.3	Périmètre Fonctionnel	3
1.4	Structure de ce Document	3
1.5	Technologies Utilisées	4
2	Vue d'Ensemble du Système	5
2.1	Architecture Globale	5
2.2	Flux de Données Principal	5
2.3	Composants Principaux	6
2.3.1	Frontend (Next.js)	6
2.3.2	Backend (FastAPI)	6
2.3.3	Base de Données	7
2.4	Rôles Utilisateurs	7
2.5	Points d'Intégration	8
2.5.1	Supabase	8
2.5.2	Ollama	8
3	Architecture Frontend	9
3.1	Vue d'Ensemble	9
3.2	Structure du Projet	9
3.2.1	Organisation des Dossiers	9
3.2.2	Pages du Dashboard	10
3.3	Système de Layout	10
3.3.1	Layout Principal	10
3.3.2	Composants de Navigation	11
3.4	Client API GMAO	11
3.5	Composants UI Réutilisables	12
3.5.1	Data Tables	12
3.5.2	Formulaires	13
3.5.3	Graphiques	13

3.6	Gestion des Thèmes	13
3.7	Internationalisation	13
3.8	Performance Frontend	14
4	Authentification et Autorisation	15
4.1	Architecture de Sécurité	15
4.2	Intégration Makerkit	15
4.2.1	Configuration Supabase	15
4.2.2	Client Supabase Frontend	16
4.3	Modèle de Données Utilisateur	16
4.3.1	Structure AuthUser	16
4.3.2	Énumération des Rôles	17
4.4	Vérification JWT	17
4.4.1	Extraction du Token	17
4.4.2	Extraction du Rôle	18
4.5	Contrôle d'Accès (RBAC)	18
4.5.1	Guards de Rôle	18
4.5.2	Application aux Routes	19
4.6	Matrice des Permissions	20
4.7	Sécurité des Sessions	21
4.8	Gestion des Erreurs d'Authentification	21
5	Architecture Backend	22
5.1	Vue d'Ensemble	22
5.2	Structure du Projet	22
5.3	Point d'Entrée Application	23
5.3.1	Initialisation FastAPI	23
5.3.2	Enregistrement des Routers	24
5.4	Couche Routers	25
5.4.1	Pattern de Router	25
5.5	Couche Services	26
5.5.1	KPI Service	26
5.5.2	Pattern Singleton	28
5.6	Gestion des Erreurs	28
5.6.1	Handler Global	28
5.6.2	HTTPException Standards	29
5.7	Endpoints de Santé	29
5.8	Documentation API	30
6	Modèle de Données	31

6.1	Vue d'Ensemble	31
6.2	Schéma Entités-Relations	31
6.3	Entités Principales	31
6.3.1	Equipment	32
6.3.2	Intervention	33
6.3.3	SparePart	34
6.3.4	Technician	35
6.4	Tables d'Association	36
6.4.1	InterventionPart	36
6.4.2	TechnicianAssignment	36
6.5	Modèles AMDEC	37
6.5.1	FailureMode	37
6.5.2	RPNAnalysis	37
6.6	Modèles RAG	38
6.7	Schémas Pydantic	39
7	Système d'Interventions	41
7.1	Vue d'Ensemble	41
7.2	Cycle de Vie d'une Intervention	41
7.3	Création d'une Intervention	41
7.3.1	Endpoint API	41
7.3.2	Schéma de Création	42
7.4	Assignation des Techniciens	43
7.4.1	Processus d'Assignation	43
7.5	Gestion des Pièces Utilisées	44
7.6	Clôture d'Intervention	45
7.7	Requêtes et Filtrage	46
7.8	Statistiques par Intervention	47
8	Système de Soumission Technicien	48
8.1	Vue d'Ensemble	48
8.2	Profil Technicien	48
8.2.1	Données du Profil	48
8.2.2	Gestion des Compétences	48
8.3	Workflow de Soumission	49
8.4	API Technicien	49
8.4.1	Liste des Assignations	49
8.4.2	Soumission d'Heures	50
8.5	Rapport d'Intervention	51
8.5.1	Schéma de Rapport	51

8.5.2 Soumission du Rapport	52
8.6 Statistiques Technicien	53
8.7 Notifications	54
9 KPI et Analytics	55
9.1 Introduction aux KPIs de Maintenance	55
9.2 Indicateurs Principaux	55
9.2.1 MTBF – Mean Time Between Failures	55
9.2.2 MTTR – Mean Time To Repair	57
9.2.3 Disponibilité	58
9.3 Dashboard KPIs	59
9.3.1 Endpoint Consolidé	59
9.3.2 Structure de Réponse	60
9.4 Distributions et Tendances	61
9.4.1 Distribution des Pannes	61
9.4.2 Répartition des Coûts	62
9.5 Visualisation Frontend	63
10 Intelligence Artificielle	64
10.1 Vue d'Ensemble	64
10.2 Système RAG	64
10.2.1 Présentation	64
10.2.2 Architecture RAG	64
10.2.3 Pipeline de Traitement	65
10.2.4 Requête RAG	66
10.3 Maintenance Copilot	68
10.3.1 Fonctionnalités	68
10.3.2 Détection d'Intent	68
10.3.3 Génération de Réponse	69
10.4 OCR Vision	70
10.4.1 Présentation	70
10.4.2 Modes d'Extraction	70
10.5 AI Forecast (Prédiction)	71
10.5.1 Objectifs	71
10.5.2 Algorithmes Utilisés	71
10.5.3 Prédiction RUL	72
10.5.4 Prévision MTBF avec Prophet	73
10.6 Intégration Ollama	74
11 Sécurité	75

11.1 Vue d'Ensemble	75
11.2 Couches de Sécurité	75
11.3 Authentification	75
11.3.1 JWT (JSON Web Tokens)	75
11.3.2 Validation du Token	76
11.4 Autorisation (RBAC)	76
11.4.1 Hiérarchie des Rôles	77
11.4.2 Implémentation des Guards	77
11.5 Validation des Entrées	77
11.5.1 Protection contre les Injections	77
11.5.2 Protection SQL Injection	78
11.6 Sécurité des Communications	78
11.6.1 CORS (Cross-Origin Resource Sharing)	79
11.6.2 Headers de Sécurité	79
11.7 Protection des Données	80
11.7.1 Données Sensibles	80
11.7.2 Variables d'Environnement	80
11.8 Audit et Logging	81
11.8.1 Logs de Sécurité	81
11.9 Checklist de Sécurité	81
12 UX et Rôles Utilisateurs	82
12.1 Principes de Design	82
12.2 Parcours Utilisateurs par Rôle	82
12.2.1 Administrateur	82
12.2.2 Superviseur	82
12.2.3 Technicien	83
12.2.4 Viewer	83
12.3 Structure de Navigation	83
12.3.1 Menu Principal	84
12.4 Composants d'Interface	84
12.4.1 Dashboard Principal	84
12.4.2 Tableaux de Données	84
12.4.3 Formulaires	85
12.5 Responsive Design	85
12.5.1 Points de Rupture	85
12.5.2 Mode Sidebar vs Header	85
12.6 Thèmes et Personnalisation	85
12.6.1 Mode Clair/Sombre	85

12.6.2 Variables CSS	86
12.7 Accessibilité	86
12.7.1 Conformité WCAG	86
12.7.2 Bonnes Pratiques	86
12.8 Widget d'Aide IA	87
13 Performances et Scalabilité	88
13.1 Objectifs de Performance	88
13.2 Optimisations Backend	88
13.2.1 Indexation Base de Données	88
13.2.2 Pagination	89
13.2.3 Caching Redis	89
13.2.4 Async I/O	90
13.3 Optimisations Frontend	91
13.3.1 Server Components	91
13.3.2 React Query Caching	91
13.3.3 Code Splitting	92
13.4 Scalabilité	92
13.4.1 Architecture Horizontale	92
13.4.2 Considérations Docker	92
13.5 Monitoring	93
13.5.1 Health Checks	93
13.5.2 Métriques Recommandées	94
14 Tests et Validation	95
14.1 Stratégie de Test	95
14.2 Tests Backend	95
14.2.1 Structure des Tests	95
14.2.2 Fixtures Pytest	95
14.2.3 Tests Unitaires	97
14.2.4 Tests d'Intégration API	98
14.3 Tests Frontend	99
14.3.1 Tests E2E avec Playwright	99
14.4 Validation des Données	101
14.4.1 Tests de Validation Pydantic	101
14.5 Couverture de Code	102
14.6 CI/CD Testing	103
15 Déploiement	104
15.1 Vue d'Ensemble	104

15.2 Déploiement Local (Développement)	104
15.2.1 Prérequis	104
15.2.2 Lancement avec Docker Compose	104
15.2.3 Commandes de Lancement	105
15.3 Configuration Supabase	105
15.3.1 Variables d'Environnement	106
15.3.2 Configuration Roles Supabase	106
15.4 Déploiement Production	106
15.4.1 Architecture Recommandée	107
15.4.2 Déploiement Frontend (Vercel)	107
15.4.3 Déploiement Backend (Docker)	107
15.5 Migrations de Base de Données	108
15.5.1 Alembic	108
15.5.2 Initialisation des Données	108
15.6 Checklist de Déploiement	109
16 Limites et Contraintes	110
16.1 Limites Techniques	110
16.1.1 Dépendance à Ollama	110
16.1.2 Scalabilité du RAG	110
16.1.3 Base de Données	111
16.2 Limites Fonctionnelles	111
16.2.1 Gestion des Permissions	111
16.2.2 Workflows	111
16.2.3 Reporting	111
16.3 Limites de l'Intelligence Artificielle	111
16.3.1 Qualité des Prédictions	111
16.3.2 Limitations LLM	111
16.3.3 OCR	112
16.4 Contraintes d'Intégration	112
16.4.1 Systèmes Externes	112
16.4.2 Authentification	112
16.5 Contraintes Opérationnelles	112
16.6 Limitations Connues	113
16.6.1 Bugs Connus	113
16.6.2 Améliorations Nécessaires	113
17 Perspectives d'Évolution	114
17.1 Feuille de Route	114
17.2 Version 2.1 – Améliorations Court Terme	114

17.2.1 Intelligence Artificielle	114
17.2.2 Performance	115
17.3 Version 2.2 – Intégrations	115
17.3.1 Connecteurs Externes	115
17.3.2 Notifications	115
17.3.3 API Avancée	115
17.4 Version 3.0 – Enterprise	116
17.4.1 Multi-tenancy	116
17.4.2 Sécurité Enterprise	116
17.4.3 Fonctionnalités Avancées	116
17.5 Évolutions Techniques	117
17.5.1 Architecture	117
17.5.2 Intelligence Artificielle	117
17.6 Contribution	117
17.7 Conclusion	117

Chapitre 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 prédition 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

Chapitre 2

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.

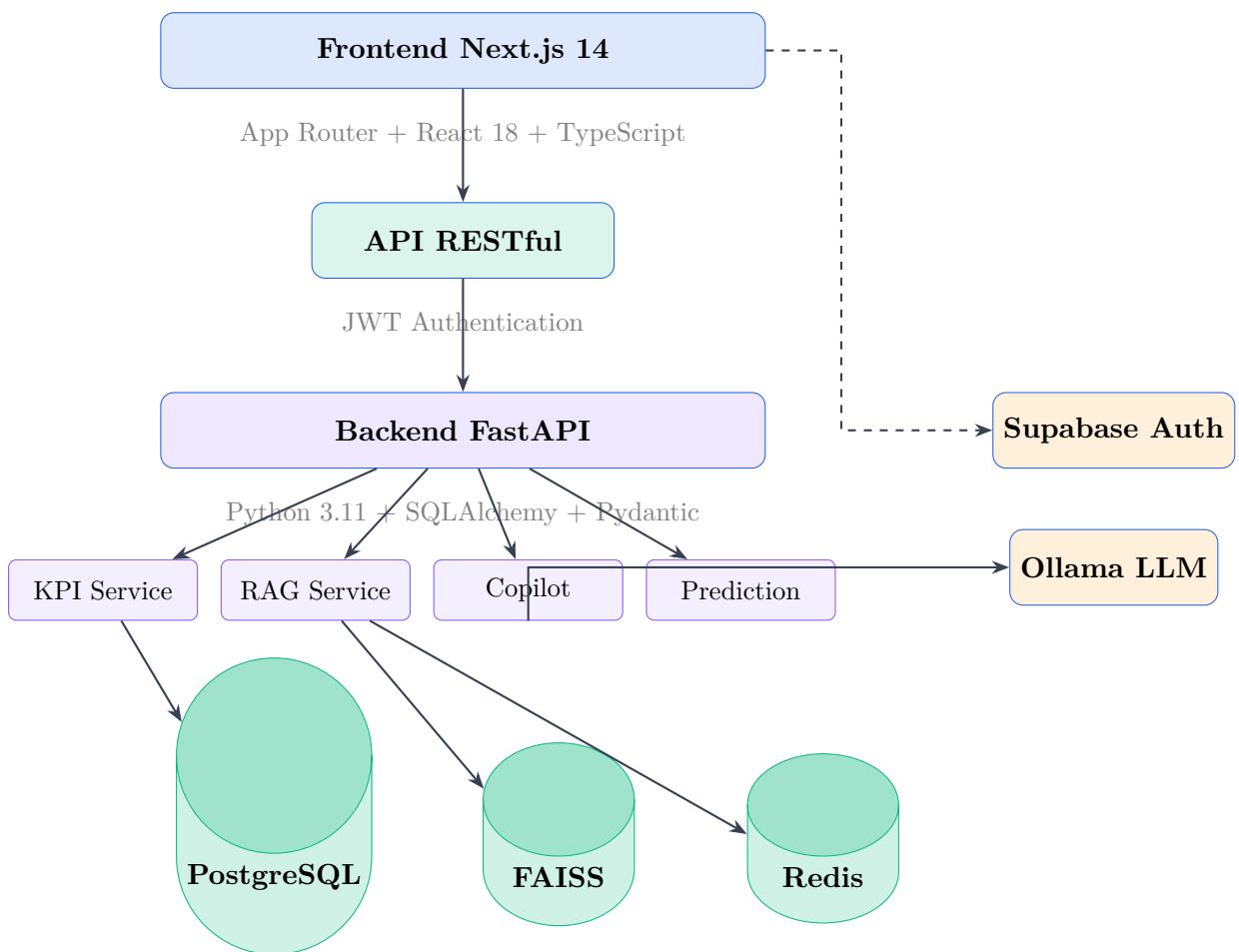


FIGURE 2.1 – Architecture globale du système ProAct

2.2 Flux de Données Principal

Le système gère plusieurs flux de données interconnectés :

- 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 à chaque module fonctionnel :

- `equipment/` – Gestion des équipements
- `interventions/` – Ordres de travail
- `spare-parts/` – Pièces de rechange
- `technicians/` – Gestion des techniciens
- `kpi/` – Tableaux de bord KPI
- `amdec/` – Analyse AMDEC/RPN
- `assistant/` – Chat RAG
- `copilot/` – Assistant Copilot
- `ocr/` – Extraction OCR
- `ai-forecast/` – Prédictions IA

2.3.2 Backend (FastAPI)

Le backend expose une API RESTful complète avec les caractéristiques suivantes :

TABLE 2.1 – Endpoints API par module

Préfixe	Tag	Définition
/api/equipment	Equipment	CRUD équipements, surveillance
/api/interventions	Interventions	Ordres de travail, assignation
/api/spare-parts	Spare Parts	Inventaire pièces de rechange
/api/technicians	Technicians	Profils, compétences
/api/kpi	KPIs & Analytics	MTBF, MTTR, Diagnostic
/api/amdec	AMDEC & RPN	Modes de défaillance
/api/rag	RAG System	Upload document
/api/copilot	Maintenance Copilot	Assistant conseil
/api/ocr	OCR Vision AI	Extraction textuelle
/api/predict	AI Forecast	Prédiction Résultats

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)
- **Skill** : Compétences techniques
- **RAGDocument** : Documents indexés pour recherche vectorielle

2.4 Rôles Utilisateurs

Le système implémente un modèle RBAC (Role-Based Access Control) avec quatre niveaux :

TABLE 2.2 – Matrice des rôles et permissions

Fonctionnalité	Admin	Supervisor	Technician	Viewer
Créer équipements	✓	✓	—	—
Modifier interventions	✓	✓	✓	—
Consulter KPIs	✓	✓	✓	✓
Gérer techniciens	✓	—	—	—
Import/Export données	✓	✓	—	—
Configurer AMDEC	✓	✓	—	—
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 :

- Génération de réponses RAG
- Traitement des requêtes Copilot
- Extraction OCR avec modèles vision (qwen2.5vl)

Configuration Requise

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

Chapitre 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
- config/ : Configuration navigation et thème

3.2.2 Pages du Dashboard

Le dashboard (app/home/) contient 15 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
ai-forecast/	Prédictions maintenance
priority-training/	Priorités de formation
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 et la configuration globale.

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 (champs requis, longueurs min/max, formats) qui génèrent automatiquement les messages d'erreur.

3.5.3 Graphiques

Les visualisations sont réalisées avec **Recharts** :

- **BarChart** : Distribution des pannes, coûts par période
- **LineChart** : Évolution temporelle des KPIs
- **PieChart** : Répartition des statuts d'équipement
- **AreaChart** : Tendances MTBF/MTTR avec zones de confiance

3.6 Gestion des Thèmes

Le système supporte les modes clair et sombre via l'attribut `data-theme` sur l'élément HTML racine. Les variables CSS sont définies pour chaque thème, permettant une personnalisation complète. La préférence utilisateur est persistée dans un cookie.

3.7 Internationalisation

L'application utilise `next-intl` pour le support multilingue. Les fichiers de traduction sont stockés dans `locales/` et le HOC `withI18n` encapsule les layouts pour fournir le contexte de traduction.

3.8 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 automatique
- **Code Splitting** : Routes dynamiques chargées à la demande
- **React Query Caching** : Mise en cache des requêtes API avec stale-while-revalidate

Chapitre 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.

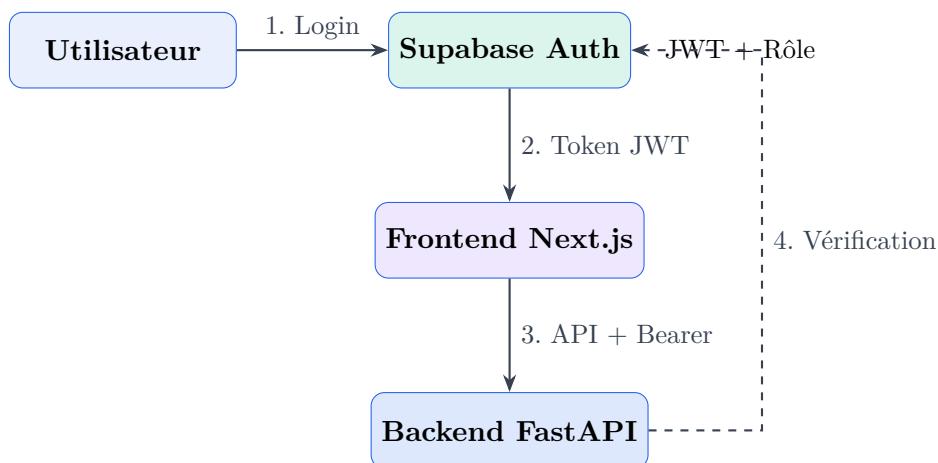


FIGURE 4.1 – Flux d'authentification

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 (sign-in, sign-up, password reset)
- Middleware de protection des routes
- Gestion des sessions avec refresh automatique

La configuration Supabase est définie dans `supabase/config.toml` avec les URLs de redirection et les paramètres d'email.

4.3 Modèle de Données Utilisateur

4.3.1 Structure AuthUser

Le backend définit une dataclass `AuthUser` représentant l'utilisateur authentifié avec les propriétés suivantes :

- `id` : Identifiant Supabase unique (claim `sub`)
- `email` : Adresse email de l'utilisateur
- `role` : Rôle extrait de `app_metadata` (Admin, Supervisor, Technician, Viewer)
- `raw_claims` : Claims JWT bruts pour accès aux métadonnées

Des propriétés calculées (`is_admin`, `is_supervisor`, etc.) facilitent les vérifications de rôle dans le code.

4.3.2 Énumération des Rôles

Le système définit quatre rôles hiérarchiques via l'enum `UserRole` :

TABLE 4.1 – Hiérarchie des rôles

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

4.4 Vérification JWT

Le processus de vérification du token JWT 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 (`sub`, `email`)
4. Extraction du rôle depuis `app_metadata.role` avec fallbacks

En cas d'échec, une erreur HTTP 401 (Unauthorized) est retournée avec le message approprié.

4.5 Contrôle d'Accès (RBAC)

4.5.1 Guards de Rôle

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

- `require_admin()` : Admin uniquement
- `require_supervisor_or_admin()` : Supervisors et Admins
- `require_technician_or_above()` : Tous sauf Viewers

Ces guards sont appliqués aux routes via le paramètre `dependencies` de FastAPI.

4.5.2 Application aux Routes

Chaque router définit ses permissions au niveau endpoint. Par exemple, la création d'équipement nécessite le rôle Supervisor ou Admin, tandis que la consultation est ouverte à tous les utilisateurs authentifiés.

4.6 Matrice des Permissions

TABLE 4.2 – Matrice détaillée des permissions par endpoint

Endpoint	Admin	Supervisor	Technician	Viewer
<i>Équipements</i>				
GET /equipment	✓	✓	✓	✓
POST /equipment	✓	✓	—	—
PUT /equipment/ :id	✓	✓	—	—
DELETE /equipment/ :id	✓	—	—	—
<i>Interventions</i>				
GET /interventions	✓	✓	✓	✓
POST /interventions	✓	✓	✓	—
PUT /interventions/ :id	✓	✓	✓	—
<i>Intelligence Artificielle</i>				
POST /rag/query	✓	✓	✓	✓
POST /copilot/query	✓	✓	✓	✓
POST /ocr/extract	✓	✓	—	—

4.7 Sécurité des Sessions

Bonnes Pratiques Implémentées

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

4.8 Gestion des Erreurs d'Authentification

TABLE 4.3 – Codes d'erreur d'authentification

Code HTTP	Situation	Action
401 Unauthorized	Token manquant ou invalide	Redirection vers login
403 Forbidden	Rôle insuffisant	Affichage message d'erreur
500 Server Error	JWT_SECRET non configuré	Alerte administrateur

Chapitre 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** avec une séparation claire des responsabilités.

Stack Backend

- **Framework** : FastAPI 0.100+
- **ORM** : SQLAlchemy 2.0 (async-compatible)
- **Validation** : Pydantic v2
- **Authentification** : python-jose (JWT)
- **ML** : Scikit-learn, XGBoost, Prophet
- **LLM** : Ollama (via ollama-python)
- **Vector Store** : FAISS
- **Cache** : Redis

5.2 Structure du Projet

Le backend suit une organisation modulaire :

- app/main.py : Point d'entrée FastAPI, configuration CORS, enregistrement des routers
- app/database.py : Configuration SQLAlchemy, session factory
- app/models.py : Modèles ORM (Equipment, Intervention, etc.)
- app/schemas.py : Schémas Pydantic pour validation
- app/security.py : Authentification JWT et guards RBAC
- app/routers/ : 15 routers pour chaque domaine fonctionnel
- app/services/ : Logique métier (KPI, Copilot, Prediction, RAG)
- alembic/ : Migrations de base de données
- tests/ : Tests unitaires et d'intégration

5.3 Point d'Entrée Application

5.3.1 Initialisation FastAPI

L'application FastAPI est configurée dans `main.py` avec :

- **Lifespan manager** : Initialisation de la base de données et du système RAG au démarrage, nettoyage à l'arrêt
- **Middleware CORS** : Configuration des origines autorisées pour les requêtes cross-origin
- **Exception handler global** : Capture et logging des erreurs non gérées
- **Métadonnées OpenAPI** : Titre, description, version, contact

5.3.2 Enregistrement des Routers

Chaque router est enregistré avec :

- Un préfixe d'URL (`/api/equipment`, `/api/kpi`, etc.)
- Un tag pour la documentation Swagger
- Une dépendance d'authentification (`get_auth_user`)

5.4 Couche Routers

Les routers définissent les endpoints REST et délèguent la logique aux services. Chaque router suit un pattern standardisé :

1. Définition des endpoints CRUD (GET, POST, PUT, DELETE)
2. Injection des dépendances (session DB, utilisateur authentifié)
3. Validation automatique via schémas Pydantic
4. Retour de réponses typées avec codes HTTP appropriés

TABLE 5.1 – Routers principaux

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

5.5 Couche Services

Les services encapsulent la logique métier complexe et sont instanciés comme singletons globaux. Les principaux services sont :

- **KPIService** : Calculs MTBF, MTTR, Disponibilité avec filtres date/équipement
- **CopilotService** : Orchestration des requêtes IA avec détection d'intent
- **PredictionService** : Modèles ML pour RUL et prévision MTBF
- **OCRService** : Extraction de texte via modèles Vision-Language
- **RAGService** : Pipeline complet de recherche documentaire augmentée

5.6 Gestion des Erreurs

Le backend implémente une gestion d'erreurs à plusieurs niveaux :

TABLE 5.2 – Codes d’erreur HTTP standards

Code	Signification	Exemple
400 Bad Request	Données invalides	Validation Pydantic échouée
401 Unauthorized	Token manquant/invalidé	JWT expiré
403 Forbidden	Permissions insuffisantes	Rôle non autorisé
404 Not Found	Ressource inexistante	Équipement ID inconnu
422 Unprocessable Entity	Erreur de validation	Champ requis manquant
500 Server Error	Erreur interne	Exception non gérée

Un handler global capture toutes les exceptions non gérées, les logue avec traceback complet, et retourne une réponse JSON standardisée.

5.7 Endpoints de Santé

Le backend expose des endpoints de monitoring :

- `GET /health` : Vérification de la connexion DB et du système RAG
- `GET /api/stats` : Compteurs des entités principales
- `GET /` : Informations sur la version et les fonctionnalités

5.8 Documentation API

FastAPI génère automatiquement la documentation interactive :

- **Swagger UI** : `/docs` – Interface interactive pour tester les endpoints
- **ReDoc** : `/redoc` – Documentation lisible avec navigation
- **OpenAPI Schema** : `/openapi.json` – Spécification JSON pour génération de clients

Tags API

Les endpoints sont organisés par tags pour une navigation claire : Equipment, Interventions, Spare Parts, Technicians, KPIs & Analytics, AMDEC & RPN, RAG System, Maintenance Copilot, OCR Vision AI, AI Forecast.

Chapitre 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 via Supabase) et SQLite (développement local).

6.2 Schéma Entités-Relations

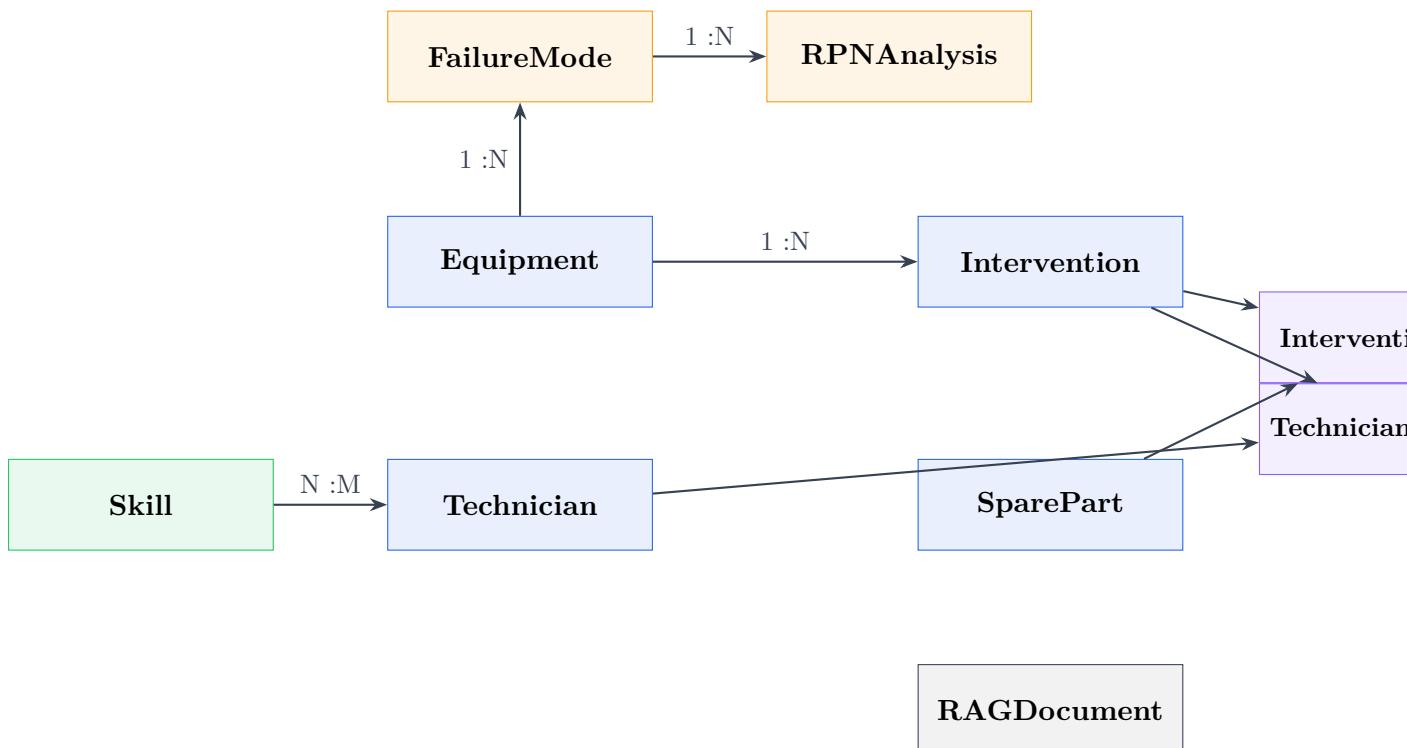


FIGURE 6.1 – Diagramme entités-relations simplifié

6.3 Entités Principales

6.3.1 Equipment

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

TABLE 6.1 – Attributs de l'entité Equipment

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

Les relations incluent une liste d'interventions et de modes de défaillance associés.

6.3.2 Intervention

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

- **Classification** : Type de panne, catégorie, cause racine
- **Timing** : Date de demande, date de réalisation, durée d'indisponibilité (heures)
- **Coûts** : Coût matériel, coût main d'œuvre, coût total
- **Statut** : Open, In Progress, Completed, Closed, Cancelled
- **Relations** : Équipement associé, pièces utilisées, techniciens assignés

Des index sont créés sur les colonnes fréquemment filtrées (date_intervention, equipment_id, status).

6.3.3 SparePart

L'entité **SparePart** gère l'inventaire des pièces de rechange :

- Désignation et référence (unique)
- Coût unitaire et stock actuel
- Seuil d'alerte pour réapprovisionnement
- Fournisseur et délai de livraison

6.3.4 Technician

L'entité **Technician** représente le personnel de maintenance :

- Informations personnelles : Nom, prénom, email (unique), téléphone
- Informations professionnelles : Spécialité, taux horaire, date d'embauche
- Statut : Active, Inactive, On Leave
- Relations : Assignations d'interventions, compétences acquises

6.4 Tables d'Association

6.4.1 InterventionPart

Association N :M entre interventions et pièces de rechange. Stocke la quantité utilisée et le coût unitaire au moment de l'utilisation (pour historique des prix).

6.4.2 TechnicianAssignment

Association N :M entre interventions et techniciens. Capture les heures travaillées, le taux horaire appliqué, et le coût de main d'œuvre calculé.

6.4.3 TechnicianSkill

Association N :M entre techniciens et compétences avec niveau de maîtrise (1-5), date d'acquisition, et informations de certification.

6.5 Modèles AMDEC

6.5.1 FailureMode

Représente un mode de défaillance potentiel pour un équipement :

- Mode de défaillance (description)
- Effet de la défaillance
- Cause potentielle
- Lien vers l'équipement concerné

6.5.2 RPNAAnalysis

Stocke les évaluations RPN (Risk Priority Number) avec les scores :

- **Severity (G)** : Gravité de l'effet (1-10)
- **Occurrence (O)** : Fréquence d'occurrence (1-10)

- **Detection (D)** : Capacité de détection (1-10)
- **RPN** : Produit $G \times O \times D$ (1-1000)

6.6 Modèles RAG

- **RAGDocument** : Métadonnées des documents indexés (filename, status, chunk count)
- **RAGDocumentChunk** : Segments de texte avec index et métadonnées pour recherche vectorielle
- **RAGQuery** : Historique des requêtes avec métriques de performance
- **RAGIndexMetadata** : Statistiques et versions de l'index FAISS

6.7 Schémas Pydantic

Les schémas Pydantic assurent la validation des données entrantes et sortantes :

- ***Base** : Attributs communs pour création et mise à jour
- ***Create** : Schéma de création (hérite de Base)
- ***Update** : Schéma de mise à jour (tous champs optionnels)
- ***Response** : Schéma de réponse avec id et timestamps
- ***WithStats** : Schéma enrichi avec statistiques calculées

Validation Automatique

Pydantic v2 génère automatiquement la validation des types, les contraintes de champs (min/max length, ge/le), et la documentation OpenAPI. Les validators personnalisés permettent des règles métier complexes (ex : date d'intervention non future).

Chapitre 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, de la demande initiale à la clôture.

Définition - Intervention

Une **intervention** représente une action de maintenance effectuée sur un équipement. Elle peut être :

- **Corrective** : Suite à une panne ou défaillance
- **Préventive** : Maintenance planifiée
- **Améliorative** : Upgrade ou optimisation

7.2 Cycle de Vie d'une Intervention

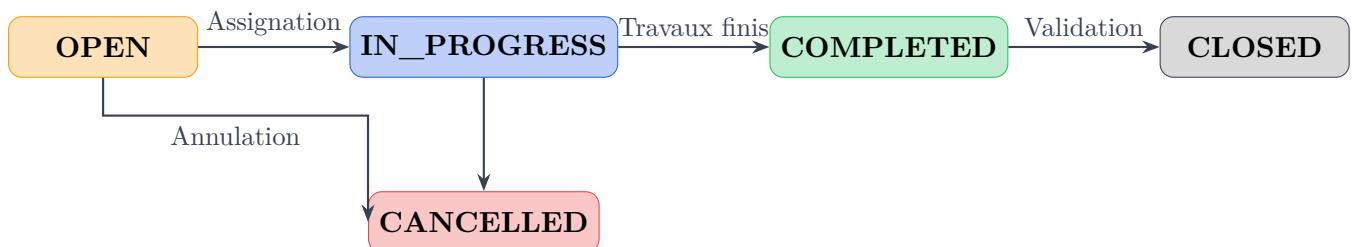


FIGURE 7.1 – États du cycle de vie d'une intervention

7.3 Crédit d'une Intervention

La création d'une intervention s'effectue via l'endpoint `POST /api/interventions/` avec les informations suivantes :

- **equipment_id** (requis) : Identifiant de l'équipement concerné
- **type_panne** : Classification du type de panne
- **categorie_panne** : Catégorie (électrique, mécanique, etc.)

- **cause** : Description de la cause identifiée
- **date_demande** : Date de la demande d'intervention

Le système vérifie l'existence de l'équipement, initialise le statut à **OPEN**, et retourne l'intervention créée avec son identifiant.

7.4 Assignation des Techniciens

7.4.1 Processus d'Assignation

1. Sélection de l'intervention à assigner
2. Choix du ou des techniciens disponibles selon leurs compétences
3. Attribution des heures estimées
4. Passage automatique au statut **IN_PROGRESS**

L'assignation crée un enregistrement **TechnicianAssignment** qui capture :

- Le lien intervention-technicien
- Les heures estimées/travaillées
- Le taux horaire au moment de l'assignation (pour historique)

7.5 Gestion des Pièces Utilisées

L'ajout de pièces à une intervention via **POST /api/interventions/{id}/parts** :

1. Vérifie la disponibilité en stock
2. Crée l'association **InterventionPart** avec quantité et coût unitaire
3. Décrémente automatiquement le stock de la pièce
4. Recalcule le coût matériel total de l'intervention

Gestion du Stock

Le système vérifie que le stock actuel est suffisant avant de permettre l'utilisation. En cas de stock insuffisant, une erreur 400 est retournée.

7.6 Clôture d'Intervention

La clôture d'une intervention (**PUT /api/interventions/{id}/complete**) :

1. Passe le statut à **COMPLETED**
2. Enregistre la date effective d'intervention
3. Capture la durée d'indisponibilité finale
4. Calcule le coût main d'œuvre à partir des assignations

-
5. Met à jour le coût total (matériel + main d'œuvre)

7.7 Requêtes et Filtrage

L'endpoint `GET /api/interventions/` supporte les filtres suivants :

TABLE 7.1 – Paramètres de filtrage des interventions

Paramètre	Type	Description
skip	int	Offset de pagination
limit	int	Nombre max de résultats (1-500)
status	enum	Filtrer par statut
equipment_id	int	Filtrer par équipement
start_date	date	Date de début de période
end_date	date	Date de fin de période
type_panne	string	Recherche par type (LIKE)

Les résultats sont triés par date d'intervention décroissante par défaut.

7.8 Données Capturées

TABLE 7.2 – Données capturées par intervention

Donnée	Type	Utilisation
Durée d'indisponibilité	Float (heures)	Calcul MTTR, Disponibilité
Coût matériel	Float (€)	Analyse coûts
Coût main d'œuvre	Float (€)	Analyse coûts
Type de panne	String	Distribution pannes
Date intervention	Date	Tendances temporelles
Techniciens assignés	List	Charge de travail
Pièces utilisées	List	Consommation stock

Traçabilité Complète

Chaque intervention maintient un historique complet incluant horodatage de création/modification, transitions de statut, et toutes les assignations et utilisations de pièces.

Chapitre 8

Système de Soumission Technicien

8.1 Vue d'Ensemble

Le module de soumission permet aux techniciens de terrain de reporter leurs activités, documenter les interventions réalisées et soumettre leurs heures de travail.

8.2 Profil Technicien

8.2.1 Données du Profil

Chaque technicien dispose d'un profil complet incluant :

TABLE 8.1 – Attributs du profil technicien

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

8.2.2 Gestion des Compétences

Le système permet d'associer des compétences techniques aux techniciens via la table [TechnicianSkill](#). Chaque association capture :

- Le niveau de maîtrise (1-5)
- La date d'acquisition
- Le statut de certification (certifié/non certifié)
- La date d'expiration de certification si applicable

8.3 Workflow de Soumission

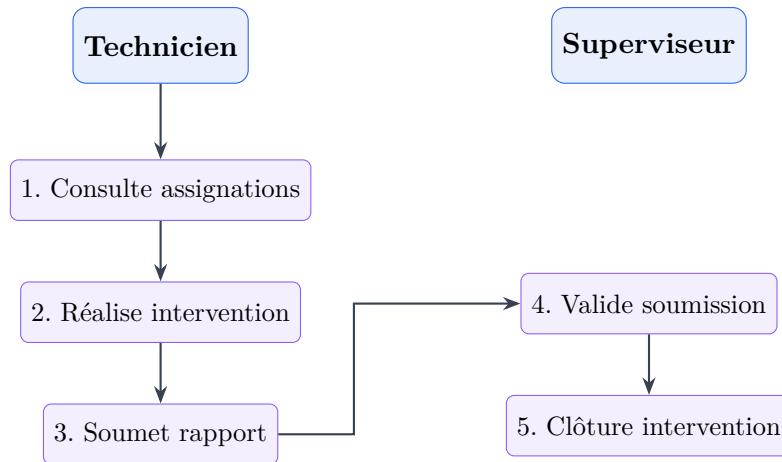


FIGURE 8.1 – Workflow de soumission technicien

8.4 Fonctionnalités API Technicien

8.4.1 Mes Assignations

L'endpoint `GET /api/technicians/me/assignments` retourne la liste des interventions assignées au technicien connecté. Le système identifie le technicien via son email (claim JWT) et filtre les assignations actives (statuts OPEN ou IN_PROGRESS).

8.4.2 Soumission d'Heures

L'endpoint `PUT /api/assignments/{id}/submit-hours` permet au technicien de rapporter ses heures travaillées. Le système :

1. Vérifie que l'assignation appartient au technicien connecté
2. Met à jour les heures travaillées
3. Calcule le coût de main d'œuvre (heures × taux horaire)
4. Recalcule le coût total de l'intervention

8.5 Rapport d'Intervention

8.5.1 Contenu du Rapport

Un rapport d'intervention complet contient :

- **Description des travaux** : Détail des actions réalisées (min. 10 caractères)
- **Cause identifiée** : Diagnostic de la cause racine

- **Actions correctives** : Mesures prises pour résoudre le problème
- **Pièces utilisées** : Liste des pièces avec quantités
- **Heures travaillées** : Temps passé sur l'intervention
- **Date de réalisation** : Date effective des travaux
- **Observations** : Notes complémentaires

8.5.2 Traitement du Rapport

La soumission du rapport via `POST /api/interventions/{id}/submit-report` :

1. Enregistre les informations du rapport
2. Ajoute les pièces utilisées (avec vérification de stock)
3. Recalcule les coûts (matériel + main d'œuvre)
4. Passe l'intervention au statut COMPLETED

8.6 Statistiques Technicien

L'endpoint `GET /api/technicians/{id}/stats` fournit des statistiques de performance :

TABLE 8.2 – Statistiques disponibles par technicien

Métrique	Description
Total interventions	Nombre d'interventions réalisées
Total heures	Cumul des heures travaillées
Coût main d'œuvre	Somme des coûts MO générés
Moyenne heures/intervention	Indicateur d'efficacité
Nombre de compétences	Polyvalence du technicien

Ces statistiques peuvent être filtrées par période (`start_date, end_date`).

8.7 Notifications

Le système peut notifier les techniciens pour :

- Nouvelle assignation reçue
- Rappel d'intervention planifiée
- Demande de rapport en attente
- Validation ou rejet de soumission

Intégration Future

L'intégration avec des services de notification (email, SMS, push mobile) est prévue pour les versions futures du système.

Chapitre 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 :

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

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 :

$$MTTR = \frac{\sum_{i=1}^n D_i}{n} \quad (9.2)$$

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 :

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

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.

Chapitre 10

Intelligence Artificielle

10.1 Vue d'Ensemble

ProAct intègre plusieurs modules d’Intelligence Artificielle pour augmenter les capacités analytiques et décisionnelles du système GMAO. Ces modules s’appuient sur des modèles de langage (LLM), des techniques de recherche vectorielle et des algorithmes de Machine Learning.

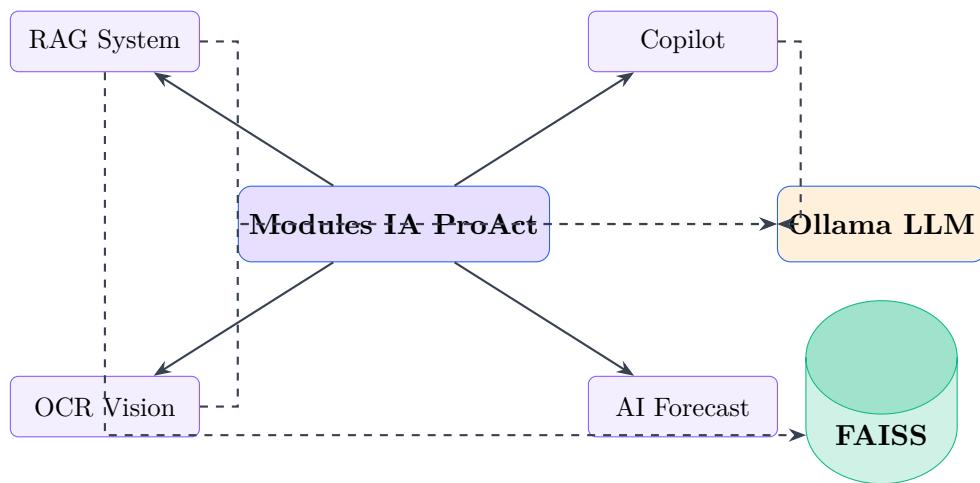


FIGURE 10.1 – Architecture des modules IA

10.2 Système RAG

10.2.1 Présentation

Le système **RAG** (Retrieval-Augmented Generation) permet d’interroger une base documentaire technique en langage naturel. Il combine trois étapes :

1. **Retrieval** : Recherche vectorielle de passages pertinents dans les documents indexés
2. **Augmentation** : Enrichissement du contexte LLM avec les passages trouvés
3. **Generation** : Production de réponses contextualisées par le LLM

10.2.2 Architecture RAG

Le sous-système RAG est organisé en modules spécialisés :

- **RAGService** : Orchestrateur principal du pipeline
- **DocumentProcessor** : Parsing (PDF, TXT, DOCX) et découpage en chunks
- **EmbeddingService** : Génération des vecteurs d'embedding via Ollama
- **VectorStore** : Index FAISS pour recherche de similarité
- **LLMService** : Interface avec le modèle de langage
- **CacheService** : Mise en cache Redis des réponses

10.2.3 Pipeline de Traitement

Upload de document :

1. Création de l'enregistrement document en base
2. Extraction et découpage du texte en chunks (500-1000 tokens)
3. Génération des embeddings pour chaque chunk
4. Stockage dans l'index vectoriel FAISS
5. Mise à jour du statut à INDEXED

Requête RAG :

1. Vérification du cache (hash de la requête)
2. Génération de l'embedding de la requête
3. Recherche des K chunks les plus similaires (cosine similarity)
4. Construction du prompt avec contexte
5. Génération de la réponse par le LLM
6. Mise en cache du résultat

10.3 Maintenance Copilot

10.3.1 Fonctionnalités

Le **Copilot** est un assistant conversationnel spécialisé en maintenance qui peut :

- Expliquer les KPIs de maintenance (MTBF, MTTR, etc.)
- Générer des résumés de santé équipement
- Produire des rapports d'intervention
- Recommander des actions de maintenance

10.3.2 Détection d'Intent

Le Copilot utilise le LLM pour classifier l'intention de l'utilisateur parmi :

TABLE 10.1 – Intentions supportées par le Copilot

Intent	Description
KPI_EXPLANATION	Questions sur les métriques de performance
EQUIPMENT_HEALTH	État et santé d'un équipement
INTERVENTION_REPORT	Demande de rapport ou résumé
GENERAL_QUESTION	Autres questions maintenance

Selon l'intent détecté, le Copilot route vers le handler approprié qui récupère les données contextuelles (KPIs, interventions, etc.) avant de générer une réponse structurée.

10.4 OCR Vision

10.4.1 Présentation

Le module **OCR** utilise des modèles Vision-Language (VLM) pour extraire du texte structuré à partir d'images de documents techniques.

Configuration OCR

- **Modèle** : qwen2.5vl :3b (configurable via OLLAMA_VISION_MODEL)
- **Provider** : Ollama local
- **Formats d'entrée** : PNG, JPEG, WebP
- **Formats de sortie** : Markdown, HTML, JSON, Texte brut

10.4.2 Modes d'Extraction

- **Markdown** : Préserve la structure (titres, tableaux, listes)
- **HTML sémantique** : Balises h1-h6, p, table, ul/li
- **JSON structuré** : Extraction en objet avec sections et métadonnées
- **Texte brut** : Contenu textuel simple

Les images sont optimisées (redimensionnement à 1024px max) avant envoi au modèle pour réduire les temps de traitement.

10.5 AI Forecast (Prédiction)

10.5.1 Objectifs

Le module de **prédition** fournit :

- **RUL** (Remaining Useful Life) : Estimation des jours restants avant prochaine panne
- **Prévision MTBF** : Tendance future de la fiabilité sur 90 jours
- **Date de prochaine panne** : Estimation probabiliste

10.5.2 Algorithmes Utilisés

TABLE 10.2 – Algorithmes de prédition

Algorithmme	Type	Usage
Random Forest	ML supervisé	Prédiction RUL (baseline)
XGBoost	ML supervisé	Prédiction RUL (amélioration)
Prophet	Séries temporelles	Prévision tendance MTBF

10.5.3 Pipeline de Prédition

Prédiction RUL :

1. Récupération de l'historique d'interventions (minimum 5)
2. Feature engineering (intervalles, moyennes mobiles, tendances)
3. Entraînement des modèles RF et XGBoost
4. Sélection du meilleur modèle (MAE minimum)
5. Prédiction sur les données les plus récentes

Forecast MTBF :

1. Calcul du MTBF glissant (rolling mean sur 3 périodes)
2. Entraînement du modèle Prophet
3. Génération des prédictions avec intervalles de confiance
4. Détection de la tendance (amélioration/dégradation)

10.6 Intégration Ollama

Tous les modules IA utilisent Ollama comme backend LLM. Configuration requise :

TABLE 10.3 – Variables d'environnement Ollama

Variable	Valeur par défaut
OLLAMA_BASE_URL	http://ollama:11434
OLLAMA_MODEL	llama3.2:3b
OLLAMA_VISION_MODEL	qwen2.5v1:3b
OLLAMA_EMBEDDING_MODEL	nomic-embed-text

Prérequis

Les modèles Ollama doivent être téléchargés avant utilisation via les commandes `ollama pull <model>`.

Chapitre 11

Sécurité

11.1 Vue d'Ensemble

La sécurité de ProAct est conçue selon le principe de **défense en profondeur**, avec plusieurs couches de protection allant de l'authentification utilisateur jusqu'à la sécurisation des données.

Authentification – JWT Supabase

Autorisation – RBAC 4 niveaux

Validation – Pydantic Schemas

Transport – HTTPS/TLS

Données – Chiffrement at-rest

FIGURE 11.1 – Couches de sécurité

11.2 Authentification

11.2.1 JWT (JSON Web Tokens)

- **Algorithme** : HS256 (HMAC-SHA256)
- **Secret** : `SUPABASE_JWT_SECRET` (256 bits minimum)
- **Expiration** : Configurable (défaut 1 heure)
- **Refresh** : Gestion automatique par Supabase côté client

11.2.2 Validation du Token

Le processus de validation vérifie :

1. Présence du header Authorization avec schéma Bearer
2. Signature cryptographique avec le secret partagé
3. Date d'expiration (claim `exp`)
4. Présence des claims requis (`sub`, `email`)

Les erreurs sont loggées et retournent des codes HTTP appropriés (401, 403).

11.3 Autorisation (RBAC)

11.3.1 Hiérarchie des Rôles

TABLE 11.1 – Hiérarchie et privilèges des rôles

Rôle	Niveau	Privilèges
<code>Admin</code>	4	Accès total, gestion utilisateurs
<code>Supervisor</code>	3	Gestion équipements, assignations, rapports
<code>Technician</code>	2	Interventions, soumissions, consultation
<code>Viewer</code>	1	Consultation seule (lecture)

11.3.2 Implémentation des Guards

Les guards de rôle sont implémentés via le pattern de dépendance FastAPI. Une factory `require_role()` génère des vérificateurs qui :

1. Extraient l'utilisateur authentifié du token
2. Vérifient l'appartenance à la liste des rôles autorisés
3. Loggent les tentatives d'accès non autorisées
4. Retournent HTTP 403 si le rôle est insuffisant

11.4 Validation des Entrées

11.4.1 Protection contre les Injections

Pydantic valide automatiquement les types et contraintes de tous les champs :

- Types primitifs (str, int, float, bool, date)

- Contraintes de longueur (min_length, max_length)
- Contraintes numériques (ge, le, gt, lt)
- Patterns regex pour formats spécifiques
- Énumérations pour valeurs prédéfinies

11.4.2 Protection SQL Injection

SQLAlchemy ORM génère des requêtes paramétrées automatiquement. Les valeurs utilisateur sont toujours liées comme paramètres, jamais interpolées dans les chaînes SQL.

11.5 Sécurité des Communications

11.5.1 CORS (Cross-Origin Resource Sharing)

La configuration CORS définit :

- **Origins autorisées** : Liste des domaines frontend
- **Méthodes** : GET, POST, PUT, DELETE, OPTIONS
- **Headers** : Authorization, Content-Type
- **Credentials** : Activé pour les cookies

Configuration Production

En production, remplacer `allow_origins=["*"]` par la liste explicite des domaines autorisés.

11.5.2 Headers de Sécurité

Headers recommandés pour le frontend Next.js :

- **X-Frame-Options** : DENY (protection clickjacking)
- **X-Content-Type-Options** : nosniff
- **X-XSS-Protection** : 1; mode=block
- **Strict-Transport-Security** : max-age=31536000

11.6 Protection des Données

11.6.1 Classification des Données Sensibles

TABLE 11.2 – Classification des données sensibles

Donnée	Sensibilité	Protection
Mots de passe	Critique	Hash bcrypt (Supabase)
Tokens JWT	Haute	HttpOnly cookies, expiration
Emails utilisateurs	Moyenne	Accès restreint par rôle
Coûts interventions	Moyenne	Visible selon rôle
Données techniques	Basse	Accès authentifié

11.6.2 Variables d’Environnement

Les secrets sont stockés dans des variables d’environnement, jamais dans le code source. En production, utiliser des gestionnaires de secrets (Vault, AWS Secrets Manager).

11.7 Audit et Logging

Le système logue les événements de sécurité :

- Connexions réussies et échouées
- Tentatives d'accès non autorisées
- Opérations sensibles (suppression, export)

11.8 Checklist de Sécurité

TABLE 11.3 – Checklist de sécurité ProAct

Mesure	Implémenté	Production
Authentification JWT	✓	✓
RBAC 4 niveaux	✓	✓
Validation Pydantic	✓	✓
Protection SQL Injection	✓	✓
Configuration CORS	✓	À configurer
HTTPS/TLS	–	Requis
Rate Limiting	–	Recommandé

Chapitre 12

UX et Rôles Utilisateurs

12.1 Principes de Design

L'interface utilisateur de ProAct est conçue selon les principes suivants :

- **Clarté** : Navigation intuitive et organisation logique
- **Efficacité** : Accès rapide aux fonctionnalités clés
- **Cohérence** : Patterns d'interaction uniformes
- **Adaptabilité** : Interface responsive (desktop, tablette, mobile)
- **Accessibilité** : Conformité WCAG niveau AA

12.2 Parcours Utilisateurs par Rôle

12.2.1 Administrateur

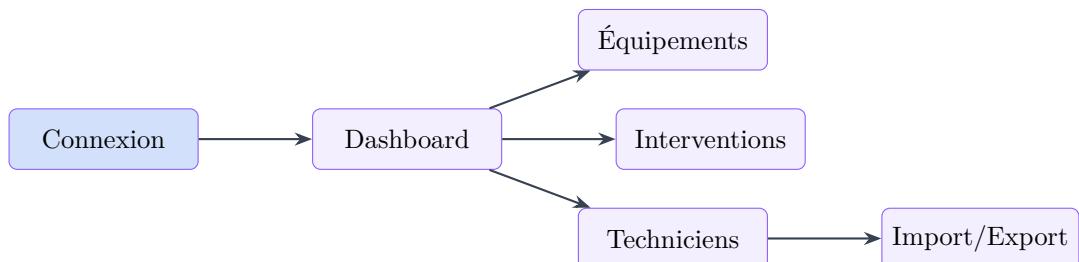


FIGURE 12.1 – Parcours Administrateur

Tâches principales :

1. Configuration du système (équipements, compétences)
2. Gestion des utilisateurs et attribution des rôles
3. Import/export massif de données
4. Analyse des KPIs globaux
5. Configuration AMDEC

12.2.2 Superviseur

Tâches principales :

1. Planification des interventions préventives
2. Assignation des techniciens aux interventions
3. Validation des rapports techniciens
4. Suivi des KPIs d'équipe
5. Gestion du stock de pièces de rechange

12.2.3 Technicien

Tâches principales :

1. Consultation des interventions assignées
2. Réalisation et documentation des interventions
3. Soumission des rapports d'intervention
4. Mise à jour des heures travaillées
5. Consultation de l'assistant IA pour aide au diagnostic

12.2.4 Viewer

Tâches principales :

1. Consultation des tableaux de bord
2. Visualisation des KPIs
3. Consultation de l'historique des interventions
4. Utilisation du RAG en lecture seule

12.3 Structure de Navigation

12.3.1 Menu Principal

TABLE 12.1 – Structure du menu de navigation

Section	Icône	Admin	Super	Tech	View
Dashboard	Home	✓	✓	✓	✓
Équipements	Settings	✓	✓	✓	✓
Interventions	Wrench	✓	✓	✓	✓
Pièces de rechange	Package	✓	✓	—	—
Techniciens	Users	✓	✓	—	—
KPI Dashboard	BarChart	✓	✓	✓	✓
AMDEC/RPN	AlertTriangle	✓	✓	—	—
<i>IA & Analytics</i>					
Assistant RAG	MessageSquare	✓	✓	✓	✓
Copilot	Bot	✓	✓	✓	✓
OCR	ScanLine	✓	✓	—	—
AI Forecast	TrendingUp	✓	✓	—	—
Import/Export	Upload	✓	✓	—	—

12.4 Composants d'Interface

12.4.1 Dashboard Principal

Le dashboard affiche une vue consolidée selon le rôle :

- **Cards KPI** : MTBF, MTTR, Disponibilité avec indicateurs de tendance
- **Graphiques** : Distribution des pannes, historique des interventions
- **Alertes** : Stock bas, interventions en retard
- **Actions rapides** : Liens vers les tâches fréquentes

12.4.2 Tableaux de Données

Fonctionnalités des data tables :

- Tri par colonnes (ascendant/descendant) avec indicateurs visuels
- Pagination configurable (10, 25, 50, 100 éléments)

- Filtrage multi-critères avec autocomplétion
- Recherche textuelle globale
- Export CSV des données filtrées
- Actions inline (édition, suppression, détails)

12.4.3 Formulaires

- Validation en temps réel avec messages d'erreur contextuels
- Autocomplétion pour les champs de référence
- Confirmation avant actions destructives (suppression)
- Sauvegarde automatique des brouillons

12.5 Responsive Design

12.5.1 Points de Rupture

TABLE 12.2 – Breakpoints responsive

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

12.5.2 Mode Sidebar vs Header

L'interface supporte deux modes de navigation (préférence stockée en cookie) :

- **Sidebar** : Navigation latérale persistante (recommandé desktop)
- **Header** : Navigation horizontale collapsible (mobile/tablette)

12.6 Thèmes et Personnalisation

Le système supporte les modes clair et sombre via l'attribut `data-theme` sur l'élément HTML racine. Les variables CSS sont définies pour chaque thème permettant une personnalisation complète des couleurs, typographies et espacements. La préférence utilisateur est persistée dans un cookie.

12.7 Accessibilité

12.7.1 Conformité WCAG

- **Contraste** : Ratio minimum 4.5 :1 pour le texte normal
- **Navigation clavier** : Tous les éléments interactifs focusables
- **Labels ARIA** : Descriptions pour lecteurs d'écran
- **Skip links** : Accès rapide au contenu principal
- **Focus visible** : Indicateur de focus clairement visible

12.8 Widget d'Aide IA

Le **GuidanceWidget** est un assistant flottant disponible sur toutes les pages. Il offre des suggestions contextuelles, maintient un historique de conversation, et s'intègre avec le Copilot backend pour fournir une aide intelligente aux utilisateurs.

Chapitre 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 (TTFB)	< 100ms	< 300ms
Disponibilité	> 99.5%	> 99%

13.2 Optimisations Backend

13.2.1 Indexation Base de Données

Des index SQL sont créés sur les colonnes fréquemment utilisées dans les filtres et jointures :

- `idx_intervention_date_type` : Recherche par date et type (calculs KPI)
- `idx_intervention_equipment_date` : Historique par équipement
- `idx_intervention_status` : Filtrage par statut (dashboard)
- `idx_equipment_serial` : Recherche par numéro de série

13.2.2 Pagination

Tous les endpoints de liste implémentent une pagination efficace avec :

- Paramètres `skip` et `limit` (max 500 par requête)
- Comptage total optimisé (query séparée)
- Indicateur `has_more` pour pagination infinie

13.2.3 Caching Redis

Le système RAG utilise Redis pour le caching des réponses :

- **Clé** : Hash MD5 de la requête utilisateur
- **TTL** : 1 heure par défaut (configurable)
- **Invalidation** : Automatique lors de l'ajout/suppression de documents

Bénéfice : Les requêtes répétitives sont servies instantanément sans solliciter le LLM.

13.2.4 Async I/O

FastAPI utilise le modèle asynchrone pour les opérations I/O non-bloquantes. Les appels concurrents (embeddings + contexte KPI) sont exécutés en parallèle via `asyncio.gather()`.

13.3 Optimisations Frontend

13.3.1 Server Components

React Server Components (RSC) réduisent le JavaScript envoyé au client :

- Les composants de données (listes, tables) sont rendus côté serveur
- Seuls les composants interactifs sont hydratés côté client
- Réduction de 40-60% du bundle JavaScript

13.3.2 React Query Caching

Configuration du cache React Query :

- **staleTime** : 5 minutes (données considérées fraîches)
- **cacheTime** : 30 minutes (données en mémoire)
- **refetchOnWindowFocus** : Désactivé pour éviter les requêtes inutiles

13.3.3 Code Splitting

Les composants lourds (graphiques, éditeurs) sont chargés dynamiquement via `next/dynamic` avec skeleton de chargement.

13.4 Scalabilité

13.4.1 Architecture Horizontale

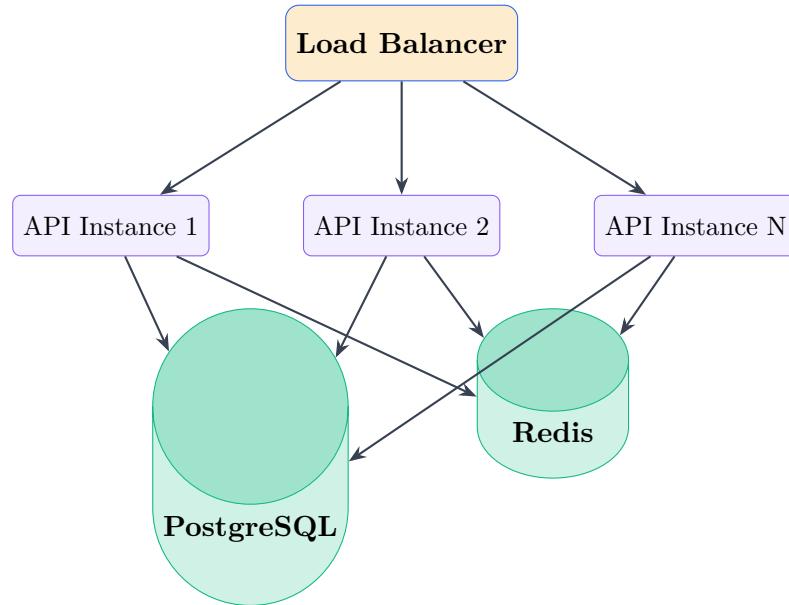


FIGURE 13.1 – Architecture horizontale scalable

L’architecture stateless du backend permet le scaling horizontal :

- Pas d’état en mémoire entre les requêtes
- Sessions gérées par Supabase (externe)
- Cache partagé via Redis
- Base de données centralisée

13.4.2 Considérations Docker

Docker Compose supporte le déploiement multi-instances avec limites de ressources (mémoire, CPU) par conteneur. L’option `deploy.replicas` permet de définir le nombre d’instances backend.

13.5 Monitoring

13.5.1 Health Checks

L’endpoint `GET /health/detailed` vérifie la connectivité de tous les composants :

- Base de données PostgreSQL
- Cache Redis
- Serveur Ollama
- Index FAISS

Retourne un status global (healthy/degraded/unhealthy) avec le détail par composant.

13.5.2 Métriques Recommandées

- **Request Rate** : Requêtes/seconde par endpoint
- **Error Rate** : Pourcentage d'erreurs 4xx/5xx
- **Latency** : P50, P95, P99 par endpoint
- **Database** : Connexions actives, temps de requête
- **Cache** : Hit ratio, mémoire utilisée
- **Memory** : Usage RAM par service

Outils Recommandés

Prometheus pour la collecte de métriques, **Grafana** pour la visualisation et l'alerting, **Sentry** pour le tracking d'erreurs, et **Vercel Analytics** pour les Web Vitals frontend.

Chapitre 14

Tests et Validation

14.1 Stratégie de Test

Le projet adopte une approche de test à plusieurs niveaux :

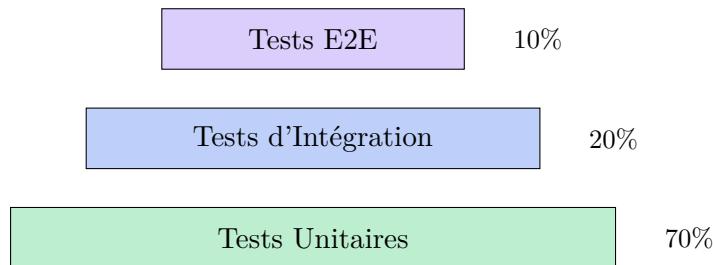


FIGURE 14.1 – Pyramide des tests

14.2 Tests Backend

14.2.1 Structure des Tests

Les tests backend sont organisés dans le dossier `backend/tests/` :

- `conftest.py` : Fixtures pytest (session DB, client test, headers auth)
- `test_equipment.py` : Tests CRUD équipements
- `test_interventions.py` : Tests workflow interventions
- `test_kpi.py` : Tests calculs MTBF, MTTR, Disponibilité
- `test_security.py` : Tests authentification et RBAC

14.2.2 Fixtures Pytest

Les fixtures fournissent :

- **db_session** : Base de données SQLite en mémoire, recréée à chaque test
- **client** : `TestClient` FastAPI avec injection de la session
- **auth_headers** : Headers Authorization avec token JWT de test
- **sample_equipment** : Données d'équipement de test

14.2.3 Tests Unitaires

Les tests unitaires vérifient la logique métier en isolation :

- **Calcul MTBF** : Vérifie les intervalles entre pannes
- **Calcul MTTR** : Vérifie la moyenne des durées de réparation
- **Disponibilité** : Vérifie le bornage 0-100%
- **Données insuffisantes** : Vérifie le retour null approprié

14.2.4 Tests d'Intégration API

Les tests d'intégration vérifient les endpoints REST :

- Création d'entité (POST) avec validation des données retournées
- Lecture avec pagination et filtres (GET)
- Mise à jour (PUT) avec vérification des changements
- Suppression (DELETE) avec vérification de cascade
- Codes d'erreur (401 sans auth, 403 sans permissions, 404 inexistant)

14.3 Tests Frontend

14.3.1 Tests E2E avec Playwright

Playwright est utilisé pour les tests end-to-end simulant des parcours utilisateur complets :

1. Navigation vers la page de login
2. Authentification avec credentials de test
3. Navigation vers le module à tester
4. Interactions (remplissage formulaires, clics, assertions)
5. Vérification des résultats (présence d'éléments, messages)

Scénarios couverts :

- Affichage de la liste des équipements
- Création d'un nouvel équipement
- Modification et suppression
- Filtrage et pagination des tableaux

14.4 Validation des Données

Les schémas Pydantic sont testés pour vérifier :

- Acceptation des données valides

- Rejet des champs vides requis
- Rejet des valeurs hors contraintes (longueur, range)
- Rejet des énumérations invalides
- Validation des dates (non futures)

14.5 Couverture de Code

TABLE 14.1 – Couverture de code cible

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

La couverture est mesurée avec [pytest-cov](#) et un seuil minimum de 70% est requis pour le CI.

14.6 CI/CD Testing

14.6.1 Pipeline GitHub Actions

Le workflow de test automatisé inclut :

1. Checkout du code source
2. Installation de Python 3.11 et des dépendances
3. Exécution des tests avec couverture
4. Upload du rapport de couverture vers Codecov
5. Échec si couverture < 70%

14.6.2 Conditions de Merge

Les pull requests doivent satisfaire :

- Tous les tests passent (status check)
- Couverture minimum maintenue
- Pas de régressions de performance
- Revue de code approuvée

Bonnes Pratiques

Chaque nouvelle fonctionnalité doit être accompagnée de tests unitaires. Les bugs corrigés doivent inclure un test de non-régression reproduisant le problème initial.

Chapitre 15

Déploiement

15.1 Vue d'Ensemble

ProAct peut être déployé selon plusieurs configurations adaptées aux besoins :

TABLE 15.1 – Options de déploiement

Configuration	Usage	Caractéristiques
Développement local	Dev/Test	Docker Compose, SQLite, hot-reload
Staging	Pré-production	Docker, PostgreSQL, Supabase test
Production	Live	K8s/Docker, Supabase prod, CI/CD

15.2 Déploiement Local (Développement)

15.2.1 Prérequis

- Docker Desktop 4.x+ avec Docker Compose
- Node.js 18.x+ et pnpm (gestionnaire de packages)
- Python 3.11+ (si exécution hors Docker)
- Git pour le contrôle de version

15.2.2 Services Docker Compose

Le fichier `docker-compose.yml` définit les services suivants :

- **api** : Backend FastAPI sur le port 8000
- **ollama** : Serveur de modèles LLM sur le port 11434
- **redis** : Cache sur le port 6379

Les volumes persistent les données Ollama (modèles) entre les redémarrages.

15.2.3 Commandes de Lancement

1. **Backend** : `docker-compose up -d` depuis le dossier `backend/`

2. **Modèles Ollama** : `ollama pull llama3.2:3b` et `ollama pull qwen2.5v1:3b`
3. **Frontend** : `pnpm install` puis `pnpm run dev` depuis le dossier frontend

15.3 Configuration Supabase

15.3.1 Variables d'Environnement Frontend

Le fichier `.env.local` du frontend doit contenir :

- `NEXT_PUBLIC_SUPABASE_URL` : URL du projet Supabase
- `NEXT_PUBLIC_SUPABASE_ANON_KEY` : Clé anonyme publique
- `NEXT_PUBLIC_API_BASE_URL` : URL du backend API

15.3.2 Variables d'Environnement Backend

Le fichier `.env` du backend doit contenir :

- `DATABASE_URL` : Chaîne de connexion PostgreSQL
- `SUPABASE_JWT_SECRET` : Secret JWT partagé avec Supabase
- `OLLAMA_BASE_URL` : URL du serveur Ollama
- `REDIS_HOST` et `REDIS_PORT` : Configuration Redis

15.3.3 Attribution des Rôles

Les rôles utilisateurs sont stockés dans `app_metadata` de Supabase. Pour attribuer un rôle admin, exécuter une requête SQL via le dashboard Supabase mettant à jour `raw_app_meta_data` avec le champ `role`.

15.4 Déploiement Production

15.4.1 Architecture Recommandée

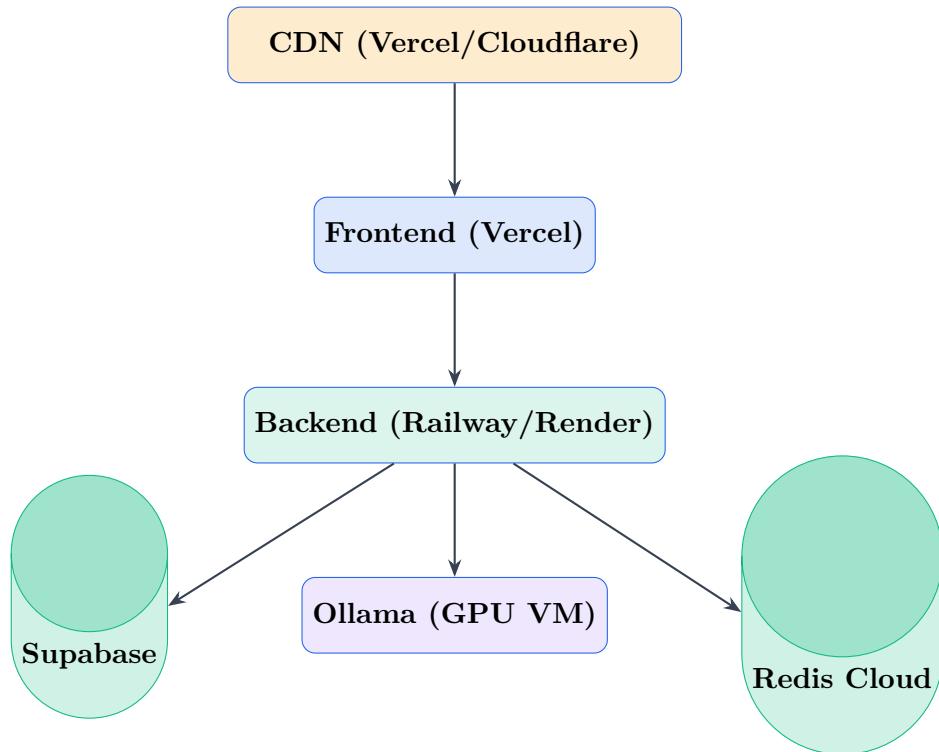


FIGURE 15.1 – Architecture de production

15.4.2 Déploiement Frontend (Vercel)

1. Connecter le repository GitHub au projet Vercel
2. Configurer les variables d'environnement de production
3. Définir la commande de build (`pnpm build`)
4. Activer le déploiement automatique sur push main

15.4.3 Déploiement Backend (Docker)

Pour la production, le Dockerfile utilise Gunicorn avec workers Uvicorn pour gérer la concurrence. Les variables sensibles sont injectées via les secrets du provider (Railway, Render, AWS).

15.5 Migrations de Base de Données

Alembic gère les migrations de schéma :

- `alembic revision -autogenerate -m "description"` : Créer une migration
- `alembic upgrade head` : Appliquer les migrations
- `alembic downgrade -1` : Rollback d'une migration

Un script d'initialisation peut peupler la base avec les données de référence (compétences, types de pannes).

15.6 Checklist de Déploiement

TABLE 15.2 – Checklist pré-déploiement

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

Important

Avant tout déploiement en production : tester complètement en staging, vérifier les configurations de sécurité (CORS, HTTPS), configurer les sauvegardes automatiques, et mettre en place le monitoring et les alertes.

Chapitre 16

Limites et Contraintes

16.1 Limites Techniques

16.1.1 Dépendance à Ollama

Le système IA repose sur le serveur Ollama local, ce qui implique :

- **Ressources GPU** : Performances significativement dégradées sans GPU dédié
- **Temps de démarrage** : Chargement initial des modèles (30-60 secondes)
- **Mémoire VRAM** : Modèles VLM requièrent 4-8 GB minimum
- **Latence** : Requêtes RAG/Copilot de 2-5 secondes

Impact Opérationnel

En l'absence de GPU ou avec des ressources insuffisantes, les fonctionnalités IA peuvent avoir des temps de réponse de 10-30 secondes, échouer sur les requêtes complexes (timeout), ou nécessiter des modèles plus légers avec qualité réduite.

16.1.2 Scalabilité du RAG

Le système RAG actuel présente des contraintes :

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

16.1.3 Base de Données

- **SQLite (développement)** : Pas de concurrence, mono-utilisateur
- **PostgreSQL** : Requiert configuration et maintenance

- **Pas de sharding** : Limite pratique à quelques millions d'enregistrements

16.2 Limites Fonctionnelles

16.2.1 Gestion des Permissions

- Modèle RBAC simple avec 4 rôles fixes
- Pas de permissions granulaires par ressource individuelle
- Pas de multi-tenancy natif
- Pas de délégation de droits entre utilisateurs

16.2.2 Workflows

- Workflow d'intervention linéaire (pas de branches conditionnelles)
- Pas d'approbations multi-niveaux configurables
- Pas de notifications push temps réel
- Pas d'intégration calendrier externe

16.2.3 Reporting

- Rapports limités aux KPIs intégrés
- Pas de générateur de rapports personnalisés
- Export PDF basique
- Pas de tableaux de bord configurables par l'utilisateur

16.3 Limites de l'Intelligence Artificielle

16.3.1 Qualité des Prédictions

- **Données requises** : Minimum 20-30 interventions par équipement pour des prédictions fiables
- **Précision RUL** : MAE typique de 15-30 jours
- **Saisonnalité** : Non prise en compte dans les modèles actuels
- **Événements exceptionnels** : Non modélisés (pandémies, arrêts prolongés)

16.3.2 Limitations LLM

- **Hallucinations** : Réponses parfois incorrectes ou inventées
- **Contexte limité** : ~4-8K tokens selon le modèle utilisé
- **Langue** : Optimisé pour le français, mais modèles majoritairement entraînés en

anglais

- **Données temps réel** : Pas de mise à jour automatique des connaissances

16.3.3 OCR

- **Qualité image** : Dépend fortement de la résolution et du contraste
- **Langues** : Principalement français/anglais
- **Formats** : Images uniquement (pas de PDF natif sans conversion)
- **Tableaux complexes** : Extraction parfois imparfaite des structures

16.4 Contraintes d'Intégration

- Pas d'intégration ERP native (SAP, Oracle)
- Pas de connecteurs IoT/SCADA pour données machines temps réel
- API REST uniquement (pas de GraphQL, pas de WebSocket pour temps réel)
- Authentification Supabase uniquement (pas de SSO entreprise SAML/LDAP)

16.5 Contraintes Opérationnelles

TABLE 16.2 – Contraintes opérationnelles

Aspect	Contrainte
Utilisateurs concurrents	~50-100 selon infrastructure
Volume de données	~1M interventions recommandées
Taille des uploads	10 MB par fichier (configurable)
Historique KPI	Recalcul complet (pas d'agrégation pré-calculée)
Backup	Non automatisé (responsabilité utilisateurs)

16.6 Améliorations Nécessaires

1. Rate limiting non implémenté (vulnérabilité DoS)
2. Cache RAG non invalidé automatiquement après modification de document
3. Timezone non géré explicitement (UTC assumé)
4. Logs de sécurité non persistés en base

Évolutions Prévues

Ces limitations sont documentées et priorisées pour les versions futures. Voir le chapitre 17 pour les perspectives d'évolution.

Chapitre 17

Perspectives d'Évolution

17.1 Feuille de Route

Cette section présente les évolutions envisagées pour les prochaines versions du système ProAct.

v2.1 – Améliorations IA et Performance

v2.2 – Intégrations Externes

v3.0 – Multi-tenancy et Enterprise

FIGURE 17.1 – Feuille de route des versions

17.2 Version 2.1 – Améliorations Court Terme

17.2.1 Intelligence Artificielle

1. RAG Amélioré

- Support PDF natif avec extraction de structure
- Recherche hybride (vectorielle + mots-clés BM25)
- Historique de conversations multi-tour
- Résumé automatique de documents longs

2. Prédiction Avancée

- Modèles LSTM pour séries temporelles longues
- Prise en compte de la saisonnalité
- Prédiction multi-équipement (fleet-level analytics)
- Détection d'anomalies en temps réel

3. OCR Amélioré

- Support multi-pages avec fusion
- Extraction de formulaires structurés
- Reconnaissance de schémas techniques

17.2.2 Performance

- Agrégation des KPIs (tables pré-calculées)
- Pagination côté serveur complète avec curseurs
- Compression des réponses API (gzip/brotli)
- Intégration Prometheus/Grafana pour monitoring

17.3 Version 2.2 – Intégrations

17.3.1 Connecteurs Externes

TABLE 17.1 – Intégrations prévues

Système	Type	Fonctionnalités
SAP PM	ERP	Import/sync équipements et ordres de travail
Oracle EAM	ERP	Synchronisation bidirectionnelle
SCADA/OPC-UA	IoT	Données temps réel machine
Microsoft 365	Productivité	Notifications, intégration calendrier
Power BI	Analytics	Export données pour reporting et dashboards

17.3.2 Notifications

- Notifications push via PWA
- Alertes email automatiques configurables
- Intégration Slack/Microsoft Teams
- SMS pour alertes critiques

17.3.3 API Avancée

- Endpoint GraphQL pour requêtes flexibles
- WebSocket pour mises à jour temps réel
- Webhooks sortants pour intégrations
- API versioning (v1, v2) pour rétrocompatibilité

17.4 Version 3.0 – Enterprise

17.4.1 Multi-tenancy

- Isolation des données par organisation
- Personnalisation par tenant (logo, couleurs, terminologie)
- Quotas et limites configurables par tenant
- Administration centralisée multi-tenant

17.4.2 Sécurité Enterprise

- SSO SAML 2.0 et OIDC
- Intégration LDAP/Active Directory
- MFA obligatoire configurable
- Audit trail immuable et requêtéable
- Conformité RGPD (export, suppression données)

17.4.3 Fonctionnalités Avancées

1. Workflows Configurables

- Designer graphique de workflows
- Approbations multi-niveaux
- Escalations automatiques
- Conditions et branchements dynamiques

2. Reporting Avancé

- Générateur de rapports personnalisés
- Tableaux de bord configurables par utilisateur
- Exports programmés (email, SFTP)

3. Mobile Natif

- Application iOS/Android native
- Mode hors-ligne avec synchronisation
- Scan QR codes équipements
- Capture photos et pièces jointes

17.5 Évolutions Techniques

- Migration optionnelle vers microservices
- Orchestration Kubernetes pour scaling
- Event sourcing pour audit trail complet
- Fine-tuning de modèles LLM sur données GMAO
- Vision AI pour inspection visuelle automatique

17.6 Conclusion

Le système ProAct constitue une base solide pour une GMAO moderne intégrant l’Intelligence Artificielle. Les évolutions prévues visent à renforcer les capacités prédictives, faciliter l’intégration dans l’écosystème existant, répondre aux exigences enterprise, et améliorer l’expérience utilisateur.

Cette documentation technique fournit les éléments nécessaires pour comprendre, déployer et étendre le système. Les retours des utilisateurs et contributeurs sont essentiels pour orienter les développements futurs.