

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

15 janvier 2026

Version 2.0.0

Table des matières

1	Introduction Générale	9
1.1	Contexte et Enjeux	9
1.2	Objectifs du Projet	9
1.3	Périmètre Fonctionnel	10
1.4	Structure de ce Document	10
1.5	Technologies Utilisées	11
2	Vue d'Ensemble du Système	12
2.1	Architecture Globale	12
2.2	Flux de Données Principal	12
2.3	Composants Principaux	13
2.3.1	Frontend (Next.js)	13
2.3.2	Backend (FastAPI)	13
2.3.3	Base de Données	14
2.4	Rôles Utilisateurs	14
2.5	Points d'Intégration	15
2.5.1	Supabase	15
2.5.2	Ollama	15
3	Architecture Frontend	16
3.1	Vue d'Ensemble	16
3.2	Structure du Projet	16
3.2.1	Organisation des Dossiers	16
3.2.2	Pages du Dashboard	17
3.3	Système de Layout	17
3.3.1	Layout Principal	17
3.3.2	Composants de Navigation	17
3.4	Client API GMAO	18
3.5	Composants UI Réutilisables	18
3.5.1	Data Tables	18
3.5.2	Formulaires	18
3.5.3	Graphiques	18

3.6	Gestion des Thèmes	19
3.7	Internationalisation	19
3.8	Optimisations de Performance	19
4	Authentification et Autorisation	20
4.1	Architecture de Sécurité	20
4.2	Intégration Makerkit	20
4.3	Modèle de Données Utilisateur	20
4.3.1	Structure AuthUser	21
4.3.2	Énumération des Rôles	21
4.4	Vérification JWT	21
4.5	Contrôle d'Accès (RBAC)	21
4.5.1	Guards de Rôle	22
4.5.2	Application aux Routes	22
4.6	Matrice des Permissions	22
4.7	Sécurité des Sessions	23
4.8	Gestion des Erreurs d'Authentification	23
5	Architecture Backend	24
5.1	Vue d'Ensemble	24
5.2	Structure du Projet	24
5.3	Point d'Entrée Application	25
5.3.1	Initialisation FastAPI	25
5.3.2	Enregistrement des Routers	25
5.4	Couche Routers	25
5.5	Couche Services	26
5.6	Gestion des Erreurs	26
5.7	Endpoints de Santé	27
5.8	Documentation API	27
6	Modèle de Données	28
6.1	Vue d'Ensemble	28
6.2	Schéma Entités-Relations	28
6.3	Entités Principales	28
6.3.1	Equipment	29
6.3.2	Intervention	29
6.3.3	SparePart	29
6.3.4	Technician	30
6.4	Tables d'Association	30
6.4.1	InterventionPart	30

6.4.2	TechnicianAssignment	30
6.4.3	TechnicianSkill	30
6.5	Modèles AMDEC	30
6.5.1	FailureMode	30
6.5.2	RPNAnalysis	30
6.6	Modèles RAG	31
6.7	Schémas Pydantic	31
7	Système d'Interventions	32
7.1	Vue d'Ensemble	32
7.2	Cycle de Vie d'une Intervention	32
7.3	Création d'une Intervention	32
7.4	Assignation des Techniciens	33
7.4.1	Processus d'Assignation	33
7.5	Gestion des Pièces Utilisées	33
7.6	Clôture d'Intervention	33
7.7	Requêtes et Filtrage	34
7.8	Données Capturées	34
8	Système de Soumission Technicien	36
8.1	Vue d'Ensemble	36
8.2	Profil Technicien	36
8.2.1	Données du Profil	36
8.2.2	Gestion des Compétences	36
8.3	Workflow de Soumission	37
8.4	Fonctionnalités API Technicien	37
8.4.1	Mes Assignations	37
8.4.2	Soumission d'Heures	37
8.5	Rapport d'Intervention	37
8.5.1	Contenu du Rapport	37
8.5.2	Traitement du Rapport	38
8.6	Statistiques Technicien	38
8.7	Notifications	38
9	KPI et Analytics	40
9.1	Introduction aux KPIs de Maintenance	40
9.2	Indicateurs Principaux	40
9.2.1	MTBF – Mean Time Between Failures	40
9.2.2	MTTR – Mean Time To Repair	41
9.2.3	Disponibilité	41

9.3	Dashboard KPIs	41
9.3.1	Paramètres de Filtrage	42
9.4	Distributions et Tendances	42
9.4.1	Distribution des Pannes	42
9.4.2	Répartition des Coûts	42
9.5	Visualisation Frontend	42
9.6	Interprétation des KPIs	43
10	Intelligence Artificielle	44
10.1	Vue d'Ensemble	44
10.2	Système RAG	44
10.2.1	Présentation	44
10.2.2	Architecture RAG	44
10.2.3	Pipeline de Traitement	45
10.3	Maintenance Copilot	45
10.3.1	Fonctionnalités	45
10.3.2	Détection d'Intent	45
10.4	OCR Vision	46
10.4.1	Présentation	46
10.4.2	Modes d'Extraction	46
10.5	AI Forecast (Prédiction)	46
10.5.1	Objectifs	47
10.5.2	Algorithmes Utilisés	47
10.5.3	Pipeline de Prédiction	47
10.6	Intégration Ollama	47
11	Sécurité	49
11.1	Vue d'Ensemble	49
11.2	Authentification	49
11.2.1	JWT (JSON Web Tokens)	49
11.2.2	Validation du Token	50
11.3	Autorisation (RBAC)	50
11.3.1	Hierarchie des Rôles	50
11.3.2	Implémentation des Guards	50
11.4	Validation des Entrées	50
11.4.1	Protection contre les Injections	50
11.4.2	Protection SQL Injection	51
11.5	Sécurité des Communications	51
11.5.1	CORS (Cross-Origin Resource Sharing)	51
11.5.2	Headers de Sécurité	51

11.6	Protection des Données	51
11.6.1	Classification des Données Sensibles	52
11.6.2	Variables d'Environnement	52
11.7	Audit et Logging	52
11.8	Checklist de Sécurité	53
12	UX et Rôles Utilisateurs	54
12.1	Principes de Design	54
12.2	Parcours Utilisateurs par Rôle	54
12.2.1	Administrateur	54
12.2.2	Superviseur	54
12.2.3	Technicien	55
12.2.4	Viewer	55
12.3	Structure de Navigation	55
12.3.1	Menu Principal	56
12.4	Composants d'Interface	56
12.4.1	Dashboard Principal	56
12.4.2	Tableaux de Données	56
12.4.3	Formulaires	57
12.5	Responsive Design	57
12.5.1	Points de Rupture	57
12.5.2	Mode Sidebar vs Header	57
12.6	Thèmes et Personnalisation	57
12.7	Accessibilité	58
12.7.1	Conformité WCAG	58
12.8	Widget d'Aide IA	58
13	Performances et Scalabilité	59
13.1	Objectifs de Performance	59
13.2	Optimisations Backend	59
13.2.1	Indexation Base de Données	59
13.2.2	Pagination	59
13.2.3	Caching Redis	60
13.2.4	Async I/O	60
13.3	Optimisations Frontend	60
13.3.1	Server Components	60
13.3.2	React Query Caching	60
13.3.3	Code Splitting	60
13.4	Scalabilité	60
13.4.1	Architecture Horizontale	61

13.4.2	Considérations Docker	61
13.5	Monitoring	61
13.5.1	Health Checks	61
13.5.2	Métriques Recommandées	62
14	Tests et Validation	63
14.1	Stratégie de Test	63
14.2	Tests Backend	63
14.2.1	Structure des Tests	63
14.2.2	Fixtures Pytest	63
14.2.3	Tests Unitaires	64
14.2.4	Tests d'Intégration API	64
14.3	Tests Frontend	64
14.3.1	Tests E2E avec Playwright	64
14.4	Validation des Données	64
14.5	Couverture de Code	65
14.6	CI/CD Testing	65
14.6.1	Pipeline GitHub Actions	65
14.6.2	Conditions de Merge	65
15	Déploiement	67
15.1	Vue d'Ensemble	67
15.2	Déploiement Local (Développement)	67
15.2.1	Prérequis	67
15.2.2	Services Docker Compose	67
15.2.3	Commandes de Lancement	67
15.3	Configuration Supabase	68
15.3.1	Variables d'Environnement Frontend	68
15.3.2	Variables d'Environnement Backend	68
15.3.3	Attribution des Rôles	68
15.4	Déploiement Production	68
15.4.1	Architecture Recommandée	69
15.4.2	Déploiement Frontend (Vercel)	69
15.4.3	Déploiement Backend (Docker)	69
15.5	Migrations de Base de Données	69
15.6	Checklist de Déploiement	70
16	Limites et Contraintes	71
16.1	Limites Techniques	71
16.1.1	Dépendance à Ollama	71

16.1.2 Scalabilité du RAG	71
16.1.3 Base de Données	71
16.2 Limites Fonctionnelles	72
16.2.1 Gestion des Permissions	72
16.2.2 Workflows	72
16.2.3 Reporting	72
16.3 Limites de l'Intelligence Artificielle	72
16.3.1 Qualité des Prédictions	72
16.3.2 Limitations LLM	72
16.3.3 OCR	73
16.4 Contraintes d'Intégration	73
16.5 Contraintes Opérationnelles	73
16.6 Améliorations Nécessaires	73
17 Perspectives d'Évolution	75
17.1 Feuille de Route	75
17.2 Version 2.1 – Améliorations Court Terme	75
17.2.1 Intelligence Artificielle	75
17.2.2 Performance	76
17.3 Version 2.2 – Intégrations	76
17.3.1 Conecteurs Externes	76
17.3.2 Notifications	76
17.3.3 API Avancée	76
17.4 Version 3.0 – Entreprise	76
17.4.1 Multi-tenancy	77
17.4.2 Sécurité Entreprise	77
17.4.3 Fonctionnalités Avancées	77
17.5 Évolutions Techniques	77
17.6 Conclusion	78

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 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

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.

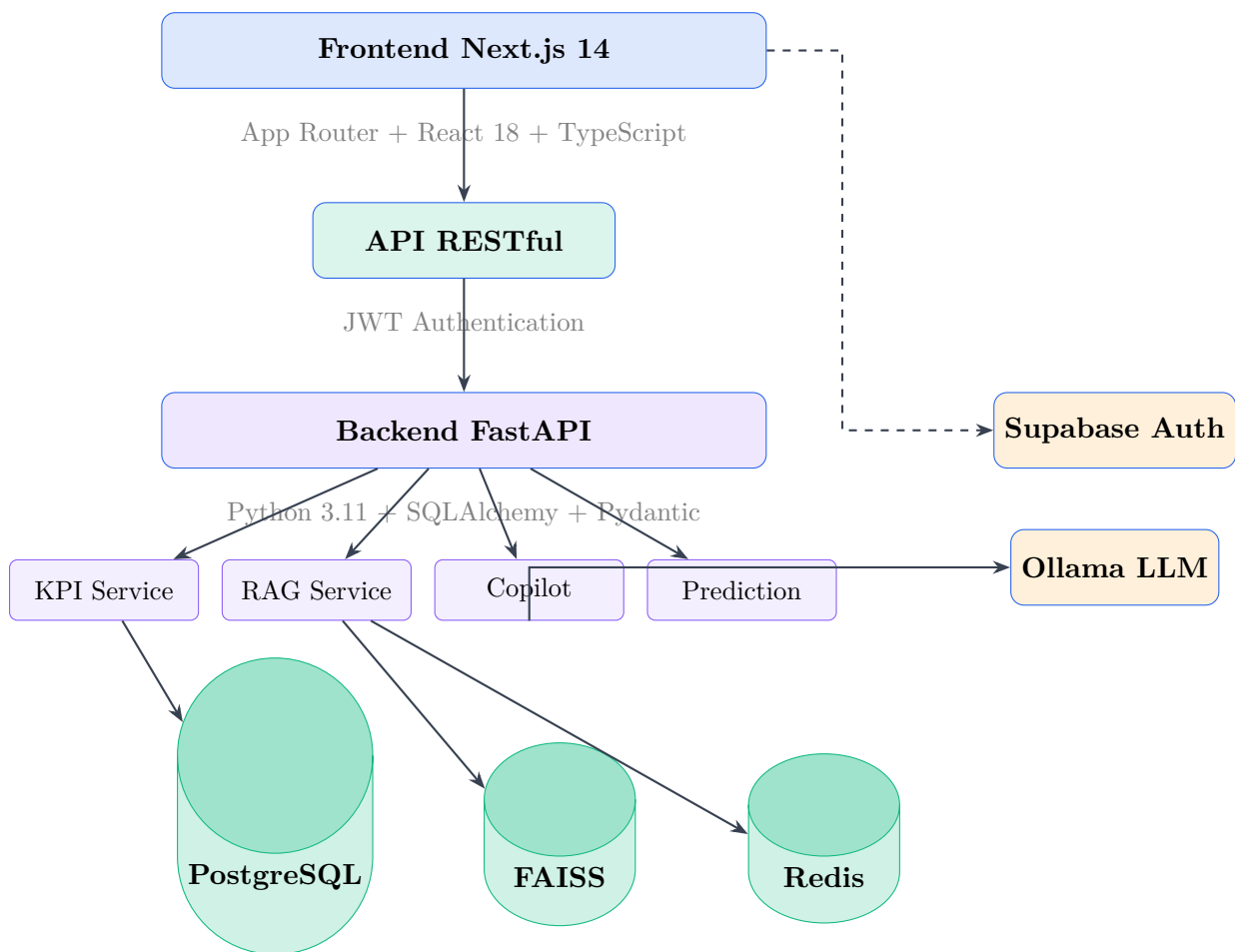


FIGURE 2.1 – Architecture globale du système ProAct

2.2 Flux de Données Principal

Le système gère plusieurs flux de données interconnectés :

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 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	De
/api/equipment	Equipment	CRUD équipements, s
/api/interventions	Interventions	Ordres de travail, as
/api/spare-parts	Spare Parts	Inventaire pièces d
/api/technicians	Technicians	Profils, co
/api/kpi	KPIs & Analytics	MTBF, MTTR, Di
/api/amdec	AMDEC & RPN	Modes de
/api/rag	RAG System	Upload document
/api/copilot	Maintenance Copilot	Assistant convo
/api/ocr	OCR Vision AI	Extraction te
/api/predict	AI Forecast	Prédiction RU

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	Fonction
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
assistant/	Interface de consultation
copilot/	Assistant IA
ocr/	Upload et extraction de documents
ai-forecast/	Prédictions maintenance
import-export/	Import/Export de données

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

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.

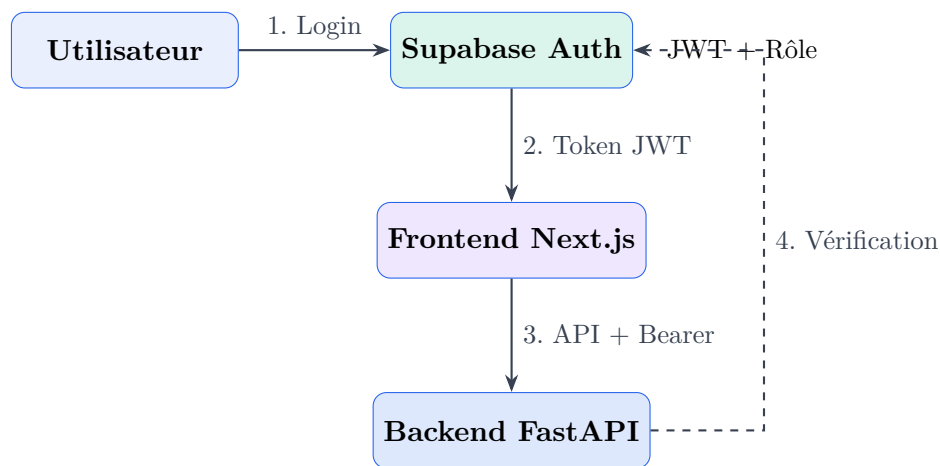


FIGURE 4.1 – Flux d'authentification

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`
- **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 des permissions par endpoint

	Endpoint	Admin	Super	Tech	View
<i>Équipements</i>					
	GET /equipment	✓	✓	✓	✓
	POST /equipment	✓	✓	–	–
	DELETE /equipment/ :id	✓	–	–	–
<i>Interventions</i>					
	GET /interventions	✓	✓	✓	✓
	POST /interventions	✓	✓	✓	–
<i>Intelligence Artificielle</i>					
	POST /rag/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
401 Unauthorized	Token manquant ou invalide
403 Forbidden	Rôle insuffisant
500 Server Error	JWT_SECRET non configuré

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
- `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 DB et RAG au démarrage
- **Middleware CORS** : Configuration des origines autorisées
- **Exception handler** : Capture et logging des erreurs
- **Métadonnées OpenAPI** : Titre, description, version

5.3.2 Enregistrement des Routers

Chaque router est enregistré avec un préfixe d'URL, un tag pour la documentation Swagger, et une dépendance d'authentification.

5.4 Couche Routers

Les routers définissent les endpoints REST et délèguent la logique aux services. Chaque router suit un pattern standardisé : définition CRUD, injection des dépendances, validation Pydantic, et retour typé.

TABLE 5.1 – Routers principaux

Router	Préfixe	
equipment	/api/equipment	CRUD é
interventions	/api/interventions	
spare_parts	/api/spare-parts	
technicians	/api/technicians	Pr
kpi	/api/kpi	MTBF, M
amdec	/api/amdec	Modes
rag	/api/rag	Upl
copilot	/api/copilot	
ocr	/api/ocr	
prediction	/api/predict	

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
- **CopilotService** : Orchestration des requêtes IA
- **PredictionService** : Modèles ML pour RUL et prévision
- **OCRService** : Extraction via modèles Vision-Language
- **RAGService** : Pipeline de recherche documentaire

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	Ex
400	Bad Request	Validation Pydantic é
401	Unauthorized	JWT expiré ou in
403	Forbidden	Rôle non a
404	Not Found	Équipement ID in
422	Unprocessable	Champ requis man
500	Server Error	Exception non

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
- **ReDoc** : [/redoc](#) – Documentation lisible
- **OpenAPI** : [/openapi.json](#) – Spécification JSON

Tags API

Les endpoints sont organisés par tags pour une navigation claire : Equipment, Interventions, Spare Parts, Technicians, KPIs, AMDEC, RAG System, 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

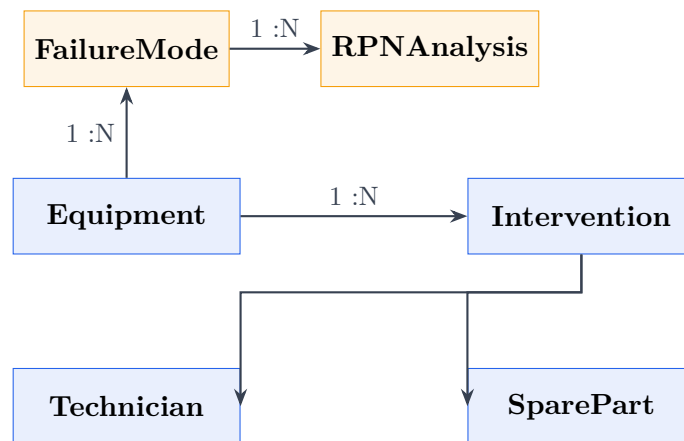


FIGURE 6.1 – Diagramme entités-relations simplifié

6.3 Entités Principales

6.3.1 Equipment

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

TABLE 6.1 – Attributs de l’entité Equipment

Attribut	Type	
id	Integer (PK)	
designation	String(200)	No
type	String(100)	
location	String(200)	Emp
status	Enum	Active, Inac
acquisition_date	Date	
manufacturer	String(100)	
serial_number	String(100)	Numér

6.3.2 Intervention

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

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

Des index sont créés sur les colonnes fréquemment filtrées.

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)
- Informations professionnelles : Spécialité, taux horaire
- Statut : Active, Inactive, On Leave

6.4 Tables d'Association

- **InterventionPart** : Association N :M interventions-pièces avec quantité utilisée
- **TechnicianAssignment** : Association N :M interventions-techniciens avec heures travaillées
- **TechnicianSkill** : Association N :M techniciens-compétences avec niveau de maîtrise

6.5 Modèles AMDEC

- **FailureMode** : Représente un mode de défaillance potentiel avec effet et cause
- **RPNAnalysis** : Stocke les évaluations RPN avec scores Gravité, Occurrence, Détection (1-10)

6.6 Modèles RAG

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

6.7 Schémas Pydantic

Les schémas Pydantic assurent la validation des données :

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

Validation Automatique

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

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), **préventive** (planifiée), ou **améliorative** (upgrade ou optimisation).

7.2 Cycle de Vie d'une Intervention

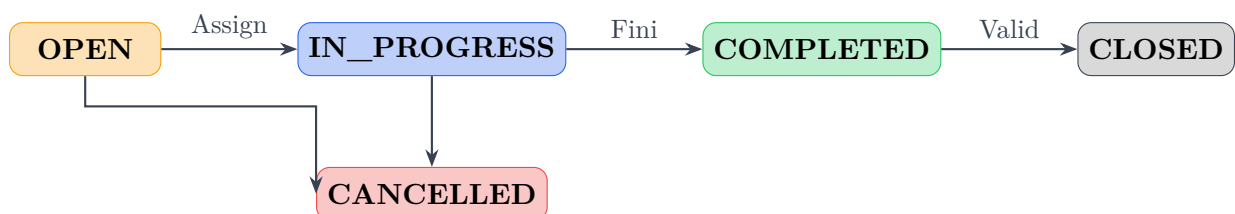


FIGURE 7.1 – États du cycle de vie d'une intervention

7.3 Création d'une Intervention

La création s'effectue via `POST /api/interventions/` avec :

- **equipment_id** (requis) : Identifiant de l'équipement
- **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

7.4 Assignment des Techniciens

L'assignation crée un enregistrement `TechnicianAssignment` qui capture le lien intervention-technicien, les heures estimées/travaillées, et le taux horaire.

7.5 Gestion des Pièces Utilisées

L'ajout de pièces via `POST /api/interventions/{id}/parts` vérifie le stock, crée l'association, décrémente automatiquement le stock, et recalcule le coût matériel.

7.6 Clôture d'Intervention

La clôture passe le statut à COMPLETED, enregistre la date effective, capture la durée d'indisponibilité, et calcule les coûts finaux.

7.7 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

7.8 Données Capturées

TABLE 7.2 – Données par intervention

Donnée	Type	
Durée indisponibilité	Float (heures)	
Coût matériel	Float (€)	
Coût main d'œuvre	Float (€)	
Type de panne	String	Distrib
Date intervention	Date	

Traçabilité Complète

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

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

TABLE 8.1 – Attributs du profil technicien

Attribut	Type	
Nom / Prénom	String	Ident
Email	String (unique)	Contact et
Téléphone	String	
Spécialité	String	Dor
Taux horaire	Float	
Statut	Enum	Active, In
Date embauche	Date	

8.2.1 Gestion des Compétences

Le système permet d'associer des compétences aux techniciens avec niveau de maîtrise (1-5), date d'acquisition, et statut de certification.

8.3 Workflow de Soumission

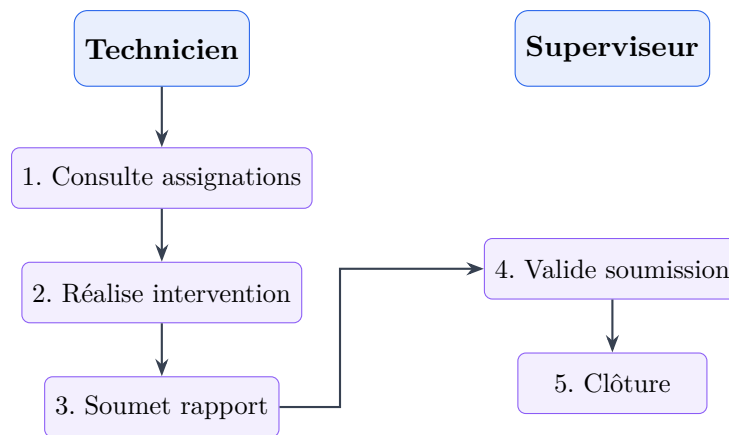


FIGURE 8.1 – Workflow de soumission technicien

8.4 Fonctionnalités API

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

8.5 Rapport d'Intervention

Un rapport complet contient :

- **Description des travaux** : Détail des actions réalisées
- **Cause identifiée** : Diagnostic de la cause racine
- **Actions correctives** : Mesures prises
- **Pièces utilisées** : Liste avec quantités
- **Heures travaillées** : Temps passé
- **Date de réalisation** : Date effective

8.6 Statistiques Technicien

TABLE 8.2 – Statistiques disponibles

Métrique	D
Total interventions	Nombre d'intervention
Total heures	Cumul des heures
Coût main d'œuvre	Somme des coûts
Moyenne heures/interv.	Indicateur
Nombre compétences	Polyvalence d

Intégration Future

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

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.

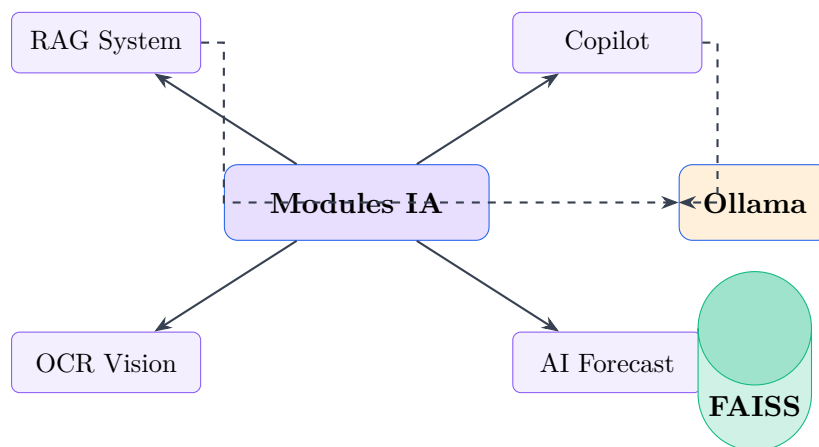


FIGURE 10.1 – Architecture des modules IA

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
2. **Augmentation** : Enrichissement du contexte LLM
3. **Generation** : Production de réponses contextualisées

10.2.2 Architecture RAG

Le sous-système RAG est organisé en modules spécialisés :

- **RAGService** : Orchestrateur principal du pipeline

- **DocumentProcessor** : Parsing et découpage en chunks
- **EmbeddingService** : Génération des vecteurs 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 : Création enregistrement, extraction texte, découpage en chunks (500-1000 tokens), génération embeddings, stockage FAISS.

Requête RAG : Vérification cache, embedding requête, recherche K chunks similaires, construction prompt avec contexte, génération réponse, mise en cache.

10.3 Maintenance Copilot

10.3.1 Fonctionnalités

Le **Copilot** est un assistant conversationnel spécialisé 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

TABLE 10.1 – Intentions supportées par le Copilot

Intent	
KPI_EXPLANATION	Questions sur
EQUIPMENT_HEALTH	É
INTERVENTION_REPORT	De
GENERAL_QUESTION	A

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)
- **Provider** : Ollama local
- **Formats d'entrée** : PNG, JPEG, WebP
- **Formats de sortie** : Markdown, HTML, JSON, Texte brut

Les images sont optimisées (redimensionnement à 1024px max) avant envoi au modèle.

10.5 AI Forecast (Prédiction)

10.5.1 Objectifs

Le module de **prédiction** fournit :

- **RUL** (Remaining Useful Life) : Estimation des jours restants
- **Prévision MTBF** : Tendence future sur 90 jours
- **Date prochaine panne** : Estimation probabiliste

10.5.2 Algorithmes Utilisés

TABLE 10.2 – Algorithmes de prédiction

Algorithme	Type	
Random Forest	ML supervisé	Prédiction
XGBoost	ML supervisé	Prédiction F
Prophet	Séries temp.	Prévision t

10.6 Intégration Ollama

TABLE 10.3 – Variables d’environnement Ollama

Variable	Valeur
OLLAMA_BASE_URL	http ://
OLLAMA_MODEL	
OLLAMA_VISION_MODEL	
OLLAMA_EMBEDDING_MODEL	non

Prérequis

Les modèles Ollama doivent être téléchargés avant utilisation via `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.

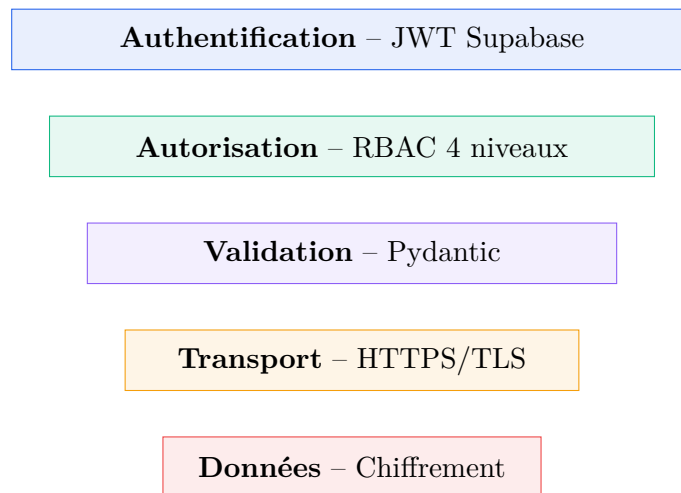


FIGURE 11.1 – Couches de sécurité

11.2 Authentification

11.2.1 JWT (JSON Web Tokens)

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

11.2.2 Validation du Token

Le processus vérifie : header Authorization, signature cryptographique, date d'expiration, et claims requis (`sub`, `email`).

11.3 Autorisation (RBAC)

TABLE 11.1 – Hiérarchie des rôles

Rôle	Niveau	
Admin	4	Accès total, gestion r
Supervisor	3	Gestion équipements, a
Technician	2	Interventions, s
Viewer	1	Consultation seu

11.4 Validation des Entrées

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

11.5 Sécurité des Communications

La configuration CORS définit les origines autorisées, méthodes (GET, POST, PUT, DELETE), et headers (Authorization, Content-Type).

Configuration Production

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

11.6 Protection des Données

TABLE 11.2 – Classification des données

Donnée	Sensibilité	
Mots de passe	Critique	Hash bcrypt
Tokens JWT	Haute	HttpC
Emails utilisateurs	Moyenne	Accès restr
Coûts	Moyenne	Visib
Données techniques	Basse	Accès

11.7 Checklist de 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

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

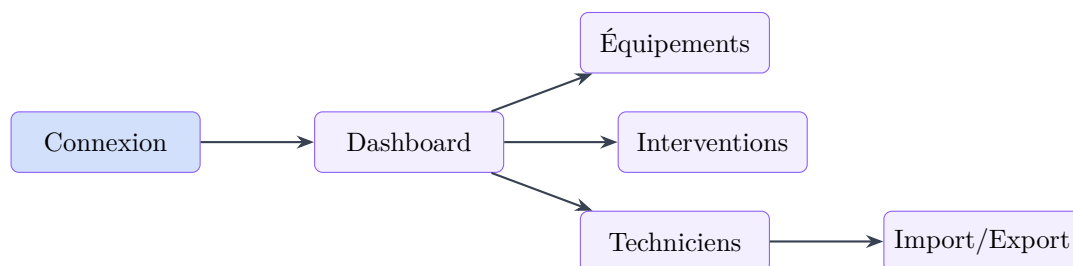


FIGURE 12.1 – Parcours Administrateur

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	Adm	Sup	Tech	View
	Dashboard	Home	✓	✓	✓	✓
	Équipements	Settings	✓	✓	✓	✓
	Interventions	Wrench	✓	✓	✓	✓
	Pièces	Package	✓	✓	–	–
	Techniciens	Users	✓	✓	–	–
	KPI	BarChart	✓	✓	✓	✓
	AMDEC/RPN	Alert	✓	✓	–	–
<i>IA & Analytics</i>						
	Assistant RAG	Message	✓	✓	✓	✓
	Copilot	Bot	✓	✓	✓	✓
	OCR	Scan	✓	✓	–	–
	AI Forecast	Trend	✓	✓	–	–
	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	
Mobile	< 640px	Menu hamburger,
Tablette	640-1024px	Sidebar réduite, g
Desktop	> 1024px	Sidebar complète, gri

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

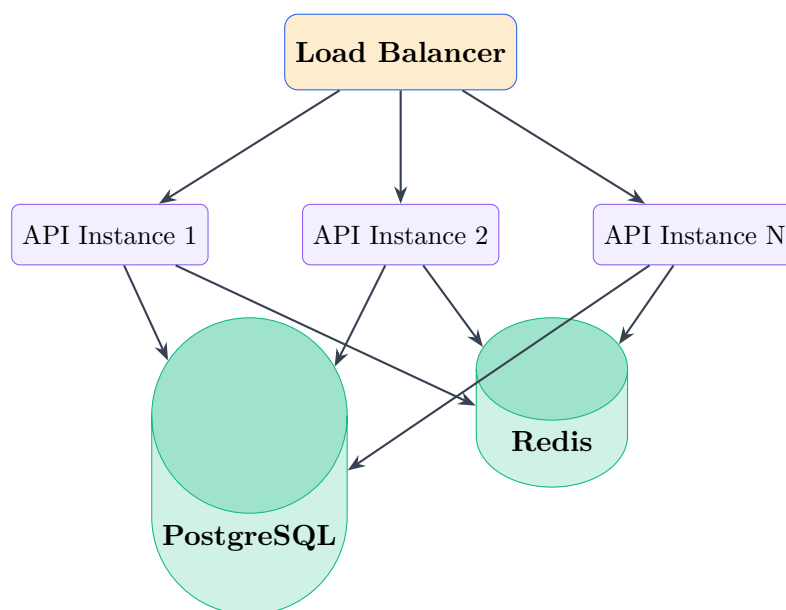


FIGURE 13.1 – Architecture horizontale scalable

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 :

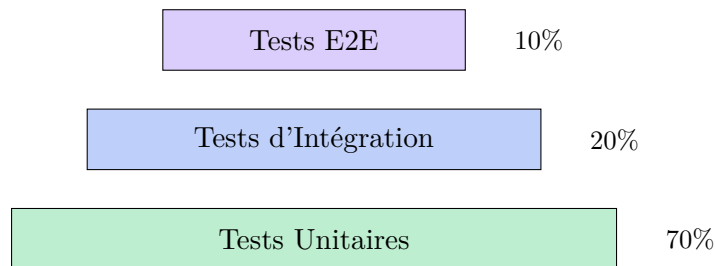


FIGURE 14.1 – Pyramide des tests

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, recrée 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
Production	Live	K8s/Docker, Supabase prod, Cloud

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 qwen2.5vl: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

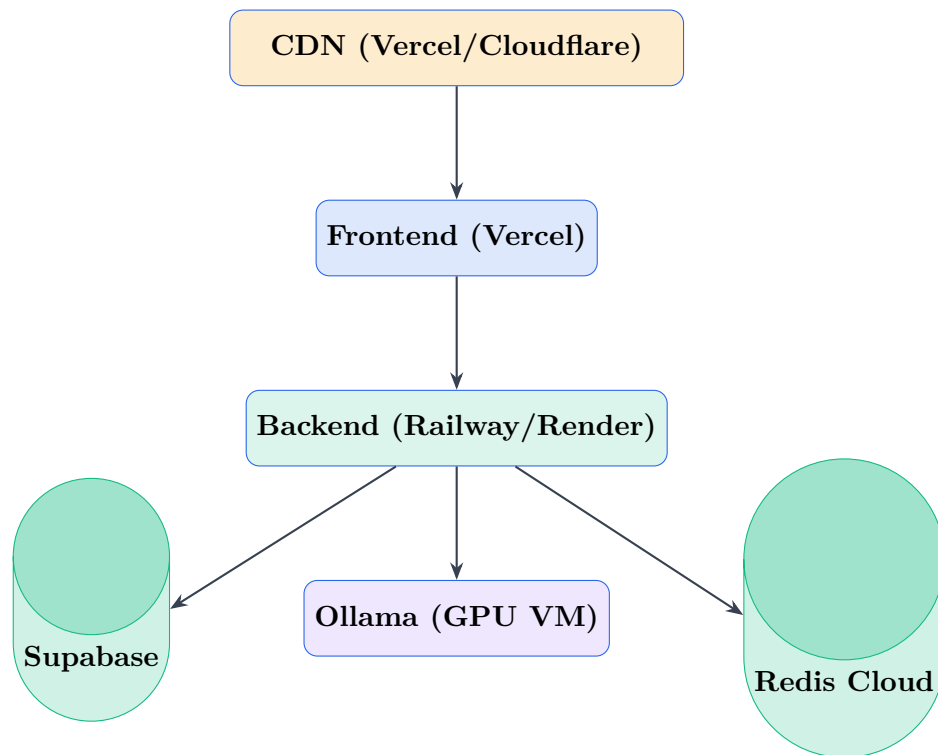


FIGURE 15.1 – Architecture de production

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 :

- **Ressources GPU** : Performances dégradées sans GPU
- **Temps de démarrage** : 30-60 secondes pour charger les modèles
- **Mémoire VRAM** : 4-8 GB minimum pour modèles VLM
- **Latence** : 2-5 secondes par requête RAG/Copilot

Impact Opérationnel

Sans GPU, les fonctionnalités IA peuvent avoir des temps de réponse de 10-30 secondes ou échouer sur les requêtes complexes.

16.1.2 Scalabilité du 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

16.2.1 Gestion des Permissions

- Modèle RBAC simple avec 4 rôles fixes
- Pas de permissions granulaires par ressource

- Pas de multi-tenancy natif

16.2.2 Workflows et Reporting

- Workflow d'intervention linéaire (pas de branches)
- Pas d'approbations multi-niveaux
- Pas de générateur de rapports personnalisés
- Pas de notifications push temps réel

16.3 Limites de l'IA

16.3.1 Qualité des Prédiction

- **Données requises** : Min. 20-30 interventions par équipement
- **Précision RUL** : MAE typique de 15-30 jours
- **Saisonnalité** : Non prise en compte

16.3.2 Limitations LLM

- **Hallucinations** : Réponses parfois incorrectes
- **Contexte limité** : ~4-8K tokens
- **Langue** : Modèles majoritairement anglais

16.4 Contraintes d'Intégration

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

16.5 Contraintes Opérationnelles

TABLE 16.2 – Contraintes opérationnelles

Aspect	
Utilisateurs concurrents	~50-100 sel
Volume de données	~1M interventions r
Taille uploads	10 MB par fichi
Backup	

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.

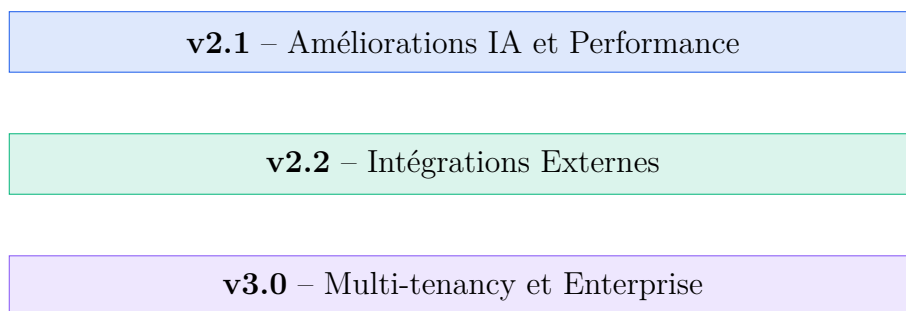


FIGURE 17.1 – Feuille de route des versions

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 Conecteurs Externes

TABLE 17.1 – Intégrations prévues

Système	Type	Fonction
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 avancé

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 immutable et requêtable
- 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 entreprise, 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.