P M

MODULE PROJECT MANAGEMENT

Spécifications Techniques

SPECIFICATIONS_TECHNIQUES.md

HR_ONIAN · Spécification Technique · Février 2026

Spécifications Techniques - Module GAC (Gestion des Achats & Commandes)

Version: 1.0
Date: 01/02/2026
Projet: HR_ONIAN
Auteur: Équipe de développement

---

1. Vue d'ensemble

1.1 Objectif du module

Le module GAC (Gestion des Achats & Commandes) a pour objectif de digitaliser et d'automatiser l'ensemble du processus d'achat de l'entreprise, de la demande initiale à la réception des marchandises.

1.2 Périmètre fonctionnel

Inclus :
- Gestion des demandes d'achat
- Workflow de validation hiérarchique
- Gestion des bons de commande
- Gestion des fournisseurs
- Catalogue produits interne
- Gestion budgétaire
- Réception et contrôle des livraisons
- Tableaux de bord et reporting

Exclus (pour versions futures) :
- Gestion comptable complète (facturation fournisseur)
- Intégration EDI (Échange de Données Informatisé)
- Gestion des appels d'offres
- Gestion des stocks complexe

1.3 Acteurs

1. **Demandeur** : Tout employé pouvant créer une demande d'achat
2. **Validateur N1** : Manager direct (première validation)
3. **Validateur N2** : Direction/Responsable achats (validation finale)
4. **Acheteur** : Personne en charge de passer les commandes
5. **Réceptionnaire** : Personne recevant et contrôlant les livraisons
6. **Gestionnaire budget** : Personne gérant les enveloppes budgétaires
7. **Admin GAC** : Administrateur du module

---

2. Architecture

2.1 Structure du module

```
gestion_achats/
├── models.py # Tous les modèles
├── apps.py # Configuration Django
├── admin.py # Interface admin
├── urls.py # Routes URL
├── permissions.py # Gestion des permissions
```

```
├── signals.py # Signaux pour notifications
├── validators.py # Validateurs personnalisés
├── utils.py # Fonctions utilitaires
├── constants.py # Constantes du module
├── services/
│ ├── __init__.py
│ ├── demande_service.py # Logique des demandes
│ ├── bon_commande_service.py # Logique des BCs
│ ├── fournisseur_service.py # Logique fournisseurs
│ ├── reception_service.py # Logique réceptions
│ ├── budget_service.py # Contrôle budgétaire
│ ├── catalogue_service.py # Gestion catalogue
│ └── notification_service.py # Notifications
├── views/
│ ├── __init__.py
│ ├── demande_views.py # Vues demandes
│ ├── bon_commande_views.py # Vues BCs
│ ├── fournisseur_views.py # Vues fournisseurs
│ ├── reception_views.py # Vues réceptions
│ ├── catalogue_views.py # Vues catalogue
│ ├── budget_views.py # Vues budgets
│ └── dashboard_views.py # Tableaux de bord
├── forms/
│ ├── __init__.py
│ ├── demande_forms.py
│ ├── bon_commande_forms.py
│ ├── fournisseur_forms.py
│ ├── reception_forms.py
│ └── budget_forms.py
├── templates/gestion_achats/
│ ├── base_gac.html # Template de base
│ ├── dashboard/
│ ├── demande/
│ ├── bon_commande/
│ ├── fournisseur/
│ ├── reception/
│ ├── catalogue/
│ └── budget/
├── static/gestion_achats/
│ ├── css/
│ │ └── gac_styles.css
│ ├── js/
│ │ ├── demande.js
│ │ ├── bon_commande.js
│ │ └── validation.js
│ └── images/
├── management/commands/
│ ├── init_categories_achats.py
│ ├── verifier_budgets.py
│ └── rappel_commandes.py
└── tests/
├── test_models.py
├── test_services.py
├── test_views.py
└── test_workflows.py
```

2.2 Pattern architectural

- **Service Layer Pattern** : Toute la logique métier dans les services
- **Repository Pattern** : Services comme couche d'accès aux données
- **MVT Django** : Models, Views, Templates
- **Signal-based events** : Pour les notifications automatiques

---

3. Modèles de données