PM

MODULE PROJECT MANAGEMENT

Guide d'Implémentation

README_IMPLEMENTATION.md

HR_ONIAN · Spécification Technique · Février 2026

Guide d'implémentation - Module GAC (Gestion des Achats & Commandes)

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

📋 Vue d'ensemble

Ce document est le guide principal pour l'implémentation complète du module GAC. Il résume l'architecture et fournit un plan d'implémentation étape par étape.

📂 Documents de spécifications

Le module GAC est spécifié dans 6 documents détaillés :

1. **MODELS_SPEC.md** - Spécifications des modèles de données (12 modèles)
2. **SERVICES_SPEC.md** - Spécifications des services métier (8 services)
3. **WORKFLOWS_SPEC.md** - Spécifications des workflows (4 workflows principaux)
4. **VIEWS_FORMS_SPEC.md** - Spécifications des vues et formulaires (40+ vues)
5. **PERMISSIONS_SPEC.md** - Spécifications des permissions (7 rôles)
6. **TEMPLATES_SPEC.md** - Spécifications des templates HTML (30+ templates)

Lire ces documents avant de commencer l'implémentation !

🎯 Objectifs du module

Fonctionnalités principales

✅ **Demandes d'achat** avec workflow de validation hiérarchique (N1, N2)
✅ **Bons de commande** avec génération PDF et envoi aux fournisseurs
✅ **Gestion des fournisseurs** avec évaluation et statistiques
✅ **Réceptions de marchandises** avec contrôle de conformité
✅ **Catalogue produits** hiérarchique
✅ **Contrôle budgétaire** avec alertes automatiques
✅ **Notifications** automatiques à chaque étape
✅ **Historique complet** de toutes les opérations

🏗️ Architecture

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

Structure du projet

```
gestion_achats/
├── __init__.py                     # ✅ Créé
├── apps.py                         # À créer
├── models.py                       # À créer (12 modèles)
├── admin.py                        # À créer
├── urls.py                         # À créer
├── permissions.py                  # À créer
├── signals.py                      # À créer
├── validators.py                   # À créer
├── utils.py                        # À créer
├── constants.py                    # À créer
├── exceptions.py                   # À créer
├── decorators.py                   # À créer
├── mixins.py                       # À créer
│
├── services/                       # 8 services
│   ├── __init__.py
│   ├── demande_service.py
│   ├── bon_commande_service.py
│   ├── fournisseur_service.py
│   ├── reception_service.py
│   ├── budget_service.py
│   ├── catalogue_service.py
│   ├── notification_service.py
│   └── historique_service.py
│
├── views/                          # 40+ vues
│   ├── __init__.py
│   ├── demande_views.py
│   ├── bon_commande_views.py
│   ├── fournisseur_views.py
│   ├── reception_views.py
│   ├── catalogue_views.py
│   ├── budget_views.py
│   └── dashboard_views.py
│
├── forms/                          # 15+ formulaires
│   ├── __init__.py
│   ├── demande_forms.py
│   ├── bon_commande_forms.py
│   ├── fournisseur_forms.py
│   ├── reception_forms.py
│   └── budget_forms.py
│
├── templates/gestion_achats/       # 30+ templates
│   ├── base_gac.html
│   ├── dashboard/
│   ├── demande/
│   ├── bon_commande/
│   ├── fournisseur/
│   ├── reception/
│   ├── catalogue/
│   ├── budget/
│   ├── pdf/
│   └── includes/
│
├── static/gestion_achats/
│   ├── css/
│   │   └── gac_styles.css
│   ├── js/
│   │   ├── gac_common.js
│   │   ├── demande.js
│   │   ├── bon_commande.js
│   │   └── dashboard_charts.js
│   └── images/
│
├── templatetags/
│   ├── __init__.py
│   └── gac_permissions.py
│
├── management/commands/
│   ├── init_roles_gac.py
│   ├── init_categories_achats.py
│   ├── verifier_budgets.py
│   ├── verifier_delais_validation.py
│   ├── verifier_delais_livraison.py
│   └── rappel_receptions_en_attente.py
│
├── tests/
│   ├── __init__.py
│   ├── test_models.py
│   ├── test_services.py
```

```
│   ├── test_views.py
│   ├── test_workflows.py
│   └── test_permissions.py
│
├── migrations/
│   └── __init__.py
│
└── SPECIFICATIONS/                # Documentation complète
    ├── MODELS_SPEC.md
    ├── SERVICES_SPEC.md
    ├── WORKFLOWS_SPEC.md
    ├── VIEWS_FORMS_SPEC.md
    ├── PERMISSIONS_SPEC.md
    ├── TEMPLATES_SPEC.md
    ├── SPECIFICATIONS_TECHNIQUES.md
    └── README_IMPLEMENTATION.md (ce fichier)
```

📊 Modèles de données (12 modèles)

Modèle	Description	Relations
`GACFournisseur`	Fournisseurs	-
`GACCategorie`	Catégories de produits	Self (parent)
`GACArticle`	Articles du catalogue	Categorie, Fournisseurs (M2M)
`GACDemandeAchat`	Demandes d'achat	Demandeur, Validateurs, Budget, Projet
`GACLigneDemandeAchat`	Lignes de demande	DemandeAchat, Article
`GACBonCommande`	Bons de commande	DemandeAchat, Fournisseur, Acheteur
`GACLigneBonCommande`	Lignes de BC	BonCommande, Article
`GACReception`	Réceptions	BonCommande, Receptionnaire
`GACLigneReception`	Lignes de réception	Reception, LigneBonCommande
`GACBudget`	Enveloppes budgétaires	Gestionnaire, Departement
`GACPieceJointe`	Pièces jointes	GenericForeignKey
`GACHistorique`	Historique	GenericForeignKey, Utilisateur

Total : ~120 champs au total

🔄 Workflows principaux

1. Workflow Demande d'achat

```
BROUILLON → SOUMISE → VALIDEE_N1 → VALIDEE_N2 → CONVERTIE_BC
              ↓          ↓            ↓
           REFUSEE    REFUSEE      REFUSEE
              ↓
           ANNULEE
```

2. Workflow Bon de commande

```
BROUILLON → EMIS → ENVOYE → CONFIRME → RECU_PARTIEL → RECU_COMPLET
              ↓       ↓          ↓
           ANNULE  ANNULE     ANNULE
```

3. Workflow Budget

```
Disponible → Engagé → Commandé → Consommé
    ↑           ↓
    └──────────── Libéré (annulation)
```

🔐 Rôles et permissions

Rôle	Niveau	Principales permissions
`DEMANDEUR`	Base	Créer et modifier ses demandes
`VALIDATEUR_N1`	Manager	Valider N1 les demandes de son équipe
`VALIDATEUR_N2`	Direction	Valider N2 les demandes importantes
`ACHETEUR`	Expert	Créer/gérer BCs, convertir demandes, gérer fournisseurs
`RECEPTIONNAIRE`	Logistique	Créer et valider réceptions
`GESTIONNAIRE_BUDGET`	Finance	Gérer budgets et alertes
`ADMIN_GAC`	Admin	Tous les droits

📝 Plan d'implémentation

Phase 1 : Fondations (2-3 jours)

Étape 1.1 : Configuration de base

- [ ] Créer `apps.py`
- [ ] Créer `constants.py` (choix, seuils, etc.)
- [ ] Créer `exceptions.py` (exceptions personnalisées)
- [ ] Créer `utils.py` (fonctions utilitaires)
- [ ] Configurer l'app dans `settings.py`

Étape 1.2 : Modèles de base

- [ ] Créer `GACFournisseur`
- [ ] Créer `GACCategorie`
- [ ] Créer `GACArticle`
- [ ] Créer `GACBudget`
- [ ] Créer et exécuter les migrations
- [ ] Tester la création manuelle de données

Étape 1.3 : Admin de base

- [ ] Configurer `admin.py` pour les modèles de base
- [ ] Tester l'ajout de données via l'admin

Phase 2 : Demandes d'achat (3-4 jours)

Étape 2.1 : Modèles demandes

- [ ] Créer `GACDemandeAchat`
- [ ] Créer `GACLigneDemandeAchat`
- [ ] Créer `GACHistorique`
- [ ] Créer `GACPieceJointe`
- [ ] Créer et exécuter les migrations

Étape 2.2 : Services demandes

- [ ] Créer `demande_service.py`
- creer_demande_brouillon()
- ajouter_ligne()
- soumettre_demande()
- valider_n1()
- valider_n2()
- refuser_demande()
- annuler_demande()
- [ ] Créer `historique_service.py`
- [ ] Créer `budget_service.py` (version basique)
- [ ] Tester les services unitairement

Étape 2.3 : Vues et formulaires demandes

- [ ] Créer `demande_forms.py` (tous les formulaires)
- [ ] Créer `demande_views.py` (toutes les vues)
- [ ] Configurer `urls.py`
- [ ] Créer le routing dans le projet principal

Étape 2.4 : Templates demandes

- [ ] Créer `base_gac.html`
- [ ] Créer tous les templates de demande
- [ ] Créer les includes réutilisables
- [ ] Créer le CSS de base

Étape 2.5 : Permissions demandes

- [ ] Créer `permissions.py`
- [ ] Créer `decorators.py`
- [ ] Créer `mixins.py`
- [ ] Créer le template tag `gac_permissions.py`
- [ ] Appliquer les permissions aux vues

Étape 2.6 : Tests demandes

- [ ] Créer `test_models.py` (demandes)
- [ ] Créer `test_services.py` (demandes)
- [ ] Créer `test_workflows.py` (workflow demandes)
- [ ] Créer `test_permissions.py` (permissions demandes)
- [ ] Exécuter tous les tests

Phase 3 : Bons de commande (3-4 jours)

Étape 3.1 : Modèles BCs

- [ ] Créer `GACBonCommande`
- [ ] Créer `GACLigneBonCommande`
- [ ] Créer et exécuter les migrations

Étape 3.2 : Services BCs

- [ ] Créer `bon_commande_service.py`
- creer_bon_commande()
- emettre_bon_commande()
- envoyer_au_fournisseur()
- confirmer_commande()
- annuler_bon_commande()

- generer_pdf_bon_commande()
- [ ] Améliorer `budget_service.py` (transferts engagé → commandé)
- [ ] Tester les services

Étape 3.3 : Génération PDF

- [ ] Installer WeasyPrint ou ReportLab
- [ ] Créer le template PDF `bon_commande.html`
- [ ] Tester la génération PDF

Étape 3.4 : Vues et formulaires BCs

- [ ] Créer `bon_commande_forms.py`
- [ ] Créer `bon_commande_views.py`
- [ ] Ajouter les routes dans `urls.py`

Étape 3.5 : Templates BCs

- [ ] Créer tous les templates de BC
- [ ] Tester l'affichage

Étape 3.6 : Tests BCs

- [ ] Tests unitaires modèles
- [ ] Tests services
- [ ] Tests workflow BC
- [ ] Tests permissions BC

Phase 4 : Réceptions (2-3 jours)

Étape 4.1 : Modèles réceptions

- [ ] Créer `GACReception`
- [ ] Créer `GACLigneReception`
- [ ] Créer et exécuter les migrations

Étape 4.2 : Services réceptions

- [ ] Créer `reception_service.py`
- creer_reception()
- enregistrer_ligne_reception()
- valider_reception()
- [ ] Améliorer `budget_service.py` (consommation)
- [ ] Tester les services

Étape 4.3 : Vues, formulaires et templates

- [ ] Créer `reception_forms.py`
- [ ] Créer `reception_views.py`
- [ ] Créer les templates
- [ ] Tester l'ensemble

Étape 4.4 : Tests réceptions

- [ ] Tests unitaires
- [ ] Tests workflow réception
- [ ] Tests intégration avec BCs

Phase 5 : Catalogue et fournisseurs (2 jours)

Étape 5.1 : Services catalogue

- [ ] Créer `catalogue_service.py`

- creer_categorie()
- creer_article()
- associer_fournisseur_article()
- rechercher_articles()
- [ ] Créer `fournisseur_service.py`
- creer_fournisseur()
- evaluer_fournisseur()
- get_statistiques_fournisseur()

Étape 5.2 : Vues et templates

- [ ] Créer `catalogue_views.py`
- [ ] Créer `fournisseur_views.py`
- [ ] Créer les templates
- [ ] Créer l'API AJAX de recherche d'articles

Étape 5.3 : Tests

- [ ] Tests services catalogue
- [ ] Tests services fournisseurs

Phase 6 : Budgets (2 jours)

Étape 6.1 : Services budgets (complet)

- [ ] Finaliser `budget_service.py`
- verifier_disponibilite()
- engager_montant()
- commander_montant()
- consommer_montant()
- liberer_montant()
- _verifier_seuils_alerte()
- get_synthese_budgets()

Étape 6.2 : Vues et templates budgets

- [ ] Créer `budget_views.py`
- [ ] Créer `budget_forms.py`
- [ ] Créer les templates
- [ ] Créer les graphiques (Chart.js)

Étape 6.3 : Tests budgets

- [ ] Tests logique budgétaire
- [ ] Tests alertes
- [ ] Tests statistiques

Phase 7 : Notifications et signaux (1-2 jours)

Étape 7.1 : Service de notifications

- [ ] Créer `notification_service.py`
- notifier_validation_n1()
- notifier_validation_n2()
- notifier_demande_validee()
- notifier_demande_refusee()
- notifier_bc_cree_depuis_demande()
- notifier_reception_validee()
- notifier_alerte_budget()

Étape 7.2 : Signaux Django

- [ ] Créer `signals.py`
- Signal post_save sur GACDemandeAchat
- Signal post_save sur GACBonCommande
- Signal post_save sur GACReception
- Signal post_save sur GACBudget

Étape 7.3 : Tests notifications

- [ ] Tests création notifications
- [ ] Tests envoi emails
- [ ] Tests déclenchement signaux

Phase 8 : Dashboard et reporting (2 jours)

Étape 8.1 : Dashboard principal

- [ ] Créer `dashboard_views.py`
- [ ] Créer le template dashboard
- [ ] Créer les cartes de statistiques
- [ ] Créer les graphiques Chart.js

Étape 8.2 : JavaScript et CSS

- [ ] Créer `gac_common.js`
- [ ] Créer `dashboard_charts.js`
- [ ] Créer `demande.js`
- [ ] Créer `bon_commande.js`
- [ ] Finaliser `gac_styles.css`

Phase 9 : Tâches planifiées (1 jour)

Étape 9.1 : Management commands

- [ ] Créer `init_roles_gac.py`
- [ ] Créer `init_categories_achats.py`
- [ ] Créer `verifier_budgets.py`
- [ ] Créer `verifier_delais_validation.py`
- [ ] Créer `verifier_delais_livraison.py`
- [ ] Créer `rappel_receptions_en_attente.py`

Étape 9.2 : Tester les commands

- [ ] Tester chaque command individuellement
- [ ] Documenter l'ordonnancement (cron, systemd, Celery)

Phase 10 : Tests finaux et documentation (2-3 jours)

Étape 10.1 : Tests d'intégration

- [ ] Tests de bout en bout (demande → BC → réception)
- [ ] Tests avec utilisateurs de différents rôles
- [ ] Tests de performance (avec volumes de données)

Étape 10.2 : Fixtures de démo

- [ ] Créer fixture de fournisseurs
- [ ] Créer fixture de catégories et articles
- [ ] Créer fixture de budgets

- [ ] Créer fixture de demandes et BCs (démo)

Étape 10.3 : Documentation utilisateur

- [ ] Guide utilisateur demandeur
- [ ] Guide utilisateur validateur
- [ ] Guide utilisateur acheteur
- [ ] Guide administrateur

Étape 10.4 : Déploiement

- [ ] Vérifier les settings de production
- [ ] Créer les migrations de production
- [ ] Initialiser les rôles
- [ ] Initialiser les catégories
- [ ] Former les utilisateurs clés

🧪 Tests

Couverture de tests cible

- **Modèles** : 100% (tous les champs, méthodes, propriétés)
- **Services** : 95% (toutes les méthodes, cas nominaux et erreurs)
- **Vues** : 80% (principales fonctionnalités)
- **Workflows** : 100% (toutes les transitions d'états)
- **Permissions** : 100% (tous les rôles et actions)

Commandes de test

```
# Tous les tests du module
python manage.py test gestion_achats

# Tests spécifiques
python manage.py test gestion_achats.tests.test_models
python manage.py test gestion_achats.tests.test_services
python manage.py test gestion_achats.tests.test_workflows
python manage.py test gestion_achats.tests.test_permissions

# Avec couverture
coverage run --source='gestion_achats' manage.py test gestion_achats
coverage report
coverage html
```

📦 Dépendances

Packages Python à ajouter

```
# PDF generation
pip install WeasyPrint
# ou
pip install reportlab

# Pour les graphiques (optionnel, peut utiliser Chart.js)
pip install plotly

# Select2 pour autocomplétion (déjà dans le projet si Bootstrap)
# Chart.js pour graphiques (CDN)
```

Ajouter dans `requirements.txt`

```
WeasyPrint==60.0
```

⚙️ Configuration

Dans `settings.py`

```
INSTALLED_APPS = [
    # ... apps existantes
    'gestion_achats',
]

# Configuration GAC
GAC_SEUIL_VALIDATION_N2 = 5000  # Montant TTC au-delà duquel validation N2 obligatoire
GAC_DELAI_SLA_VALIDATION = 5    # Jours ouvrés max pour validation complète
GAC_TAUX_TVA_DEFAUT = 20.0      # Taux TVA par défaut (%)

# Email
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
DEFAULT_FROM_EMAIL = 'noreply@hr-onian.com'

# URL du site (pour les notifications)
SITE_URL = 'https://hr-onian.com'

# Entreprise (pour les PDFs)
COMPANY_NAME = 'HR ONIAN'
COMPANY_ADDRESS = '...'
COMPANY_SIRET = '...'
COMPANY_LOGO_PATH = os.path.join(STATIC_ROOT, 'images/logo.png')
```

Dans `urls.py` du projet

```
urlpatterns = [
    # ... autres URLs
    path('gestion-achats/', include('gestion_achats.urls')),
]
```

📊 Métriques du projet

Estimation du code

Type	Nombre	Lignes estimées
Modèles	12	~1500
Services	8	~2500
Vues	40	~2000
Formulaires	15	~800
Templates	30	~3000
Tests	20 fichiers	~2000
JavaScript	5 fichiers	~500
CSS	1 fichier	~300
TOTAL		**~12 600 lignes**

Temps d'implémentation estimé

- **Développement** : 20-25 jours
- **Tests** : 5 jours
- **Documentation** : 3 jours
- **Déploiement et formation** : 2 jours

Total : **30-35 jours** (1,5 à 2 mois)

🚀 Mise en production

Checklist avant déploiement

- [ ] Tous les tests passent
- [ ] Migrations créées et testées
- [ ] Settings de production configurés
- [ ] Rôles initialisés
- [ ] Catégories initialisées
- [ ] Données de démo créées (optionnel)
- [ ] Documentation utilisateur rédigée
- [ ] Formation des utilisateurs clés effectuée
- [ ] Backup de la base avant migration
- [ ] Plan de rollback préparé

Commandes de déploiement

```
# 1. Sauvegarder la base
python manage.py dumpdata > backup_before_gac.json

# 2. Exécuter les migrations
python manage.py migrate gestion_achats

# 3. Initialiser les rôles
python manage.py init_roles_gac

# 4. Initialiser les catégories
python manage.py init_categories_achats

# 5. Créer un superuser GAC si nécessaire
python manage.py createsuperuser

# 6. Collecter les fichiers statiques
python manage.py collectstatic --noinput

# 7. Redémarrer l'application
systemctl restart hr-onian
```

📞 Support et maintenance

Contacts

- **Développeur principal** : [Nom]
- **Chef de projet** : [Nom]
- **Admin système** : [Nom]

Ressources

- **Documentation complète** : `/gestion_achats/SPECIFICATIONS/`
- **Issues GitHub** : [URL si applicable]
- **Wiki interne** : [URL si applicable]

🔄 Évolutions futures

Version 2.0 (à planifier)

- [ ] Intégration EDI (Échange de Données Informatisé)
- [ ] Gestion des appels d'offres
- [ ] Gestion des stocks complète
- [ ] Module de facturation fournisseur

- [ ] Analyse prédictive des achats
- [ ] API REST complète
- [ ] Application mobile

✅ Checklist de validation finale

Fonctionnalités

- [ ] Création de demande d'achat
- [ ] Workflow de validation complet (N1, N2)
- [ ] Conversion demande → BC
- [ ] Génération PDF de BC
- [ ] Envoi email au fournisseur
- [ ] Création de réception
- [ ] Validation de réception avec conformité
- [ ] Contrôle budgétaire (engagement, commande, consommation)
- [ ] Alertes budgétaires (seuils 1 et 2)
- [ ] Gestion des fournisseurs et évaluations
- [ ] Catalogue produits hiérarchique
- [ ] Dashboard avec statistiques et graphiques
- [ ] Historique complet de toutes les opérations
- [ ] Notifications automatiques (email + in-app)
- [ ] Permissions par rôle
- [ ] Tâches planifiées (vérifications, rappels)

Qualité

- [ ] Code conforme PEP8
- [ ] Couverture de tests > 80%
- [ ] Pas de vulnérabilités de sécurité
- [ ] Performance acceptable (< 2s pour pages principales)
- [ ] Interface responsive (mobile, tablette, desktop)
- [ ] Accessibilité WCAG 2.1 niveau AA
- [ ] Documentation complète et à jour

Bon courage pour l'implémentation ! 🚀

Pour toute question, consulter les spécifications détaillées ou contacter l'équipe de développement.