PM

MODULE PROJECT MANAGEMENT

Spécification des Permissions & Rôles

PERMISSIONS_SPEC.md

HR_ONIAN · Spécification Technique · Février 2026

Spécifications des Permissions - Module GAC

Version: 1.0
Date: 01/02/2026
Projet: HR_ONIAN
Module: Gestion des Achats & Commandes

Vue d'ensemble

Ce document spécifie le système de permissions et de rôles du module GAC, ainsi que les règles d'accès pour chaque fonctionnalité.

---

1. Rôles du module GAC

1.1 Liste des rôles

Les rôles suivants doivent être créés dans le modèle `ZYRO` (table des rôles) :

Code	Libellé	Description
`DEMANDEUR`	Demandeur	Tout employé pouvant créer une demande d'achat (rôle par défaut)
`VALIDATEUR_N1`	Validateur niveau 1	Manager validant les demandes de son équipe
`VALIDATEUR_N2`	Validateur niveau 2	Direction ou responsable achats validant les demandes importantes
`ACHETEUR`	Acheteur	Personne en charge de créer et gérer les bons de commande
`RECEPTIONNAIRE`	Réceptionnaire	Personne habilitée à réceptionner les marchandises
`GESTIONNAIRE_BUDGET`	Gestionnaire de budget	Personne gérant les enveloppes budgétaires
`ADMIN_GAC`	Administrateur GAC	Administrateur complet du module

1.2 Hiérarchie des rôles

```
ADMIN_GAC (tous les droits)
    │
    ├── GESTIONNAIRE_BUDGET (gestion budgets)
    │
    ├── ACHETEUR (gestion BCs + validation N2)
    │   │
    │   └── VALIDATEUR_N2 (validation niveau 2)
    │       │
    │       └── VALIDATEUR_N1 (validation niveau 1)
    │           │
    │           └── RECEPTIONNAIRE (réceptions)
    │               │
    │               └── DEMANDEUR (création demandes)
```

Note : Un utilisateur peut avoir plusieurs rôles simultanément.

---

2. Matrice de permissions

2.1 Demandes d'achat

Action	DEMANDEUR	VALIDATEUR_N1	VALIDATEUR_N2	ACHETEUR	ADMIN_GAC
Créer demande	✅ Oui	✅ Oui	✅ Oui	✅ Oui	✅ Oui
Voir ses demandes	✅ Oui	✅ Oui	✅ Oui	✅ Oui	✅ Oui
Voir toutes les demandes	❌ Non	❌ Non	❌ Non	✅ Oui	✅ Oui
Modifier demande (brouillon)	✅ Si demandeur	✅ Si demandeur	✅ Si demandeur	✅ Oui	✅ Oui
Soumettre demande	✅ Si demandeur	✅ Si demandeur	✅ Si demandeur	✅ Si demandeur	✅ Oui
Valider N1	❌ Non	✅ Si validateur_n1	❌ Non	❌ Non	✅ Oui
Valider N2	❌ Non	❌ Non	✅ Si validateur_n2	✅ Oui	✅ Oui
Refuser demande	❌ Non	✅ Si validateur_n1	✅ Si validateur_n2	✅ Si validateur_n2	✅ Oui
Annuler demande	✅ Si demandeur	✅ Si demandeur	✅ Si demandeur	✅ Oui	✅ Oui
Convertir en BC	❌ Non	❌ Non	❌ Non	✅ Oui	✅ Oui
Supprimer demande	❌ Non	❌ Non	❌ Non	❌ Non	✅ Oui

2.2 Bons de commande

Action	DEMANDEUR	ACHETEUR	RECEPTIONNAIRE	ADMIN_GAC
Créer BC	❌ Non	✅ Oui	❌ Non	✅ Oui
Voir ses BCs (via demandes)	✅ Oui	✅ Oui	✅ Oui	✅ Oui
Voir tous les BCs	❌ Non	✅ Oui	❌ Non	✅ Oui
Modifier BC (brouillon)	❌ Non	✅ Oui	❌ Non	✅ Oui
Émettre BC	❌ Non	✅ Oui	❌ Non	✅ Oui
Envoyer BC	❌ Non	✅ Oui	❌ Non	✅ Oui
Confirmer BC	❌ Non	✅ Oui	❌ Non	✅ Oui
Annuler BC	❌ Non	✅ Oui	❌ Non	✅ Oui
Télécharger PDF	✅ Si demandeur	✅ Oui	✅ Oui	✅ Oui
Supprimer BC	❌ Non	❌ Non	❌ Non	✅ Oui

2.3 Fournisseurs

Action	DEMANDEUR	ACHETEUR	ADMIN_GAC
Voir liste fournisseurs	✅ Oui	✅ Oui	✅ Oui
Voir détail fournisseur	✅ Oui	✅ Oui	✅ Oui
Créer fournisseur	❌ Non	✅ Oui	✅ Oui
Modifier fournisseur	❌ Non	✅ Oui	✅ Oui
Désactiver fournisseur	❌ Non	✅ Oui	✅ Oui
Évaluer fournisseur	❌ Non	✅ Oui	✅ Oui
Supprimer fournisseur	❌ Non	❌ Non	✅ Oui

2.4 Réceptions

Action	DEMANDEUR	ACHETEUR	RECEPTIONNAIRE	ADMIN_GAC
Créer réception	❌ Non	✅ Oui	✅ Oui	✅ Oui
Voir réceptions	✅ Si BC lié à sa demande	✅ Oui	✅ Oui	✅ Oui
Modifier réception (brouillon)	❌ Non	✅ Oui	✅ Oui	✅ Oui
Valider réception	❌ Non	✅ Oui	✅ Oui	✅ Oui
Annuler réception	❌ Non	✅ Oui	✅ Oui	✅ Oui

2.5 Catalogue

Action	DEMANDEUR	ACHETEUR	ADMIN_GAC
Consulter articles	✅ Oui	✅ Oui	✅ Oui
Créer article	❌ Non	✅ Oui	✅ Oui
Modifier article	❌ Non	✅ Oui	✅ Oui
Désactiver article	❌ Non	✅ Oui	✅ Oui
Créer catégorie	❌ Non	✅ Oui	✅ Oui
Modifier catégorie	❌ Non	✅ Oui	✅ Oui

2.6 Budgets

Action	DEMANDEUR	ACHETEUR	GESTIONNAIRE_BUDGET	ADMIN_GAC
Voir budgets	✅ Les siens	✅ Tous	✅ Les siens	✅ Tous
Créer budget	❌ Non	❌ Non	✅ Oui	✅ Oui
Modifier budget	❌ Non	❌ Non	✅ Si gestionnaire	✅ Oui
Voir consommation	✅ Si lié	✅ Oui	✅ Si gestionnaire	✅ Oui
Modifier seuils alerte	❌ Non	❌ Non	✅ Si gestionnaire	✅ Oui

3. Permissions Django

3.1 Fichier de permissions

Fichier: `gestion_achats/permissions.py`

```
from django.core.exceptions import PermissionDenied

class GACPermissions:
    """Classe pour gérer les permissions du module GAC."""

    # ========== Demandes d'achat ==========

    @staticmethod
    def can_view_demande(user, demande):
        """Vérifie si l'utilisateur peut voir une demande."""
        if user.has_role('ADMIN_GAC') or user.has_role('ACHETEUR'):
            return True

        # L'utilisateur peut voir ses propres demandes
        if demande.demandeur == user:
            return True

        # Les validateurs peuvent voir les demandes qu'ils doivent valider
        if demande.validateur_n1 == user or demande.validateur_n2 == user:
            return True

        return False

    @staticmethod
    def can_create_demande(user):
        """Vérifie si l'utilisateur peut créer une demande."""
        # Tout utilisateur authentifié peut créer une demande
        return user.is_authenticated

    @staticmethod
    def can_modify_demande(user, demande):
        """Vérifie si l'utilisateur peut modifier une demande."""
        if user.has_role('ADMIN_GAC'):
            return True

        # Seul le demandeur peut modifier sa demande en brouillon
        if demande.statut == 'BROUILLON' and demande.demandeur == user:
            return True

        return False

    @staticmethod
    def can_submit_demande(user, demande):
        """Vérifie si l'utilisateur peut soumettre une demande."""
        if user.has_role('ADMIN_GAC'):
            return True

        # Seul le demandeur peut soumettre sa demande en brouillon
        if demande.statut == 'BROUILLON' and demande.demandeur == user:
            return True

        return False

    @staticmethod
    def can_validate_n1(user, demande):
        """Vérifie si l'utilisateur peut valider N1 une demande."""
        if user.has_role('ADMIN_GAC'):
            return True

        # Seul le validateur N1 assigné peut valider
        if demande.statut == 'SOUMISE' and demande.validateur_n1 == user:
            return True

        return False

    @staticmethod
    def can_validate_n2(user, demande):
        """Vérifie si l'utilisateur peut valider N2 une demande."""
        if user.has_role('ADMIN_GAC'):
            return True

        # Le validateur N2 assigné ou un acheteur peut valider
        if demande.statut == 'VALIDEE_N1':
            if demande.validateur_n2 == user or user.has_role('ACHETEUR'):
                return True

        return False

    @staticmethod
    def can_refuse_demande(user, demande):
        """Vérifie si l'utilisateur peut refuser une demande."""
        if user.has_role('ADMIN_GAC'):
```

```
            return True

        # Le validateur N1 peut refuser si demande soumise
        if demande.statut == 'SOUMISE' and demande.validateur_n1 == user:
            return True

        # Le validateur N2 ou acheteur peut refuser si validée N1
        if demande.statut == 'VALIDEE_N1':
            if demande.validateur_n2 == user or user.has_role('ACHETEUR'):
                return True

        return False

    @staticmethod
    def can_cancel_demande(user, demande):
        """Vérifie si l'utilisateur peut annuler une demande."""
        if user.has_role('ADMIN_GAC'):
            return True

        # Le demandeur peut annuler sa demande si pas encore convertie en BC
        if demande.demandeur == user and demande.statut != 'CONVERTIE_BC':
            return True

        return False

    @staticmethod
    def can_convert_to_bc(user, demande):
        """Vérifie si l'utilisateur peut convertir une demande en BC."""
        if demande.statut != 'VALIDEE_N2':
            return False

        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    # ========== Bons de commande ==========

    @staticmethod
    def can_view_bon_commande(user, bc):
        """Vérifie si l'utilisateur peut voir un BC."""
        if user.has_role('ADMIN_GAC') or user.has_role('ACHETEUR'):
            return True

        # L'utilisateur peut voir le BC s'il est lié à une demande qu'il a créée
        if bc.demande_achat and bc.demande_achat.demandeur == user:
            return True

        return False

    @staticmethod
    def can_create_bon_commande(user):
        """Vérifie si l'utilisateur peut créer un BC."""
        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_modify_bon_commande(user, bc):
        """Vérifie si l'utilisateur peut modifier un BC."""
        if bc.statut != 'BROUILLON':
            return False

        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_emit_bon_commande(user, bc):
        """Vérifie si l'utilisateur peut émettre un BC."""
        if bc.statut != 'BROUILLON':
            return False

        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_send_bon_commande(user, bc):
        """Vérifie si l'utilisateur peut envoyer un BC."""
        if bc.statut != 'EMIS':
            return False

        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_confirm_bon_commande(user, bc):
        """Vérifie si l'utilisateur peut confirmer un BC."""
        if bc.statut != 'ENVOYE':
            return False

        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')
```

```
    @staticmethod
    def can_cancel_bon_commande(user, bc):
        """Vérifie si l'utilisateur peut annuler un BC."""
        if bc.statut in ['RECU_COMPLET', 'RECU_PARTIEL']:
            return False

        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_download_pdf(user, bc):
        """Vérifie si l'utilisateur peut télécharger le PDF d'un BC."""
        return GACPermissions.can_view_bon_commande(user, bc)

    # ========== Fournisseurs ==========

    @staticmethod
    def can_view_fournisseur(user):
        """Vérifie si l'utilisateur peut voir les fournisseurs."""
        # Tous les utilisateurs peuvent voir les fournisseurs
        return user.is_authenticated

    @staticmethod
    def can_create_fournisseur(user):
        """Vérifie si l'utilisateur peut créer un fournisseur."""
        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_modify_fournisseur(user):
        """Vérifie si l'utilisateur peut modifier un fournisseur."""
        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_evaluate_fournisseur(user):
        """Vérifie si l'utilisateur peut évaluer un fournisseur."""
        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    # ========== Réceptions ==========

    @staticmethod
    def can_view_reception(user, reception):
        """Vérifie si l'utilisateur peut voir une réception."""
        if user.has_role('ADMIN_GAC') or user.has_role('ACHETEUR'):
            return True

        # Peut voir si BC lié à sa demande
        bc = reception.bon_commande
        if bc.demande_achat and bc.demande_achat.demandeur == user:
            return True

        return False

    @staticmethod
    def can_create_reception(user):
        """Vérifie si l'utilisateur peut créer une réception."""
        return (user.has_role('RECEPTIONNAIRE') or
                user.has_role('ACHETEUR') or
                user.has_role('ADMIN_GAC'))

    @staticmethod
    def can_modify_reception(user, reception):
        """Vérifie si l'utilisateur peut modifier une réception."""
        if reception.statut != 'BROUILLON':
            return False

        return (user.has_role('RECEPTIONNAIRE') or
                user.has_role('ACHETEUR') or
                user.has_role('ADMIN_GAC'))

    @staticmethod
    def can_validate_reception(user, reception):
        """Vérifie si l'utilisateur peut valider une réception."""
        if reception.statut != 'BROUILLON':
            return False

        return (user.has_role('RECEPTIONNAIRE') or
                user.has_role('ACHETEUR') or
                user.has_role('ADMIN_GAC'))

    # ========== Catalogue ==========

    @staticmethod
    def can_view_catalogue(user):
```

```
        """Vérifie si l'utilisateur peut voir le catalogue."""
        return user.is_authenticated

    @staticmethod
    def can_manage_catalogue(user):
        """Vérifie si l'utilisateur peut gérer le catalogue."""
        return user.has_role('ACHETEUR') or user.has_role('ADMIN_GAC')

    # ========== Budgets ==========

    @staticmethod
    def can_view_budget(user, budget):
        """Vérifie si l'utilisateur peut voir un budget."""
        if user.has_role('ADMIN_GAC') or user.has_role('ACHETEUR'):
            return True

        # Peut voir si gestionnaire du budget
        if budget.gestionnaire == user:
            return True

        # Peut voir si une de ses demandes est liée
        from gestion_achats.models import GACDemandeAchat
        if GACDemandeAchat.objects.filter(budget=budget, demandeur=user).exists():
            return True

        return False

    @staticmethod
    def can_create_budget(user):
        """Vérifie si l'utilisateur peut créer un budget."""
        return user.has_role('GESTIONNAIRE_BUDGET') or user.has_role('ADMIN_GAC')

    @staticmethod
    def can_modify_budget(user, budget):
        """Vérifie si l'utilisateur peut modifier un budget."""
        if user.has_role('ADMIN_GAC'):
            return True

        # Peut modifier si gestionnaire du budget
        if budget.gestionnaire == user and user.has_role('GESTIONNAIRE_BUDGET'):
            return True

        return False


# Fonction helper pour vérifier les permissions avec exception
def require_permission(permission_func, *args):
    """
    Vérifie une permission et lève PermissionDenied si refusée.

    Usage:
        require_permission(GACPermissions.can_view_demande, request.user, demande)
    """
    if not permission_func(*args):
        raise PermissionDenied
```

3.2 Utilisation dans les vues

```
# Dans une vue
from gestion_achats.permissions import GACPermissions, require_permission

def demande_detail(request, pk):
    demande = get_object_or_404(GACDemandeAchat, pk=pk)

    # Vérifier la permission
    require_permission(GACPermissions.can_view_demande, request.user, demande)

    # Ou manuellement
    if not GACPermissions.can_view_demande(request.user, demande):
        raise PermissionDenied

    # ... reste de la vue
```

3.3 Utilisation dans les templates

```
{# Dans un template #}
{% load gac_permissions %}

{% if can_modify_demande user demande %}
    <a href="{% url 'gestion_achats:demande_update' demande.pk %}">Modifier</a>
{% endif %}

{% if can_validate_n1 user demande %}
    <button type="submit" name="action" value="valider_n1">Valider (N1)</button>
{% endif %}
```

3.4 Template tags personnalisés

Fichier: `gestion_achats/templatetags/gac_permissions.py`

```
from django import template
from gestion_achats.permissions import GACPermissions

register = template.Library()

@register.simple_tag
def can_modify_demande(user, demande):
    return GACPermissions.can_modify_demande(user, demande)

@register.simple_tag
def can_submit_demande(user, demande):
    return GACPermissions.can_submit_demande(user, demande)

@register.simple_tag
def can_validate_n1(user, demande):
    return GACPermissions.can_validate_n1(user, demande)

@register.simple_tag
def can_validate_n2(user, demande):
    return GACPermissions.can_validate_n2(user, demande)

@register.simple_tag
def can_refuse_demande(user, demande):
    return GACPermissions.can_refuse_demande(user, demande)

@register.simple_tag
def can_convert_to_bc(user, demande):
    return GACPermissions.can_convert_to_bc(user, demande)

# ... autres template tags
```

---

4. Décorateurs de permission

Fichier: `gestion_achats/decorators.py`

```python
from functools import wraps
from django.core.exceptions import PermissionDenied
from django.shortcuts import get_object_or_404

def require_role(role_code):
    """Décorateur pour vérifier qu'un utilisateur a un rôle spécifique."""
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            if not request.user.has_role(role_code):
                raise PermissionDenied(f"Rôle requis: {role_code}")
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator


def require_any_role(*role_codes):
    """Décorateur pour vérifier qu'un utilisateur a au moins un des rôles spécifiés."""
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            if not any(request.user.has_role(role) for role in role_codes):
                raise PermissionDenied(f"Un de ces rôles est requis: {', '.join(role_codes)}")
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator


def require_demande_access(view_func):
    """Décorateur pour vérifier l'accès à une demande."""
    @wraps(view_func)
    def _wrapped_view(request, pk, *args, **kwargs):
        from gestion_achats.models import GACDemandeAchat
        from gestion_achats.permissions import GACPermissions

        demande = get_object_or_404(GACDemandeAchat, pk=pk)

        if not GACPermissions.can_view_demande(request.user, demande):
            raise PermissionDenied("Vous n'avez pas accès à cette demande")

        # Passer la demande à la vue pour éviter une double requête
        kwargs['demande'] = demande

        return view_func(request, pk, *args, **kwargs)
    return _wrapped_view


# Exemples d'utilisation
@login_required
@require_role('ACHETEUR')
def create_bon_commande(request):
    ...

@login_required
@require_any_role('ACHETEUR', 'RECEPTIONNAIRE')
def create_reception(request):
    ...

@login_required
@require_demande_access
def demande_detail(request, pk, demande):
    # La demande est déjà chargée et passée en paramètre
    ...
```

5. Mixins pour les Class-Based Views

Fichier: `gestion_achats/mixins.py`

```python
from django.core.exceptions import PermissionDenied
from django.contrib.auth.mixins import LoginRequiredMixin

class RoleRequiredMixin(LoginRequiredMixin):
    """Mixin pour vérifier qu'un utilisateur a un rôle spécifique."""
    required_role = None

    def dispatch(self, request, *args, **kwargs):
        if not request.user.has_role(self.required_role):
            raise PermissionDenied(f"Rôle requis: {self.required_role}")
        return super().dispatch(request, *args, **kwargs)


class AnyRoleRequiredMixin(LoginRequiredMixin):
    """Mixin pour vérifier qu'un utilisateur a au moins un des rôles spécifiés."""
    required_roles = []

    def dispatch(self, request, *args, **kwargs):
        if not any(request.user.has_role(role) for role in self.required_roles):
            raise PermissionDenied(f"Un de ces rôles est requis: {', '.join(self.required_roles)}")
        return super().dispatch(request, *args, **kwargs)


class DemandeAccessMixin(LoginRequiredMixin):
    """Mixin pour vérifier l'accès à une demande."""

    def dispatch(self, request, *args, **kwargs):
        from gestion_achats.permissions import GACPermissions

        demande = self.get_object()

        if not GACPermissions.can_view_demande(request.user, demande):
            raise PermissionDenied("Vous n'avez pas accès à cette demande")

        return super().dispatch(request, *args, **kwargs)


class BonCommandeAccessMixin(LoginRequiredMixin):
    """Mixin pour vérifier l'accès à un bon de commande."""

    def dispatch(self, request, *args, **kwargs):
        from gestion_achats.permissions import GACPermissions

        bc = self.get_object()

        if not GACPermissions.can_view_bon_commande(request.user, bc):
            raise PermissionDenied("Vous n'avez pas accès à ce bon de commande")

        return super().dispatch(request, *args, **kwargs)


# Exemple d'utilisation
class BonCommandeCreateView(RoleRequiredMixin, CreateView):
    required_role = 'ACHETEUR'
    model = GACBonCommande
    ...

class ReceptionCreateView(AnyRoleRequiredMixin, CreateView):
    required_roles = ['ACHETEUR', 'RECEPTIONNAIRE']
    model = GACReception
    ...

class DemandeDetailView(DemandeAccessMixin, DetailView):
    model = GACDemandeAchat
    ...
```

6. Méthode has_role sur le modèle ZY00

Pour que le système de permissions fonctionne, il faut s'assurer que le modèle `ZY00` (employé) dispose d'une méthode `has_role()`.

Fichier: `employee/models.py`

```python
class ZY00(models.Model):
    # ... autres champs

    def has_role(self, role_code):
        """
        Vérifie si l'employé a un rôle spécifique actif.

        Args:
            role_code (str): Code du rôle (ex: 'ACHETEUR', 'DRH', etc.)

        Returns:
            bool: True si l'employé a le rôle actif
        """
        return self.roles_attribues.filter(
            role__CODE=role_code,
            actif=True
        ).exists()

    def get_roles(self):
        """
        Récupère tous les rôles actifs de l'employé.

        Returns:
            QuerySet: QuerySet des rôles actifs
        """
        return ZYRO.objects.filter(
            id__in=self.roles_attribues.filter(actif=True).values_list('role_id', flat=True)
        )

    def has_any_role(self, *role_codes):
        """
        Vérifie si l'employé a au moins un des rôles spécifiés.

        Args:
            *role_codes: Codes des rôles à vérifier

        Returns:
            bool: True si l'employé a au moins un des rôles
        """
        return self.roles_attribues.filter(
            role__CODE__in=role_codes,
            actif=True
        ).exists()
```

---

7. Initialisation des rôles

7.1 Management command

Fichier: `gestion_achats/management/commands/init_roles_gac.py`

```
from django.core.management.base import BaseCommand
from employee.models import ZYRO


class Command(BaseCommand):
    help = 'Initialise les rôles du module GAC'

    def handle(self, *args, **options):
        roles = [
            {
                'CODE': 'DEMANDEUR',
                'LIBELLE': 'Demandeur',
                'DESCRIPTION': 'Tout employé pouvant créer une demande d\'achat'
            },
            {
                'CODE': 'VALIDATEUR_N1',
                'LIBELLE': 'Validateur niveau 1',
                'DESCRIPTION': 'Manager validant les demandes de son équipe'
            },
            {
                'CODE': 'VALIDATEUR_N2',
                'LIBELLE': 'Validateur niveau 2',
                'DESCRIPTION': 'Direction ou responsable achats validant les demandes importantes'
            },
            {
                'CODE': 'ACHETEUR',
                'LIBELLE': 'Acheteur',
                'DESCRIPTION': 'Personne en charge de créer et gérer les bons de commande'
            },
            {
                'CODE': 'RECEPTIONNAIRE',
                'LIBELLE': 'Réceptionnaire',
                'DESCRIPTION': 'Personne habilitée à réceptionner les marchandises'
            },
            {
                'CODE': 'GESTIONNAIRE_BUDGET',
                'LIBELLE': 'Gestionnaire de budget',
                'DESCRIPTION': 'Personne gérant les enveloppes budgétaires'
            },
            {
                'CODE': 'ADMIN_GAC',
                'LIBELLE': 'Administrateur GAC',
                'DESCRIPTION': 'Administrateur complet du module'
            },
        ]

        created_count = 0
        updated_count = 0

        for role_data in roles:
            role, created = ZYRO.objects.update_or_create(
                CODE=role_data['CODE'],
                defaults={
                    'LIBELLE': role_data['LIBELLE'],
                    'DESCRIPTION': role_data['DESCRIPTION']
                }
            )

            if created:
                created_count += 1
                self.stdout.write(self.style.SUCCESS(f"✅ Rôle créé: {role.CODE} - {role.LIBELLE}"))
            else:
                updated_count += 1
                self.stdout.write(self.style.WARNING(f"⚠️  Rôle mis à jour: {role.CODE} - {role.LIBELLE}"))

        self.stdout.write(self.style.SUCCESS(f"\n{'='*70}"))
        self.stdout.write(self.style.SUCCESS(f"Résumé:"))
        self.stdout.write(self.style.SUCCESS(f"  - Rôles créés: {created_count}"))
        self.stdout.write(self.style.SUCCESS(f"  - Rôles mis à jour: {updated_count}"))
        self.stdout.write(self.style.SUCCESS(f"{'='*70}"))
```

Exécution :

```
python manage.py init_roles_gac
```

---

8. Tests des permissions

Fichier: `gestion_achats/tests/test_permissions.py`

```python
from django.test import TestCase
from django.core.exceptions import PermissionDenied
from employee.models import ZY00, ZYRO, ZYAT
from gestion_achats.models import GACDemandeAchat
from gestion_achats.permissions import GACPermissions


class PermissionsTestCase(TestCase):
    def setUp(self):
        # Créer les rôles
        self.role_acheteur = ZYRO.objects.create(CODE='ACHETEUR', LIBELLE='Acheteur')
        self.role_demandeur = ZYRO.objects.create(CODE='DEMANDEUR', LIBELLE='Demandeur')

        # Créer les utilisateurs
        self.demandeur = ZY00.objects.create(
            matricule='TEST001',
            nom='DUPONT',
            prenoms='Jean'
        )

        self.acheteur = ZY00.objects.create(
            matricule='TEST002',
            nom='MARTIN',
            prenoms='Marie'
        )

        # Assigner les rôles
        ZYAT.objects.create(
            employe=self.demandeur,
            role=self.role_demandeur,
            actif=True
        )

        ZYAT.objects.create(
            employe=self.acheteur,
            role=self.role_acheteur,
            actif=True
        )

        # Créer une demande
        self.demande = GACDemandeAchat.objects.create(
            numero='DA-2026-0001',
            demandeur=self.demandeur,
            objet='Test',
            justification='Test',
            statut='BROUILLON'
        )

    def test_demandeur_can_view_own_demande(self):
        """Le demandeur peut voir sa propre demande."""
        self.assertTrue(
            GACPermissions.can_view_demande(self.demandeur, self.demande)
        )

    def test_acheteur_can_view_any_demande(self):
        """L'acheteur peut voir toutes les demandes."""
        self.assertTrue(
            GACPermissions.can_view_demande(self.acheteur, self.demande)
        )

    def test_demandeur_can_modify_own_brouillon(self):
        """Le demandeur peut modifier sa demande en brouillon."""
        self.assertTrue(
            GACPermissions.can_modify_demande(self.demandeur, self.demande)
        )

    def test_demandeur_cannot_modify_submitted_demande(self):
        """Le demandeur ne peut pas modifier une demande soumise."""
        self.demande.statut = 'SOUMISE'
        self.demande.save()

        self.assertFalse(
            GACPermissions.can_modify_demande(self.demandeur, self.demande)
        )

    def test_acheteur_can_convert_to_bc(self):
        """L'acheteur peut convertir une demande validée en BC."""
        self.demande.statut = 'VALIDEE_N2'
        self.demande.save()

        self.assertTrue(
            GACPermissions.can_convert_to_bc(self.acheteur, self.demande)
```

```
        )

    def test_demandeur_cannot_convert_to_bc(self):
        """Le demandeur ne peut pas convertir en BC."""
        self.demande.statut = 'VALIDEE_N2'
        self.demande.save()

        self.assertFalse(
            GACPermissions.can_convert_to_bc(self.demandeur, self.demande)
        )
```

---

9. Résumé des permissions par rôle

DEMANDEUR (Tout employé)

- ✅ Créer et modifier ses propres demandes (brouillon)
- ✅ Soumettre ses demandes
- ✅ Annuler ses demandes (si pas encore converties)
- ✅ Consulter le catalogue
- ✅ Voir ses demandes, BCs liés et réceptions
- ❌ Valider des demandes
- ❌ Créer des BCs
- ❌ Gérer les fournisseurs
- ❌ Créer des réceptions

VALIDATEUR_N1 (Manager)

- ✅ Tous les droits de DEMANDEUR
- ✅ Valider N1 les demandes de son équipe
- ✅ Refuser les demandes de son équipe
- ❌ Valider N2
- ❌ Convertir en BC

VALIDATEUR_N2 (Direction/Responsable achats)

- ✅ Tous les droits de VALIDATEUR_N1
- ✅ Valider N2 les demandes importantes
- ✅ Refuser les demandes au niveau N2
- ❌ Créer des BCs (sauf si aussi ACHETEUR)

ACHETEUR

- ✅ Tous les droits de VALIDATEUR_N2
- ✅ Créer, modifier, émettre, envoyer des BCs
- ✅ Convertir les demandes validées en BCs
- ✅ Gérer les fournisseurs
- ✅ Évaluer les fournisseurs
- ✅ Créer et valider des réceptions
- ✅ Voir toutes les demandes et tous les BCs
- ✅ Gérer le catalogue

RECEPTIONNAIRE

- ✅ Tous les droits de DEMANDEUR
- ✅ Créer et valider des réceptions
- ❌ Créer des BCs
- ❌ Gérer les fournisseurs

GESTIONNAIRE_BUDGET

- ✅ Créer et modifier ses budgets
- ✅ Voir la consommation budgétaire
- ✅ Configurer les seuils d'alerte
- ❌ Créer des demandes/BCs (sauf si aussi DEMANDEUR)

ADMIN_GAC (Administrateur)

- ✅ Tous les droits sur toutes les fonctionnalités
- ✅ Supprimer des demandes/BCs
- ✅ Modifier tous les budgets
- ✅ Accès complet au module

Fin des spécifications des permissions