PM

MODULE PROJECT MANAGEMENT

Spécification des Workflows

WORKFLOWS_SPEC.md

HR_ONIAN · Spécification Technique · Février 2026

Spécifications des Workflows - Module GAC

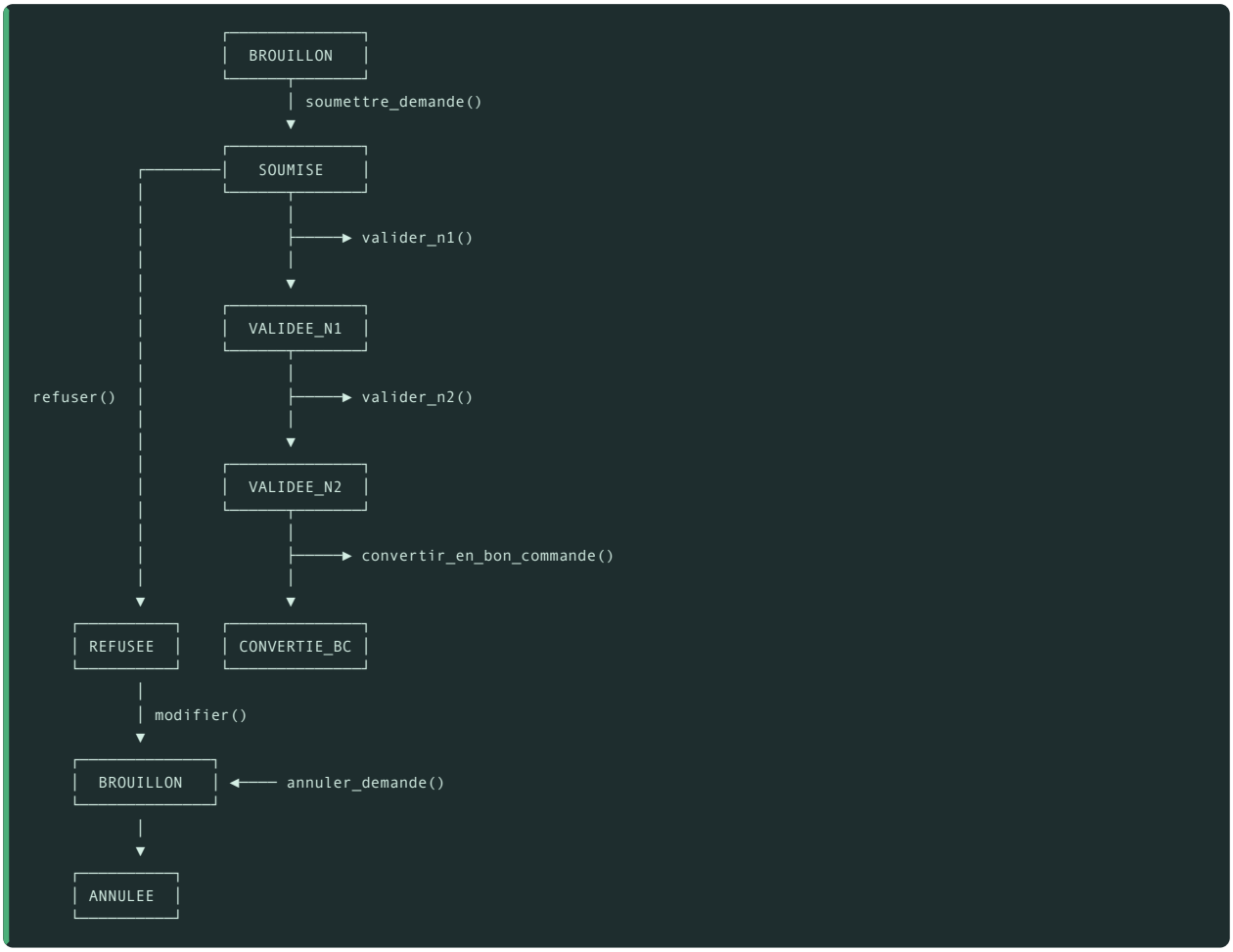
Version: 1.0
Date: 01/02/2026
Projet: HR_ONIAN
Module: Gestion des Achats & Commandes

Vue d'ensemble

Ce document décrit en détail tous les workflows du module GAC, incluant les états, les transitions, les conditions, les validations et les actions automatiques.

1. Workflow des Demandes d'Achat

1.1 Diagramme d'états



1.2 États

État	Code	Description	Actions possibles
Brouillon	`BROUILLON`	Demande en cours de rédaction	Modifier, Soumettre, Annuler
Soumise	`SOUMISE`	En attente validation N1	Valider N1, Refuser N1
Validée N1	`VALIDEE_N1`	Validée par le manager, en attente validation N2	Valider N2, Refuser N2
Validée N2	`VALIDEE_N2`	Validée par la direction/achats	Convertir en BC
Convertie BC	`CONVERTIE_BC`	Transformée en bon de commande	Consulter uniquement
Refusée	`REFUSEE`	Refusée par un validateur	Modifier (retour BROUILLON), Consulter
Annulée	`ANNULEE`	Annulée par le demandeur	Consulter uniquement

1.3 Transitions détaillées

1.3.1 BROUILLON → SOUMISE

Déclencheur : `DemandeService.soumettre_demande(demande, utilisateur)`

Conditions préalables :
- Demande au statut `BROUILLON`
- Au moins une ligne de demande
- Montant total > 0
- Demandeur a un manager (validateur N1)

Actions :
1. Passer le statut à `SOUMISE`
2. Enregistrer `date_soumission`
3. Déterminer et assigner `validateur_n1` (manager du demandeur)
4. Créer notification in-app pour validateur N1
5. Envoyer email au validateur N1
6. Créer entrée historique
7. Logger l'opération

Résultat :
- Statut = `SOUMISE`
- Demande non modifiable
- Validateur N1 notifié

Rollback en cas d'erreur :
- Transaction atomique → aucun changement si échec
- Statut reste `BROUILLON`

1.3.2 SOUMISE → VALIDEE_N1

Déclencheur : `DemandeService.valider_n1(demande, validateur, commentaire)`

Conditions préalables :
- Demande au statut `SOUMISE`
- Utilisateur = validateur_n1 de la demande
- Pas de blocage budgétaire

Actions :
1. Passer le statut à `VALIDEE_N1`
2. Enregistrer `date_validation_n1`
3. Enregistrer `commentaire_validation_n1` si fourni
4. **Déterminer si validation N2 nécessaire** :
- Si `montant_total_ttc` < `SEUIL_VALIDATION_N2` (défaut: 5000 €)
→ Passer directement à `VALIDEE_N2` (validation automatique)
- Sinon → Déterminer et assigner `validateur_n2`

5. Si validation N2 nécessaire :
- Créer notification pour validateur N2
- Envoyer email au validateur N2
6. Si validation N2 automatique :
- Passer directement au statut `VALIDEE_N2`
- Notifier le demandeur
7. Créer entrée historique
8. Logger l'opération

Résultat :
- Statut = `VALIDEE_N1` ou `VALIDEE_N2` (selon montant)
- Validateur N2 notifié (si nécessaire)
- Demande non modifiable

1.3.3 VALIDEE_N1 → VALIDEE_N2

Déclencheur : `DemandeService.valider_n2(demande, validateur, commentaire)`

Conditions préalables :
- Demande au statut `VALIDEE_N1`
- Utilisateur = validateur_n2 OU a le rôle `ACHETEUR` ou `VALIDATEUR_N2`
- Budget disponible si budget assigné

Actions :
1. Passer le statut à `VALIDEE_N2`
2. Enregistrer `date_validation_n2`
3. Enregistrer `commentaire_validation_n2` si fourni
4. **Si budget assigné** :
- Appeler `BudgetService.engager_montant()`
- Incrémenter `montant_engage` du budget
- Vérifier seuils d'alerte budgétaire
5. Créer notification pour le demandeur (demande validée)
6. Créer notification pour les acheteurs (prête pour BC)
7. Créer entrée historique
8. Logger l'opération

Résultat :
- Statut = `VALIDEE_N2`
- Budget engagé (si applicable)
- Demandeur et acheteurs notifiés
- Prête pour conversion en BC

1.3.4 VALIDEE_N2 → CONVERTIE_BC

Déclencheur : `DemandeService.convertir_en_bon_commande(demande, utilisateur, fournisseur, date_livraison)`

Conditions préalables :
- Demande au statut `VALIDEE_N2`
- Utilisateur a le rôle `ACHETEUR`
- Fournisseur sélectionné et actif

Actions :
1. Appeler `BonCommandeService.creer_bon_commande()`
- Générer numéro BC
- Créer BC en statut `BROUILLON`
- Copier toutes les lignes de la demande vers BC
- Calculer totaux BC
2. Passer le statut de la demande à `CONVERTIE_BC`
3. Assigner `bon_commande` (relation)
4. **Si budget assigné** :
- Transférer l'engagement vers le BC

- `montant_engage` du budget reste inchangé (sera transféré à `montant_commande` lors de l'émission du BC)
5. Créer notification pour le demandeur (BC créé)
6. Créer entrée historique sur demande
7. Créer entrée historique sur BC
8. Logger l'opération

Résultat :
- Statut = `CONVERTIE_BC`
- BC créé et lié
- Demande non modifiable, archivée
- Demandeur notifié

1.3.5 SOUMISE/VALIDEE_N1 → REFUSEE

Déclencheur : `DemandeService.refuser_demande(demande, validateur, motif_refus)`

Conditions préalables :
- Si statut = `SOUMISE` → utilisateur = validateur_n1
- Si statut = `VALIDEE_N1` → utilisateur = validateur_n2 OU rôle `ACHETEUR`
- Motif de refus fourni (obligatoire)

Actions :
1. **Si budget engagé** (état `VALIDEE_N1` ou `VALIDEE_N2`) :
- Appeler `BudgetService.liberer_montant()`
- Décrémenter `montant_engage` du budget
2. Passer le statut à `REFUSEE`
3. Enregistrer `motif_refus`
4. Enregistrer `date_refus`
5. Créer notification pour le demandeur (demande refusée + motif)
6. Envoyer email au demandeur
7. Créer entrée historique
8. Logger l'opération

Résultat :
- Statut = `REFUSEE`
- Budget libéré (si engagé)
- Demandeur notifié avec motif
- Demande modifiable (peut retourner à `BROUILLON`)

1.3.6 REFUSEE → BROUILLON

Déclencheur : Modification manuelle par le demandeur

Conditions préalables :
- Demande au statut `REFUSEE`
- Utilisateur = demandeur de la demande

Actions :
1. Passer le statut à `BROUILLON`
2. Réinitialiser `motif_refus`, `date_refus`
3. Réinitialiser validations (dates, commentaires)
4. Créer entrée historique
5. Logger l'opération

Résultat :
- Statut = `BROUILLON`
- Demande modifiable
- Peut être resoumise

1.3.7 BROUILLON/SOUMISE/VALIDEE_N1/VALIDEE_N2 → ANNULEE

Déclencheur : `DemandeService.annuler_demande(demande, utilisateur, motif_annulation)`

Conditions préalables :
- Demande pas encore convertie en BC
- Utilisateur = demandeur OU utilisateur a rôle `ADMIN_GAC`
- Motif d'annulation fourni

Actions :
1. **Si budget engagé** :
- Appeler `BudgetService.liberer_montant()`
- Décrémenter `montant_engage` du budget
2. Passer le statut à `ANNULEE`
3. Enregistrer `motif_annulation`
4. Enregistrer `date_annulation`
5. Créer notifications pour les validateurs concernés
6. Créer entrée historique
7. Logger l'opération

Résultat :
- Statut = `ANNULEE`
- Budget libéré (si engagé)
- Validateurs notifiés
- Demande archivée, non modifiable

1.4 Règles de gestion

Règle RG-DA-001 : Seuil de validation N2

- **Description** : Déterminer si une demande nécessite une validation N2
- **Paramètre** : `GAC_SEUIL_VALIDATION_N2` (défaut: 5000 €)
- **Logique** :
- Si `montant_total_ttc` < seuil → Validation N2 automatique
- Sinon → Validation N2 manuelle requise

Règle RG-DA-002 : Détermination du validateur N2

- **Description** : Choisir le validateur N2 selon le type de demande
- **Logique** :
  `python if demande.montant_total_ttc > 10000: validateur_n2 = get_directeur_general() elif demande.categorie_principale in ['INFORMATIQUE', 'MATERIEL_IT']: validateur_n2 = get_responsable_it() else: validateur_n2 = get_responsable_achats()`

Règle RG-DA-003 : Contrôle budgétaire

- **Description** : Vérifier la disponibilité budgétaire avant validation N2
- **Moment** : Lors de la validation N2
- **Action** : Si budget insuffisant → lever exception `BudgetInsuffisantError`

Règle RG-DA-004 : Délai de traitement

- **Description** : Suivre les délais de traitement des demandes
- **SLA** :
- Validation N1 : 2 jours ouvrés
- Validation N2 : 3 jours ouvrés
- Total : 5 jours ouvrés maximum
- **Action** : Créer alerte automatique si SLA dépassé

Règle RG-DA-005 : Modification après refus

- **Description** : Une demande refusée peut être modifiée et resoumise
- **Logique** :

- Le demandeur peut modifier une demande `REFUSEE`
- La modification repasse automatiquement à `BROUILLON`
- Les anciennes validations sont effacées
- L'historique conserve la trace du refus

---

2. Workflow des Bons de Commande

2.1 Diagramme d'états

2.2 États

État	Code	Description	Actions possibles
Brouillon	`BROUILLON`	BC en cours de rédaction	Modifier, Émettre, Supprimer
Émis	`EMIS`	BC finalisé, PDF généré	Envoyer, Annuler
Envoyé	`ENVOYE`	BC envoyé au fournisseur	Confirmer, Relancer, Annuler
Confirmé	`CONFIRME`	Commande confirmée par le fournisseur	Créer réception, Annuler
Reçu partiel	`RECU_PARTIEL`	Livraison partielle reçue	Créer nouvelle réception
Reçu complet	`RECU_COMPLET`	Livraison complète reçue	Consulter uniquement
Annulé	`ANNULE`	BC annulé	Consulter uniquement

2.3 Transitions détaillées

2.3.1 BROUILLON → EMIS

Déclencheur : `BonCommandeService.emettre_bon_commande(bc, utilisateur)`

Conditions préalables :
- BC au statut `BROUILLON`
- Au moins une ligne de commande
- Fournisseur renseigné
- Montant total > 0
- Utilisateur a le rôle `ACHETEUR`

Actions :
1. Générer le PDF du BC
- Utiliser template `templates/gestion_achats/pdf/bon_commande.html`
- Inclure : logo, infos entreprise, infos fournisseur, lignes, totaux, conditions
2. Sauvegarder le PDF dans `media/gestion_achats/bons_commande/`
3. Passer le statut à `EMIS`
4. Enregistrer `date_emission`
5. **Si lié à une demande avec budget** :
- Appeler `BudgetService.commander_montant()`
- Décrémenter `montant_engage` du budget
- Incrémenter `montant_commande` du budget
6. Créer entrée historique
7. Logger l'opération

Résultat :
- Statut = `EMIS`
- PDF généré et sauvegardé
- BC verrouillé (non modifiable)
- Budget transféré de "engagé" à "commandé"

2.3.2 EMIS → ENVOYE

Déclencheur : `BonCommandeService.envoyer_au_fournisseur(bc, utilisateur, email_destinataire)`

Conditions préalables :
- BC au statut `EMIS`
- PDF généré
- Email destinataire valide (fournisseur ou custom)
- Utilisateur a le rôle `ACHETEUR`

Actions :
1. Construire l'email
- Sujet : "Bon de commande {numero}"
- Corps : Message formaté avec infos BC
- Pièce jointe : PDF du BC
2. Envoyer l'email via `django.core.mail.EmailMessage`
3. Passer le statut à `ENVOYE`
4. Enregistrer `date_envoi`
5. Enregistrer `email_envoi`
6. Créer notification in-app pour traçabilité
7. Créer entrée historique
8. Logger l'opération

Résultat :
- Statut = `ENVOYE`
- Email envoyé au fournisseur
- Date et destinataire tracés

2.3.3 ENVOYE → CONFIRME

Déclencheur : `BonCommandeService.confirmer_commande(bc, utilisateur, numero_confirmation, date_livraison_confirmee)`

Conditions préalables :
- BC au statut `ENVOYE`
- Utilisateur a le rôle `ACHETEUR`

Actions :
1. Passer le statut à `CONFIRME`
2. Enregistrer `numero_confirmation_fournisseur` si fourni
3. Enregistrer `date_livraison_confirmee` si fournie
4. **Si date_livraison_confirmee > date_livraison_souhaitee** :
- Créer alerte automatique (retard prévu)
- Notifier l'acheteur et le demandeur
5. Créer notification pour le demandeur (commande confirmée)
6. Créer entrée historique
7. Logger l'opération

Résultat :
- Statut = `CONFIRME`
- Informations de confirmation enregistrées
- Alerte créée si retard prévu

2.3.4 CONFIRME → RECU_PARTIEL

Déclencheur : `ReceptionService.valider_reception(reception, utilisateur)` où réception partielle

Conditions préalables :
- BC au statut `CONFIRME` ou `RECU_PARTIEL`
- Réception validée avec quantités < quantités commandées

Actions :
1. Calculer les quantités totales reçues par ligne
- Sommer toutes les réceptions validées
2. Mettre à jour `quantite_recue` sur chaque ligne BC
3. Vérifier si réception complète :
- Si toutes les lignes sont complètement reçues → `RECU_COMPLET`
- Sinon → `RECU_PARTIEL`
4. Passer le statut à `RECU_PARTIEL`
5. **Si budget lié** :
- Appeler `BudgetService.consommer_montant()` pour le montant reçu
- Décrémenter `montant_commande`
- Incrémenter `montant_consomme`
6. Créer notification pour acheteur et demandeur
7. Créer entrée historique
8. Logger l'opération

Résultat :
- Statut = `RECU_PARTIEL`
- Quantités reçues mises à jour
- Budget partiellement consommé
- Possibilité de créer nouvelle réception pour le reste

2.3.5 RECU_PARTIEL → RECU_COMPLET

Déclencheur : `ReceptionService.valider_reception(reception, utilisateur)` où réception finale

Conditions préalables :
- BC au statut `RECU_PARTIEL`
- Dernière réception complète les quantités commandées

Actions :
1. Calculer les quantités totales reçues
2. Vérifier que toutes les lignes sont complètes
3. Passer le statut à `RECU_COMPLET`
4. Enregistrer `date_reception_complete`
5. **Si budget lié** :
- Consommer le montant restant
- `montant_commande` → 0
- `montant_consomme` augmenté
6. Créer notification pour acheteur et demandeur
7. **Évaluation du fournisseur** :
- Proposer à l'acheteur d'évaluer le fournisseur
- Calculer taux de respect des délais
8. Créer entrée historique
9. Logger l'opération

Résultat :
- Statut = `RECU_COMPLET`
- BC archivé, workflow terminé
- Budget totalement consommé
- Fournisseur peut être évalué

2.3.6 BROUILLON/EMIS/ENVOYE/CONFIRME → ANNULE

Déclencheur : `BonCommandeService.annuler_bon_commande(bc, utilisateur, motif_annulation)`

Conditions préalables :
- BC pas encore reçu (pas `RECU_PARTIEL` ou `RECU_COMPLET`)
- Utilisateur a le rôle `ACHETEUR` ou `ADMIN_GAC`
- Motif d'annulation fourni

Actions :
1. **Si budget lié** :
- Appeler `BudgetService.liberer_montant()`
- Décrémenter `montant_engage` ou `montant_commande` selon l'état
2. Passer le statut à `ANNULE`
3. Enregistrer `motif_annulation`
4. Enregistrer `date_annulation`
5. **Si BC déjà envoyé au fournisseur** :
- Envoyer email d'annulation au fournisseur
6. Notifier le demandeur (si lié à une demande)
7. Créer entrée historique
8. Logger l'opération

Résultat :
- Statut = `ANNULE`
- Budget libéré
- Fournisseur informé (si envoyé)
- BC archivé

2.4 Règles de gestion

Règle RG-BC-001 : Génération du PDF

- **Description** : Le PDF doit contenir toutes les informations légales
- **Contenu obligatoire** :
- Logo et coordonnées entreprise (SIRET, adresse)
- Numéro et date du BC
- Coordonnées fournisseur

- Liste des articles (référence, désignation, quantité, prix unitaire, montant)
- Totaux HT, TVA, TTC
- Conditions de paiement
- Date de livraison souhaitée
- Adresse de livraison
- Signature électronique (nom de l'acheteur)

Règle RG-BC-002 : Verrouillage après émission

- **Description** : Un BC émis ne peut plus être modifié
- **Logique** : Seule l'annulation est possible (avec motif)

Règle RG-BC-003 : Réceptions multiples

- **Description** : Un BC peut avoir plusieurs réceptions (livraisons partielles)
- **Logique** :
- Le statut passe à `RECU_PARTIEL` dès la première réception
- Le statut passe à `RECU_COMPLET` quand toutes les quantités sont reçues

Règle RG-BC-004 : Alerte retard de livraison

- **Description** : Alerter automatiquement si retard prévu
- **Déclencheurs** :
- Date confirmée > date souhaitée (lors de la confirmation)
- Date du jour > date confirmée + 7 jours (tâche planifiée quotidienne)
- **Action** : Créer notification pour acheteur et demandeur

---

3. Workflow des Réceptions

3.1 Diagramme d'états

3.2 États

État	Code	Description
Brouillon	`BROUILLON`	Réception en cours de saisie
Validée	`VALIDEE`	Réception validée et enregistrée
Annulée	`ANNULEE`	Réception annulée

3.3 Transitions détaillées

3.3.1 BROUILLON → VALIDEE

Déclencheur : `ReceptionService.valider_reception(reception, utilisateur)`

Conditions préalables :
- Réception au statut `BROUILLON`

- Toutes les lignes ont des quantités renseignées
- Utilisateur a le rôle `RECEPTIONNAIRE` ou `ACHETEUR`

Actions :
1. Vérifier que toutes les lignes sont renseignées
2. Calculer la conformité globale
- `conforme = True` si toutes les lignes sont conformes
3. Passer le statut à `VALIDEE`
4. Enregistrer `date_validation`
5. Pour chaque ligne de réception :
- Mettre à jour `quantite_recue` sur la ligne BC correspondante
6. Calculer le statut du BC :
- Si toutes les lignes BC sont complètement reçues → `RECU_COMPLET`
- Sinon → `RECU_PARTIEL`
7. Mettre à jour le BC
8. **Si budget lié** :
- Calculer le montant total reçu (quantité_acceptée × prix_unitaire)
- Appeler `BudgetService.consommer_montant()`
9. **Si non conforme** :
- Créer alerte automatique (litige fournisseur)
- Notifier l'acheteur et le responsable achats
10. Notifier le demandeur (marchandises reçues)
11. Créer entrée historique sur réception
12. Créer entrée historique sur BC
13. Logger l'opération

Résultat :
- Statut = `VALIDEE`
- BC mis à jour (statut + quantités reçues)
- Budget consommé
- Alertes créées si non-conformité
- Tous les acteurs notifiés

3.3.2 BROUILLON → ANNULEE

Déclencheur : Annulation manuelle

Actions :
1. Passer le statut à `ANNULEE`
2. Enregistrer le motif
3. Créer entrée historique
4. Logger l'opération

Résultat :
- Statut = `ANNULEE`
- Aucun impact sur le BC
- Réception archivée

3.4 Règles de gestion

Règle RG-REC-001 : Contrôle de conformité

- **Description** : Chaque ligne doit être contrôlée individuellement
- **Critères de conformité** :
- Quantité reçue = quantité commandée
- Qualité conforme aux spécifications
- Conditionnement intact
- Références correctes

Règle RG-REC-002 : Gestion des refus

- **Description** : Les quantités refusées doivent être tracées
- **Logique** :
- `quantite_refusee` > 0 → `conforme = False` sur la ligne
- Motif de refus obligatoire si quantité refusée
- Créer litige fournisseur automatiquement

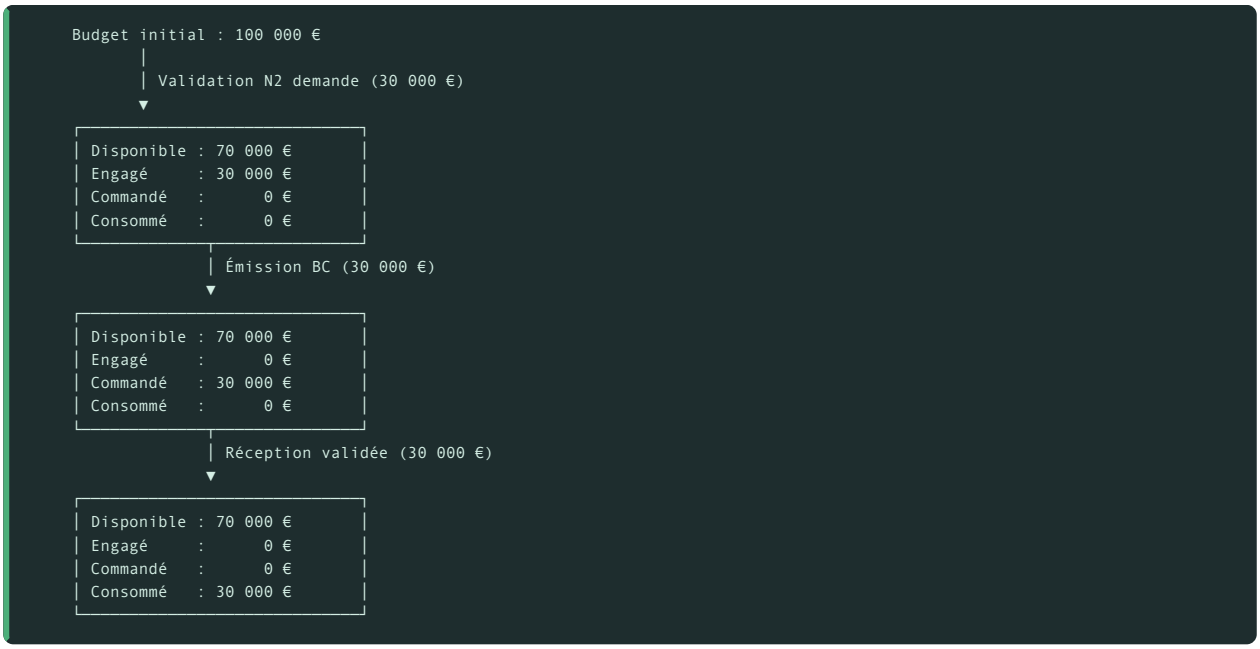
Règle RG-REC-003 : Impact budgétaire

- **Description** : Seules les quantités acceptées consomment le budget
- **Logique** :
- Montant consommé = Σ (quantite_acceptee × prix_unitaire)
- Les quantités refusées ne consomment pas le budget

---

4. Workflow Budgétaire

4.1 Diagramme du cycle budgétaire



4.2 États budgétaires

Un budget n'a pas d'états formels, mais plusieurs montants qui évoluent :

Montant	Description	Événement déclencheur
`montant_initial`	Budget alloué en début d'exercice	Création budget
`montant_engage`	Montant des demandes validées	Validation N2 demande
`montant_commande`	Montant des BCs émis	Émission BC
`montant_consomme`	Montant réellement dépensé	Validation réception
`montant_disponible`	= initial - (engagé + commandé + consommé)	Calculé

4.3 Transitions budgétaires

4.3.1 Engagement (Validation N2)

Déclencheur : `BudgetService.engager_montant(budget, montant, reference)`

Actions :
1. Vérifier disponibilité : `montant <= montant_disponible()`
2. Si insuffisant → lever `BudgetInsuffisantError`
3. Incrémenter `montant_engage`
4. Vérifier seuils d'alerte
5. Créer entrée historique

Formule :

```
montant_engage_nouveau = montant_engage_ancien + montant
montant_disponible_nouveau = montant_initial - (montant_engage_nouveau + montant_commande + montant_consomme)
```

4.3.2 Commande (Émission BC)

Déclencheur : `BudgetService.commander_montant(budget, montant, reference)`

Actions :
1. Décrémenter `montant_engage`
2. Incrémenter `montant_commande`
3. Créer entrée historique

Formule :

```
montant_engage_nouveau = montant_engage_ancien - montant
montant_commande_nouveau = montant_commande_ancien + montant
```

Note : Le `montant_disponible` ne change pas (le montant était déjà compté dans "engagé")

4.3.3 Consommation (Validation réception)

Déclencheur : `BudgetService.consommer_montant(budget, montant, reference)`

Actions :
1. Décrémenter `montant_commande`
2. Incrémenter `montant_consomme`
3. Vérifier seuils d'alerte
4. Créer entrée historique

Formule :

```
montant_commande_nouveau = montant_commande_ancien - montant
montant_consomme_nouveau = montant_consomme_ancien + montant
```

4.3.4 Libération (Annulation/Refus)

Déclencheur : `BudgetService.liberer_montant(budget, montant, reference)`

Actions :
1. Si `montant_engage` > 0 → décrémenter `montant_engage`
2. Sinon → décrémenter `montant_commande`
3. Créer entrée historique

Formule :

```
Si montant_engage > 0:
    montant_engage_nouveau = montant_engage_ancien - montant
Sinon:
    montant_commande_nouveau = montant_commande_ancien - montant

montant_disponible augmente
```

4.4 Alertes budgétaires

Règle RG-BUD-001 : Seuil d'alerte 1 (80%)

Condition : `taux_consommation() >= seuil_alerte_1` (défaut: 80%)

Actions :
1. Créer notification `AVERTISSEMENT` pour le gestionnaire budget
2. Envoyer email au gestionnaire
3. Marquer `alerte_1_envoyee = True`

Message : "Le budget {code} a atteint {taux}% de consommation"

Règle RG-BUD-002 : Seuil d'alerte 2 (95%)

Condition : `taux_consommation() >= seuil_alerte_2` (défaut: 95%)

Actions :
1. Créer notification `CRITIQUE` pour le gestionnaire budget
2. Envoyer email au gestionnaire ET à la direction
3. Marquer `alerte_2_envoyee = True`

Message : "⚠️ ALERTE CRITIQUE : Le budget {code} a atteint {taux}% de consommation"

Règle RG-BUD-003 : Dépassement budgétaire

Condition : `montant_disponible() < 0`

Actions :
1. Bloquer toute nouvelle validation de demande
2. Créer alerte `BLOQUANT` pour le gestionnaire et la direction
3. Envoyer email urgent
4. Logger en ERROR

Message : "🚨 BUDGET DÉPASSÉ : Le budget {code} est en dépassement de {montant_dépassement} €"

---

5. Tâches planifiées automatiques

5.1 Vérification des délais de validation

Fréquence : Quotidienne (6h00)

Commande Django : `python manage.py verifier_delais_validation`

Actions :
1. Récupérer toutes les demandes au statut `SOUMISE` ou `VALIDEE_N1`
2. Pour chaque demande :
- Calculer le délai écoulé depuis la soumission
- Si délai > SLA (5 jours) → créer alerte retard
- Notifier le validateur en attente
- Notifier le demandeur (information)

5.2 Vérification des délais de livraison

Fréquence : Quotidienne (7h00)

Commande Django : `python manage.py verifier_delais_livraison`

Actions :
1. Récupérer tous les BCs au statut `CONFIRME`
2. Pour chaque BC :
- Si `date_livraison_confirmee` < `date_du_jour` → créer alerte retard

- Notifier l'acheteur
- Notifier le demandeur

5.3 Rappel de réception en attente

Fréquence : Hebdomadaire (lundi 8h00)

Commande Django : `python manage.py rappel_receptions_en_attente`

Actions :
1. Récupérer tous les BCs au statut `CONFIRME` depuis > 30 jours
2. Pour chaque BC :
- Créer notification pour l'acheteur
- Suggérer de créer une réception ou de relancer le fournisseur

---

5.4 Vérification budgétaire

Fréquence : Quotidienne (23h00)

Commande Django : `python manage.py verifier_budgets`

Actions :
1. Récupérer tous les budgets actifs
2. Pour chaque budget :
- Recalculer les montants (cohérence)
- Vérifier les seuils d'alerte
- Générer les alertes si nécessaire

---

6. Notifications automatiques - Résumé

6.1 Demandes d'achat

Événement	Destinataires	Type
Soumission demande	Validateur N1	Email + In-app
Validation N1	Validateur N2 (si nécessaire)	Email + In-app
Validation N1	Demandeur (si validation auto N2)	In-app
Validation N2	Demandeur, Acheteurs	Email + In-app
Refus demande	Demandeur	Email + In-app
Annulation demande	Validateurs concernés	In-app
Conversion en BC	Demandeur	In-app

6.2 Bons de commande

Événement	Destinataires	Type
Émission BC	-	-
Envoi BC	Fournisseur	Email
Confirmation BC	Demandeur	In-app
Retard prévu	Acheteur, Demandeur	Email + In-app
Annulation BC	Fournisseur (si envoyé), Demandeur	Email + In-app

6.3 Réceptions

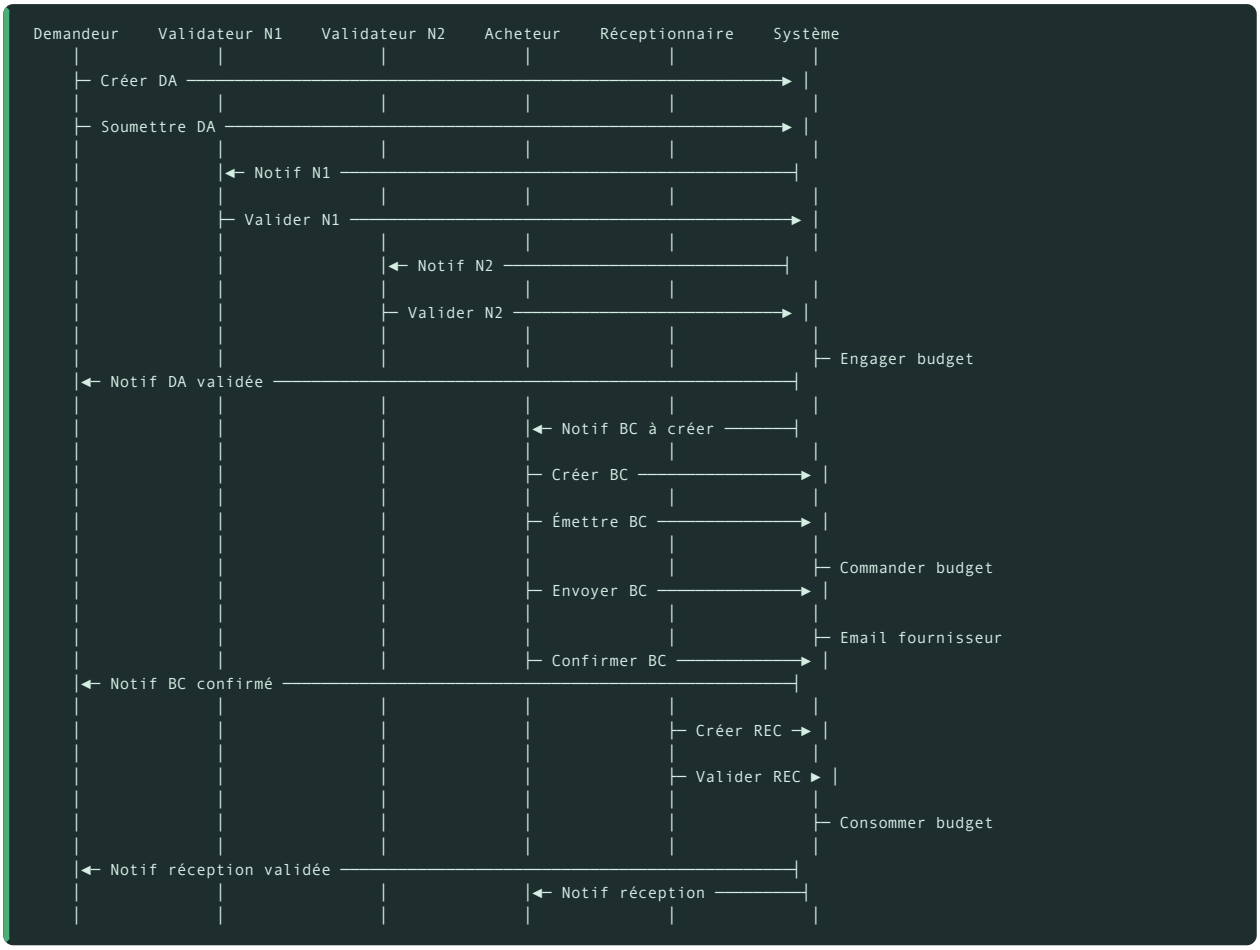
Événement	Destinataires	Type
Validation réception	Acheteur, Demandeur	Email + In-app
Non-conformité	Acheteur, Responsable achats	Email + In-app
Réception complète	Demandeur	Email + In-app

6.4 Budget

Événement	Destinataires	Type
Seuil 1 atteint (80%)	Gestionnaire budget	Email + In-app
Seuil 2 atteint (95%)	Gestionnaire budget, Direction	Email + In-app
Dépassement budgétaire	Gestionnaire budget, Direction, DG	Email + In-app

7. Diagrammes de séquence

7.1 Séquence complète : De la demande à la réception



Fin des spécifications des workflows