Guide d'Intégration

Planification Améliorée par Rôles

Système de Gestion des Commandes Pokémon

29 septembre 2025

Table des matières

1 Vue d'ensemble

Le nouveau système de planification améliore l'algorithme existant en apportant les fonctionnalités suivantes :

Améliorations Principales

- **Suppression du filtre de date initiale** : Traite TOUTES les commandes en attente
- **Assignment par rôle** : ROLE_NOTEUR pour A_NOTER, ROLE_CERTIFICATEUR pour A_CERTIFIER
- **Optimisation de la charge** : Trouve l'employé le moins chargé dans chaque rôle
- **Respect des heures de travail** : Planifie dans les heures ouvrables (9h-18h)

1.1 Architecture du Système

Le nouveau système introduit le service `ImprovedPlanningService` qui travaille en conjonction avec le contrôleur existant pour fournir une planification basée sur les rôles des employés.

2 Étape 1 : Ajouter le Nouveau Service

2.1 Création du Fichier

Emplacement : `src/main/java/com/pcagrade/order/service/ImprovedPlanningService.java`

2.2 Structure du Service

Le service `ImprovedPlanningService` contient les méthodes principales suivantes :

Listing 1 – Méthodes principales du service

```
// M thode principale d'ex cution
public Map<String, Object> executeRoleBasedPlanning(
    LocalDate planningDate,
    boolean cleanFirst
)

// R cup re les commandes par status (sans filtre de date)
private List<Map<String, Object>> getOrdersByStatus(int status)

// R cup re les employ s ayant un r le sp cifique
private List<Map<String, Object>> getEmployeesByRole(String roleName)

// Assigne les commandes en trouvant l'employ  le moins charg
private int assignOrdersToEmployees(...)
```

2.3 Constantes du Service

```
private static final int TIME_PER_CARD_MINUTES = 3;
private static final LocalTime WORK_START_TIME = LocalTime.of(9, 0);
private static final LocalTime WORK_END_TIME = LocalTime.of(18, 0);

// Order Status Constants
private static final int STATUS_A_NOTER = 2;      // To be evaluated
private static final int STATUS_A_CERTIFIER = 3;  // To be encapsulated
```

3 Étape 2 : Modifier le Contrôleur

3.1 Injection du Service

Dans `PlanningController.java`, ajoutez l'injection du nouveau service :

```
@Autowired
private ImprovedPlanningService improvedPlanningService;
```

3.2 Ajout de l'Endpoint

Ajoutez la méthode suivante à `PlanningController.java` :

Listing 2 – Endpoint de planification basée sur les rôles

```
@PostMapping("/generate-role-based")
@Transactional
public ResponseEntity<Map<String, Object>> generateRoleBasedPlanning(
        @RequestBody Map<String, Object> request) {

    Map<String, Object> result = new HashMap<>();

    try {
        String planningDateStr = (String) request.getOrDefault(
            "planningDate",
            LocalDate.now().toString()
        );
        boolean cleanFirst = (Boolean) request.getOrDefault(
            "cleanFirst",
            true
        );

        LocalDate planningDate = LocalDate.parse(planningDateStr);

        log.info("Generating role-based planning for date: {}",
                planningDate);

        result = improvedPlanningService.executeRoleBasedPlanning(
            planningDate,
            cleanFirst
        );

        if ((Boolean) result.get("success")) {
            return ResponseEntity.ok(result);
        } else {
            return ResponseEntity.status(500).body(result);
        }

    } catch (Exception e) {
        log.error("Error in role-based planning generation", e);
        result.put("success", false);
        result.put("message", "Planning failed: " + e.getMessage());
        return ResponseEntity.internalServerError().body(result);
    }
}
```

Endpoint créé :

`POST /api/planning/generate-role-based`

4 Étape 3 : Test du Système

4.1 Test avec cURL

```
curl -X POST http://localhost:8080/api/planning/generate-role-based \
  -H "Content-Type: application/json" \
  -d '{
    "planningDate": "2025-06-01",
    "cleanFirst": true
  }'
```

4.2 Réponse Attendue

Listing 3 – Exemple de réponse JSON


```
{
  "success": true,
  "message": "Planning completed: 25 orders assigned",
  "planningDate": "2025-06-01",
  "totalOrdersAssigned": 25,
  "ordersToNote": 15,
  "ordersToCertify": 10,
  "noteursUsed": 3,
  "certificateursUsed": 2,
  "totalCards": 450,
  "totalMinutes": 1350,
  "totalHours": "22.5",
  "algorithm": "ROLE_BASED_GREEDY",
  "noteurWorkloads": [
    {
      "employeeId": "abc123...",
      "employeeName": "John Doe",
      "totalMinutes": 480,
      "totalHours": "8.0",
      "workloadPercentage": 100,
      "status": "busy"
    }
  ],
  "certificateurWorkloads": [...]
}
```


5 Comparaison Ancien vs Nouveau

Ancien Système (Round-Robin)	**Nouveau Système (Role-Based)**
× Filtre de date requis (startDate)	Traite toutes les commandes en attente
× Pas de distinction de rôles	Assigne selon le rôle requis
× Distribution simple en alternance	Trouve l'employé le moins chargé
× Pas d'optimisation de charge	Optimise la charge de travail
Résultat	
Employés mal utilisés, certains surchargés	Meilleure distribution, respect des compétences

Table 1 – Comparaison des deux systèmes

6 Fonctionnement Détaillé

6.1 Récupération des Commandes par Status

Le système récupère les commandes en fonction de leur status, en les triant par priorité :

Listing 4 – Requête SQL de récupération des commandes

```
SELECT
    HEX(o.id) as id,
    o.num_commande as orderNumber,
    o.delai as priority,
    o.status,
    COALESCE(
        (SELECT COUNT(*) FROM card_certification_order cco
         WHERE cco.order_id = o.id),
        10
    ) as cardCount
FROM `order` o
WHERE o.status = ? -- 2 pour A_NOTER, 3 pour A_CERTIFIER
  AND o.annulee = 0
  AND o.paused = 0
ORDER BY
    CASE o.delai
        WHEN 'X' THEN 1     -- Excelsior en premier
        WHEN 'F+' THEN 2    -- Fast+ ensuite
        WHEN 'F' THEN 3     -- Fast
        WHEN 'C' THEN 4     -- Classic
        WHEN 'E' THEN 5     -- Economy
        ELSE 6
    END ASC,
    o.date ASC
```

6.2 Récupération des Employés par Rôle

Les employés sont récupérés en fonction de leur appartenance aux groupes de rôles :

Listing 5 – Requête SQL de récupération des employés

```
SELECT DISTINCT
    HEX(e.id) as id,
    e.first_name as firstName,
    e.last_name as lastName,
    e.email,
    e.work_hours_per_day as workHoursPerDay,
    e.active
FROM j_employee e
INNER JOIN j_employee_group eg ON e.id = eg.employee_id
INNER JOIN j_group g ON eg.group_id = g.id
WHERE g.name = ? -- 'ROLE_NOTEUR' ou 'ROLE_CERTIFICATEUR'
  AND e.active = 1
  AND g.active = 1
ORDER BY e.last_name, e.first_name
```

6.3 Algorithme d'Assignment

Algorithme d'Assignment Intelligent

Pour chaque commande :

1. **Trouve l'employé le moins chargé** dans le rôle approprié
2. **Calcule la durée** : $cardCount \times 3minutes$
3. **Détermine l'heure de début** :
   - Si l'employé n'a pas de tâches : 9h00
   - Sinon : Après la dernière tâche + 15min de pause
4. **Sauvegarde en base** dans `j_planning`
5. **Met à jour la charge** de l'employé

7 Configuration des Rôles

7.1 Vérification des Rôles Existants

```
SELECT name, description, permission_level
FROM j_group
WHERE name IN ('ROLE_NOTEUR', 'ROLE_CERTIFICATEUR');
```

7.2 Assigner un Rôle à un Employé

Étape 1 : Vérifier l'ID du groupe

```
SELECT HEX(id) as group_id, name
FROM j_group
WHERE name = 'ROLE_NOTEUR';
```

Étape 2 : Assigner l'employé au groupe

```
INSERT INTO j_employee_group (employee_id, group_id)
VALUES (
  UNHEX('employee_id_here'),
  UNHEX('group_id_here')
);
```

7.3 Via l'Interface Web

1. Allez dans **Groups**
2. Sélectionnez `ROLE_NOTEUR` ou `ROLE_CERTIFICATEUR`
3. Cliquez sur **Assign Employees**
4. Ajoutez les employés souhaités

8 Métriques et Monitoring

Le système retourne des statistiques détaillées dans la réponse JSON :

8.1 Métriques par Type de Commande

- `ordersToNote` : Nombre de commandes A_NOTER
- `ordersToCertify` : Nombre de commandes A_CERTIFIER

8.2 Métriques par Rôle

— `noteursUsed` : Nombre de noteurs utilisés

— `certificateursUsed` : Nombre de certificateurs utilisés

8.3 Métriques par Employé

Dans les objets `noteurWorkloads` et `certificateurWorkloads` :

— `totalMinutes` : Temps total assigné

— `workloadPercentage` : Pourcentage de la capacité (0-100%+)

— `status` :

  — `"available"` : $< 80\%$

  — `"busy"` : 80-99%

  — `"overloaded"` : $\geq 100\%$

9 Utilisation depuis le Frontend

9.1 Appel depuis Vue.js/TypeScript

Listing 6 – Exemple d'appel depuis Vue.js

```
const generateRoleBasedPlanning = async () => {
  try {
    const response = await fetch(
      `${API_BASE_URL}/api/planning/generate-role-based`,
      {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          planningDate: '2025-06-01',
          cleanFirst: true
        })
      }
    );

    const result = await response.json();

    if (result.success) {
      console.log('Planning generated:', result.message);
      console.log('Orders assigned:', result.totalOrdersAssigned);
      console.log('Noteurs:', result.noteurWorkloads);
      console.log('Certificateurs:', result.certificateurWorkloads);
    } else {
      console.error('Planning failed:', result.message);
    }
  } catch (error) {
    console.error('Error:', error);
  }
}
```

10 Points Importants

10.1 Validation des Rôles

Attention

Le système échoue si :

- Il y a des commandes A_NOTER mais aucun ROLE_NOTEUR
- Il y a des commandes A_CERTIFIER mais aucun ROLE_CERTIFICATEUR

Solution : Assignez au moins un employé à chaque rôle nécessaire.

10.2 Surcharge des Employés

Si un employé dépasse 100% de sa capacité :

- Le système continue à lui assigner des tâches
- Son status devient `"overloaded"`
- Visible dans les workload summaries

Solution : Ajoutez plus d'employés ou réduisez le `workHoursPerDay`.

10.3 Nettoyage des Plannings

Attention

`cleanFirst: true` supprime **tous** les plannings de la date spécifiée avant d'en créer de nouveaux.
Attention : Les plannings déjà commencés seront également supprimés.

11 Migration depuis l'Ancien Système

11.1 Option 1 : Remplacement Complet

Dans votre frontend, remplacez l'appel à :

```
// ANCIEN
POST /api/planning/generate-unified

// PAR
POST /api/planning/generate-role-based
```

11.2 Option 2 : Coexistence

Gardez les deux endpoints et laissez l'utilisateur choisir :

```
if (useRoleBasedPlanning) {
  await generateRoleBasedPlanning();
} else {
  await generateUnifiedPlanning();
}
```

12 Améliorations Futures

1. **Support de plus de status** : A_SCANNER, A_PREPARER, etc.
2. **Compétences avancées** : Efficiency rating par employé
3. **Contraintes horaires** : Pauses obligatoires, jours fériés
4. **Optimisation multi-critères** : Priorité + charge + compétences
5. **Prévisualisation** : Voir le planning avant de le sauvegarder

13 FAQ

13.1 Que se passe-t-il si un employé a plusieurs rôles ?

Il sera utilisé pour tous les types de commandes correspondant à ses rôles.

13.2 Comment gérer les commandes urgentes (X, F+) ?

Le système les traite déjà en priorité grâce au `ORDER BY` dans la requête SQL.

13.3 Peut-on limiter le nombre de commandes par employé ?

Actuellement non, mais c'est une amélioration future possible.

13.4 Comment voir la répartition avant de valider ?

Ajoutez un paramètre `preview: true` qui retourne le résultat sans sauvegarder (à implémenter).

14 Support et Dépannage

Pour toute question ou problème :

1. **Vérifiez les logs :** `tail -f logs/application.log`
2. **Vérifiez les rôles :** `SELECT * FROM j_group`
3. **Vérifiez les assignations :** `SELECT * FROM j_employee_group`

Bonne planification !