**Laboratoire 1**

**Développement Avancé des Fonctionnalités CRUD pour une Application de Gestion de Cartes**

Date limite de remise : 2 février 2024

**1 . Contexte :**

Vous allez améliorer une application de gestion de cartes de crédit en utilisant Spring Boot. Votre objectif est d'implémenter des méthodes CRUD personnalisées dans le service associé à l'entité Cards.

**2. Modèle de Carte de Crédit**

Le modèle Cards est déjà défini comme suit :

- cardId : ID unique de la carte (auto-généré).

- customerId : ID du client possédant la carte.

- cardNumber : Numéro de la carte de crédit.

- cardType : Type de la carte (par exemple, Visa, MasterCard).

- totalLimit, amountUsed, availableAmount : Détails financiers de la carte.

- createDt : Date de création de la carte.

**3. Tâches à Réaliser par Fonctionnalité**

**1. Suppression de Carte par Numéro**

- Endpoint: `DELETE /deleteCardByNumber/{cardNumber}`

- Documentation Swagger:

- Description: Supprime une carte spécifique à l'aide de son numéro.

- Paramètres: `cardNumber` (dans le chemin URL).

- Réponses: Succès (200 OK), Carte non trouvée (404), Erreur interne (500).

**2. Suppression des Cartes d'un Client**

- Endpoint: `DELETE /deleteCardsByCustomerId/{customerId}`

- Documentation Swagger:

- Description: Supprime toutes les cartes associées à un client donné.

- Paramètres: `customerId` (dans le chemin URL).

- Réponses: Succès (200 OK), Aucune carte trouvée (404), Erreur interne (500).

**3. Suppression en Lots**

- Endpoint: `DELETE /deleteMultipleCards`

- Documentation Swagger:

- Description: Supprime un ensemble de cartes spécifiées par leurs IDs.

- Corps de la Requête: Liste d'`cardId` (format JSON).

- Réponses: Succès (200 OK), Échec de suppression (400/500).

**4. Mise à Jour de la Limite de Crédit**

- Endpoint: `PUT /updateCardLimit/{cardId}`

- Documentation Swagger:

- Description: Met à jour la limite de crédit d'une carte spécifique.

- Paramètres: `cardId` (dans le chemin URL), nouvelles données de limite (dans le corps de la requête).

- Réponses: Mise à jour réussie (200 OK), Carte non trouvée (404), Données invalides (400).

**5. Recherche de Cartes par Type**

- Endpoint: `GET /cardsByType/{cardType}`

- Documentation Swagger:

- Description: Récupère toutes les cartes d'un type spécifié.

- Paramètres: `cardType` (dans le chemin URL).

- Réponses: Liste de cartes (200 OK), Aucune carte trouvée (404).

**Questions Clés pour la Documentation Swagger**

- Description détaillée de chaque fonctionnalité : Quel est le rôle spécifique de l'endpoint ?

- Paramètres requis : Quelles informations doivent être fournies pour utiliser l'endpoint ?

- Formats de requête et de réponse : Comment les données doivent-elles être structurées ?

- Codes de réponse HTTP : Quels sont les différents résultats possibles et leurs significations ?

- Cas d'erreur : Quelles erreurs peuvent survenir et comment les gérer ?

**4. Critères d'Évaluation de la Documentation**

- Fonctionnalité : Chaque méthode doit être pleinement fonctionnelle et traiter correctement tous les cas d'usage.

- Qualité du Code : Le code doit être clair, bien structuré et suivre les principes de développement de Spring Boot.

- Gestion des Erreurs : Les méthodes doivent gérer correctement les exceptions et les cas d'erreur.

- Exhaustivité : Tous les aspects de chaque fonctionnalité doivent être couverts.

- Clarté et Précision : Les informations doivent être faciles à comprendre et précises.

- Cohérence : La documentation doit avoir un format et un style uniformes.

**5. Conclusion**

Votre documentation Swagger doit servir de guide complet pour comprendre et interagir avec les fonctionnalités avancées de votre application. Elle doit être suffisamment détaillée pour permettre aux développeurs de l'utiliser efficacement sans ambiguïté.

Important : Testez chaque endpoint avec Swagger UI pour vous assurer que la documentation est correcte et que les endpoints fonctionnent comme prévu.