**Exercice : Modélisation et Génération des Données dans Laravel 12 – Gestion de Stock**

**Objectif :**

Dans cet exercice, vous allez implémenter un ensemble de modèles avec leurs relations, migrations, factories et seeders en utilisant Laravel 12. L’objectif est de comprendre comment structurer et gérer une base de données relationnelle pour un système de gestion de stock.

**Important** : Vous devez réaliser cet exercice sans utiliser d’intelligence artificielle. L’objectif est de vous habituer à lire et comprendre la documentation officielle, qui est votre meilleure alliée en tant que développeur. Pour vous aider, vous pouvez consulter la page suivante :  
<https://laravel.com/docs/12.x/eloquent-relationships>  
ou utiliser la solution de l’exercice precedent.

**Contexte** :

**Vous devez concevoir un système de gestion de stock où :**

1. Les fournisseurs (suppliers) fournissent des produits.
2. Les produits (products) ont un seul fournisseur et appartiennent à une catégorie (categories) (One-to-Many).
3. Chaque produit a un stock (stocks) qui enregistre la quantité actuelle et l’entrepôt où il est stocké (One-to-One) et un stock est liée a un et un seul produit, chaque produit a une et une seule image.
4. Les clients (customers) peuvent passer plusieurs commandes (orders) et une commande peut contenir plusieurs produits (Many-to-Many avec une table pivot order\_product contenant la quantité commandée et le prix de vente).
5. Les transactions (transactions) enregistrent les achats et ventes des produits. Une transaction peut être soit un achat (purchase) lié à un fournisseur, soit une vente (sale) liée à un client (Polymorphisme).
6. Un magasin (store) a plusieurs produits via la relation hasManyThrough, en passant par le stock.

**Important : toutes les tables doivent contenir les timeStamps.**

**Travail demandé :**

1. **Créer les modèles et définir les relations entre eux :**
   * Un produit appartient à une seule catégorie.
   * Un produit appartient à un seul fournisseur.
   * Un produit possède un stock unique.
   * Un client peut passer plusieurs commandes et une commande peut contenir plusieurs produits (Many-to-Many).
   * Une transaction peut être une vente ou un achat (Polymorphisme).
   * Un magasin possède plusieurs produits via une relation hasManyThrough.
   * Une commande peut être récupérée directement via un produit (hasOneThrough).
2. **Créer les migrations pour chaque table en respectant les contraintes suivantes :**
   * Clés primaires et étrangères bien définies.
   * Les colonnes nécessaires pour chaque entité avec les bons types de données.
   * Ajout des timestamps (created\_at et updated\_at).
3. **Créer les factories pour générer des données réalistes pour les entités suivantes :**
   * Suppliers : générer des noms et contacts aléatoires.
   * Categories : générer des catégories de produits.
   * Products : assignés à des catégories et des fournisseurs existants. Avec image
   * Stocks : générer des stocks avec des quantités aléatoires.
   * Customers : générer des clients avec des informations personnelles.
   * Orders : attribuées à des clients avec des produits aléatoires.
   * Transactions : générer des achats et ventes associées aux fournisseurs et clients.
4. **Créer les seeders pour injecter des données de test en utilisant les factories.**
5. **Tester le bon fonctionnement des relations en exécutant les migrations et les seeders.**

**Livrables attendus :**

* Les fichiers de **modèles** avec les relations bien définies.
* Les **migrations** de chaque table respectant les contraintes.
* Les **factories** pour générer des données réalistes.
* Les **seeders** permettant d’initialiser la base avec des données de test.