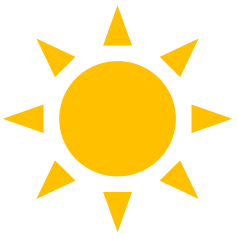

REGLES DE GESTION DE L'APPLICATION HELIGXIAM

Document officiel de modelisation

Projet : Marketplace HELIGXIAM

Annee : 2025

Table des matières

1. Utilisateurs

- 1.1 Un utilisateur peut créer un compte sur HELIGXIAM via POST /auth/register (username unique 3-30 chars, password min 8 chars, email valide unique, rôle default "client").
- 1.2 Un utilisateur peut acheter des produits uniquement après authentification (JWT via POST /auth/login).
- 1.3 Un utilisateur peut consulter ses commandes, informations et statistiques (ex. : ordersCount) via GET /auth/me ou GET /users/:userId.
- 1.4 Un utilisateur doit avoir un email unique et validé (vérification email recommandée pour éviter spam).
- 1.5 Un utilisateur doit être authentifié pour passer une commande (token requis pour POST /orders).
- 1.6 Un utilisateur peut avoir plusieurs adresses (billing/shipping, voir section 12).
- 1.7 Amélioration : Les utilisateurs peuvent changer de rôle via PUT /users/:userId/role (admin seulement), avec rôles "client", "vendeur", "admin" (SQL pour ACID sur users-service:3001).
- 1.8 Amélioration : Soft delete pour désactivation (DELETE /users/:userId marque comme inactif, conserve historique pour audits RGPD).
- 1.9 Exemple d'erreur : 409 si username/email existe déjà.

2. Vendeurs / Boutique

- 2.1 Un vendeur doit créer un compte utilisateur pour ouvrir une boutique (rôle "vendeur" requis).
- 2.2 Un vendeur possède une seule boutique (relation 1:1 avec user, nom_boutique unique).
- 2.3 Le nom de la boutique doit être unique (vérifié lors de création).
- 2.4 Une boutique doit être active pour publier des produits (statut "active" par défaut).
- 2.5 Un vendeur ne peut gérer que ses propres produits (sellerId = userId, via token).
- 2.6 Une boutique suspendue rend ses produits invisibles (filtre available=true dans GET /products).
- 2.7 Un vendeur reçoit des commandes uniquement pour ses produits (notification automatisée recommandée).
- 2.8 Un vendeur peut consulter les statistiques de sa boutique (ex. : productsCount, ordersCount via GET /users/:userId).
- 2.9 Une boutique ne peut être supprimée que si elle n'a plus de produits actifs (soft delete recommandé).
- 2.10 Amélioration : Ajout de description et images pour boutique (extension API possible : PUT /boutiques/:id).
- 2.11 Amélioration : Vendeur peut ajouter des attributs personnalisés (NoSQL flexible pour scalabilité).

3. Produits

- 3.1 Un produit doit appartenir à une catégorie (categoryId requis, relation n:1).
- 3.2 Un produit doit avoir un nom (1-200 chars), un prix (>0), et une description (max 2000 chars).
- 3.3 Le prix doit être strictement positif (validation 400 si <=0).
- 3.4 Un produit peut avoir plusieurs images (array URLs via POST /products/:productId/images).
- 3.5 Un produit ne peut pas être supprimé s'il est présent dans une commande (soft delete via DELETE /products/:productId).
- 3.6 La note moyenne est calculée automatiquement (avgRating basé sur avis, voir section 13).
- 3.7 Amélioration : Produits stockés en NoSQL (MongoDB, catalog:3002) pour flexibilité (attributes object, images array, recherche full-text).

- 3.8 Amélioration : Ajout de currency (default "EUR"), stock (>=0), sellerId auto (du token).
- 3.9 Amélioration : Recherche paginée (GET /products avec q, minPrice, sort=price:desc).
- 3.10 Exemple : Erreur 404 si categoryId invalide lors de POST /products.

4. Commandes

- 4.1 Une commande appartient à un seul utilisateur (userId du token).
- 4.2 Une commande doit contenir au moins un produit (items array non vide).
- 4.3 Le montant total est calculé automatiquement (somme quantity * price).
- 4.4 Une commande a un statut obligatoire (pending, confirmed, shipped, delivered, canceled).
- 4.5 Une commande ne peut être annulée que si elle n'est pas expédiée (PUT /orders/:orderId/cancel avant "shipped").
- 4.6 Une commande peut contenir plusieurs articles (order_items avec quantity >0).
- 4.7 La quantité doit être strictement supérieure à 0 (check en SQL).
- 4.8 Si une commande est supprimée, ses articles le sont aussi (cascade delete en SQL).
- 4.9 Un produit ne doit apparaître qu'une seule fois par commande (unique productId par order).
- 4.10 Amélioration : Stockés en SQL (PostgreSQL, orders:3003) pour transactions ACID (réservation stock atomique).
- 4.11 Amélioration : Requiert addresses (shipping/billing) et paymentMethod lors de POST /orders.
- 4.12 Amélioration : Transition de statuts validée (ex. : pending → confirmed après paiement).

5. Livraison

- 5.1 Une livraison est associée à une commande (1:1, via shippingAddressId).
- 5.2 Une commande ne peut être expédiée que si elle est payée (statut "confirmed").
- 5.3 Chaque commande expédiée doit avoir un numéro de suivi (trackingNumber ajouté via PUT /orders/:orderId/status).
- 5.4 Le statut de livraison doit être mis à jour automatiquement (ex. : shipped → delivered via API externe).
- 5.5 Amélioration : Intégration avec providers (ex. : Colissimo, UPS) pour tracking réel.
- 5.6 Amélioration : Notification utilisateur/vendeur lors de changements (webhook recommandé).
- 5.7 Amélioration : Adresse de livraison validée (postalCode, country ISO).

6. Panier

- 6.1 Un utilisateur ne peut avoir qu'un seul panier actif (userId unique en NoSQL).
- 6.2 Un produit ajouté doit exister et être disponible (vérifier stock via catalog).
- 6.3 La quantité doit être >=1 (max 99 pour limiter abus).
- 6.4 Le panier se vide après validation de la commande (après POST /orders).
- 6.5 Amélioration : Stocké en NoSQL (MongoDB, cart:3004) avec TTL 30 jours pour éphémère et scalabilité.
- 6.6 Amélioration : Mise à jour quantité via PUT /cart/items/:productId (0 = suppression).
- 6.7 Amélioration : Calcul totalAmount en temps réel (currency unifiée).

7. Paiements

- 7.1 Un paiement est associé à une seule commande (1:1, orderId unique).
- 7.2 Une commande ne peut avoir qu'un seul paiement (vérifié en SQL).
- 7.3 Le montant doit être égal au total de la commande (validation amount = total).
- 7.4 Le paiement doit être approuvé (statut "success" via webhook).

- 7.5 Le statut du paiement met à jour celui de la commande (pending → confirmed).
- 7.6 Amélioration : Stocké en SQL (PostgreSQL, payments:3005) pour conformité PCI-DSS et audits.
- 7.7 Amélioration : Providers (stripe/paypal) avec redirectUrl et confirmation webhook (POST /payments/:paymentId/confirm).
- 7.8 Amélioration : Gestion des échecs (statut "failed" libère stock).

8. Stock

- 8.1 Le stock doit être un entier >=0 (champ stock en produit NoSQL).
- 8.2 Une commande ne peut être validée que si le stock est suffisant (vérification atomique lors de POST /orders).
- 8.3 Lorsqu'une commande est validée, le stock est décrémenté (update via catalog-service).
- 8.4 Si le stock passe sous le seuil d'alerte (ex. : 5), une notification est générée (au vendeur).
- 8.5 Amélioration : Réservation temporaire lors d'ajout panier (pour éviter survente, timeout 15min).
- 8.6 Amélioration : Restauration stock si commande annulée/échouée.
- 8.7 Amélioration : Multi-entrepôts possible (extension : stock par location).

9. Retours

- 9.1 Un retour est initié par l'utilisateur (après "delivered").
- 9.2 Un retour est associé à une commande (ou item spécifique).
- 9.3 Un retour peut être accepté ou refusé (par vendeur/admin).
- 9.4 Amélioration : Motif obligatoire (ex. : defectueux, erreur taille), avec remboursement automatique si accepté.
- 9.5 Amélioration : Mise à jour stock si retour accepté (incrément).
- 9.6 Amélioration : Délai max 14 jours (conforme droit UE), statut "returned".

10. Catégories

- 10.1 Une catégorie doit avoir un nom unique (libelle unique).
- 10.2 Une catégorie peut contenir plusieurs produits (relation n:m via categoryIds).
- 10.3 Une catégorie ne peut être supprimée si elle contient des produits (vérifier childrenCount).
- 10.4 Amélioration : Hiérarchie (parentId, path pour arbre), stockée en NoSQL pour flexibilité.
- 10.5 Amélioration : Recherche par catégorie (GET /categories/:categoryId avec produits paginés).
- 10.6 Amélioration : Description et images pour catégories (extension API).

11. Administration

- 11.1 Un administrateur peut gérer les comptes utilisateurs et vendeurs (PUT /users/:userId/role, DELETE /users/:userId).
- 11.2 Un administrateur peut gérer les produits et commandes (tous endpoints avec rôle admin).
- 11.3 Un administrateur applique les règles de gestion (ex. : suspendre boutique).
- 11.4 Amélioration : Accès à toutes commandes (GET /orders sans filtre user).
- 11.5 Amélioration : Logs et audits (ex. : changements statuts tracés).
- 11.6 Amélioration : Dashboards stats (extension : ordersCount global).

12. Adresses (nouvelle section enrichie)

- 12.1 Un utilisateur peut avoir plusieurs adresses (billing/shipping, via POST /addresses).
- 12.2 Une adresse doit inclure line1, city, postalCode, country (ISO, requis).
- 12.3 Une adresse peut être par défaut (isDefault boolean).
- 12.4 Stockées en SQL (PostgreSQL, addresses:3006) pour relations fortes avec users/orders.
- 12.5 Requis pour commandes (shippingAddressId, billingAddressId).
- 12.6 Amélioration : Validation format (ex. : postalCode regex par pays).
- 12.7 Amélioration : Soft delete pour historique.

13. Avis / Notes (nouvelle section enrichie)

- 13.1 Un utilisateur peut laisser un avis sur un produit acheté (après "delivered").
- 13.2 Un avis inclut note (1-5), commentaire (max 1000 chars).
- 13.3 La moyenne est calculée automatiquement (avgRating, reviewsCount dans produit).
- 13.4 Avis stockés en NoSQL (extension catalog pour scalabilité).
- 13.5 Amélioration : Modération admin (flag abusif).
- 13.6 Amélioration : Un avis par produit par commande (unique).

14. Coupons / Promotions (nouvelle section enrichie)

- 14.1 Un coupon peut être appliqué lors de commande (couponCode optionnel).
- 14.2 Réduction fixe ou pourcentage (vérifié lors POST /orders).
- 14.3 Coupons gérés par admin/vendeur (expiration, usage unique).
- 14.4 Stockés en SQL pour transactions (extension orders).
- 14.5 Amélioration : Validation (ex. : min achat, produits éligibles).
- 14.6 Amélioration : Impact sur totalAmount.

15. Considérations Générales (nouvelle section enrichie)

- 15.1 SQL (PostgreSQL) pour sections critiques : Users, Orders, Payments, Addresses (ACID, relations, audits).
- 15.2 NoSQL (MongoDB) pour scalable : Products, Categories, Cart (recherche, flexibilité, TTL).
- 15.3 Sécurité : JWT avec X-User-Id/Role, HTTPS, RGPD (consentement data).
- 15.4 Scalabilité : Pagination partout (page/limit), cache (Redis pour cart).
- 15.5 Erreurs : Standardisées {error: message, code: ERR_XXX} (ex. : 400 validation).
- 15.6 Internationalisation : Currency multi (EUR default), langues (extension).
- 15.7 Tests : Unitaires par service, intégration flux (ex. : achat complet).
- 15.8 Amélioration globale : Ajout microservice notifications (emails/SMS pour statuts).