Découpage en microservices

Bounded Contexts et Microservices identifiés :

1. User Service (Contexte : Gestion des identités)

- **Responsabilités** : Authentification, autorisation, gestion des profils, wallet utilisateur
- **API** : POST /users/register, POST /users/login, GET /users/{id}, PUT /users/{id}/profile, GET /users/{id}/wallet
- **Événements publiés** : UserRegistered, UserProfileUpdated, WalletBalanceChanged
- **Base de données** : PostgreSQL (données structurées, ACID)

2. Catalog Service (Contexte : Gestion du catalogue)

- **Responsabilités** : CRUD des articles, gestion des catégories, statut des articles
- **API** : POST /articles, GET /articles/{id}, PUT /articles/{id}, DELETE /articles/{id}, GET /categories
- **Événements publiés** : ArticlePublished, ArticleUpdated, ArticleDeleted, ArticleSold
- **Événements consommés** : TransactionCompleted (pour mettre à jour le statut)
- **Base de données** : MongoDB + Redis (cache)
- **Justification MongoDB :** Les articles ont des attributs variables selon la catégorie (taille pour vêtements, pointure pour chaussures, dimensions pour accessoires). MongoDB offre la flexibilité nécessaire pour gérer ces schémas hétérogènes sans migrations complexes.

3. Search Service (Contexte : Recherche et découverte)

- **Responsabilités** : Recherche plein texte, filtrage, recommandations, trending
- **API** : GET /search?q=..., GET /recommendations/{userId}, GET /search/suggestions
- **Événements consommés** : ArticlePublished, ArticleUpdated, UserInteractionLogged
- **Base de données** : Elasticsearch (recherche full-text) + Redis (cache)

4. Transaction Service (Contexte : Gestion des transactions)

- **Responsabilités** : Orchestration des transactions, suivi du cycle de vie, calcul des frais
- **API** : POST /transactions, GET /transactions/{id}, PUT /transactions/{id}/status, PATCH /transactions/{id}/cancel
- **Événements publiés** : TransactionCreated, TransactionCompleted, TransactionCancelled
- **Événements consommés** : PaymentCompleted, PaymentFailed, DeliveryShipped, DeliveryDelivered
- **Base de données** : PostgreSQL + Event Store (saga pattern)
- **Pattern** : Saga orchestration-based avec compensation automatique

5. Payment Service (Contexte : Gestion des paiements)

- **Responsabilités** : Traitement des paiements, gestion du wallet, remboursements
- **API** : POST /payments/process, POST /payments/refund
- **Événements publiés** : PaymentCompleted, PaymentFailed, PaymentRefunded
- **Événements consommés** : TransactionCreated
- **Base de données** : PostgreSQL (haute cohérence requise)
- **Intégration** : Stripe API (paiement par carte), PayPal API (paiement alternatif), Wallet interne (solde utilisateur)

-

6. Delivery Service (Contexte : Gestion des livraisons)

- **Responsabilités** : Génération d'étiquettes, suivi des colis, intégration transporteurs
- **API** : POST /deliveries, GET /deliveries/{id}/tracking
- **Événements publiés** : DeliveryCreated, DeliveryShipped, DeliveryDelivered
- **Événements consommés** : TransactionCompleted
- **Base de données** : PostgreSQL
- **Intégration** : API Colissimo, Mondial Relay, Chronopost API.

7. Messaging Service (Contexte : Communication inter-utilisateurs)

- **Responsabilités** : Messagerie en temps réel, conversations, historique
- **API** : POST /conversations, POST /messages, GET /conversations/{id}/messages, GET /users/{userId}/conversations
- **Événements publiés** : MessageSent, ConversationCreated
- **Base de données** : MongoDB (flexibilité pour messages) + Redis (temps réel)
- **WebSocket** : Pour communication temps réel

8. Notification Service (Contexte : Notifications)

- **Responsabilités** : Envoi de notifications push, email, in-app
- **API** : POST /notifications/send, GET /users/{userId}/notifications
- **Événements consommés**: Tous les événements métier (pour notifications appropriées)
- **Base de données** : MongoDB (notifications flexibles)
- **Intégration** : Firebase Cloud Messaging (push mobile), SendGrid (emails), Twilio (SMS optionnel)

9. Evaluation Service (Contexte : Réputation)

- **Responsabilités** : Gestion des notes et avis, calcul de réputation
- **API** : POST /evaluations, GET /users/{id}/evaluations, GET /users/{id}/rating
- **Événements publiés** : EvaluationCreated, UserReputationUpdated
- **Événements consommés** : TransactionCompleted
- **Base de données** : PostgreSQL

10. Favorite Service (Contexte : Préférences utilisateur)

- **Responsabilités** : Gestion des favoris, recommandations personnalisées, suivi d'articles
- **API** : POST /favorites, DELETE /favorites/{id}, GET /users/{id}/favorites, GET /articles/{articleId}/likes/count
- **Événements publiés** : ArticleFavorited, ArticleUnfavorited
- **Événements consommés** : ArticleDeleted` (supprimer favoris associés)
- **Base de données** : Redis (accès rapide) + PostgreSQL (persistance)

11. Moderation Service (Contexte : Administration)

- **Responsabilités** : Gestion des signalements, modération de contenu, tableau de bord admin
- **API** : POST /reports, GET /reports, PUT /reports/{id}/resolve, POST /users/{userId}/ban, GET /admin/status
- **Événements publiés** : UserBanned, ArticleModerated

- **Événements consommés** : ArticlePublished (pour modération automatique), tous les événements (pour détection de fraude)
- **Base de données** : PostgreSQL
- **ML** : Détection automatique de contenu inapproprié

Flux principaux et interactions :

Flux 1 : Publication d'un article

1. User → API Gateway → Catalog Service (REST)
2. Catalog Service → Message Bus : ArticlePublished event
3. Search Service consomme l'événement → indexation dans Elasticsearch
4. Notification Service consomme l'événement → notifie les followers

Flux 2 : Achat d'un article (Saga distribuée)

1. Buyer → API Gateway → Transaction Service : CreateTransaction
2. Transaction Service → Message Bus : TransactionCreated
3. Payment Service consomme → traite le paiement → PaymentCompleted
4. Transaction Service orchestre → met à jour le statut
5. Delivery Service consomme → génère étiquette → DeliveryCreated
6. Catalog Service consomme → met à jour article (sold)
7. Notification Service → notifie acheteur et vendeur

Flux 3 : Messagerie en temps réel

1. User → WebSocket → Messaging Service
2. Messaging Service stocke message → MessageSent event
3. Notification Service → push notification au destinataire