TD2 — Architecture Logicielle

Conception d'une plateforme E-commerce : ShopFast

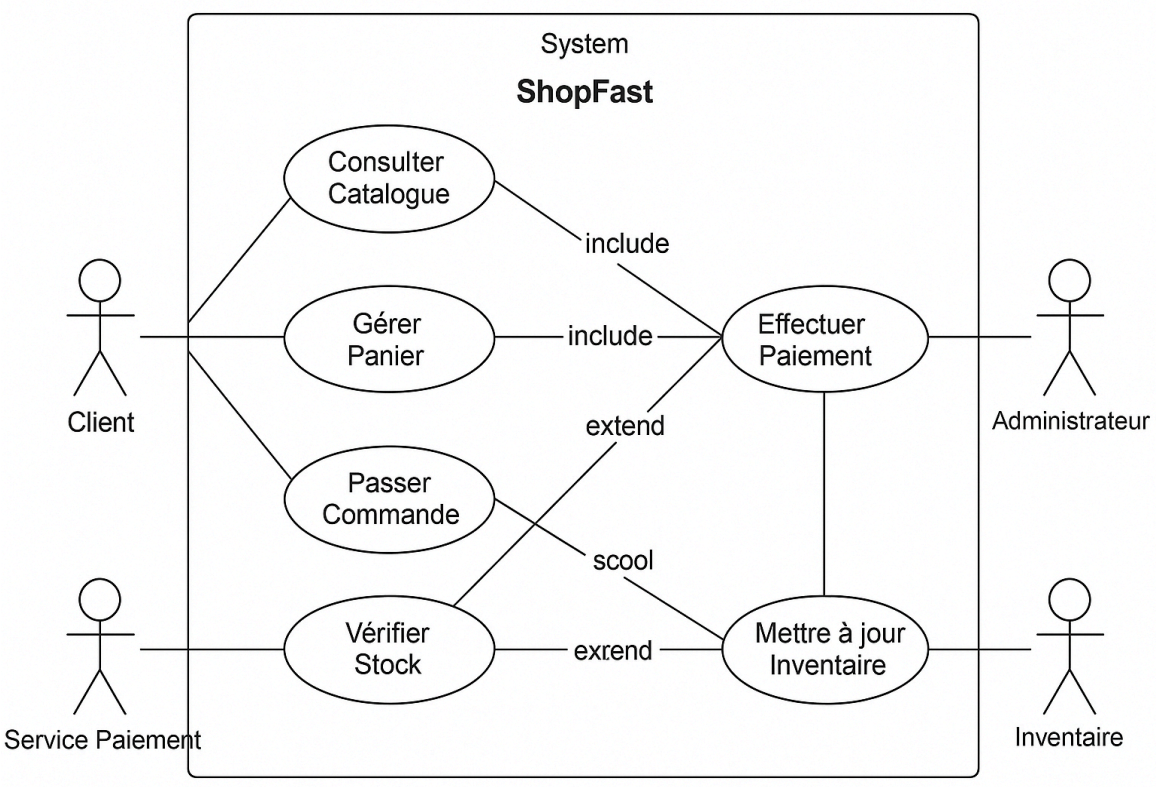
Enseignant : Dr. Ghazala Hcini
Auteur : Wajiha Missaoui
Ghalia Mahdhaoui

Contexte

Modules métier clés : Catalogue, Panier & Commandes, Authentification & Utilisateurs, Paiement (service externe), Inventaire (service externe).

Scénarios critiques :
- Consultation massive du catalogue (performance)
- Validation et paiement de commande (fiabilité)

1) Vue des cas d'utilisation



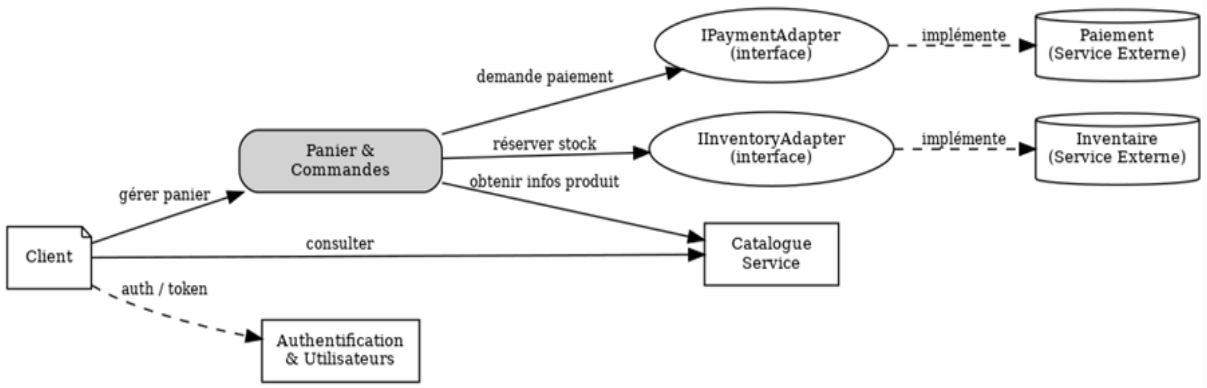
Acteurs : Client, Administrateur, Système de paiement externe, Service d'inventaire externe.

Fonctionnalités principales :

- Consulter catalogue
- Ajouter produit au panier
- Passer commande (validation + paiement)
- Gérer compte utilisateur
- Gérer inventaire

2) Vue logique:

Composants proposés : CatalogueService, CartService, OrderService, AuthService, IPaymentAdapter, IInventoryAdapter.



Tâche 1.1 — Découplage par Interface :

L'utilisation d'une interface IPaymentAdapter permet d'injecter dynamiquement l'implémentation (ex: StripeAdapter, PaypalAdapter) sans modifier OrderService. Ceci respecte le principe d'inversion des dépendances (DIP) et facilite les tests et la substitution du fournisseur.

Tâche 1.2 — Cohésion et Couplage :

Organisation interne proposée pour Panier & Commandes : CartManager, PricingPolicy, PromotionEngine, OrderValidator, OrderOrchestrator. Accès aux autres modules uniquement via interfaces.

3) Vue des processus (dynamique):

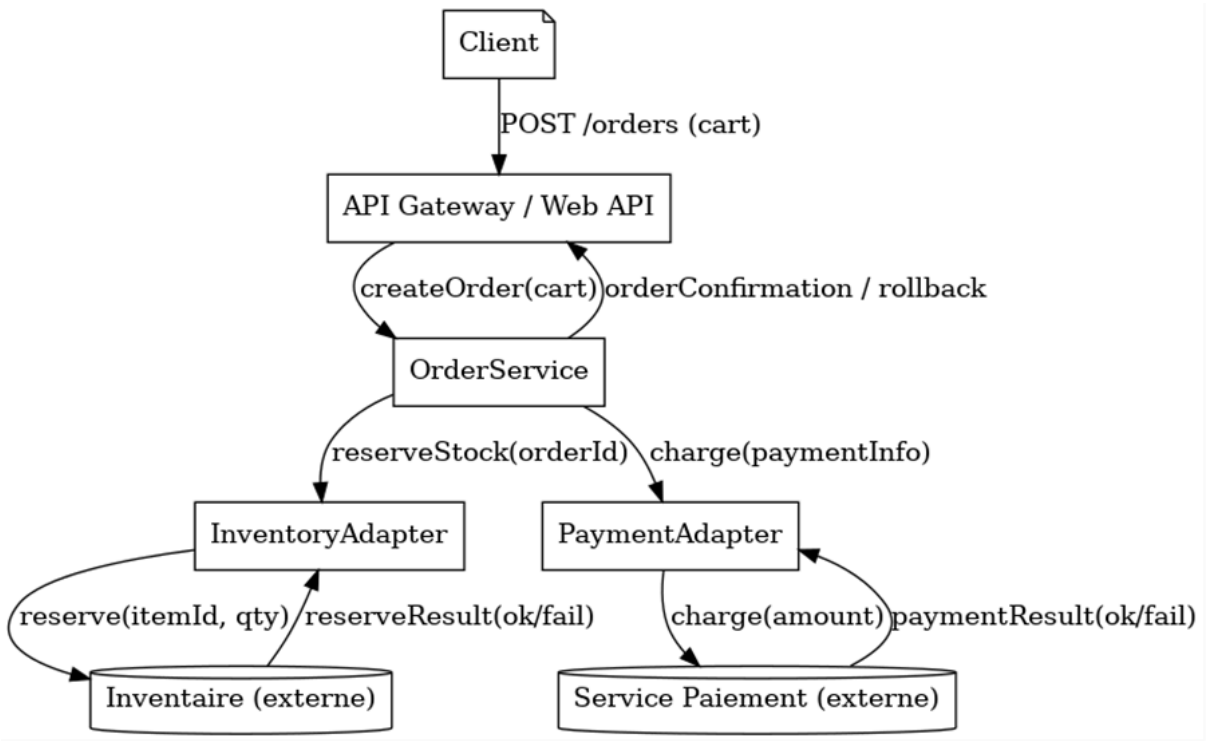
Tâche 2.1 — Inversion de dépendance (DIP) :

ValidationCommande dépend d'une interface IInventoryAdapter. Les modules de bas niveau implémentent cette interface. L'interface est empaquetée dans shopfast.api.inventory utilisé par les deux parties.

Tâche 2.2 — Scalabilité et performance (Consultation massive du catalogue) :

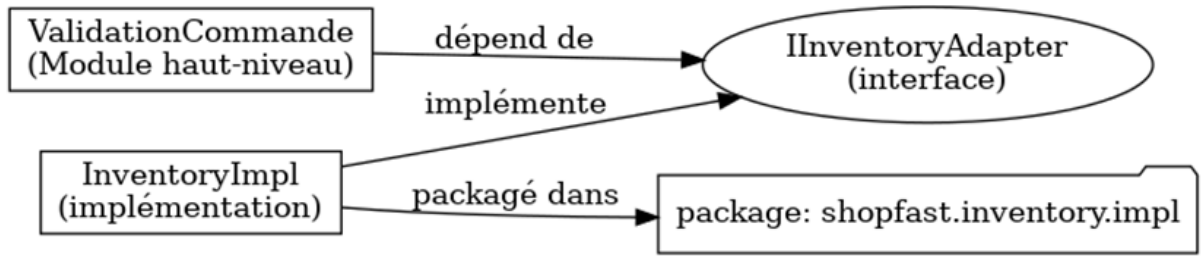
- Caching (CDN + Redis)
- Pagination et indexation (Elasticsearch)
- Réplication en lecture
- Circuit breaker
- Considérer CQRS pour séparation lecture/écriture.

Temps de réponse cible: <200 ms; sous pics: <500 ms si cacheable.



4) DIP — Packaging:

Illustration du principe DIP : empilement des interfaces dans un package API partagé et des implémentations dans des packages séparés pour permettre le rebuild uniquement du module modifié.



5) Vue de réalisation (développement):

Tâche 3.1 — Choix de granularité :

Architecture Service-Based (3 à 12 services) recommandée pour ShopFast. Services proposés : auth-service, catalog-service, cart-order-service, payment-facade, inventory-service, admin-service.

Tâche 3.2 — Modularité et réutilisation :

- shopfast.api.* (interfaces, DTOs)
- shopfast.core.* (logique métier)
- shopfast.impl.* (implémentations)
- shopfast.web.* (controllers/API)

6) Vue de déploiement:

Tâche 4.1 — Stratégie de mise à l'échelle :

Horizontal scaling (scale out) via réplication d'instances derrière un Load Balancer et autoscaling.

Tâche 4.2 — Interopérabilité et élasticité :

Utiliser Kubernetes / autoscaling groups, métriques (latence, CPU), cooldown windows et forecast pour éviter oscillations.