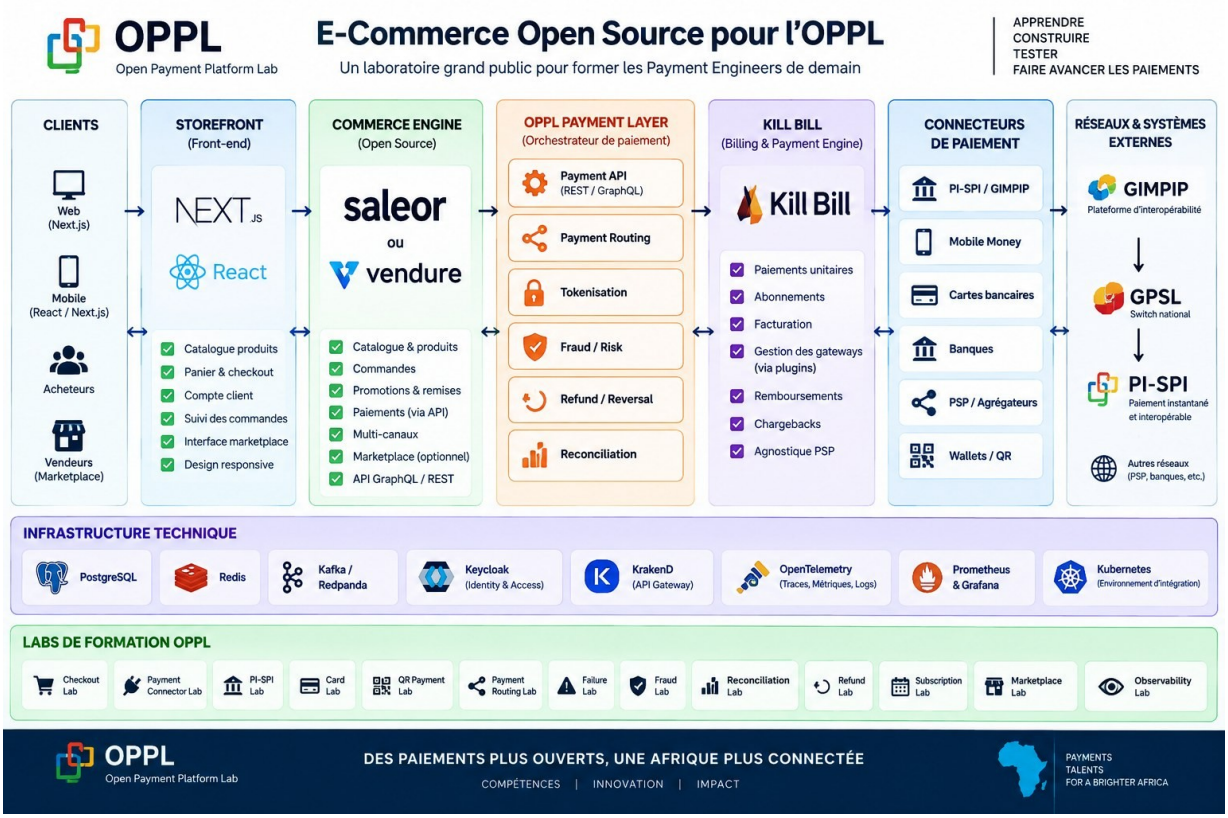
MISE EN ŒUVRE · DÉPLOIEMENT · DÉVELOPPEMENT · FORMATION

E-Commerce Open Source pour l'OPPL

Un laboratoire pour former les Payment Engineers de demain.

Apprendre · Construire · Tester · Faire avancer les paiements.



Document	Guide de référence — OPPL E-Commerce Lab
Version	1.0 — 13 septembre 2026
Auteur	Lionel SAKITI — Professionnel des infrastructures numériques
Statut	Document de travail — environnement non productif, données synthétiques
Gouvernance	Rédaction assistée par IA, gouvernée par ISO/IEC 42001 ; validation humaine requise

Des paiements plus ouverts, une Afrique plus connectée

Sommaire

1. Synthèse exécutive **4**
2. Vision, objectifs et périmètre **6**
2.1 Le e-commerce comme Payment Engineering Sandbox 6
2.2 Objectifs 6
2.3 Ce que le Lab est, et ce qu'il n'est pas 6
2.4 Publics cibles 6
2.5 Principes directeurs 7
2.6 Périmètre 7
3. Architecture de référence **8**
3.1 Les sept couches 8
3.2 Principes d'intégration 8
3.3 Le flux nominal 9
3.4 Le modèle de statuts 9
3.5 Matrice des interfaces 10
4. Choisir le moteur de commerce **11**
4.1 Saleor 11
4.2 Vendure 11
4.3 Comparaison et recommandation 12
5. Infrastructure et environnement **14**
5.1 Trois niveaux d'environnement 14
5.2 Pile technique 14
5.3 Dimensionnement indicatif (niveau 2) 15
5.4 Topologie et cloisonnement 15
5.5 Organisation des dépôts 16
5.6 Points d'accès 16
5.7 Données synthétiques 17
6. Déploiement pas à pas **18**
6.1 Phase 0 — Socle 18
6.2 Phase 1 — Identité et passerelle 19
6.3 Phases 2 et 3 — Données, événements, observabilité 19
6.4 Phase 4 — Moteur de commerce et storefront 20
6.5 Phase 5 — Couche de paiement OPPL 20
6.6 Phase 6 — Kill Bill 21
6.7 Phase 7 — Application de paiement du moteur de commerce 21
6.8 Phase 8 — Connecteurs et simulateurs 22
6.9 Vérification globale 23
7. La couche de paiement OPPL **24**
7.1 Le modèle canonique 24
7.2 L'API de paiement 24
7.3 L'idempotence, en pratique 25
7.4 Le routage 25
7.5 La tokenisation 26
7.6 La détection de fraude 27
7.7 Le rapprochement 27
7.8 Les événements 28
8. Intégrer le moteur de commerce **29**
8.1 Le principe : le commerce demande, le paiement décide 29
8.2 L'application de paiement Saleor 29
8.3 Les autres événements à traiter 30
8.4 Le plugin Vendure 30
9. Kill Bill : facturation et exécution des paiements **32**
9.1 Pourquoi un moteur dédié 32
9.2 Le plugin de paiement OPPL 32
9.3 Abonnements et relances 33
9.4 Marketplace : répartition et règlement 33
10. Les connecteurs de paiement **35**
10.1 Un cadre commun 35
10.2 Connecteur paiement instantané (cible PI-SPI) 35
10.3 Connecteur Mobile Money 36

10.4 Connecteur carte....37
10.5 Connecteurs QR, PSP et banque....37
10.6 Industrialiser : l'usine à connecteurs....37
11. Les labs de formation....39
11.1 Checkout Lab....39
11.2 Payment Connector Lab....39
11.3 Card Lab....40
11.4 QR Payment Lab....40
11.5 PI-SPI Lab....40
11.6 Payment Routing Lab....40
11.7 Failure Lab....41
11.8 Fraud Lab....41
11.9 Refund Lab....41
11.10 Reconciliation Lab....42
11.11 Subscription Lab....42
11.12 Marketplace Lab....42
11.13 Observability Lab....42
11.14 Progression et évaluation....43
12. Tests, résilience et performance....44
12.1 Pyramide de tests....44
12.2 Catalogue de tests de référence....44
12.3 Injection de pannes....45
12.4 Performance....45
13. Sécurité, conformité et licences....47
13.1 Sécurité par défaut....47
13.2 Maîtriser le périmètre de conformité carte....47
13.3 Cadre réglementaire régional....47
13.4 Conformité des licences....48
14. Payment Engineers et offres de service....50
14.1 Le référentiel de compétences....50
14.2 Les quatre profils visés....50
14.3 Offres de service envisageables....50
15. Gouvernance et feuille de route....52
15.1 Organisation....52
15.2 Feuille de route....52
15.3 Risques et parades....52
15.4 Indicateurs de réussite....53
Annexes....54
Annexe A — Glossaire....54
Annexe B — Check-list de mise en service....54
Annexe C — Références....55

1. Synthèse exécutive

L'**OPPL E-Commerce Lab** transforme une plateforme de commerce en ligne open source en **banc d'essai d'ingénierie des paiements**. L'objectif n'est pas de construire une boutique de plus, encore moins un concurrent des grandes plateformes de commerce : c'est de disposer d'un environnement réel, complet et reproductible dans lequel des équipes peuvent apprendre, construire, casser, observer, corriger et industrialiser une chaîne de paiement de bout en bout.

La chaîne assemblée va du clic du client jusqu'au règlement : storefront Next.js, moteur de commerce headless (Saleor ou Vendure), **couche de paiement OPPL** (API, routage, tokenisation, fraude, remboursement, rapprochement), moteur de facturation et de paiement Kill Bill, puis une famille de connecteurs — carte, Mobile Money, QR, PSP, banques et paiement instantané — vers les réseaux de la zone UEMOA. Le tout repose sur un socle PostgreSQL, Redis, Kafka, Keycloak, KrakenD, OpenTelemetry, Prometheus, Grafana et Kubernetes.

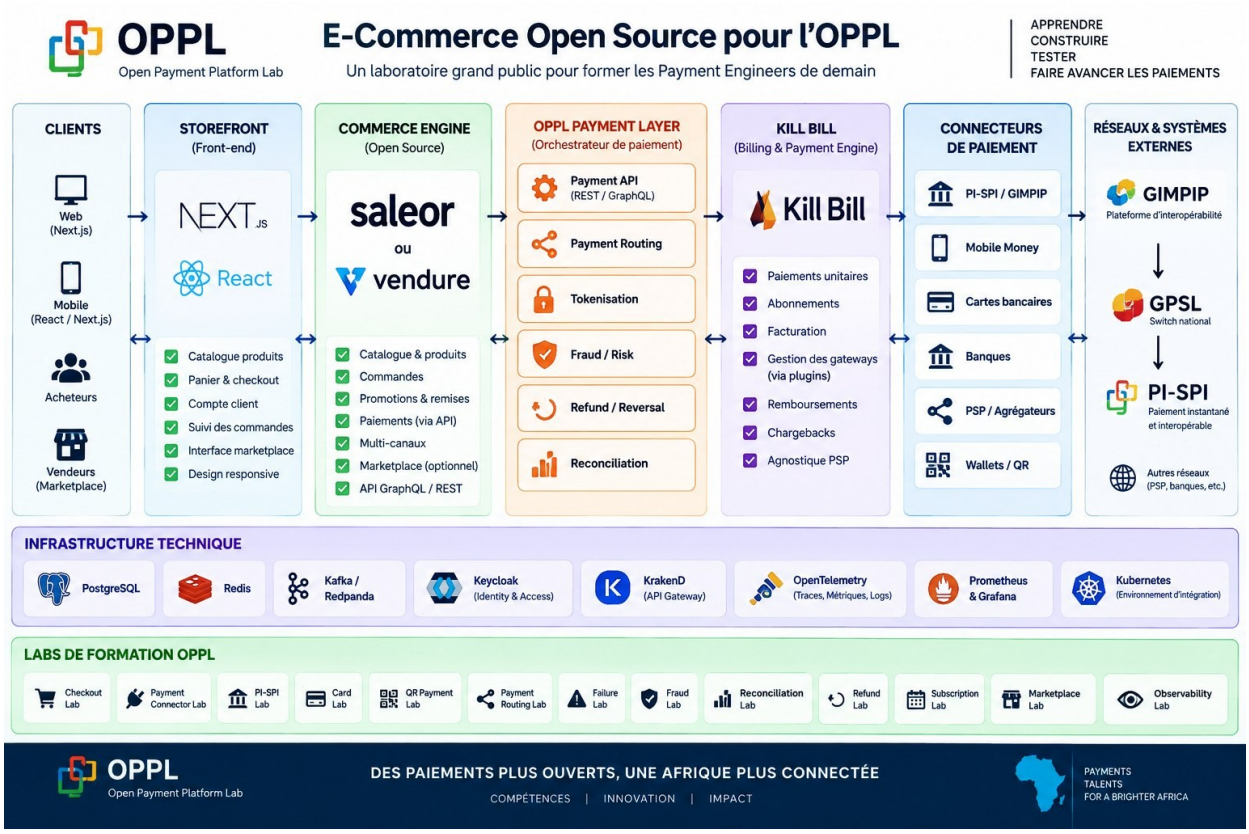


Figure 1 — Architecture cible de l'OPPL E-Commerce Lab

Le e-commerce est un excellent cas d'usage d'apprentissage parce qu'il concentre, sur une seule commande, presque toutes les difficultés du métier : choix du moyen de paiement, authentification, autorisation, capture, confirmation asynchrone, remboursement total ou partiel, annulation, impayé, rapprochement, règlement vendeur, abonnement, fraude, idempotence et observabilité. Un apprenant qui maîtrise une commande de 25 000 FCFA de bout en bout possède les fondamentaux d'un **Payment Engineer**.

Tableau 1 — Apports du Lab par dimension

Dimension	Ce que le Lab apporte
Compétences	Un parcours de 13 labs qui conduit un profil BAC+3 vers le métier de Payment Engineer par la pratique, pas par la théorie.
Ingénierie	Une architecture de référence testable : contrats d'API, idempotence, gestion des erreurs, réconciliation, observabilité.

Dimension	Ce que le Lab apporte
Industrialisation	Une usine à connecteurs : un cadre commun, des tests, un pipeline, une méthode de certification.
Décision	Une base factuelle pour arbitrer Build, Buy, Reuse, Extend ou Replace, composant par composant.
Souveraineté	Maîtrise du code, des données et du déploiement ; capacité à contribuer en amont aux projets utilisés.
Écosystème	Un sandbox partageable avec banques, fintechs, PSP, agrégateurs et e-commerçants de la zone.

Positionnement du document

L'OPPL E-Commerce Lab est un environnement **non productif**. Il manipule exclusivement des données synthétiques, des numéros de carte de test et des simulateurs de réseaux. Aucune connexion à un système réel n'est établie depuis le Lab.

Les captures d'écran de ce guide sont des **maquettes illustratives reconstituées** à partir des interfaces et sorties documentées des outils ; les chiffres qui y figurent sont fictifs. Elles sont destinées à être remplacées par les captures réelles de votre instance.

Les références à GIMPIP, GPSL et PI-SPI décrivent une **cible d'intégration**, pas une implémentation. Les spécifications de ces systèmes ne sont pas publiques et toute connexion effective relève des processus d'homologation de leurs opérateurs.

2. Vision, objectifs et périmètre

2.1 Le e-commerce comme Payment Engineering Sandbox

Former des ingénieurs paiement se heurte à un obstacle pratique : les systèmes réels sont fermés. On n'apprend pas la monétique sur un switch de production, ni l'ISO 20022 sur une plateforme interbancaire. Les formations se replient alors sur la théorie, et les équipes découvrent les vraies difficultés en incident de production.

Le e-commerce résout ce problème. C'est le seul domaine où l'on peut, légalement et sans risque, construire une chaîne de paiement complète de bout en bout avec des composants ouverts : un vrai panier, un vrai checkout, de vraies transactions, de vrais remboursements, de vrais écarts de rapprochement. Le commerce devient le prétexte ; le paiement est la matière.

Le principe directeur du Lab tient en une phrase : **APPRENDRE → CONSTRUIRE → CASSER → OBSERVER → CORRIGER → INDUSTRIALISER**. Chaque lab reprend ce cycle sur un thème différent.

2.2 Objectifs

- **Former** des Payment Engineers, Payment Architects, Payment SRE et Payment Platform Engineers capables d'intervenir sur une infrastructure de paiement réelle.
- **Expérimenter** les architectures : orchestration, routage multi-prestataires, tokenisation, sagas, réconciliation, détection de fraude.
- **Industrialiser** des connecteurs vers les systèmes de paiement de la zone : cartes, Mobile Money, wallets, QR, banques, paiement instantané.
- **Tester et certifier** l'intégration de partenaires dans un environnement mutualisé et instrumenté.
- **Éclairer les décisions** d'architecture par des mesures reproductibles plutôt que par des présentations commerciales.

2.3 Ce que le Lab est, et ce qu'il n'est pas

Tableau 2 — Positionnement du Lab

Le Lab EST	Le Lab N'EST PAS
Un environnement d'apprentissage et d'expérimentation par la pratique	Une boutique en ligne destinée à vendre, ni un concurrent des plateformes commerciales
Une architecture de référence intégrée, versionnée et reproductible	Une plateforme de production ni un produit prêt à l'emploi
Une usine à connecteurs et un banc de certification d'intégration	Un organisme de certification réglementaire ou de schéma
Un sandbox mutualisable avec l'écosystème régional	Un environnement traitant des données clients réelles ou des PAN réels
Un outil de dialogue entre DSI, métiers, risques, conformité et partenaires	Une réplique des systèmes GIMPIP, GPSL ou PI-SPI

2.4 Publics cibles

- **Apprenants et jeunes diplômés** (BAC+3 et plus) visant le métier de Payment Engineer.
- **Équipes Software et Intégration** des banques, EME, fintechs, PSP et agrégateurs.

- **Équipes Platform Engineering et SRE** chargées du socle, de l'observabilité et de la résilience.
- **Architectes** cherchant à valider des patterns avant de les engager en production.
- **Risques, conformité et sécurité**, pour comprendre concrètement les contrôles et la traçabilité.
- **E-commerçants et éditeurs** souhaitant tester leur intégration avant mise en service.

2.5 Principes directeurs

Tableau 3 — Principes directeurs

Principe	Traduction concrète
Le paiement d'abord	Le commerce est le décor ; toutes les décisions d'architecture sont prises du point de vue du paiement.
Standards ouverts	ISO 20022, ISO 8583, OAuth2 / OIDC, OpenAPI, GraphQL, CloudEvents, OpenTelemetry, W3C Trace Context.
Tout est code	Infrastructure, configuration, jeux de données, politiques et scénarios de test sont versionnés dans Git.
Idempotence partout	Toute opération mutante porte une clé d'idempotence propagée jusqu'au moteur de paiement.
Données synthétiques	Aucune donnée réelle. Générateur reproductible, PAN de test, alias et numéros fictifs.
Observabilité native	Une commande doit pouvoir être suivie sur une trace unique, du clic au règlement.
Décisions documentées	Chaque arbitrage fait l'objet d'un ADR daté, argumenté et, si possible, mesuré.

2.6 Périmètre

Dans le périmètre : le storefront de démonstration, le moteur de commerce, la couche de paiement OPPL, Kill Bill, les connecteurs et leurs simulateurs de contrepartie, le socle technique, les 13 labs de formation, la chaîne de tests et d'observabilité.

Hors périmètre de la version 1.0 : toute connexion à un système réel (réseaux cartes, opérateurs Mobile Money, GIMPIP, GPSL, PI-SPI, core banking) ; la certification PCI DSS ; les HSM physiques, remplacés par un HSM logiciel de test ; la haute disponibilité multi-sites ; l'exploitation commerciale de la boutique.

3. Architecture de référence

3.1 Les sept couches

L'architecture se lit de gauche à droite, du client jusqu'aux réseaux externes. Chaque couche a une responsabilité unique et un contrat explicite avec ses voisines.

Tableau 4 — Les sept couches de l'architecture

Couche	Rôle	Composants
1. Clients	Points d'entrée : acheteurs, vendeurs marketplace, terminaux	Navigateur web, application mobile
2. Storefront	Parcours d'achat : catalogue, panier, checkout, compte, suivi	Next.js, React
3. Commerce Engine	Catalogue, commandes, promotions, marketplace, API	Saleor ou Vendure (headless)
4. OPPL Payment Layer	Cœur du Lab : API de paiement, routage, tokenisation, fraude, remboursement, rapprochement	Services développés par le Lab
5. Billing & Payment Engine	Paiements unitaires, abonnements, facturation, remboursements, impayés, abstraction PSP	Kill Bill + plugins
6. Connecteurs	Traduction vers chaque réseau : protocole, format, sécurité, statuts	Connecteurs développés par le Lab
7. Réseaux & systèmes externes	Destinataires finaux (simulés dans le Lab)	GIMPIP, GPSL, PI-SPI, PSP, banques, Mobile Money

3.2 Principes d'intégration

Six règles gouvernent les échanges entre couches. Elles constituent en elles-mêmes une part importante de l'enseignement du Lab.

- **Contrats explicites.** Chaque interface synchrone est décrite en OpenAPI 3 (REST) ou par un schéma GraphQL ; chaque flux asynchrone en AsyncAPI. Les contrats sont versionnés et testés avant intégration.
- **Idempotence de bout en bout.** La clé d'idempotence naît dans le storefront, traverse la passerelle, le moteur de commerce, la couche de paiement et Kill Bill, et parvient jusqu'au connecteur. Un rejeu ne crée jamais un second débit.
- **Un identifiant métier unique.** Le `paymentId` accompagne la commande jusqu'au règlement et sert de clé de partition Kafka, d'attribut de trace et de critère de rapprochement.
- **Aucune donnée sensible en clair.** Les numéros de carte sont tokenisés au plus tôt ; les alias et numéros de téléphone sont hachés dans les journaux, les traces et les événements.
- **Les statuts sont finis et explicites.** Un paiement est dans l'un des états du modèle du Lab ; aucun état implicite, aucun statut « inconnu » toléré sans mécanisme de réconciliation.
- **Le commerce ne pilote pas le paiement.** Le moteur de commerce demande, la couche de paiement décide. Ce découplage permet de changer de moteur de commerce sans réécrire la chaîne de paiement.

3.3 Le flux nominal

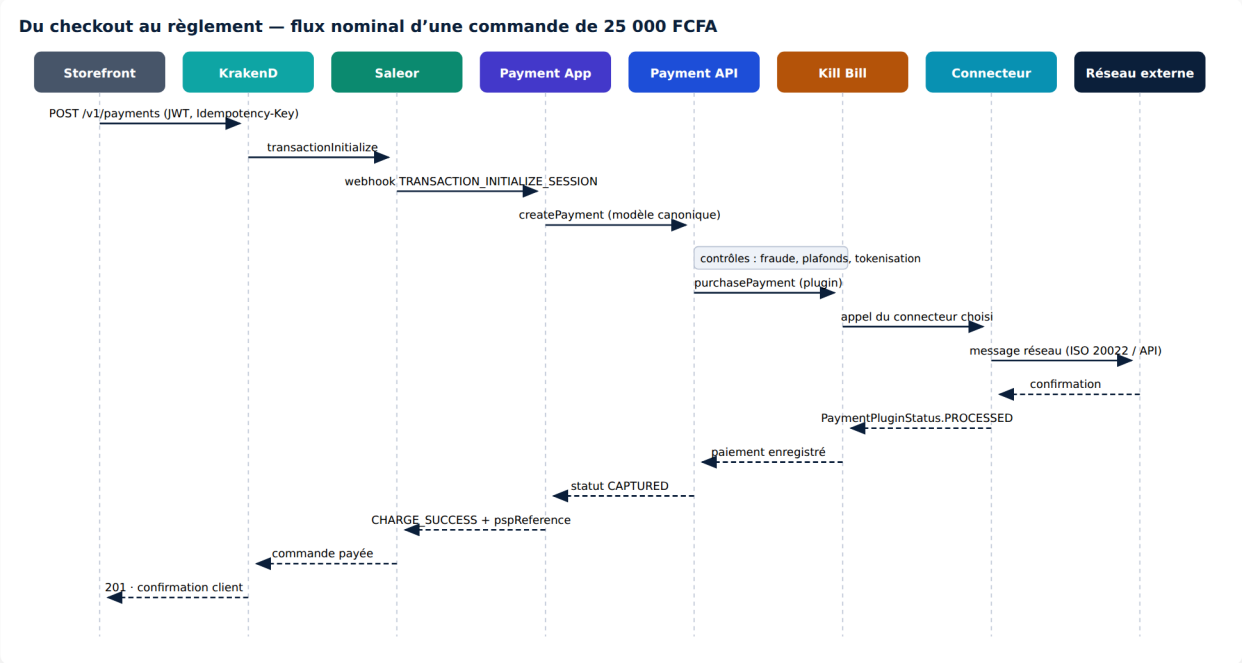


Figure 2 — Du checkout au règlement — flux nominal d'une commande de 25 000 FCFA

1. Le storefront appelle la passerelle KrakenD avec un jeton OAuth2 émis par Keycloak et une clé d'idempotence égale au numéro de commande.
2. Le moteur de commerce crée une transaction et sollicite l'application de paiement OPPL par un appel synchrone.
3. L'application de paiement traduit la demande dans le modèle canonique et appelle l'API de paiement OPPL.
4. La couche de paiement applique les contrôles (plafonds, fraude, tokenisation) puis sélectionne un connecteur par ses règles de routage.
5. Kill Bill enregistre le paiement et délègue l'exécution au plugin, qui appelle le connecteur retenu.
6. Le connecteur traduit vers le protocole du réseau cible et attend la confirmation.
7. Le statut remonte la chaîne ; la commande est marquée payée ; un événement est publié sur Kafka ; la trace est complète de bout en bout.

3.4 Le modèle de statuts

Un vocabulaire commun des statuts évite la confusion la plus fréquente dans les intégrations de paiement : chaque système utilise ses propres termes. Le Lab impose une table de correspondance unique.

Tableau 5 — Table de correspondance des statuts

Statut canonique OPPL	Signification	Saleor	Kill Bill
PENDING	Demande enregistrée, pas encore de réponse du réseau	CHARGE_REQUEST / PENDING	PENDING
ACTION_REQUIRED	Une action du client est attendue (code, 3-D Secure, scan QR)	CHARGE_ACTION_REQUIRED	PENDING
AUTHORIZED	Fonds réservés, capture à venir	AUTHORIZATION_SUCCESS	SUCCESS (AUTHORIZE)

Statut canonique OPPL	Signification	Saleor	Kill Bill
CAPTURED	Paiement encaissé	CHARGE_SUCCESS	SUCCESS (PURCHASE / CAPTURE)
FAILED	Refus définitif du réseau ou du contrôle	CHARGE_FAILURE	PAYMENT_FAILURE
CANCELLED	Autorisation annulée avant capture	CANCEL_SUCCESS	SUCCESS (VOID)
REFUNDED / PARTIALLY_REFUNDED	Remboursement total ou partiel exécuté	REFUND_SUCCESS	SUCCESS (REFUND)
UNKNOWN	Aucune réponse exploitable — à réconcilier impérativement	—	UNDEFINED (à éviter)

La règle la plus importante du Lab

Un connecteur ne doit **jamais** renvoyer un statut définitif d'échec lorsqu'il n'a pas obtenu de réponse. En cas de doute — timeout, coupure, erreur réseau — il renvoie un statut non terminal et le système déclenche une réconciliation. Confondre « échec » et « je ne sais pas » est la cause n° 1 des doubles débits et des commandes perdues.

3.5 Matrice des interfaces

Tableau 6 — Matrice des interfaces

De → Vers	Protocole	Contrat	Sécurité
Navigateur → Storefront	HTTPS	Pages et routes Next.js	Session, protection CSRF
Storefront → KrakenD	HTTPS REST	OpenAPI « OPPL Payments v1 »	OAuth2 / OIDC, PKCE
Storefront → Commerce Engine	HTTPS GraphQL	Schéma Saleor ou Vendure	Jeton client, permissions par canal
Commerce Engine → Payment App	Webhook synchrone	Événements de session de transaction	Signature, jeton d'application
Payment App → Payment API	HTTPS REST	Modèle canonique OPPL	Client credentials, mTLS interne
Payment API → Kill Bill	HTTPS REST	API Kill Bill (comptes, paiements)	Clé API + secret de tenant
Kill Bill → Plugin → Connecteur	Interne / HTTPS	Interface de plugin de paiement	Réseau interne, NetworkPolicy
Connecteur → Réseau	HTTPS / ISO 20022 / API	Guide d'implémentation par réseau	mTLS, signature, liste blanche
Tous → Kafka	Kafka	CloudEvents JSON versionnés	TLS + SASL, ACL par topic
Tous → OTel Collector	OTLP	Traces, métriques, journaux	Réseau interne

4. Choisir le moteur de commerce

Le Lab supporte deux moteurs headless : **Saleor** et **Vendure**. Ils remplissent la même fonction mais reposent sur des écosystèmes, des modèles d'extension et surtout des **licences** très différentes. Le choix n'est pas seulement technique.

4.1 Saleor

Caractéristique	Description
Nature	Plateforme de commerce headless pilotée par une API GraphQL
Technologie	Python / Django, PostgreSQL, Celery ; tableau de bord React séparé
Licence	BSD 3-Clause pour le cœur (saleor/saleor) et pour le tableau de bord (saleor-dashboard) — licence permissive, sans obligation de réciprocité
Extension paiement	Applications de paiement branchées sur l'API de transactions, via des webhooks synchrones
Atouts pour le Lab	Modèle de transaction moderne et découplé ; idempotence native dans l'API ; multi-canaux ; marketplace
Points d'attention	Empreinte mémoire plus élevée ; les webhooks synchrones imposent des délais de réponse courts

Le modèle d'extension de Saleor est celui qui se rapproche le plus d'une architecture de production réelle : le moteur de commerce ne connaît aucun prestataire de paiement, il se contente de solliciter une application externe, qui lui répond par un statut et une référence. C'est exactement la séparation que le Lab veut enseigner.

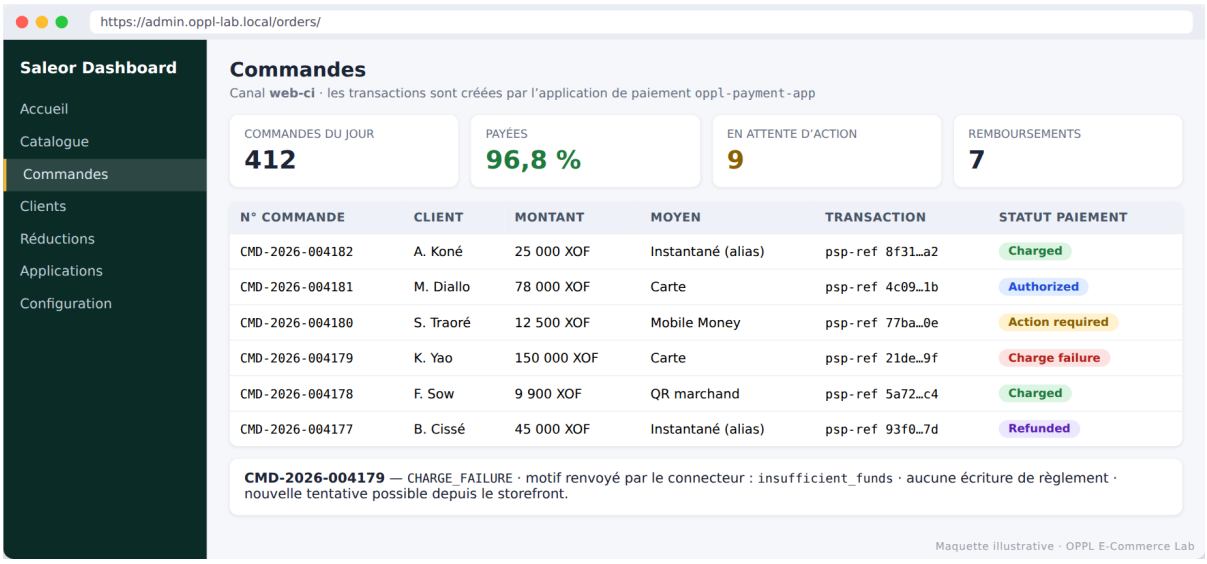


Figure 3 — Saleor Dashboard — commandes et statuts de transaction (capture illustrative)

4.2 Vendure

Caractéristique	Description
Nature	Plateforme de commerce headless TypeScript / Node.js
Technologie	NestJS, TypeORM, GraphQL ; tableau de bord Angular
Licence	Double licence : GPLv3 pour l'édition communautaire, ou licence commerciale Vendure (VCL)

Caractéristique	Description
Extension paiement	Mécanisme de plugins et de gestionnaires de moyens de paiement, appelés dans la transaction de commande
Atouts pour le Lab	Stack homogène avec le storefront ; démarrage rapide ; API de plugin très lisible
Points d'attention	La GPLv3 impose une analyse juridique pour tout usage au-delà du Lab ; appels de paiement synchrones dans la transaction

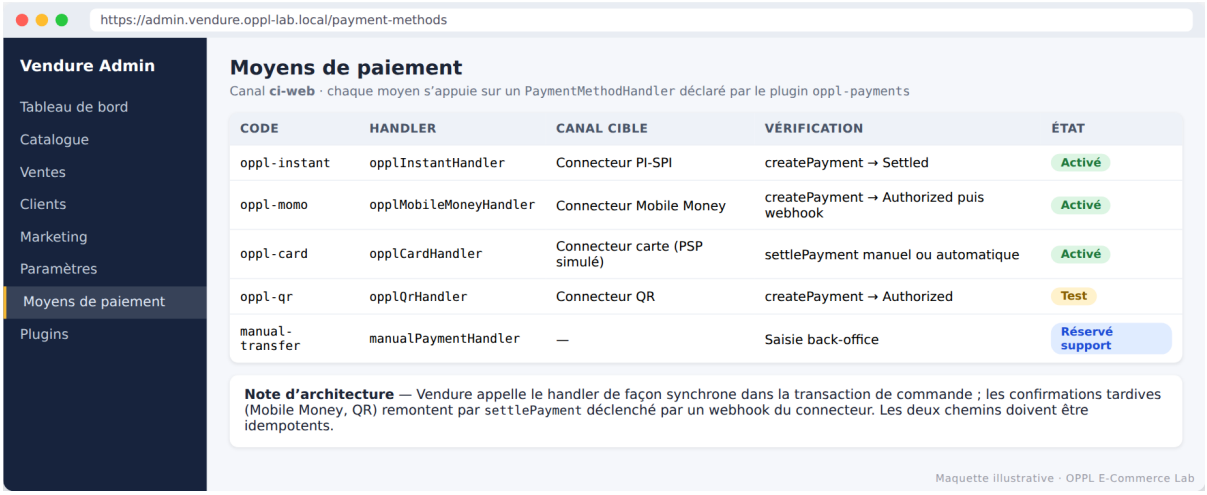


Figure 4 — Vendure Admin — moyens de paiement branchés sur le plugin OPPL (capture illustrative)

Le point de licence, vérifié

Saleor est publié sous licence BSD 3-Clause, permissive : vous pouvez modifier et déployer sans obligation de publier vos modifications.

Vendure est publié en double licence : GPLv3 pour l'édition communautaire, ou licence commerciale Vendure. La GPLv3 est une licence à réciprocité forte ; son usage dans un produit commercial dérivé exige une analyse juridique préalable, voire l'acquisition de la licence commerciale.

Ces informations doivent être revérifiées sur les dépôts officiels au moment de l'installation, les éditeurs faisant évoluer leurs modèles.

4.3 Comparaison et recommandation

Tableau 7 — Comparaison Saleor / Vendure

Critère	Saleor	Vendure
Langage	Python	TypeScript
Licence communautaire	BSD 3-Clause (permissive)	GPLv3 (réciprocité forte) ou VCL
Modèle d'intégration paiement	Application externe + webhooks synchrones	Plugin exécuté dans le serveur
Découplage commerce / paiement	Fort — proche d'une architecture réelle	Moyen — le plugin vit dans le moteur
Idempotence de l'API de paiement	Clé d'idempotence prévue dans l'API de transactions	À implémenter dans le plugin

Critère	Saleor	Vendure
Cohérence avec le storefront Next.js	Bonne (GraphQL)	Très bonne (même écosystème)
Ressources nécessaires	Plus élevées	Plus légères
Effort de mise en route	Moyen	Faible

Recommandation du Lab : déployer **Saleor comme moteur de référence**, parce que son modèle d'application de paiement externe enseigne le bon découplage et parce que sa licence permissive n'introduit aucune contrainte pour les travaux dérivés. **Conserver Vendure comme second moteur**, précisément pour démontrer que la couche de paiement OPPL est indépendante du moteur de commerce : brancher le même connecteur sur deux moteurs différents est l'un des exercices les plus formateurs du parcours.

Exercice associé

Le lab « Commerce Engine Swap » consiste à faire passer une commande identique par Saleor puis par Vendure, avec la même couche de paiement et le même connecteur, puis à comparer les traces. Si le comportement diffère, c'est que du métier de paiement a fui dans le moteur de commerce.

5. Infrastructure et environnement

5.1 Trois niveaux d'environnement

Tableau 8 — Les trois niveaux d'environnement

Niveau	Cible	Socle	Portée	Usage
1 — Poste apprenant	1 personne	Docker Compose, machine ≥ 16 Go RAM	Storefront + un moteur + la couche paiement + un connecteur	Labs individuels, développement d'un connecteur
2 — Lab de promotion	10 à 25 personnes	Kubernetes (k3s ou équivalent), 3 nœuds	Chaîne complète + simulateurs + observabilité	Parcours complet, démonstrations, certification d'intégration
3 — Sandbox mutualisé	Partenaires externes	Kubernetes multi-nœuds, isolation par tenant	Chaîne complète, quotas et cloisonnement par partenaire	Tests d'intégration partenaires, campagnes de performance

5.2 Pile technique

Tableau 9 — Pile technique du Lab

Domaine	Technologie retenue	Licence (à revérifier)	Rôle dans le Lab
Storefront	Next.js / React	MIT	Parcours d'achat et checkout de démonstration
Commerce Engine	Saleor (référence) / Vendure (second moteur)	BSD 3-Clause / GPLv3 ou commerciale	Catalogue, commandes, marketplace
Billing & Payment	Kill Bill + Kaui	Apache 2.0	Paiements, abonnements, facturation, plugins
API Gateway	KrakenD (édition communautaire)	Apache 2.0	Exposition, jetons, quotas, agrégation
IAM	Keycloak	Apache 2.0	Identités clients, marchands, services, opérateurs
Base de données	PostgreSQL	PostgreSQL License	Données de commerce et de paiement
Cache / files courtes	Redis	Voir § 13.4 — modèle de licence à vérifier selon la version	Cache, verrous, idempotence court terme
Event streaming	Apache Kafka	Apache 2.0	Événements de paiement, audit, découplage
Traces et métriques	OpenTelemetry	Apache 2.0	Instrumentation homogène de toute la chaîne
Métriques	Prometheus	Apache 2.0	Collecte et alerting
Tableaux de bord	Grafana	AGPLv3	Visualisation métier et technique
Orchestration	Kubernetes	Apache 2.0	Déploiement, isolation, quotas

Deux vigilances de licence dès le socle

Grafana est publié sous AGPLv3 depuis 2021. L'usage interne est sans difficulté ; la redistribution ou l'exposition d'un service dérivé appelle une analyse.

Redis a changé de modèle : les versions 7.2 et antérieures sont sous licence BSD 3-Clause, tandis que Redis 8 adopte un modèle tri-licence laissant le choix entre RSALv2, SSPLv1 et AGPLv3. Le choix de version n'est donc pas neutre et doit être tranché explicitement. Des alternatives compatibles existent si une licence permissive est exigée.

Redpanda, parfois proposé en remplacement de Kafka, est distribué sous Business Source License 1.1 : c'est une licence *source-available*, pas une licence open source, assortie d'une restriction d'usage. Le Lab retient Kafka pour cette raison.

5.3 Dimensionnement indicatif (niveau 2)

Tableau 10 — Dimensionnement indicatif du Lab de promotion

Domaine	vCPU	RAM	Stockage	Remarques
Socle (ingress, GitOps, secrets, registre)	4	8 Go	100 Go	Registre local recommandé
Keycloak + KrakenD	3	6 Go	20 Go	Keycloak avec sa base
Storefront Next.js	2	4 Go	10 Go	Plusieurs instances possibles
Saleor (API, worker, dashboard, base)	6	12 Go	60 Go	Poste le plus lourd de la couche commerce
Vendure (second moteur, optionnel)	2	4 Go	20 Go	Peut rester éteint hors exercice
OPPL Payment Layer (5 services)	5	8 Go	30 Go	API, routeur, tokenisation, fraude, rapprochement
Kill Bill + Kaui + base	4	8 Go	40 Go	JVM à calibrer
Connecteurs et simulateurs	4	6 Go	20 Go	Un pod par connecteur
Kafka (3 brokers)	3	6 Go	150 Go	Stockage rapide
Observabilité (OTel, Prometheus, Grafana)	5	10 Go	150 Go	Rétention à calibrer
Total indicatif	≈ 38	≈ 72 Go	≈ 600 Go	3 nœuds de 16 vCPU / 64 Go avec marge

5.4 Topologie et cloisonnement

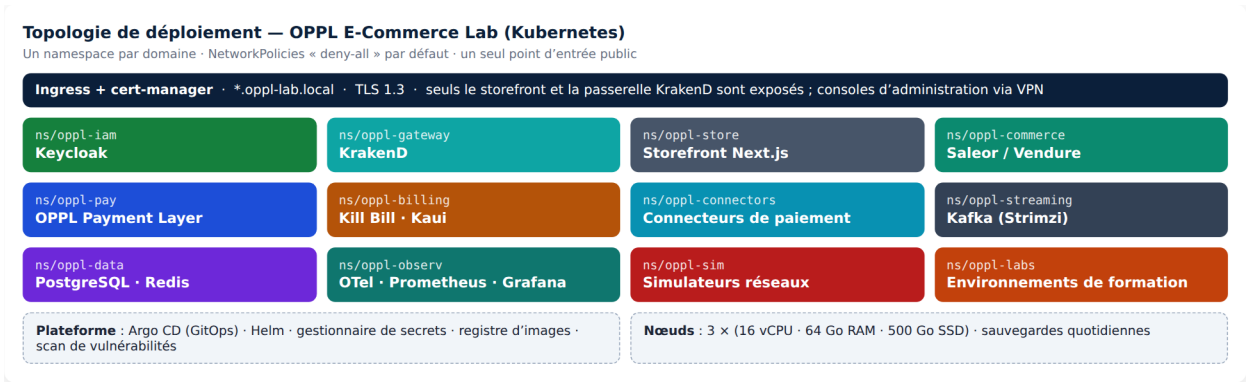


Figure 5 — Topologie du Lab — un namespace par domaine (capture illustrative)

Le cloisonnement réseau est une exigence pédagogique autant que technique : les apprenants doivent constater qu'un service ne peut appeler que ce qu'on l'a explicitement autorisé à appeler. La politique par défaut interdit tout trafic ; chaque flux de la matrice des interfaces fait l'objet d'une règle nommée et versionnée.

platform/policies/oppl-billing.yaml — seul l'API de paiement atteint Kill Bill

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-payment-api-to-killbill
  namespace: oppl-billing
spec:
  podSelector:
    matchLabels: { app: killbill }
  policyTypes: [Ingress]
  ingress:
    - from:
        - namespaceSelector:
            matchLabels: { kubernetes.io/metadata.name: oppl-pay }
          podSelector:
            matchLabels: { app: payment-api }
      ports:
        - { protocol: TCP, port: 8080 }
```

5.5 Organisation des dépôts

Arborescence des deux dépôts

```
oppl-gitops/                    # état désiré de la plateforme
├── bootstrap/                  # amorçage GitOps
├── platform/                   # ingress, certificats, secrets, registre, politiques
├── apps/
│   ├── oppl-iam/  oppl-gateway/  oppl-store/  oppl-commerce/
│   ├── oppl-pay/  oppl-billing/  oppl-connectors/
│   └── oppl-streaming/  oppl-data/  oppl-observ/  oppl-sim/
├── environments/               # niveau 1 (compose), niveau 2, niveau 3
└── docs/adr/                   # décisions d'architecture

oppl-services/                  # code développé par le Lab
├── contracts/                  # OpenAPI, AsyncAPI, schémas d'événements
├── storefront/                 # Next.js
├── payment-app-saleor/         # application de paiement Saleor
├── vendure-plugin-oppl/        # plugin Vendure
├── payment-api/                # API de paiement canonique
├── payment-router/             # moteur de règles de routage
├── tokenisation-svc/  fraud-svc/  recon-worker/
├── killbill-oppl-connector/    # plugin de paiement Kill Bill
├── connectors/                 # pi-spi, momo, card, qr, psp, bank
├── simulators/                 # contreparties simulées
├── data-generator/  labs/  perf/
```

5.6 Points d'accès

Tableau 11 — Points d'accès du Lab

Service	URL du Lab	Accès
Storefront	https://shop.oppl-lab.local	Public Lab
Passerelle API	https://api.oppl-lab.local	Public Lab (jeton requis)
Saleor Dashboard	https://admin.oppl-lab.local	VPN / rôle staff
Vendure Admin	https://admin.vendure.oppl-lab.local	VPN / rôle staff
Console OPPL Payment Layer	https://pay.oppl-lab.local	VPN / rôle ops
Kill Bill (Kaui)	https://kaui.oppl-lab.local	VPN / rôle billing
Keycloak	https://iam.oppl-lab.local	OIDC public, administration par VPN

Service	URL du Lab	Accès
Grafana	https://grafana.oppl-lab.local	VPN / rôle ops
OpenSearch Dashboards	https://observ.oppl-lab.local	VPN / rôle ops
Simulateurs de réseaux	internes uniquement	Connecteurs seulement

5.7 Données synthétiques

Le générateur produit, à partir d'une graine fixe, un catalogue de produits, des comptes clients, des vendeurs marketplace, des historiques de commandes et des profils de comportement. Il crée également les éléments propres au paiement : alias de paiement fictifs, PAN de test valides au sens de Luhn mais non émis, comptes de simulateurs, plafonds et niveaux de vérification d'identité.

- **Profils clients** : régulier, occasionnel, nouveau, à risque — chacun avec une distribution de montants et de fréquences.
- **Profils marchands** : petit commerce, forte volumétrie, marketplace multi-vendeurs.
- **Scénarios de fraude** : rafales, comptes mules, incohérences géographiques, réutilisation d'instruments.
- **Jeux d'erreurs** : fonds insuffisants, alias inconnu, timeout, double webhook, panier expiré.

Règle absolue

Aucune donnée issue d'un système de production — même anonymisée — n'entre dans le Lab sans validation formelle du responsable de la protection des données et du RSSI. L'anonymisation de données de paiement est notoirement difficile ; la génération synthétique est la voie par défaut.

6. Déploiement pas à pas

Le déploiement suit dix phases. L'ordre construit d'abord les services transverses, afin que chaque composant ajouté ensuite soit immédiatement sécurisé, tracé et observable. Chaque phase se termine par un critère de sortie vérifiable et une preuve à archiver dans le journal du Lab.

Tableau 12 — Séquencement du déploiement

Phase	Contenu	Durée	Critère de sortie
0	Socle : Kubernetes, ingress, certificats, GitOps, secrets	2 j	Toutes les applications synchronisées
1	Keycloak et KrakenD	2 j	Appel API rejeté sans jeton, accepté avec jeton
2	Données et événements : PostgreSQL, Redis, Kafka	1 j	Topic de test produit et consommé
3	Observabilité : OpenTelemetry, Prometheus, Grafana	2 j	Trace de test visible de bout en bout
4	Saleor et storefront Next.js	3 j	Commande créée avec un paiement manuel
5	OPPL Payment Layer	3 j	Paiement canonique créé et consultable
6	Kill Bill et plugin de paiement	3 j	Achat, capture et remboursement en PROCESSED
7	Application de paiement Saleor	2 j	Checkout complet sur le storefront
8	Connecteurs et simulateurs	4 j	Les quatre moyens de paiement fonctionnent
9	Vendure (second moteur) et labs	3 j	Même connecteur branché sur les deux moteurs

Convention

Les commandes ci-dessous sont fournies à titre de référence pédagogique. Les noms de charts, de variables et les versions évoluent : se référer à la documentation officielle de chaque projet et épingler les versions dans le dépôt GitOps.

6.1 Phase 0 — Socle

Phase 0 — amorçage

```
 # Cluster léger, trois nœuds
 curl -sfL https://get.k3s.io | sh -s - server --cluster-init --disable traefik

 # Ingress, certificats, GitOps
 helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx -n ingress --create-namespace
 helm upgrade --install cert-manager jetstack/cert-manager -n cert-manager --create-namespace --set
crds.enabled=true
 kubectl create namespace argocd
 kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/
install.yaml
 kubectl apply -f bootstrap/app-of-apps.yaml

 # Namespaces du Lab
 for ns in iam gateway store commerce pay billing connectors streaming data observ sim labs; do
   kubectl create namespace oppl-$ns
 done
```

6.2 Phase 1 — Identité et passerelle

Keycloak porte un realm `oppl` importé depuis Git. Il distingue quatre familles d'identités : les clients de la boutique, les comptes de service internes, les partenaires externes du sandbox et les opérateurs du Lab.

Phase 1 — IAM et passerelle

```
# Realm versionné puis importé au démarrage
kubectl -n oppl-iam create configmap oppl-realm --from-file=apps/oppl-iam/realm/oppl.json
helm upgrade --install keycloak <chart-keycloak> -n oppl-iam -f apps/oppl-iam/values.yaml

# Passerelle KrakenD : configuration déclarative, validée avant déploiement
krakend check -c apps/oppl-gateway/krakend.json -t
helm upgrade --install krakend <chart-krakend> -n oppl-gateway -f apps/oppl-gateway/values.yaml

# Vérification : sans jeton l'appel doit être rejeté par la passerelle
curl -s -o /dev/null -w '%{http_code}\n' -X POST https://api.oppl-lab.local/v1/payments   # 401
TOKEN=$(curl -s -d grant_type=client_credentials -d client_id=storefront -d client_secret=$SECRET \
  https://iam.oppl-lab.local/realms/oppl/protocol/openid-connect/token | jq -r .access_token)
curl -s -H "Authorization: Bearer $TOKEN" https://api.oppl-lab.local/v1/health
```



Figure 6 — KrakenD — validation de la configuration et rejet d'un appel sans jeton (capture illustrative)

6.3 Phases 2 et 3 — Données, événements, observabilité

Kafka est déployé par son opérateur ; topics et utilisateurs sont déclarés en ressources Kubernetes, donc placés sous contrôle GitOps. La chaîne d'observabilité comprend un collecteur OpenTelemetry qui masque les attributs sensibles avant export, Prometheus pour les métriques et Grafana pour les tableaux de bord.

Déclaration d'un topic sous contrôle GitOps

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: oppl.payments.captured.v1
  namespace: oppl-streaming
  labels: { strimzi.io/cluster: oppl-kafka }
spec:
  partitions: 12
  replicas: 3
  config:
    retention.ms: 1209600000        # 14 jours
    min.insync.replicas: 2
```

Collecteur OpenTelemetry — masquage systématique avant export

```
processors:
  attributes/mask:
    actions:
      - { key: card.pan,        action: delete }
      - { key: payment.alias,   action: hash }
      - { key: customer.msisdn, action: hash }
```

```
      - { key: customer.email,  action: hash }
service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [k8sattributes, attributes/mask, batch]
      exporters: [otlp/observability]
```

6.4 Phase 4 — Moteur de commerce et storefront

Phase 4 — Saleor et storefront

```
# Saleor : API, worker, tableau de bord et base
helm upgrade --install saleor <chart-saleor> -n oppl-commerce -f apps/oppl-commerce/saleor-values.yaml

# Canal de vente et devise du Lab
saleor-cli channel create web-ci --currency XOF --country CI

# Storefront : construction et déploiement
cd oppl-services/storefront && npm ci && npm run build
docker build -t registry.oppl-lab.local/storefront:1.0.0 . && docker push registry.oppl-lab.local/storefront:1.0.0
```

Point de vigilance : la devise

Le franc CFA (XOF) n'a pas de subdivision. Les moteurs de commerce et les bibliothèques de paiement manipulent le plus souvent des montants décimaux ou des unités mineures avec deux décimales implicites. Toute la chaîne doit être configurée et testée sur ce point : un facteur 100 mal placé est l'erreur la plus coûteuse et la plus fréquente des intégrations régionales. Le Lab impose un test dédié dès la phase 4.

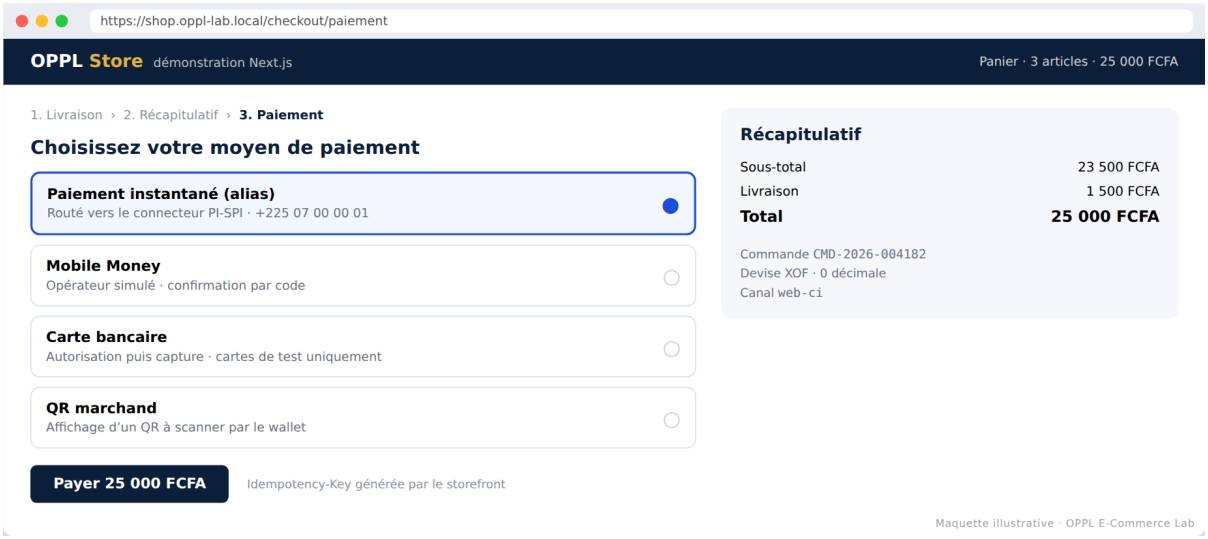


Figure 7 — Storefront — sélection du moyen de paiement au checkout (capture illustrative)

6.5 Phase 5 — Couche de paiement OPPL

Les cinq services de la couche de paiement sont déployés ensemble. Ils partagent le modèle canonique et la base de données des paiements, mais restent indépendants dans leur cycle de vie.

Phase 5 — couche de paiement

```
kubectl apply -k apps/oppl-pay/          # payment-api, payment-router,
                                         # tokenisation-svc, fraud-svc, recon-worker
# Vérification : création d'un paiement canonique de bout en bout
curl -s -X POST https://api.oppl-lab.local/v1/payments \
  -H "Authorization: Bearer $TOKEN" -H "Idempotency-Key: CMD-TEST-0001" \
  -H 'Content-Type: application/json' -d '{
    "orderId": "CMD-TEST-0001",
```

```
    "amount": { "value": 25000, "currency": "XOF" },
    "method": "INSTANT_ALIAS",
    "customer": { "reference": "cust-test-01" }
  }' | jq '{ paymentId, status, connector }'
```

6.6 Phase 6 — Kill Bill

Phase 6 — Kill Bill et plugin

```
helm upgrade --install killbill <chart-killbill> -n oppl-billing -f apps/oppl-billing/values.yaml

# Création du tenant du Lab
curl -X POST $KB/1.0/kb/tenants -u admin:password \
  -H 'Content-Type: application/json' -H 'X-Killbill-CreatedBy: lab' \
  -d '{ "apiKey": "oppl-lab", "apiSecret": "<secret>" }'

# Installation du plugin de paiement du Lab
kpm install_java_plugin oppl-connector --from-source-file=killbill-oppl-connector-1.0.0.jar \
   --destination=/var/lib/killbill/bundles
```

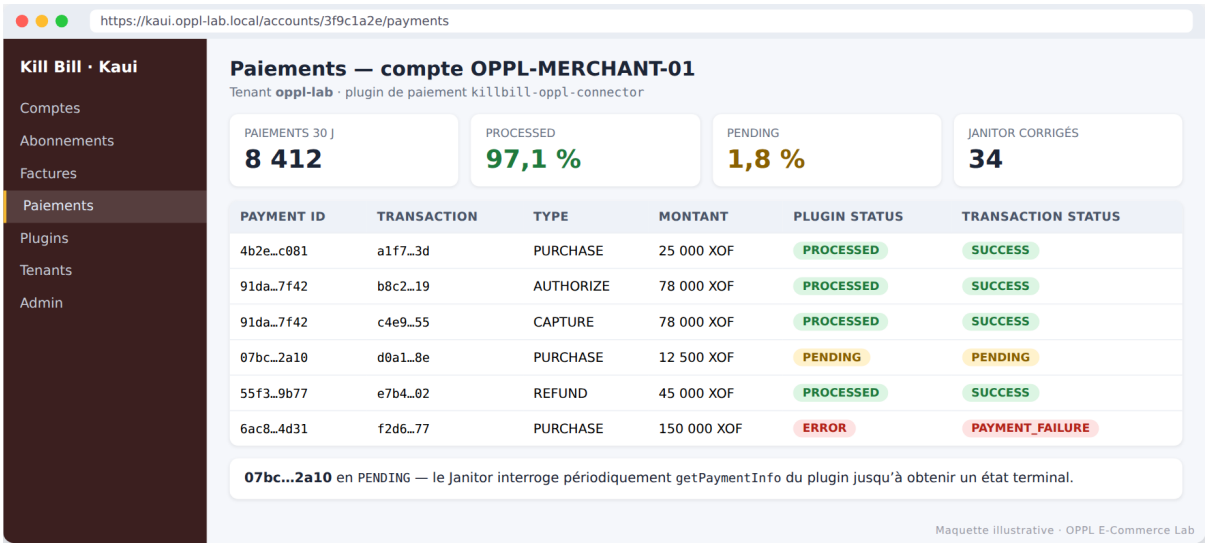


Figure 8 — Kill Bill (Kaui) — paiements du compte marchand et statuts de plugin (capture illustrative)

6.7 Phase 7 — Application de paiement du moteur de commerce

L'application de paiement est un service externe qui déclare son manifeste au moteur de commerce, s'abonne aux événements de session de transaction et répond de façon synchrone. Elle ne contient **aucune logique de paiement** : elle traduit et délègue.

Phase 7 — installation de l'application de paiement

```
# Installation de l'application dans Saleor (manifeste exposé par l'application)
saleor-cli app install --manifest https://payapp.oppl-lab.local/api/manifest

# Permissions demandées : HANDLE_PAYMENTS, MANAGE_ORDERS
# Webhooks déclarés : TRANSACTION_INITIALIZE_SESSION, TRANSACTION_PROCESS_SESSION,
#                     TRANSACTION_REFUND_REQUESTED, TRANSACTION_CANCELATION_REQUESTED,
#                     TRANSACTION_CHARGE_REQUESTED
```

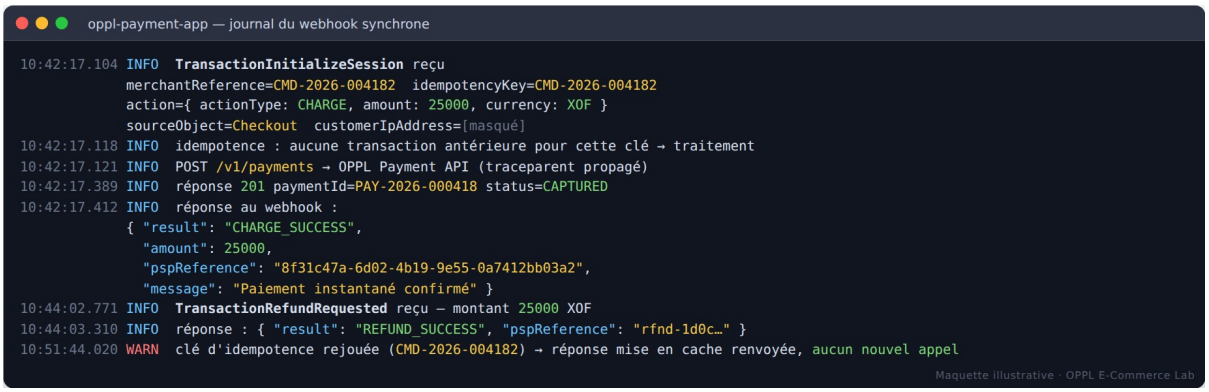


Figure 9 — Application de paiement — journal du webhook synchrone et rejeu idempotent (capture illustrative)

6.8 Phase 8 — Connecteurs et simulateurs

Chaque connecteur est déployé avec le simulateur de la contrepartie qu'il adresse. Le simulateur est paramétrable : taux de succès, latence, codes d'erreur, délai de webhook, comportement en double envoi. C'est lui qui rend les labs d'échec possibles.

Tableau 13 — Connecteurs et simulateurs

Connecteur	Simulateur associé	Comportements simulables
pi-spi-connector	Opérateur central ISO 20022	ACSC, RJCT avec code motif, retard, absence de réponse
momo-connector	Opérateur Mobile Money	Confirmation par code, webhook différé, double webhook, timeout
card-connector	PSP carte	Autorisation, refus, capture, remboursement, impayé
qr-connector	Wallet scanneur	Scan, expiration du QR, annulation par le client
psp-connector	Agrégateur	Bascule entre prestataires, indisponibilité, latence variable
bank-connector	Banque	Virement, fichier de règlement, rejet différé

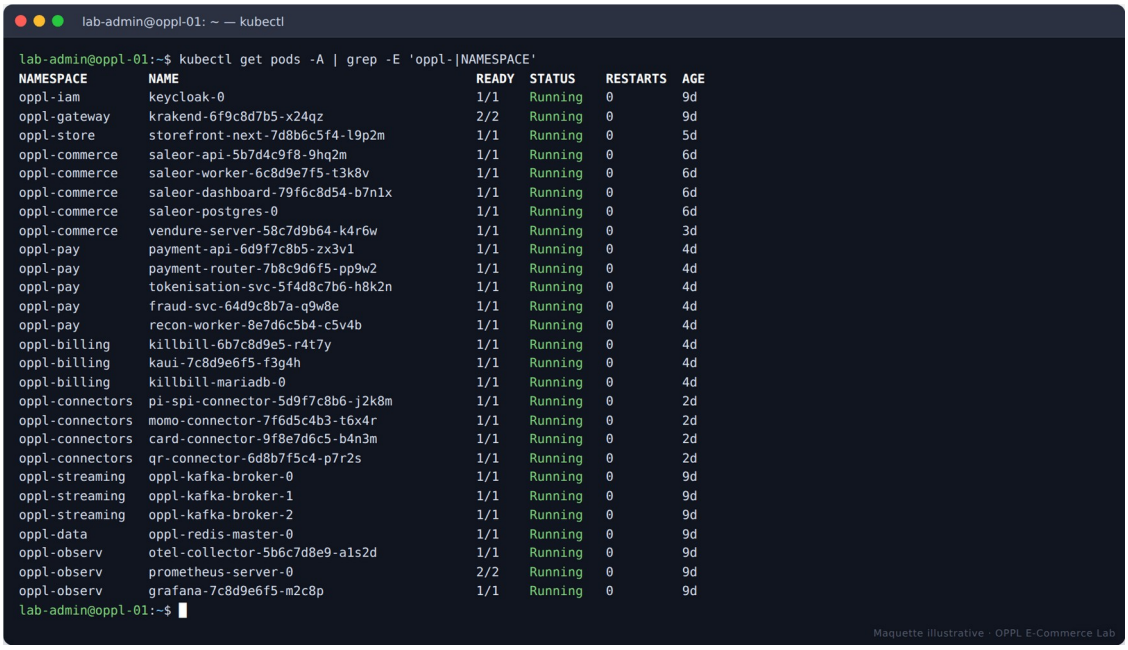


Figure 10 — État des pods après les dix phases (capture illustrative)

6.9 Vérification globale

- Une commande de bout en bout aboutit en moins de deux secondes sur le moyen instantané.
- La trace correspondante couvre au moins neuf services, du storefront au connecteur.
- Le rejeu d'une même requête de paiement ne crée aucune seconde transaction.
- Un remboursement total puis un remboursement partiel s'exécutent et se retrouvent dans le rapprochement.
- Les quatre moyens de paiement du storefront fonctionnent, y compris leurs chemins d'erreur.
- Les tableaux de bord Grafana affichent des volumes cohérents avec les enregistrements de Kill Bill.

7. La couche de paiement OPPL

La couche de paiement est le cœur pédagogique et technique du Lab. C'est le seul composant entièrement écrit par l'équipe : tout le reste est assemblé. Elle expose une API unique, applique les contrôles, choisit un chemin d'exécution et garantit que chaque paiement atteint un état terminal explicable.

7.1 Le modèle canonique

Chaque système de la chaîne parle sa propre langue : GraphQL pour le moteur de commerce, l'API de Kill Bill pour la facturation, ISO 20022 pour l'instantané, ISO 8583 pour la carte, des API propriétaires pour les opérateurs. Un modèle canonique interne évite la multiplication des conversions point à point : chaque adaptateur convertit une seule fois.

contracts — le paiement canonique

```
export type Rail = 'INSTANT_ALIAS' | 'MOBILE_MONEY' | 'CARD' | 'QR' | 'BANK_TRANSFER';

export interface Money { minorUnits: number; currency: string; }   // XOF : 0 décimale

export interface CanonicalPayment {
  paymentId: string;            // PAY-2026-000418 — identifiant métier unique
  orderId: string;              // référence de la commande côté commerce
  idempotencyKey: string;       // fournie par le storefront, propagée jusqu'au connecteur
  rail: Rail;
  amount: Money;
  customer: { reference: string; aliasHash?: string; kycLevel: 'L0'|'L1'|'L2' };
  merchant: { reference: string; vendors?: VendorSplit[] };   // marketplace
  status: PaymentStatus;
  connector?: string;
  networkReference?: string;    // référence renvoyée par le réseau
  createdAt: string; updatedAt: string;
  metadata: Record<string, string>;
}
```

Pourquoi des unités mineures entières

Le montant est toujours un entier d'unités mineures accompagné de sa devise. En XOF, l'unité mineure est le franc lui-même : 25 000 FCFA s'écrit `{ minorUnits: 25000, currency: 'XOF' }`. Aucun nombre à virgule flottante n'est autorisé dans le code monétaire du Lab — c'est une règle de revue de code, pas une recommandation.

7.2 L'API de paiement

Tableau 14 — Surface de l'API de paiement

Endpoint	Méthode	Rôle	Idempotent
/v1/payments	POST	Créer un paiement (autorisation ou achat direct)	Oui, par en-tête `Idempotency-Key`
/v1/payments/{id}	GET	Consulter l'état courant	Sans objet
/v1/payments/{id}/capture	POST	Capturer une autorisation	Oui
/v1/payments/{id}/cancel	POST	Annuler avant capture	Oui
/v1/payments/{id}/refunds	POST	Rembourser tout ou partie	Oui
/v1/payments/{id}/events	GET	Historique complet des événements	Sans objet
/v1/webhooks/{connector}	POST	Recevoir les confirmations asynchrones	Oui, par référence réseau

Création d'un paiement

```
POST /v1/payments
Authorization: Bearer <jeton>
Idempotency-Key: CMD-2026-004182
Content-Type: application/json

{ "orderId": "CMD-2026-004182",
  "amount": { "value": 25000, "currency": "XOF" },
  "method": "INSTANT_ALIAS",
  "customer": { "reference": "cust-0042", "alias": "+2250700000001" },
  "capture": true }

201 Created
{ "paymentId": "PAY-2026-000418", "status": "CAPTURED",
  "connector": "pi-spi-connector",
  "networkReference": "OPPL-20260913-000418",
  "createdAt": "2026-09-13T10:42:17.121Z" }
```

7.3 L'idempotence, en pratique

L'idempotence n'est pas une case à cocher : c'est une mécanique à trois niveaux, que chaque apprenant doit savoir décrire et tester.

Tableau 15 — Les trois niveaux d'idempotence

Niveau	Mécanisme	Ce qu'il protège
Entrée	En-tête `Idempotency-Key` + empreinte du corps, stockés 24 h	Double clic, retentative du navigateur, rejeu réseau
Traitement	Contrainte d'unicité sur (orderId, rail, montant) en base	Deux requêtes concurrentes qui passent le premier filtre
Sortie	Référence réseau unique enregistrée avant appel, vérifiée au retour	Double envoi vers le réseau, webhook reçu en double

payment-api — traitement idempotent d'une création de paiement

```
async function createPayment(req: PaymentRequest, key: string) {
  const fingerprint = sha256(canonicalJson(req));
  const existing = await idempotency.get(key);
  if (existing) {
    if (existing.fingerprint !== fingerprint) {
      throw new HttpError(422, 'IDEMPOTENCY_KEY_REUSED_WITH_DIFFERENT_BODY');
    }
    return existing.response;                // même réponse, aucun effet de bord
  }
  const lock = await locks.acquire(`pay:${key}`, { ttl: 30_000 });
  try {
    const payment = await process(req);      // contrôles, routage, exécution
    await idempotency.put(key, { fingerprint, response: payment, ttl: 86_400 });
    return payment;
  } finally { await lock.release(); }
}
```

Le cas d'école est le rejeu avec un corps différent : la même clé accompagnée d'un montant modifié doit produire une erreur explicite, jamais un second paiement ni une acceptation silencieuse.

7.4 Le routage

Le routeur choisit un connecteur en évaluant des règles ordonnées. La première règle satisfaite l'emporte ; un connecteur de repli est déclenché sur échec **technique** uniquement — jamais sur un refus métier, sous peine de créer des doubles paiements.

payment-router — règles de routage déclaratives

```
rules:
```

```
- id: instant-ci
  when: { method: INSTANT_ALIAS, country: CI }
  route: pi-spi-connector

- id: momo-primary
  when: { method: MOBILE_MONEY }
  route: momo-connector-A
  fallback: momo-connector-B          # uniquement sur 5xx, timeout ou circuit ouvert

- id: card-high-value
  when: { method: CARD, amountMin: 100000 }
  route: psp-card-A
  fallback: psp-card-B

- id: card-default
  when: { method: CARD }
  split: { psp-card-A: 70, psp-card-B: 30 }

- id: default
  route: psp-card-A
```

La distinction qui évite les doubles débits

Un **refus métier** (fonds insuffisants, alias inconnu, carte bloquée) est un résultat définitif : il ne doit jamais déclencher de repli. Un **échec technique** (timeout, 5xx, connexion coupée) est une absence de résultat : le repli n'est autorisé que si l'on a la certitude que la première tentative n'a pas abouti — ce que seule une référence réseau préalablement enregistrée permet d'établir.

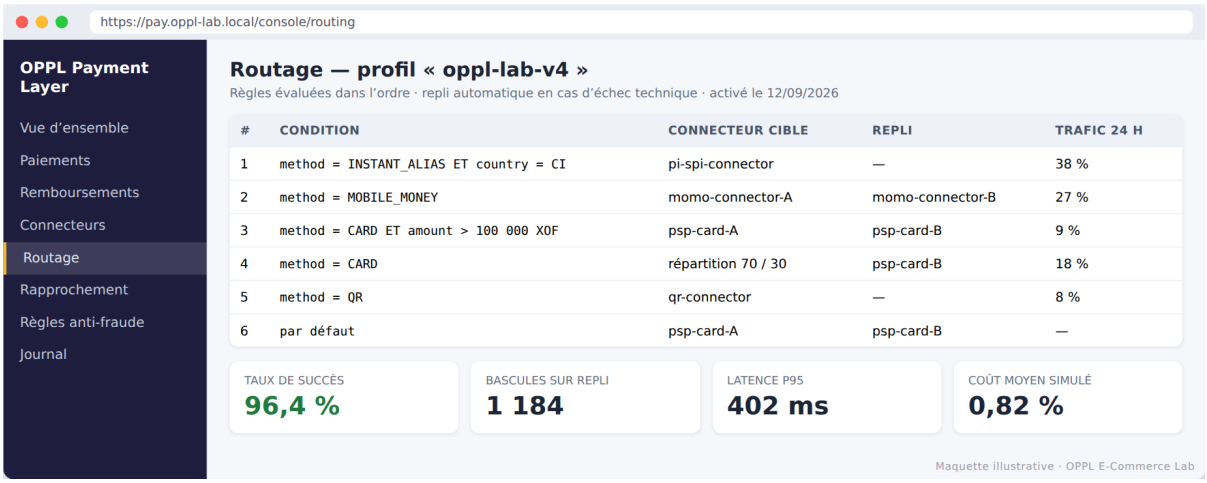


Figure 11 — Console OPPL — règles de routage et repli (capture illustrative)

7.5 La tokenisation

Le service de tokenisation remplace les données d'instrument par un jeton opaque dès le point d'entrée. Il poursuit trois objectifs : réduire la surface d'exposition, permettre le paiement en un clic et les abonnements, et surtout **réduire le périmètre de conformité** de tous les autres composants.

- Le storefront ne transmet jamais de numéro de carte au moteur de commerce : il obtient un jeton et transmet le jeton.
- Le jeton est propre au marchand et au Lab ; il n'a aucune valeur hors de l'environnement.
- Les journaux, traces et événements ne contiennent que l'empreinte du jeton et les six premiers ou quatre derniers chiffres masqués.
- Pour les alias de paiement, le principe est identique : l'alias circule sous forme hachée, sauf dans le connecteur qui en a strictement besoin.

7.6 La détection de fraude

Le service de fraude expose une décision synchrone, appelée avant l'exécution. Il additionne le poids des règles déclenchées et compare à deux seuils : alerte et blocage. Les règles sont des données versionnées, distinctes du code, ce qui permet à la conformité de proposer une évolution par demande de modification revue et tracée.

fraud-svc — règles et seuils (valeurs illustratives à calibrer)

```
rules:
  - { id: R-01, name: velocity,   when: "attempts(customer, 10m) > 5",        weight: 35 }
  - { id: R-02, name: amount,     when: "amount > 8 * avg(customer, 90d)",    weight: 25 }
  - { id: R-03, name: newAccount, when: "age(customer) < 24h",                weight: 20 }
  - { id: R-04, name: geoMismatch,when: "cardCountry != shippingCountry",     weight: 30 }
  - { id: R-05, name: watchlist,  when: "inWatchlist(customer.aliasHash)",    weight: 90 }
thresholds: { alert: 60, block: 85 }
```



Figure 12 — Console OPPL — règles anti-fraude et statistiques de déclenchement (capture illustrative)

> **Le vrai enseignement du Fraud Lab**
>
> Le taux de détection n'est jamais le bon indicateur seul. Un jeu de règles qui bloque 100 % de la fraude en bloquant 15 % des paiements légitimes détruit plus de valeur qu'il n'en protège. Les apprenants doivent apprendre à raisonner sur le couple détection / faux positifs, et à mesurer la latence ajoutée au parcours client.

7.7 Le rapprochement

Le rapprochement est l'exercice le plus sous-estimé et le plus formateur. Il consiste à vérifier la cohérence de cinq niveaux : la commande, la transaction du moteur de commerce, le paiement enregistré, le règlement annoncé par le réseau et l'écriture comptable.

Tableau 16 — Typologie des écarts de rapprochement

Type d'écart	Cause fréquente	Traitement dans le Lab
Paiement sans commande	Webhook arrivé après expiration du panier	Compte d'attente, puis remboursement ou rattachement manuel
Commande sans paiement	Abandon au checkout, échec silencieux	Annulation automatique après délai, alerte si récurrent
Montant divergent	Frais opérateur non modélisés, arrondi, conversion	Analyse ; correction du modèle de frais
Statut divergent	Confirmation tardive, webhook perdu	Interrogation planifiée du réseau, réconciliation

Type d'écart	Cause fréquente	Traitement dans le Lab
Règlement absent	Décalage J+1 ou J+2 selon le réseau	Attendu ; surveillé par une règle d'âge
Doublon	Repli déclenché à tort sur un échec technique	Incident majeur : analyse de cause, remboursement



Figure 13 — Console OPPL — rapprochement quotidien et typologie des écarts (capture illustrative)

7.8 Les événements

Chaque changement d'état significatif est publié sur Kafka au format CloudEvents, avec le `paymentId` en clé de partition afin de garantir l'ordre des événements d'un même paiement.

Tableau 17 — Topics du Lab

Topic	Producteur	Consommateurs
oppl.payments.initiated.v1	Payment API	Analytique, audit, anti-fraude
oppl.payments.pending.v1	Connecteurs	Suivi des confirmations tardives
oppl.payments.captured.v1	Payment API	Commerce, comptabilité, notifications
oppl.payments.failed.v1	Payment API	Support, analytique, alerting
oppl.refunds.v1	Payment API	Comptabilité, service client
oppl.settlements.v1	Connecteurs	Rapprochement, règlement vendeurs
oppl.fraud.alerts.v1	Fraud service	Gestion des cas, conformité
oppl.audit.trail.v1	Tous	Archivage et traçabilité

8. Intégrer le moteur de commerce

8.1 Le principe : le commerce demande, le paiement décide

L'erreur classique consiste à implanter la logique de paiement dans le moteur de commerce : appels directs aux prestataires, statuts propres, règles de routage dans un plugin. Le système devient alors indissociable de son moteur de commerce, et tout changement de plateforme impose de réécrire l'intégralité de la chaîne de paiement.

Le Lab impose le découplage inverse : le moteur de commerce ne connaît qu'une seule « passerelle », l'application de paiement OPPL. Celle-ci ne contient aucune logique métier — elle traduit et délègue. Toute l'intelligence réside dans la couche de paiement.

8.2 L'application de paiement Saleor

Saleor sollicite l'application par des **webhooks synchrones** : la plateforme attend la réponse pour poursuivre. Trois contraintes en découlent, et elles structurent toute l'implémentation.

- **Le temps de réponse est borné.** La plateforme impose une limite de l'ordre de vingt secondes ; le Lab se fixe un budget interne bien plus strict — deux secondes — au-delà duquel l'application renvoie un statut d'action requise plutôt qu'un échec.
- **La réponse doit être un statut, pas une promesse.** Les valeurs possibles couvrent le succès et l'échec d'autorisation ou d'encaissement, ainsi que l'action requise lorsque le client doit encore intervenir.
- **La clé d'idempotence est fournie par la plateforme.** Elle doit être propagée telle quelle jusqu'au connecteur : c'est elle qui garantit l'unicité de bout en bout.

payment-app-saleor — traitement du webhook d'initialisation de transaction

```
export default transactionInitializeSessionWebhook.createHandler(async (req, res, ctx) => {
  const { payload } = ctx;
  const canonical = toCanonical(payload);            // traduction, aucune logique métier

  try {
    const payment = await opplApi.createPayment(canonical, {
      idempotencyKey: payload.idempotencyKey,        // fournie par la plateforme
      timeoutMs: 2_000,
    });

    return res.status(200).json({
      result: mapResult(payment.status),             // CHARGE_SUCCESS, CHARGE_ACTION_REQUIRED…
      amount: payload.action.amount,
      pspReference: payment.networkReference ?? payment.paymentId,
      message: payment.statusMessage,
      data: payment.clientAction ?? {},              // transmis au storefront si besoin
    });
  } catch (e) {
    if (e instanceof TimeoutError) {
      // Jamais d'échec définitif sur un timeout : on rend la main et on réconcilie
      return res.status(200).json({
        result: 'CHARGE_ACTION_REQUIRED',
        amount: payload.action.amount,
        pspReference: canonical.paymentId,
        message: 'Confirmation en cours',
      });
    }
    return res.status(200).json({ result: 'CHARGE_FAILURE', amount: payload.action.amount,
                                  pspReference: canonical.paymentId, message:
toPublicMessage(e) });
  }
});
```

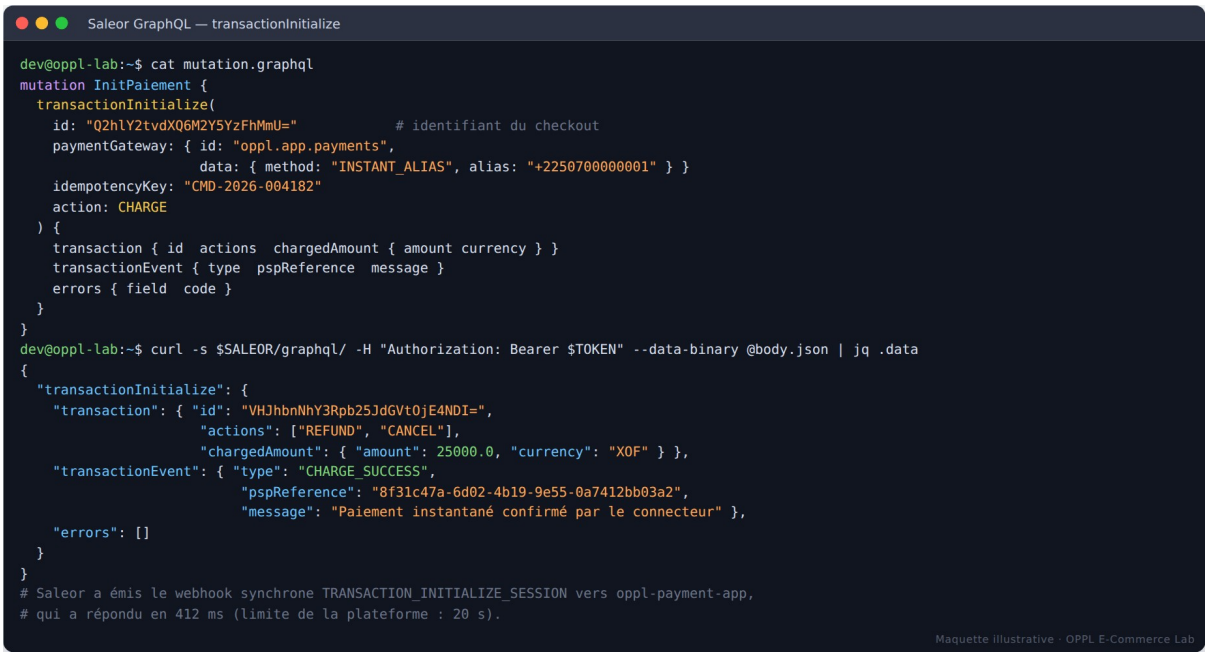


Figure 14 — Appel GraphQL d'initialisation de transaction et réponse de l'application (capture illustrative)

8.3 Les autres événements à traiter

Tableau 18 — Événements traités par l'application de paiement

Événement	Déclencheur	Action de l'application
Initialisation de session	Le client valide son paiement au checkout	Créer le paiement canonique et renvoyer un statut
Traitement de session	Le client a terminé une action (code, 3-D Secure, scan)	Interroger l'état réel et renvoyer le statut consolidé
Demande d'encaissement	Un agent capture une autorisation depuis le back-office	Appeler la capture sur l'API de paiement
Demande d'annulation	Un agent annule avant capture	Appeler l'annulation
Demande de remboursement	Un agent rembourse tout ou partie	Appeler le remboursement, gérer le partiel

8.4 Le plugin Vendure

Vendure appelle son gestionnaire de moyen de paiement à l'intérieur de la transaction de commande. Le plugin doit donc être rapide et ne jamais bloquer : les confirmations tardives remontent par un second chemin, déclenché par le webhook du connecteur.

vendure-plugin-oppl — gestionnaire de moyen de paiement

```
export const opplInstantHandler = new PaymentMethodHandler({
  code: 'oppl-instant',
  description: [{ languageCode: LanguageCode.fr, value: 'Paiement instantané (alias)' }],
  args: { connector: { type: 'string', defaultValue: 'pi-spi-connector' } },

  createPayment: async (ctx, order, amount, args, metadata) => {
    const payment = await opplApi.createPayment({
      orderId: order.code,
      amount: { value: amount, currency: order.currencyCode },   // unités mineures
      rail: 'INSTANT_ALIAS',
      customer: { reference: order.customer.id, alias: metadata.alias },
    }, { idempotencyKey: order.code });

    return {
```

```
      amount,
      state: payment.status === 'CAPTURED' ? 'Settled' : 'Authorized',
      transactionId: payment.paymentId,
      metadata: { networkReference: payment.networkReference },
    };
  },

  settlePayment: async () => ({ success: true }),   // appelé par le webhook du connecteur
});
```

L'exercice décisif

Le même connecteur doit servir les deux moteurs sans une ligne de code spécifique. Si l'implémentation Vendure exige d'adapter le connecteur, c'est le signe que de la logique de paiement a fui dans la couche commerce. C'est exactement ce que le lab de bascule est conçu pour révéler.

9. Kill Bill : facturation et exécution des paiements

9.1 Pourquoi un moteur dédié

On pourrait croire la couche de paiement OPPL suffisante. Elle ne l'est pas dès que l'on sort du paiement unitaire immédiat : facturation périodique, abonnements, impayés, relances, avoirs, factures consolidées, plans tarifaires. Kill Bill apporte ce socle, avec un modèle de paiement robuste et une abstraction de prestataire par plugins.

Tableau 19 — Apports de Kill Bill

Capacité	Apport pour le Lab
Paiements unitaires	Cycle complet autorisation, capture, annulation, remboursement, avoir
Abonnements et facturation	Périodicité, proratisation, changements de plan, essais
Relances et impayés	Politique de nouvelles tentatives configurable et observable
Abstraction de prestataire	Un plugin par prestataire ; le cœur ignore les spécificités
Réconciliation intégrée	Mécanisme interne de correction des états non terminaux
Impayés et rejets	Modélisation des rejets postérieurs au paiement

9.2 Le plugin de paiement OPPL

Le plugin implémente l'interface de plugin de paiement de Kill Bill. Chaque méthode traduit une opération métier vers l'API de la couche de paiement OPPL et renvoie un statut d'exécution.

Tableau 20 — Correspondance entre le plugin et l'API OPPL

Méthode du plugin	Opération	Appel OPPL correspondant
authorizePayment	Réserver des fonds (carte)	POST /v1/payments avec capture différée
capturePayment	Encaisser une autorisation	POST /v1/payments/{id}/capture
purchasePayment	Autorisation + capture en une fois	POST /v1/payments avec capture immédiate
voidPayment	Annuler avant capture	POST /v1/payments/{id}/cancel
refundPayment	Rembourser tout ou partie	POST /v1/payments/{id}/refunds
creditPayment	Créditer sans paiement préalable	POST /v1/payments type CREDIT
getPaymentInfo	Consulter l'état réel	GET /v1/payments/{id} — clé de la réconciliation
addPaymentMethod	Enregistrer un instrument	Création d'un jeton de tokenisation

killbill-oppl-connector — implémentation de l'achat

```
public PaymentTransactionInfoPlugin purchasePayment(
        UUID kbAccountId, UUID kbPaymentId, UUID kbTransactionId, UUID kbPaymentMethodId,
        BigDecimal amount, Currency currency,
        Iterable<PluginProperty> properties, CallContext context) {

    CanonicalPayment req = CanonicalMapper.from(kbPaymentId, amount, currency, properties);
    try {
        // La clé d'idempotence est l'identifiant de transaction Kill Bill : stable sur rejeu
        OpplPayment p = oppl.createPayment(req, kbTransactionId.toString());
        return build(p, PaymentPluginStatus.PROCESSED);

    } catch (BusinessDeclineException e) {          // refus métier : résultat définitif
```

```
        return build(e, PaymentPluginStatus.ERROR);

    } catch (TimeoutException | IOException e) {     // absence de réponse : surtout pas ERROR
        return build(req, PaymentPluginStatus.PENDING);
    }
}
```

Le piège du statut « indéfini »

Le modèle de Kill Bill distingue quatre issues : traité, en attente, erreur et indéfini. La dernière signifie littéralement « le paiement a peut-être eu lieu, peut-être pas », et c'est la seule que le moteur ne peut pas rejouer automatiquement. Un plugin bien écrit ne la renvoie presque jamais : il préfère le statut d'attente, qui permet au mécanisme de réconciliation interne d'interroger périodiquement l'état réel jusqu'à obtenir une issue définitive. Comprendre cette nuance est l'un des apprentissages les plus transférables du Lab.

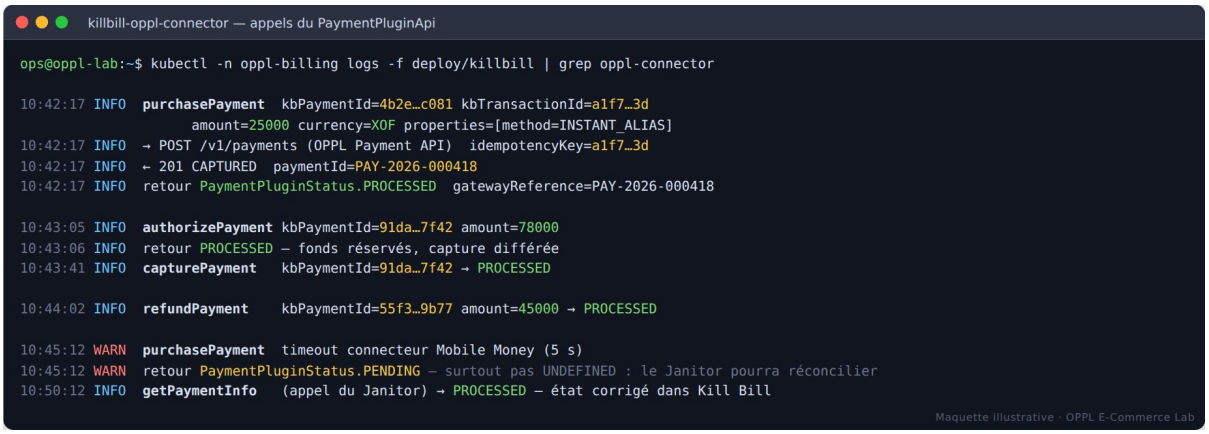


Figure 15 — Journal du plugin — achat, capture, remboursement et rattrapage d'un paiement en attente (capture illustrative)

9.3 Abonnements et relances

Le lab d'abonnement met en évidence une différence majeure avec le paiement de commerce : le client n'est pas devant son écran. Aucune action interactive n'est possible, la stratégie de relance devient le seul levier, et le taux d'échec au premier prélèvement est structurellement élevé.

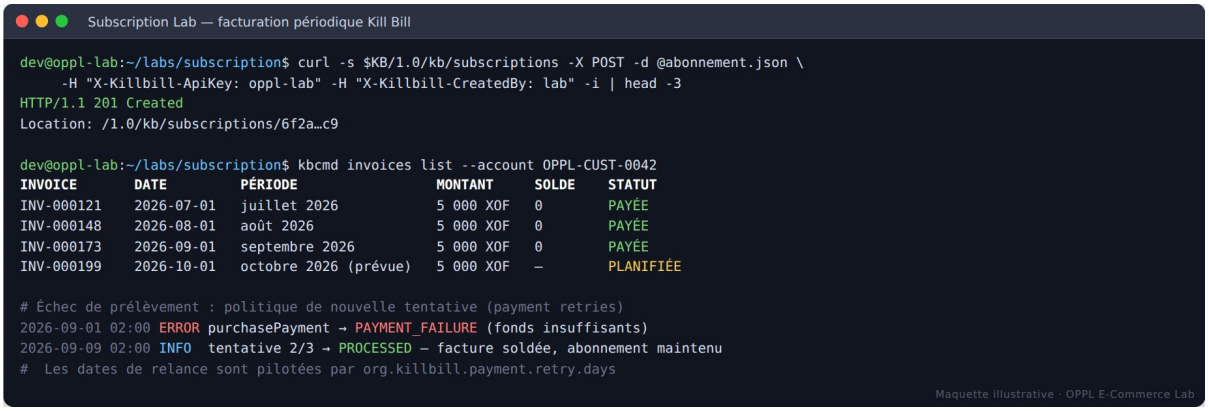


Figure 16 — Facturation périodique et politique de relance (capture illustrative)

9.4 Marketplace : répartition et règlement

La marketplace introduit la répartition d'un paiement entre plusieurs bénéficiaires et le règlement différé des vendeurs. Le Lab modélise la commission de la plateforme, le net vendeur, le calendrier de règlement et les blocages — vendeur non vérifié, litige en cours, seuil minimal non atteint.



Figure 17 — Répartition d'une commande multi-vendeurs et règlement (capture illustrative)

Limite à énoncer clairement

L'encaissement pour compte de tiers est une activité réglementée. Le Lab modélise les mécanismes à des fins pédagogiques ; il ne préjuge en rien du statut requis pour les exercer réellement, ni de l'agrément nécessaire. Cette distinction doit être rappelée dans toute démonstration à un partenaire.

10. Les connecteurs de paiement

10.1 Un cadre commun

Un connecteur est un composant délibérément mince. Toute la complexité métier reste dans la couche de paiement ; le connecteur ne fait que traduire et transporter. Cette discipline est ce qui permet d'en industrialiser la production.

connectors — interface commune

```
export interface PaymentConnector {
  readonly name: string;
  readonly supportedRails: Rail[];

  /** Exécute le paiement. Doit être idempotent sur idempotencyKey. */
  execute(p: CanonicalPayment, ctx: ConnectorContext): Promise<ConnectorResult>;

  /** Interroge l'état réel auprès du réseau. Base de toute réconciliation. */
  inquire(networkReference: string): Promise<ConnectorResult>;

  /** Rembourse tout ou partie. */
  refund(p: CanonicalPayment, amount: Money, ctx: ConnectorContext): Promise<ConnectorResult>;

  /** Traite une notification entrante ; doit tolérer les doublons. */
  handleWebhook(raw: RawWebhook): Promise<ConnectorResult | null>;
}

export interface ConnectorResult {
  status: 'CAPTURED' | 'AUTHORIZED' | 'PENDING' | 'ACTION_REQUIRED' | 'FAILED' | 'UNKNOWN';
  networkReference?: string;
  reasonCode?: string;      // code du réseau, conservé tel quel
  reason?: string;          // message lisible, jamais destiné au client final tel quel
  raw?: unknown;            // trace brute, stockée chiffrée pour investigation
}
```

Tableau 21 — Exigences applicables à tout connecteur du Lab

Exigence	Vérifiée par
Idempotence sur la clé fournie	Test automatisé : double appel, un seul effet
Méthode d'interrogation toujours implémentée	Test : après un timeout simulé, l'état réel est retrouvé
Aucune donnée sensible dans les journaux	Analyse statique + revue de code
Timeout explicite sur chaque appel sortant	Revue de configuration
Tolérance aux notifications en double	Test : webhook envoyé deux fois, un seul effet
Codes réseau conservés sans réinterprétation	Revue ; le mapping se fait dans la couche de paiement

10.2 Connecteur paiement instantané (cible PI-SPI)

Ce connecteur construit et interprète des messages ISO 20022 : demande de virement, statut, notification. Il constitue la préparation la plus directe à une intégration réelle à une plateforme de paiement instantané.

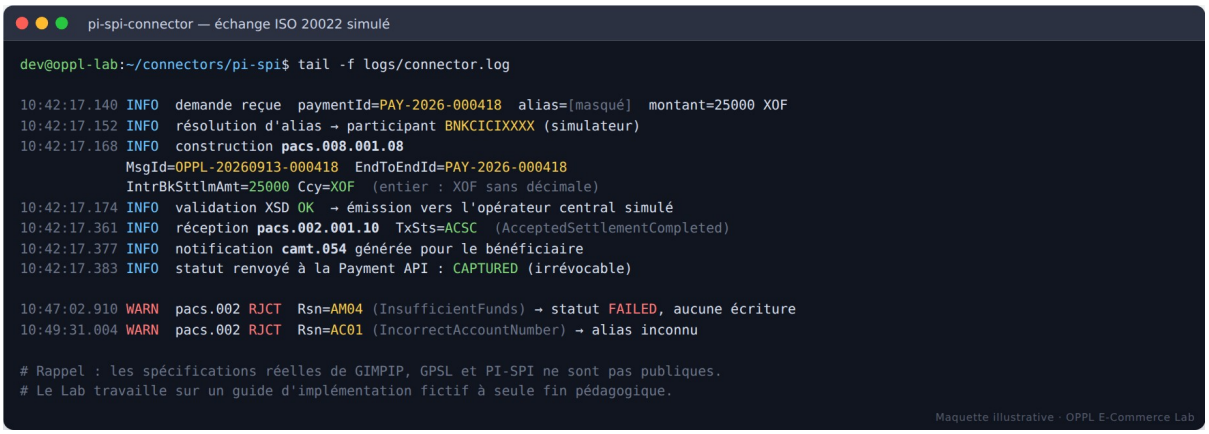


Figure 18 — Connecteur instantané — échange ISO 20022 simulé et rejets motivés (capture illustrative)

Tableau 22 — Messages manipulés par le connecteur instantané

Message	Rôle	Points d'attention
Demande de virement (pacs.008)	Instruction de paiement	Identifiant de bout en bout stable, montant entier en XOF
Statut (pacs.002)	Acceptation ou rejet motivé	Le code motif doit être conservé et journalisé tel quel
Notification (camt.054)	Information du bénéficiaire	Distinguer notification et preuve de règlement
Demande d'annulation (camt.056)	Rappel de fonds	Souvent non garanti : un instantané est en principe irrévocable

Cadre de la simulation

Les spécifications de GIMPIP, GPSL et PI-SPI ne sont pas publiques. Le Lab travaille sur un guide d'implémentation fictif, construit à partir des standards ISO 20022 génériques, dans un but exclusivement pédagogique. Toute intégration réelle relève des processus d'homologation définis par les opérateurs de ces systèmes et suppose des accords préalables.

10.3 Connecteur Mobile Money

Ce connecteur illustre le cas le plus courant en Afrique de l'Ouest et le plus riche en enseignements : la confirmation est asynchrone et arrive par notification, parfois en double, parfois jamais.

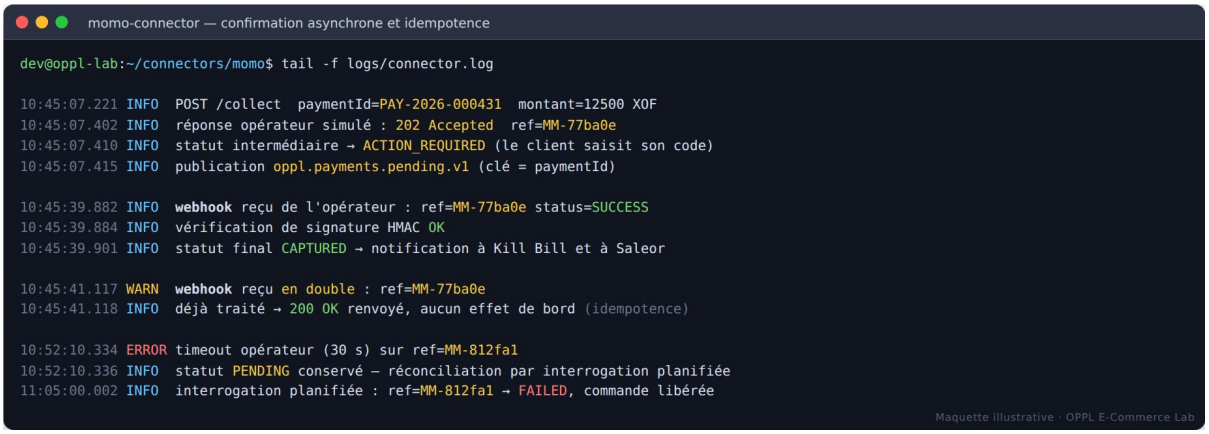


Figure 19 — Connecteur Mobile Money — confirmation asynchrone, doublon et reprise (capture illustrative)

1. Le connecteur appelle l'opérateur, qui accepte la demande sans la confirmer.
2. Le paiement passe en état d'action requise : le client saisit son code sur son téléphone.

3. L'opérateur envoie une notification signée ; le connecteur en vérifie la signature avant tout traitement.
4. Si la même notification arrive deux fois, la seconde est reconnue et ignorée sans effet de bord.
5. Si aucune notification n'arrive, une interrogation planifiée détermine l'état réel et débloque la commande.

10.4 Connecteur carte

Le connecteur carte reproduit le cycle monétique classique : autorisation, capture, annulation, remboursement, puis impayé éventuel. Il est aussi celui qui porte le plus d'exigences de sécurité, puisqu'il est le seul à manipuler — brièvement — des données d'instrument.

Tableau 23 — Cycle de vie d'un paiement par carte

Opération	Effet	Piège fréquent
Autorisation	Réserve les fonds pour une durée limitée	L'autorisation expire : capturer trop tard échoue
Capture	Encaisse tout ou partie de l'autorisation	Capture partielle mal gérée en comptabilité
Annulation	Libère l'autorisation	Confondue avec un remboursement dans les rapports
Remboursement	Recrédite après capture	Le remboursement partiel multiple doit être plafonné
Impayé	Contestation postérieure au paiement	Souvent oublié du modèle de données

10.5 Connecteurs QR, PSP et banque

- **QR marchand** : génération d'un code à durée de vie limitée, lecture par un wallet, confirmation asynchrone. L'exercice porte sur l'expiration et sur l'annulation par le client après scan.
- **PSP / agrégateur** : plusieurs prestataires simulés avec des taux de succès, des latences et des coûts différents, pour donner de la matière au lab de routage.
- **Banque** : virement et fichier de règlement, avec rejet différé et rapprochement par fichier — le cas le plus proche des flux de masse.

10.6 Industrialiser : l'usine à connecteurs

La valeur durable du Lab ne réside pas dans les connecteurs eux-mêmes mais dans la **méthode** qui permet d'en produire un nouveau de façon prévisible. Le Lab formalise cette méthode en sept étapes, chacune produisant un livrable vérifiable.

1. **Cadrage** : obtenir le guide d'implémentation du réseau, identifier les messages, les statuts, les codes d'erreur et les limites.
2. **Génération du squelette** : partir du modèle de connecteur du Lab, qui fournit déjà l'interface, les tests d'idempotence et l'instrumentation.
3. **Simulateur de contrepartie** : écrire d'abord le simulateur, avant le connecteur. C'est lui qui rend les tests possibles.
4. **Traduction** : implémenter l'aller et le retour, en conservant les codes du réseau sans les réinterpréter.
5. **Chemins d'erreur** : couvrir timeout, doublon, rejet, réponse tardive, réponse incohérente.
6. **Certification interne** : exécuter la batterie de tests du Lab et produire le rapport de conformité au cadre commun.

7. **Documentation et transfert** : fiche de connecteur, matrice de statuts, limites connues, guide d'exploitation.

Tableau 24 — Livrables de l'usine à connecteurs

Étape	Livrable	Critère d'acceptation
Cadrage	Fiche de cadrage du réseau	Tous les statuts et codes d'erreur recensés
Squelette	Projet initialisé, pipeline vert	Tests d'interface passants
Simulateur	Contrepartie paramétrable	Permet de jouer tous les scénarios d'erreur
Traduction	Connecteur fonctionnel	Flux nominal complet
Chemins d'erreur	Suite de tests négatifs	100 % des scénarios couverts
Certification	Rapport de conformité	Les six exigences du cadre commun sont vérifiées
Documentation	Fiche et guide d'exploitation	Un tiers peut exploiter le connecteur sans son auteur

11. Les labs de formation

Le parcours compte treize labs. Chacun suit le même cycle — apprendre, construire, casser, observer, corriger, industrialiser — et se conclut par un livrable évaluable. Les six premiers constituent le socle obligatoire ; les suivants approfondissent des domaines spécialisés.



Figure 20 — Suivi de progression d'une promotion sur les treize labs (capture illustrative)

11.1 Checkout Lab

Élément	Description
Objectif	Comprendre le parcours d'achat et le point exact où naît un paiement.
Contenu	Panier, calcul du total, sélection du moyen de paiement, création de la transaction, confirmation, mise à jour de la commande.
Ce qu'on casse volontairement	Double clic sur le bouton de paiement ; fermeture du navigateur avant la confirmation ; panier modifié après l'initiation du paiement.
Livrable évalué	Un parcours complet fonctionnel, plus une note expliquant ce qui se produit dans chacun des trois cas d'échec.

11.2 Payment Connector Lab

Élément	Description
Objectif	Savoir écrire un connecteur complet à partir du cadre du Lab.
Contenu	Simulateur de contrepartie, traduction aller et retour, gestion des statuts, interrogation, notifications entrantes.
Ce qu'on casse volontairement	Timeout du réseau, notification en double, réponse tardive après abandon, code d'erreur inconnu.
Livrable évalué	Un connecteur passant la batterie de certification interne et sa fiche de documentation.

11.3 Card Lab

Élément	Description
Objectif	Maîtriser le cycle monétique complet.
Contenu	Autorisation, capture totale et partielle, annulation, remboursement, impayé, expiration d'autorisation.
Ce qu'on casse volontairement	Capture après expiration de l'autorisation ; remboursement supérieur au montant capturé ; impayé sur une commande déjà expédiée.
Livrable évalué	Une matrice d'états documentée et les tests automatisés correspondants.

11.4 QR Payment Lab

Élément	Description
Objectif	Comprendre l'initiation par présentation et ses contraintes de temps.
Contenu	Génération du code, durée de vie, lecture par le wallet, confirmation, notification au marchand.
Ce qu'on casse volontairement	Expiration du code pendant le scan ; scan du même code par deux wallets ; annulation par le client après scan.
Livrable évalué	Un connecteur QR et une note sur le choix de la durée de vie du code.

11.5 PI-SPI Lab

Élément	Description
Objectif	Manipuler des messages ISO 20022 et comprendre l'irrévocabilité.
Contenu	Construction et validation des messages, résolution d'alias, statuts d'acceptation et de rejet motivés, notification.
Ce qu'on casse volontairement	Montant décimal en XOF ; alias inconnu ; identifiant de message dupliqué ; tentative d'annulation après règlement.
Livrable évalué	Un connecteur validant les messages et un tableau des codes motifs rencontrés.

11.6 Payment Routing Lab

Élément	Description
Objectif	Choisir un chemin d'exécution et gérer le repli sans créer de doublon.
Contenu	Règles ordonnées, répartition de trafic, connecteur de repli, disjoncteur, mesure du taux de succès par chemin.
Ce qu'on casse volontairement	Repli déclenché sur un refus métier — l'erreur qui crée des doubles paiements ; prestataire lent mais fonctionnel ; bascule en boucle.
Livrable évalué	Un jeu de règles argumenté et la démonstration qu'aucun doublon n'est produit sous panne.

11.7 Failure Lab

Élément	Description
Objectif	Faire tomber la chaîne et vérifier qu'elle se relève sans perte ni doublon.
Contenu	Injection de pannes, rejeu, duplicata, idempotence à trois niveaux, reprise après redémarrage, files d'attente de reprise.
Ce qu'on casse volontairement	Coupure du connecteur en plein paiement ; redémarrage d'un service entre l'appel et la réponse ; base en lecture seule.
Livrable évalué	Un rapport de campagne : hypothèse, mesure, incident constaté, correctif, nouvelle mesure.



Figure 21 — Failure Lab — campagne avec injection de panne : un seuil tenu, un seuil manqué (capture illustrative)

11.8 Fraud Lab

Élément	Description
Objectif	Construire un dispositif de détection et mesurer son coût réel.
Contenu	Règles, pondérations, seuils d'alerte et de blocage, listes de surveillance, mesure des faux positifs et de la latence ajoutée.
Ce qu'on casse volontairement	Règle trop large bloquant des clients légitimes ; contournement par fractionnement des montants ; liste obsolète.
Livrable évalué	Un jeu de règles calibré, accompagné de la mesure du couple détection / faux positifs.

11.9 Refund Lab

Élément	Description
Objectif	Gérer tous les chemins de retour d'argent.
Contenu	Remboursement total, partiel, multiple, annulation avant capture, remboursement après règlement, avoir.
Ce qu'on casse volontairement	Remboursement cumulé dépassant le montant payé ; remboursement d'un paiement jamais capturé ; remboursement sur un instrument expiré.
Livrable évalué	Une matrice de décision remboursement / annulation / avoir et son implémentation.

11.10 Reconciliation Lab

Élément	Description
Objectif	Expliquer chaque écart entre commande, paiement, règlement et comptabilité.
Contenu	Rapprochement à cinq niveaux, typologie des écarts, compte d'attente, règles d'âge, rapports quotidiens.
Ce qu'on casse volontairement	Webhook perdu ; frais opérateur non modélisés ; règlement décalé ; doublon issu d'un repli mal configuré.
Livrable évalué	Un rapport de rapprochement quotidien où chaque écart est expliqué et tracé.

11.11 Subscription Lab

Élément	Description
Objectif	Comprendre le paiement sans client devant l'écran.
Contenu	Abonnement, périodicité, facturation, prélèvement, échec, relance, suspension, changement de plan.
Ce qu'on casse volontairement	Échec du premier prélèvement ; instrument expiré en cours d'abonnement ; changement de plan en milieu de période.
Livrable évalué	Une politique de relance argumentée et mesurée sur une simulation de trois mois.

11.12 Marketplace Lab

Élément	Description
Objectif	Répartir un paiement et régler plusieurs bénéficiaires.
Contenu	Commission, net vendeur, calendrier de règlement, blocages, remboursement d'une commande multi-vendeurs.
Ce qu'on casse volontairement	Remboursement partiel touchant un seul vendeur ; vendeur bloqué après encaissement ; arrondi de commission.
Livrable évalué	Un modèle de répartition documenté et le rapport de règlement associé.

11.13 Observability Lab

Élément	Description
Objectif	Retrouver n'importe quelle commande en moins d'une minute.
Contenu	Instrumentation, propagation du contexte de trace, attributs métier normalisés, tableaux de bord, alertes, runbooks.
Ce qu'on casse volontairement	Trace interrompue par un appel non instrumenté ; donnée sensible qui fuit dans un attribut ; alerte non actionnable.
Livrable évalué	Un tableau de bord opérationnel et trois runbooks d'incident rédigés et testés.

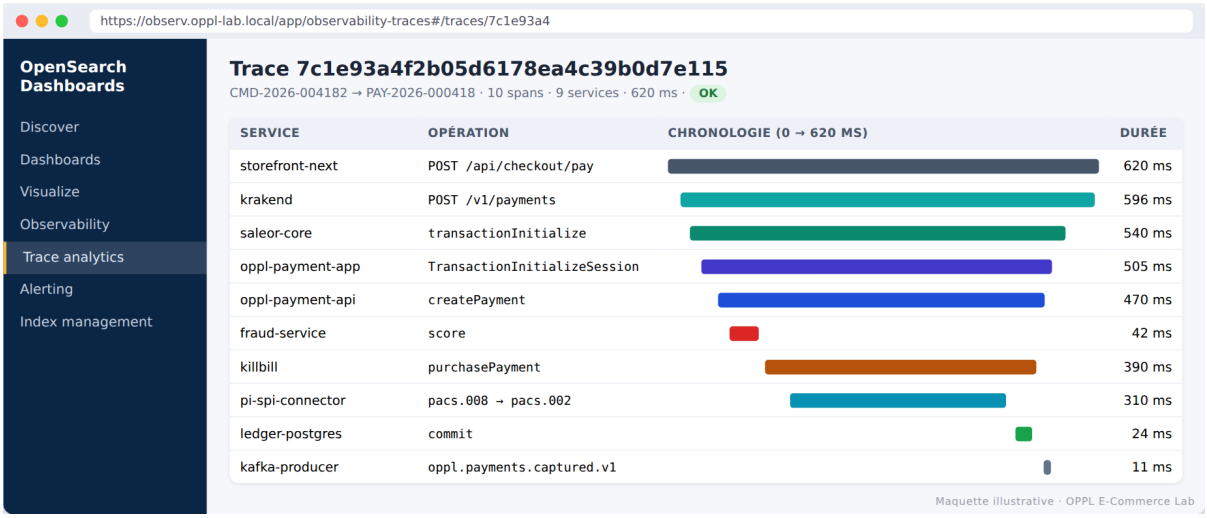


Figure 22 — Trace complète d'une commande, du storefront au connecteur (capture illustrative)

11.14 Progression et évaluation

Tableau 25 — Progression type sur douze semaines

Semaine	Labs	Compétence acquise
S1–S2	Checkout, Observability (initiation)	Lire une chaîne de paiement et la suivre de bout en bout
S3–S4	Payment Connector, Card	Écrire un connecteur et maîtriser le cycle monétique
S5–S6	QR, PI-SPI	Manipuler des protocoles et des messages normalisés
S7	Payment Routing	Décider d'un chemin d'exécution et gérer le repli
S8	Failure	Garantir l'absence de perte et de doublon sous panne
S9	Fraud, Refund	Arbitrer entre protection et expérience client
S10	Reconciliation	Expliquer chaque écart, du clic à la comptabilité
S11	Subscription, Marketplace	Traiter les cas sans client présent et multi-bénéficiaires
S12	Observability (approfondissement), restitution	Exploiter, alerter, documenter, transmettre

12. Tests, résilience et performance

12.1 Pyramide de tests

Tableau 26 — Pyramide de tests

Niveau	Portée	Outils	Déclenchement
Unitaires	Traductions, règles, calculs de montants	Framework de test du langage	À chaque commit
Contrat	Conformité aux schémas OpenAPI, GraphQL, AsyncAPI	Validation de schéma, tests de contrat	À chaque demande de modification
Intégration	Deux ou trois composants réels avec simulateur	Conteneurs de test, environnement de niveau 1	À chaque demande de modification
Bout en bout	Commande complète sur le Lab de promotion	Scénarios automatisés, navigateur sans interface	Quotidien et avant démonstration
Non fonctionnels	Charge, panne, sécurité	Outil de charge, injection de pannes, analyse dynamique	Hebdomadaire ou par campagne

12.2 Catalogue de tests de référence

Tableau 27 — Catalogue de tests de référence

ID	Scénario	Type	Résultat attendu
T-01	Commande de 25 000 XOF payée en instantané	Positif	Statut encaissé, commande payée, trace complète
T-02	Rejeu de T-01 avec la même clé d'idempotence	Négatif	Même réponse, aucune seconde transaction
T-03	Rejeu avec la même clé mais un montant différent	Négatif	Erreur explicite, aucun paiement créé
T-04	Fonds insuffisants	Négatif	Échec définitif, aucun repli déclenché
T-05	Timeout du connecteur puis rétablissement	Résilience	Statut non terminal puis réconciliation, aucun doublon
T-06	Notification Mobile Money reçue deux fois	Résilience	Un seul effet, seconde notification acquittée
T-07	Notification jamais reçue	Résilience	Interrogation planifiée, commande débloquée
T-08	Capture partielle puis remboursement partiel	Positif	Montants cohérents dans commerce, facturation et rapprochement
T-09	Remboursement cumulé supérieur au montant payé	Négatif	Refus, aucune écriture
T-10	Panne du connecteur primaire	Résilience	Bascule vers le repli, taux de succès maintenu
T-11	Montant décimal en XOF	Négatif	Rejet explicite dès la validation
T-12	Jeton expiré à la passerelle	Sécurité	Rejet avant d'atteindre les services métier
T-13	Redémarrage d'un service pendant un paiement	Résilience	Reprise sans perte ni doublon
T-14	Même connecteur sur Saleor puis Vendure	Architecture	Comportement identique, traces comparables

12.3 Injection de pannes

Tableau 28 — Expériences d'injection de pannes

Expérience	Hypothèse	Mesure	Critère de succès
Connecteur carte indisponible 3 minutes	Le repli absorbe la panne	Taux de succès, doublons	Aucun doublon, succès > 95 %
Latence de 800 ms sur le réseau instantané	Les délais et reprises absorbent la dégradation	Latence p95, abandons	Abandons < 3 %
Suppression du service de paiement en plein appel	La reprise rétablit l'état	Commandes bloquées	Aucune commande sans état final
Base de paiement en lecture seule	Les échecs sont propres	États finaux	Aucun paiement « à moitié » enregistré
Notifications rejouées en masse	L'idempotence tient	Effets de bord	Un seul effet par référence

12.4 Performance

Les campagnes de performance visent la **comparabilité**, pas le record. Chaque campagne fixe l'environnement, les versions, le profil de charge, la durée et les seuils, puis publie ses résultats bruts. Un chiffre de performance obtenu dans le Lab ne préjuge en rien des capacités d'un système de production.

perf — profil de charge du checkout

```
export const options = {
  scenarios: {
    checkout: { executor: 'constant-arrival-rate', rate: 200, timeUnit: '1s',
                duration: '15m', preAllocatedVUs: 150 },
  },
  thresholds: {
    http_req_duration:            ['p(95)<600'],
    http_req_failed:              ['rate<0.005'],
    'checks{check:no_duplicate}': ['rate==1.00'],   // exigence absolue
  },
};
```

Le seuil non négociable

Parmi tous les seuils d'une campagne, un seul ne tolère aucune exception : l'absence de doublon. Un taux d'erreur légèrement dépassé se corrige ; un double débit détruit la confiance et coûte bien plus cher que la panne qui l'a provoqué.

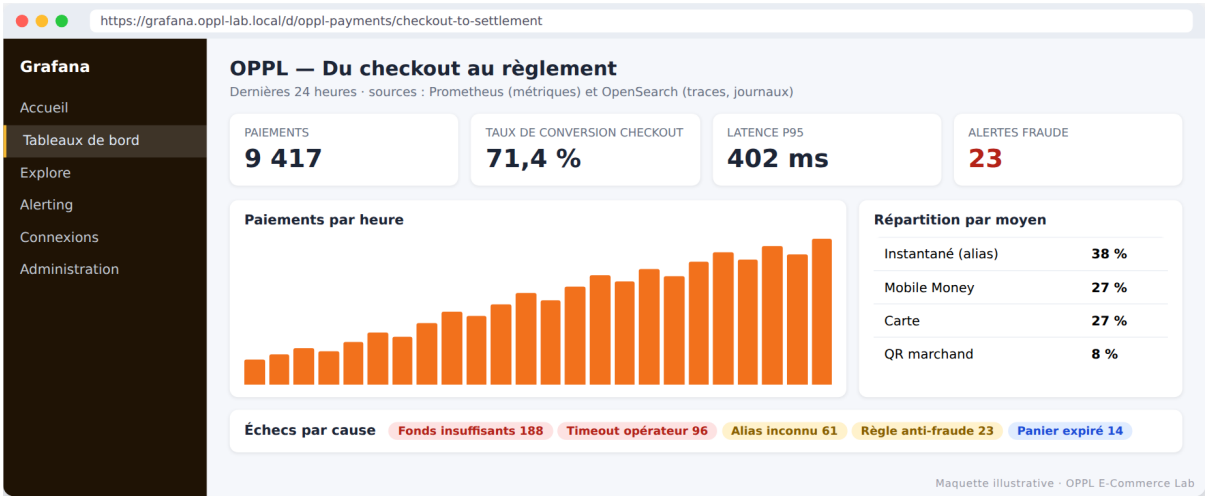


Figure 23 — Tableau de bord Grafana — volumes, conversion, latence et causes d'échec (capture illustrative)

13. Sécurité, conformité et licences

13.1 Sécurité par défaut

Tableau 29 — Contrôles de sécurité du Lab

Domaine	Contrôles appliqués
Identité et accès	Authentification par jeton pour tout appel ; comptes de service à privilèges minimaux ; authentification renforcée sur les consoles ; séparation stricte des rôles client, marchand, opérateur
Réseau	Interdiction par défaut, autorisations explicites ; chiffrement de bout en bout ; consoles d'administration accessibles uniquement par réseau privé
Données	Données synthétiques exclusivement ; tokenisation au plus tôt ; masquage systématique dans journaux, traces et événements ; chiffrement des volumes
Secrets	Gestionnaire de secrets dédié ; aucun secret dans les dépôts ; détection automatique en intégration continue ; rotation planifiée
Chaîne logicielle	Inventaire des composants généré automatiquement ; analyse de vulnérabilités ; signature des images ; admission restreinte aux images signées
Applications	Validation stricte des entrées ; protection contre le rejeu ; limitation de débit par client ; journalisation des décisions de sécurité
Audit	Piste d'audit immuable des événements de paiement ; historique Git ; journaux d'administration conservés

13.2 Maîtriser le périmètre de conformité carte

Dès qu'un composant stocke, traite ou transmet des données de carte, il entre dans le périmètre de la norme de sécurité des données de l'industrie des cartes de paiement. Le Lab n'est pas certifié et n'a pas vocation à l'être, puisqu'il n'utilise que des numéros de test. Il est en revanche l'endroit idéal pour **concevoir la réduction de périmètre** avant toute trajectoire de production.

- Tokeniser dès le point de collecte : le storefront ne transmet jamais de numéro de carte au moteur de commerce.
- Isoler le service de tokenisation dans son propre cloisonnement réseau, avec ses propres droits et sa propre journalisation.
- Interdire par construction l'apparition d'un numéro de carte dans un journal, une trace ou un événement — et le vérifier automatiquement.
- Documenter le flux de données de carte : qui le voit, où il transite, combien de temps il est conservé.
- Traiter la gestion des clés comme un sujet à part entière, même avec un module logiciel de test.

13.3 Cadre réglementaire régional

Tableau 30 — Références réglementaires (liste indicative, à valider par la conformité)

Domaine	Portée pour le Lab
Systèmes et moyens de paiement dans l'UEMOA	Cadre général de référence pour la modélisation des acteurs et des flux du Lab

Domaine	Portée pour le Lab
Monnaie électronique et établissements de paiement	Modélisation des rôles simulés : marchand, prestataire, établissement, agrégateur
Paiement instantané interopérable régional	Cible d'intégration simulée ; aucune connexion réelle depuis le Lab
Lutte contre le blanchiment et le financement du terrorisme	Justification des scénarios de surveillance et de la traçabilité exigée
Protection des données personnelles	Justification du recours exclusif aux données synthétiques
Encaissement pour compte de tiers	Point à clarifier pour tout scénario marketplace dépassant la démonstration
Sécurité de l'information et gouvernance de l'IA	Référentiels applicables à l'exploitation du Lab et à ses usages assistés par IA

À valider

Les références réglementaires ci-dessus sont indicatives. Elles doivent être vérifiées dans leur version en vigueur par la direction de la conformité avant toute communication externe, toute ouverture du sandbox à des partenaires et tout usage au-delà de la formation interne.

13.4 Conformité des licences

Le Lab tient un inventaire des licences généré automatiquement à partir de l'inventaire des composants. Toute dépendance nouvelle sous licence à réciprocité forte ou sous licence non open source déclenche une revue. Le tableau ci-dessous synthétise les points de vigilance de la pile retenue ; il doit être revérifié à chaque montée de version.

Tableau 31 — Synthèse des licences — à confirmer par dépôt et par version

Composant	Licence	Vigilance	Point d'attention
Saleor (cœur et tableau de bord)	BSD 3-Clause	Faible	Licence permissive ; attribution
Vendure	GPLv3 ou licence commerciale	Élevée	Réciprocité forte : analyse juridique requise pour tout produit dérivé
Kill Bill	Apache 2.0	Faible	Vérifier la licence de chaque plugin utilisé
KrakenD (édition communautaire)	Apache 2.0	Moyenne	Certaines fonctions relèvent de l'édition entreprise
Keycloak, Prometheus, Kafka, OpenTelemetry	Apache 2.0	Faible	Attribution
Next.js	MIT	Faible	Attribution
PostgreSQL	PostgreSQL License	Faible	Licence permissive
Grafana	AGPLv3	Moyenne	Usage interne sans difficulté ; redistribution ou service dérivé à analyser
Redis	Selon version : BSD 3-Clause jusqu'à 7.2 ; modèle tri-licence à partir de 8	Élevée	Le choix de version engage le modèle de licence : à trancher explicitement

Composant	Licence	Vigilance	Point d'attention
Redpanda (alternative à Kafka)	Business Source License 1.1	Élevée	Source-available, pas open source ; restriction d'usage — écarté par le Lab

La leçon de licence du Lab

« Open source » n'est pas une catégorie homogène. Entre une licence permissive, une licence à réciprocité forte et une licence simplement *source-available*, les conséquences juridiques et commerciales sont radicalement différentes. Cette distinction fait partie intégrante de la formation : un Payment Engineer qui choisit un composant sans lire sa licence engage son institution sans le savoir.

14. Payment Engineers et offres de service

14.1 Le référentiel de compétences

Le parcours vise une cible précise : un profil BAC+3 capable, à l'issue de la formation, d'intervenir sur une infrastructure de paiement réelle. Dix compétences constituent le référentiel d'évaluation.

Tableau 32 — Référentiel de compétences Payment Engineer

#	Compétence	Preuve attendue
1	Comprendre et décrire une chaîne de paiement complète	Schéma commenté et soutenance
2	Intégrer une API de paiement	Intégration fonctionnelle avec gestion des erreurs
3	Développer un connecteur de bout en bout	Connecteur certifié par le cadre du Lab
4	Gérer les erreurs et les cas limites	Suite de tests négatifs complète
5	Implémenter l'idempotence aux trois niveaux	Démonstration sous rejeu et sous panne
6	Comprendre le règlement et ses décalages	Analyse d'un cycle complet
7	Réaliser un rapprochement et expliquer les écarts	Rapport quotidien argumenté
8	Instrumenter un paiement	Trace complète et tableau de bord
9	Appliquer les contrôles de sécurité	Revue de code et test d'absence de fuite de données
10	Connecter une application métier à une infrastructure de paiement	Projet final de bout en bout

14.2 Les quatre profils visés

Tableau 33 — Les quatre profils du parcours

Profil	Centre de gravité	Labs déterminants
Payment Engineer	Intégration, connecteurs, gestion des erreurs	Connector, Card, PI-SPI, Failure
Payment Architect	Architecture, contrats, arbitrages	Routing, Reconciliation, Marketplace
Payment SRE	Exploitation, résilience, observabilité	Failure, Observability, campagnes de performance
Payment Platform Engineer	Socle, sécurité, industrialisation	Infrastructure, sécurité, usine à connecteurs

14.3 Offres de service envisageables

Les capacités construites pour la formation constituent également une base d'offres de service. Chacune suppose un niveau de maturité et un cadre contractuel propres : le tableau ci-dessous décrit des possibilités, pas un catalogue arrêté.

Tableau 34 — Offres de service envisageables

Offre	Contenu	Bénéficiaires	Prérequis
Payment Engineering Academy	Parcours de 12 semaines, 13 labs, évaluation et attestation	Jeunes diplômés, équipes internes	Lab de niveau 2 stable, formateurs certifiés
Payment Connector Factory	Production industrialisée de connecteurs selon le cadre du Lab	Banques, fintechs, PSP, agrégateurs	Cadre commun mature, méthode en sept étapes éprouvée
Payment Sandbox mutualisé	Environnement de test partagé et instrumenté	E-commerçants, éditeurs, intégrateurs	Isolation par tenant, quotas, engagement de service
Payment Integration Testing	Campagnes de tests d'intégration, de charge et de résilience	Partenaires en phase d'intégration	Catalogue de tests et rapports normalisés
E-Commerce Payment Certification Lab	Vérification de conformité d'une intégration au cadre du Lab	Marchands et prestataires	Référentiel publié et gouvernance de la certification

Précision importante

La « certification » proposée par le Lab est une **conformité à son propre cadre technique**. Elle n'a aucune valeur réglementaire et ne remplace ni l'homologation d'un opérateur de système de paiement, ni une certification de schéma de cartes, ni un agrément. Cette limite doit figurer explicitement dans tout document remis à un client.

15. Gouvernance et feuille de route

15.1 Organisation

Tableau 35 — Organisation type

Rôle	Responsabilités	Charge indicative
Sponsor	Porte l'initiative, arbitre, valorise les résultats	Ponctuelle
Responsable du Lab	Pilotage, feuille de route, qualité, animation de la communauté	50 %
Architecte référent	Architecture, contrats, cadre des connecteurs, décisions documentées	50 %
Ingénieurs plateforme (2)	Socle, sécurité, observabilité, injection de pannes	100 %
Ingénieurs logiciels (3)	Couche de paiement, connecteurs, simulateurs, tests	100 %
Formateurs (2)	Animation des labs, évaluation, accompagnement des apprenants	60 %
Correspondants risques et conformité	Scénarios de fraude, revue des contrôles, validation des références	10 à 20 %

15.2 Feuille de route

Tableau 36 — Feuille de route sur six mois

Période	Jalon	Livrables
Mois 1	Socle et chaîne minimale	Infrastructure, identité, observabilité, Saleor, storefront, paiement manuel
Mois 2	Chaîne de paiement complète	Couche de paiement, Kill Bill, application de paiement, premier connecteur
Mois 3	Couverture des moyens de paiement	Connecteurs carte, Mobile Money, QR, instantané et leurs simulateurs
Mois 4	Robustesse	Labs d'échec, de routage et de rapprochement ; première campagne de performance
Mois 5	Formation	Supports des 13 labs, référentiel d'évaluation, première promotion pilote
Mois 6	Ouverture	Second moteur de commerce, sandbox partenaires, bilan et décisions d'industrialisation

15.3 Risques et parades

Tableau 37 — Principaux risques

Risque	Probabilité	Impact	Parade
Le Lab dérive vers un projet de boutique en ligne	Élevée	Élevé	Charte du Lab : le commerce est un décor, le paiement est la matière
Un environnement de formation devient un pilote de production	Moyenne	Élevé	Séparation stricte, données synthétiques, revue d'architecture obligatoire

Risque	Probabilité	Impact	Parade
Mauvaise lecture des licences	Moyenne	Élevé	Inventaire automatique, revue juridique, tableau de vigilance tenu à jour
Dépendance à quelques experts	Élevée	Moyen	Binômage, documentation, rotation des animateurs de labs
Simulateurs trop permissifs	Moyenne	Moyen	Chaque simulateur doit pouvoir reproduire tous les modes de défaillance
Chiffres de performance mal interprétés	Moyenne	Moyen	Protocole publié, résultats bruts, avertissement systématique
Attentes réglementaires mal cadrées	Moyenne	Élevé	Mention explicite des limites dans chaque support remis

15.4 Indicateurs de réussite

- Nombre d'apprenants ayant validé les treize labs et le projet final.
- Temps de reconstruction complète du Lab depuis les dépôts — objectif : moins d'une journée.
- Nombre de connecteurs produits selon le cadre commun et durée moyenne de production.
- Taux de réussite des campagnes de tests de référence, et notamment l'absence totale de doublon.
- Nombre de partenaires accueillis dans le sandbox et de campagnes d'intégration menées.
- Nombre de contributions en amont aux projets open source utilisés.

Annexes

Annexe A — Glossaire

Tableau 38 — Glossaire

Terme	Définition
ADR	Décision d'architecture consignée par écrit, datée et argumentée
Alias de paiement	Identifiant simple, souvent un numéro de téléphone, permettant d'atteindre un compte
Autorisation	Réservation de fonds sur un instrument, sans encaissement immédiat
Capture	Encaissement de tout ou partie d'une autorisation
Clé d'idempotence	Identifiant fourni par l'appelant garantissant qu'un rejeu ne produit pas un second effet
CloudEvents	Spécification décrivant l'enveloppe standard d'un événement
Headless	Architecture où le moteur de commerce n'impose aucune interface : il expose une API
Impayé	Contestation ou rejet postérieur à l'encaissement
ISO 8583	Norme de messages pour les transactions initiées par carte
ISO 20022	Norme universelle de messagerie financière
Marketplace	Place de marché réunissant plusieurs vendeurs sous une même boutique
Rapprochement	Vérification de la cohérence entre commande, paiement, règlement et comptabilité
Règlement	Transfert effectif des fonds, souvent différé par rapport au paiement
Repli	Connecteur de secours activé sur échec technique du connecteur principal
Storefront	Interface de vente vue par le client
Tokenisation	Remplacement d'une donnée sensible par un jeton sans valeur intrinsèque
Unité mineure	Plus petite subdivision d'une devise ; en XOF, le franc lui-même
Webhook synchrone	Appel sortant dont l'émetteur attend la réponse pour poursuivre son traitement

Annexe B — Check-list de mise en service

Tableau 39 — Check-list de mise en service

#	Contrôle	Fait
1	Charte du Lab signée : non-production, données synthétiques, périmètre	☐
2	Dépôts créés, branches protégées, revue obligatoire	☐
3	Socle déployé et synchronisé automatiquement depuis Git	☐
4	Gestionnaire de secrets initialisé, aucun secret dans les dépôts	☐
5	Identités et rôles importés depuis Git, authentification renforcée active	☐
6	Cloisonnement réseau actif et flux autorisés testés un par un	☐
7	Masquage des données sensibles vérifié sur journaux, traces et événements	☐

#	Contrôle	Fait
8	Devise XOF configurée et testée sur toute la chaîne (test du facteur 100)	☐
9	Générateur de données synthétiques exécuté et vérifié	☐
10	Les dix phases de déploiement validées et prouvées	☐
11	Catalogue de tests de référence exécuté, aucun doublon constaté	☐
12	Inventaire des licences généré et revu	☐
13	Tableaux de bord et trois runbooks d'incident en place	☐
14	Limites réglementaires rappelées dans les supports destinés aux tiers	☐

Annexe C — Références

- Documentation officielle des projets utilisés : Saleor, Vendure, Kill Bill, KrakenD, Keycloak, PostgreSQL, Redis, Apache Kafka, OpenTelemetry, Prometheus, Grafana, Next.js, Kubernetes.
- Documentation de l'API de transactions et des applications de paiement du moteur de commerce retenu.
- Documentation de l'interface de plugin de paiement de Kill Bill et de son mécanisme de réconciliation interne.
- Spécifications : ISO 8583, ISO 20022, OAuth 2.0, OpenID Connect, OpenAPI 3, GraphQL, CloudEvents, W3C Trace Context.
- Textes réglementaires applicables dans la zone UEMOA, à obtenir dans leur version en vigueur auprès de la direction de la conformité.
- Licences citées : BSD 3-Clause, MIT, Apache 2.0, GPLv3, AGPLv3, Business Source License 1.1, licences de Redis selon version.

Signature

Open Payment Platform Lab — Des paiements plus ouverts, une Afrique plus connectée.

Compétences · Innovation · Impact

Document assisté par IA, gouverné par ISO/IEC 42001. Relecture humaine et validation technique requises avant diffusion.