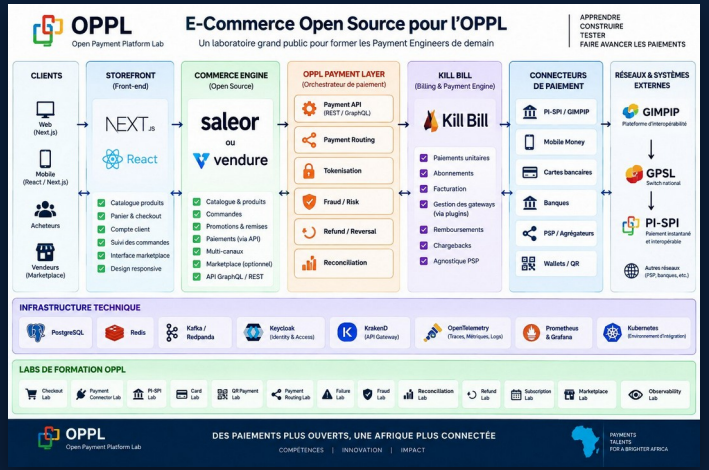
MISE EN ŒUVRE · DÉPLOIEMENT · DÉVELOPPEMENT · FORMATION

E-Commerce Open Source pour l'OPPL



Un laboratoire pour former les Payment Engineers de demain, en construisant une chaîne de paiement complète, du clic au règlement.

Apprendre · Construire · Tester · Faire avancer les paiements

Lionel SAKITI
Professionnel des infrastructures numériques · Septembre 2026

Les systèmes de paiement réels sont fermés

On n’apprend pas les paiements dans une salle de cours. On les apprend en construisant une chaîne qui tombe, puis en la relevant.

Les formations se replient sur la théorie. Les équipes découvrent les vraies difficultés en incident de production — au pire moment, au pire coût.



Environnements fermés

Aucun accès aux systèmes réels



Formation théorique

Des schémas, pas des pannes

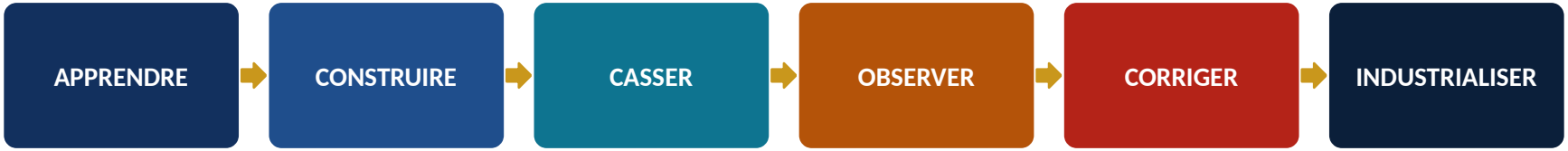


Apprentissage en production

Le plus coûteux qui soit

Le e-commerce, seul terrain de jeu légal et complet

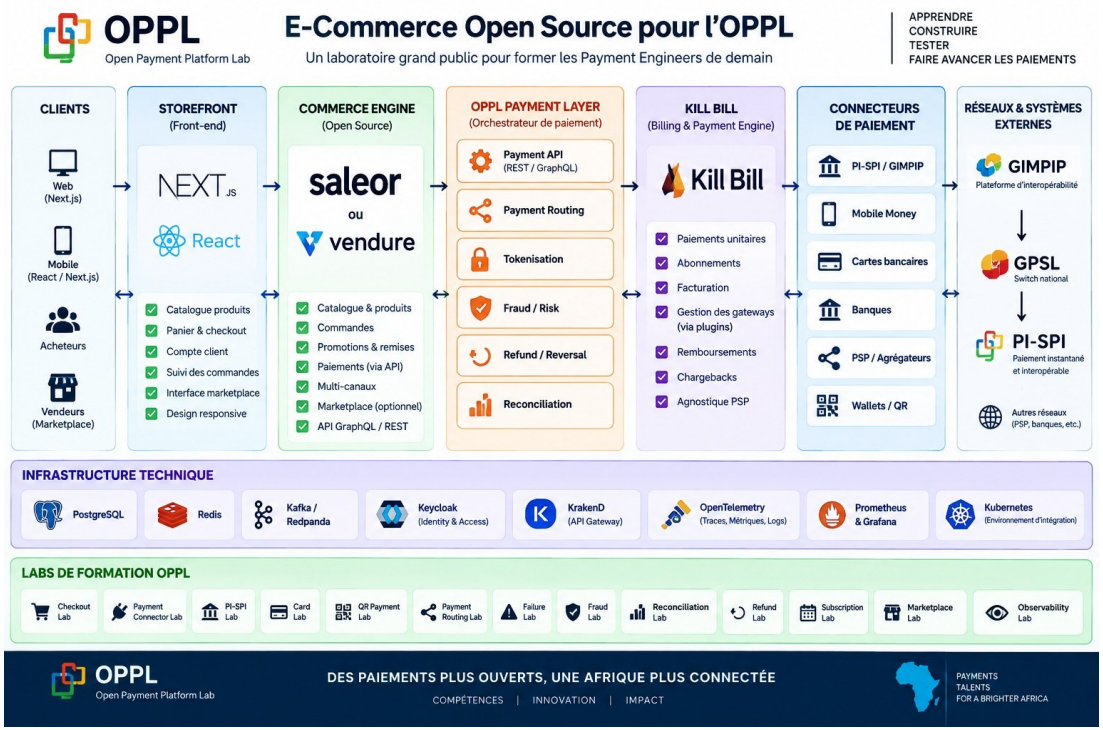
C'est le seul domaine où l'on peut construire, sans risque et avec des composants ouverts, une chaîne de paiement complète : vrai panier, vraies transactions, vrais remboursements, vrais écarts de rapprochement.



Une commande de 25 000 FCFA concentre presque toutes les difficultés du métier

Choix du moyen de paiement · authentification · autorisation · capture · confirmation asynchrone · remboursement total et partiel · annulation · impayé · rapprochement · règlement vendeur · abonnement · fraude · idempotence · observabilité

L'architecture cible



- **Clients** — acheteurs, vendeurs marketplace
- **Storefront** — Next.js / React
- **Commerce Engine** — Saleor ou Vendure
- **OPPL Payment Layer** — le cœur du Lab
- **Kill Bill** — facturation et exécution
- **Connecteurs** — carte, Mobile Money, QR, instantané
- **Réseaux externes** — simulés dans le Lab

Sept couches, une responsabilité chacune

1. Clients	Acheteurs, vendeurs, terminaux
2. Storefront	Next.js · React
3. Commerce Engine	Saleor · Vendure
4. OPPL Payment Layer	API · routage · tokenisation · fraude · rapprochement
5. Billing & Payment	Kill Bill + plugins
6. Connecteurs	Carte · Mobile Money · QR · instantané · banque
7. Réseaux externes	GIMPIP · GPSL · PI-SPI · PSP · banques

Six règles non négociables

Contrats explicites

OpenAPI, GraphQL, AsyncAPI — versionnés et testés

Idempotence de bout en bout

Du storefront jusqu'au connecteur, une seule clé

Un identifiant métier unique

Le paymentId suit la commande jusqu'au règlement

Aucune donnée sensible en clair

Tokenisation au plus tôt, masquage systématique

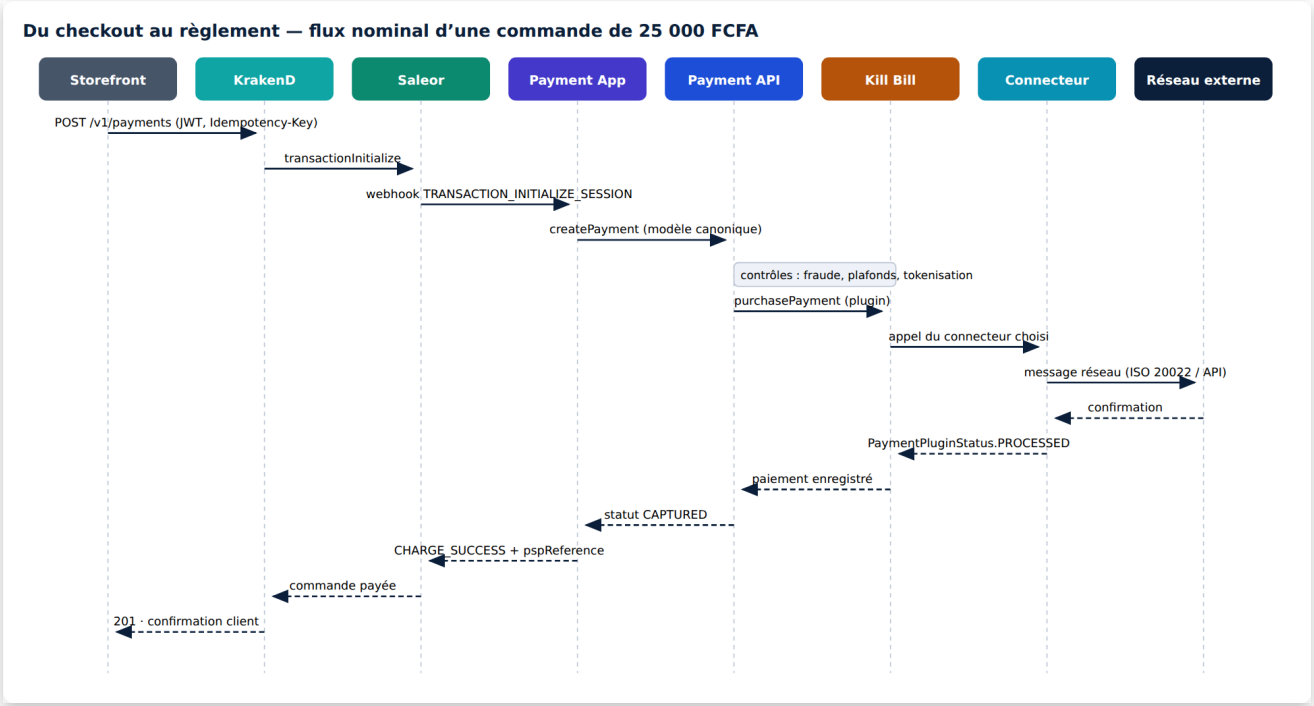
Des statuts finis et explicites

Aucun état implicite, aucun « inconnu » sans reprise

Le commerce ne pilote pas le paiement

Il demande ; la couche paiement décide

Du checkout au règlement



Le storefront génère la clé d'idempotence ; elle est propagée jusqu'au connecteur

Un vocabulaire de statuts commun

Statut canonique OPPL	Signification	Moteur de commerce	Kill Bill
PENDING	Demande enregistrée, pas de réponse	CHARGE_REQUEST	PENDING
ACTION_REQUIRED	Le client doit encore agir	CHARGE_ACTION_REQUIRED	PENDING
AUTHORIZED	Fonds réservés, capture à venir	AUTHORIZATION_SUCCESS	SUCCESS (AUTHORIZE)
CAPTURED	Paiement encaissé	CHARGE_SUCCESS	SUCCESS (PURCHASE)
FAILED	Refus définitif	CHARGE_FAILURE	PAYMENT_FAILURE
REFUNDED	Remboursement exécuté	REFUND_SUCCESS	SUCCESS (REFUND)
UNKNOWN	Aucune réponse — à réconcilier	—	UNDEFINED (à éviter)

La règle la plus importante du Lab Un connecteur ne renvoie jamais un échec définitif quand il n'a pas de réponse. Confondre « échec » et « je ne sais pas » est la cause n° 1 des doubles débits.

I

Le socle

Choisir le moteur de commerce, dimensionner, déployer en dix phases.

Saleor ou Vendure ? Le choix n’est pas que technique

Critère	Saleor	Vendure
Langage	Python	TypeScript
Licence communautaire	BSD 3-Clause (permissive)	GPLv3 ou licence commerciale
Intégration paiement	Application externe + webhooks	Plugin dans le serveur
Découplage commerce / paiement	Fort — proche du réel	Moyen
Idempotence de l’API	Prévue nativement	À implémenter dans le plugin
Ressources	Plus élevées	Plus légères

Recommandation Saleor comme moteur de référence — son modèle d’application externe enseigne le bon découplage et sa licence n’impose rien. Vendure comme second moteur, pour démontrer que la couche de paiement est indépendante du moteur de commerce.

« Open source » n'est pas une catégorie homogène

Composant	Licence	Vigilance	Point d'attention
Vendure	GPLv3 ou commerciale	**Élevée**	Réciprocité forte : analyse juridique avant tout produit dérivé
Redis	BSD jusqu'à 7.2 ; tri-licence à partir de 8	**Élevée**	Le choix de version engage le modèle de licence
Redpanda	Business Source License 1.1	**Élevée**	Source-available, pas open source — écarté par le Lab
Grafana	AGPLv3	**Moyenne**	Usage interne sans difficulté ; service dérivé à analyser
KrakenD (CE)	Apache 2.0	**Moyenne**	Certaines fonctions en édition entreprise
Saleor	BSD 3-Clause	**Faible**	Licence permissive ; attribution
Kill Bill, Keycloak, Kafka, Prometheus, OTel	Apache 2.0	**Faible**	Attribution

Un ingénieur qui choisit un composant sans lire sa licence engage son institution sans le savoir.

Trois niveaux d'environnement

1 Poste apprenant

Docker Compose · 16 Go RAM

- 1 personne
- Un moteur + un connecteur
- Labs individuels

2 Lab de promotion

Kubernetes · 3 nœuds

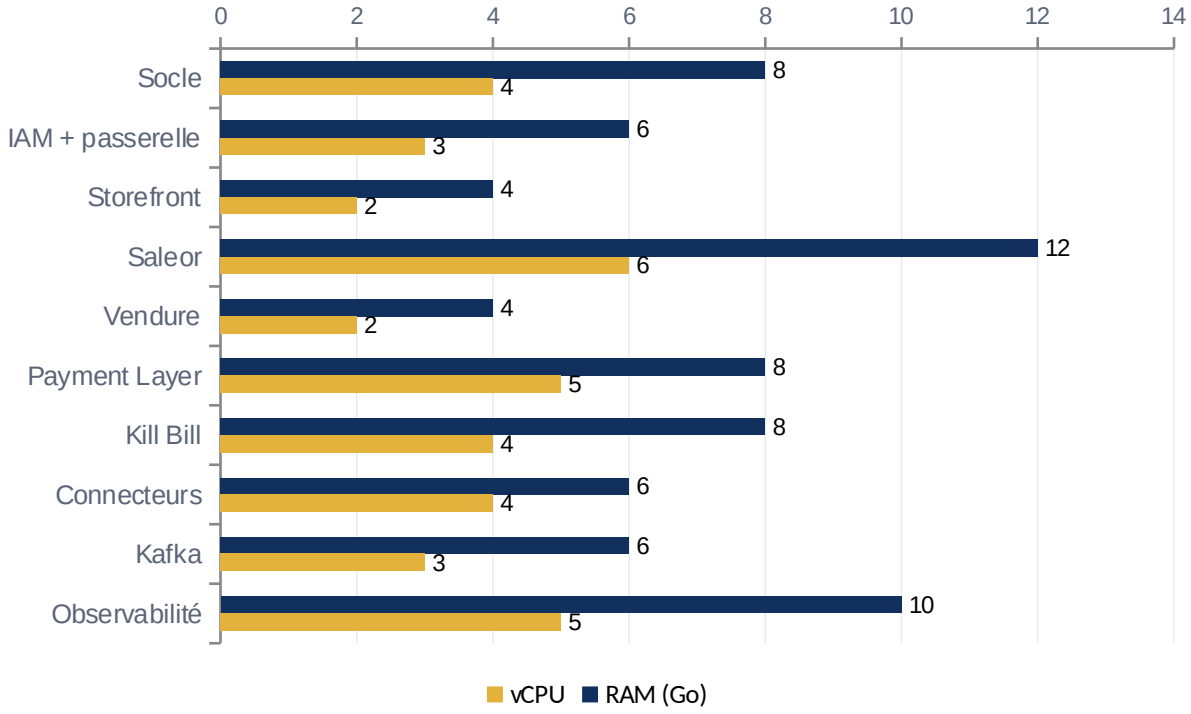
- 10 à 25 personnes
- Chaîne complète + simulateurs
- Parcours et démonstrations

3 Sandbox mutualisé

Kubernetes · isolation par tenant

- Partenaires externes
- Quotas et cloisonnement
- Tests d'intégration

Dimensionnement du Lab de promotion



≈ 38

vCPU

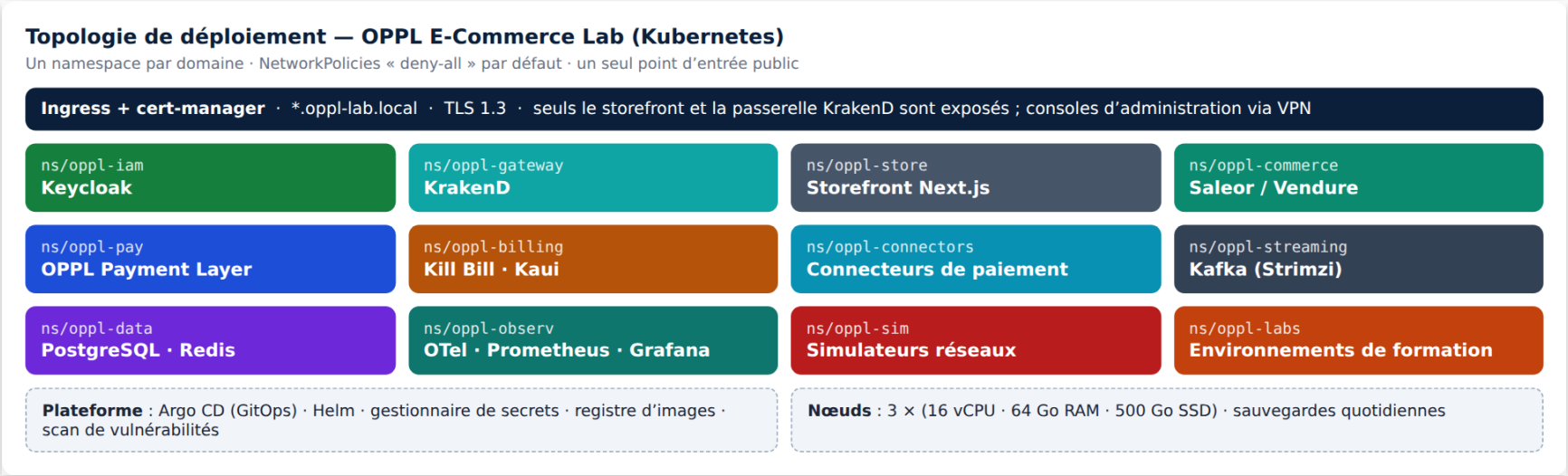
≈ 72 Go

RAM

≈ 600 Go

Stockage

Un namespace par domaine, deny-all par défaut



Dix phases de déploiement

Environ 25 jours-ingénieur · chaque phase a un critère de sortie vérifiable et une preuve archivée

0 — Socle
K8s, GitOps, secrets

1 — Identité
Keycloak, KrakenD

2 — Données
PostgreSQL, Redis, Kafka

3 — Observabilité
OTel, Prometheus, Grafana

4 — Commerce
Saleor + storefront

5 — Payment Layer
5 services OPPL

6 — Billing
Kill Bill + plugin

7 — Payment App
Webhooks synchrones

8 — Connecteurs
4 moyens + simulateurs

9 — Second moteur
Vendure + labs

La passerelle valide avant d'atteindre les services

- Configuration **déclarative**, validée avant déploiement
- Jetons vérifiés par la passerelle, pas par chaque service
- Limitation de débit **par client**
- Traçage activé dès le point d'entrée
- En-têtes d'idempotence et de trace propagés

KrakenD — configuration de la passerelle

```
dev@oppl-lab:~/gateway$ cat krakend.json | jq '.endpoints[0:2]'
[
  {
    "endpoint": "/v1/payments",
    "method": "POST",
    "input_headers": ["Authorization", "Idempotency-Key", "traceparent"],
    "extra_config": {
      "auth/validator": { "alg": "RS256",
        "jwk_url": "http://keycloak.oppl-iam/realms/oppl/protocol/openid-connect/certs",
        "roles": ["payments:write"], "cache": true },
      "qos/ratelimit/router": { "max_rate": 50, "client_max_rate": 5, "strategy": "header" },
      "telemetry/opentelemetry": { "exporters": ["otlp"] }
    },
    "backend": [{ "host": ["http://payment-api.oppl-pay:8080"], "url_pattern": "/payments" }]
  },
  { "endpoint": "/v1/payments/{id}", "method": "GET", "…": "…" }
]
dev@oppl-lab:~/gateway$ krakend check -c krakend.json -t
Parsing configuration file: krakend.json
Syntax OK!
dev@oppl-lab:~/gateway$ curl -s -o /dev/null -w '%{http_code}\n' -X POST https://api.oppl-lab.local/v1/payments
401   # jeton absent : la passerelle rejette avant d'atteindre le service
```

Maquette illustrative · OPPL E-Commerce Lab

Capture illustrative — maquette reconstituée, chiffres fictifs

Le piège du franc CFA

Le 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. Un facteur 100 mal placé transforme 25 000 FCFA en 250 FCFA — ou en 2 500 000 FCFA.

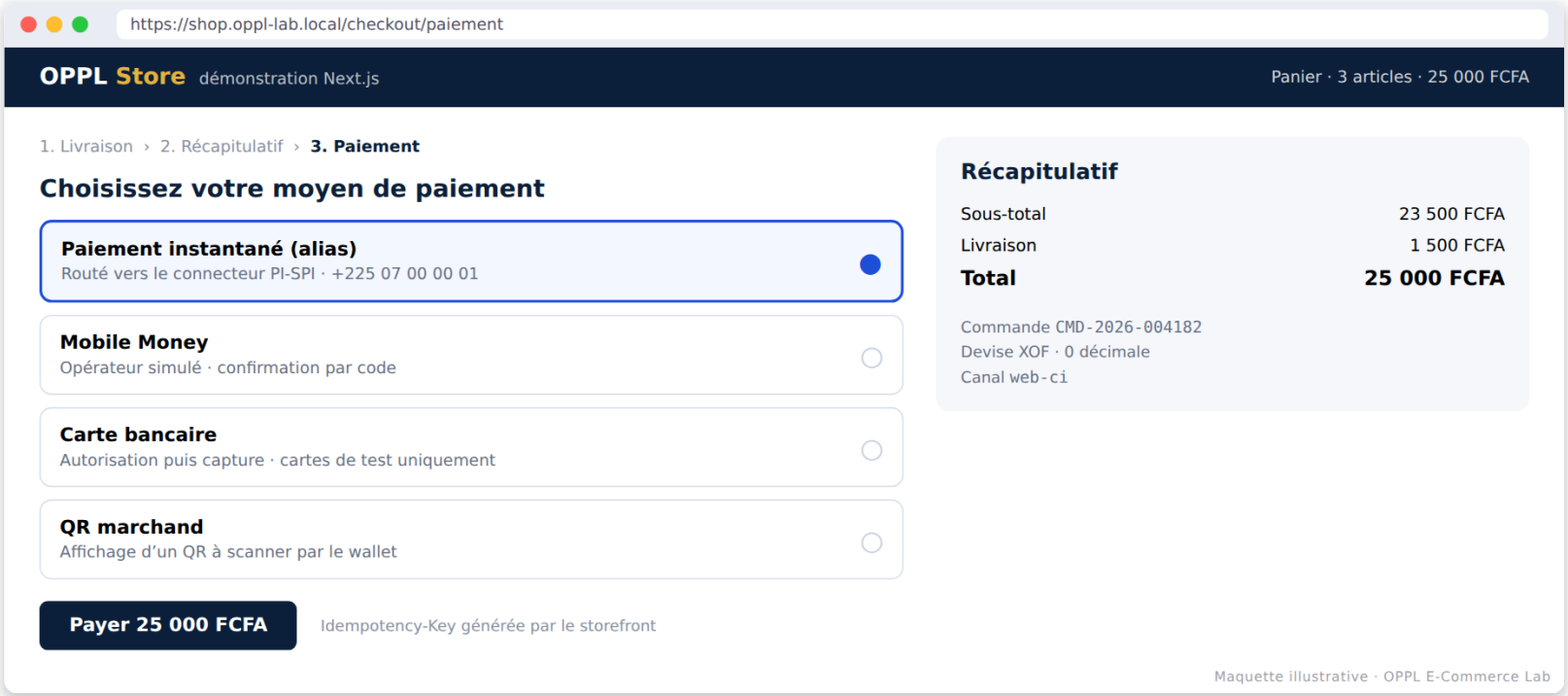
```
// La règle du Lab : entier d'unités mineures + devise
interface Money { minorUnits: number; currency: string; }

const total: Money = { minorUnits: 25000, currency:
'XOF' };  // 25 000 FCFA

// Interdit en revue de code :
// const total = 25000.00;        // flottant
// const total = amount * 100;    // conversion implicite
```

- Aucun **nombre à virgule flottante** dans le code monétaire
- Test dédié dès la phase 4 de déploiement
- Rejet explicite d’un montant décimal en XOF
- Vérifié en revue de code, pas seulement en test

Le point exact où naît un paiement



Capture illustrative — maquette reconstituée, chiffres fictifs

La chaîne complète, après dix phases

lab-admin@oppl-01: ~ — kubectl

```
lab-admin@oppl-01:~$ kubectl get pods -A | grep -E 'oppl-|NAMESPACE'
NAMESPACE        NAME                                  READY   STATUS    RESTARTS   AGE
oppl-iam         keycloak-0                            1/1     Running   0          9d
oppl-gateway     krakend-6f9c8d7b5-x24qz               2/2     Running   0          9d
oppl-store       storefront-next-7d8b6c5f4-l9p2m       1/1     Running   0          5d
oppl-commerce    saleor-api-5b7d4c9f8-9hq2m            1/1     Running   0          6d
oppl-commerce    saleor-worker-6c8d9e7f5-t3k8v         1/1     Running   0          6d
oppl-commerce    saleor-dashboard-79f6c8d54-b7n1x      1/1     Running   0          6d
oppl-commerce    saleor-postgres-0                     1/1     Running   0          6d
oppl-commerce    vendure-server-58c7d9b64-k4r6w        1/1     Running   0          3d
oppl-pay         payment-api-6d9f7c8b5-zx3v1           1/1     Running   0          4d
oppl-pay         payment-router-7b8c9d6f5-pp9w2        1/1     Running   0          4d
oppl-pay         tokenisation-svc-5f4d8c7b6-h8k2n      1/1     Running   0          4d
oppl-pay         fraud-svc-64d9c8b7a-q9w8e             1/1     Running   0          4d
oppl-pay         recon-worker-8e7d6c5b4-c5v4b          1/1     Running   0          4d
oppl-billing     killbill-6b7c8d9e5-r4t7y              1/1     Running   0          4d
oppl-billing     kaui-7c8d9e6f5-f3g4h                  1/1     Running   0          4d
oppl-billing     killbill-mariadb-0                    1/1     Running   0          4d
oppl-connectors  pi-spi-connector-5d9f7c8b6-j2k8m      1/1     Running   0          2d
oppl-connectors  momo-connector-7f6d5c4b3-t6x4r        1/1     Running   0          2d
oppl-connectors  card-connector-9f8e7d6c5-b4n3m        1/1     Running   0          2d
oppl-connectors  qr-connector-6d8b7f5c4-p7r2s          1/1     Running   0          2d
oppl-streaming   oppl-kafka-broker-0                   1/1     Running   0          9d
oppl-streaming   oppl-kafka-broker-1                   1/1     Running   0          9d
oppl-streaming   oppl-kafka-broker-2                   1/1     Running   0          9d
oppl-data        oppl-redis-master-0                   1/1     Running   0          9d
oppl-observ      otel-collector-5b6c7d8e9-a1s2d        1/1     Running   0          9d
oppl-observ      prometheus-server-0                   2/2     Running   0          9d
oppl-observ      grafana-7c8d9e6f5-m2c8p               1/1     Running   0          9d
lab-admin@oppl-01:~$
```

Maquette illustrative · OPPL E-Commerce Lab

Capture illustrative — maquette reconstituée, chiffres fictifs

II

La couche de paiement

Le seul composant entièrement écrit par l'équipe. Tout le reste est assemblé.

Un modèle canonique pour cinq protocoles

```
interface CanonicalPayment {
  paymentId: string;        // PAY-2026-000418
  orderId: string;
  idempotencyKey: string;   // du storefront au
connecteur
  rail: 'INSTANT_ALIAS' | 'MOBILE_MONEY'
      | 'CARD' | 'QR' | 'BANK_TRANSFER';
  amount: Money;            // entier + devise
  customer: { reference, aliasHash, kycLevel };
  merchant: { reference, vendors? };
  status: PaymentStatus;
  connector?: string;
  networkReference?: string;
}
```

- **Sans modèle canonique** : chaque système parle à chaque autre — la complexité croît au carré
- **Avec** : chaque adaptateur ne connaît que le modèle central
- Le moteur de commerce, Kill Bill et les connecteurs deviennent interchangeables
- Changer de moteur ou ajouter un réseau n’impacte qu’un seul adaptateur

Une API de paiement, sept opérations

Endpoint	Méthode	Rôle	Idempotent
/v1/payments	POST	Créer un paiement	Oui — en-tête dédié
/v1/payments/{id}	GET	Consulter l'état courant	—
/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	—
/v1/webhooks/{connector}	POST	Confirmations asynchrones	Oui — par référence réseau

Le moteur de commerce ne connaît qu'une seule « passerelle » : cette API.

LE SUJET DÉCISIF

L’idempotence, à trois niveaux

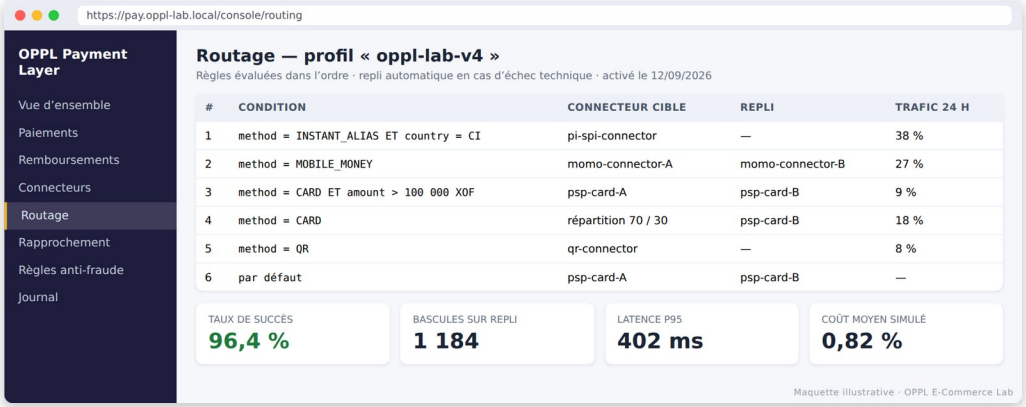
1	**Entrée**	Clé + empreinte du corps, conservées 24 h	Double clic, retentative du navigateur, rejeu réseau
2	**Traitement**	Contrainte d’unicité en base + verrou	Deux requêtes concurrentes qui passent le premier filtre
3	**Sortie**	Référence réseau enregistrée avant l’appel	Double envoi au réseau, notification reçue deux fois

Cas d’école : même clé, montant différent → erreur explicite. Jamais un second paiement, jamais une acceptation silencieuse.

Router sans créer de doublon

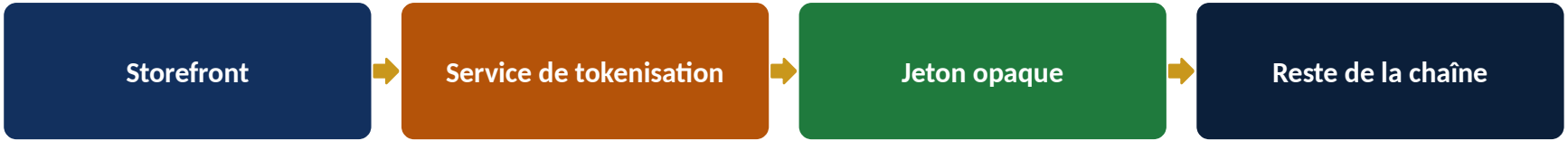
Refus métier
Fonds insuffisants, alias inconnu, carte bloquée.
Résultat définitif → jamais de repli.

Échec technique
Timeout, 5xx, coupure.
Absence de résultat → repli seulement si la première tentative n'a pas abouti.



Capture illustrative — maquette reconstituée, chiffres fictifs

Tokeniser pour réduire le périmètre

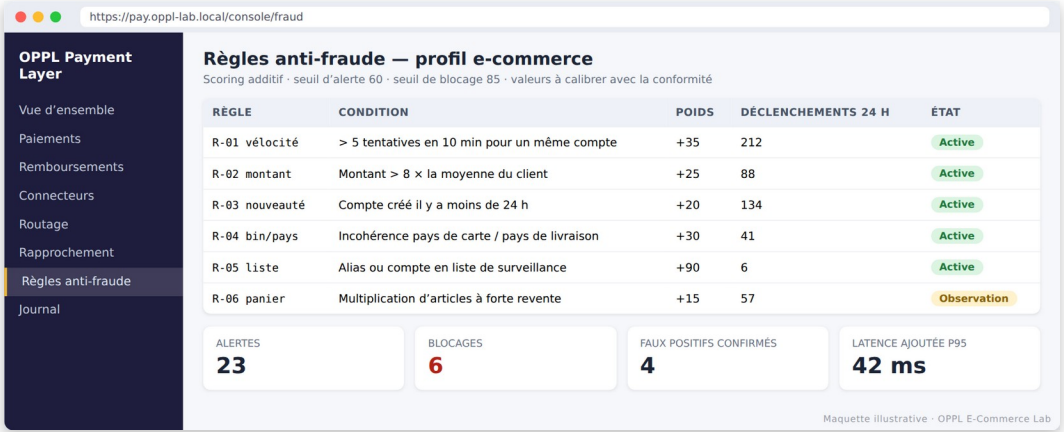


- Le storefront ne transmet **jamais** de numéro de carte au moteur de commerce
- Le jeton est propre au marchand et sans valeur hors du Lab
- Journaux, traces et événements ne contiennent que l'empreinte
- Les alias de paiement circulent sous forme hachée
- Seul le connecteur concerné manipule la donnée réelle, brièvement

Trois bénéfices

1. Surface d'exposition réduite
2. Paiement en un clic et abonnements possibles
3. Périmètre de conformité limité à un seul service

Détecter sans détruire la conversion



Capture illustrative — maquette reconstituée, chiffres fictifs

- Scoring additif, deux seuils : **alerte** et **blocage**
- Les règles sont des **données versionnées**, pas du code
- La conformité propose une évolution par demande revue et tracée
- Trois indicateurs suivis ensemble : détection, faux positifs, latence ajoutée

Bloquer 100 % de la fraude en bloquant 15 % des clients légitimes détruit plus de valeur qu'il n'en protège.

Expliquer chaque écart, du clic à la comptabilité





Capture illustrative — maquette reconstituée, chiffres fictifs

- Paiement sans commande
- Commande sans paiement
- Montant divergent
- Statut divergent
- Règlement absent
- **Doublon — incident majeur**

Le rapprochement à 100 % n'est pas l'objectif. Expliquer chaque écart, si.

III

Commerce et facturation

Brancher le moteur de commerce et le moteur de facturation sans y enfermer le métier du paiement.

Le commerce demande, le paiement décide

L'anti-pattern

- Appels directs aux prestataires depuis un plugin
- Statuts propres au moteur de commerce
- Règles de routage dans le plugin
- Résultat : la chaîne de paiement est **prisonnière** du moteur de commerce

Le modèle du Lab

- Le moteur ne connaît qu'une seule passerelle : l'application de paiement OPPL
- Celle-ci **traduit et délègue**, sans logique métier
- Toute l'intelligence réside dans la couche de paiement
- Résultat : changer de moteur ne coûte qu'un adaptateur

Le webhook synchrone et ses trois contraintes

```
const canonical = toCanonical(payload);   // traduction seule

try {
  const payment = await opplApi.createPayment(canonical, {
    idempotencyKey: payload.idempotencyKey,   // fournie par la
plateforme
    timeoutMs: 2_000,
  });
  return { result: mapResult(payment.status),
           amount: payload.action.amount,
           pspReference: payment.networkReference };
} catch (e) {
  if (e instanceof TimeoutError) {
    // jamais d'échec définitif sur un timeout
    return { result: 'CHARGE_ACTION_REQUIRED', ... };
  }
}
```

- **Le temps de réponse est borné** — budget interne de 2 s, bien en deçà de la limite de la plateforme
- **La réponse est un statut, pas une promesse**
- **La clé d'idempotence vient de la plateforme** et doit être propagée telle quelle

L’initiation d’une transaction

Saleor GraphQL — transactionInitialize

```
dev@oppl-lab:~$ cat mutation.graphql
mutation InitPaiement {
  transactionInitialize(
    id: "Q2hlY2tvdXQ6M2Y5YzFhMmU="            # identifiant du checkout
    paymentGateway: { id: "oppl.app.payments",
                      data: { method: "INSTANT_ALIAS", alias: "+2250700000001" } }
    idempotencyKey: "CMD-2026-004182"
    action: CHARGE
  ) {
    transaction { id  actions  chargedAmount { amount currency } }
    transactionEvent { type  pspReference  message }
    errors { field  code }
  }
}
dev@oppl-lab:~$ curl -s $SALEOR/graphql/ -H "Authorization: Bearer $TOKEN" --data-binary @body.json | jq .data
{
  "transactionInitialize": {
    "transaction": { "id": "VHJhbnNhY3Rpb25JdGVtOjE4NDI=",
                     "actions": ["REFUND", "CANCEL"],
                     "chargedAmount": { "amount": 25000.0, "currency": "XOF" } },
    "transactionEvent": { "type": "CHARGE_SUCCESS",
                          "pspReference": "8f31c47a-6d02-4b19-9e55-0a7412bb03a2",
                          "message": "Paiement instantané confirmé par le connecteur" },
    "errors": []
  }
}
# Saleor a émis le webhook synchrone TRANSACTION_INITIALIZE_SESSION vers oppl-payment-app,
# qui a répondu en 412 ms (limite de la plateforme : 20 s).
```

Maquette illustrative · OPPL E-Commerce Lab

Capture illustrative — maquette reconstituée, chiffres fictifs

Idempotence vue depuis l'application

oppl-payment-app — journal du webhook synchrone

```
10:42:17.104 INFO  TransactionInitializeSession reçu
             merchantReference=CMD-2026-004182  idempotencyKey=CMD-2026-004182
             action={ actionType: CHARGE, amount: 25000, currency: XOF }
             sourceObject=Checkout  customerIpAddress=[masqué]
10:42:17.118 INFO  idempotence : aucune transaction antérieure pour cette clé → traitement
10:42:17.121 INFO  POST /v1/payments → OPPL Payment API (traceparent propagé)
10:42:17.389 INFO  réponse 201 paymentId=PAY-2026-000418 status=CAPTURED
10:42:17.412 INFO  réponse au webhook :
             { "result": "CHARGE_SUCCESS",
               "amount": 25000,
               "pspReference": "8f31c47a-6d02-4b19-9e55-0a7412bb03a2",
               "message": "Paiement instantané confirmé" }
10:44:02.771 INFO  TransactionRefundRequested reçu — montant 25000 XOF
10:44:03.310 INFO  réponse : { "result": "REFUND_SUCCESS", "pspReference": "rfnd-1d0c…" }
10:51:44.020 WARN  clé d'idempotence rejouée (CMD-2026-004182) → réponse mise en cache renvoyée, aucun nouvel appel
```

Maquette illustrative · OPPL E-Commerce Lab

Capture illustrative — maquette reconstituée, chiffres fictifs

Pourquoi un moteur de facturation dédié



Paiements unitaires

Autorisation, capture, annulation, remboursement, avoir



Abonnements

Périodicité, proratisation, changement de plan, essais



Facturation

Factures, soldes, avoirs, consolidation



Relances et impayés

Politique de nouvelles tentatives configurable



Abstraction de prestataire

Un plugin par prestataire, cœur inchangé



Réconciliation intégrée

Correction automatique des états non terminaux

Le plugin traduit, la couche de paiement décide

Méthode du plugin	Opération	Appel OPPL
authorizePayment	Réserver des fonds	POST /payments, capture différée
capturePayment	Encaisser une autorisation	POST /payments/{id}/capture
purchasePayment	Autorisation + capture	POST /payments, capture immédiate
voidPayment	Annuler avant capture	POST /payments/{id}/cancel
refundPayment	Rembourser	POST /payments/{id}/refunds
getPaymentInfo	Consulter l'état réel	GET /payments/{id} — clé de la réconciliation

Le piège du statut « indéfini » C'est le seul que le moteur ne peut pas rejouer. Un bon plugin préfère le statut d'attente, que la réconciliation interne corrigera.

Le rattrapage automatique en action

killbill-oppl-connector — appels du PaymentPluginApi

```
ops@oppl-lab:~$ kubectl -n oppl-billing logs -f deploy/killbill | grep oppl-connector

10:42:17 INFO  purchasePayment  kbPaymentId=4b2e…c081 kbTransactionId=a1f7…3d
                    amount=25000 currency=XOF properties=[method=INSTANT_ALIAS]
10:42:17 INFO  → POST /v1/payments (OPPL Payment API)  idempotencyKey=a1f7…3d
10:42:17 INFO  ← 201 CAPTURED  paymentId=PAY-2026-000418
10:42:17 INFO  retour PaymentPluginStatus.PROCESSED  gatewayReference=PAY-2026-000418

10:43:05 INFO  authorizePayment kbPaymentId=91da…7f42 amount=78000
10:43:06 INFO  retour PROCESSED — fonds réservés, capture différée
10:43:41 INFO  capturePayment   kbPaymentId=91da…7f42 → PROCESSED

10:44:02 INFO  refundPayment    kbPaymentId=55f3…9b77 amount=45000 → PROCESSED

10:45:12 WARN  purchasePayment  timeout connecteur Mobile Money (5 s)
10:45:12 WARN  retour PaymentPluginStatus.PENDING — surtout pas UNDEFINED : le Janitor pourra réconcilier
10:50:12 INFO  getPaymentInfo   (appel du Janitor) → PROCESSED — état corrigé dans Kill Bill
```

Maquette illustrative · OPPL E-Commerce Lab

Capture illustrative — maquette reconstituée, chiffres fictifs

IV

Les connecteurs

Un cadre commun, des composants minces, une méthode industrialisable.

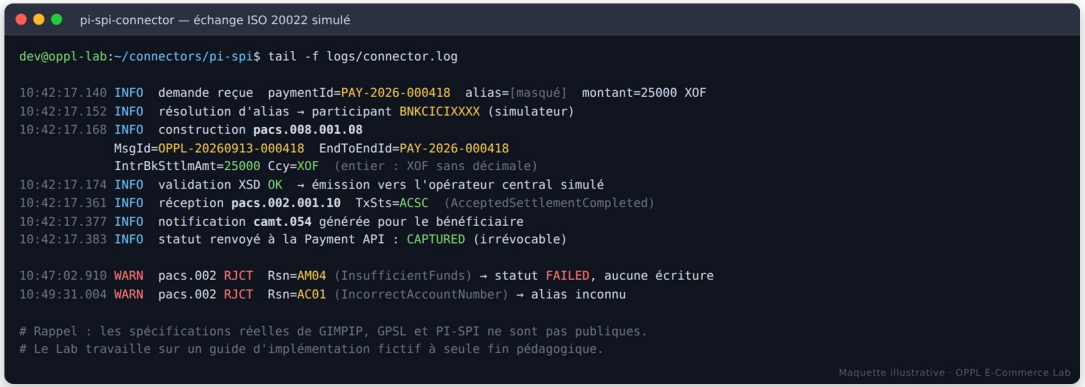
Un connecteur est délibérément mince

```
interface PaymentConnector {
  execute(p, ctx): Promise<ConnectorResult>;
  inquire(networkRef): Promise<ConnectorResult>;
  refund(p, amount, ctx): Promise<ConnectorResult>;
  handleWebhook(raw): Promise<ConnectorResult |
null>;
}
```

- Le connecteur **traduit et transporte** — rien de plus
- Le mapping des codes réseau se fait dans la couche de paiement
- Un connecteur certifié est interchangeable

Six exigences	Vérifiée par
Idempotence sur la clé fournie	Double appel, un seul effet
Méthode d'interrogation implémentée	Timeout simulé, état retrouvé
Aucune donnée sensible journalisée	Analyse statique + revue
Timeout explicite sur chaque appel	Revue de configuration
Tolérance aux notifications en double	Webhook envoyé deux fois
Codes réseau conservés tels quels	Revue de code

Connecteur paiement instantané



Capture illustrative — maquette reconstituée, chiffres fictifs

Message	Rôle
pacs.008	Instruction de paiement
pacs.002	Acceptation ou rejet motivé
camt.054	Notification du bénéficiaire
camt.056	Rappel de fonds (rarement garanti)

Le Lab travaille sur un guide d'implémentation fictif, construit à partir des standards génériques. Toute intégration réelle relève des processus d'homologation des opérateurs.

Connecteur Mobile Money

momo-connector — confirmation asynchrone et idempotence

```
dev@oppl-lab:~/connectors/momo$ tail -f logs/connector.log

10:45:07.221 INFO  POST /collect  paymentId=PAY-2026-000431  montant=12500 XOF
10:45:07.402 INFO  réponse opérateur simulé : 202 Accepted  ref=MM-77ba0e
10:45:07.410 INFO  statut intermédiaire → ACTION_REQUIRED (le client saisit son code)
10:45:07.415 INFO  publication oppl.payments.pending.v1 (clé = paymentId)

10:45:39.882 INFO  webhook reçu de l'opérateur : ref=MM-77ba0e status=SUCCESS
10:45:39.884 INFO  vérification de signature HMAC OK
10:45:39.901 INFO  statut final CAPTURED → notification à Kill Bill et à Saleor

10:45:41.117 WARN  webhook reçu en double : ref=MM-77ba0e
10:45:41.118 INFO  déjà traité → 200 OK renvoyé, aucun effet de bord (idempotence)

10:52:10.334 ERROR timeout opérateur (30 s) sur ref=MM-812fa1
10:52:10.336 INFO  statut PENDING conservé — réconciliation par interrogation planifiée
11:05:00.002 INFO  interrogation planifiée : ref=MM-812fa1 → FAILED, commande libérée
```

Maquette illustrative · OPPL E-Commerce Lab

Capture illustrative — maquette reconstituée, chiffres fictifs

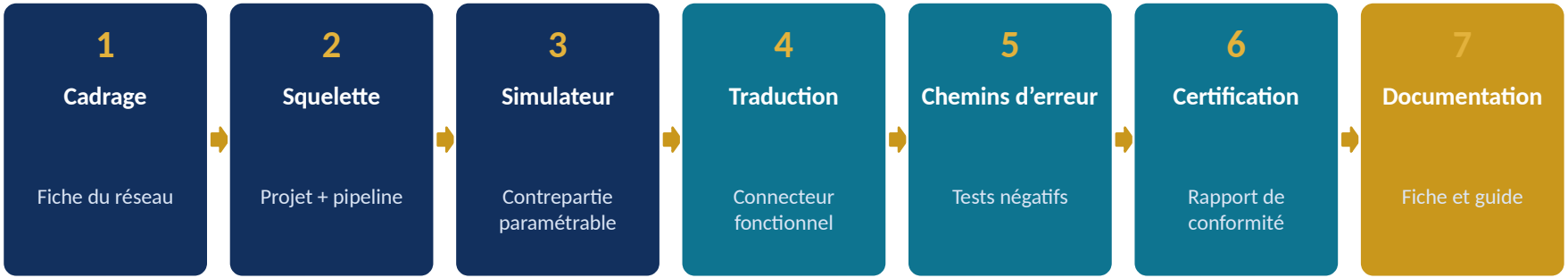
1. L'opérateur accepte sans confirmer
2. Le client saisit son code
3. La notification arrive, signée
4. Le doublon est reconnu et ignoré
5. Sans notification : interrogation planifiée

Connecteur carte : le cycle complet

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

Le connecteur carte est aussi le seul à manipuler — brièvement — des données d'instrument.

L’usine à connecteurs, en sept étapes



Le point de méthode contre-intuitif On écrit le simulateur de la contrepartie AVANT le connecteur. Sans lui, il est impossible de tester les chemins d’erreur — c’est-à-dire tout ce qui compte vraiment en production.

V

Former et tester

Treize labs, un catalogue de tests, un référentiel de compétences.

Treize labs, un même cycle

1. Checkout
2. Payment Connector
3. Card
4. QR Payment
5. PI-SPI
6. Payment Routing
7. Failure
8. Fraud
9. Refund
10. Reconciliation
11. Subscription
12. Marketplace
13. Observability

Socle obligatoire (1 à 6) : lire une chaîne, écrire un connecteur, maîtriser un protocole, router sans doublon.

Progression sur douze semaines

Semaine	Labs	Compétence acquise
S1–S2	Checkout, Observability	Lire une chaîne de paiement et la suivre de bout en bout
S3–S4	Connector, Card	Écrire un connecteur et maîtriser le cycle monétique
S5–S6	QR, PI-SPI	Manipuler des protocoles et des messages normalisés
S7–S8	Routing, Failure	Router sans doublon, garantir l'absence de perte sous panne
S9–S10	Fraud, Refund, Reconciliation	Arbitrer protection / conversion, expliquer chaque écart
S11–S12	Subscription, Marketplace, restitution	Traiter les cas sans client présent, exploiter et transmettre

Dix compétences évaluées, chacune adossée à une preuve produite par l'apprenant.

Casser volontairement, puis mesurer

Failure Lab — campagne k6 avec injection de pannes

```
perf@oppl-lab:~/labs/failure$ k6 run --vus 150 --duration 15m scenarios/checkout-mixed.js

  scenarios: (100.00%) 1 scénario, 150 VUs max, durée 15m30s max
  injection de pannes : connecteur carte indisponible de T+5min à T+8min

     ✓ checkout crée une transaction ..................  99.62%  ✓ 178 204  ✗ 682
     ✓ clé d'idempotence respectée (0 doublon) .......  100.00%
     ✓ bascule vers le connecteur de repli ...........  100.00%  1 184 bascules
     ✗ aucune commande bloquée en PENDING ............  99.81%  34 commandes

     http_req_duration...: avg=214ms med=168ms p(90)=389ms p(95)=498ms p(99)=1.24s
     http_reqs...........: 178 886  198.8/s
     ✓ seuil p(95) < 600 ms ......................... ATTEINT
     ✗ seuil commandes bloquées = 0 ................. NON ATTEINT (34)

# Analyse : les 34 commandes bloquées correspondent à des webhooks Mobile Money reçus
# pendant la panne. Correctif : file d'attente de reprise + interrogation planifiée.
```

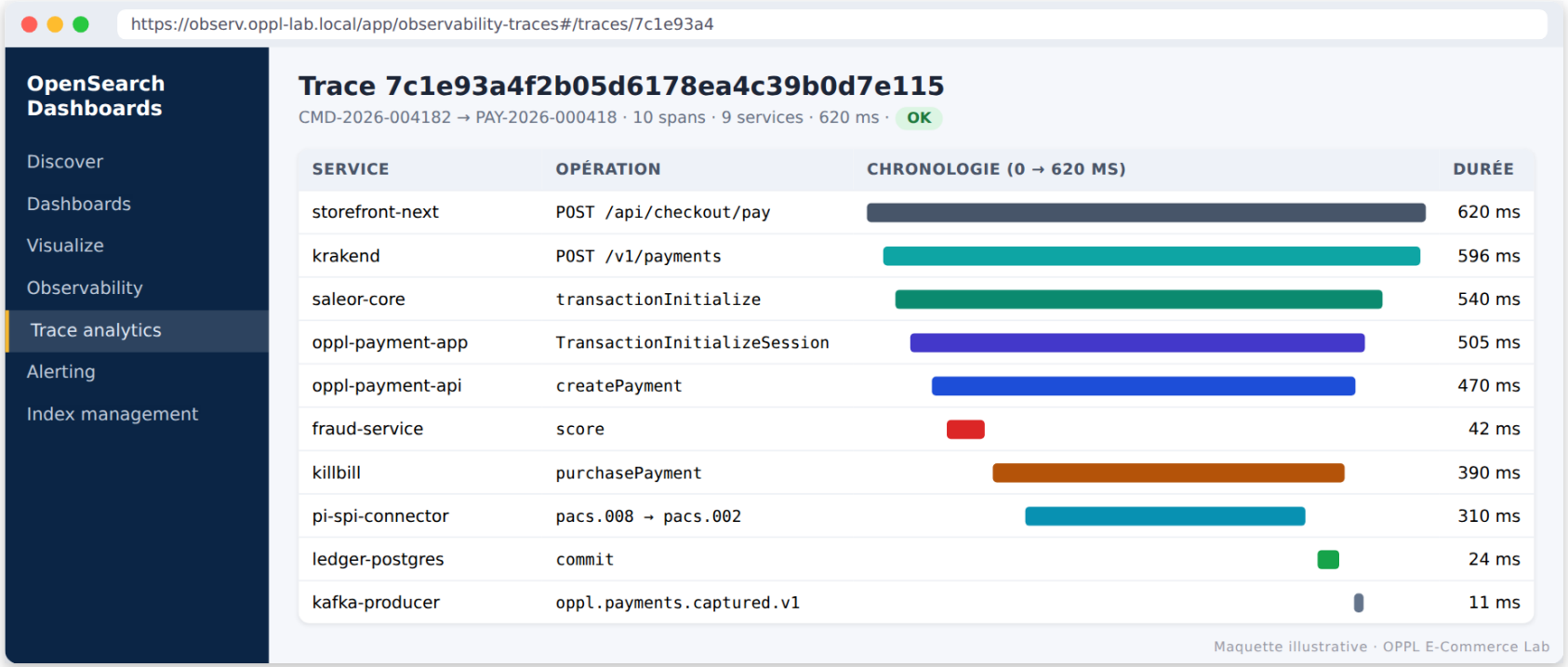
Maquette illustrative · OPPL E-Commerce Lab

Capture illustrative — maquette reconstituée, chiffres fictifs

- Connecteur coupé 3 minutes en pleine campagne
- Bascule vers le repli : **1 184 fois, sans doublon**
- Mais 34 commandes bloquées en attente
- Correctif : file de reprise + interrogation planifiée
- Puis nouvelle mesure — le cycle recommence

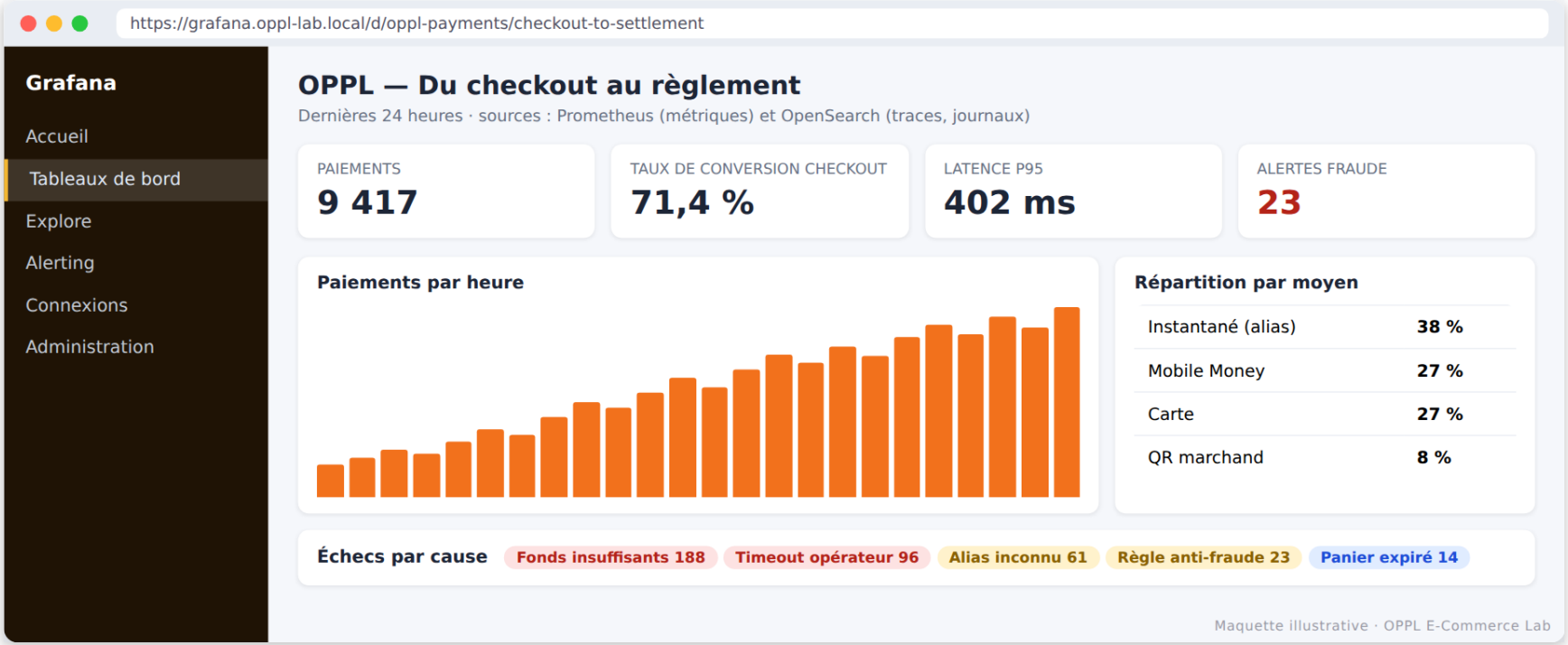
Un seul seuil ne tolère aucune exception : zéro doublon.

Retrouver n'importe quelle commande en une minute



Capture illustrative — maquette reconstituée, chiffres fictifs

Voir la chaîne vivre



Capture illustrative — maquette reconstituée, chiffres fictifs

Sécurité par défaut et périmètre maîtrisé

Identité

Jeton pour tout appel, privilèges minimaux

Réseau

Interdiction par défaut, chiffrement partout

Données

Synthétiques, tokenisées, masquées

Chaîne logicielle

Inventaire, analyse, signature

Conformité carte

Réduire le périmètre, pas le subir

Cadre régional

Références à valider par la conformité

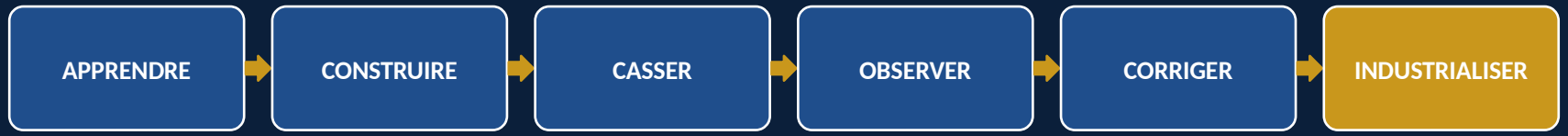
De la formation aux offres de service

Offre	Contenu	Bénéficiaires
Payment Engineering Academy	Parcours de 12 semaines, 13 labs, évaluation	Jeunes diplômés, équipes internes
Payment Connector Factory	Production industrialisée de connecteurs	Banques, fintechs, PSP, agrégateurs
Payment Sandbox mutualisé	Environnement de test partagé et instrumenté	E-commerçants, éditeurs, intégrateurs
Payment Integration Testing	Campagnes d'intégration, charge et résilience	Partenaires en phase d'intégration
E-Commerce Certification Lab	Conformité d'une intégration au cadre du Lab	Marchands et prestataires

Précision indispensable La « certification » du Lab est une conformité à son propre cadre technique. Elle n'a aucune valeur réglementaire et ne remplace ni homologation, ni certification de schéma, ni agrément.

Compétences · Innovation · Impact

L’objectif n’est pas de construire une boutique de plus. C’est de former ceux qui construiront les paiements de demain.



Des paiements plus ouverts, une Afrique plus connectée.

Lionel SAKITI · Professionnel des infrastructures numériques · assisté par IA, gouverné par ISO/IEC 42001