Grille d’auto-évaluation

Partie 3 : Labos 6 et 7 — LOG430

Niveaux d’évaluation

Chaque critère est évalué selon l’un des quatre niveaux suivants :

— Excellent : Travail dépassant les attentes, complet et justifié.

— Suffisant : Répond aux attentes avec des justifications claires.

— Suffisant avec améliorations : Efforts visibles, mais lacunes à combler.

— Insuffisant : Critère absent ou mal exécuté.

Grille d’évaluation détaillée

Critère	Excellent	Suffisant	Suffisant avec améliorations	Insuffisant
1. Définition du scénario métier	Processus métier en moins que 4 étapes, diagramme et README complet	Scénario clair, étapes identifiées	Scénario partiel, diagramme manquant	Scénario absent ou incomplet
2. Saga orchestrée (Labo 6)	Orchestrateur central fonctionnel, appels synchrones complets	Orchestrateur implémenté, flux basique	Orchestrateur partiel, flux incomplet	Pas de saga orchestrée
3. Machine d’état (Labo 6)	Diagramme + code clair, persistance de l’état et logs complets	Diagramme fourni et persistance basique	Modèle partiel, persistance manquante	Pas demachine d’état
4. Gestion des échecs & compensation (Labo 6)	Rollback/compensation documentés, tests d’échec inclus	Compensation implémentée, tests manquants	Gestion d’erreurs minimale	Pas de gestion d’échecs
5. Simulation de cas d’échec (Labo 6)	Tests automatisés et robustesse	Quelques scripts d’échec ou scénarios manuels	Tests partiels, non automatisés	Aucun test d’échec

6. Producteurs d'événements (Labo 7)	Événements JSON horodatés, ID unique, retry + DLQ	Événements corrects, sans replay ni DLQ	Producteurs partiels, schéma incomplet	Pas de producteurs

Critère	Excellent	Suffisant avec améliorations	Insuffisant	Suffisant
7. Topics/Queues & schéma (Labo 7)	Nommage clair, versionnement, gestion de schéma	Nommage cohérent, schéma de base	Organisation confuse, pas de version	Pas de topics/queues
8. Consommateurs idempotents (Labo 7)	Plus que 2 consommateurs, idempotence et logs JSON structurés	1 consommateur ou idempotence partielle	Consommateurs minimaux, sans idempotence	Pas de consommateurs
9. Event Store & replay (Labo 7)	Stockage complet, endpoint de replay, reconstruction OK	Stockage et replay basique	Stockage sans replay	Pas d'Event Store
10. CQRS & read models (Labo 7)	Séparation claire Command/Query, projections fonctionnelles	CQRS partiel, read models limités	Read models manquants	Pas de CQRS

11. Observabilité & traçabilité	Prometheus exposé, Grafana, trace & corrélation IDs	Métriques ou logs structurés seuls	Observabilité limitée	Pas d'observabilité
12. Tests automatisés & e2e	CI complète, tests bout à bout, scénarios échec/replay	Tests unitaires + quelques e2e	Tests manuels, pas d'automatisation	Pas de tests
Critère	Excellent	Suffisant	Suffisant avec améliorations	Insuffisant
13. Documentation & ADR	README détaillé, guide de déploiement, plus de 2 ADR	README et rapport succincts	Documentation partielle, ADR manquants	Pas de documentation
14. Structure du dépôt & CI/CD	Répertoires lab6/, lab7/, pipelines CI/CD, manifests k8s	Organisation de base, CI partielle	Structure confuse, CI manquante	Pas de CI/CD
15. Saga chorégraphiée (Labo 7)	Flux d'événements complet, coordination sans orchestrateur, diagramme clair	Saga implémentée, coordination partielle	Saga partielle, non testée	Pas de saga

16. Événements compensatoires (Labo 7)	Événements de compensation robustes, testés et idempotents	Présents mais sans test automatisé	Schéma incomplet ou non testé	Aucun événement compensatoire

Autoévaluation à compléter par l'étudiant(e)

Nom : Maksym Pravdin Date : 25 juillet 2025

Pour chaque critère, cochez la case correspondant à votre niveau et ajoutez un court commentaire :

Critère 1. Définition du scénario métier

(✓) Excellent (□) Suffisant (□) À améliorer Comr(□) Insuffisant

Le scénario métier choisi est le traitement des commandes en ligne. Cela implique les services auth, stocks et sales (et orchestr-sales). Voici les étapes de la saga : 1) Récupérer le panier du client avec le service stocks. 2) Enregistrer la vente avec le service sales 3) Mettre à jour l'inventaire avec le service stocks 4) Augmenter la réputation/rank de l'utilisateur. Plus d'informations sont dans le rapport et dans le fichier README

Critère 2. Saga orchestrée (Labo 6)

(✓) Excellent (□) Suffisant (□) À améliorer Comr(□) Insuffisant

Un service orchestrateur est ajouté au système : orchestr-sales. Celui-ci s'occupe d'orchestrer toute la saga avec des étapes définies dans le code source. Pour chaque étape, il vérifie si l'étape s'est complétée avec succès et sinon, il fait des actions compensatoires s'ils sont nécessaires (sauf pour la 1ere étape par exemple, car c'est le début)

Critère 3. Machine d'état (Labo 6)

(□) Excellent (✓) Suffisant (□) À améliorer Comr(□) Insuffisant

Orchestr-sales possède son propre container pour la persistence qui est mongo-events qui est un container mongoDB pour garder les états de chaque saga qui se met à jour. Reste quand même assez basique sous cette forme :

{
  _id: ObjectId('687c027b6fd1392e87664f1c'),
  name: 'Commande de test magasin StockCentral',
  type: 'CommandeConfirmee',
  details: 'Étape 4 OK. Vente enregistrée avec 1 produits et inventaire mis à jour pour test.',
  date: ISODate('2025-07-19T20:39:23.541Z'),
  __v: 0
 },

Critère 4. Gestion des échecs & compensation (Labo 6)

(✓) Excellent (□) Suffisant (□) À améliorer Comr(□) Insuffisant

Pour les étapes intermédiaires de la saga, il y a des actions compensatoires dans le cas qu'une étape échoue comme par exemple si le service stocks tombe en panne après que la vente soit

enregistrée, celle-ci est automatiquement supprimée de sa bd une fois que la 3e étape échoue et l'état de la saga est mis à jour pour indiquer qu'un problème est survenu.

Critère 5. Simulation de cas d'échec (Labo 6)

(□) Excellent (✓) Suffisant (□) À améliorer Comr(□) Insuffisant

La simulation des cas d'échecs était testé avec quelques tests automatisées, mais qui ne couvrent pas tous les cas possibles d'échecs + testé avec des scénarios manuels.

Critère 6. Producteurs d'événements (Labo 7)

(✓) Excellent (□) Suffisant (□) À améliorer Comr(□) Insuffisant

RabbitMQ est utilisé comme système de messagerie. Ainsi, il est possible d'attribuer 2 types de id aux événements (id unique et idaggregate qui serait partageable entre différents autres services pour pouvoir se retrouver). Ces événements sont publiés sur une chaine « permanente » qui permet de les recevoir et les ditribuer à ces consommateurs sous forme de JSON avec toutes les informations nécessaires.

Critère 7. Topics/Queues & schéma (Labo 7)

(□) Excellent (✓) Suffisant (□) À améliorer Comr(□) Insuffisant

Un diagramme était inclus dans le rapport pour représenter la manière que la saga chorégraphiée fonctionne. Dans le rapport et dans le README et dans le code, il y a plus d'informations sur les états des événements et sur les destinations de ces événements.

Critère 8. Consommateurs idempotents (Labo 7)

(✓) Excellent (□) Suffisant (□) À améliorer Comr(□) Insuffisant

Les consommateurs des événements sont : Audit (qui stocke les logs de TOUS les événements dans sa base de données propre à lui). Notif (qui sert de service de notification simulé, qui une fois qu'il consomme un évènement va l'afficher mais ne possède pas sa propre base de données pour la persistance). Supplies-Event-Store (qui ressemble à Audit, mais conçu uniquement pour les évènements des demandes de réapprovisionnement et possède des informations plus spécifiques qu'Audit.) Supplies-Query (qui est la service « Query » du CQRS et qui possède sa propre base de données avec les projections des évènements pour les garder à jour avec les vrais instances de demandes de réapprovisionnement.)

Critère 9. Event Store & replay (Labo 7)

(✓) Excellent (□) Suffisant (□) À améliorer Comr(□) Insuffisant

Le replay permet de reconstruire une projection d'un évènement en particulier à partir du event-store comme output. Par la suite, supplies-query va supprimer l'instance courante de l'évènement avec le même aggregateId pour reconstruire l'état du replay.

Critère 10. CQRS & read models (Labo 7)

(✓) Excellent (✓) Suffisant (□) À améliorer Comr(□) Insuffisant

J'ai mis Excellent et Suffisant, car mon implémentation est une combinaison des 2 évaluations. Consulter mon ADR #10 pour plus d'informations sur le choix d'utiliser le service existant Supplies comme « Command ». Aussi, Supplies-Query est le service qui fait la gestion « Query » de CQRS et utilise des Read Models pour le traitement de CQRS.

Critère 11. Observabilité & traçabilité

(✓) Excellent (□) Suffisant (□) À améliorer Comr(□) Insuffisant

Des métriques custom Prometheus sont ajoutées au producer donc service supplies et aux services des consommateurs comme audit, notif, supplies-query et supplies-event-store pour suivre l'état de la saga chorégraphié et les événements envoyés.
NOTE : Pour le labo 6, j'ai fait la Visualisation l'évolution des états via Grafana avant que ceci soit retirée et j'ai mis les preuves de Grafana sous forme de captures d'écran ou en consultant le dashboard directement

Critère 12. Tests automatisés & e2e

(□) Excellent (✓) Suffisant (□) À améliorer Comr(□) Insuffisant

Des tests unitaires sont faits pour tous les services ajoutés pour les labos 6 et 7 mais ceux -ci ne couvrent pas forcément tous les cas de tests possibles. Ces tests sont intégrés comme des jobs isolés dans le pipeline CI/CD aussi.

Critère 13. Documentation & ADR

(✓) Excellent (□) Suffisant (□) À améliorer (□) Insuffisant Commentaire :

Le README était mis à jour pour le labo 6 et 7. 2 ADRs sont faits pour la labo 6 et 2 ADRs sont fait pour la labo 7 et sont inclus à la fin du rapport. Des instructions de déploiement sont inclus dans le README aussi.

Critère 14. Structure du dépôt & CI/CD

(✓) Excellent (✓) Suffisant (□) À améliorer (□) Insuffisant Commentaire :

Pipeline CI/CD fonctionnel avec toutes les étapes demandées qui peut être visualisé dans le repo GitHub ou dans le rapport ou dans le README. Des tags sont utilisés pour séparer chaque étape du laboratoire pour avoir un accès rapide aux version antérieures. La seule chose qui n'est pas ajouté pour atteindre Excellent est le manifest k8s, uniquement des fichiers docker-compose sont inclus dans le repo (1 pour l'ensemble du système et un pour chaque service pour lancer une partie de l'application).

Critère 15. Saga chorégraphiée (Labo 7)

(✓) Excellent (□) Suffisant (□) À améliorer (□) Insuffisant Commentaire :

Il n'y a pas de service orchestrateur pour la saga chorégraphiée. Le flux des événements est décrit dans le rapport et possède des diagrammes pour expliquer la séquence des événements et la manière que les services se communiquent entre eux.

Critère 16. Événements compensatoires (Labo 7)

(□) Excellent (✓) Suffisant (□) À améliorer (□) Insuffisant Commentaire :
Des tests automatisés sont faits pour la grande partie des états, mais pas son entièreté. Pour les évènements de compensations, les chaines permanentes de RabbitMQ permettent aux consommateurs « tardifs » ou qui n'ont pas pu être en ligne lorsque l'évènement était publié de se « rattraper » et de se mettre à jour. Il y a des détection du côté du service supplies pour que lorsque celui-ci publie un événement si des consommateurs sont down ou si RabbitMQ lui-même est down.