Exercice 7 : Application RESFUL avec LARAVEL et ETL avec WSO2/Apache kafka

Membres du groupe 4 MSI

DJIGUEMDE Arold

MOUNKORO Séraphin

RABO Souleymane

SEHOU Emmanuel

ZONGO Ramzi

Table des matières

Cahier de Tests – Application RESTful Laravel avec ETL

NB : Pour exécuter le projet, lire le fichier README.md qui se trouve dans le répertoire racine.

1. Objectif du cahier de tests

Ce cahier de tests décrit les scénarios permettant de vérifier :

- Le bon fonctionnement de l'API RESTful (CRUD Clients)
- Le fonctionnement du moteur ETL
- Le traitement via la file d'attente (Queue Worker)
- La cohérence et l'intégrité des données entre les bases source et cible
- La conformité avec les règles métier (validation, filtres, pagination)
- Le bon comportement de l'application dans divers cas limites (edge cases)

2. Périmètre des tests

A. API RESTful

- Création, lecture, mise à jour, suppression de clients
- Recherche et filtrage
- Pagination

B. ETL

- Extraction depuis la base source
- Transformation (normalisation, nettoyage, format)
- Chargement via la queue
- Mise à jour de la base cible

C. Queue / Jobs

- Execution normale
- Gestion des erreurs
- Retry / tries

D. Base de données

- Intégrité des données
- Contraintes de validation

3. **Pré-requis techniques**

- PHP 8.1+
- Composer
- Laravel 10+
- MySQL pour les BD laravel_source et laravel_target
- Queue configurée en database
- Trois terminaux nécessaires :
    - Serveur Laravel
    - Queue Worker
    - Commandes/tests

4. **Scénarios de Test**

TEST 1 : Démarrage du serveur Laravel

Objectif

Vérifier que l'application démarre sans erreur.

Pré-conditions

Dépendances installées, .env configuré.

Étapes

1. Lancer php artisan serve
2. Ouvrir http://localhost:8000

Résultat attendu

✓ La page d'accueil Laravel s'affiche correctement
✓ Aucun message d'erreur dans le terminal

TEST 2 : Démarrage du Queue Worker

Objectif

Vérifier que la queue fonctionne.

Étapes

1. Lancer php artisan queue:work

Résultat attendu

✓ Le worker est en écoute
✓ Les jobs futurs s'exécuteront sans erreur

SECTION API — Tests CRUD Clients

TEST 3 : Création d'un client

Endpoint

POST /api/v1/clients

Données envoyées

```
{
  "nom": "Test",
  "prenom": "User",
  "email": "test@example.com",
  "telephone": "0612345678",
  "ville": "Paris",
  "statut": "actif"
}
```

Résultat attendu

✓ Retour HTTP 201 Created
✓ Le client est inséré en BD cible
✓ Le retour contient :

- data.id
- les données envoyées

TEST 4 : Récupération de tous les clients

Endpoint

GET /api/v1/clients

Résultat attendu

✓ Statut HTTP 200 OK
✓ Objet JSON avec :

- data.data (liste paginée)

- data.total
  ✓ Le client créé précédemment est présent

TEST 5 : Récupération d'un client par ID

Endpoint

GET /api/v1/clients/{id}

Résultat attendu

✓ Statut 200 OK
✓ Retour contenant les informations du client demandé

TEST 6 : Mise à jour d'un client

Endpoint

PUT /api/v1/clients/{id}

Données mises à jour

```
{
  "telephone": "0700000000",
  "statut": "inactif"
}
```

Résultat attendu

✓ Statut 200 OK
✓ Les champs sont mis à jour en BD
✓ Retour contient les valeurs modifiées

TEST 7 : Suppression d'un client

Endpoint

DELETE /api/v1/clients/{id}

Résultat attendu

✓ Statut 200 OK ou 204 No Content
✓ Le client n'apparaît plus dans la liste
✓ Récupération de cet Id renvoie 404

TEST 8 : Recherche d'un client

Endpoint

GET /api/v1/clients/search?q=Test

Résultat attendu

✓ Statut 200 OK
✓ Liste filtrée
✓ total >= 1

TEST 9 : Filtrage par statut

Endpoint

GET /api/v1/clients?statut=actif

Résultat attendu

✓ Uniquement des clients actifs dans les résultats

TEST 10 : Pagination

Endpoint

GET /api/v1/clients?per_page=5

Résultat attendu

✓ 5 clients maximum par page
✓ Présence de links et meta

SECTION ETL — Tests du processus ETL

TEST 11 : Insertion de données dans la base source

Méthode

Via Tinker.

Résultat attendu

✓ Deux clients insérés dans laravel_source

TEST 12 : Exécution du processus ETL

Commande

php artisan etl:run

Résultat attendu

✓ Extraction OK
✓ Transformation appliquée

- Nom en MAJUSCULES
- Prénom capitalisé
- Téléphone normalisé
  ✓ Chargement dans la queue
  ✓ Le Queue Worker exécute les jobs

TEST 13 : Vérification de la transformation

Vérifier que :

Champ	Transformation	Résultat attendu
nom	MAJUSCULE	DUPONT
prenom	ucfirst	Jean
telephone	normalisé	format cohérent
ville	string	inchangé

TEST 14 : Vérifier le chargement en BD cible

Résultat attendu

✓ Les entrées de la source sont présentes dans la cible
✓ Les transformations sont appliquées
✓ Aucun doublon sauf si stratégie prévue

TEST 15 : Gestion des erreurs ETL

Tester un cas volontairement erroné

Insérer un client **sans email** dans la base source.

Résultat attendu

✓ Le job échoue proprement
✓ Message dans les logs
✓ La queue continue à traiter les autres jobs

SECTION QUEUE — Tests Worker/Jobs

TEST 16 : Lancer un job manuellement

Commande

php artisan queue:work

Résultat attendu

✓ Le job apparaît comme exécuté
✓ Le worker ne plante pas

TEST 17 : Test des retries

Simuler une erreur (email manquant).

Résultat attendu

✓ Le job est retenté --tries=3
✓ Puis marqué comme failed
✓ Inscrit dans failed_jobs

TEST 18 : Vérification de la table jobs

Étapes

1. Exécuter ETL
2. Vérifier jobs et failed_jobs

Résultat attendu

✓ Tous les jobs sont traités
✓ Table jobs vide après traitement

5. **Résultats de tests — Table de synthèse**

Test	**Résultat attendu**
Démarrage serveur	OK
Queue Worker	OK
CRUD Create	201
CRUD Read	200
CRUD Update	200
CRUD Delete	200
Recherche	Filtrage OK
ETL Extract	OK
ETL Transform	OK
ETL Load	OK via queue
Worker Retry	OK

(Coche les cases après exécution)

6. **Démarrer les services**

Exécuter la commande docker-compose up -d

7. **Accéder aux services**

- **Prometheus**: http://localhost:9090
- **kakfa**: http://localhost:9092
- **Dashboard Laravel**: http://localhost:8000/dashboard.html

8. **Conclusion**

Ce cahier de tests permet de couvrir l'ensemble des fonctionnalités prévues dans l'application.

Il inclut des scénarios dédiés à l'API, au processus ETL, à la gestion des files d'attente, à l'architecture applicative ainsi qu'aux interactions avec les bases de données source et cible. Il prend en compte à la fois les scénarios standards, les cas d'erreur et les situations limites afin d'assurer une validation complète du système.