

Rapport du Projet Groupe 5

Thème : Développer une application RESTful en utilisant Spring Boot qui utilise Talend Open Studio pour se connecter à une base de données MySQL/PostgreSQL, avec monitoring via Prometheus.

Sommaire

- I. Talend Open Studio for data intégration / ETL
- II. Application RESTful avec Springboot
- III. Métriques Prometheus intégré

I. Talend Open Studio for data intégration / ETL

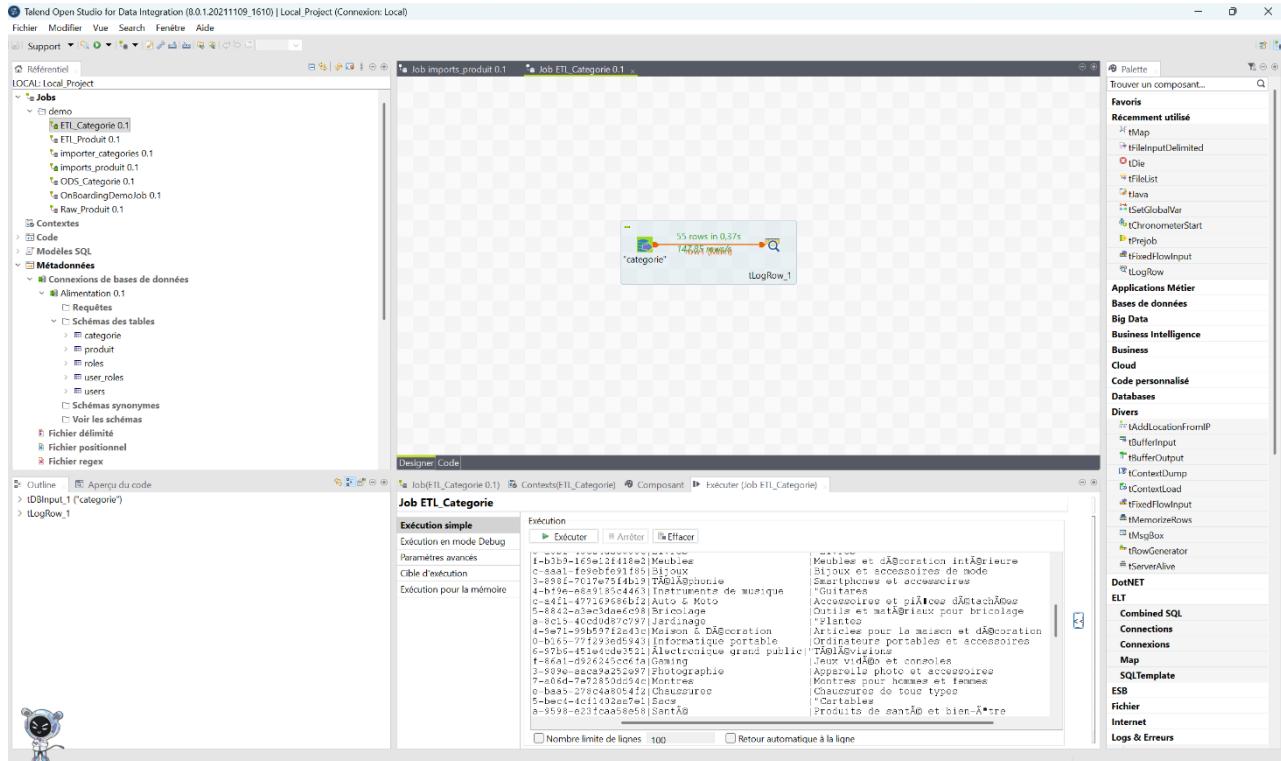
Introduction

Talend Open Studio est un **outil d'intégration de données (data integration / ETL)**. ETL signifie **Extract–Transform–Load** (Extraction, Transformation, Chargement) : l'idée est de **récupérer des données depuis des sources diverses, les nettoyer / transformer / enrichir**, puis les **charger** vers une base de données, un entrepôt de données (data Warehouse), un fichier, un service, etc.

Pour notre application nous avons créer 4 ETL pour la gestion des données.

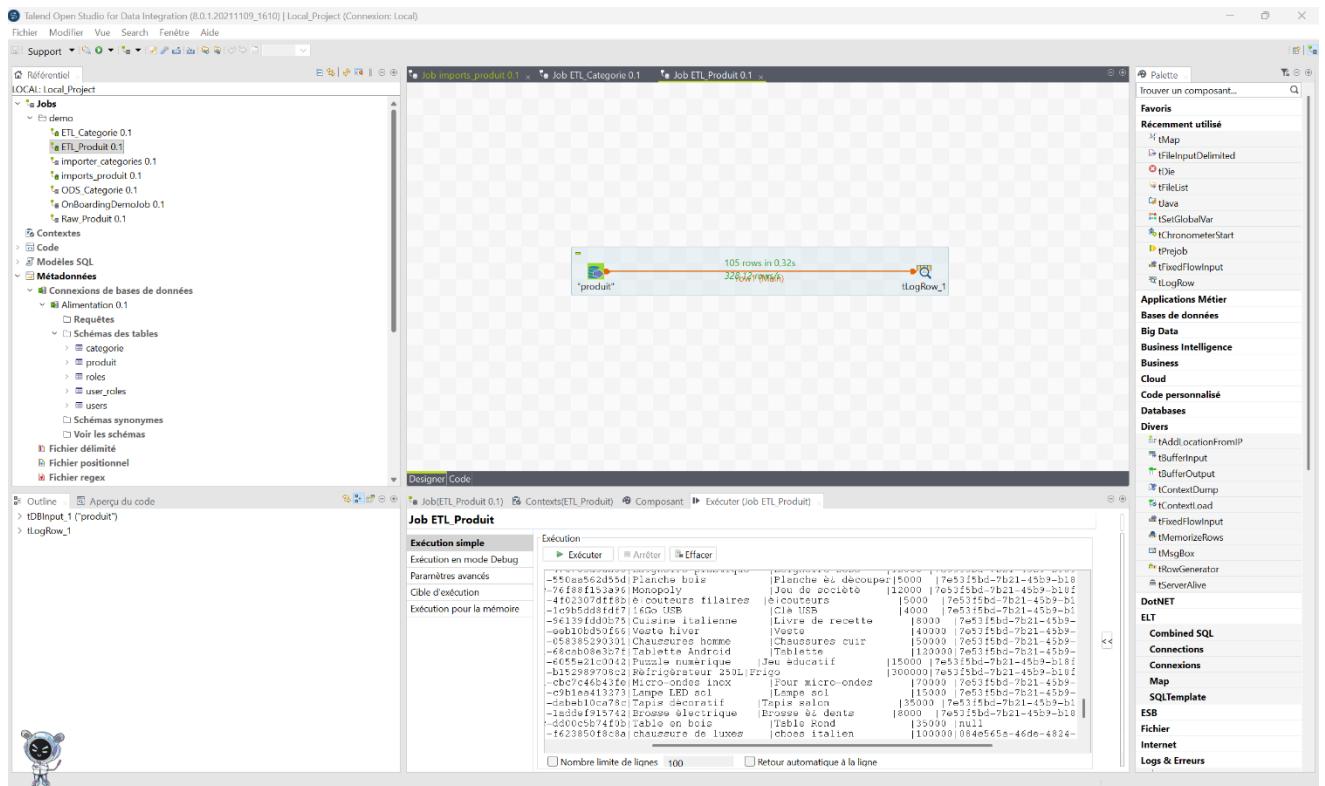
ETL CATEGORIE

ETL CATEGORIE permet d'afficher la liste des catégories enregistrer dans la base de données directement dans Talend



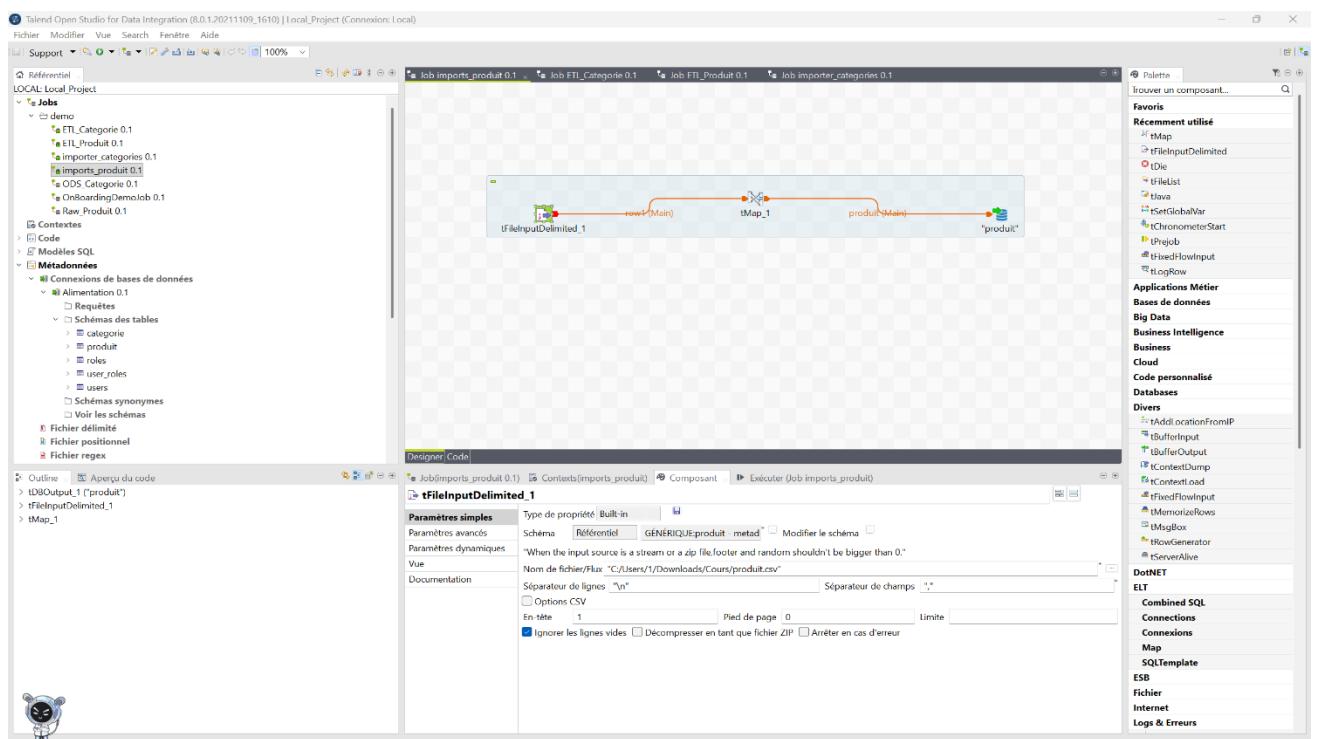
ETL PRODUIT

ETL PRODUIT permet d'afficher la liste des produits enregistrer dans la base de données directement dans Talend



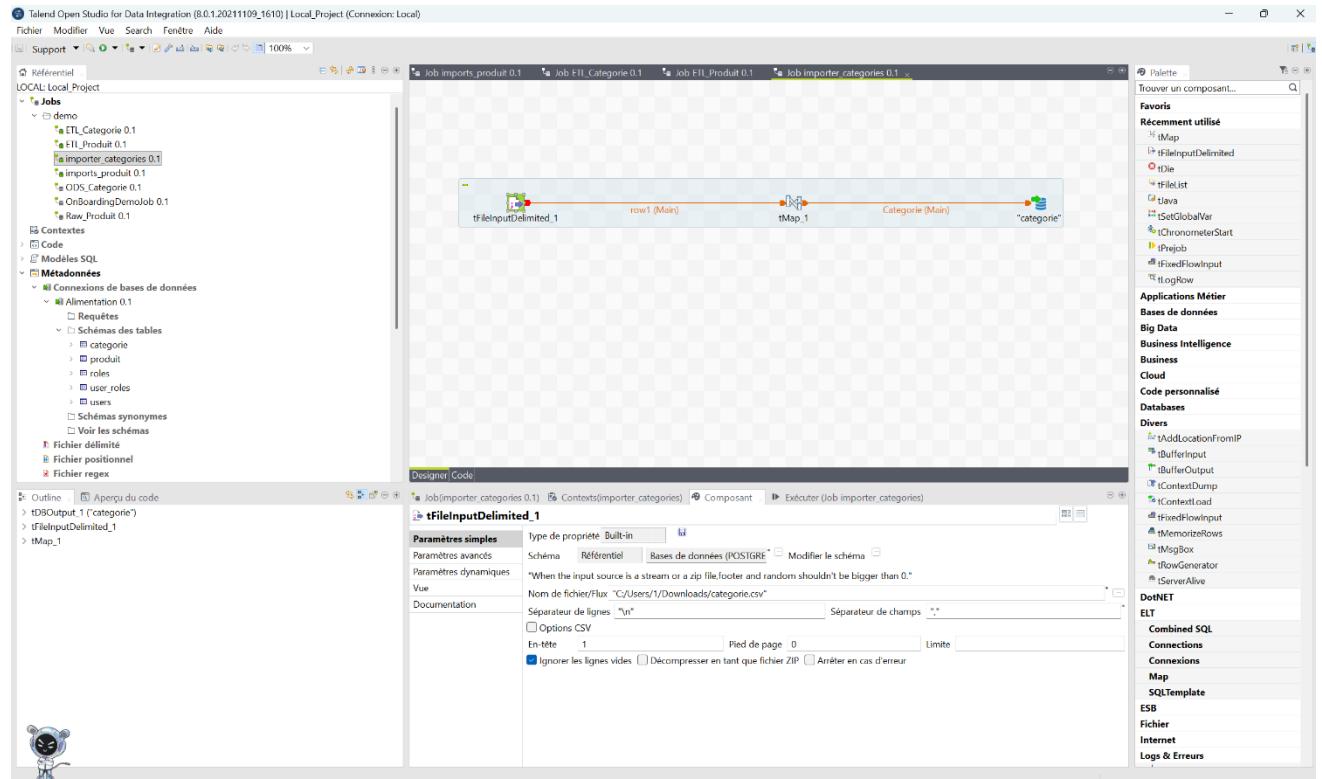
ETL IMPORT PRODUIT

ETL IMPORT PRODUIT permet d'importer une liste de produits de format csv nettoyer ,traiter et enrichie pour qu'on puisse charger (enregistrer) dans la base de données directement dans Talend



ETL IMPORT CATEGORIE

ETL IMPORT CATEGORIE permet d'importer une liste de catégories de format csv nettoyer ,traiter et enrichie pour qu'on puisse charger (enregistrer) dans la base de données directement dans Talend



II. Application RESTful avec Springboot

Spring Boot est un Framework Java open source basé sur Spring, conçu pour faciliter le développement d'applications autonomes grâce à un ensemble de bibliothèques intégrées. Il permet un démarrage rapide des projets et une gestion simplifiée de la configuration.

Dans le cadre de ce projet, nous allons développer une API RESTful sécurisée destinée à la gestion d'un catalogue de produits organisés en catégories. L'application distingue deux types d'utilisateurs : les administrateurs, qui peuvent créer, modifier et supprimer des produits et des catégories, et les utilisateurs classiques, qui disposent uniquement d'un droit de consultation des produits.

L'API est développée avec Spring Boot, sécurisée à l'aide de JWT via Spring Security, et utilise Spring Data JPA pour l'accès aux données. Une journalisation quotidienne permet de tracer les opérations sensibles pour assurer le suivi et l'audit. Les tests des différentes fonctionnalités sont réalisés à l'aide de Postman.

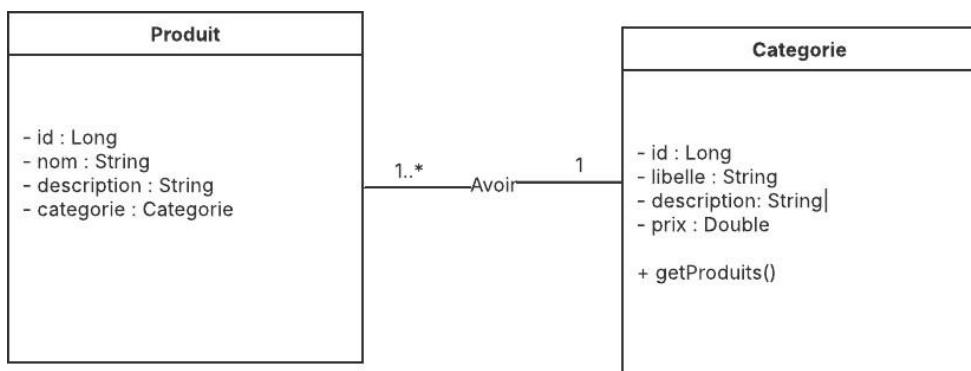
L'architecture repose sur une séparation claire des responsabilités entre les couches contrôleur, service et persistance, ce qui garantit une application fiable, sécurisée, évolutive et facile à maintenir.

I. Modélisation des Données

1. Diagramme de classes

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que leurs relations.

Le schéma ci-dessous représente le diagramme de classe de notre projet



2. Description des attributs et des relations

- Entité Catégorie

Attribut	Type	Annotation JPA	Description
id (<i>hérité</i>)	Long	@Id, @GeneratedValue (hérité)	Identifiant unique (hérité de AbstractId)
libelle	String	—	Libellé (nom) de la catégorie
description	String	—	Description textuelle de la catégorie
produits	List<Produit>	@OneToMany(mappedBy = "categorie")	Liste des produits rattachés à cette catégorie

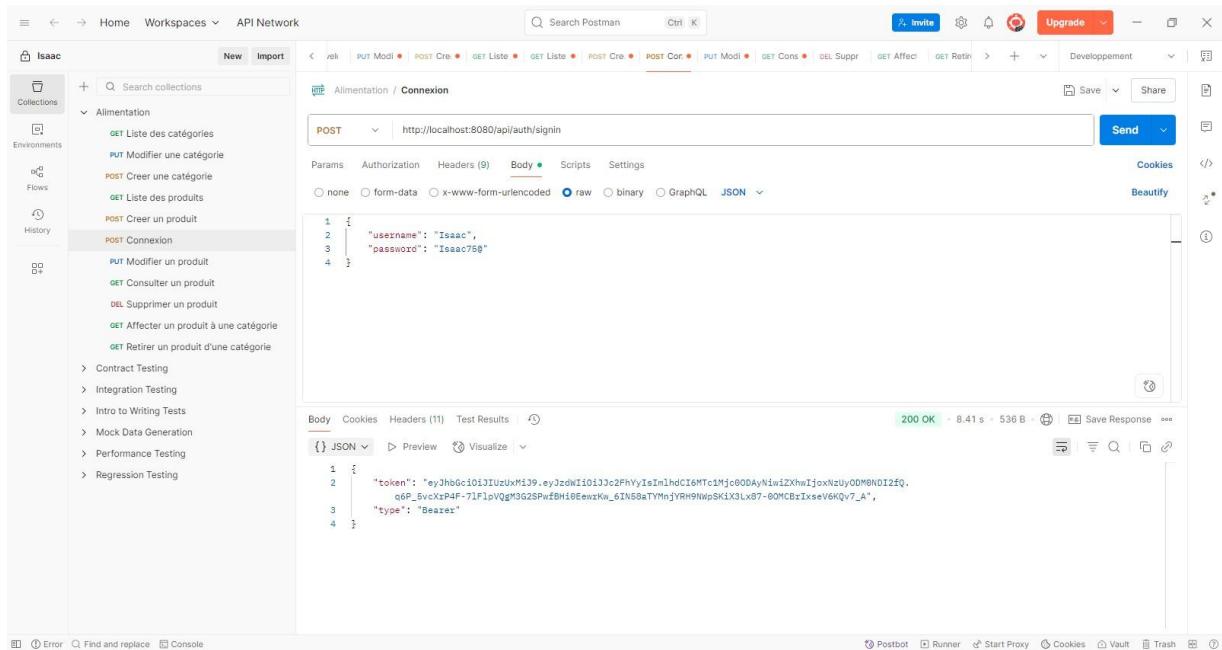
- Entité Produit

Attribut	Type	Description	Contraintes / Annotations
id	Long	Identifiant unique hérité	Hérité de AbstractId (généralement @Id)
nom	String	Nom du produit	Champ simple
description	String	Description détaillée du produit	Champ simple
prix	int	Prix du produit	Champ simple
categorie	Categorie	Catégorie du produit (relation)	@ManyToOne avec @JoinColumn(name = "categorie") + @JsonIgnore pour ignorer en JSON

II. Présentation de l'application

1. Interface de connexion

L'interface de connexion permet à tout utilisateur légitime de se connecter à la plateforme et de disposer légitimement des droits qui lui sont réservés



The screenshot shows the Postman interface with the following details:

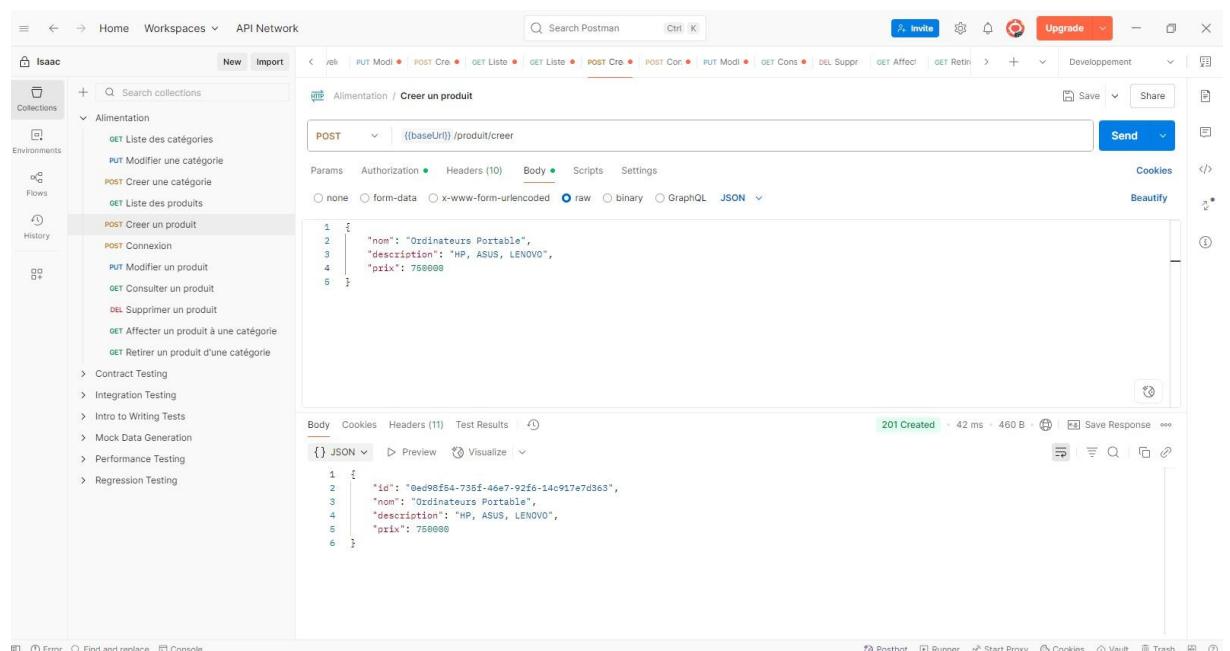
- Collection:** Isaac
- Request:** POST /api/auth/signin
- Body (JSON):**

```
1: {  
2:   "username": "Isaac",  
3:   "password": "Isaac750"  
4: }
```
- Response:** 200 OK (8.41 s, 536 B)

```
1: {  
2:   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3c2PhYyIsImIhdCI6MTc1MjQ0ODAyNiwzKhwIjoxNzUyODM0NDI2fQ,  
3:   "q6P_6vcXsP4F-7]FlpVgQM3Q2SPw#BH10ewzKw_6IN50aTYMnjyRH9NwpSkix3Lx87-0MCBrIxseV6KQv7_A",  
4:   "type": "Beaizer"  
5: }
```

2. Interface ajouté un produit

Cette interface permet à l'administrateur d'ajouter de nouveau produit dans l'alimentation



The screenshot shows the Postman interface with the following details:

- Collection:** Isaac
- Request:** POST /product/creer
- Body (JSON):**

```
1: {  
2:   "nom": "Ordinateurs Portable",  
3:   "description": "HP, ASUS, LENOVO",  
4:   "prix": 750000  
5: }
```
- Response:** 201 Created (42 ms, 460 B)

```
1: {  
2:   "id": "0ed98fb4-735f-4e67-92f6-14c917e7d363",  
3:   "nom": "Ordinateurs Portable",  
4:   "description": "HP, ASUS, LENOVO",  
5:   "prix": 750000  
6: }
```

3. Interface liste des produits

Cette interface permet de voir la liste de tous les produits enregistrés

The screenshot shows the Postman application interface. On the left, there's a sidebar with collections, environments, flows, and history. The main area displays a collection named 'Isaac' under 'Alimentation / Liste des produits'. A specific GET request is selected, with the URL being `((baseUrl)) /produit/liste`. The 'Authorization' tab is open, showing 'Bearer Token' selected, and a token value is pasted into the input field. Below the request, the response body is shown as JSON, containing two product objects:

```
1 [  
2 {  
3     "id": "a82ed4a7-ffea-4553-ac1-94d04cf436ef",  
4     "nom": "chaos Chinois",  
5     "description": "chaussure de luxe",  
6     "prix": 10000  
7 },  
8 {  
9     "id": "8ed98f54-738f-46e7-92f6-1ac917e7d363",  
10    "nom": "Ordinateurs Portable",  
11    "description": "HP, ASUS, LENOVO",  
12    "prix": 750000
```

4. Interface modifier un produit

Cette Interface permet de modifier un produit en utilisant l'identifiant du produit

The screenshot shows the Postman application interface. On the left, there's a sidebar with collections, environments, flows, and history. The main area displays a collection named "Alimentation". A specific endpoint "Modifier un produit" is selected, showing a PUT request with the URL `"/product/update/9d125e7b-1914-4ce7-93c3-708dec438f9e"`. The "Body" tab is active, containing the following JSON payload:

```

1 {
2   "nom": "chesse italien",
3   "description": "RDF est prévu pour des situations dans lesquelles",
4   "prix": 100000
5 }

```

Below the request, the response section shows a 200 OK status with a response time of 30 ms and a size of 482 B. The response body is identical to the request body.

5. Interface supprimer un produit

Cette interface permet de supprimer un produit en utilisant l'identifiant du produit

The screenshot shows the Postman application interface. The sidebar and collection structure are identical to the previous screenshot. The endpoint "Supprimer un produit" is selected, showing a DELETE request with the URL `"/product/delete/99df5460-9921-43eb-8a5e-150d32e9dbc6"`. The "Body" tab is active, indicating that this request does not have a body.

Below the request, the response section shows a 204 No Content status with a response time of 235 ms and a size of 282 B. The response body is empty, containing only the number 1.

6. Interface liste des catégories

L'interface liste des catégories permet d'avoir la liste de toutes les catégories disponibles

7. Interface modifier une catégorie

Cette interface permet de modifier une catégorie en utilisant l'id de la catégorie

8. Interface des logs

Permet d'afficher les logs de l'application.

The screenshot shows the Postman interface with the following details:

- Collection:** Isaac
- Request:** GET /Alimentation/logs
- Response Status:** 200 OK
- Response Body (JSON):**

```

1 [
2   "18:19:48.669 [main] INFO c.p.a.AlimentationApplication - Starting AlimentationApplication using Java 17 with PID 4460
3   (C:\DevOps\SpringBoot\Wurzel\alimentation\target\classes started by Isaac in C:\DevOps\SpringBoot\Wurzel\alimentation)",
4   "18:19:48.615 [main] INFO c.p.a.AlimentationApplication - No active profile set, falling back to 1 default profile: 'default'",
5   "18:19:49.938 [main] INFO o.s.d.r.c.RepositoryConfigurationDelegate - Bootstrapping Spring Data JPA repositories in DEFAULT mode.",
6   "18:19:50.076 [main] INFO o.s.d.r.c.RepositoryConfigurationDelegate - Finished Spring Data repository scanning in 123 ms. Found 3 JPA
7   repository interfaces.",
8   "18:19:51.776 [main] INFO o.s.b.w.e.tomcat.TomcatWebServer - Tomcat initialized with port 8080 (http)",
9   "18:19:51.798 [main] INFO o.a.coyote.http11.Http11NioProtocol - Initializing ProtocolHandler [\"http-nio-8080\"]",
10  "18:19:51.882 [main] INFO o.a.catalina.core.StandardService - Starting service [Tomcat]",
11  "18:19:51.883 [main] INFO o.a.catalina.core.StandardEngine - Starting Servlet engine: [Apache Tomcat/10.1.41]"

```

9. Interface créer une catégorie

Cette interface permet la création d'une nouvelle catégorie

The screenshot shows the Postman interface with the following details:

- Collection:** Isaac
- Request:** POST /Alimentation/categorie/creer
- Response Status:** 201 Created
- Response Body (JSON):**

```

1 {
2   "id": "68e3cf8c-b36a-4195-bf47-0f28e5d99d74",
3   "libelle": "Boissons",
4   "description": "Toutes sortes de boissons",
5   "products": []
6 }

```

III. Métriques Prometheus intégré

Prometheus est un **outil open-source de monitoring et d'alerting** (supervision) utilisé pour surveiller les performances d'applications, de services, de bases de données, de serveurs, et d'infrastructures cloud.

Il a été créé par **Sound Cloud** et fait maintenant partie de la **Cloud Native Computing Foundation (CNCF)**, comme Kubernetes.

Pour cette application nous avons créé plusieurs métriques en plus de celle proposer par prometheus

Nous avons entre autres :

1) Counter : **produit_cree_total, produit_modifie_total, produit_supprime_total**

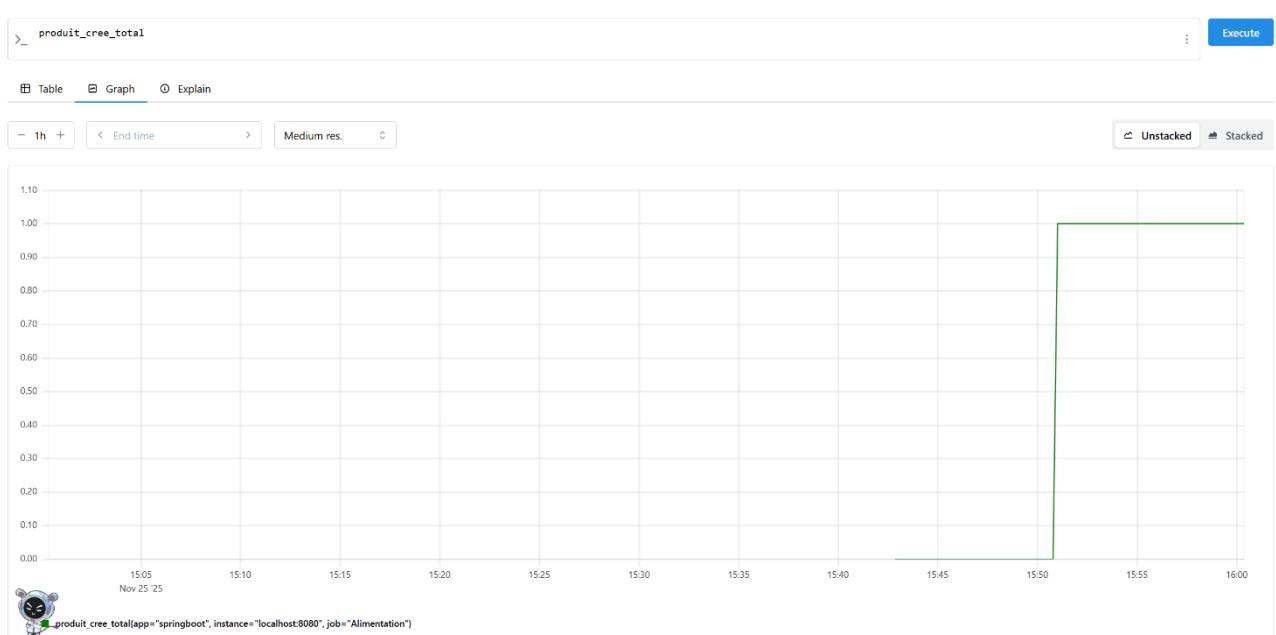
Un Counter est un compteur qui **ne fait qu'augmenter**.

produit_cree_total

→ Nombre total de produits **créés** depuis le démarrage de l'application.

Chaque fois que tu exécutes POST /produits, tu fais :

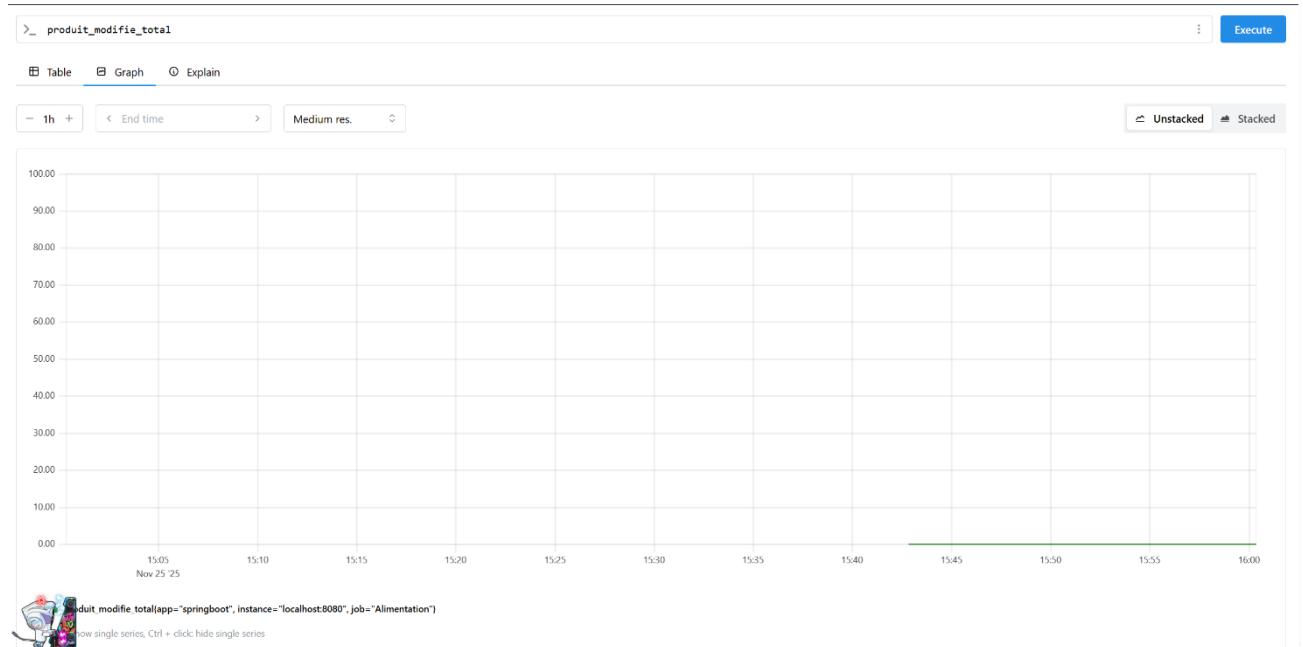
counter.increment() → et la valeur augmente.



produit_modifie_total

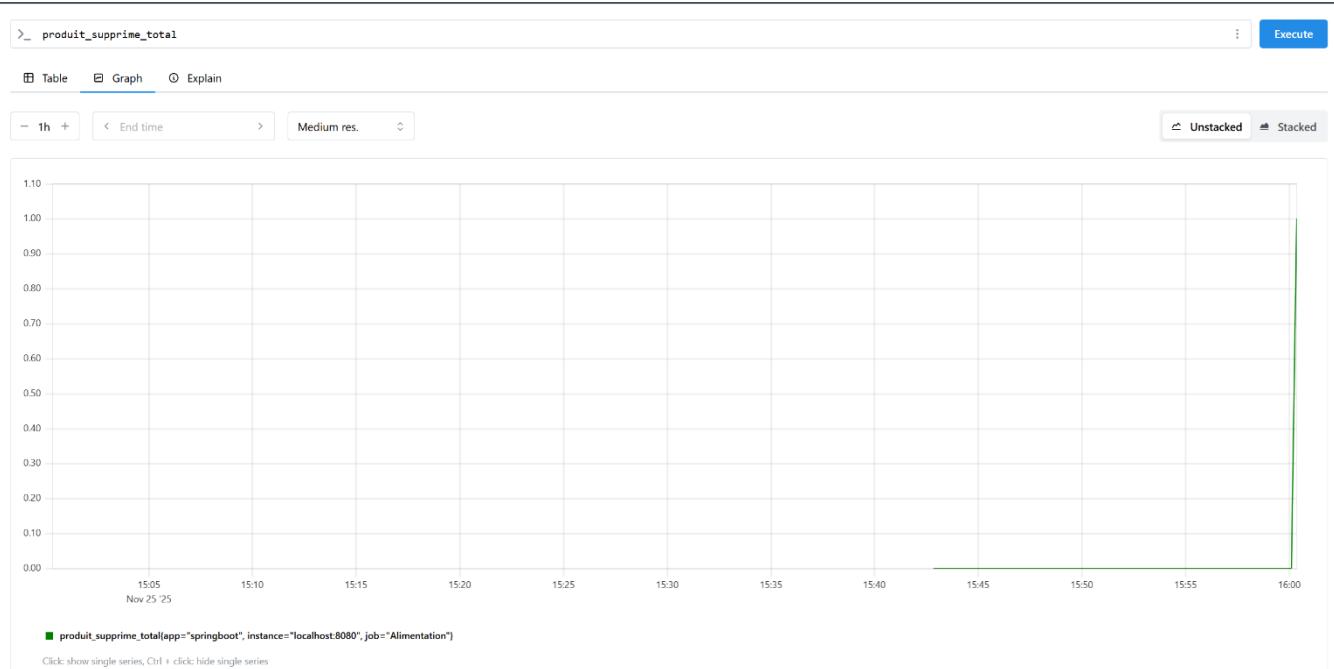
→ Nombre total de produits **modifiés**.

Chaque PUT /produits/{id} ou update incrémentale ce compteur.



produit_supprime_total

→ Nombre total de produits **supprimés**.
Chaque DELETE /produits/{id} l'augmente.



2) Timer : temps_creation_produit_seconds_*

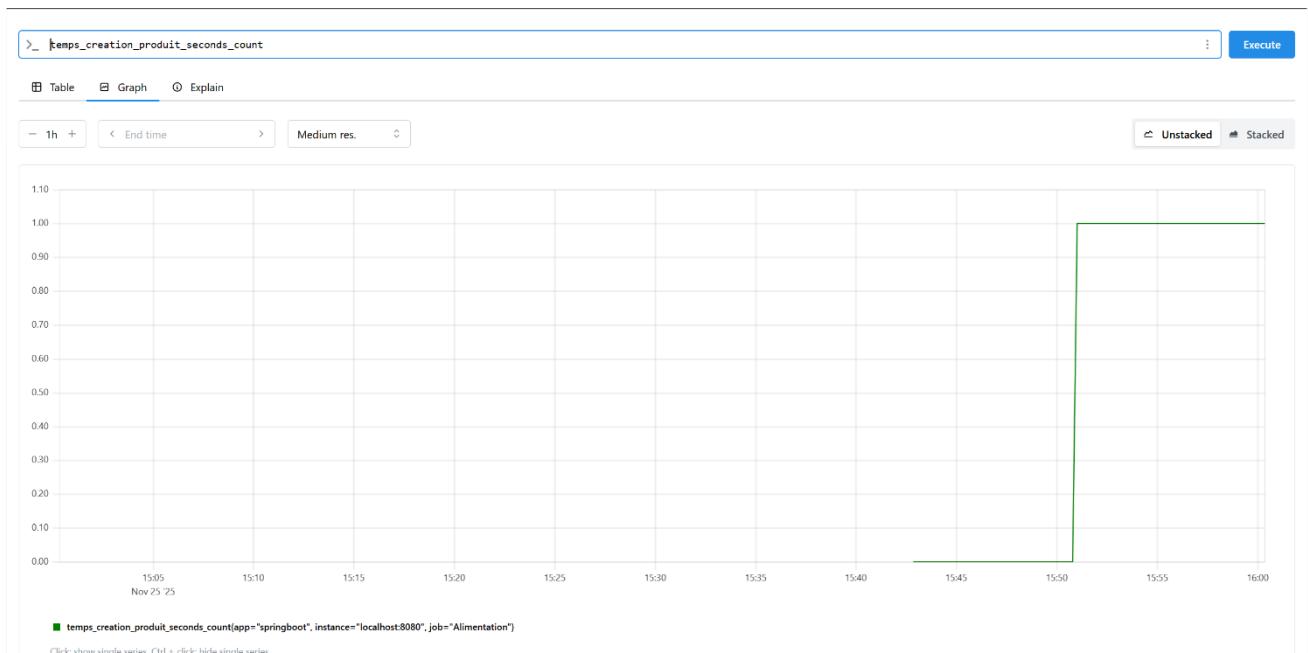
Un **Timer** mesure un **temps d'exécution** d'une opération.
Pour Timer tempsCreationProduit, Prometheus expose **3 métriques importantes** :

temps_creation_produit_seconds_count

Le **nombre de fois** où l'opération a été mesurée.

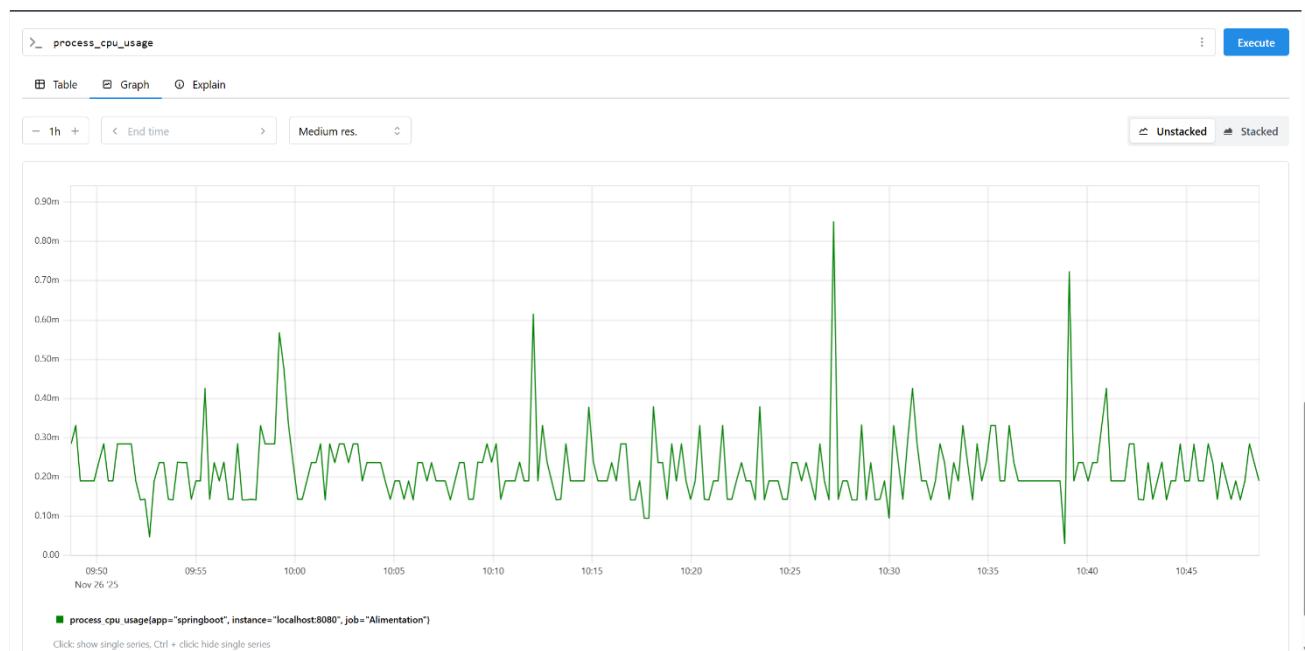
Exemple :

Si tu as créé 12 produits → count = **12**



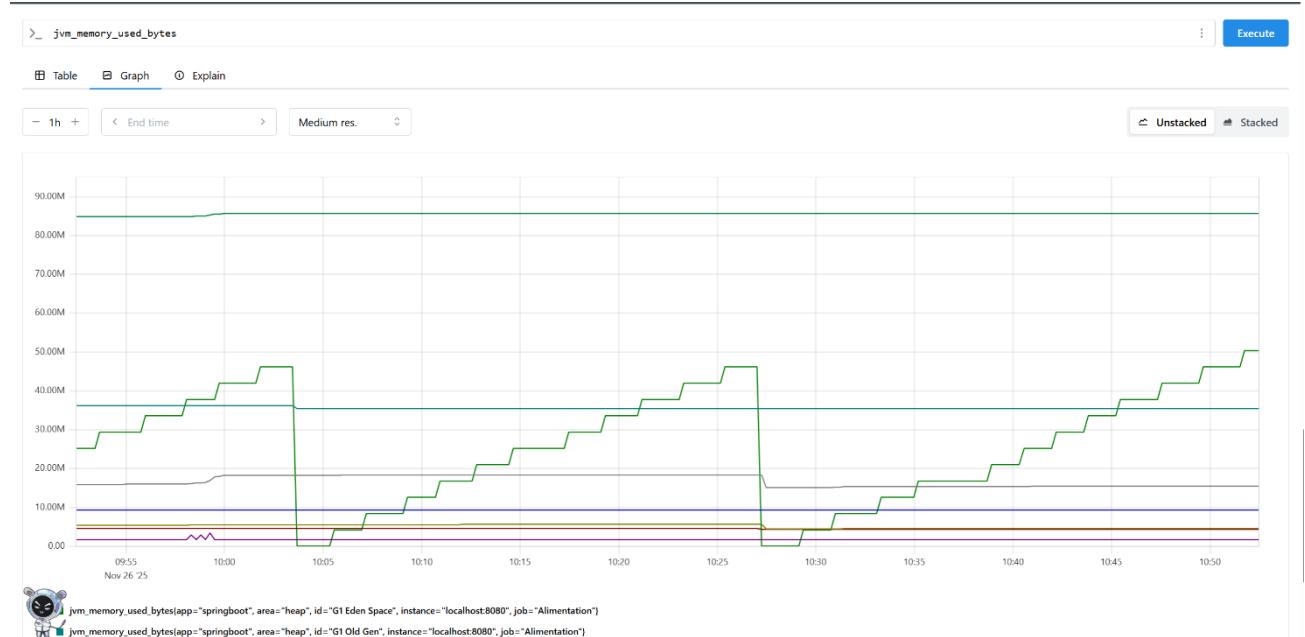
process_cpu_usage

Pourcentage du CPU utilisé par le processus Java.
Une valeur proche de 1.0 = 100% de CPU utilisé.



jvm_memory_used_bytes

Quantité de mémoire actuellement utilisée par la JVM (heap + non-heap).
Permet de détecter les fuites de mémoire et les risques d'OutOfMemoryError.



■ G1 Eden Space (courbe verte)

C'est la mémoire où les nouveaux objets sont créés.

C'est elle qui monte et descend le plus fréquemment.

Si Eden monte → tu crées beaucoup d'objets.

Si Eden redescend → GC a nettoyé.

● G1 Survivor Space

C'est une petite zone intermédiaire.

Les objets survivant au GC de Eden sont mis ici.

● G1 Old Gen (gris)

C'est la mémoire longue durée :

- objets persistants
- caches
- configuration Spring
- beans de l'application

Elle monte plus lentement et ne redescend que rarement.

Conclusion

En résumé, le développement de cette API RESTful avec Spring Boot nous a permis de construire une application robuste et sécurisée pour gérer un catalogue de produits et leurs catégories. Grâce à l'intégration de Spring Security et JWT, nous avons assuré une gestion fine des droits d'accès entre administrateurs et utilisateurs classiques. L'utilisation de Spring Data JPA a grandement simplifié l'accès aux données, tandis que la mise en place d'une journalisation quotidienne renforce la traçabilité et la sécurité des opérations sensibles. Enfin, en structurant l'application selon une architecture en couches claires (contrôleur, service, persistance), nous avons obtenu une solution non seulement fonctionnelle mais également maintenable et évolutive, prête à répondre aux besoins futurs.