

UNIVERSITE JOSEPH KI-ZERBO
(UJKZ)

INSTITUT BURKINABE DES ARTS ET METIERS
(IBAM)



Master informatique

PROJET DE DEVELOPPEMENT A BASE DE COMPOSANT ET SERVICES WEB

GROUPE 8

EXERCICE 1 :

DEVELOPPEMENT D'UNE APPLICATION RESTFUL AVEC SPRING
BOOT UTILISANT OPEN TALEND STUDIO POUR GERER LE
PROCESSUS ETL ET PROMETHEUS POUR LE MONITORING

Membres du groupe

COMPAORE Ibrahim

OUATTARA Sié Lamoussa

OUEDRAOGO Yasmine

SAWADOGO Tatiana

Enseignant :

Monsieur Oumar KY

Année Académique 2023-2022

Introduction.....	4
1 Objectif du projet.....	5
2 Technologies utilisées.....	5
2.1 Spring Boot.....	5
2.2 Talend Open Studio.....	5
2.3 MySQL	5
2.4 PostgreSQL.....	6
2.5 Prometheus.....	6
2.6 Postman.....	6
2.7 Docker	6
3 Méthodologie.....	7
3.1 Extraction et Transformation des Données avec Talend Open Studio	7
3.2 Développement de l'API RESTful avec Spring Boot	7
3.3 Intégration de Prometheus pour le Monitoring	9
4 Guide d'installation	10
4.1 Prérequis.....	10
4.2 Installation de l'Application Spring Boot	10
4.2.1 Clone du projet depuis le repository git.....	10
4.2.2 Configuration de la base de données	10
4.2.3 Modification de l'adresse IP dans la configuration de Prometheus.....	10
4.2.4 Lancer l'application avec Maven	11
5 Tests des endpoints avec Postman	13
5.1 GET http://localhost:8081/api/produits.....	13
5.2 GET http://localhost:8081/api/produits/1012.....	14
5.3 POST http://localhost:8081/api/produits	14
5.4 PUT http://localhost:8081/api/produits/1012.....	15
5.5 DELETE http://localhost:8081/api/produits/1012.....	16
6 Monitoring de l'application	16
7 Résultats	17
Conclusion.....	19

Figure 1: Résumé du processus ETL.....	7
Figure 2: Initialisation du projet	8
Figure 3: Entité produit	8
Figure 4: Repository.....	8
Figure 5: Controller.....	9
Figure 6:Dépendance prometheus.....	9
Figure 7:Configuration de prometheus.....	9
Figure 8:Résultat de la récupération du projet depuis git	10
Figure 9: Configuration de la base de données.....	10
Figure 10: Docker démarré	11
Figure 11: Terminal ouvert dans le projet.....	11
Figure 12: Résultat de la commande docker-compose	11
Figure 13: Conteneur prometheus démarré dans docker	12
Figure 14: Page d'accueil de Prometheus	12
Figure 15: Résultat de la commande mvn clean install	13
Figure 16: L'application spring boot démarré	13
Figure 17 : Test d'affichage de la liste des produits	14
Figure 18: Test de récupération d'un produit.....	14
Figure 19: Test de création d'un produit	15
Figure 20: Test de modification d'un produit	16
Figure 21 : Test de suppression d'un produit.....	16
Figure 22 : Prometheus connecté à notre application.....	17
Figure 23 : Les métriques de Prometheus.....	17
Figure 24 : Evolution de l'usage du CPU.....	17

Introduction

Dans le cadre de ce projet, nous avons pour objectif de développer une application RESTful en utilisant Spring Boot, qui s'appuie sur Talend Open Studio pour se connecter et manipuler des données dans une base de données MySQL. Le but est de construire une API exposant ces données sous forme d'endpoints REST, tout en assurant un suivi des performances et de l'intégrité de l'application à l'aide de Prometheus pour le monitoring.

Ce projet fait partie d'une démarche globale visant à intégrer des technologies modernes comme Talend pour l'extraction et la transformation des données, et Spring Boot pour fournir une architecture RESTful performante et évolutive. Le monitoring de l'application avec Prometheus permet de surveiller les métriques système en temps réel, ce qui garantit une visibilité accrue sur l'utilisation des ressources et les performances de l'application en production.

L'architecture repose sur une interaction fluide entre plusieurs composants clés :

- Talend Open Studio pour l'intégration et le traitement des données.
- Spring Boot pour construire et déployer une API RESTful capable de gérer efficacement les données extraites.
- Prometheus pour superviser l'application et assurer une surveillance continue des ressources et des performances.

Ce rapport décrit en détail les étapes de mise en œuvre du projet, y compris l'installation, le développement et les tests, afin d'assurer que chaque composant fonctionne harmonieusement au sein de l'écosystème.

1 Objectif du projet

Afin pouvoir évaluer le succès du projet, nous allons axer notre analyse sur les objectifs suivants :

- Utiliser Talend Open Studio pour extraire et transformer des données de produits dans une base de données MySQL ou PostgreSQL.
- Développer une application Spring Boot pour exposer ces données via une API RESTful.
- Intégrer des métriques de Prometheus pour surveiller les performances de l'application.
- Tester l'API avec Postman et monitorer l'application avec Prometheus

2 Technologies utilisées

2.1 Spring Boot



Spring Boot est un framework Java permettant de créer rapidement des applications autonomes et basées sur le web. Il fournit des abstractions puissantes pour développer des API RESTful, ce qui simplifie la gestion des requêtes HTTP et l'interaction avec les bases de données via JPA.

2.2 Talend Open Studio



Talend Open Studio est un outil d'intégration de données open source permettant de concevoir des workflows pour extraire, transformer et charger (ETL) des données à partir de diverses sources de données. Dans ce projet, Talend est utilisé pour connecter la base de

données MySQL/PostgreSQL et effectuer des opérations d'extraction et de transformation des données de produits.

2.3 MySQL



MySQL est un Système de Gestion de Bases de Données Relationnelles (SGBDR). Il a été utilisé dans ce projet pour stocker et gérer les données sources du processus ETL.

2.4 PostgreSQL



PostgreSQL est également un Système de Gestion de Bases de Données Relationnelles (SGBDR) ayant servi pour stocker et gérer les données issues du processus ETL. C'est avec ce dernier que notre API Spring Boot interagit pour effectuer les opérations CRUD (Create, Read, Update, Delete).

2.5 Prometheus



Prometheus est une plateforme open-source de surveillance et d'alerte conçue pour surveiller les systèmes en temps réel. Dans ce projet, Prometheus est utilisé pour collecter des métriques liées aux performances et à l'état de l'application, qui sont exposées

via l'endpoint /actuator/prometheus de Spring Boot.

2.6 Postman



Postman est un outil permettant de tester les API web. Il facilite l'envoi de requêtes HTTP pour interagir avec les différents endpoints REST de l'application, garantissant que l'API fonctionne comme prévu.

2.7 Docker



Docker est une plateforme permettant de créer, déployer et exécuter des applications dans des conteneurs. Dans ce projet, Docker a été utilisé pour containeriser Prometheus, facilitant ainsi son déploiement et sa gestion. La containerisation de Prometheus permet de

garantir une configuration uniforme et un environnement isolé, simplifiant la gestion des dépendances et la portabilité de l'outil de monitoring. Les conteneurs Docker assurent que Prometheus fonctionne de manière cohérente sur différentes machines et environnements de déploiement.

3 Méthodologie

3.1 Extraction et Transformation des Données avec Talend Open Studio

Le processus ETL est décrit comme suit :

- Connexion à la Base de Données source : Nous avons configuré une « Métadonnée » dans Talend Open Studio pour se connecter à la base de données MySQL.
- Création d'un Job Talend : Les données des produits sont extraites, puis transformées pour être prêtes à l'utilisation.
- Insertion des données dans PostgreSQL : Les données sont transformées et stockées dans un format approprié (dans notre cas une base de données PostgreSQL).

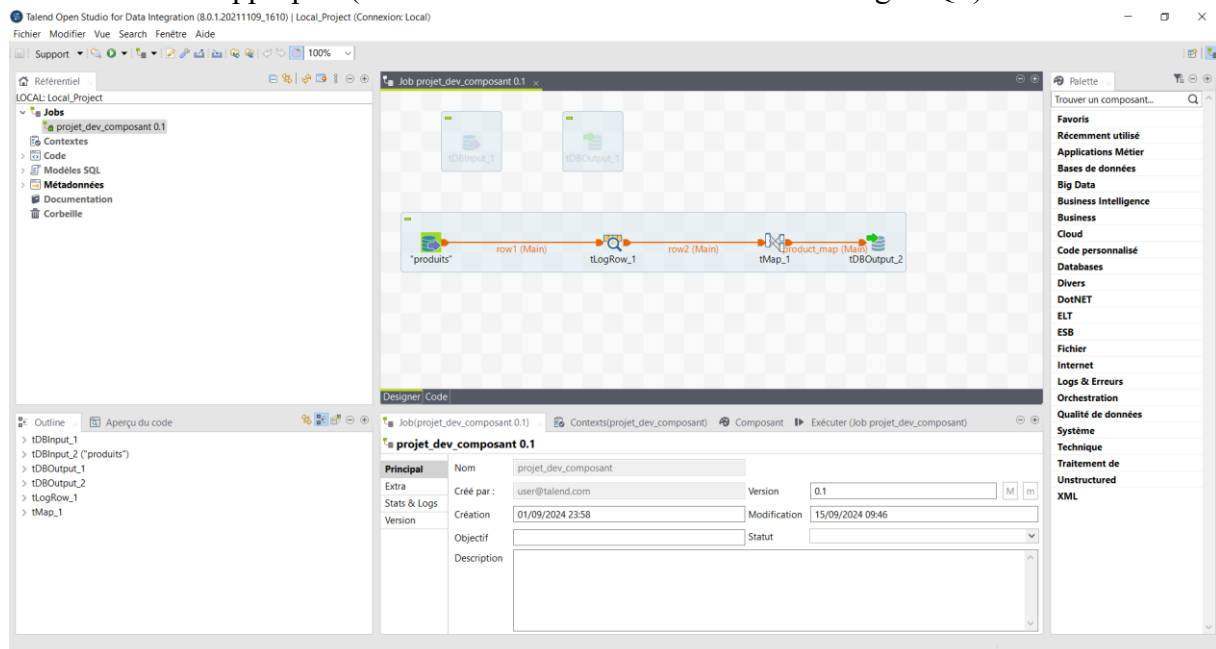


Figure 1: Résumé du processus ETL

3.2 Développement de l'API RESTful avec Spring Boot

- Initialisation du Projet : Utilisation de Spring Boot avec des dépendances nécessaires

Project

☒ Gradle - Groovy
 ☐ Gradle - Kotlin
 ☐ Maven

Language

☒ Java
 ☐ Kotlin
 ☐ Groovy

Spring Boot

☐ 3.4.0 (SNAPSHOT)
 ☐ 3.4.0 (M2)
 ☐ 3.3.4 (SNAPSHOT)
 ☒ 3.3.3
 ☐ 3.2.10 (SNAPSHOT)
 ☐ 3.2.9

Project Metadata

Group

com.ibam.master.groupe8

Artifact

product-restfull

Name

product-restfull

Description

APPLICATION RESTFULL DE GESTION DE PRODUITS AVEC MONITEUR

Package name

com.ibam.master.groupe8

Packaging

☒ Jar
 ☐ War

Java

☐ 22
 ☐ 21
 ☒ 17

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

PostgreSQL Driver

SQL

A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

Spring Boot Actuator

OPS

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Lombok

DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Figure 2: Initialisation du projet

- Modélisation des Entités : Création d'une entité Product pour gérer les produits extraits par Talend.

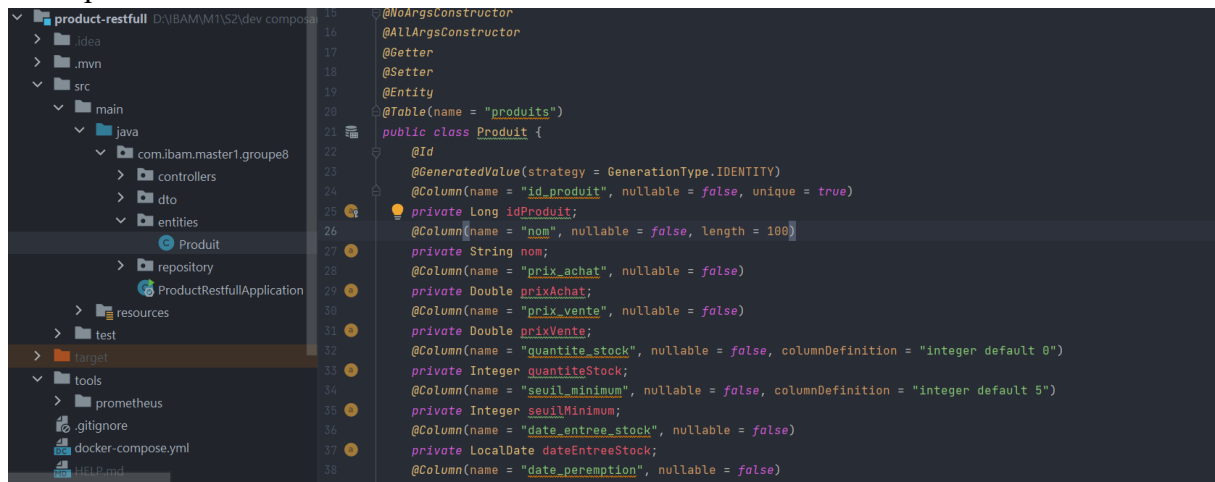


Figure 3: Entité produit

- Mise en place du Repository : Utilisation de Spring Data JPA pour interagir avec la base de données.

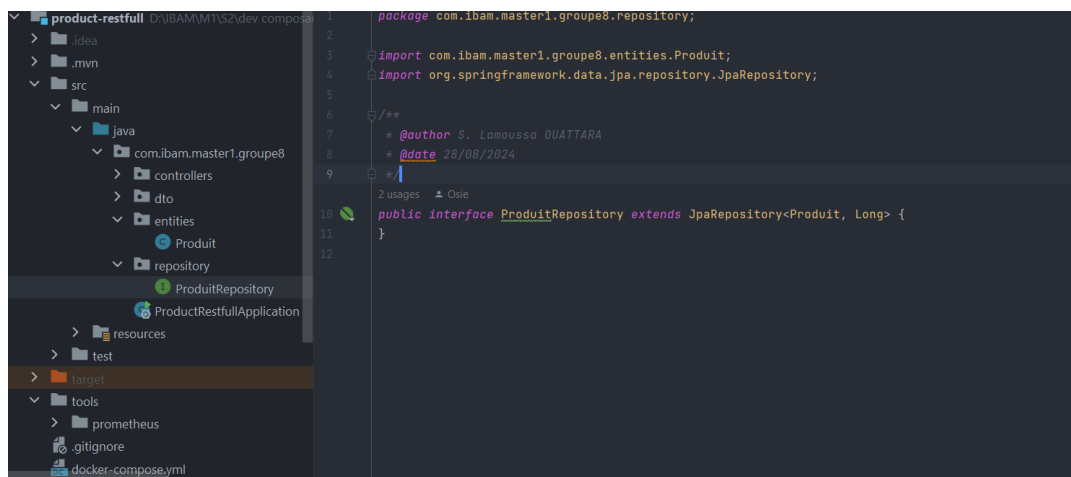


Figure 4: Repository

- Création du Contrôleur REST : Développement du contrôleur `ProduitRestController` pour exposer les données via des endpoints REST comme GET `/api/products`, POST `/api/products`, PUT `/api/products/{id}`, et DELETE `/api/products/{id}`.

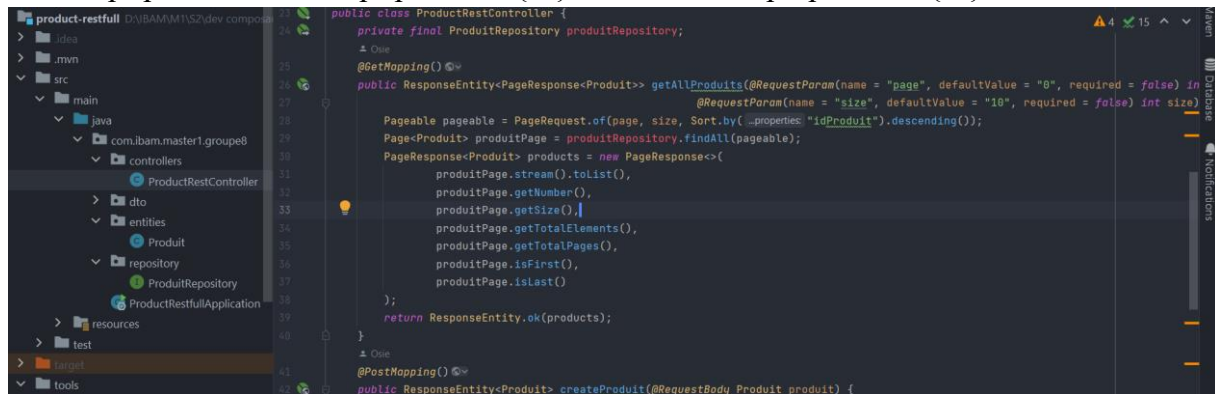


Figure 5: Controller

3.3 Intégration de Prometheus pour le Monitoring

- Ajout de Prometheus : Inclusion de la dépendance Prometheus pour Spring Boot Actuator afin de générer et exposer des métriques sur l'URL `/actuator/prometheus`.

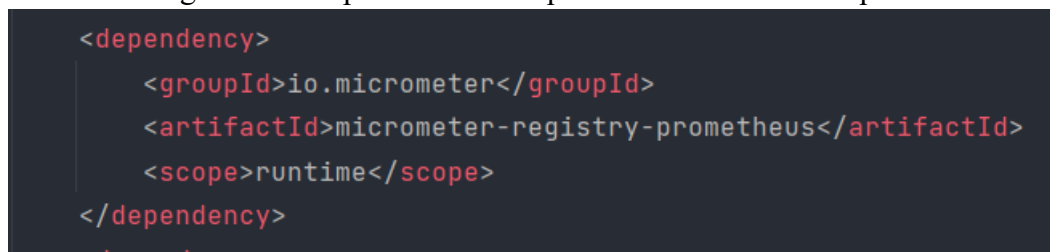


Figure 6: Dépendance prometheus

- Configuration de Prometheus : Configuration du fichier `prometheus.yml` pour surveiller l'application Spring Boot.

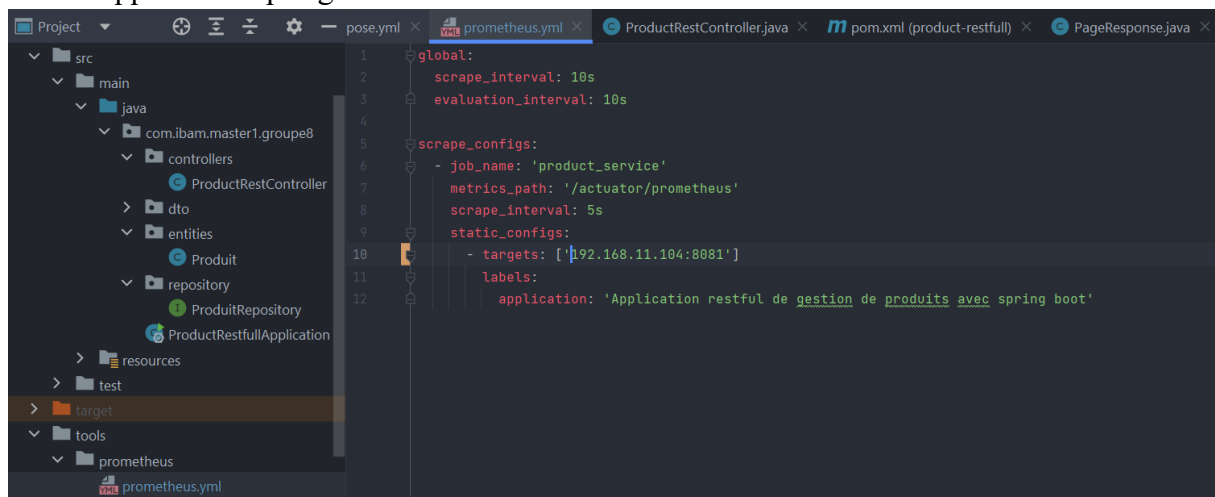


Figure 7: Configuration de prometheus

4 Guide d'installation

4.1 Prérequis

- Java 17 ou supérieur
- Maven pour gérer les dépendances du projet
- PostgreSQL pour la base de données
- Talend Open Studio pour l'intégration de données
- Docker pour containeriser Prometheus
- Postman pour tester l'API

4.2 Installation de l'Application Spring Boot

4.2.1 Clone du projet depuis le repository git

Dans un terminal taper la commande suivante « git clone <https://github.com/ibam-master1-groupe8/product-restfull.git> ». Ensuite toujours dans le même terminal taper « cd product-restfull ». La figure suivante illustre le résultat des actions précédentes :

```
Osie.lamoussa@OSIE-DESKTOP-3P9D60B MINGW64 /d/IBAM/M1/S2/dev composant et web services
$ git clone https://github.com/ibam-master1-groupe8/product-restfull.git
Cloning into 'product-restfull'...
remote: Enumerating objects: 105, done.
remote: Counting objects: 100% (105/105), done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 105 (delta 33), reused 82 (delta 13), pack-reused 0 (from 0)
Receiving objects: 100% (105/105), 3.12 MiB | 3.04 MiB/s, done.
Resolving deltas: 100% (33/33), done.

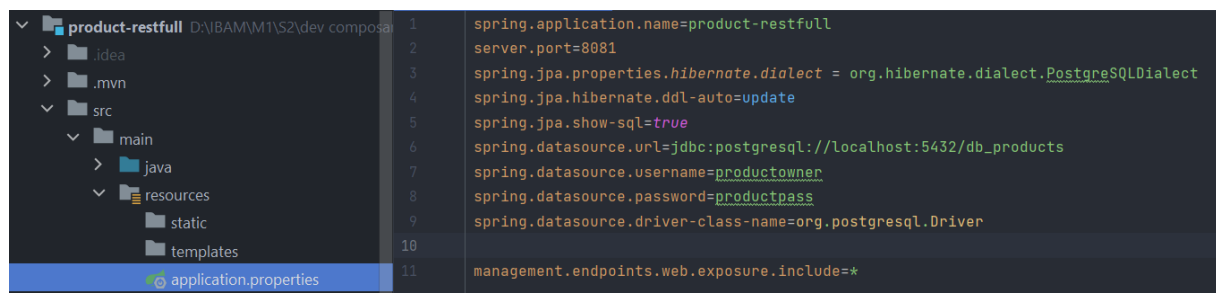
Osie.lamoussa@OSIE-DESKTOP-3P9D60B MINGW64 /d/IBAM/M1/S2/dev composant et web services
$ cd product-restfull/

Osie.lamoussa@OSIE-DESKTOP-3P9D60B MINGW64 /d/IBAM/M1/S2/dev composant et web services/product-restfull (main)
$ |
```

Figure 8: Résultat de la récupération du projet depuis git

4.2.2 Configuration de la base de données

Dans le fichier « application.properties » changer « producowner » et « productpass » respectivement par l'utilisateur de votre base de données PostgreSQL et son mot de passe.



```
1 spring.application.name=product-restfull
2 server.port=8081
3 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
4 spring.jpa.hibernate.ddl-auto=update
5 spring.jpa.show-sql=true
6 spring.datasource.url=jdbc:postgresql://localhost:5432/db_products
7 spring.datasource.username=productowner
8 spring.datasource.password=productpass
9 spring.datasource.driver-class-name=org.postgresql.Driver
10
11 management.endpoints.web.exposure.include=*
```

Figure 9: Configuration de la base de données

4.2.3 Modification de l'adresse IP dans la configuration de Prometheus

Dans le fichier « prometheus.yml » qui se trouve dans le dossier « tools/prometheus », remplacer l'adresse IP de la ligne contenant « targets : [192.168.11.104 :8081] » par la vôtre.

4.2.4 Lancer l'application avec Maven

S'assurer que docker est démarré comme suit :

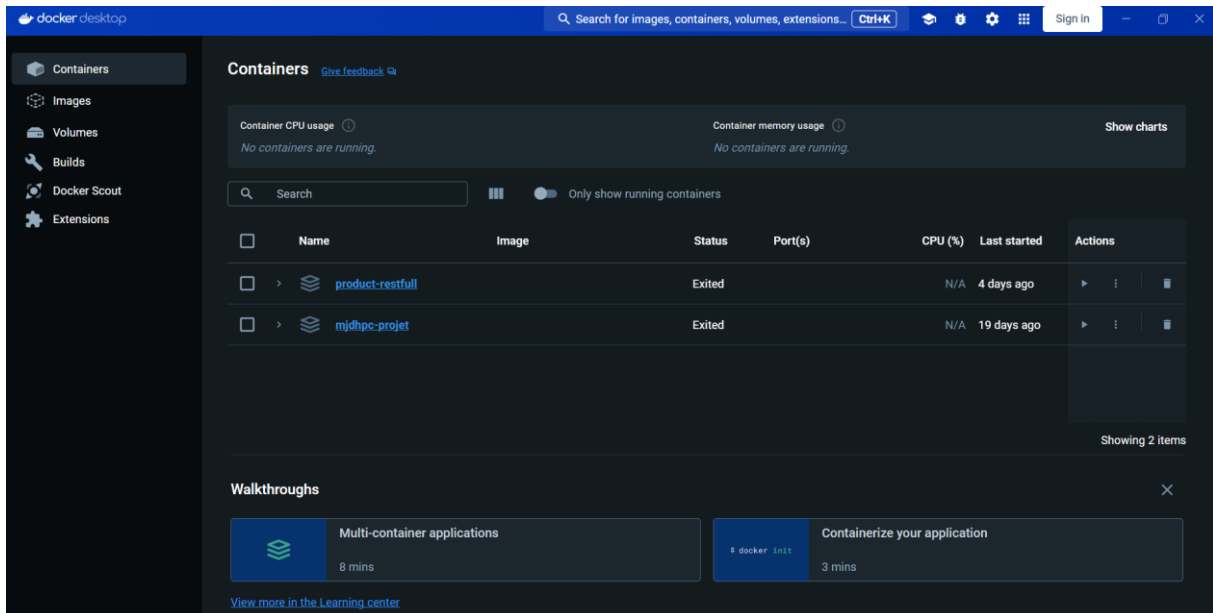


Figure 10: Docker démarré

Se positionner dans le repertoire racine du projet via un terminal comme le montre la figure suivante :

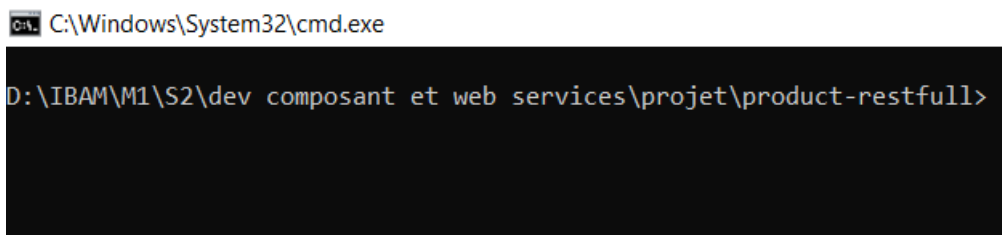


Figure 11: Terminal ouvert dans le projet

Taper les commandes suivantes dans :

- Docker-compose up -d

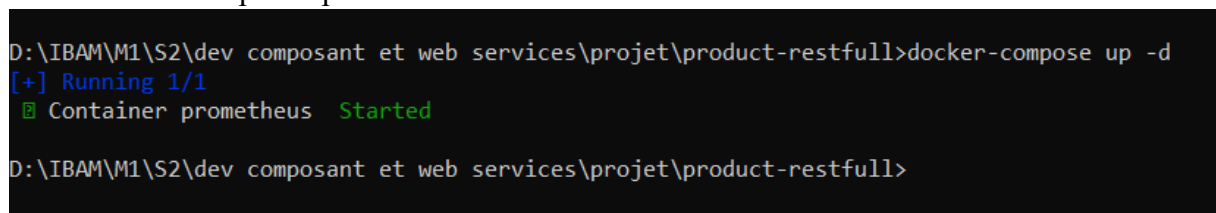


Figure 12: Résultat de la commande docker-compose

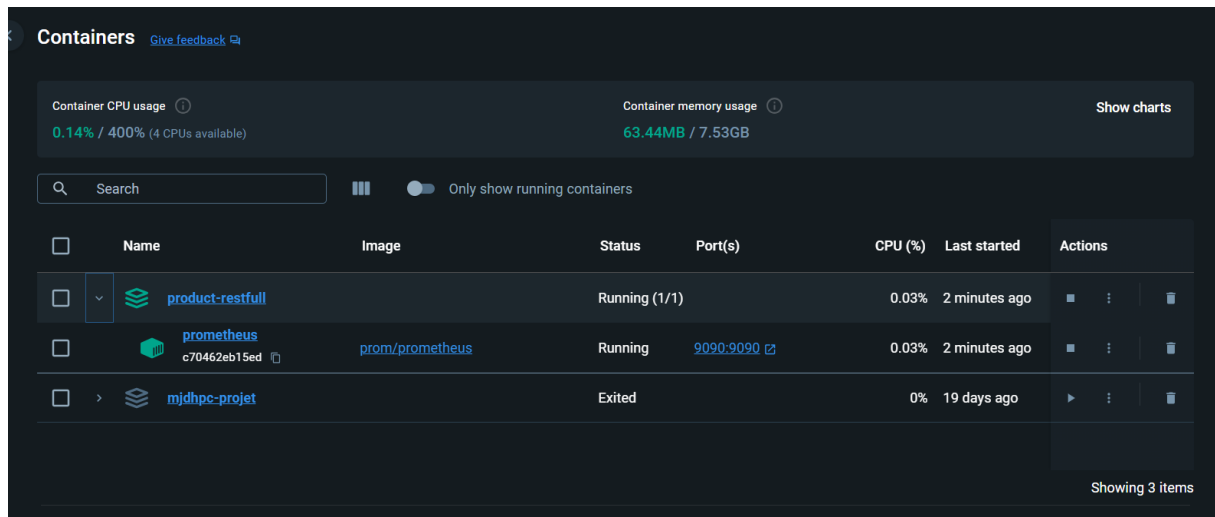


Figure 13: Conteneur prometheus démarré dans docker

Vous pouvez aussi vérifier le lancement de Prometheus en tapant dans un navigateur le lien ci-dessous :

<http://localhost:9090/>

Vous obtenez ce résultat :

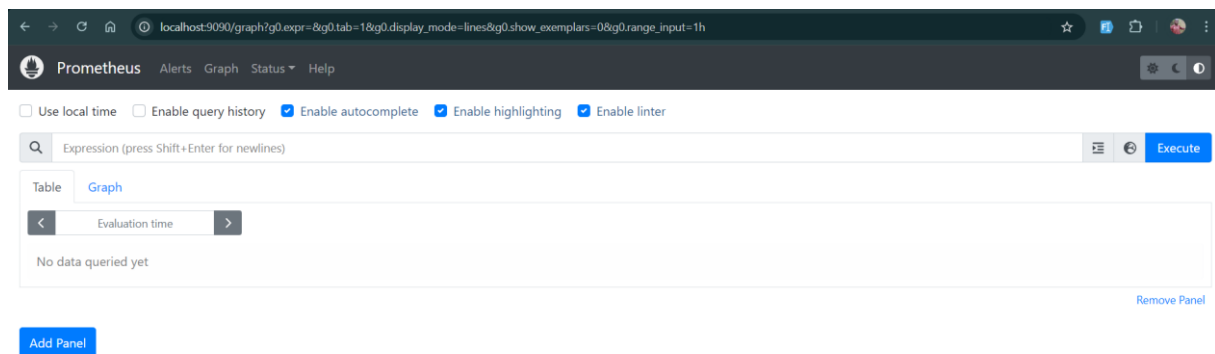


Figure 14: Page d'accueil de Prometheus

- mvn clean install

```
C:\Windows\System32\cmd.exe
fo [name: default]
2024-09-18T21:31:07.544Z INFO 1680 --- [product-restfull] [ main] [ org.hibernate.Version : HH0000412: Hibernate ORM core version
5.2.Final : HH0000026: Second-level cache disabled
2024-09-18T21:31:07.677Z INFO 1680 --- [product-restfull] [ main] [ o.h.c.internal.RegionFactoryInitiator : HH0000026: Second-level cache disabled
2024-09-18T21:31:10.419Z INFO 1680 --- [product-restfull] [ main] [ o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA
class transformer
2024-09-18T21:31:10.685Z INFO 1680 --- [product-restfull] [ main] [ com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-09-18T21:31:12.143Z INFO 1680 --- [product-restfull] [ main] [ com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection org.po
gresql.jdbc.PgConnection@2e0fd83
2024-09-18T21:31:12.148Z INFO 1680 --- [product-restfull] [ main] [ com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-09-18T21:31:12.979Z WARN 1680 --- [product-restfull] [ main] [ org.hibernate.orm.deprecation : HH00000025: PostgreSQLDialect does not
need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be selected by default)
2024-09-18T21:31:20.287Z INFO 1680 --- [product-restfull] [ main] [ o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000489: No JTA platform available (C
et 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2024-09-18T21:31:21.302Z INFO 1680 --- [product-restfull] [ main] [ j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory fo
r persistence unit 'default'
2024-09-18T21:31:24.903Z WARN 1680 --- [product-restfull] [ main] [ jpabase.configurations.JpaWebConfiguration : spring.jpa.open-in-view is enabled by
default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-09-18T21:31:30.306Z INFO 1680 --- [product-restfull] [ main] [ o.s.b.a.e.web.EndpointLinksResolver : Exposing 14 endpoints beneath base pat
'/actuator'
2024-09-18T21:31:31.052Z INFO 1680 --- [product-restfull] [ main] [ c.i.m.g.ProductRestfullApplicationTests : Started ProductRestfullApplicationTest
In 40.269 seconds (process running for 46.264)
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 46.99 s -- in com.ibam.master1.groupe8.ProductRestfullApplicationTests
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.4.2:jar (default-jar) @ product-restfull ---
[INFO] Building jar: D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull\target\product-restfull-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot:3.3.1:repackage (repackage) @ product-restfull ---
[INFO] Replacing main artifact D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull\target\product-restfull-0.0.1-SNAPSHOT.jar with repackaged archive, adding nested dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull\target\product-restfull-0.0.1-SNAPSHOT.jar.original
[INFO]
[INFO] --- install:3.1.2:install (default-install) @ product-restfull ---
[INFO] Installing D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull\pom.xml to C:\Users\Osie.lamoussa.m2\repository\com\ibam\master1\groupe8\product-restfull\0.0.1-SNAPSHOT\product-restfull-0
0.1-SNAPSHOT.jar
[INFO] Installing D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull\target\product-restfull-0.0.1-SNAPSHOT.jar to C:\Users\Osie.lamoussa.m2\repository\com\ibam\master1\groupe8\product-restfull
0.0.1-SNAPSHOT\product-restfull-0.0.1-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:33 min
[INFO] Finished at: 2024-09-18T21:31:43Z
[INFO] -----
D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull>
```

Figure 15: Résultat de la commande mvn clean install

• mvn spring-boot :run

```
C:\Windows\System32\cmd.exe - mvn spring-boot:run
OSBIBAMVM1
*****
:: Spring Boot :: (v3.3.1)

2024-09-18T21:34:41.782Z INFO 14340 --- [product-restfull] [ main] [ c.i.m.g.ProductRestfullApplication : Starting ProductRestfullApplication us
ing Java 17.0.2 with PID 14340 (D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull\target\classes started by Osie.lamoussa in D:\IBAM\VM1\S2\dev composant et web services\project\product-restfull)
2024-09-18T21:34:41.788Z INFO 14340 --- [product-restfull] [ main] [ c.i.m.g.ProductRestfullApplication : No active profile set, falling back to
1 default profile: 'default'
2024-09-18T21:35:00.908Z INFO 14340 --- [product-restfull] [ main] [ .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA reposi
ries in DEFAULT mode.
2024-09-18T21:35:01.116Z INFO 14340 --- [product-restfull] [ main] [ .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scan
ing in 109 ms. Found 1 JPA repository interface.
2024-09-18T21:35:02.666Z INFO 14340 --- [product-restfull] [ main] [ o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8081 (htt
p)
2024-09-18T21:35:02.759Z INFO 14340 --- [product-restfull] [ main] [ o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-09-18T21:35:02.760Z INFO 14340 --- [product-restfull] [ main] [ o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomc
e/10.1.25]
2024-09-18T21:35:03.006Z INFO 14340 --- [product-restfull] [ main] [ o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplic
ationContext
2024-09-18T21:35:03.012Z INFO 14340 --- [product-restfull] [ main] [ w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialize
tion completed in 21152 ms
2024-09-18T21:35:03.906Z INFO 14340 --- [product-restfull] [ main] [ o.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitI
fo [name: default]
2024-09-18T21:35:03.991Z INFO 14340 --- [product-restfull] [ main] [ org.hibernate.Version : HH0000412: Hibernate ORM core version
5.2.Final : HH0000026: Second-level cache disabled
2024-09-18T21:35:04.045Z INFO 14340 --- [product-restfull] [ main] [ o.h.c.internal.RegionFactoryInitiator : HH0000026: Second-level cache disabled
2024-09-18T21:35:04.476Z INFO 14340 --- [product-restfull] [ main] [ o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA
class transformer
2024-09-18T21:35:04.522Z INFO 14340 --- [product-restfull] [ main] [ com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-09-18T21:35:04.770Z INFO 14340 --- [product-restfull] [ main] [ com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection org.po
gresql.jdbc.PgConnection@508f4bb5
2024-09-18T21:35:04.774Z INFO 14340 --- [product-restfull] [ main] [ com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-09-18T21:35:04.837Z WARN 14340 --- [product-restfull] [ main] [ org.hibernate.orm.deprecation : HH00000025: PostgreSQLDialect does not
need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be selected by default)
2024-09-18T21:35:06.474Z INFO 14340 --- [product-restfull] [ main] [ o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000489: No JTA platform available (C
et 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2024-09-18T21:35:06.569Z INFO 14340 --- [product-restfull] [ main] [ j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory f
or persistence unit 'default'
2024-09-18T21:35:07.822Z WARN 14340 --- [product-restfull] [ main] [ jpabase.configurations.JpaWebConfiguration : spring.jpa.open-in-view is enabled by
default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-09-18T21:35:08.266Z INFO 14340 --- [product-restfull] [ main] [ o.s.b.a.e.web.EndpointLinksResolver : Exposing 15 endpoints beneath base pat
h '/actuator'
2024-09-18T21:35:08.878Z INFO 14340 --- [product-restfull] [ main] [ o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8081 (http) wit
h context path '/'
2024-09-18T21:35:08.979Z INFO 14340 --- [product-restfull] [ main] [ c.i.m.g.ProductRestfullApplication : Started ProductRestfullApplication in
27.749 seconds (process running for 28.511)
```

Figure 16: L'application spring boot démarré

5 Tests des endpoints avec Postman

Une fois l'application démarrée, utilisez Postman pour tester les différents endpoints de l'API REST :

5.1 GET <http://localhost:8081/api/produits>

Liste des produits paginés

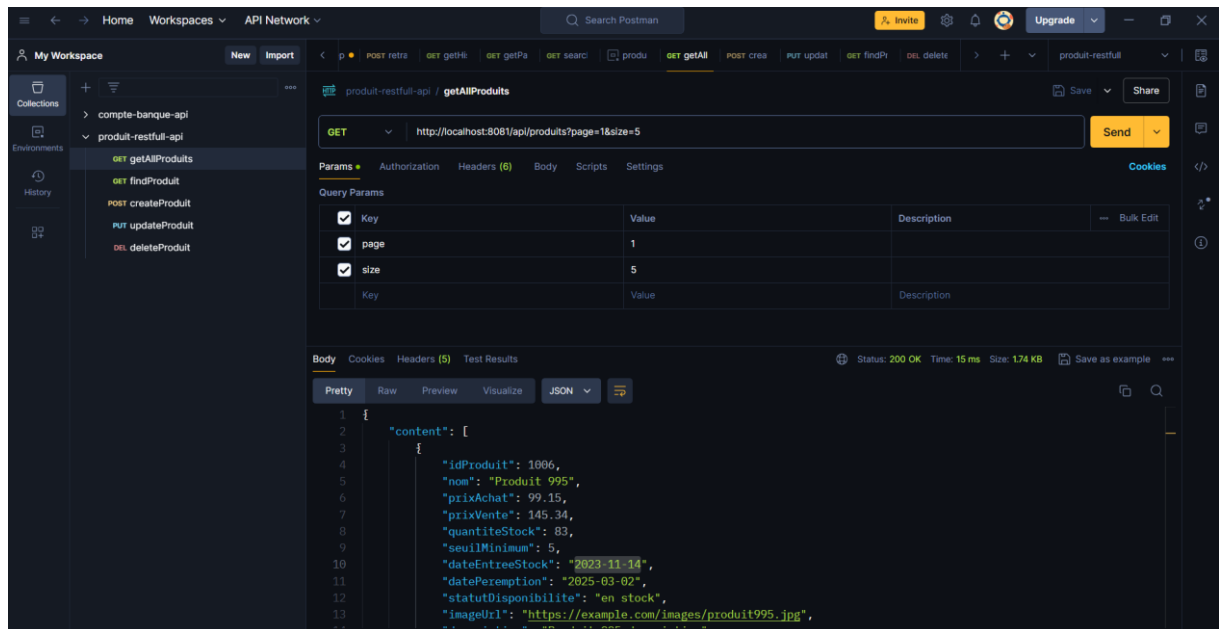


Figure 17 : Test d'affichage de la liste des produits

5.2 GET <http://localhost:8081/api/produits/1012>

Récupération du produit dont l'identifiant est 1012

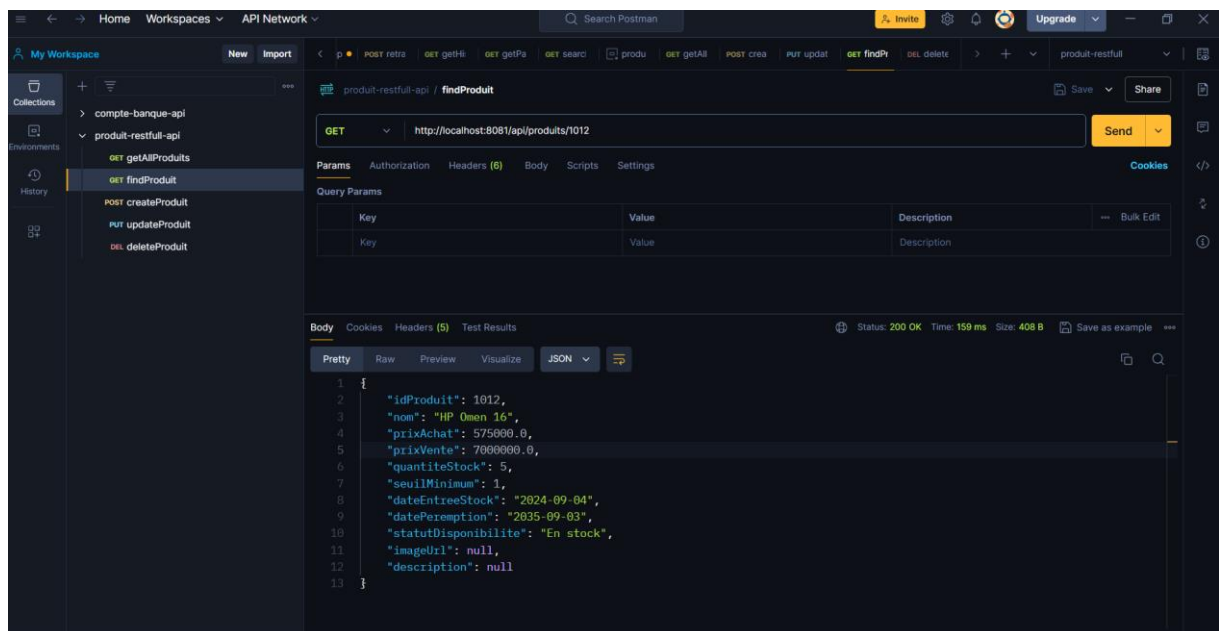


Figure 18: Test de récupération d'un produit

5.3 POST <http://localhost:8081/api/produits>

Creation de produit

Paramètre à fournir:

```

{
  "idProduit": null,
  "nom": "",
  "prixAchat": 0.0,
  "prixVente": 0.0,
  "quantiteStock": 00,
  "seuilMinimum": 00,
  "dateEntreeStock": "2024-09-14",
  "datePeremption": "2031-09-14",
  "statutDisponibilite": "En stock",
  "imageUrl": null,
  "description": null
}

```

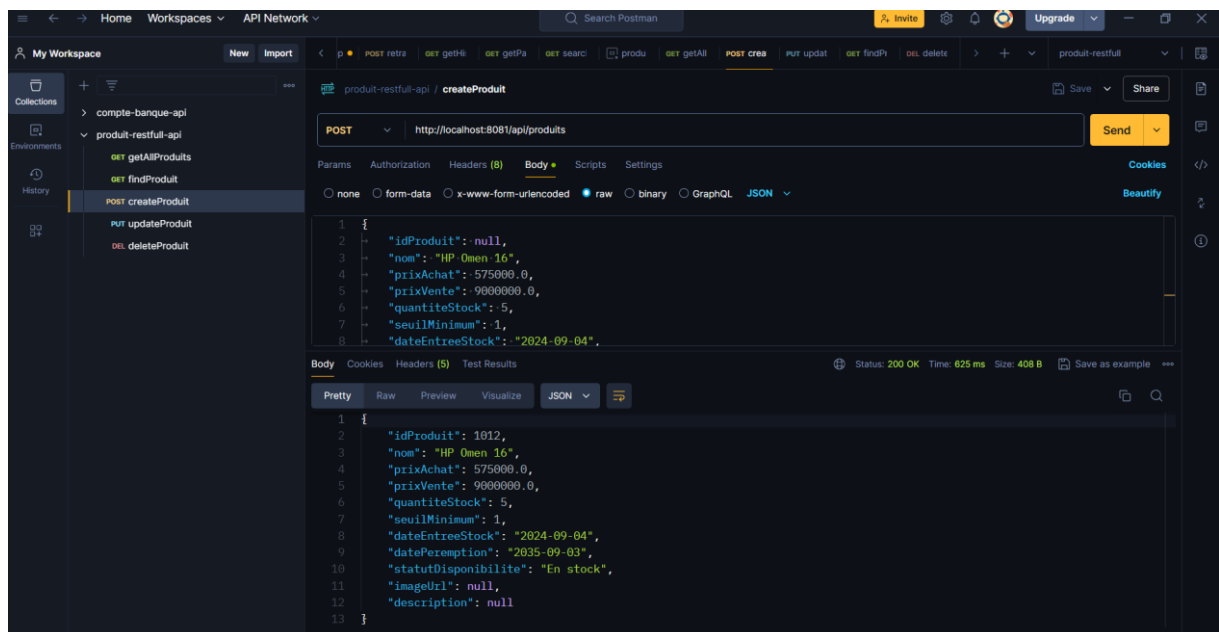


Figure 19: Test de création d'un produit

5.4 PUT <http://localhost:8081/api/produits/1012>

Modification du produit don't l'identifiant est 1012

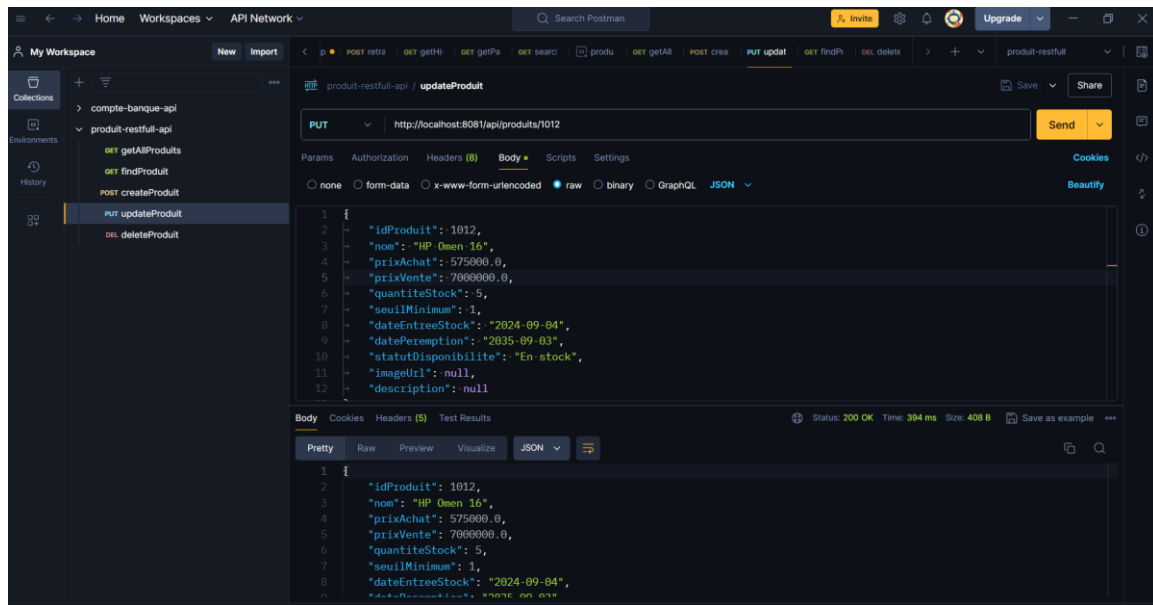


Figure 20: Test de modification d'un produit

5.5 DELETE <http://localhost:8081/api/produits/1012>

Suppression du produit dont l'identifiant est 1012

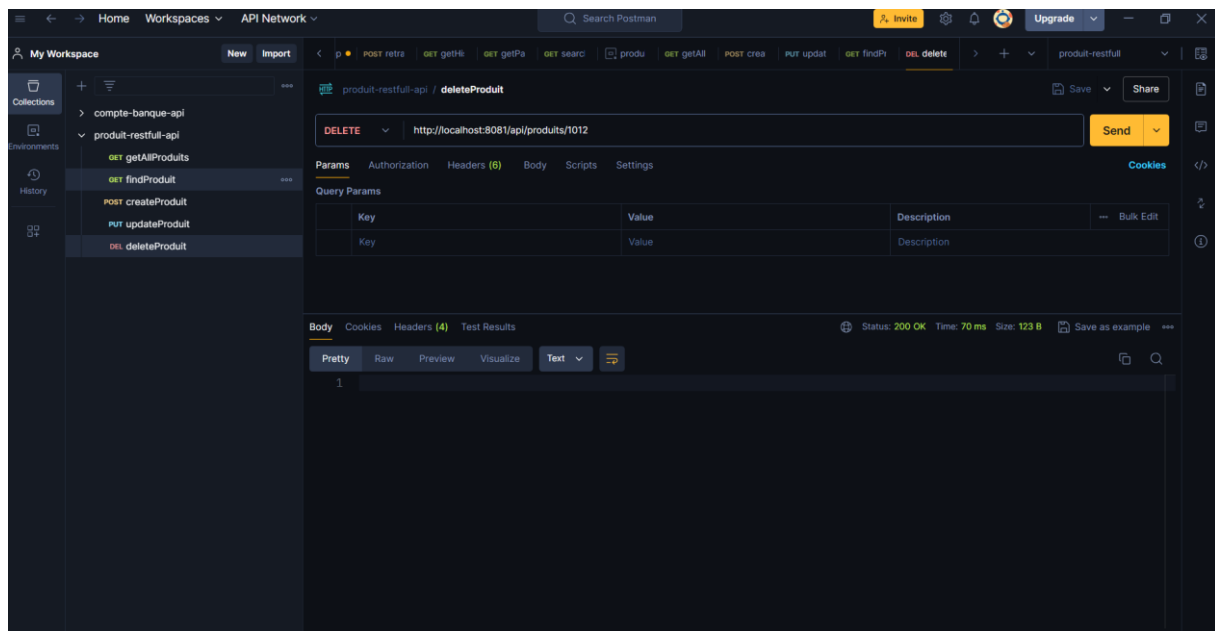


Figure 21 : Test de suppression d'un produit

6 Monitoring de l'application

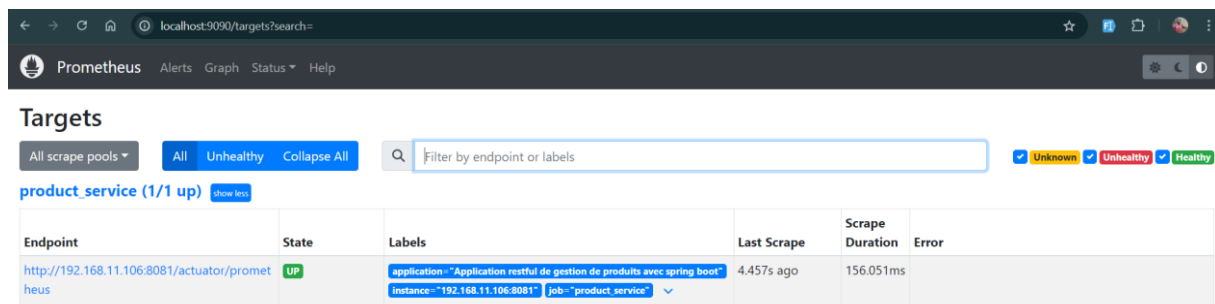


Figure 22 : Prometheus connecté à notre application

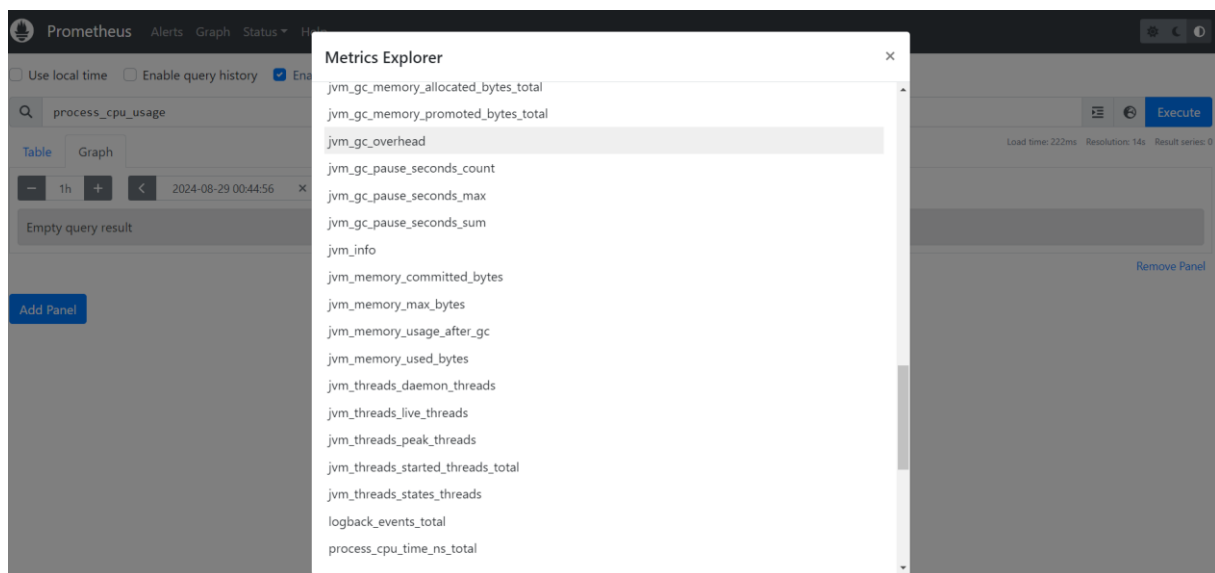


Figure 23 : Les métriques de Prometheus

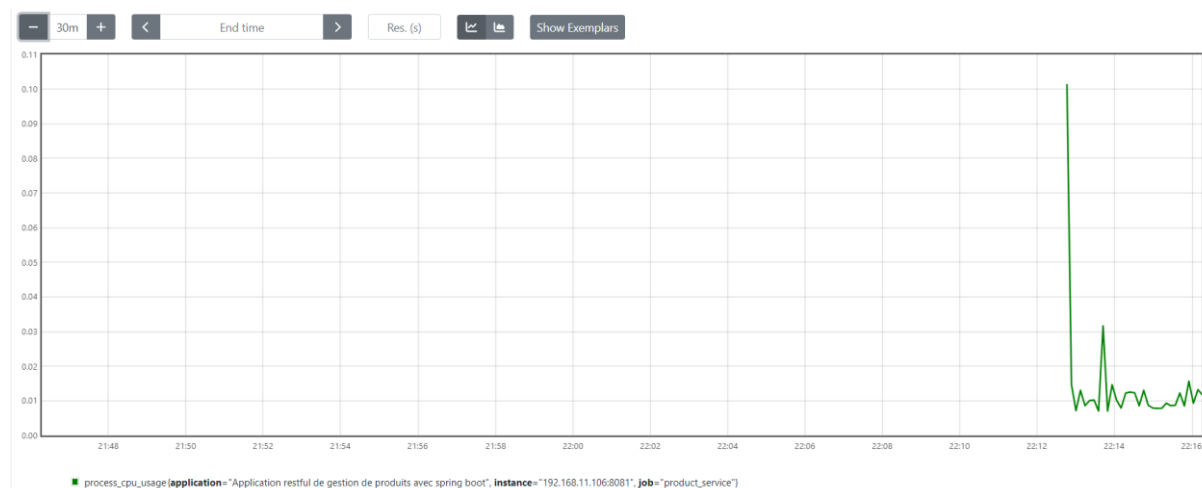


Figure 24 : Evolution de l'usage du CPU

7 Résultats

Les résultats obtenus à l'issue du développement et des tests de l'application RESTful se présentent comme suit :

- API REST fonctionnelle : Les endpoints REST permettent de récupérer et manipuler les données produites avec succès.

- Intégration des données via Talend Open Studio
- Monitoring : Prometheus collecte et affiche les métriques de l'application telles que le nombre de requêtes HTTP, l'utilisation de la mémoire, et les temps de réponse.
- Tests réussis : Les requêtes via Postman ont montré que l'API fonctionne correctement avec les différentes opérations CRUD.

Conclusion

Le projet a permis de développer une application RESTful fiable, capable de communiquer avec une base de données PostgreSQL contenant les données transformées à l'aide de Open Talend Studio. Grâce à l'intégration de Prometheus, le monitoring des performances et de l'utilisation des ressources est assuré, permettant une surveillance continue de l'application en production.

À l'avenir, plusieurs pistes d'amélioration pourraient être envisagées pour optimiser et enrichir cette application RESTful. Tout d'abord, l'implémentation d'un mécanisme d'authentification basé sur OAuth 2.0 ou JWT permettrait de renforcer la sécurité de l'API, en assurant un accès contrôlé et sécurisé aux ressources, surtout dans un contexte où les données sensibles sont en jeu. Cette démarche garantirait que seules les requêtes authentifiées et autorisées puissent interagir avec l'API, répondant ainsi aux besoins croissants de protection des données.

Ensuite, l'introduction d'un pipeline d'intégration et de déploiement continu (CI/CD) pourrait grandement faciliter la gestion des versions et la mise à jour de l'application. Un tel pipeline permettrait d'automatiser les tests et le déploiement à chaque changement de code, réduisant ainsi les risques d'erreur humaine et accélérant la mise en production de nouvelles fonctionnalités ou correctifs.

Enfin, l'intégration d'une solution de visualisation des métriques, telle que Grafana, couplée à Prometheus, pourrait apporter une meilleure visibilité sur les performances et l'état de santé de l'application. Grafana offrirait une interface graphique intuitive pour visualiser les données collectées par Prometheus, permettant ainsi aux équipes techniques de surveiller les ressources consommées, d'identifier rapidement les anomalies, et d'améliorer la réactivité en cas de problèmes en production.

Ces évolutions contribueraient à rendre l'application plus robuste, sécurisée et facile à maintenir, tout en offrant une meilleure expérience utilisateur dans le cadre de sa gestion et de son déploiement.