

**UNIVERSITE JOSEPH
KI-ZERBO
(UJKZ)**



**INSTITUT BURKINABE
DES ARTS ET METIERS
(IBAM)**



Master 1 informatique

Module : DEVELOPPEMENT DE BASE DE COMPOSANT ET SERVICES WEB

Rapport de Projet :

**Application RESTful avec Spring Boot, Talend Open
Studio et Prométhéus**

GROUPE 2

Professeur : M KY OUMAR

Membres du groupe 2:

- ***TRAORE /KABORE Hawa***
- ***YOUGBARE Simon***
- ***SAWADOGO Ernestine***
- ***OUEDRAOGO Latifatou***
- ***KOANDA Moustapha***

Année Académique 2023-2024

SOMMAIRE

Table des matières

I. Généralités	3
1. Présentation	3
2. Objectifs	3
3. Tâches	3
II. Outils et Technologies.....	3
1. Technologies Utilisées	3
2. Environnement de Développement	4
3. Répertoire Git	4
III. Développement de l'application	4
1. Extraction et Transformation de Données avec Talend Open Studio	4
2. Création de l'API RESTful avec Spring Boot.....	6
3. Monitoring avec Prometheus.....	9
4. Tests avec Postman	10
IV. Procédure d'installation et de test	10
1. Prérequis	10
2. Installation de l'application	11
3. Monitoring avec Prometheus.....	12
4. Test avec Postman.....	14
Conclusion.....	17

I. Généralités

1. Présentation

Le présent projet est réalisé dans le cadre du développement d'une application RESTful utilisant Spring Boot pour exposer des données extraites et transformées par Talend Open Studio à partir d'une base de données MySQL. Le monitoring des métriques est assuré via Prometheus.

Le choix du groupe s'est porté sur le thème « Gestion de la clientèle pour une entreprise informatique. »

2. Objectifs

L'objectif principal du projet est de créer une application RESTful qui permet :

- L'extraction et la transformation de données de produits à partir d'une base de données MySQL/PostgreSQL via Talend Open Studio.
- L'exposition de ces données via une API RESTful développée avec Spring Boot.
- Le monitoring des performances et des métriques via Prometheus.
- Le test de l'application à l'aide de Postman et l'analyse des métriques dans Prometheus.

3. Tâches

Utilisez Talend Open Studio pour extraire et transformer des données de produits dans MySQL/PostgreSQL.

- Développez une application Spring Boot pour exposer ces données via une API RESTful.
- Intégrez des métriques de Prometheus dans l'application pour le monitoring.
- Testez l'application avec Postman et surveillez les métriques avec Prometheus.

II. Outils et Technologies

1. Technologies Utilisées

- **Spring Boot** (version 2.7.18): Pour le développement de l'API RESTful.
- **Talend Open Studio** (TOS_DI-20211109_1610-V8.0.1) pour extraire et transformer des données dans MySQL.
- **JDK 11** : pour la version de java compatible avec TOS.
- **MySQL** : Base de données utilisée pour stocker les données de produits.
- **Prometheus** (Prometheus-2.54.0.windows-amd64) pour le monitoring des métriques.
- **Postman** : Outil utilisé pour tester l'API RESTful.

2. Environnement de Développement

- **IDE** : VS Code
- **Outils de versionnement** : Git avec hébergement sur GitHub
- **Base de données** : MySQL
- **Serveur d'applications** : Tomcat intégré à Spring Boot
- **Outils de tests** : Postman pour tester l'API REST

3. Répertoire Git

Le projet est disponible publiquement sur un dépôt Git afin de faciliter l'accès, le téléchargement et les tests :

- Lien Git : https://github.com/Yougsim/Developpement-base-de-composants-et-de-services-web_Projet-SpringBoot_Talend.git

Le répertoire contient :

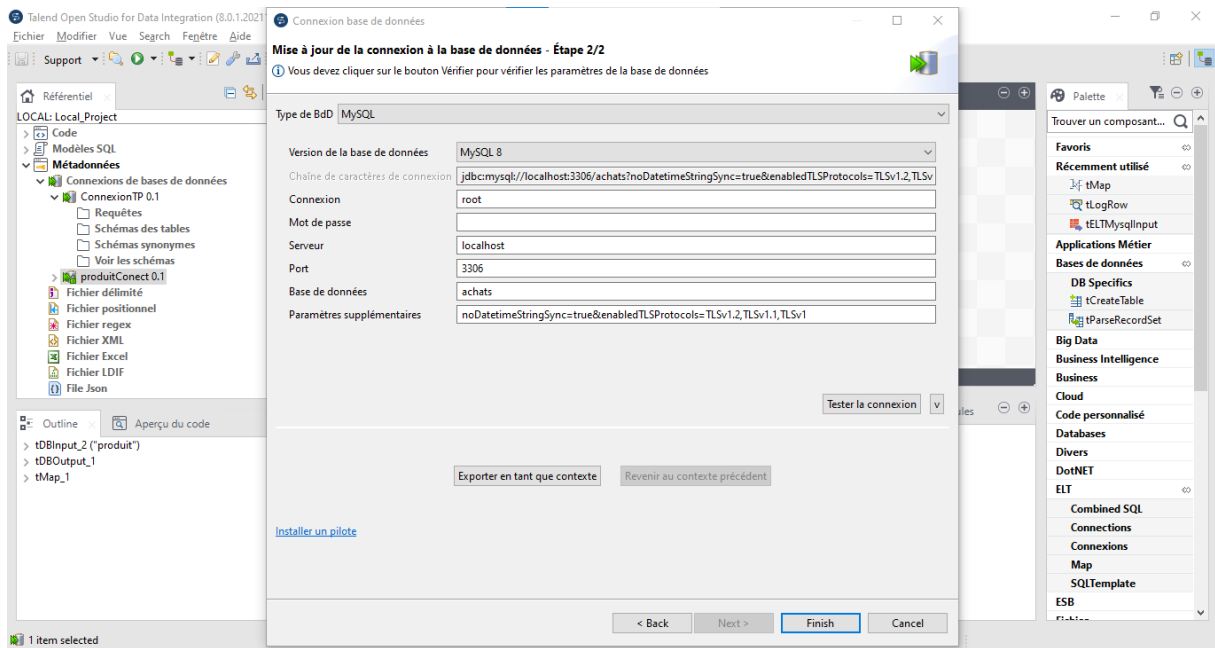
- Le code source de l'application Spring Boot.
- Les fichiers de configuration pour Talend Open Studio.
- Les fichiers de monitoring pour Prometheus.
- La procédure d'installation et de test.
- La présentation PowerPoint du projet.
- Le rapport du projet.

III. Développement de l'application

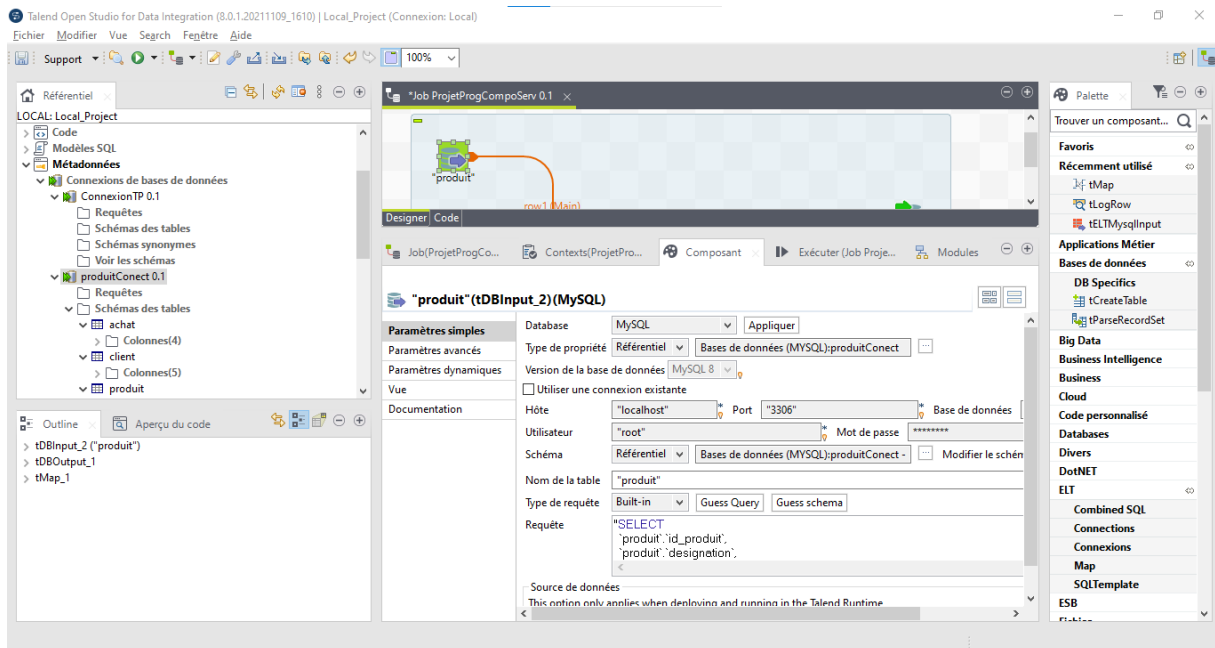
1. Extraction et Transformation de Données avec Talend Open Studio

L'outil Talend Open Studio a été utilisé pour connecter la base de données MySQL et extraire les informations relatives aux produits. Le flux ETL a permis de transformer les données avant de les insérer dans la base de données cible.

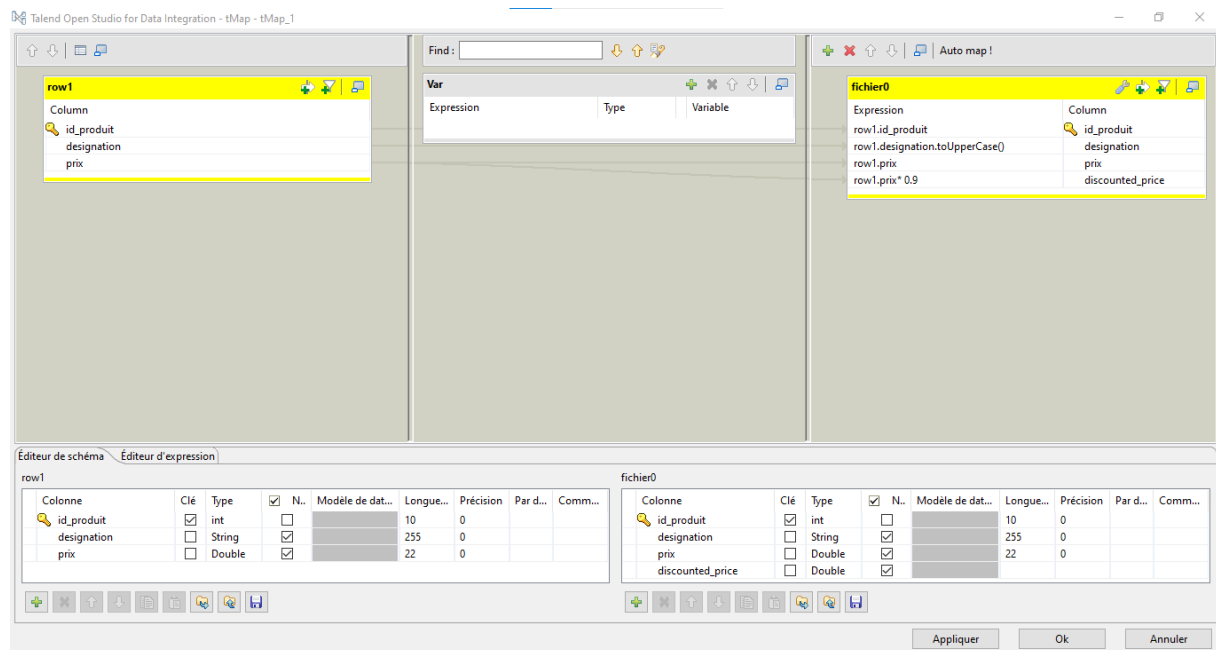
- Connexion à la base de données : cela a été fait avec le sous onglet « connexion de bases de données » de l'onglet « métadonnée » dans lequel on a créé notre connexion « produitconnect 0.1 »



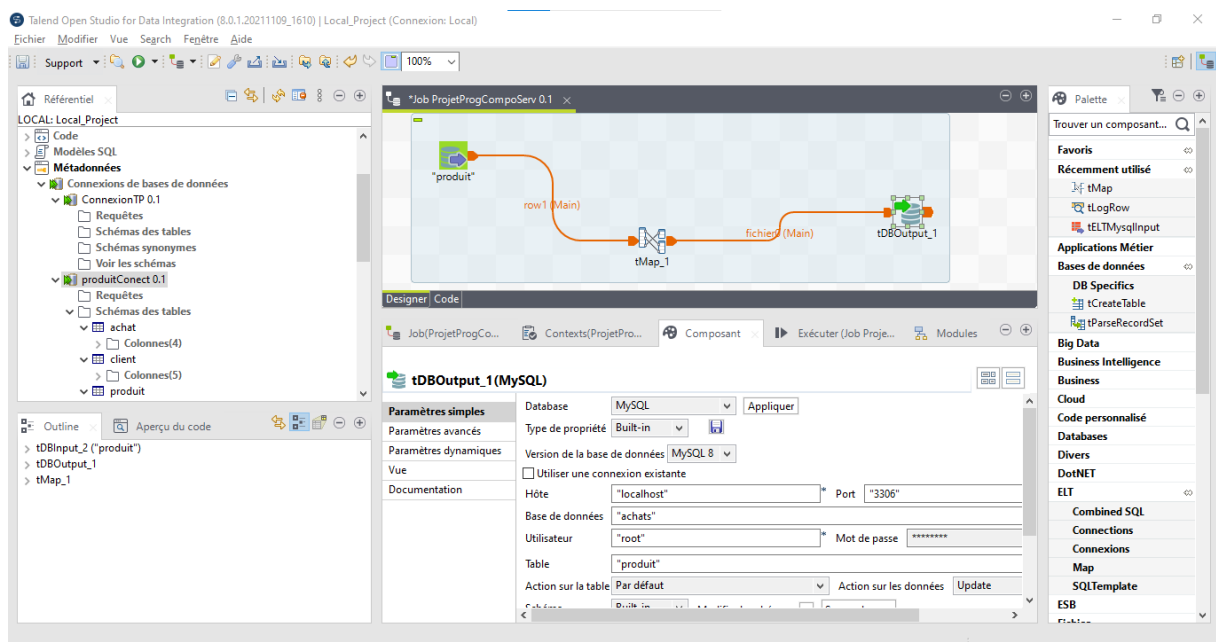
- Extraction des données avec tDBInput :



- Transformation des données avec Tmap dans lequel nous avons ajouté une colonne « discounted_price » dans notre base de données qui permet de faire une réduction



- Insertion des données dans la base de données avec tDBOutput



2. Création de l'API RESTful avec Spring Boot

L'API RESTful a été développée avec Spring Boot pour exposer les données de produits transformées via les points d'API suivants :

- **GET** `http://localhost:8081/allProduits`: Retourne la liste de tous les produits.

```
• @GetMapping("/allProduits")
• public ResponseEntity<List<Produit>> getAllProduits() {
•     try {
•         List<Produit> ProduitList = new ArrayList<>();
•         produitRepo.findAll().forEach(ProduitList::add);
•
•         if (ProduitList.isEmpty()) {
•             return new ResponseEntity<>(HttpStatus.NO_CONTENT);
•         }
•
•         return new ResponseEntity<>(ProduitList, HttpStatus.OK);
•     } catch (Exception ex) {
•         return new
• ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
•     }
• }
```

- **GET** `http://localhost:8081/showProduit/{id}` Retourne les détails d'un produit spécifique.

```
• @GetMapping("/showProduit/{id}")
• public ResponseEntity<Produit> showProduit(@PathVariable(value =
• "id") Integer id) {
•
•     Optional<Produit> optional = produitRepo.findById(id);
•     Produit produit = null;
•     if (optional.isPresent()) {
•         produit = optional.get();
•     } else {
•         throw new RuntimeException(" product not found for id :: " +
• id);
•     }
•
•     return new ResponseEntity<>(produit, HttpStatus.OK);
• }
•
```

- **POST** <http://localhost:8081/addProduit>: Permet d'ajouter un nouveau produit.

```
• @PostMapping("/addProduit")
•     public ResponseEntity<Produit> addProduit(@RequestBody Produit
Produit) {
•         Produit produitObj = produitRepo.save(Produit);
•
•         return new ResponseEntity<>(ProduitObj, HttpStatus.OK);
•     }
```

- **PUT** <http://localhost:8081/produit/{id}>: Met à jour les informations d'un produit.

```
• @PutMapping("/produit/{id}")
•     public ResponseEntity<Produit> updateProduit(@PathVariable Integer
id, @RequestBody Produit updateProduit) {
•         try {
•             Produit updated = produService.updateProduit(id,
updateProduit);
•             return ResponseEntity.ok(updated);
•         } catch (Exception e) {
•             return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
•         }
•     }
```

- **DELETE** <http://localhost:8081/deleteProduit/{id}>: Supprime un produit.

```
• @DeleteMapping("/deleteProduit/{id}")
•     public ResponseEntity<Void> deleteProduit(@PathVariable Integer id)
{
•         produitRepo.deleteById(id);
•         return ResponseEntity.noContent().build();
•     }
```


3. Monitoring avec Prometheus

Prometheus, un outil de monitoring open-source, il a été choisi pour ce projet en raison de sa capacité à collecter et à stocker des métriques de manière efficace, ainsi que son intégration native avec Spring Boot. Il sera chargé dans notre application de surveiller les performances du système et collecter les métriques des services exposés via une API RESTful. Ces métriques fournies permettront de suivre l'utilisation des ressources, la latence des requêtes, et d'autres indicateurs essentiels pour assurer la stabilité et la fiabilité de l'application.

- Afin de permettre à Prometheus de collecter des métriques depuis l'application Spring Boot, nous avons utilisé **Micrometer**, qui est une bibliothèque de surveillance intégrée avec Spring Boot Actuator. Les dépendances suivantes ont été ajoutées au fichier pom.xml du projet :

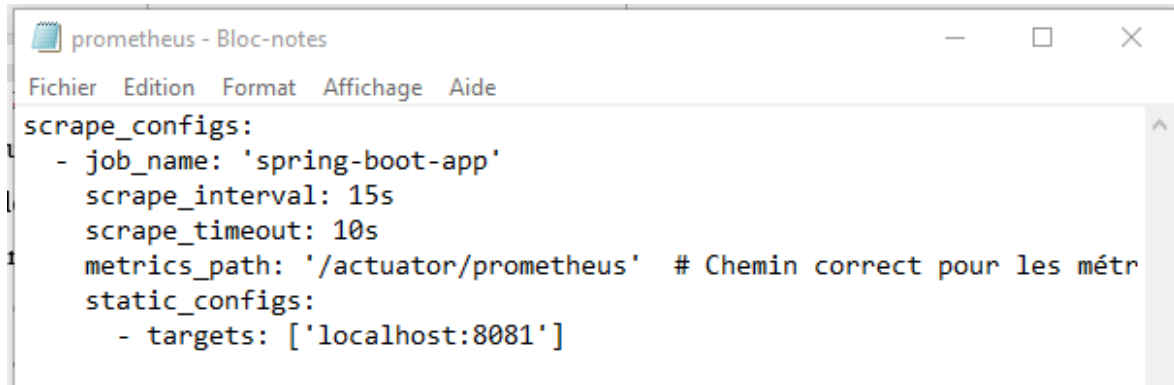
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

- **Configuration d'Actuator** : Pour exposer les métriques via une API RESTful compatible avec Prometheus, le module **Spring Boot Actuator** a été configuré dans le fichier application.properties comme suit :

- management.endpoints.web.exposure.include=health,info,prometheus
- management.endpoint.prometheus.enabled=true

Cela expose les métriques au format Prometheus à l'URL : /actuator/prometheus.

- **Configuration de Prometheus** : Une fois l'application prête à fournir les métriques, nous avons configuré **Prometheus** pour collecter ces données. Le fichier de configuration `prometheus.yml` a été ajusté pour scraper les métriques de l'application sur le port 8081 :



```
prometheus - Bloc-notes
Fichier Edition Format Affichage Aide
scrape_configs:
- job_name: 'spring-boot-app'
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: '/actuator/prometheus' # Chemin correct pour les métr
  static_configs:
    - targets: ['localhost:8081']
```

- **Déploiement de Prometheus** : Après avoir téléchargé Prometheus depuis son site officiel, il a été installé et lancé en utilisant la commande suivante depuis le répertoire d'installation :
- ```
./prometheus.exe --config.file=prometheus.yml
```

#### 4. Tests avec Postman

**Postman** est une application largement utilisée pour effectuer des tests API, permettant de simuler des requêtes HTTP et de valider les réponses des services web. Il offre une interface conviviale pour tester les fonctionnalités de notre application Spring Boot avant son déploiement en production.

Ce qui nous a permis de tester et valider les endpoints de l'API exposée par notre application Spring Boot.

- Envoyer des requêtes HTTP (GET, POST, PUT, DELETE) à l'API,
- Ajout, mise à jour et suppression de produits. Les tests ont permis de valider les performances et la précision des données exposées.

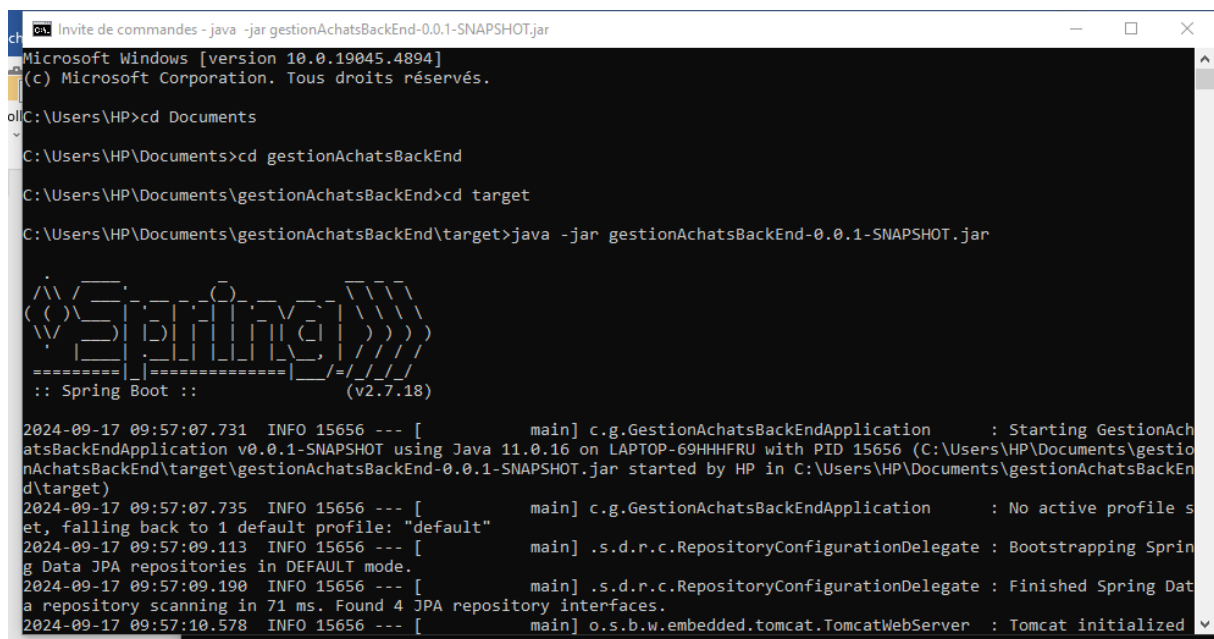
## IV. Procédure d'installation et de test

### 1. Prérequis

- Java JDK 11+ installé
- MySQL installé et configuré
- Talend Open Studio installé
- Prometheus installé
- Postman pour le test des requêtes API

## 2. Installation de l'application

- ✓ Clonez le répertoire Git : `git clone [https://github.com/Yougsim/Developpement-base-de-composants-et-de-services-web_Projet-SpringBoot_Talend.git]`
- Télécharger et Installer JDK 11 :
  - Télécharger JDK11 depuis Oracle ou AdoptOpenJDK.
  - Installer JDK11
- Téléchargement du Projet :
  - Télécharger le fichier ZIP du projet depuis le dépôt Git.
  - Dézipper le fichier dans le répertoire de votre choix.
- Lancement de l'Application :
  - Ouvrer un terminal ou une invite de commandes.
  - Naviguer jusqu'au répertoire `gestionAchatsBackEnd\target`.
  - Exécutez « : `java -jar gestionAchatsBackEnd-0.0.1-SNAPSHOT.jar` » pour lancer l'application



```
Microsoft Windows [version 10.0.19045.4894]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\HP>cd Documents
C:\Users\HP\Documents>cd gestionAchatsBackEnd
C:\Users\HP\Documents\gestionAchatsBackEnd>cd target
C:\Users\HP\Documents\gestionAchatsBackEnd\target>java -jar gestionAchatsBackEnd-0.0.1-SNAPSHOT.jar

:: Spring Boot ::
(v2.7.18)

2024-09-17 09:57:07.731 INFO 15656 --- [main] c.g.GestionAchatsBackEndApplication : Starting GestionAchatsBackEndApplication v0.0.1-SNAPSHOT using Java 11.0.16 on LAPTOP-69HHHFRU with PID 15656 (C:\Users\HP\Documents\gestionAchatsBackEnd\target\gestionAchatsBackEnd-0.0.1-SNAPSHOT.jar started by HP in C:\Users\HP\Documents\gestionAchatsBackEnd\target)
2024-09-17 09:57:07.735 INFO 15656 --- [main] c.g.GestionAchatsBackEndApplication : No active profile set, falling back to 1 default profile: "default"
2024-09-17 09:57:09.113 INFO 15656 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-09-17 09:57:09.190 INFO 15656 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 71 ms. Found 4 JPA repository interfaces.
2024-09-17 09:57:10.578 INFO 15656 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized
```

- Configuration de la Base de Données
  - Installer le SGBD MySQL
  - Créer une base de données "achats"

Configurez les accès à la base de données MySQL dans le fichier `application.properties`.

```
spring.application.name=gestionAchatsBackEnd
#developpement
spring.datasource.url=jdbc:mysql://localhost:3306/achats
spring.datasource.username=root
```

```

spring.datasource.password=

#production
#spring.datasource.url=jdbc:mysql://192.168.33.12:3306/achats
#spring.datasource.username=groupe2
#spring.datasource.password=1234
#spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

```

### 3. Monitoring avec Prometheus

#### Configuration de Prometheus

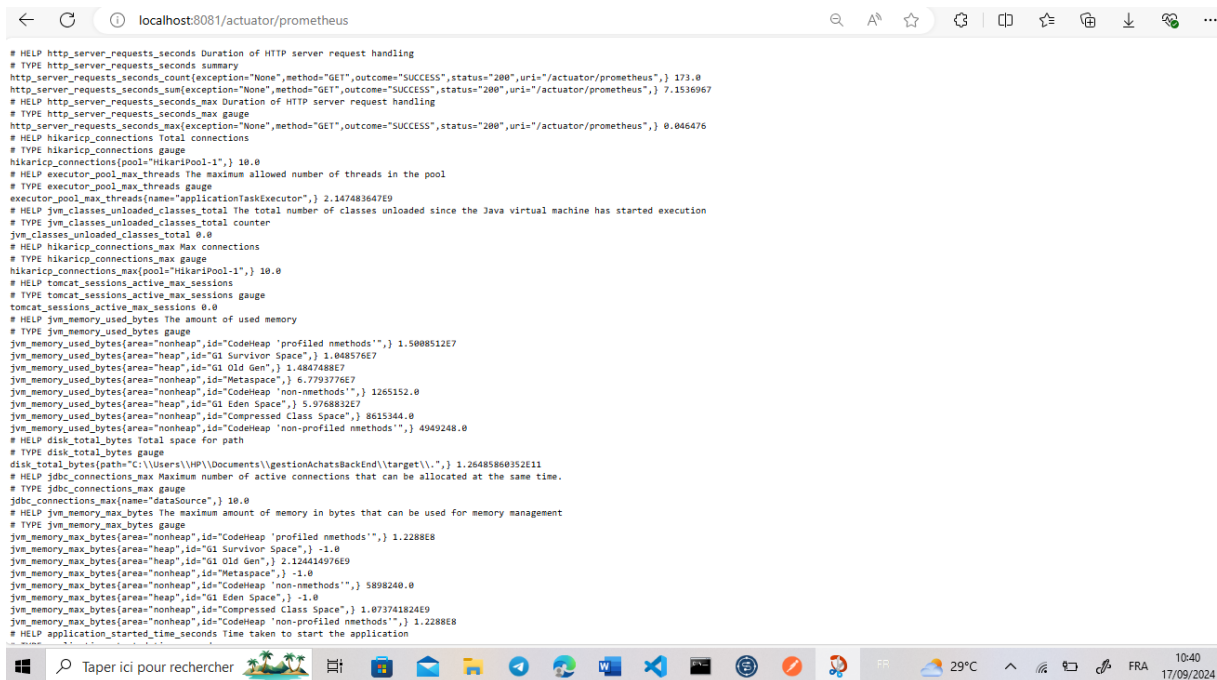
- Téléchargement et Décompression :
  - Télécharger prometheus-2.54.0.windows-amd64.zip depuis le dépôt Git.
  - Dézipper le dossier dans un répertoire de votre choix
- Lancement de Prometheus :
  - Ouvrir un terminal et naviguez jusqu'au répertoire contenant prometheus.yml et prometheus.exe.
  - Exécuter la commande suivante pour démarrer Prometheus : 'prometheus.exe --config.file=prometheus.yml'

```

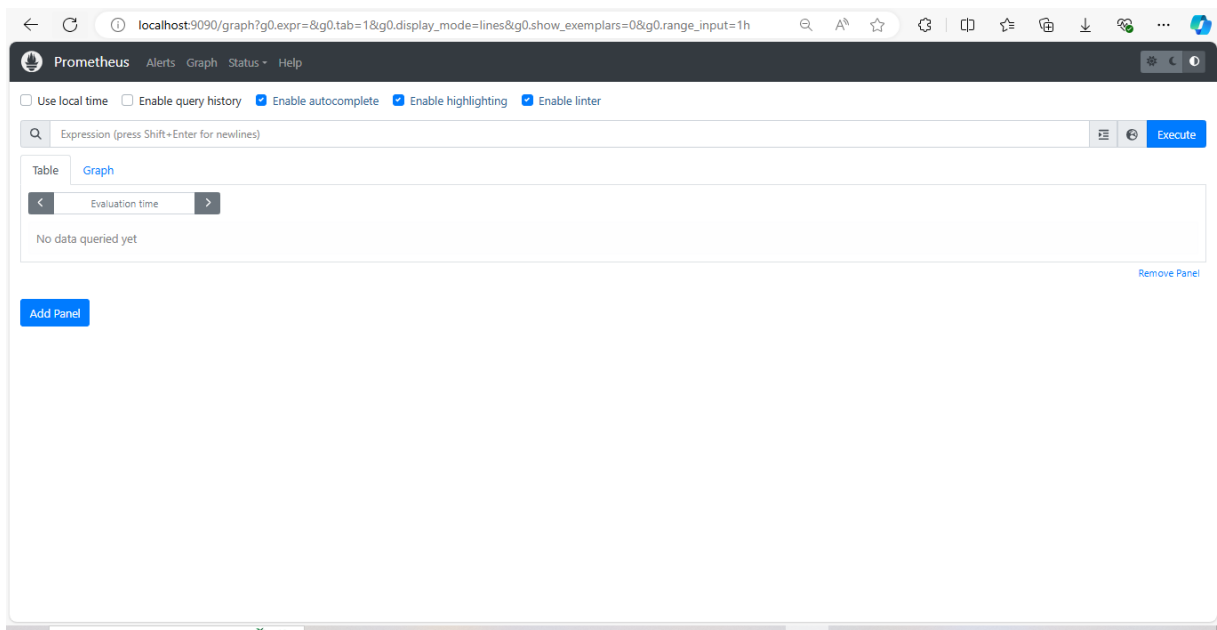
C:\Users\HP\Documents\prometheus-2.54.0.windows-amd64\>prometheus.exe --config.file=prometheus.yml
ts=2024-09-17T09:55:48.002Z caller=main.go:601 level=info msg="No time or size retention was set so using the default time retention" duration=15s
ts=2024-09-17T09:55:48.003Z caller=main.go:645 level=info msg="Starting Prometheus Server" mode=server version="(version=2.54.0, branch=HEAD, revision=5354e87a70d3eb26b81b601b286d66ff983990f6)"
ts=2024-09-17T09:55:48.004Z caller=main.go:650 level=info build_context="(go=go1.22.6, platform=windows/amd64, user=root@55f08ee1c009, date=20240809-11:38:49, tags=builtinassets,stringlabels)"
ts=2024-09-17T09:55:48.006Z caller=main.go:651 level=info host_details="(windows)"
ts=2024-09-17T09:55:48.007Z caller=main.go:652 level=info fd_limits=N/A
ts=2024-09-17T09:55:48.007Z caller=main.go:653 level=info vm_limits=N/A
ts=2024-09-17T09:55:48.020Z caller=web.go:571 level=info component=web msg="Start listening for connections" address=0.0.0.0:9090
ts=2024-09-17T09:55:48.022Z caller=main.go:1160 level=info msg="Starting TSDB ..."
ts=2024-09-17T09:55:48.024Z caller=tls_config.go:313 level=info component=web msg="Listening on" address=[::]:9090
ts=2024-09-17T09:55:48.029Z caller=tls_config.go:316 level=info component=web msg="TLS is disabled." http2=false address=[::]:9090
ts=2024-09-17T09:55:48.037Z caller=repair.go:56 level=info component=tsdb msg="Found healthy block" mint=1725013796034 maxt=1725033600000 ulid=01J6SM09BKR24BS7P7RKPMVHFJ
ts=2024-09-17T09:55:48.050Z caller=repair.go:56 level=info component=tsdb msg="Found healthy block" mint=1725273941035 maxt=1725292800000 ulid=01J7R53E3X0N1TBRG7YAHDGJ4C
ts=2024-09-17T09:55:48.062Z caller=repair.go:56 level=info component=tsdb msg="Found healthy block" mint=1726313771042 maxt=1726315200000 ulid=01J7RFCYXX98YWNISKY5VY7WX61
ts=2024-09-17T09:55:48.070Z caller=repair.go:56 level=info component=tsdb msg="Found healthy block" mint=1726315200000 maxt=1726322400000 ulid=01J7RGRXSZTHPJBEXXM8GHZSKE
ts=2024-09-17T09:55:48.079Z caller=repair.go:56 level=info component=tsdb msg="Found healthy block" mint=1725352136042 maxt=1725364800000 ulid=01J7RGRXADVGSRDWS44FNK3RY0
ts=2024-09-17T09:55:48.081Z caller=dir_locker.go:77 level=warn component=tsdb msg="A lockfile from a previous execution already existed. It was replaced" file=C:\Users\HP\Documents\prometheus-2.54.0.windows-amd64\data\lock
ts=2024-09-17T09:55:48.252Z caller=head.go:626 level=info component=tsdb msg="Replaying on-disk memory mappable chunks if any"

```

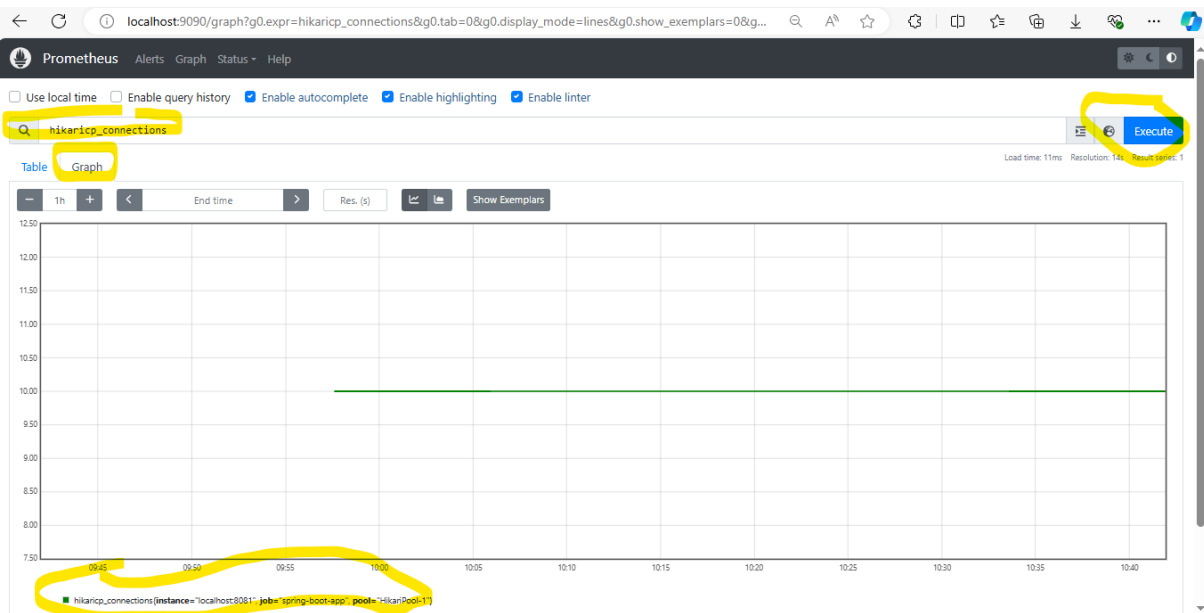
-Ouvrer votre navigateur et allez à l'adresse 'localhost:8081/actuator/prometheus' pour vérifier les métriques.



-Accéder à l'interface web de Prometheus en tapant 'http://localhost:9090/' dans votre navigateur.



- Dans l'onglet 'Graph', saisir les requêtes dans le champ de requête PromQL
- Cliquer sur Exécute pour voir les requêtes en cours d'exécution sur votre application



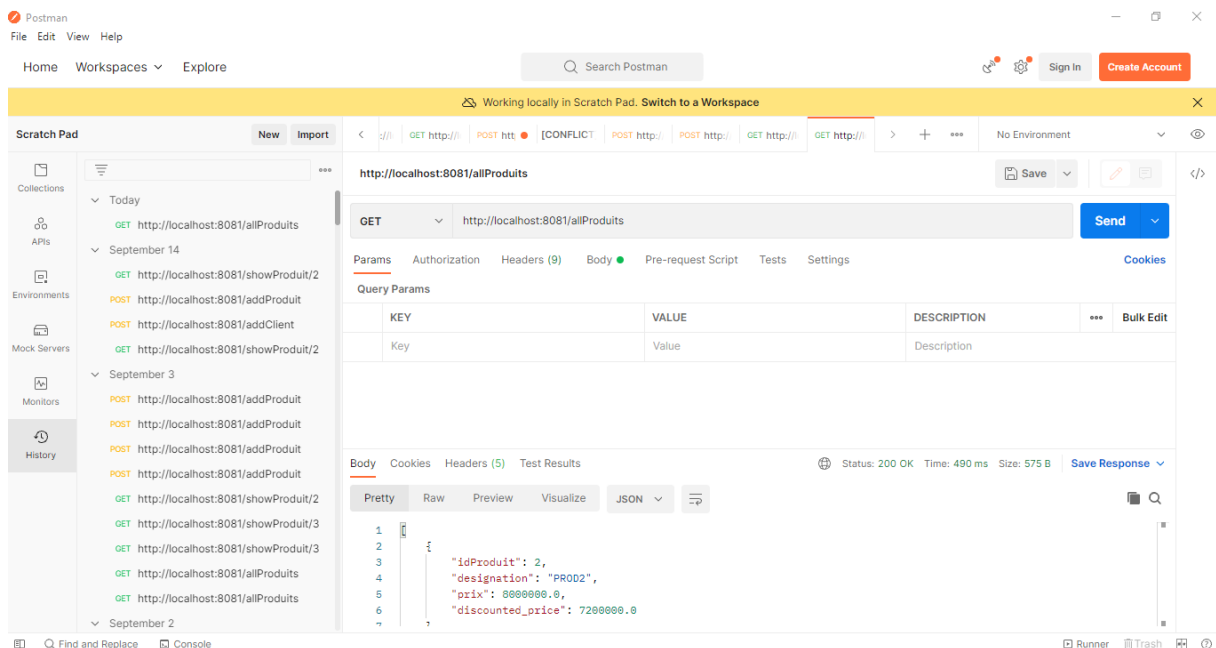
#### 4. Test avec Postman

Ouvrez Postman.

#### Tests de l'Application avec Postman

Test des endpoints de l'API pour tester l'application :

- ❑ GET : <http://localhost:8081/allProduits> (Récupère la liste de tous les produits).



❑ GET : <http://localhost:8081/showProduit/3> (Affiche les détails du produit avec l'ID 3).

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8081/showProduit/2`
- Method:** GET
- Response Status:** 200 OK, Time: 20 ms, Size: 247 B
- Response Body (JSON):**

```
{
 "idProduit": 2,
 "designation": "PROD2",
 "prix": 8000000.0,
 "discounted_price": 7200000.0
}
```

❑ POST : <http://localhost:8081/addProduit> (Ajoute un nouveau produit).

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8081/addProduit`
- Method:** POST
- Request Body (JSON):**

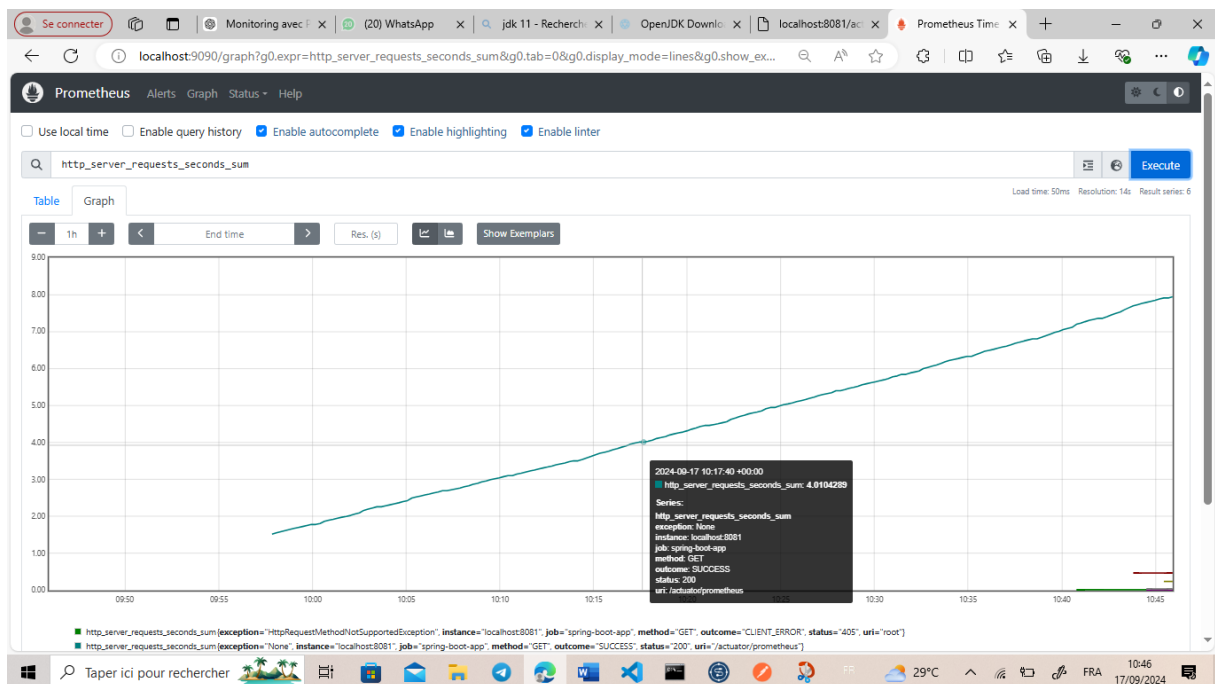
```
{
 "designation": "PROD2",
 "prix": 500000.0
}
```
- Response Status:** 200 OK, Time: 245 ms, Size: 241 B
- Response Body (JSON):**

```
{
 "idProduit": 8,
 "designation": "PROD2",
 "prix": 500000.0,
 "discounted_price": null
}
```

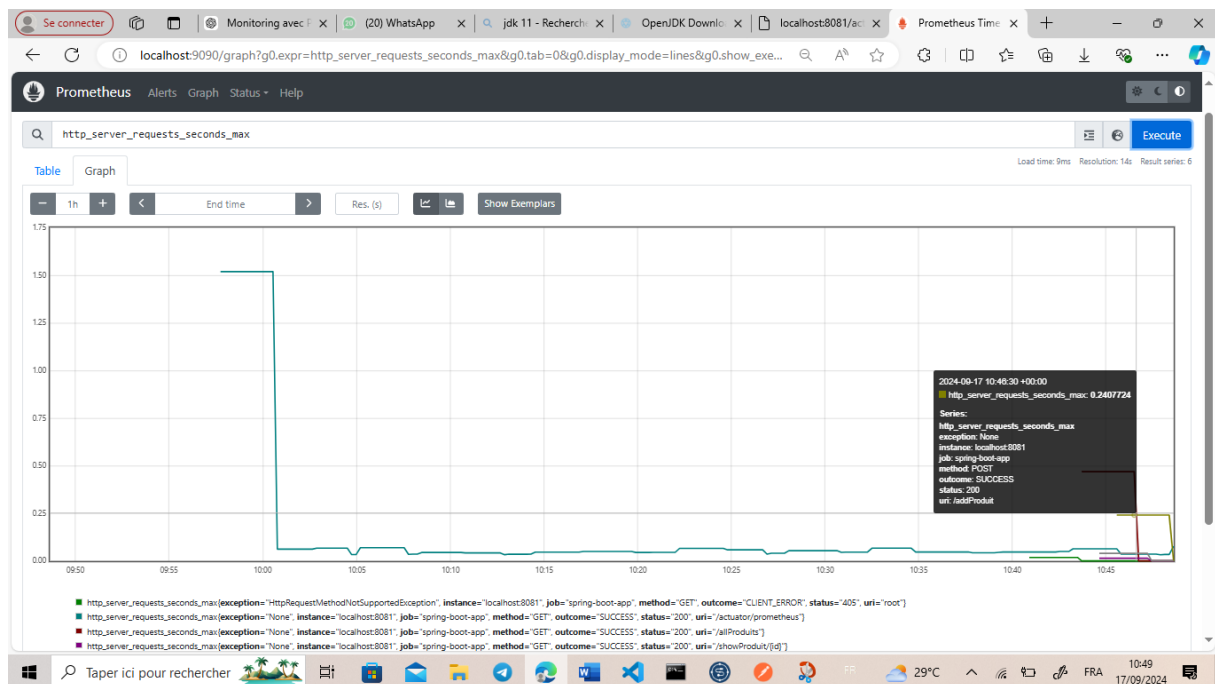
## Requêtes Prometheus :

Utilisation des requêtes suivantes dans l'interface Prometheus :

❑ `http_server_requests_seconds_sum`

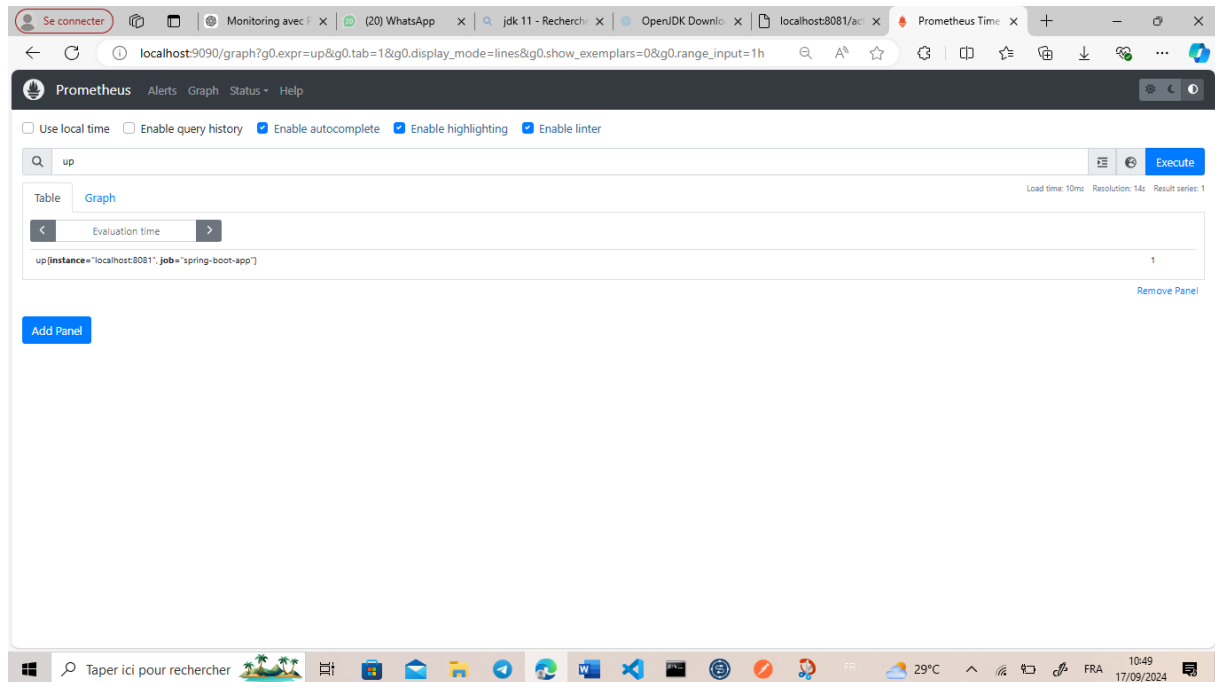


❑ `http_server_requests_seconds_max`





❑ up pour verifier que le prometheus marche avec le projet



## Conclusion

Le développement de cette application RESTful a permis de mettre en pratique différentes technologies, notamment Spring Boot, Talend Open Studio et Prometheus. Ce projet a permis de renforcer les compétences de l'équipe en intégration de systèmes, en gestion des API, et en monitoring d'applications.