TP1 : Implémentation d'une architecture Microservices (Spring Boot 3)

I. Objectifs du TP :

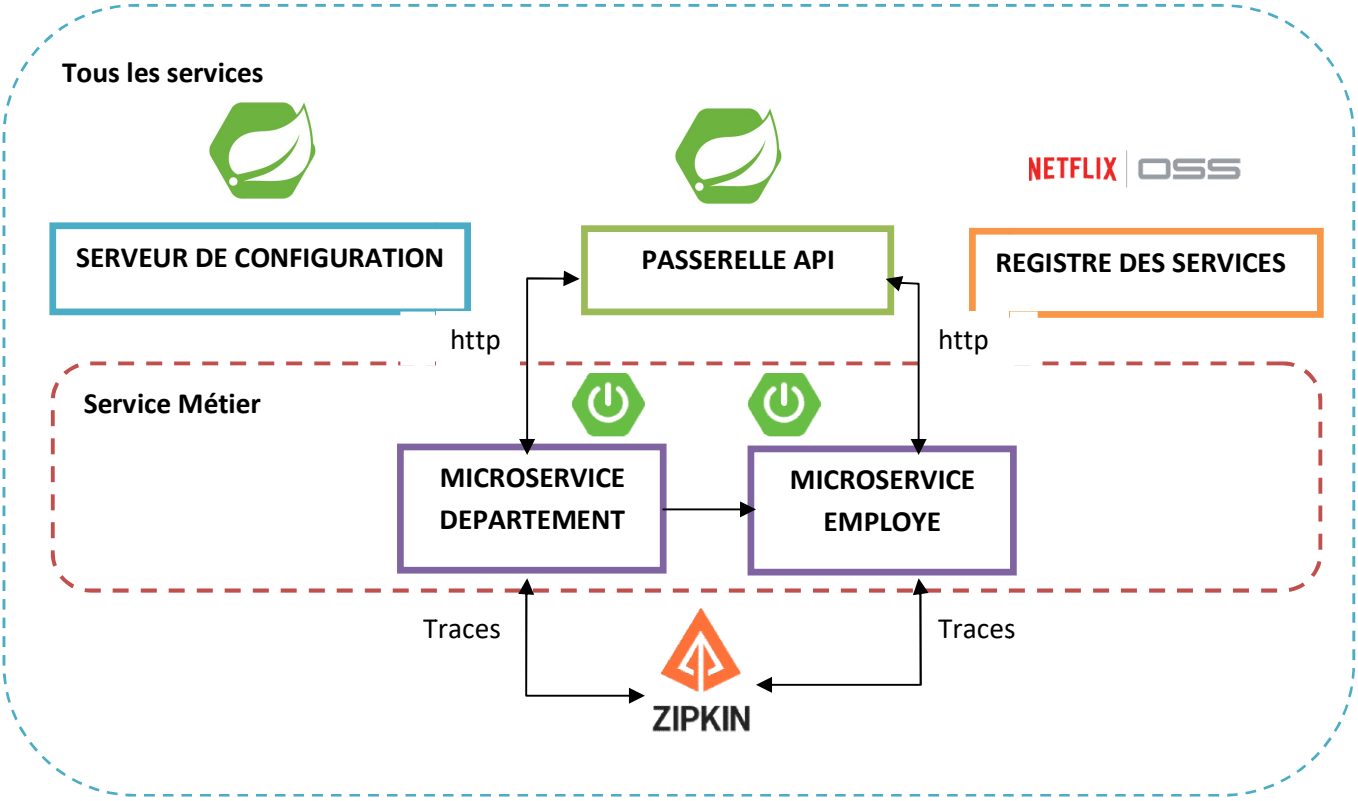
- Mettre en place une architecture microservices avec SpringBoot 3.
- Tester les microservices implémentés à l'aide d'un outil de test.

II. Pré-requis :

- Installation de Postman.

III. Description du projet :

Notre projet aura l'architecture suivante :



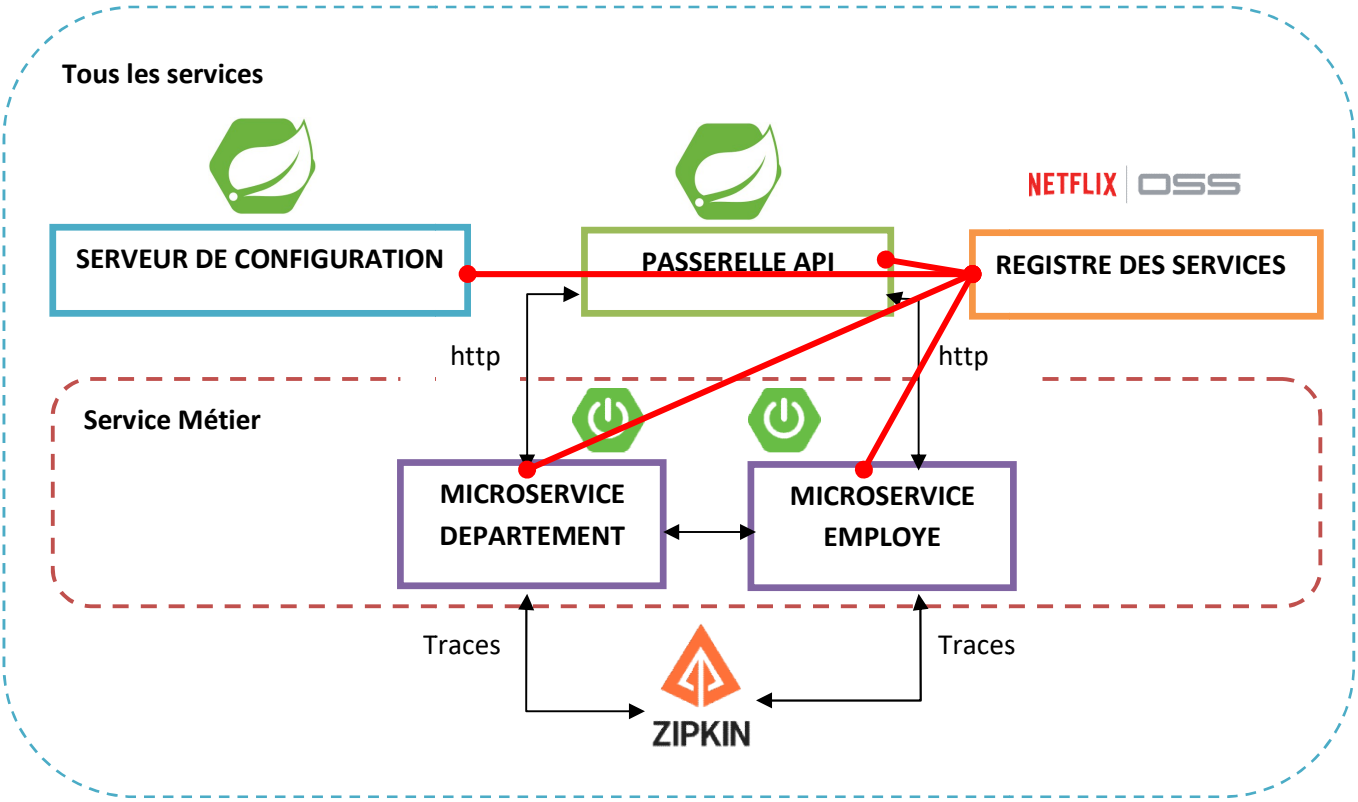
- Chaque microservice aura son propre objectif et service :
  - Le microservice département va gérer uniquement les départements :
    - Récupérer les détails d'un département
    - Ajouter un nouveau département
    - Mettre à jour les infos d'un département
    - …

- Le microservice employé va gérer uniquement les employés :
  - Récupérer les détails d'un employé
  - Ajouter un nouvel employé
  - Mettre à jour les infos d'un employé
  - …

- Les deux microservices peuvent communiquer ensemble :
  - Si je veux récupérer tous les employés d'un département particulier.
- Chaque microservice aura sa propre base de données.

a) Le registre des services :

Il contiendra tous les détails concernant les microservices connectés à lui. Ainsi que le nombre d'instances de ce service qui sont attachés à lui.

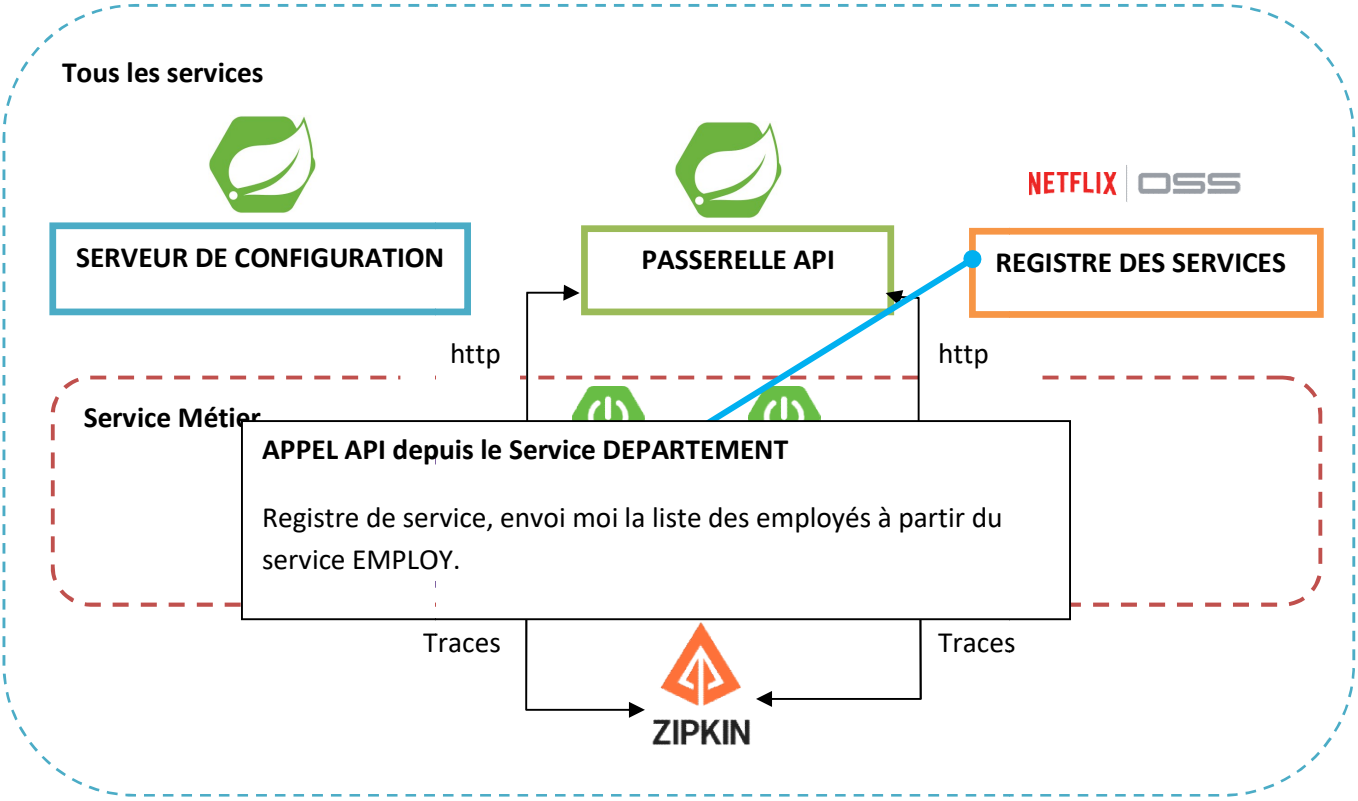


Note importante : avec les microservices on peut mettre à l'échelle (scale-up ou scale-down) un microservice bien déterminé sans affecter les autres.

Donc si j'aurai quatre instances du microservice Employe et/ou deux instances du microservice département le registre des services aura cette information.

Question : s'il existe deux instances du microservice département et quatre instances du microservice Employe, lorsque le microservice département veut se connecter au

microservice Employe, comment allait-il savoir quelle instance appeler des quatre et quelle instance va répondre ? La réponse c'est qu'il faut appeler le registre des services.



Le registre des services va gérer l'équilibrage de charge des 4 instances et va m'envoyer les données demandées.

b) PASSERELLE API

C'est la passerelle par laquelle passe toutes les requêtes publiques qui interagissent avec votre architecture microservices.

Ainsi, au lieu d'appeler le microservice DEPARTEMENT ou le microservice EMPLOYE, on va appeler la passerelle API. Et c'est cette dernière qui va accéder aux microservices.
La passerelle va imposer la sécurisation par l'envoi des requêtes à ZIPKIN.

c) SERVEUR DE CONFIGURATION

Il va enregistrer toutes les configurations par défaut des différents composants de notre architecture.

Lorsque vous créez le serveur de configuration, il nous donne différentes options comme : où nous pouvons stocker ces configurations, la valeur par défaut sera dans Git. Vous pouvez stocker toutes les configurations dans le git et à partir de là, les configurations git seront servies à tous les différents services disponibles, vous pouvez stocker toutes les

configurations dans un logiciel de *gestion de versions* et nous pouvons stocker toutes les configurations dans ce serveur natif lui-même et à partir de là, toutes les configurations seront résolues.

Ainsi, le serveur de configuration nous aidera à servir toutes les configurations communes et les configurations spécifiques pour un service particulier et tous les services peuvent se connecter à ce serveur de configuration et obtenir toutes les configurations liées à un service particulier afin que ce serveur de configuration nous aide à stocker les configurations communes et les configurations spécifiques.

d) Traçage des journaux distribués

Nous allons prendre soin du traçage des journaux distribués, c'est ainsi qu'on pourra retracer nos requêtes.

Comment allons-nous tracer nos requêtes ainsi que déboguer notre application ?

supposons qu'une requête arrive à API Gateway et que de cette requête API Gateway se déplace vers le service du département et de ce service, la demande de service est allée au service des employé et supposons que s'il y a plusieurs services, la demande y est également allée pour obtenir toutes les données collectées et les renvoyer maintenant à un moment donné, quelque chose se plante et nous en avons besoin pour identifier où cela s'est produit, où la requête a échoué et quelle est la trace complète de ce journal particulier.

Tout le traçage complet du journal peut être effectué à l'aide de Zipkin.

Zipkin est donc un outil de traçage de journaux distribué qui nous permettra de tracer l'intégralité du journal pour tous les différents services dont nous disposons. Il nous fournira aussi un tableau de bord où nous pourrons voir ce qui s'est passé pour une demande particulière.

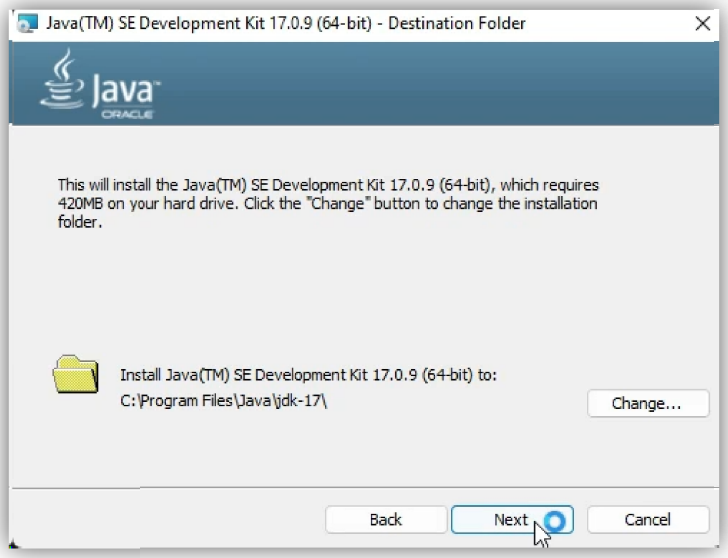
e) Étapes d'implémentation d'une architecture Microservice :

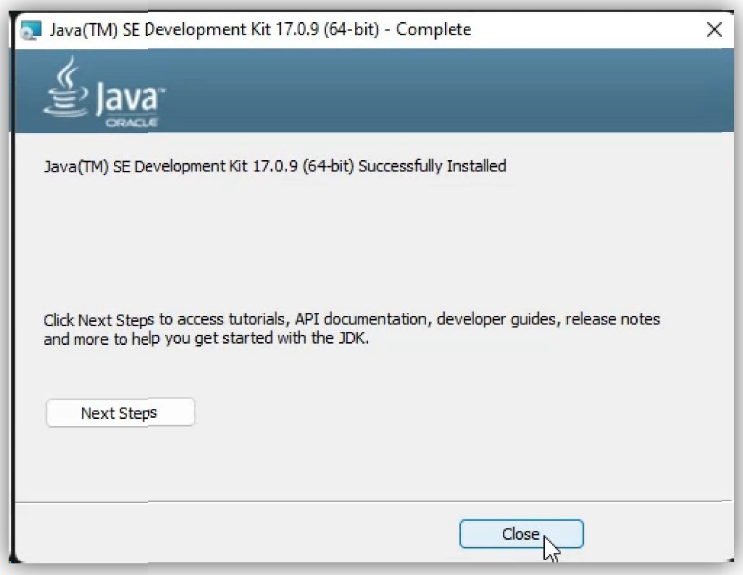
1. Création du Serveur Eureka (registre des services) ;
2. Création du microservice DEPARTEMENT ;
3. Création du serveur de configuration ;
4. Création du serveur ZIPKIN qui sauvegarder tous les logs ;
5. Implémentation du microservice DEPARTEMENT;
6. Test du microservice DEPARTEMENT en utilisant Postman ;
7. Création du microservice EMPLOYE ;
8. Implémentation du microservice EMPLOYE ;
9. Test du microservice EMPLOYE en utilisant Postman ;

10. Connecter les deux microservices au registre des services ;
11. Test de la communication entre le microservice DEPARTEMENT et le microservice EMPLOYE en utilisant Postman ;
12. Création de la passerelle API ;
13. Test du passerelle API en utilisant Postman.

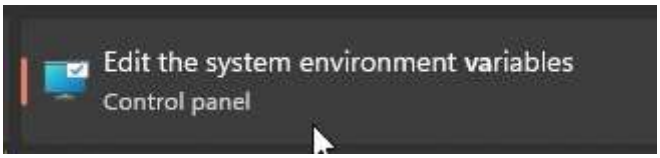
IV. Création du projet :

1. Installez jdk-17 :

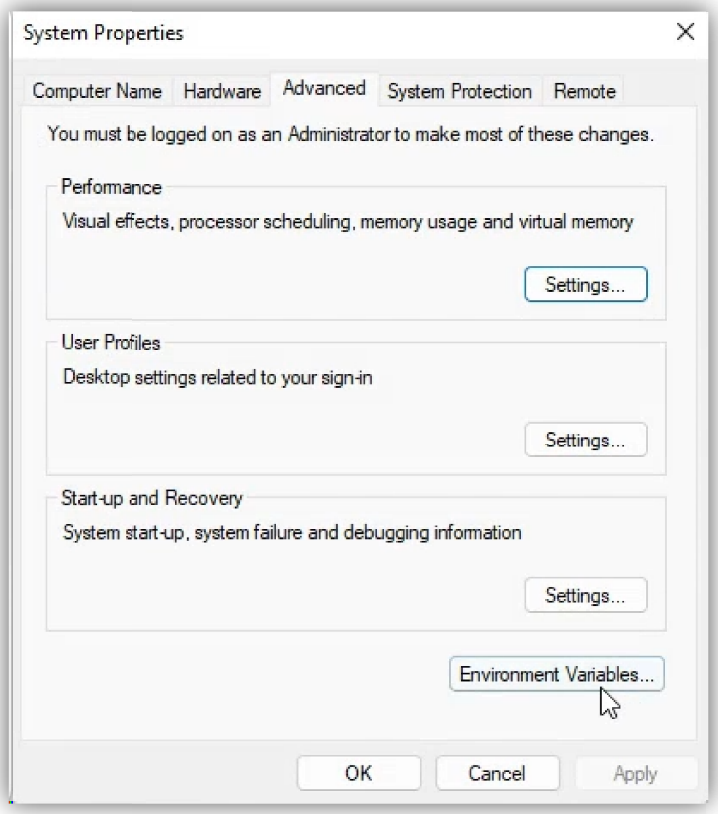




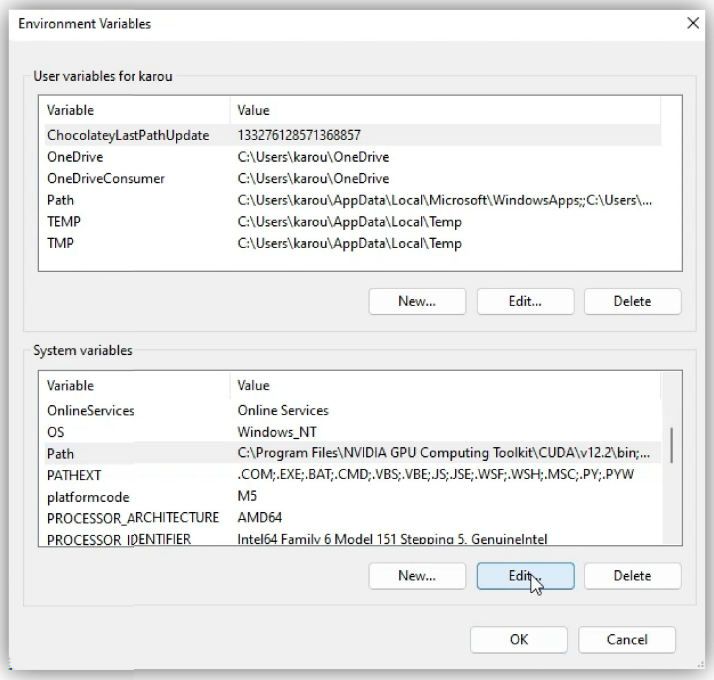
2. Ouvrir la fenêtre dédiée à l'édition des variables d'environnement :



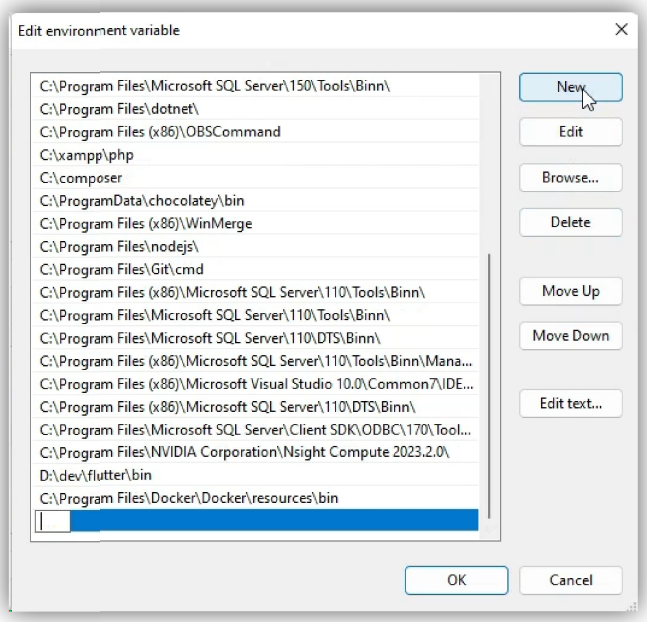
3. Cliquez sur (**Environment Variables…**) :



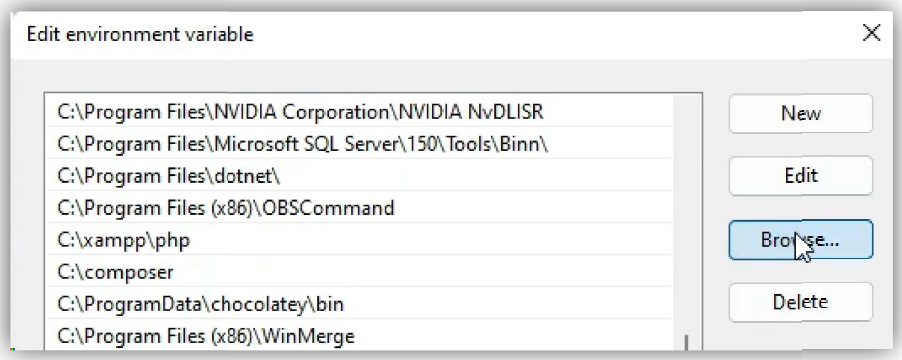
4. Cherchez la variable système (**Path**)



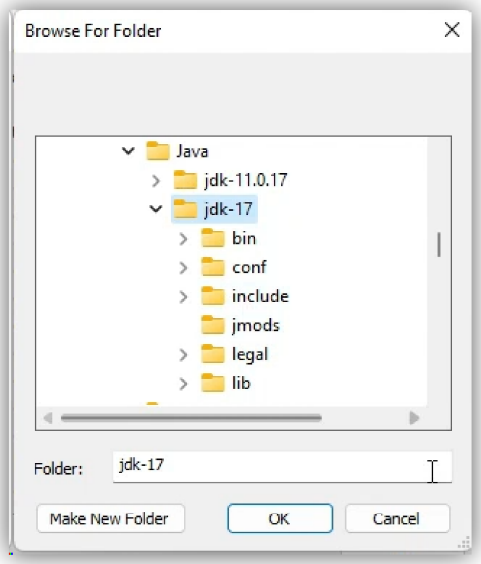
5. Créer une nouvelle variable :



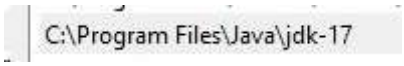
6. Cliquez sur le bouton (Browse)



7. Sélectionner le dossier (**jdk-17**) :

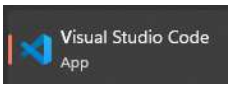


Et voici le résultat :



8. Cliquez sur le bouton (OK) pour valider l'action.

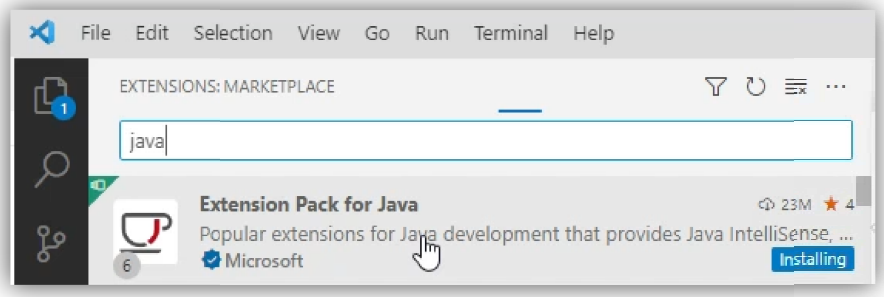
9. Lancez Microsoft Visual Studio Code :



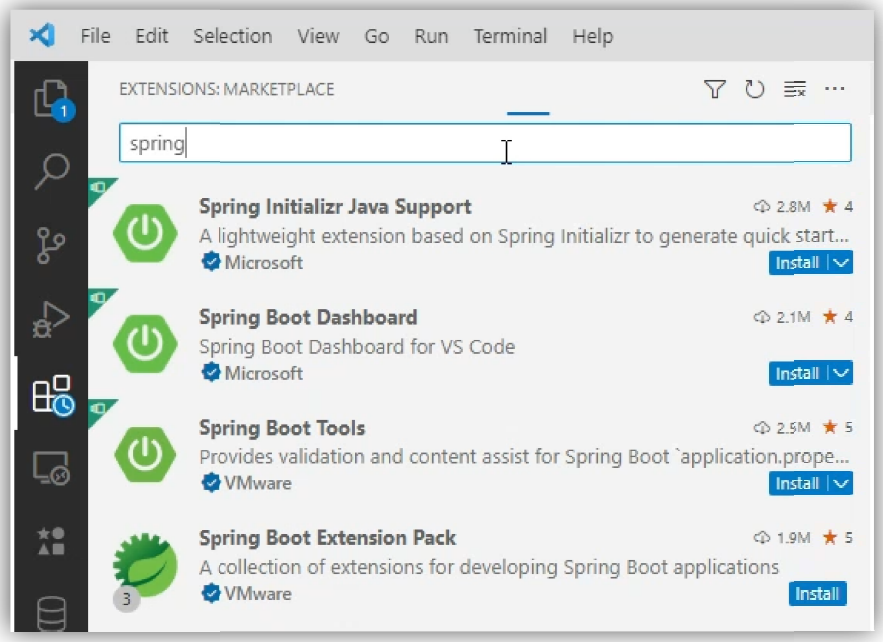
10. Ouvrez l'onglet d'extensions :

11. Installez les extensions suivantes :

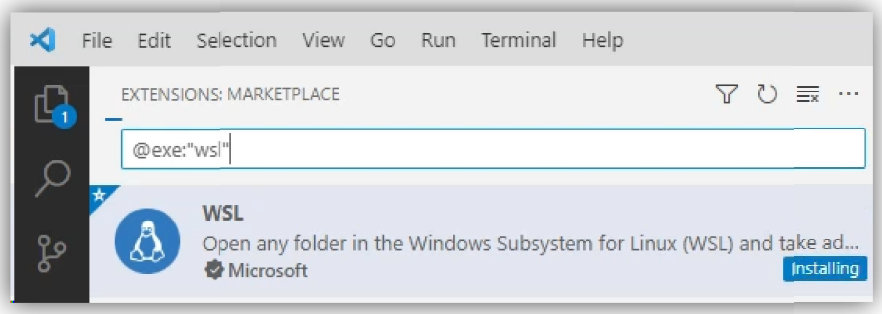
Extension Pack for Java



Extensions Spring Boot suivantes : (quatre extensions)



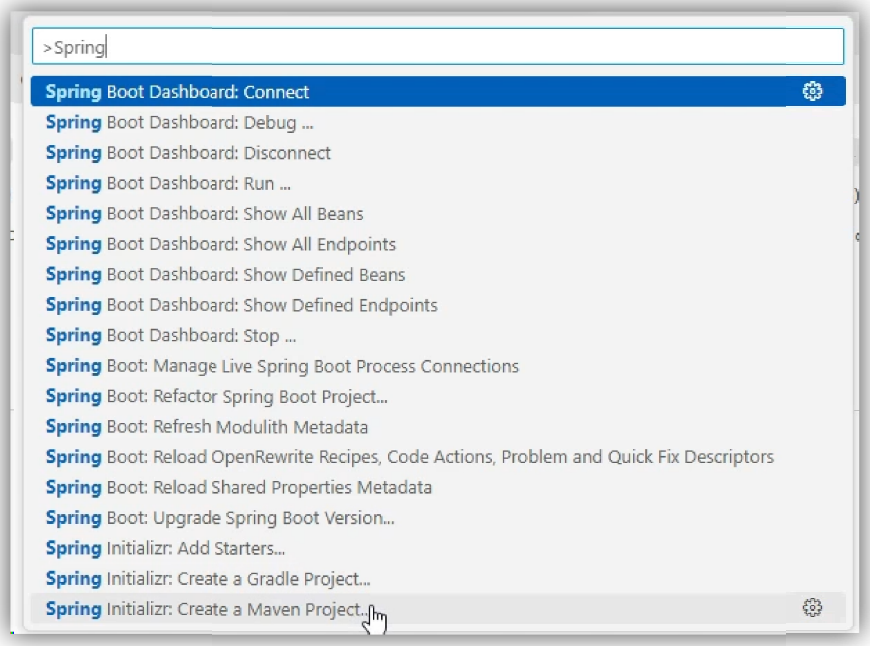
Extension wsl:



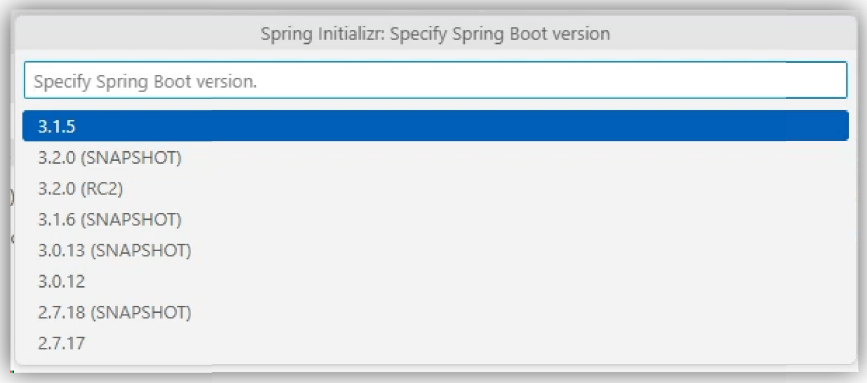
12. Redémarrez Microsoft Visual Studio Code

V. Création du Serveur Eureka

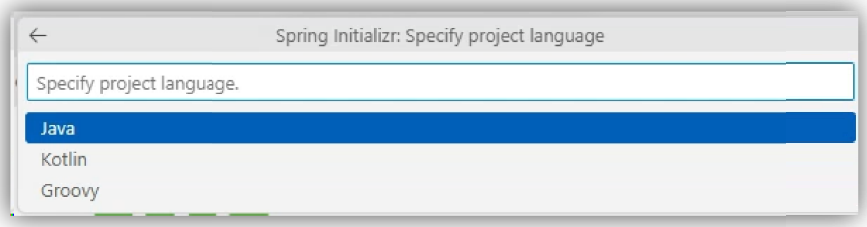
13. Appuyer sur Ctrl + Shift + P afin de lancer la palette des commandes.
14. Cliquez sur (**Spring Initializr : Create a Maven Project …**):



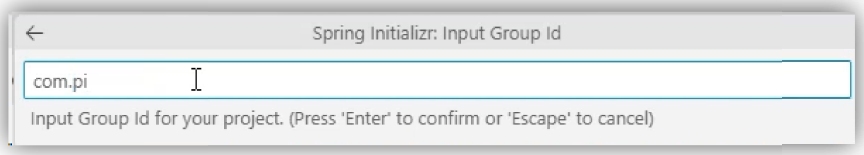
15. Sélectionnez la version (**3.1.5**)



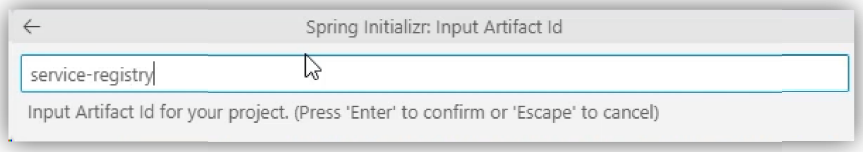
16. Sélectionnez le langage de programmation (**Java**)



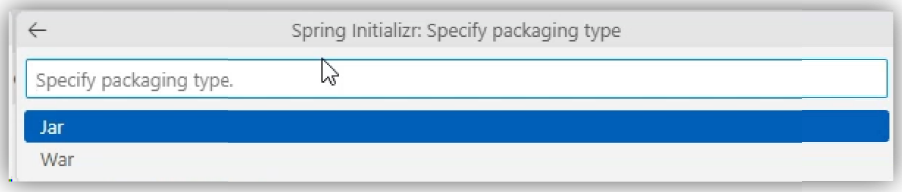
17. Choisir comme nom de package (**com.pi**)



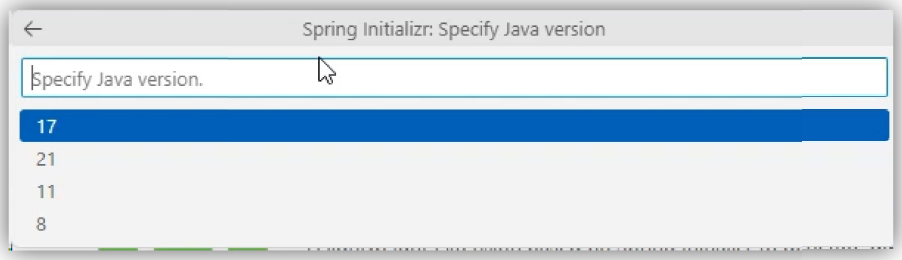
18. Indiquez le nom de projet (**service-registry**):



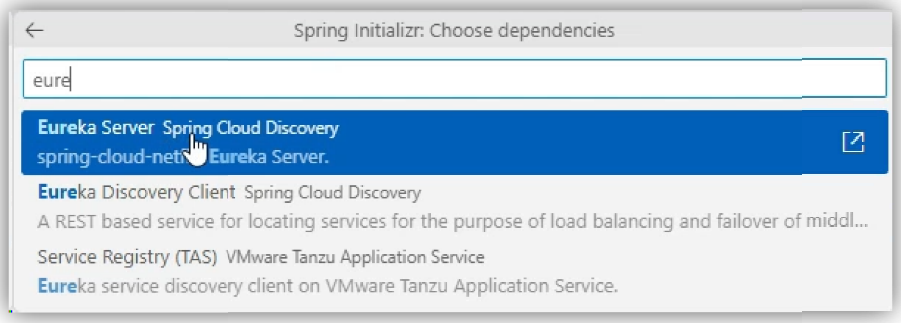
19. Choisir comme type de package (**Jar**)



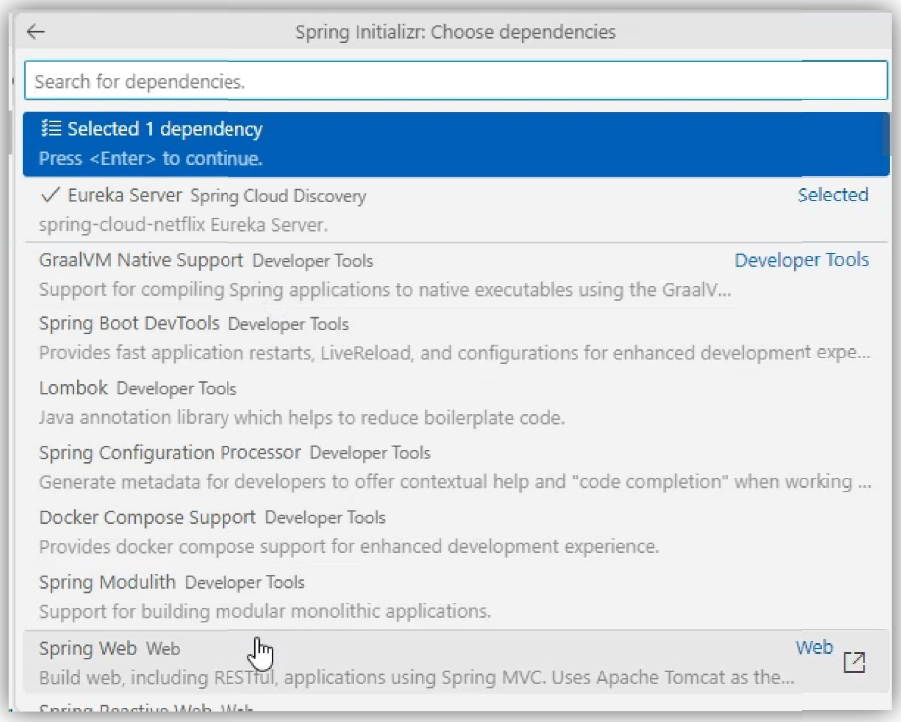
20. Choisir la version de Java Development Kit (**17**)



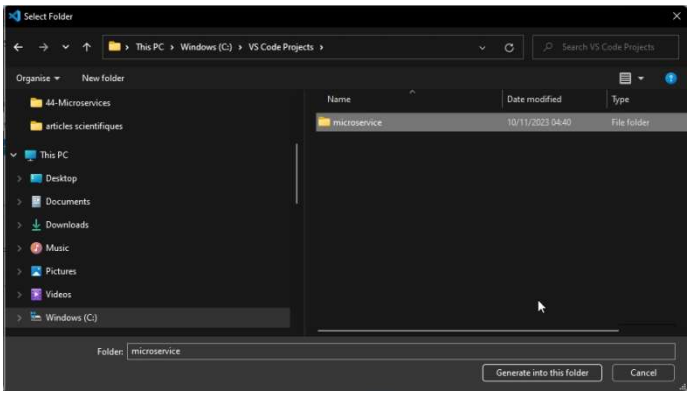
21. Ajoutez la dépendance (**Eureka Server – Spring Cloud Discovery**)



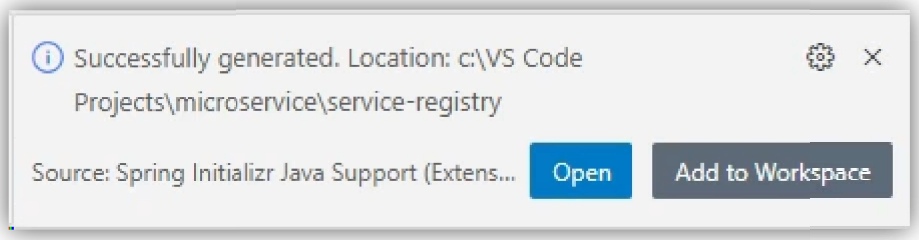
22. Ajoutez la dépendance (**Spring Web – Web**)



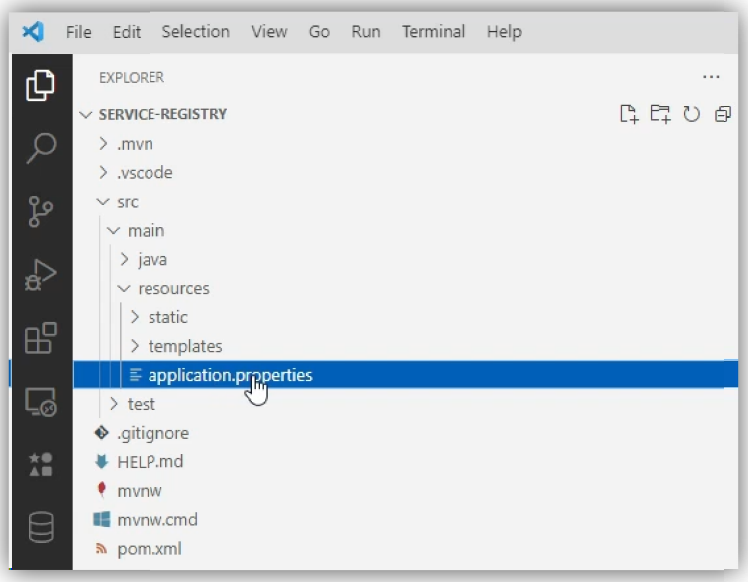
23. Sous le lecteur (**C:**), créer le chemin suivant (**VS Code Projects\microservice**)
24. Sélectionnez le dossier (**microservice**) puis cliquez sur (**Genrate into this folder**) :

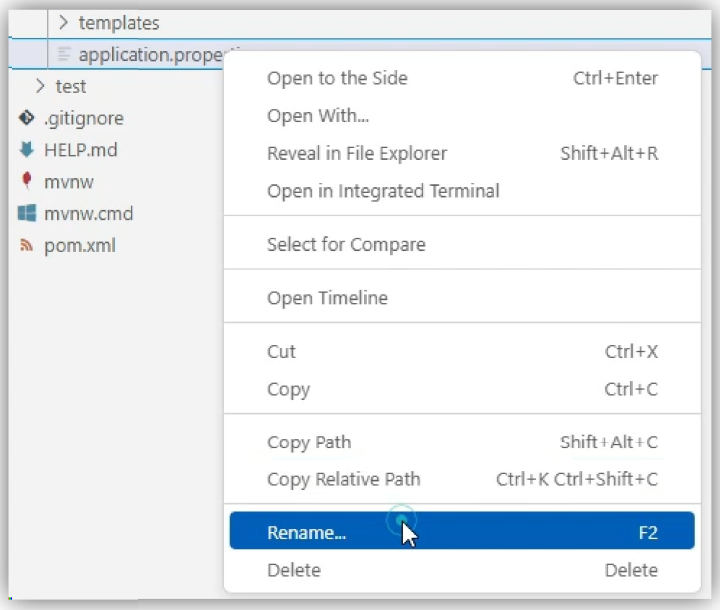


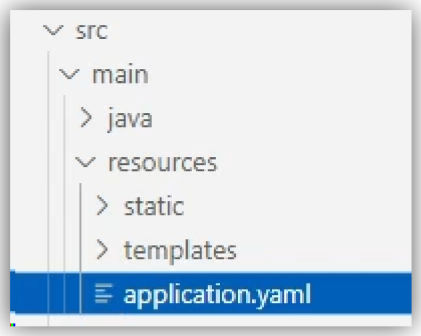
25. Cliquez sur (**Open**):



26. Sélectionnez le fichier (SERVICE-REGISTRY\src\main\resources\application.properties), puis modifiez son extension pour qu'elle devienne (application.yaml)







27. Ouvrir le fichier (**ServiceRegistryApplication.java**), ajouter l'annotation (@EnableEurekaServer) (Ligne7):

ServiceRegistryApplication.java × pom.xml application.yaml

src > main > java > com > pi > serviceregistry > ServiceRegistryApplication.java > ServiceRegistryApplication

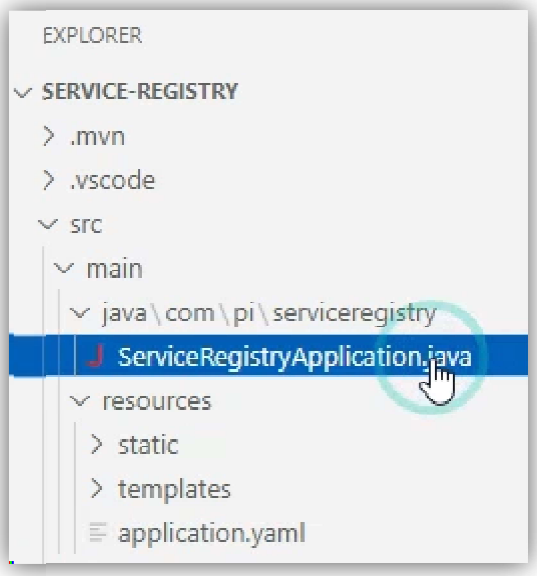
```
1  package com.pi.serviceregistry;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7  @SpringBootApplication
8  @EnableEurekaServer
9  public class ServiceRegistryApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(ServiceRegistryApplication.class, args);
13     }
14
15 }
16
```

28. Se déplacer au fichier (**application.yaml**), et écrire le code suivant :

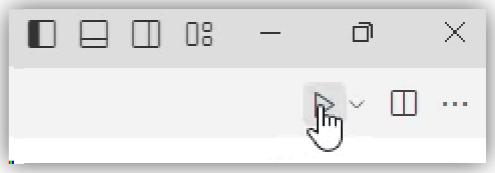
src > main > resources > application.yaml

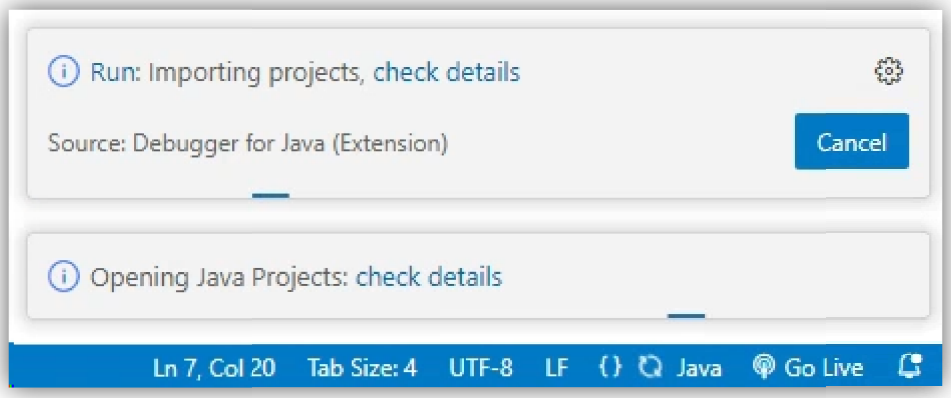
```yaml
server:
    port: 8761
spring:
    application:
        name: service-registry
eureka:
    instance:
        hostname: localhost
    client:
        register-with-eureka: false
        fetch-registry: false
        serviceUrl:
            defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

```

29. Sélectionner le fichier (**ServiceRegistryApplication.java**):



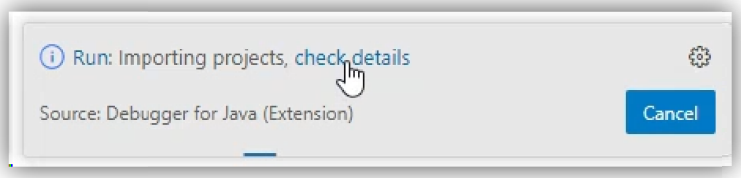
30. Puis, Exécuter la fonction main :





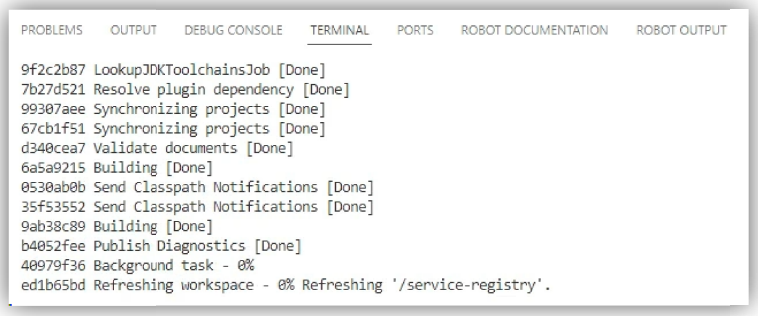
31. Affichez le terminal :

32. Cliquez sur (check details) :



33. Et voici l'avancement de l'importation du projet Maven :

...

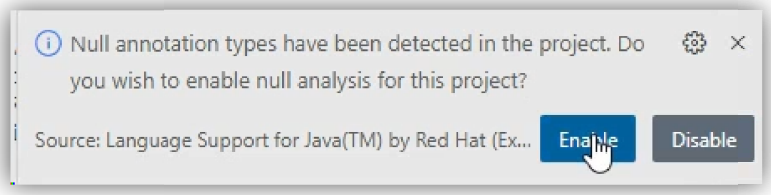


34. Autorisez l'accès à Internet pour (Open JDK Platform binary) et (java™ Plateform SE binary) :

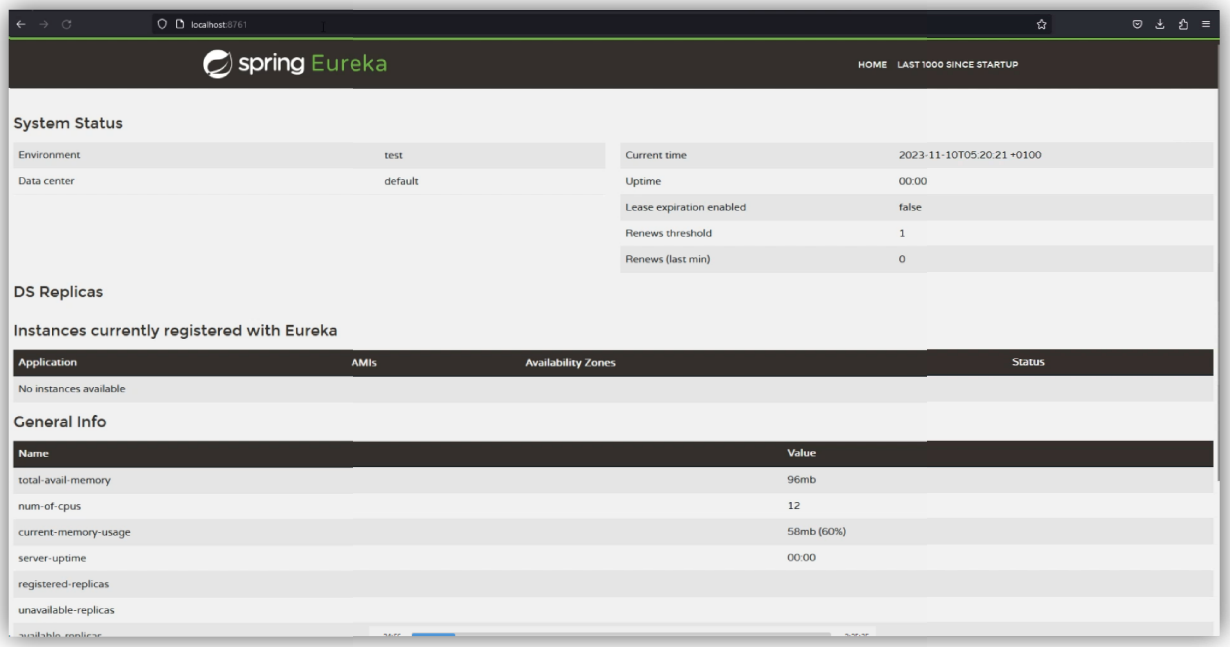

Windows Security Alert
Windows Defender Firewall has blocked some features of this app
Windows Defender Firewall has blocked some features of OpenJDK Platform binary on all public and private networks.
Name: OpenJDK Platform binary
Publisher: Eclipse Adoptium
Path: C:\users\karou\.vscode\extensions\redhat.java-1.24.0-win32-x64\jre\17.0.8.1-win32-x86_64\bin\java.exe
Allow OpenJDK Platform binary to communicate on these networks:
Private networks, such as my home or work network
Public networks, such as those in airports and cafés (not recommended because these networks often have little or no security)
What are the risks of allowing an app through a firewall?
Allow access
Cancel


PS C:\VS Code Projects\microservice\service-registry> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '@C:\Users\karou\AppData\Local\Temp\cp_w4p8co1czxcG4xieoeb7b2sy.argfile' 'com.pi.serviceregistry.ServiceRegistryApplication'
:: Spring Boot :: (v3.1.5)
2023-11-10T05:20:07.964+01:00 INFO 23916 --- [main] c.p.s.ServiceRegistryApplication : Starting ServiceReg
Code Projects\microservice\service-registry\target\classes started by karou in C:\VS Code Projects\microservice\service-regis
2023-11-10T05:20:07.968+01:00 INFO 23916 --- [main] c.p.s.ServiceRegistryApplication : No active profile s
Null annotation types have been detected in the project. Do you wish to enable null analysis for this project?
Source: Language Support for Java(TM) by Red Hat (Ex...
Enable
Disable

35. Activer l'analyse de types d'annotation null :

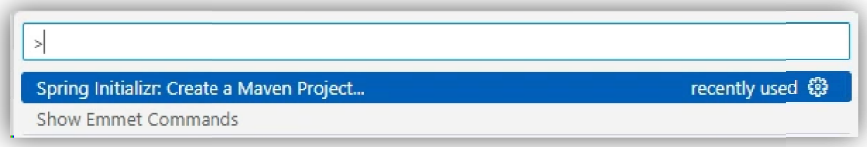


36. Lancez votre navigateur, puis écrire l'adresse suivante : **localhost:8761**

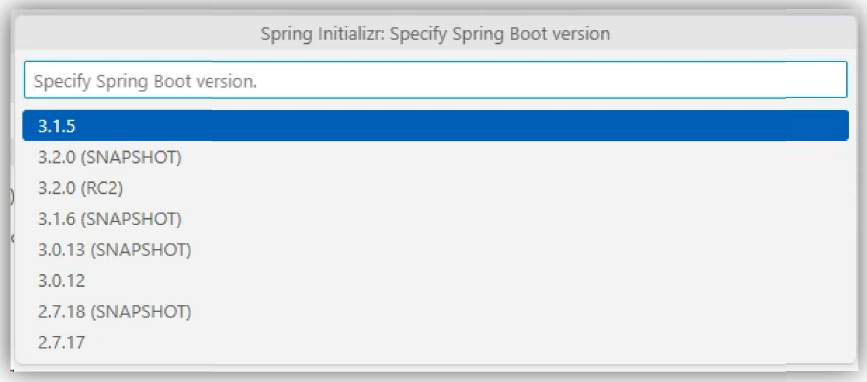


VI. Création du Microservice « Department »

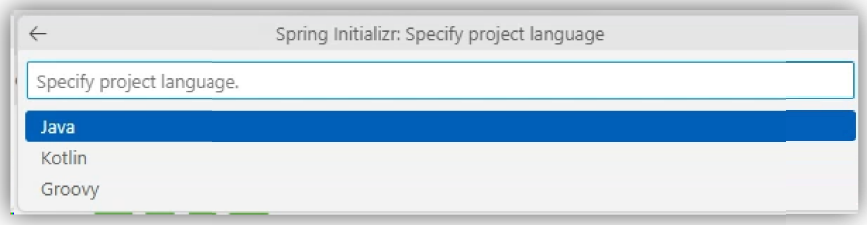
37. Appuyer sur Ctrl + Shift + P afin de lancer la palette des commandes.
38. Cliquez sur (Spring Initializr : Create a Maven Project …):



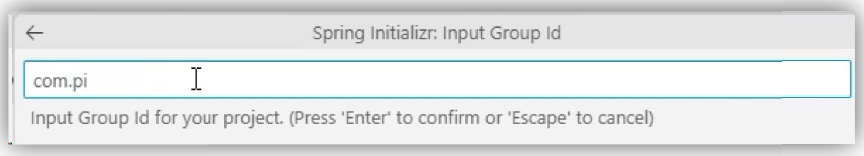
39. Sélectionnez la version (3.1.5)



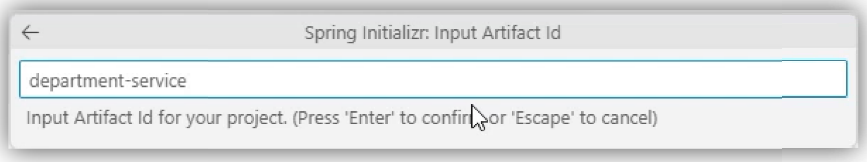
40. Sélectionnez le langage de programmation (Java)



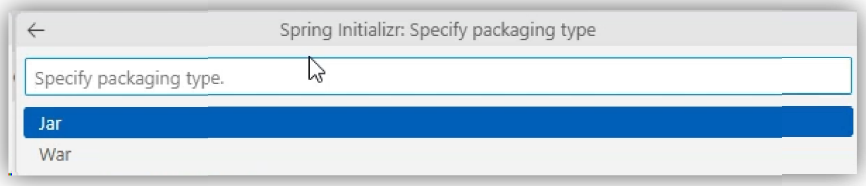
41. Choisir comme nom de package (com.pi)



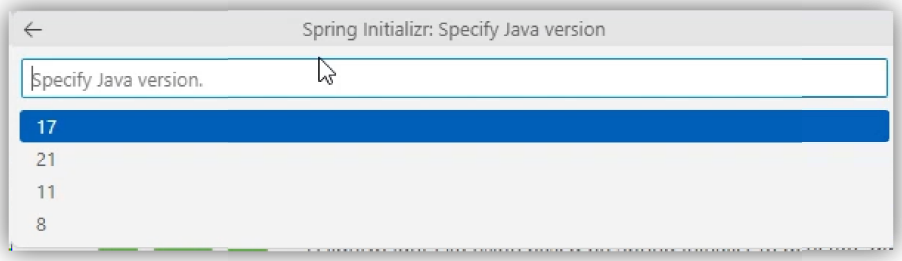
42. Indiquez le nom de projet (department-service):



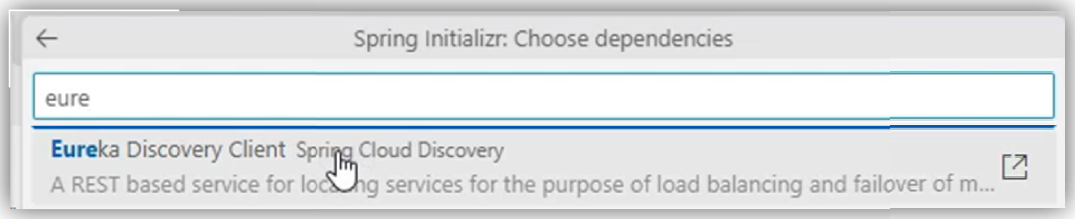
43. Choisir comme type de package (Jar)



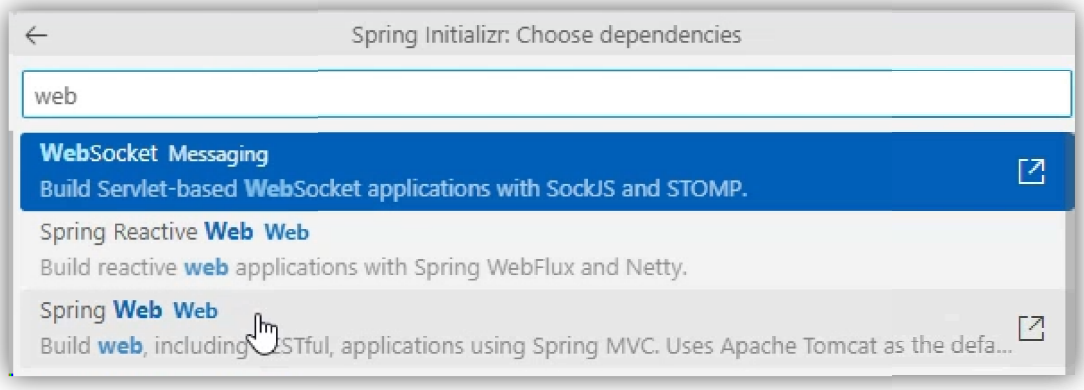
44. Choisir la version de Java Development Kit (**17**)



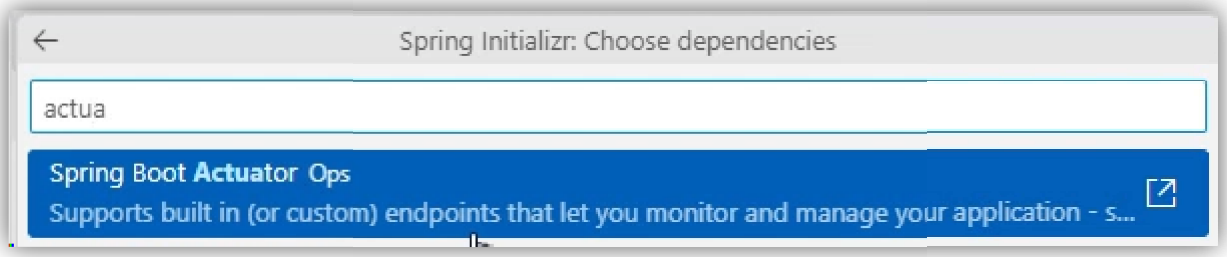
45. Ajoutez la dépendance (**Eureka Discovery Client – Spring Cloud Discovery**)



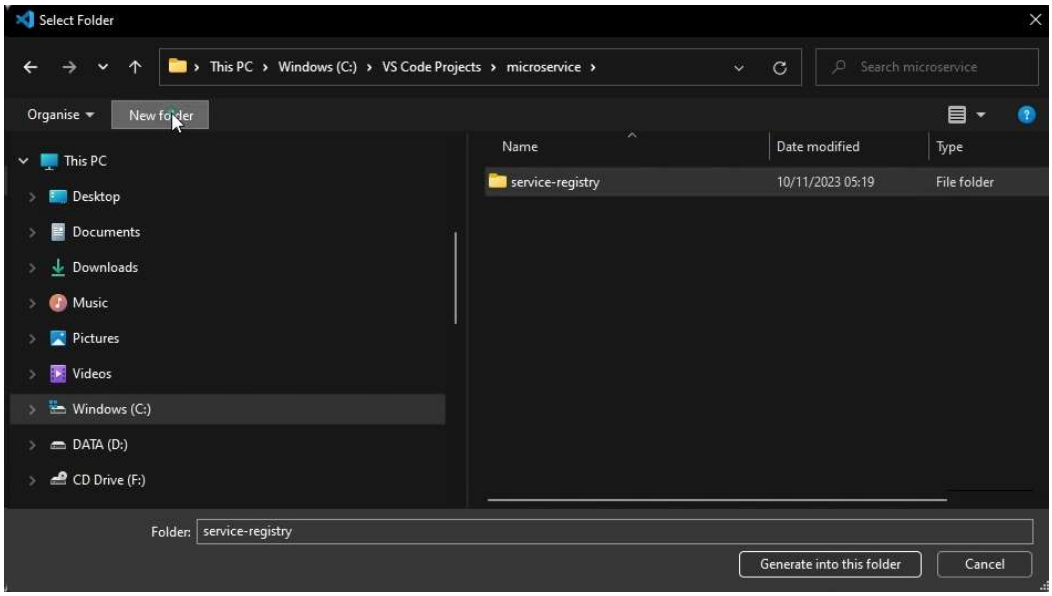
46. Ajoutez la dépendance (**Spring Web – Web**)



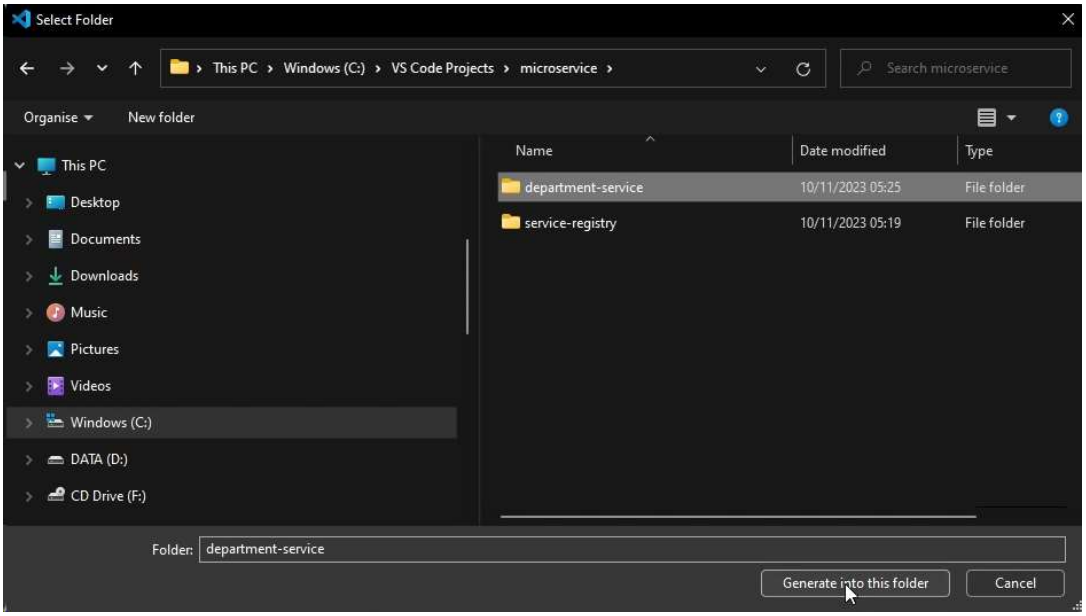
47. Ajoutez la dépendance (**Spring Boot Actuator – Ops**)



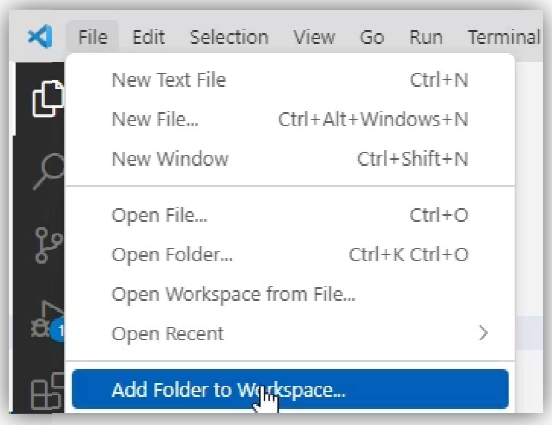
48. Appuyer sur (Entrée).
49. Créez un nouveau répertoire sous (C:\VS Code Projects\microservice) et appelez le (department-service)

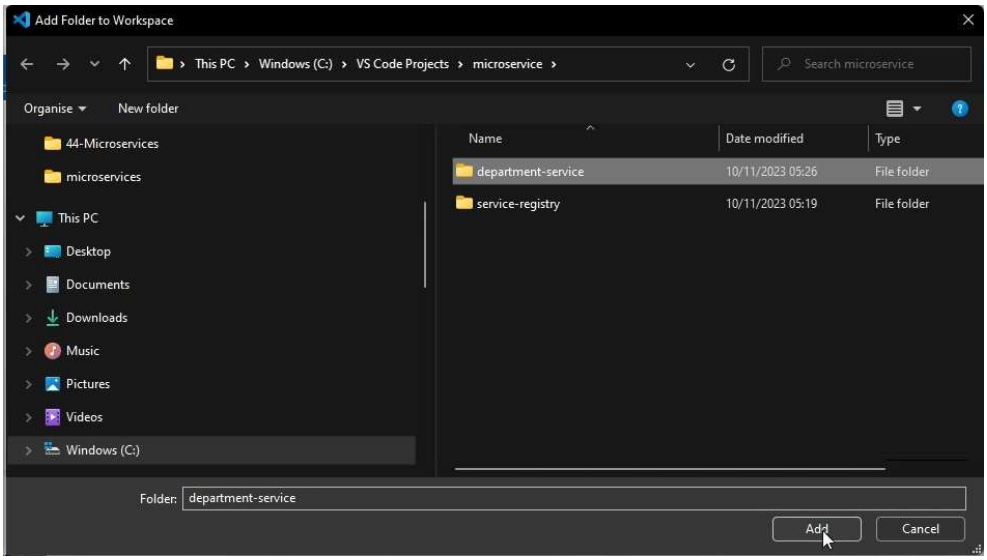


50. Sélectionnez le répertoire (department-service) puis cliquez sur (Generate into this folder)

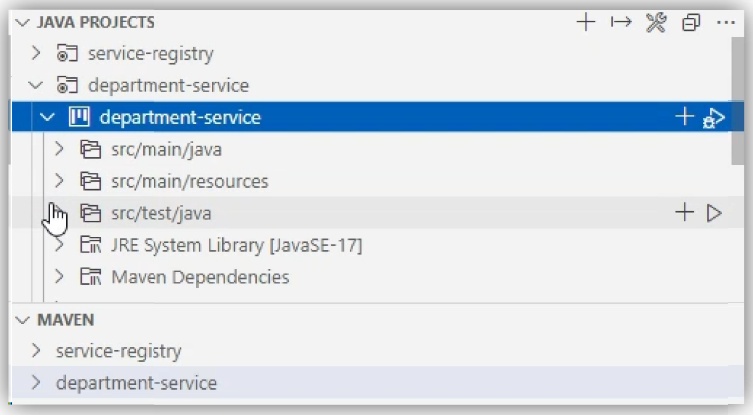


51. Depuis le projet (service-registry), ajoutez projet (department-service) à l'espace de travail:

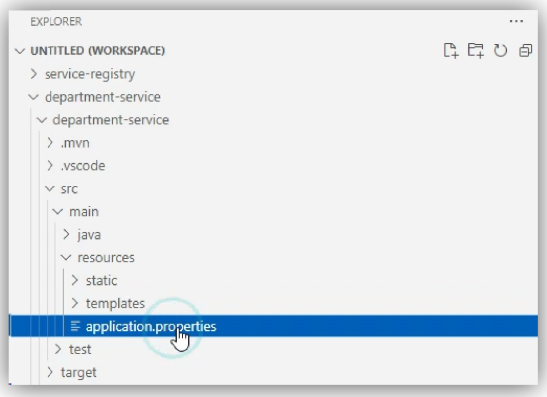




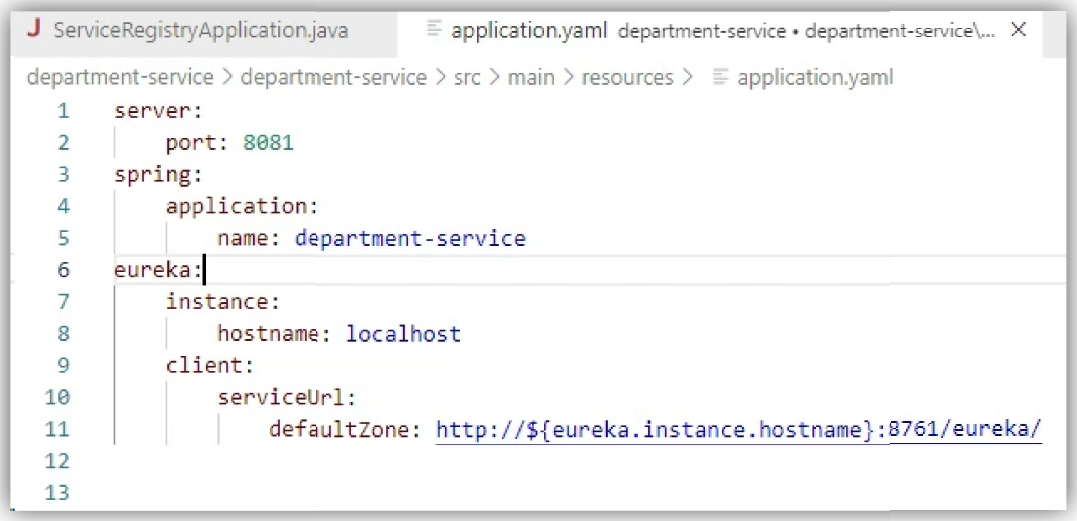
52. Et voici le résultat après l'appui sur le bouton (add) :



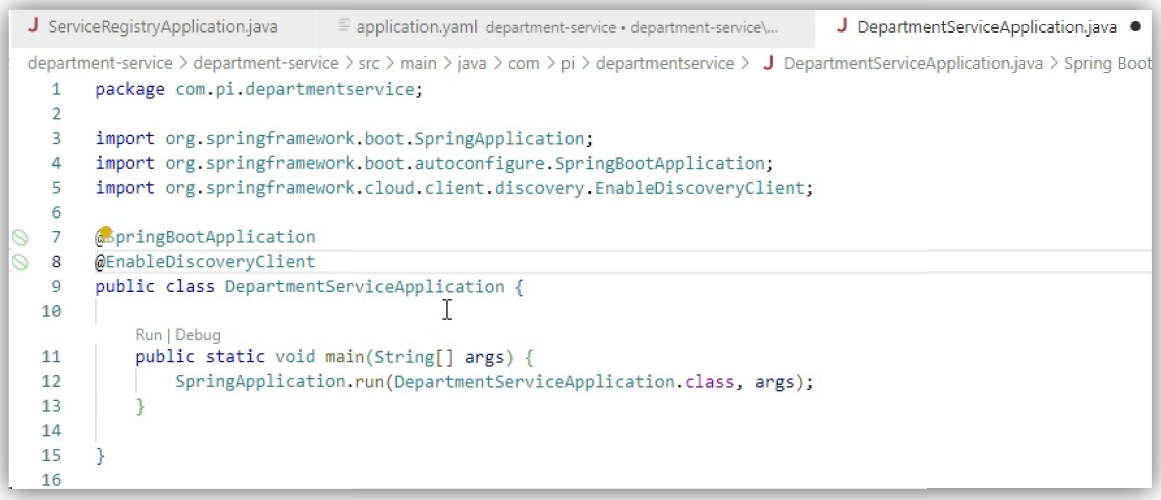
53. Changez l'extension du fichier (application.properties) pour qu'il devienne (application.yaml)



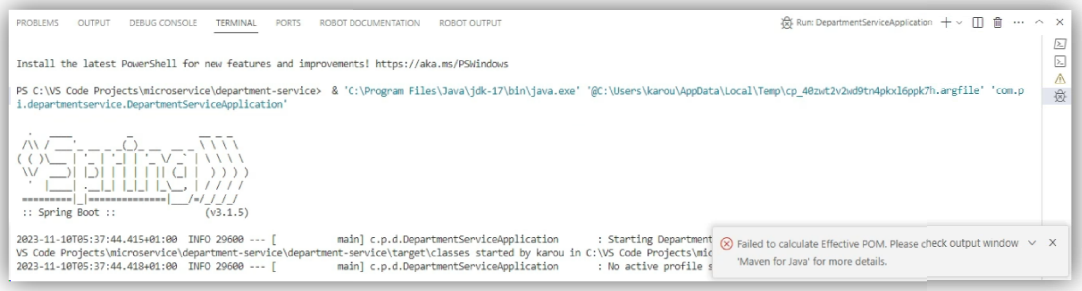
54. Saisir le code suivant dans le fichier (application.yaml) du microservice (department-service)



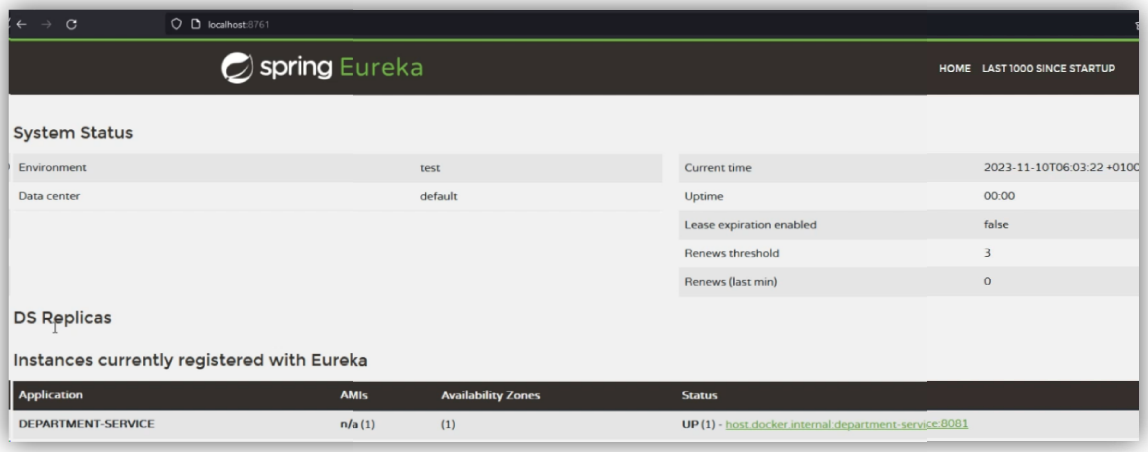
55. Ajoutez l'annotation (@EnableDiscoveryClient) au niveau du fichier (DepartmentServiceApplication.java) :



56. Exécuter le microservice (**department-service**):

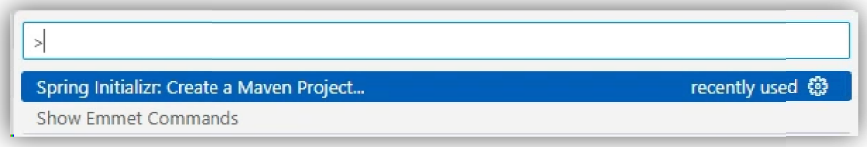


57. Rafraîchir la page **localhost:8761** et notez de la présence de (DEPARTMENT-SERVICE) dans la liste des applications reconnues :

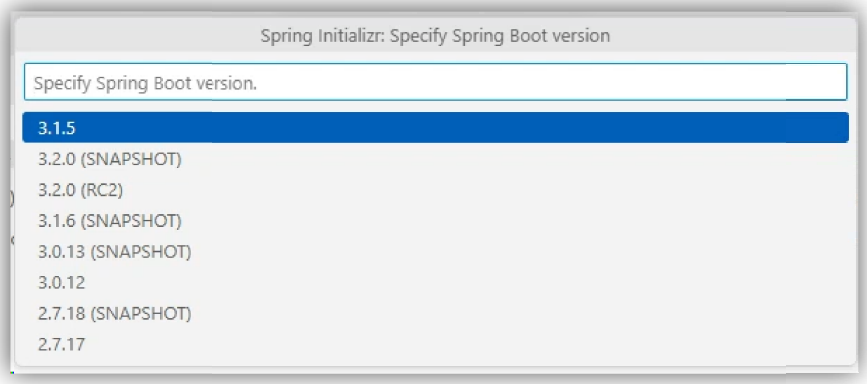


VII. Création du Serveur de Configuration

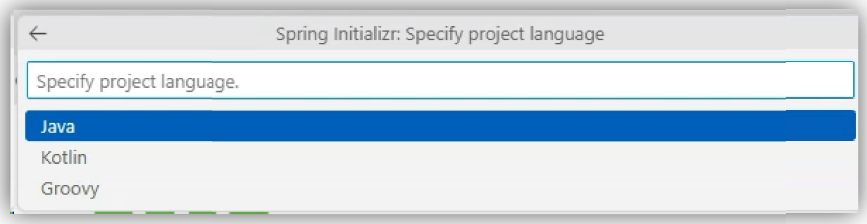
58. Lancez une nouvelle instance de Microsoft Visual Studio Code.
59. Appuyer sur **Ctrl + Shift + P** afin de lancer la palette des commandes.
60. Cliquez sur (**Spring Initializr : Create a Maven Project …**):



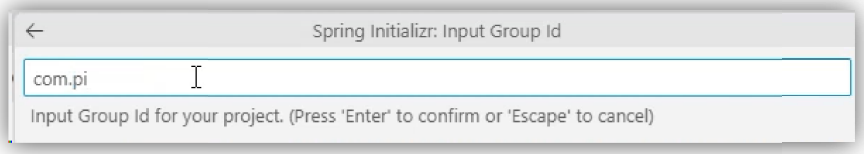
61. Sélectionnez la version (**3.1.5**)



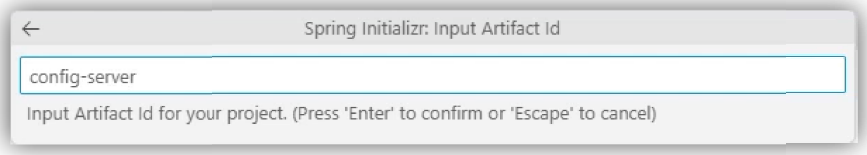
62. Sélectionnez le langage de programmation (**Java**)



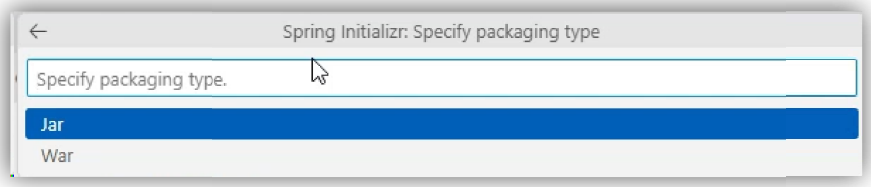
63. Choisir comme nom de package (**com.pi**)



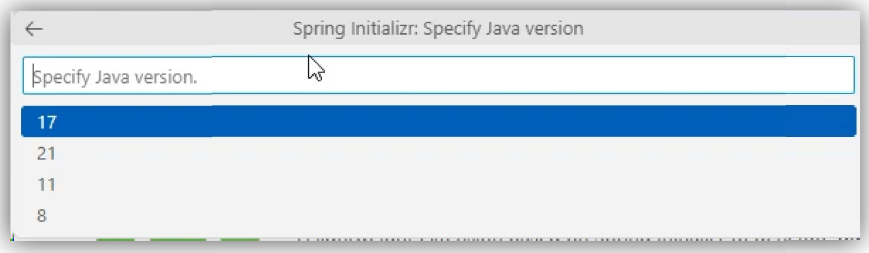
64. Indiquez le nom de projet (**config-server**):



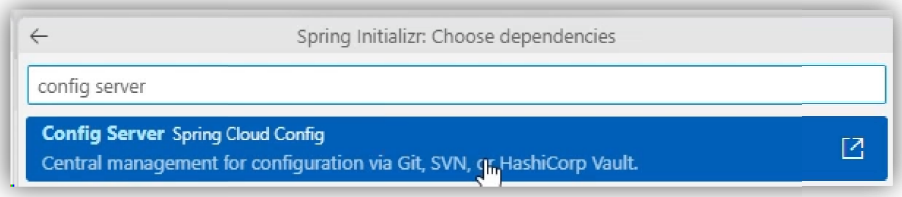
65. Choisir comme type de package (**Jar**)



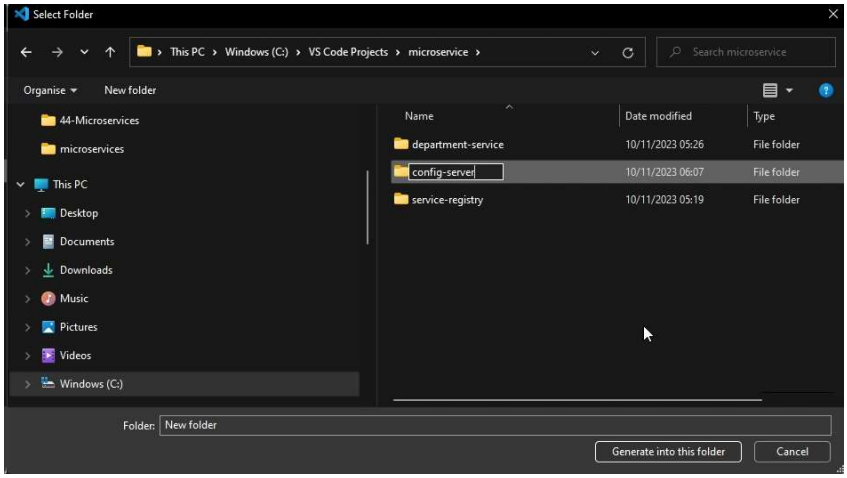
66. Choisir la version de Java Development Kit (**17**)



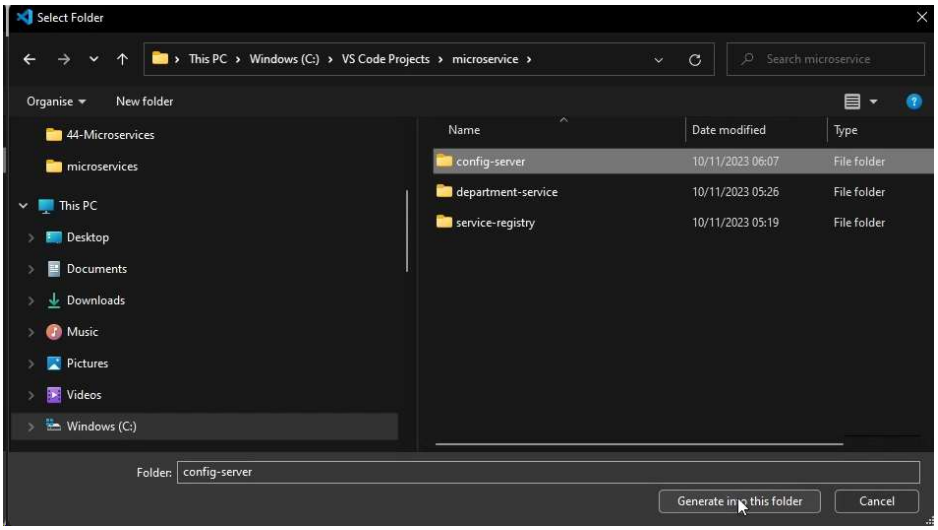
67. Ajouter la dépendance (**Config Server – Spring Cloud Config**)



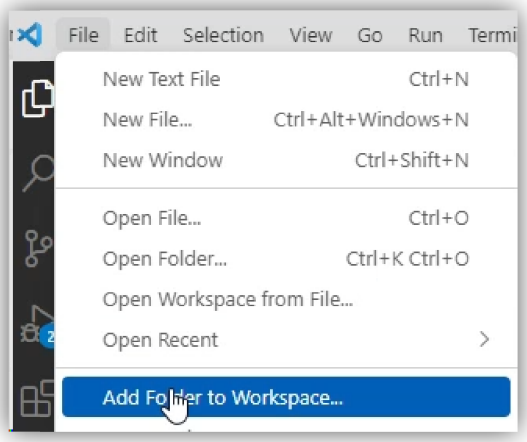
68. Créer un nouveau répertoire appelé (config-server) sous (C:\VS Code Projects\microservice), puis le sélectionner.



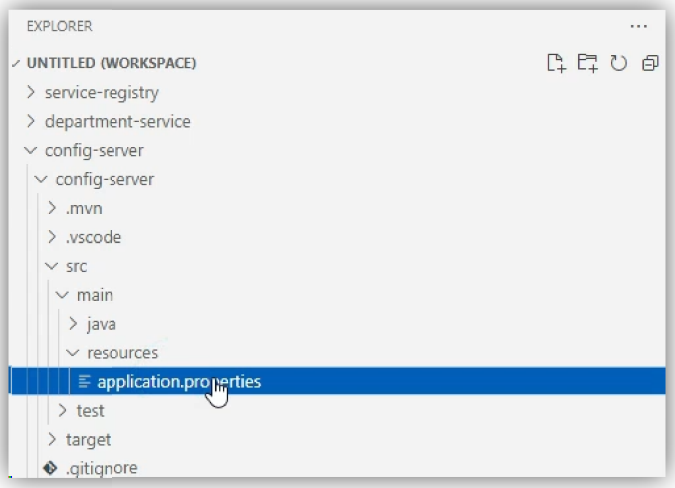
69. Cliquez sur le bouton (**Generate into this folder**) :



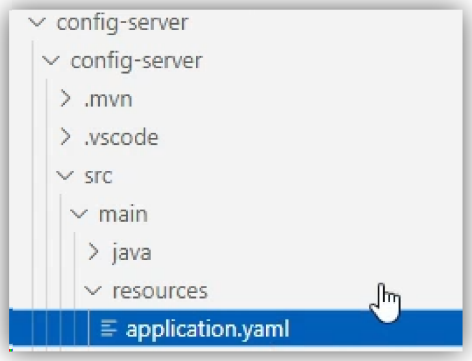
70. Allez au projet principal, puis ajouter le projet ajouté récemment au principal :



71. Sélectionnez le fichier (config-server\src\main\resources\application.properties), puis modifiez son extension pour qu'elle devienne (application.yaml)

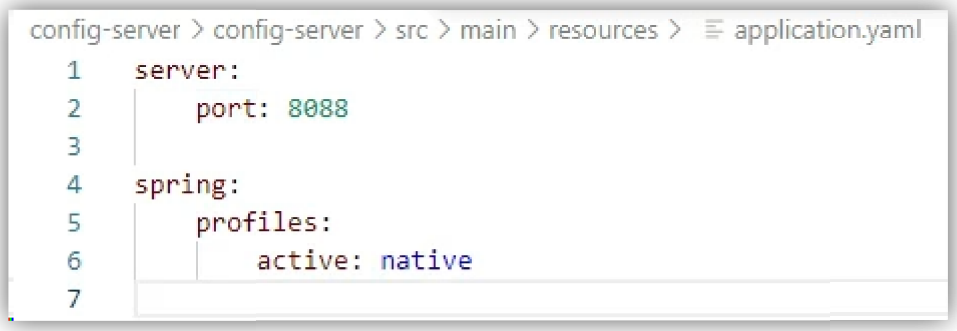


72. Et voici le résultat :

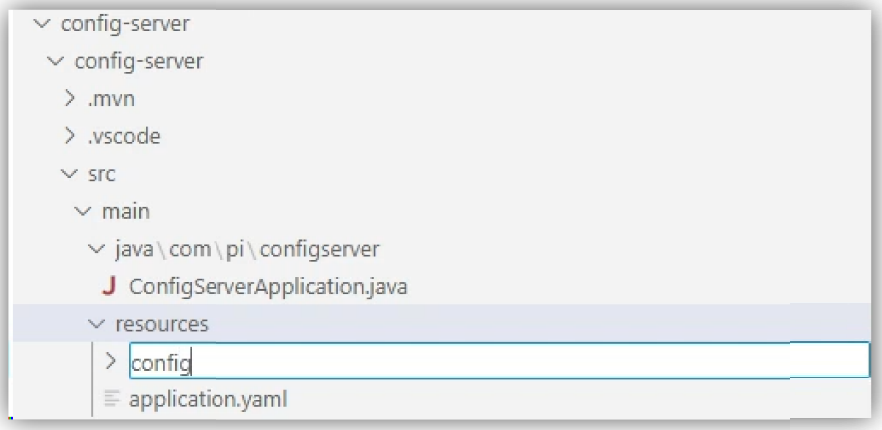


73. Ajoutez l'annotation (@EnableConfigServer) au niveau du fichier (ConfigServerApplication.java) :

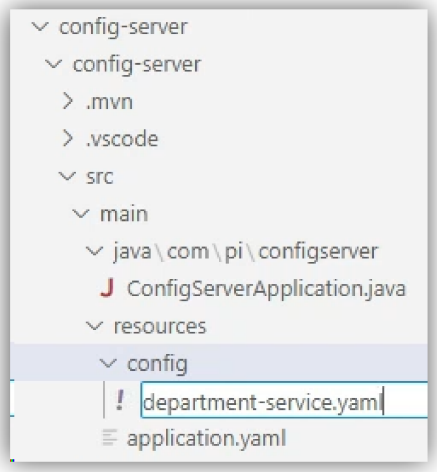
74. Se déplacer au fichier (**application.yaml**), et écrire le code suivant :



75. Créez un nouveau répertoire sous les dossier (**resources**) appelé (**config**) :



76. Créez un fichier appelé (department-service.yaml) sous (resources\config\)

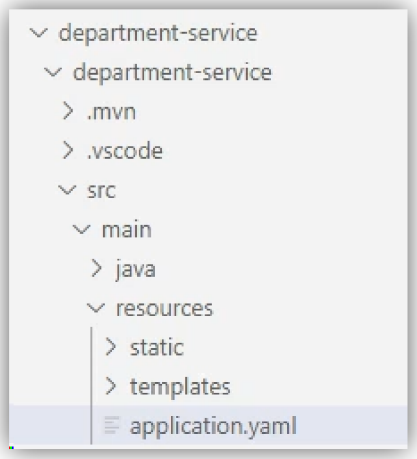


77. Ecrivez le code suivant :

config-server > config-server > src > main > resources > config > department-service.yaml

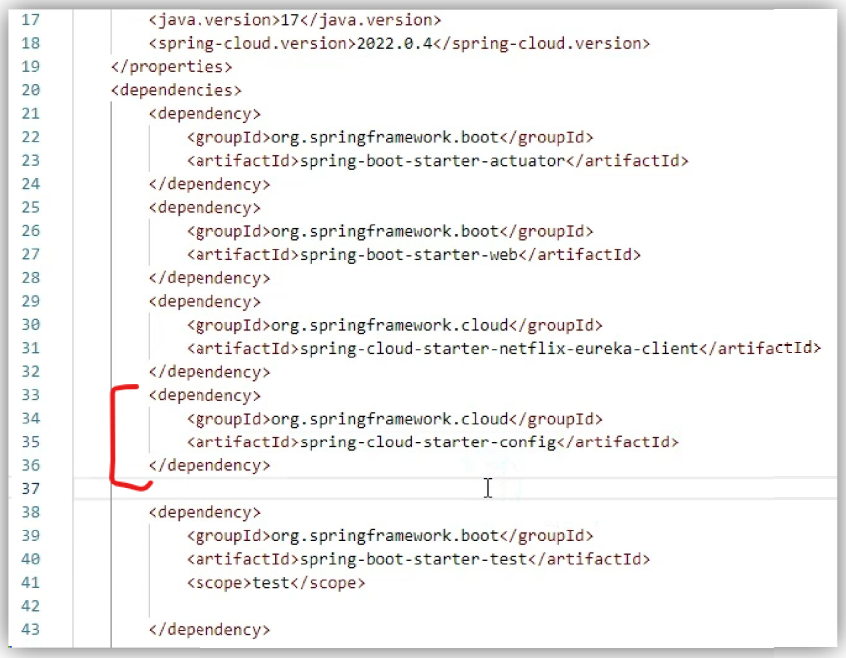
```yaml
server:
    port: 8081

eureka:
    instance:
        hostname: localhost
    client:
        serviceUrl:
            defaultZone: http://${eureka.instance.hostname}:8761/eureka/

```

78. Allez vers le fichier (**application.yaml**) du microservice (**department-service**) :



79. Modifiez son contenu pour qu'il soit ainsi:

80. Ajoutez la dépendance suivante au fichier pom.xml du microservice (**department-service**) pour qu'il puisse accéder à la configuration:



81. Arrêter toutes les applications:

82. puis les relancer de nouveau dans l'ordre suivant :

(1) service-Regitry : depuis le fichier :

(SERVICE-REGISTRY\src\main\java\ServiceRegistryApplication.java)

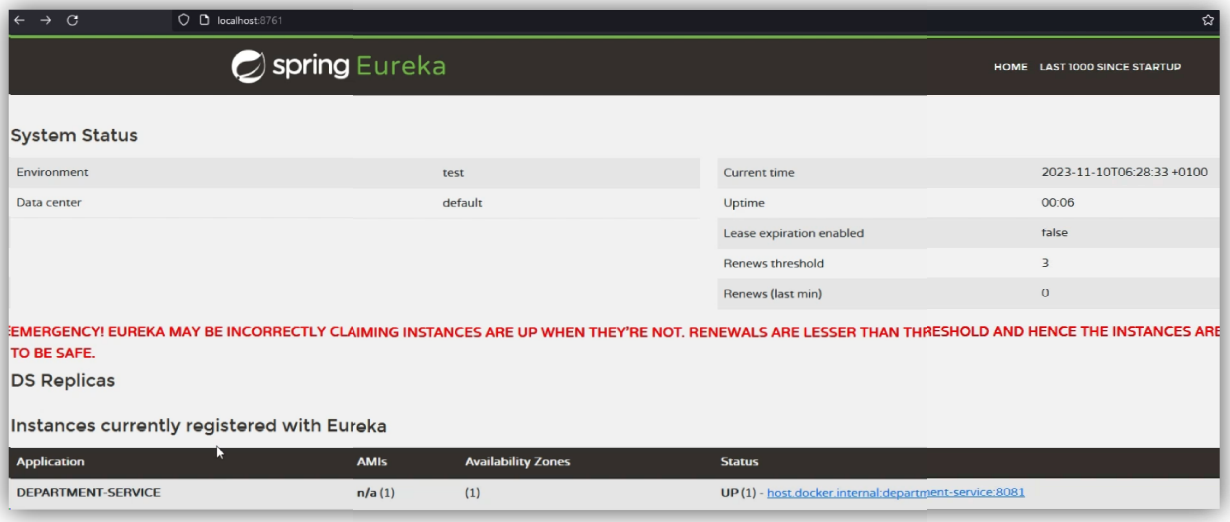
(2) config-server : depuis le fichier :

(CONFIG-SERVER\src\main\java\ConfigServerApplication.java)

(3) department-service : depuis le fichier :

(DEPARTMENT-SERVICE\src\main\java\DepartmentServiceApplication.java)

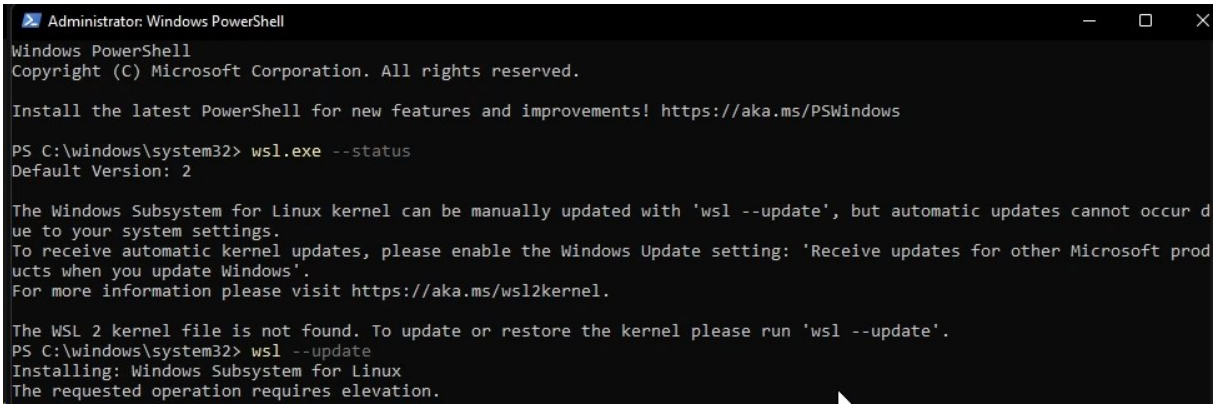
83. Rafraîchir la page **localhost:8761** et notez de la présence de (DEPARTMENT-SERVICE) dans la liste des applications reconnues :



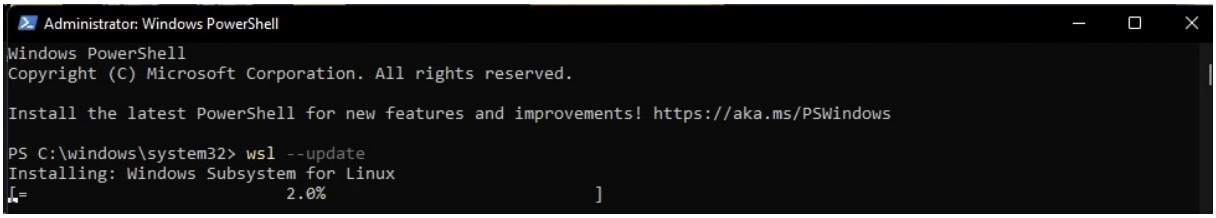
VIII. Création du Serveur ZIPKIN

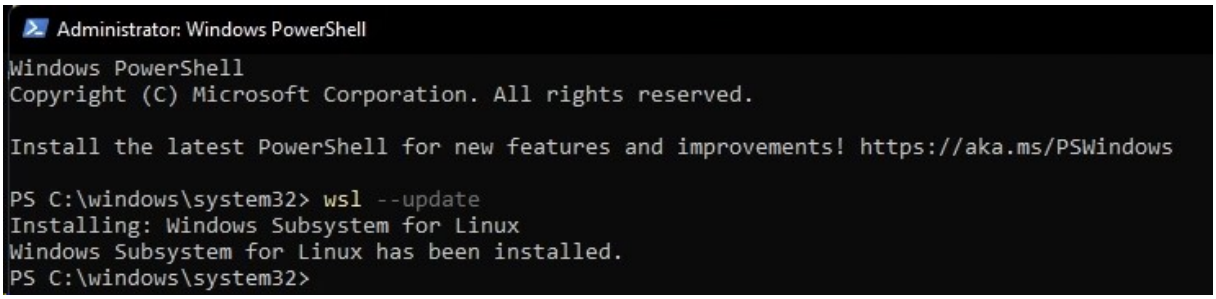
Pré-requis :

- Installation de Docker Desktop
- Installation de WSL2 :
  - depuis le terminal, lancez la commande suivante :
    - wsl --install -d Ubuntu-20.04

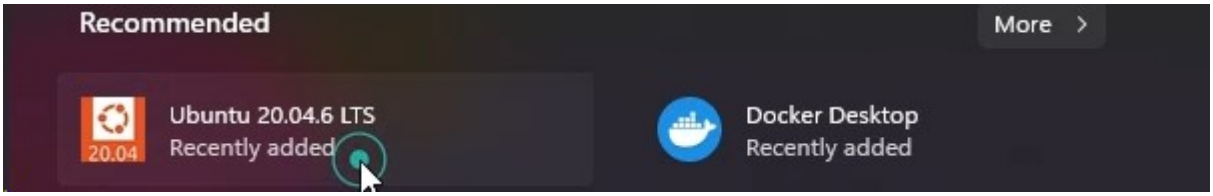


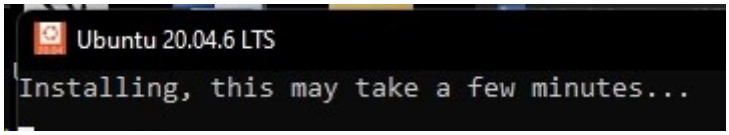
Redémarrez votre PC en cas de problèmes.





Lancez Ubuntu 20.04.6 LTS depuis le menu Démarrer:





```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username:
```

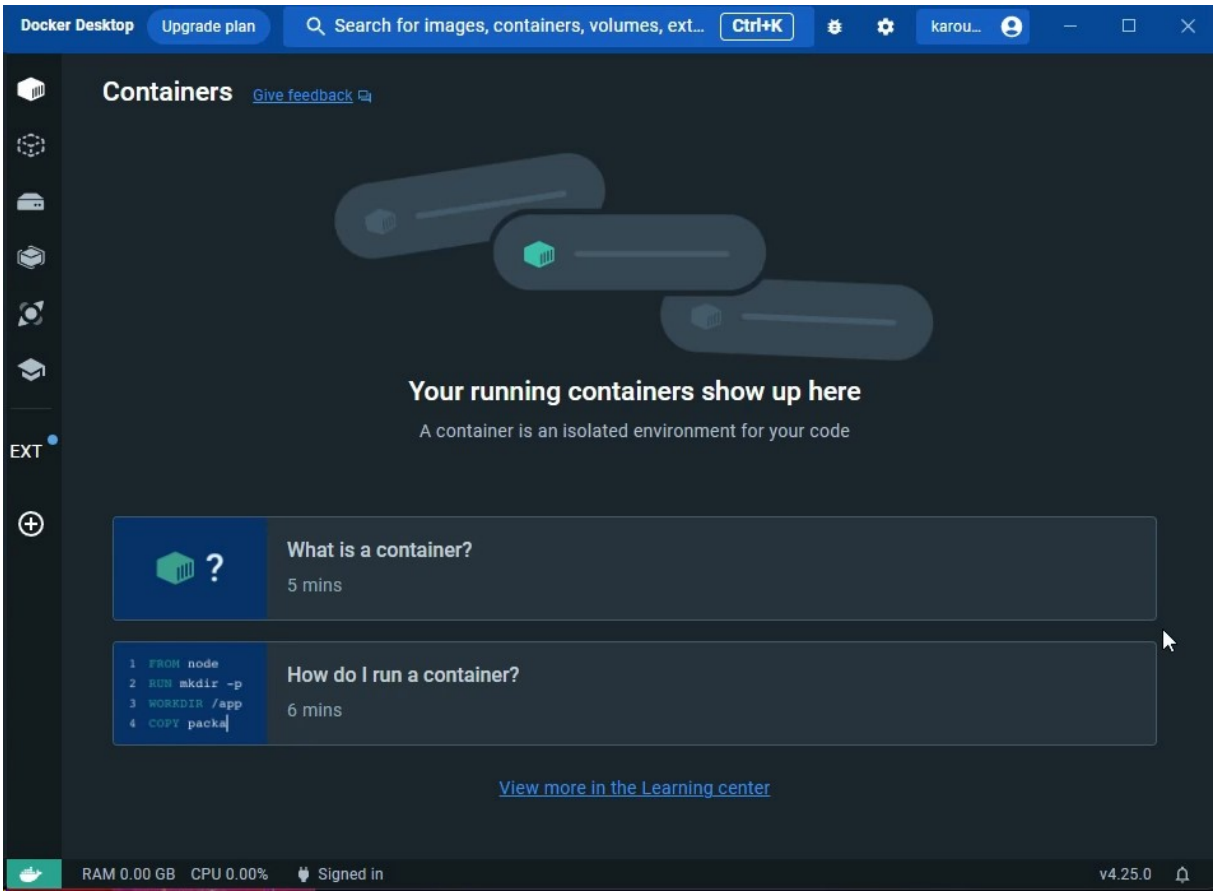
Saisir admin, puis tapez votre mot de passe :

```
Enter new UNIX username: sa
New password:
Retype new password:
passwd: password updated successfully
The operation completed successfully.
Installation successful!
```

Et vous aurez un message de bienvenu :

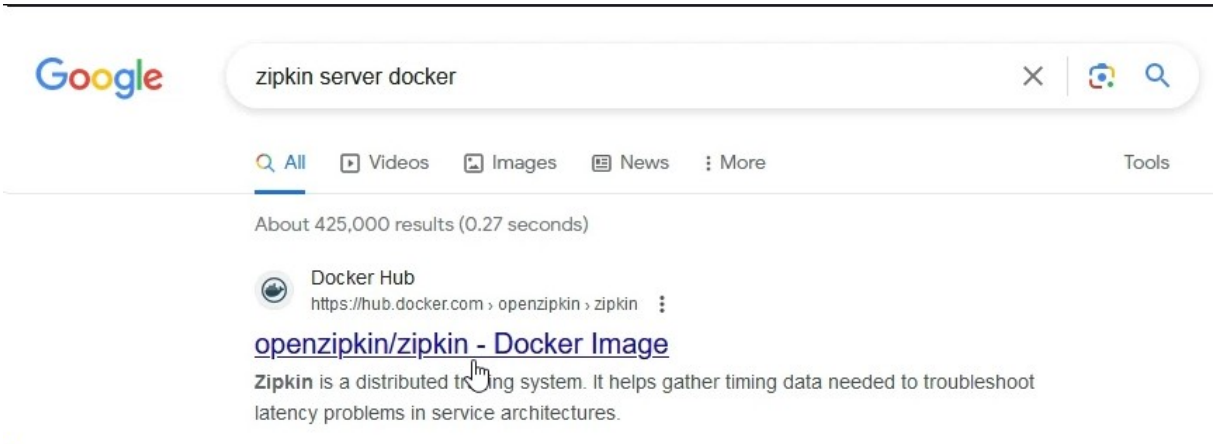
```
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Nov 12 09:46:33 CET 2023

  System load:  0.59               Processes:             77
  Usage of /:   0.1% of 1006.85GB  Users logged in:       0
  Memory usage: 3%                 IPv4 address for eth0: 172.29.235.163
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


This message is shown once a day. To disable it please create the
/home/sa/.hushlogin file.
sa@BEN:~$
```

Lancez l'application Docker Desktop :

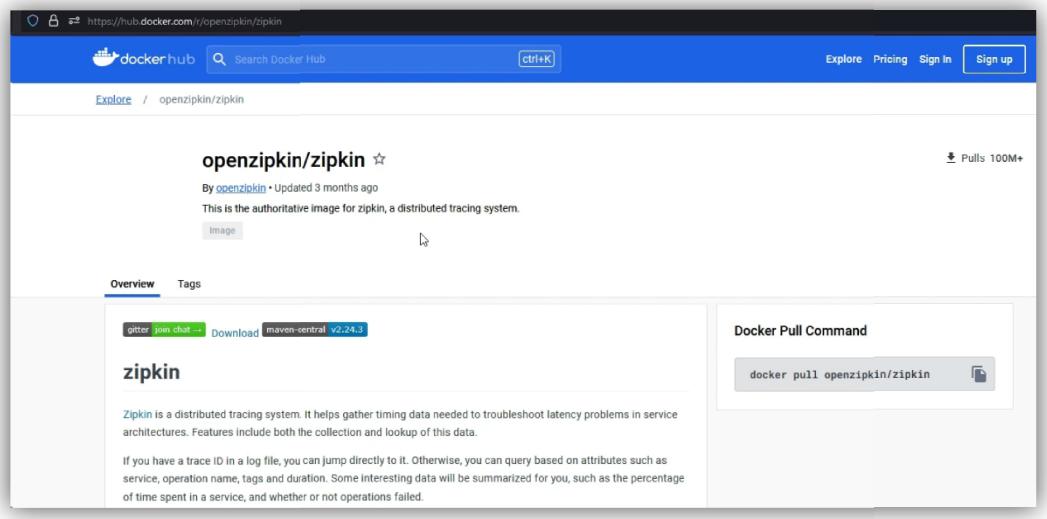


Installation de ZIPKIN:

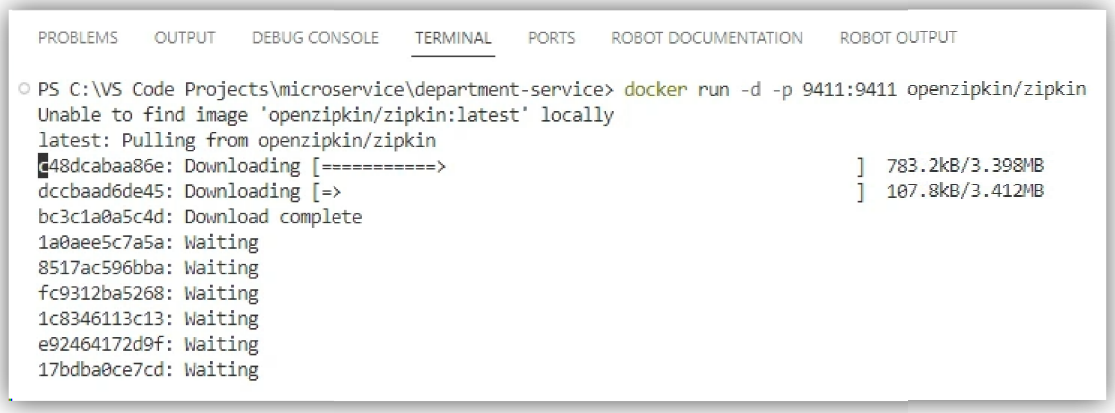
84. Ouvrir le navigateur et cherchez l'image du docker du serveur ZIPKIN :



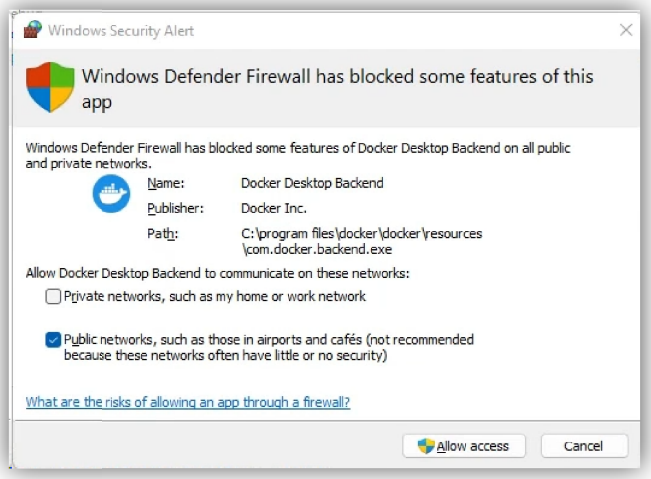
85. Cliquez sur le lien ci-dessus pour accéder à l'image du docker du serveur ZIPKIN :



86. Ouvrir un nouveau terminal sous Microsoft Visual Studio Code, puis lancez le docker ZIPKIN depuis la commande suivante :



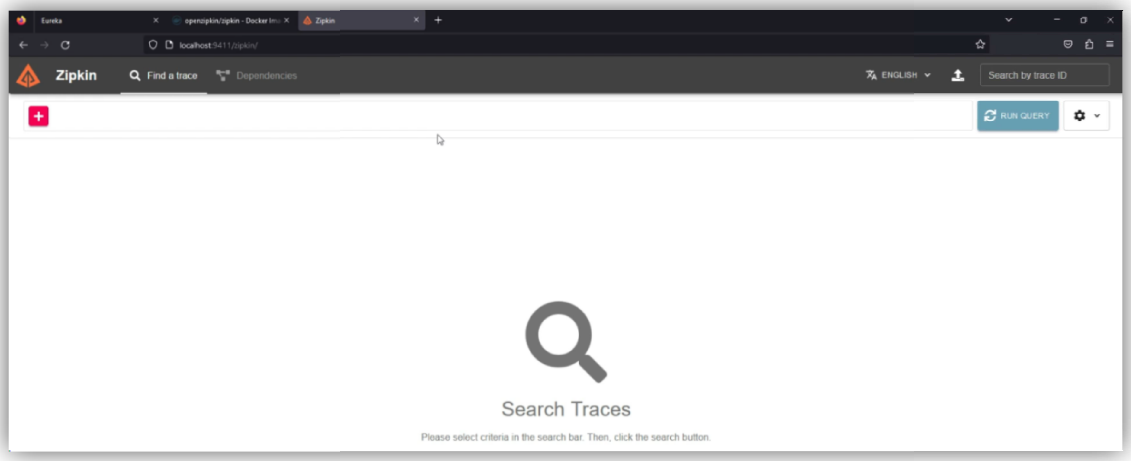
87. Autorisez l'accès :



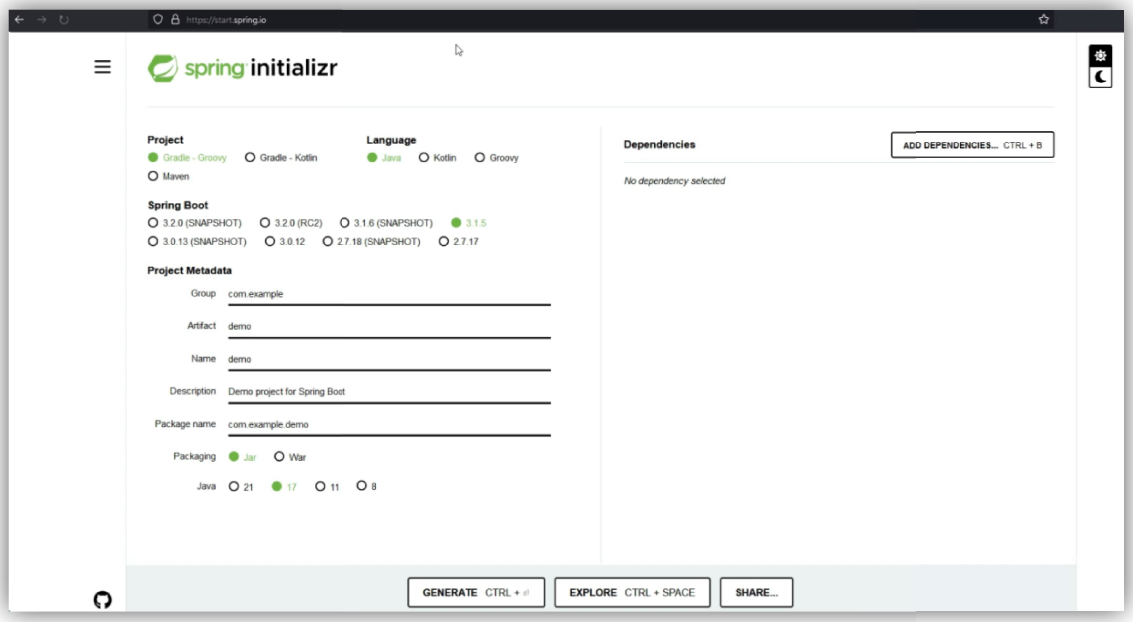
88. Testez l'adresse suivante :

localhost:9411/zipkin/

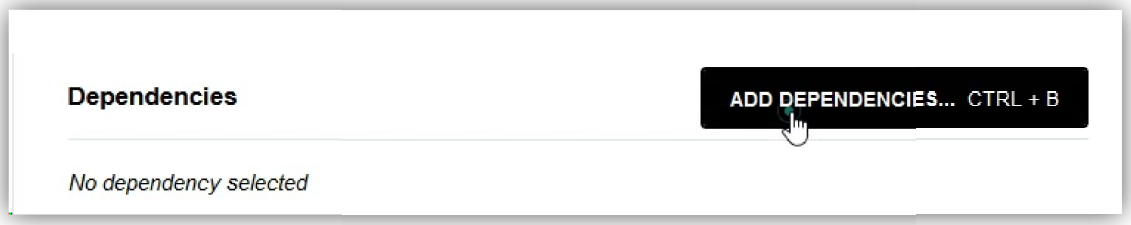
89. Voici le résultat :



90. Allez à l'adresse suivante : https://start.spring.io

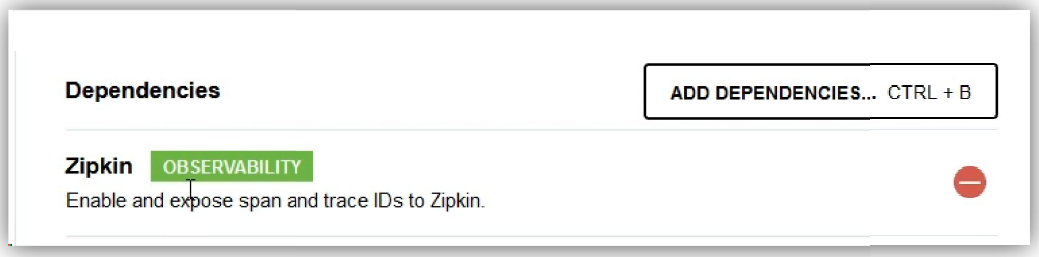


91. Cliquez sur (ajouter dépendances) :

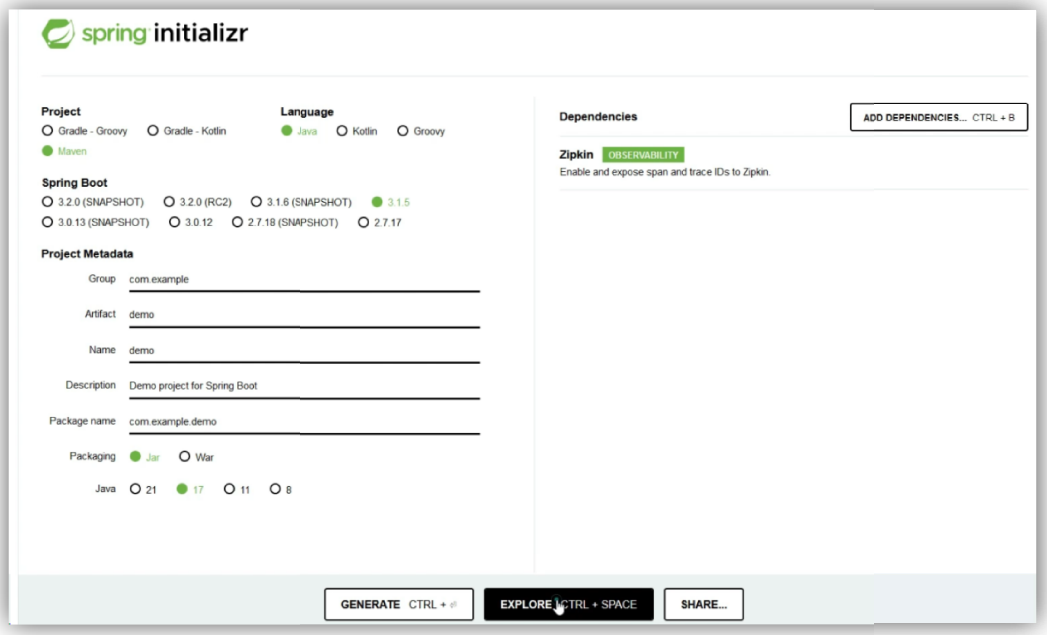


92. Saisir (zipkin) puis cliquez sur le lien de la dépendance:

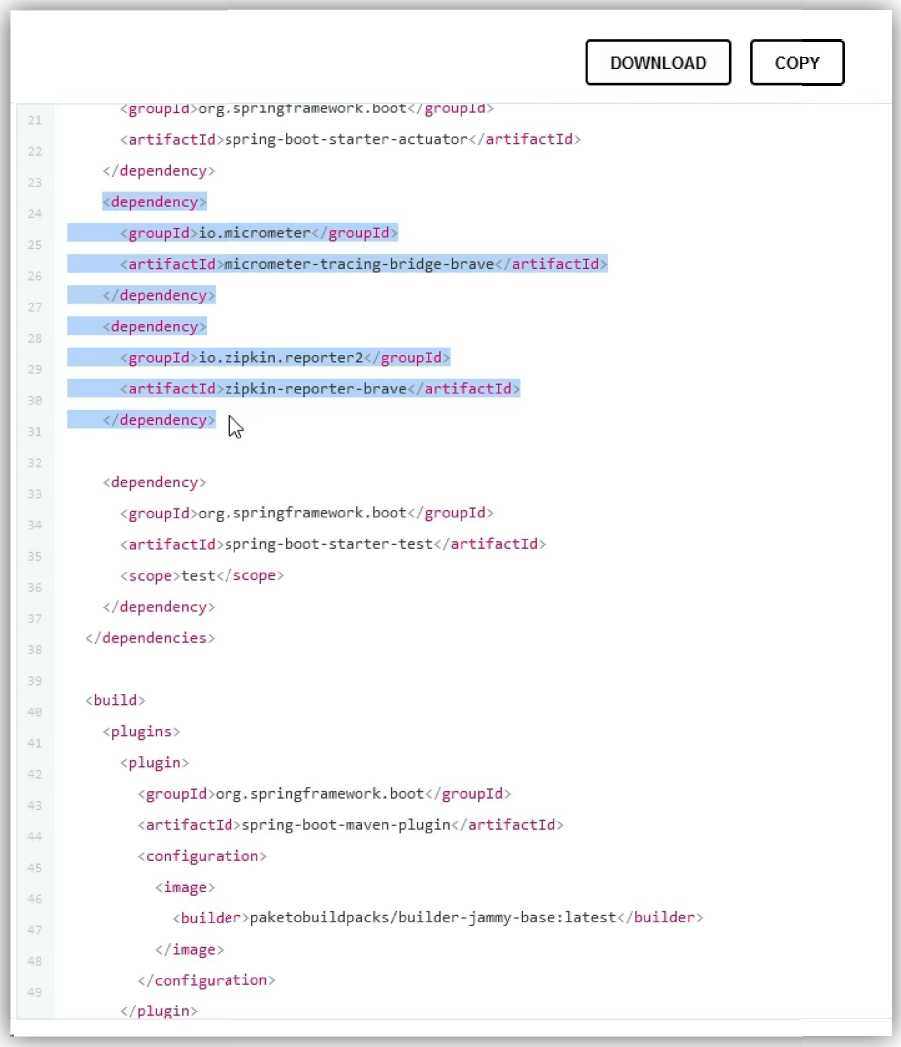
93. Vous sauriez redirigés vers ceci :



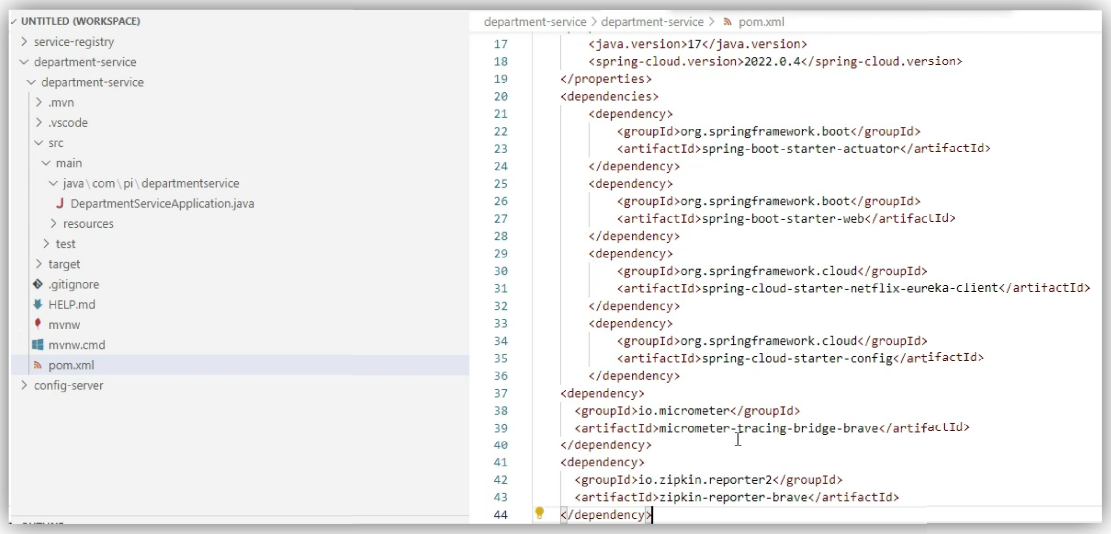
94. Cliquez sur (Explore) :



95. Copiez les deux dépendances suivantes (micrometer et zipkin):



96. Collez ces dépendances dans le fichier pom.xml du projet (department-service)



97. Modifiez le fichier de configuration de (department-service.yaml) dans le projet (config-server) en ajoutant la partie (management) indiquant qu'il faut tracer toutes les actions sur ZIPKIN (probability :1.0)

config-server > config-server > src > main > resources > config > ! department-service.yaml

```yaml
server:
    port: 8081

eureka:
    instance:
        hostname: localhost
    client:
        serviceUrl:
            defaultZone: http://${eureka.instance.hostname}:8761/eureka/
management:
    tracing:
        sampling:
            probability: 1.0
```

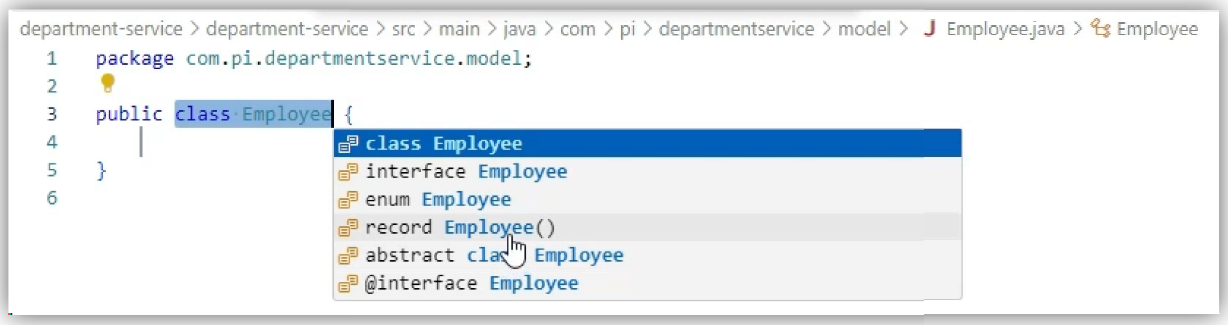
98. Redémarrez (**config-server**) ainsi que (**department-service**) :

IX. Implémentation du Microservice « Department »

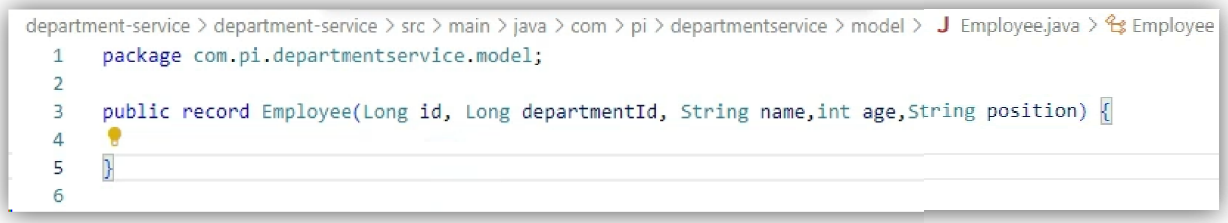
99. Sous le dossier (C:\VS Code Projects\microservice\department-service\department-service\src\main\java\com\pi\departmentservice) créez les répertoires suivants :
    - controller ;
    - repository ;
    - model ;
100. Sous le répertoire (model) créez une classe appelée (**Department.java**), écrire le code suivant :

```
vice > department-service > src > main > java > com > pi > departmentservice > model > J Department.java
1   package com.pi.departmentservice.model;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   public class Department {
7       private Long id;
8       private String name;
9       private List<Employee> employees = new ArrayList<>();
```

101. Sous le répertoire (model) créez une classe appelée (Employee.java) et marquez le en tant que Record :



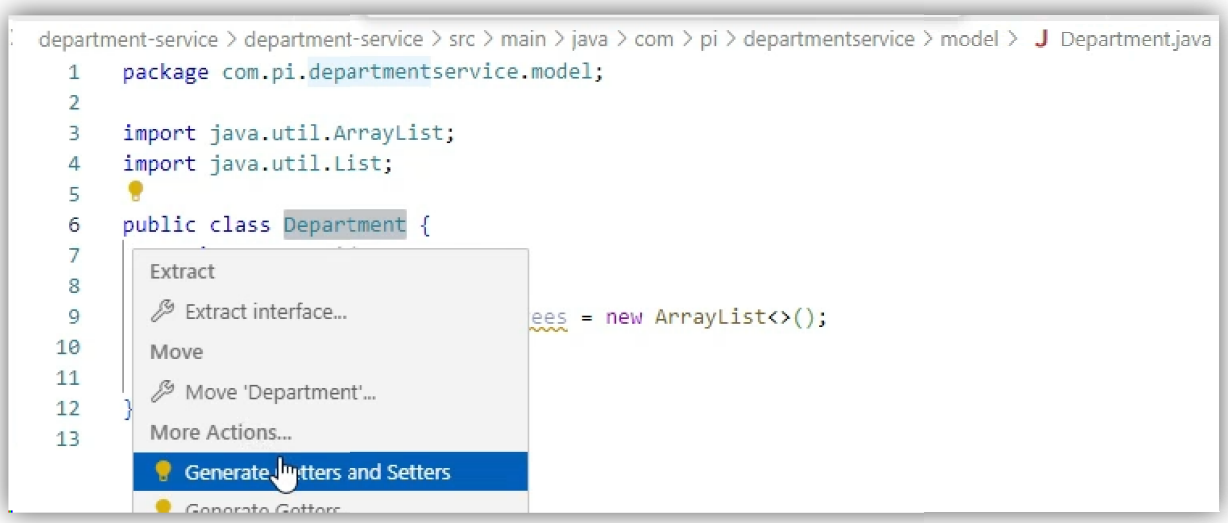
102. Modifiez le code pour qu'il soit comme suit :



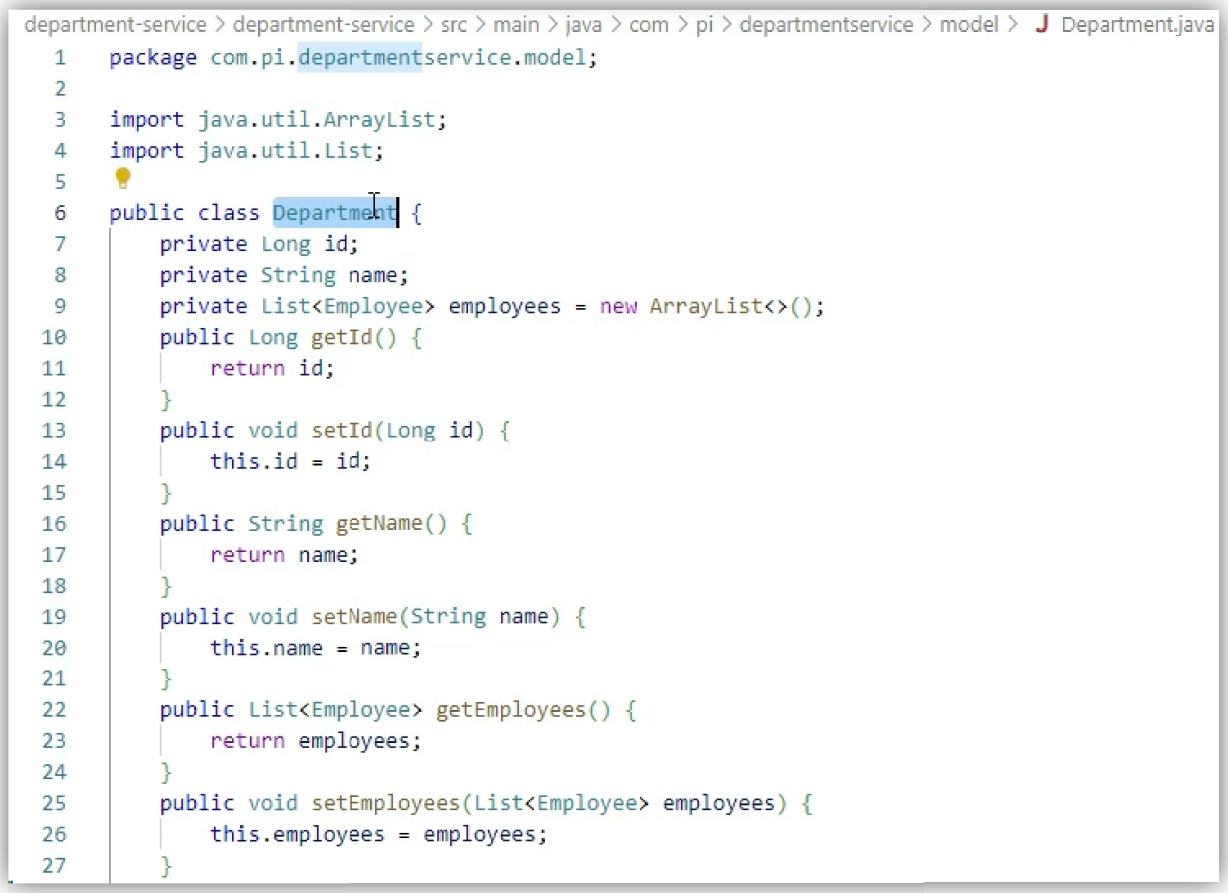
Génération des Getters and Setters :

Première Méthode :

103. Allez au fichier (Department.java), sélectionnez le nom de la classe, puis click sur l'ampoule jaune à gauche, ensuite cliquez sur (Generate Getters and Setters)

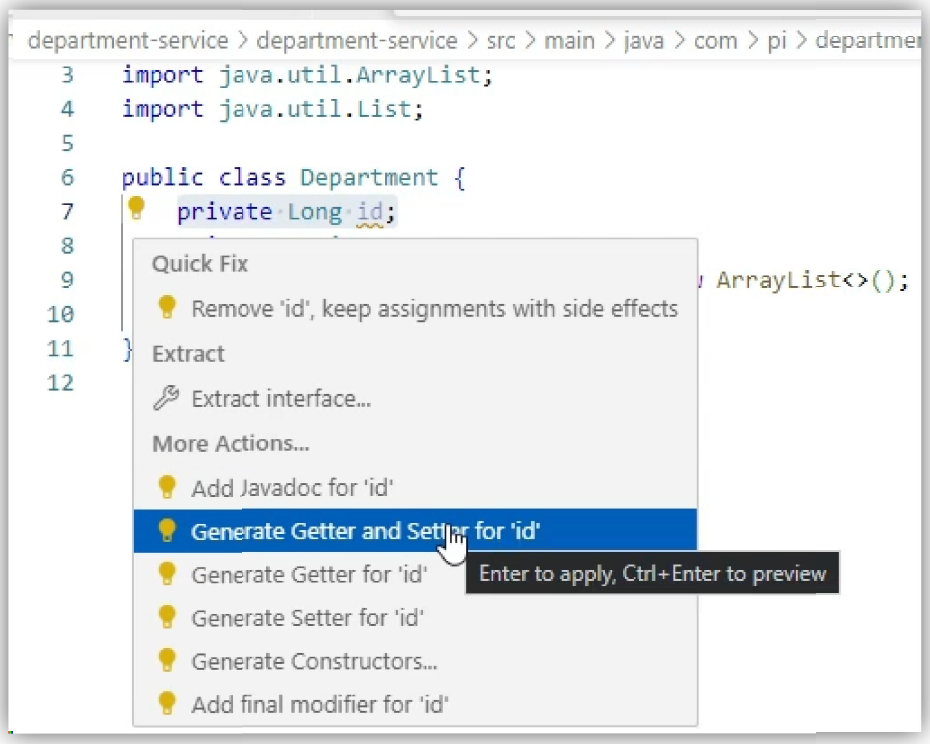


104. Voici le résultat :



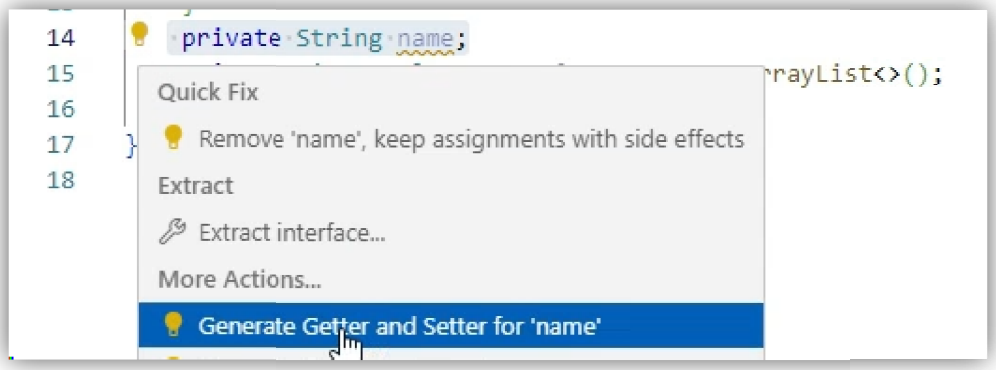
Deuxième méthode :

Allez au fichier (Department.java), sélectionnez la ligne qui contient le id, puis click droit pour lancer le menu contextuel ensuite cliquez sur (Generate Getter and Setter for « id »)

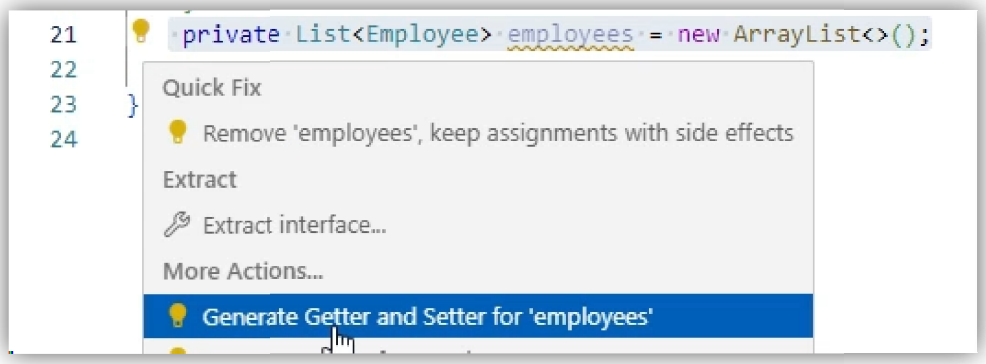


Et voici le résultat :

Refaire la même chose avec l'attribut (name) :



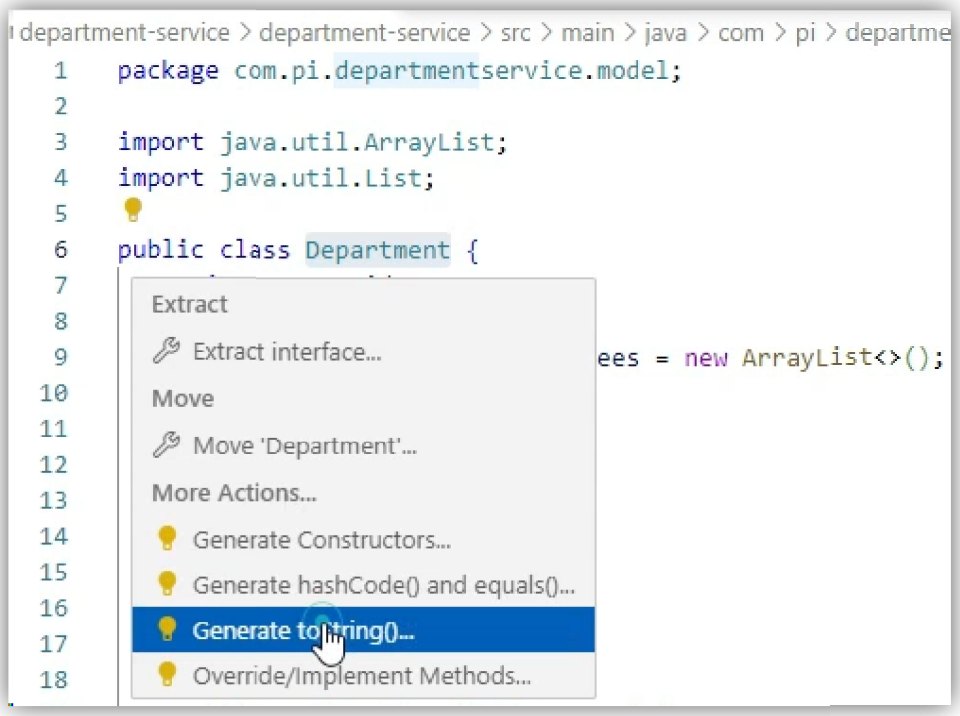
De même pour employees :



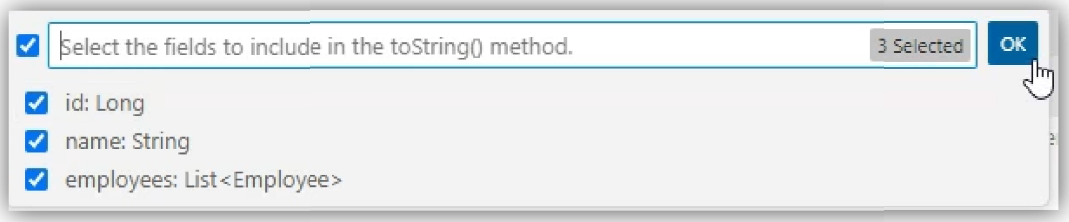
Et voici le code final après génération des Getters et Setters :

Génération de la méthode toString() :

105. Cliquez sur l'ampoule jaune à gauche, ensuite cliquez sur (**Generate toString()…**)



106. Gardez la sélection des trois attributs et cliquez sur (**OK**):

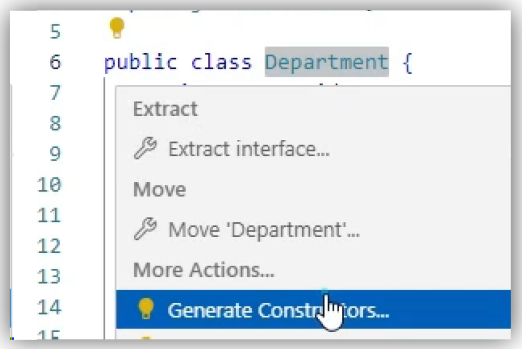


107. Et voici le résultat :

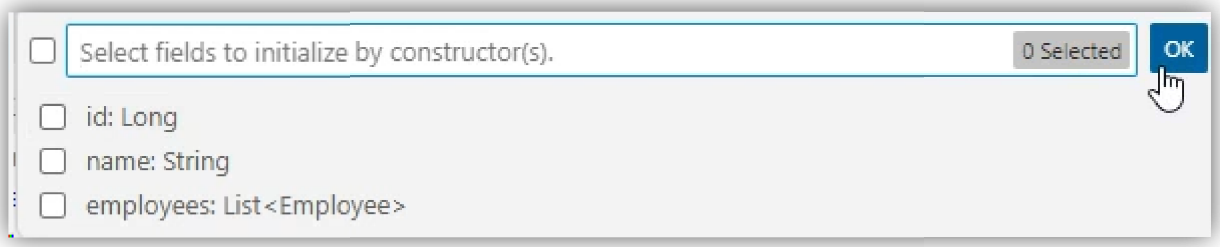
```
28    @Override
29    public String toString() {
30        return "Department [id=" + id + ", name=" + name + ", employees=" + employees + "]";
31    }
```

Génération des constructeurs :

108. Cliquez sur l'ampoule jaune à gauche, ensuite cliquez sur (**Generate Constructors**…)



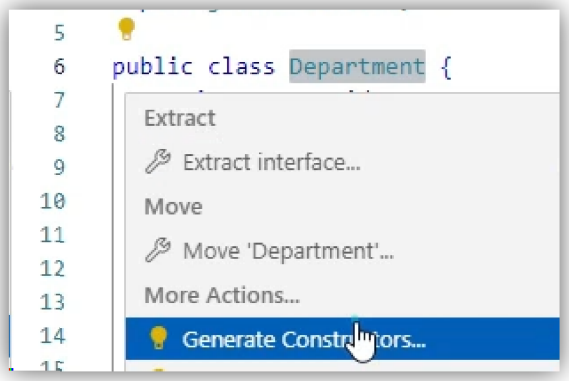
109. Puis créez un constructeur vide (gardez la sélection vide):



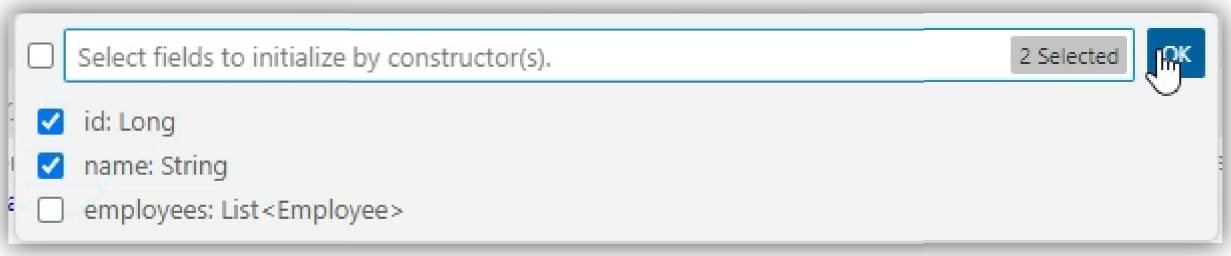
110. Voici le résultat de la génération:

```
6   public class Department {
7       private Long id;
8       private String name;
9       private List<Employee> employees = new ArrayList<>();
10      public Department() {
11      }
```

111. Cliquez sur l'ampoule jaune à gauche, ensuite cliquez sur (Generate Constructors…) afin de créer un deuxième constructeur :



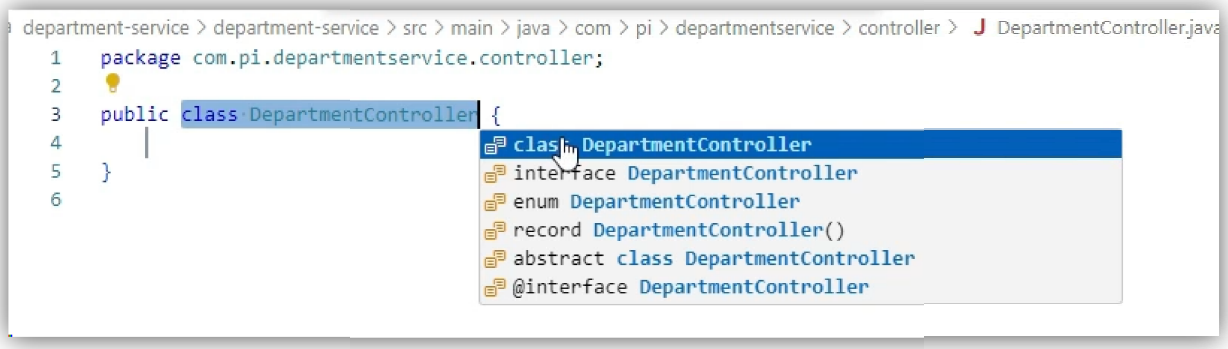
112. Sélectionnez les attributs « id » et « name », puis cliquez sur (OK) :



113. Et voici le résultat :

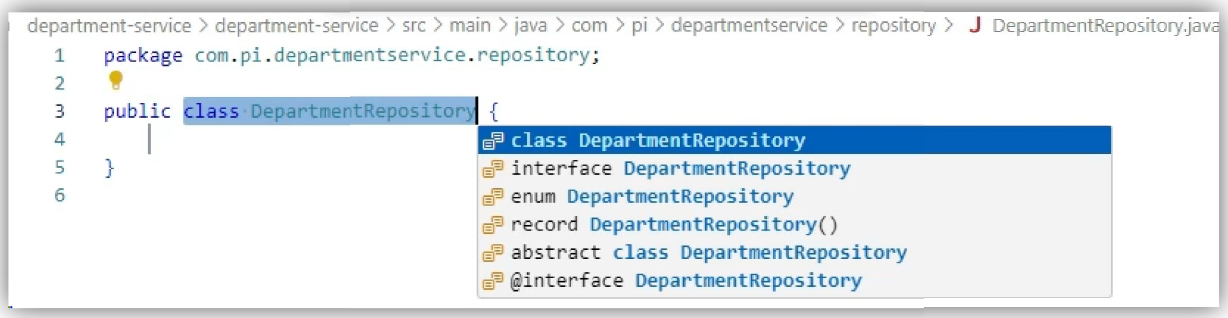
```java
public class Department {
    private Long id;
    private String name;
    private List<Employee> employees = new ArrayList<>();
    public Department(Long id, String name) {
        this.id = id;
        this.name = name;
    }
```

114. Au niveau du dossier (**controller**), ajouter un nouveau fichier appelé (DepartmentController.java) :



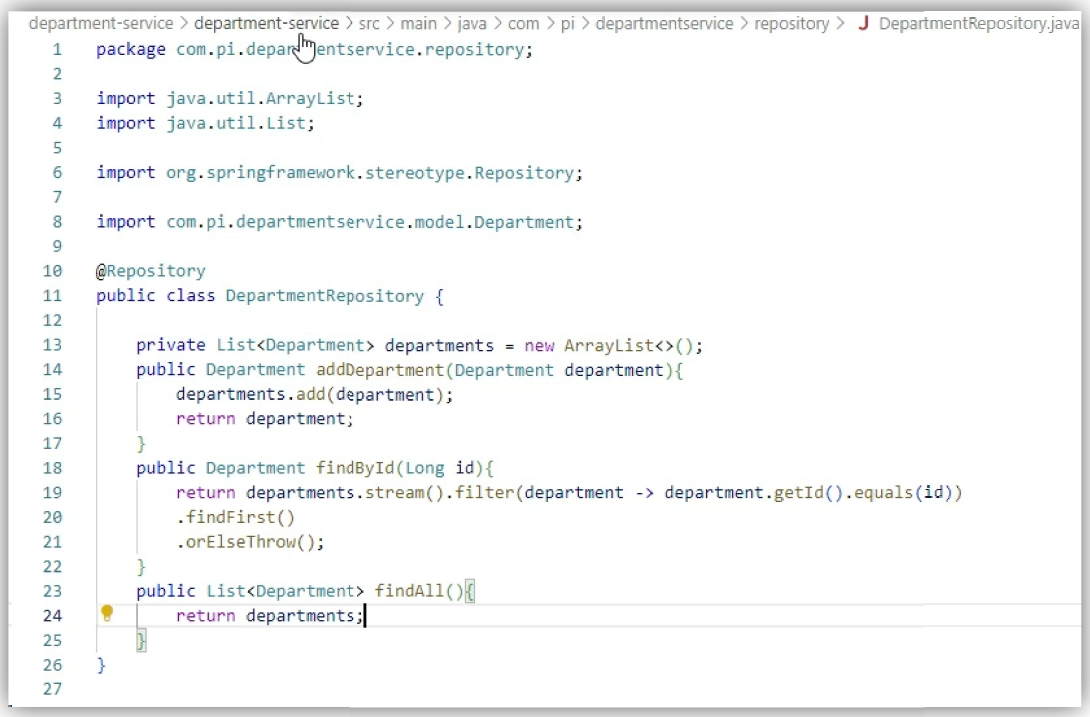
115. Ajouter l'annotation (@RestController) et (@RequestMapping("/department")):

116. Au niveau du dossier (**repository**), ajouter un nouveau fichier appelé (**DepartmentRepository.java**) :



117. Ajouter l'annotation (**@Repository**):

118. Ecrire le code suivant qui implémente les méthodes (**addDepartment**),(**findById(Long id)**) et **findAll()**):



119. Implémentez la classe (**DepartmentController**) pour qu'elle soit comme suit :

DepartmentController.java 1 ●

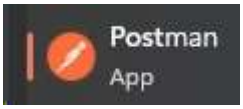
department-service > department-service > src > main > java > com > pi > departmentservice > controller > DepartmentController.java >

```java
import com.pi.departmentservice.repository.DepartmentRepository;

@RestController
@RequestMapping("/department")
public class DepartmentController {
    private static final Logger LOGGER = LoggerFactory.getLogger(clazz:DepartmentController.class);
    @Autowired
    private DepartmentRepository repository;
    @PostMapping
    public Department add(@RequestBody Department department){
        LOGGER.info(format:"Department add:{}",department);
        return repository.addDepartment(department);
    }
    @GetMapping
    public List<Department> findAll(){
        LOGGER.info(msg:"Department find");
        return repository.findAll();
    }
    @GetMapping("/{id}")
    public Department findById(@PathVariable Long id){
        LOGGER.info(format:"Department find: id={}",id);
        return repository.findById(id);
    }
}
```

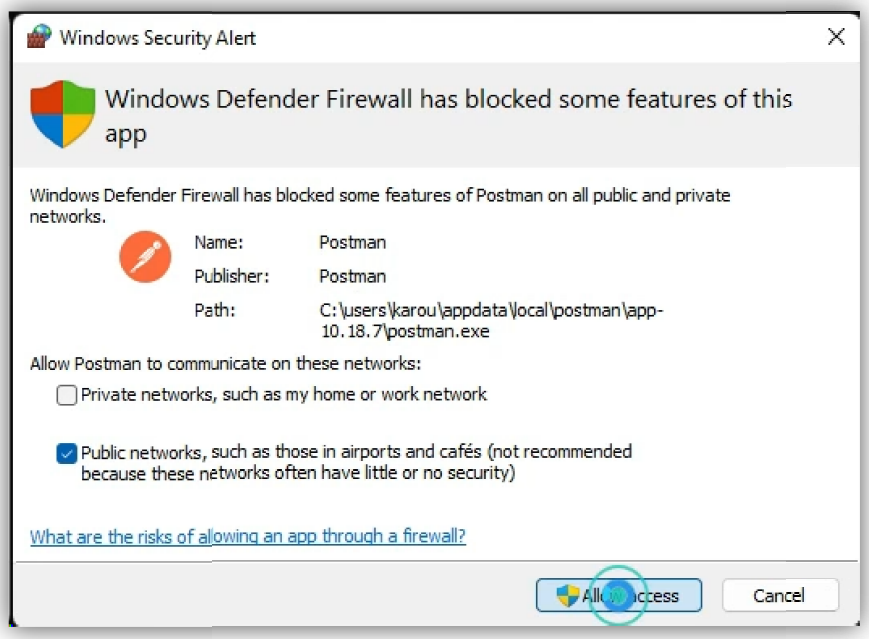
120. Redémarrez le microservice (**Department**) :

X. Test du Microservice « Department »

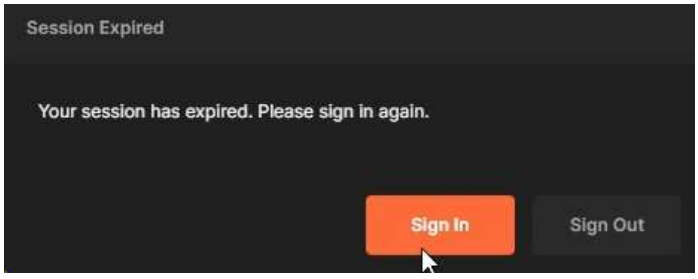
121. Afin de tester notre microservice (**Department**) on va lancer l'application (**Postman**) :



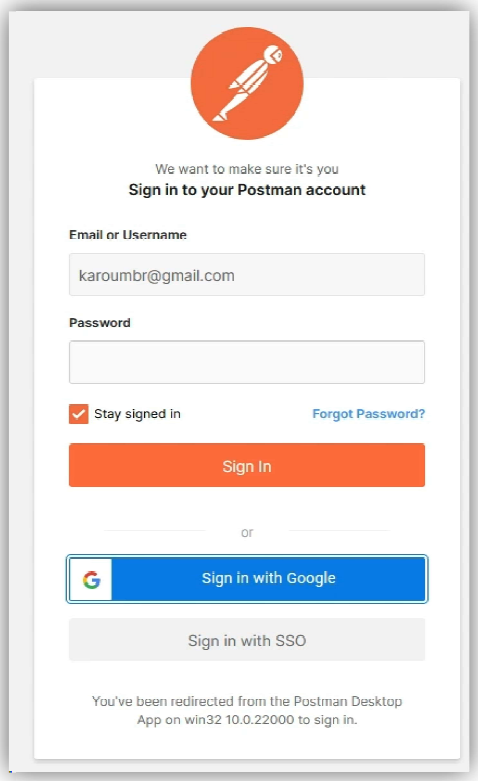
122. En cas de besoin, autorisez postman pour se connecter à Internet :



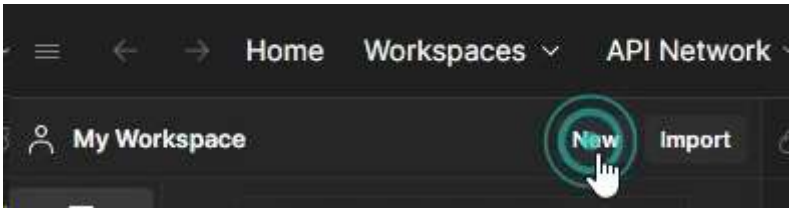
123. Connectez-vous à votre compte sur (**Postman**) :



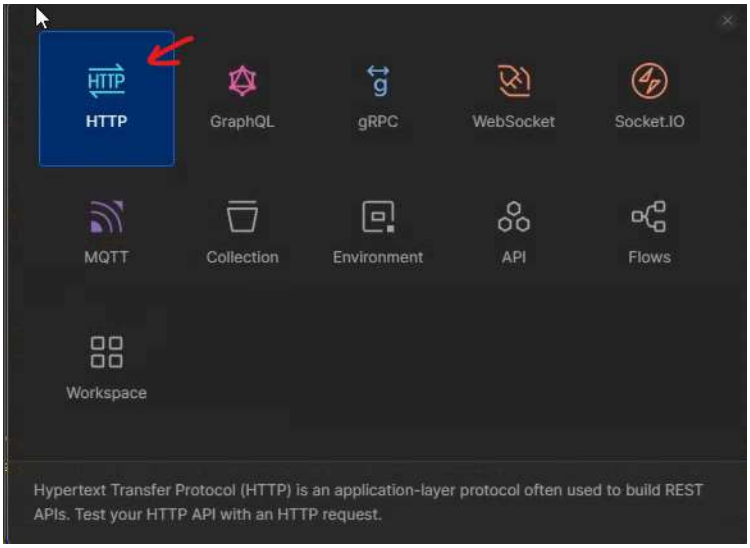
124. Entrez vos données d'authentification :



125. Créez une nouvelle connexion:

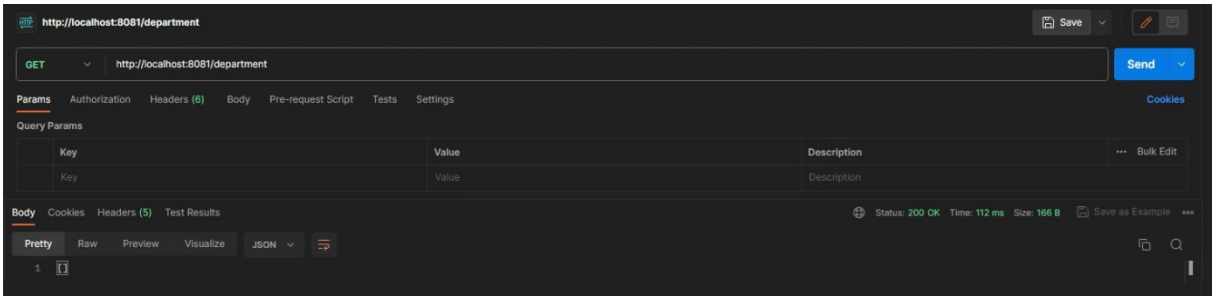


126. Sélectionnez (**HTTP**):

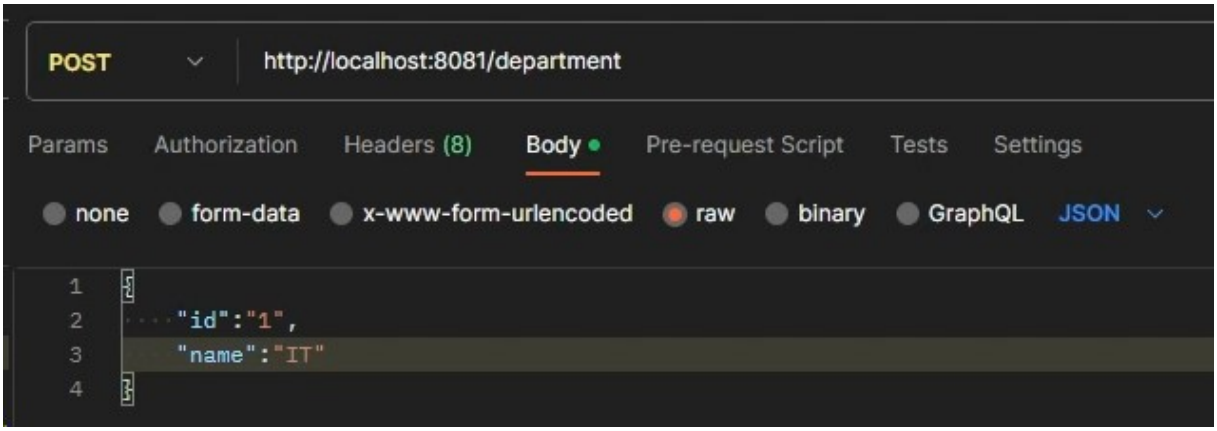


127. Choisir la méthode (GET) puis écrire l'adresse suivante et cliquez sur (SEND) :

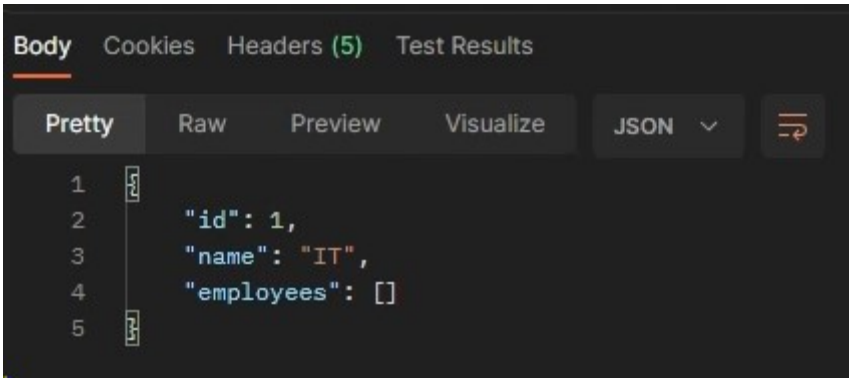
128. Et voici le résultat (un contenu vide):



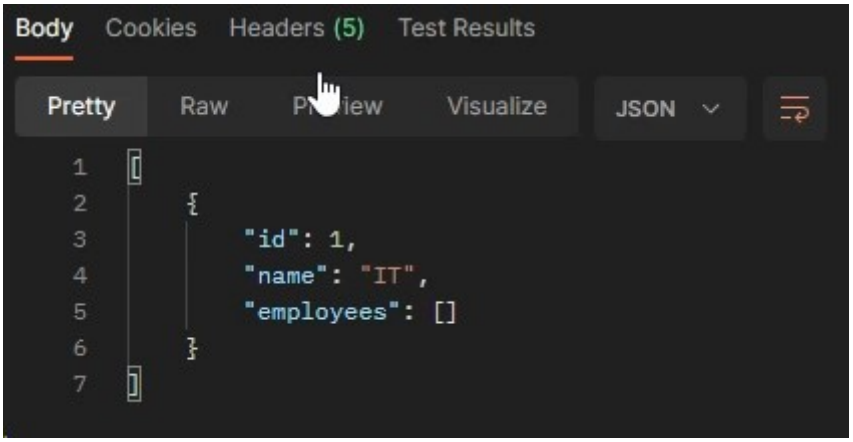
129. Sélectionnez la méthode (**POST**), puis cliquez sur l'onglet (**Body**), ensuite (**raw**) et sélectionnez (**JSON**) comme format d'échange :



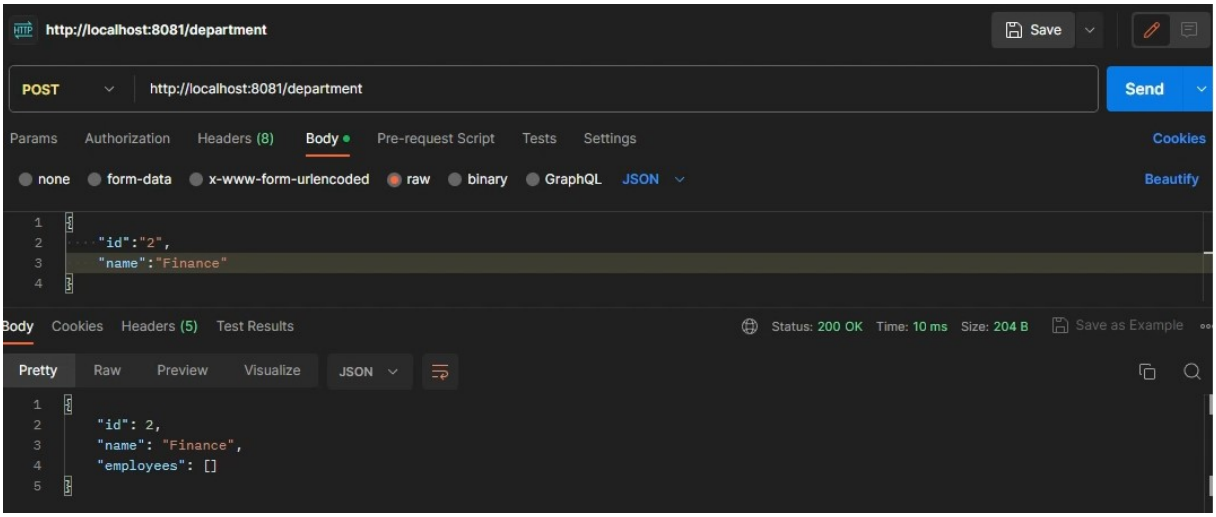
130. Cliquez sur (SEND) et vous aurez le résultat suivant :



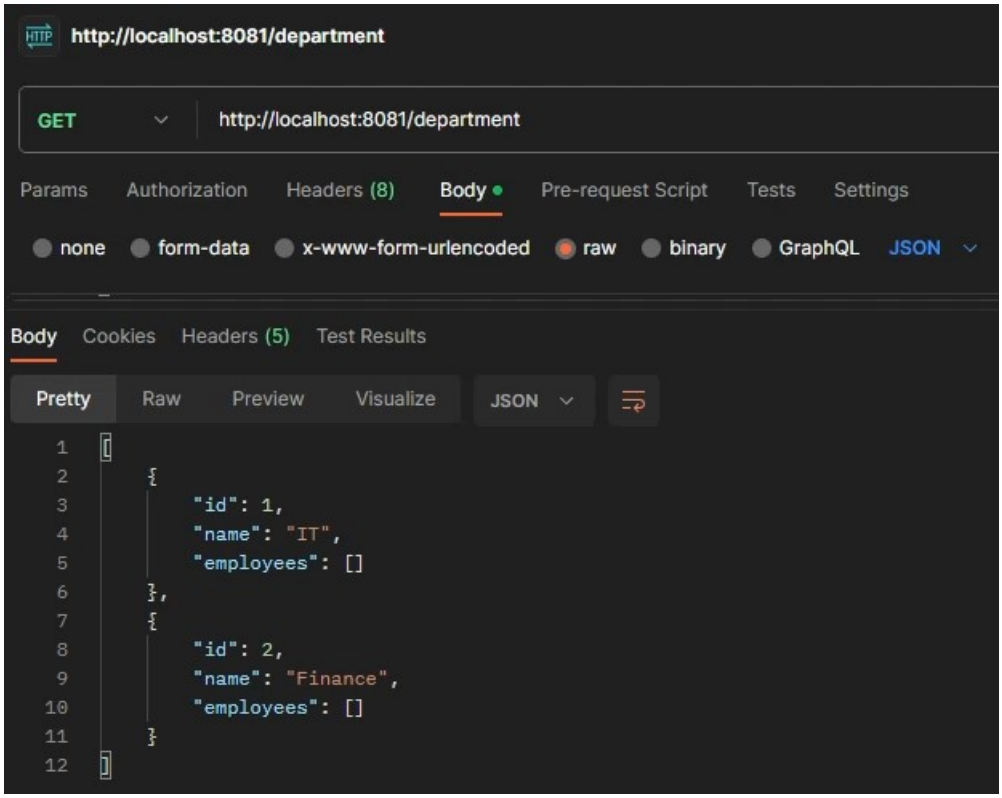
131. Testez la liste des départements par la méthode (GET) :



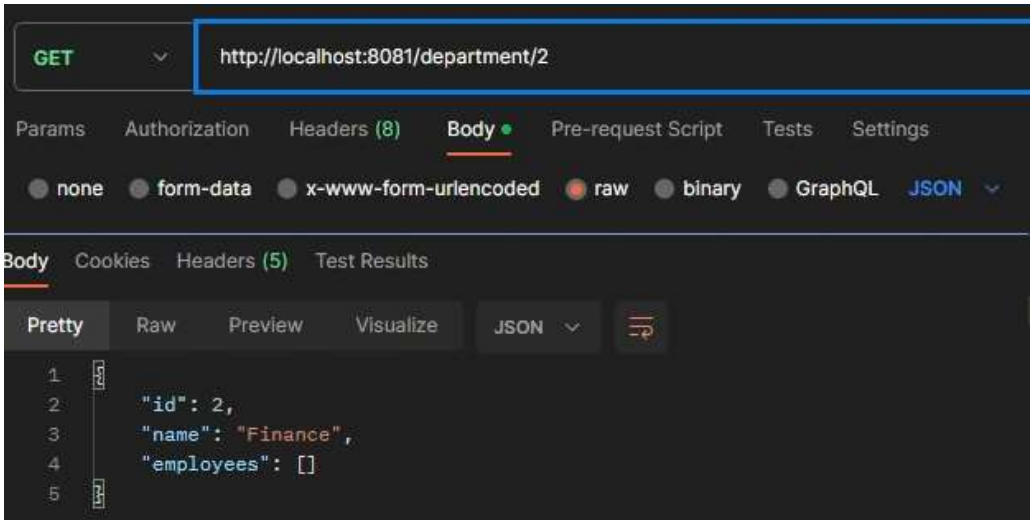
132. Ajoutez un deuxième département de id (2) et name (Finance) puis cliquez sur (SEND) :



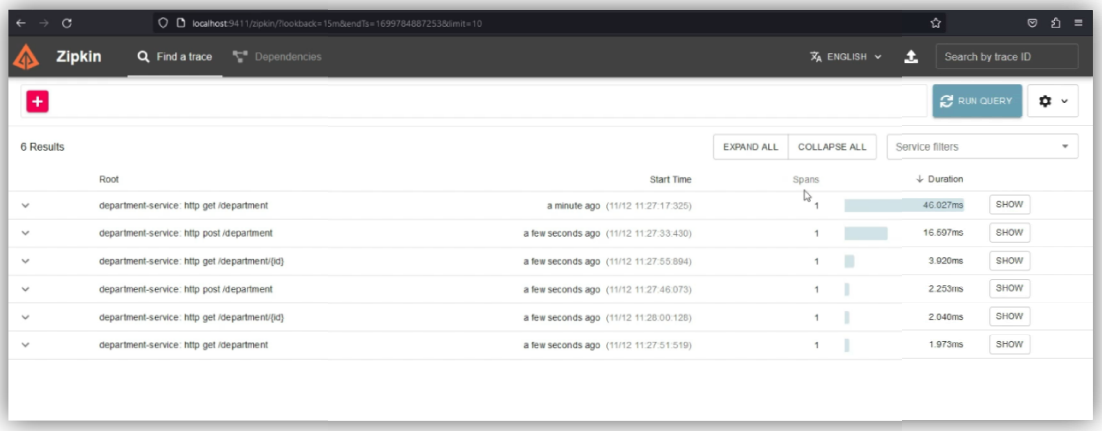
133. Testez de nouveau la méthode (GET) :



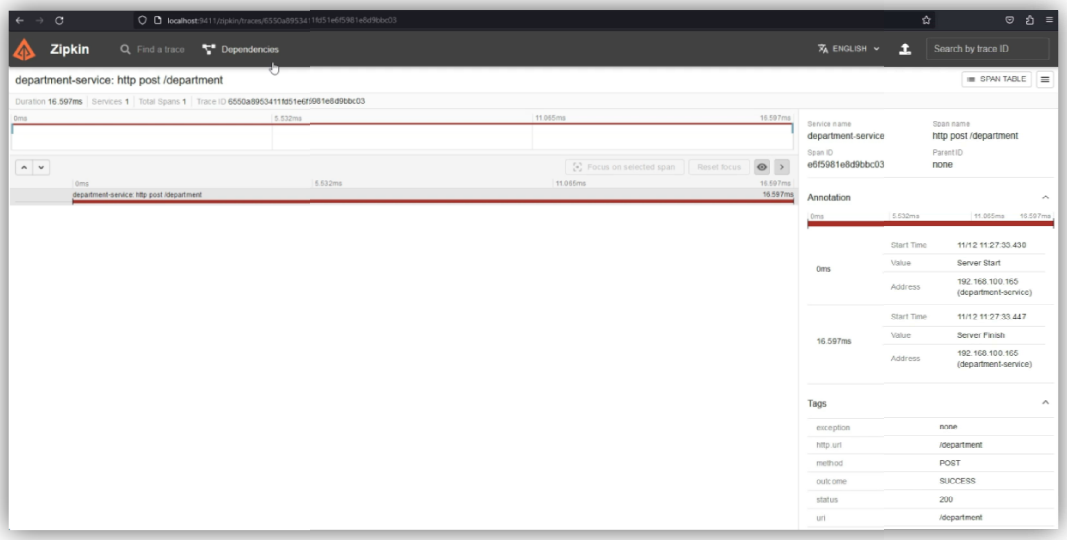
134. Testez la méthode (GET) en indiquant le paramètre id de valeur (2) :



135. Testez le service de traçage (**ZIPKIN**) en cliquant sur le bouton (**RUN QUERY**) :

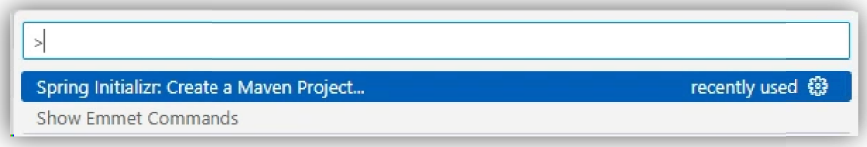


136. En cliquant sur (**SHOW**), on peut voir plus de détails sur l'action :

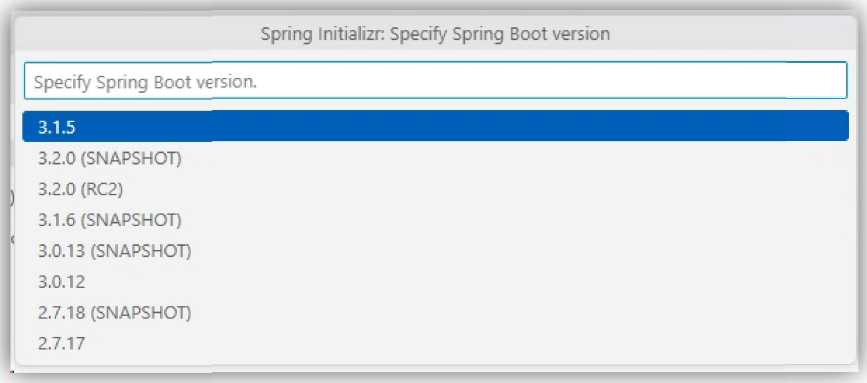


XI. Création du Microservice « Employee »

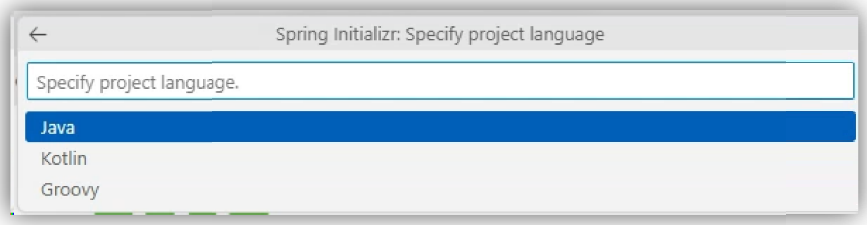
137. Appuyer sur **Ctrl + Shift + P** afin de lancer la palette des commandes.
138. Cliquez sur (**Spring Initializr : Create a Maven Project** …):



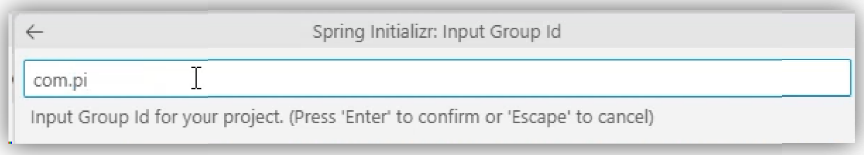
139. Sélectionnez la version (**3.1.5**)



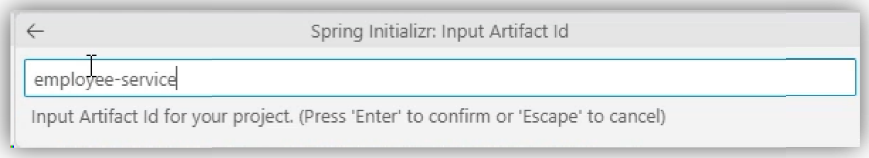
140. Sélectionnez le langage de programmation (**Java**)



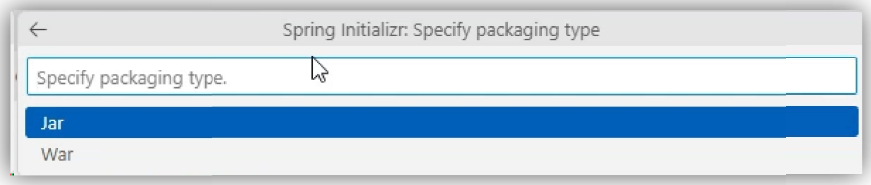
141. Choisir comme nom de package (**com.pi**)



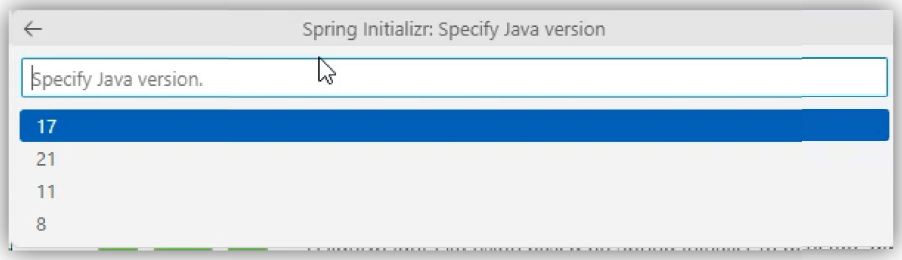
142. Indiquez le nom de projet (**employee-service**):



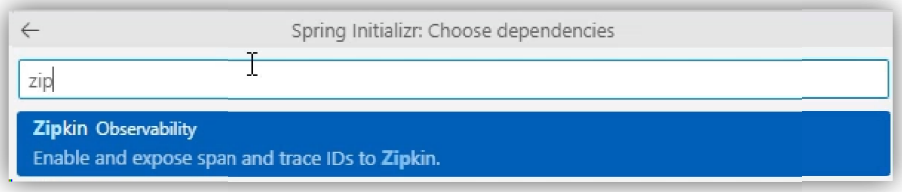
143. Choisir comme type de package (**Jar**)



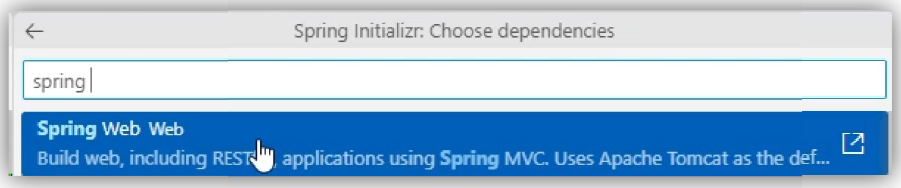
144. Choisir la version de Java Development Kit (**17**)



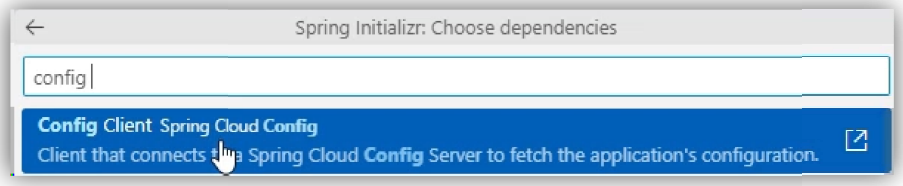
145. Ajoutez la dépendance (**Zipkin – Observability**)



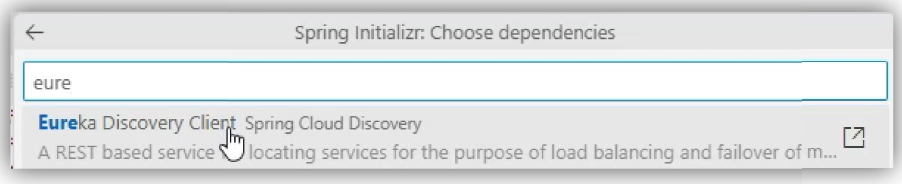
146. Ajoutez la dépendance (**Spring Web – Web**)



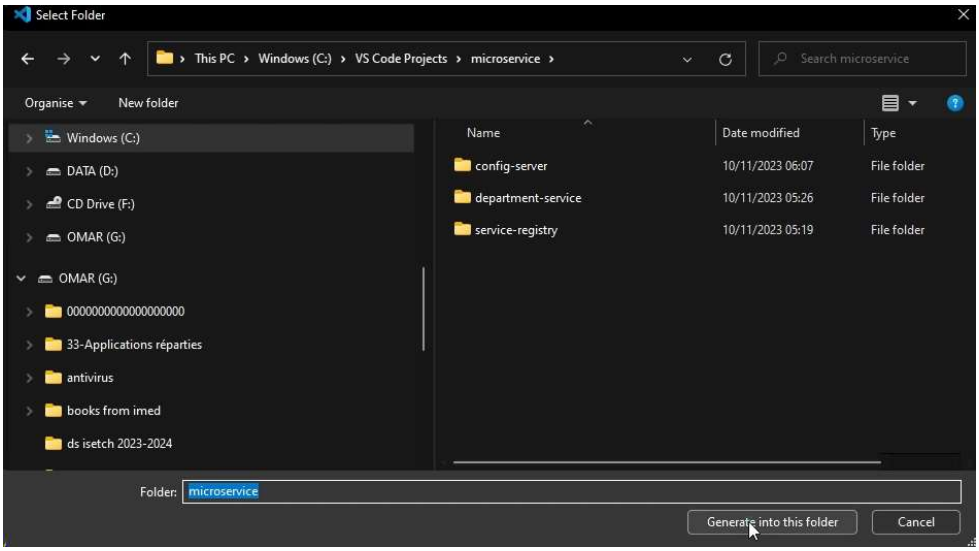
147. Ajoutez la dépendance (**Config Client – Spring Cloud Config**)



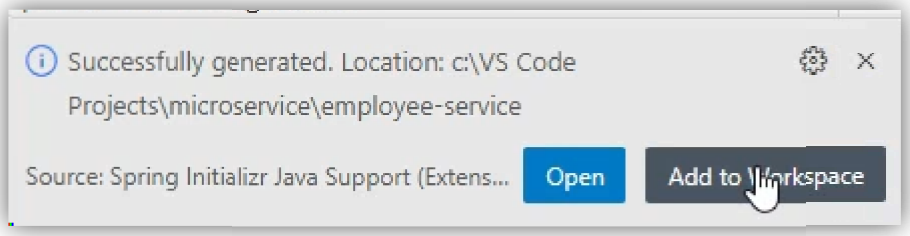
148. Ajoutez la dépendance (**Eureka Discovery Client – Spring Cloud Discovery**)



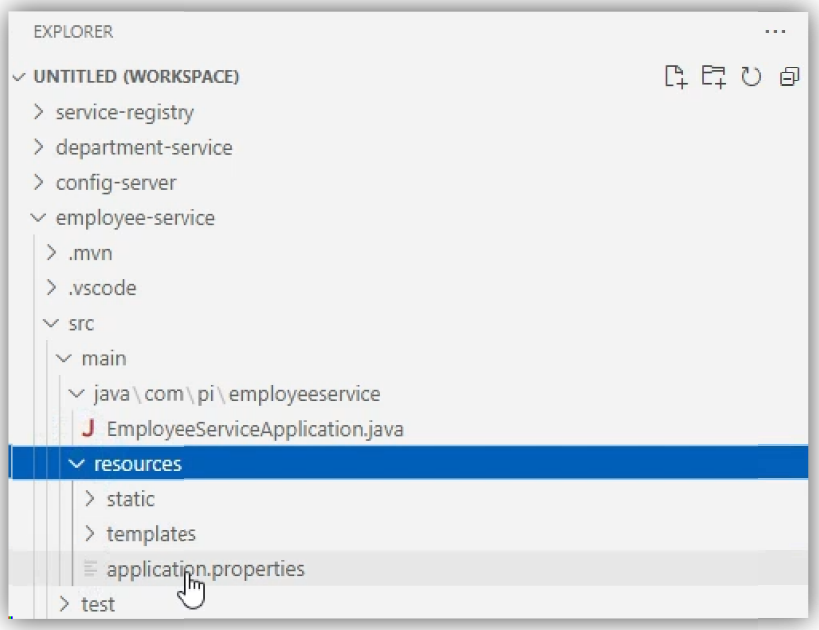
149. Créez un nouveau répertoire sous (**C:\VS Code Projects\microservice**) et appelez le (employee-service) puis cliquez sur (**Generate into this folder**)



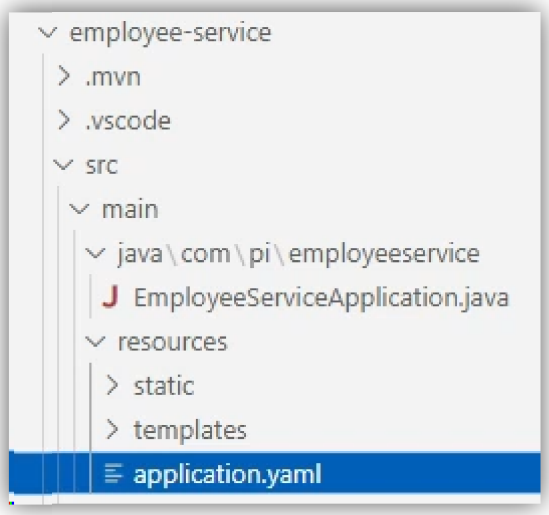
150. Ajouter ce dernier projet à l'espace de travail :



151. Changez l'extension du fichier (**application.properties**) pour qu'il devienne (**application.yaml**) :

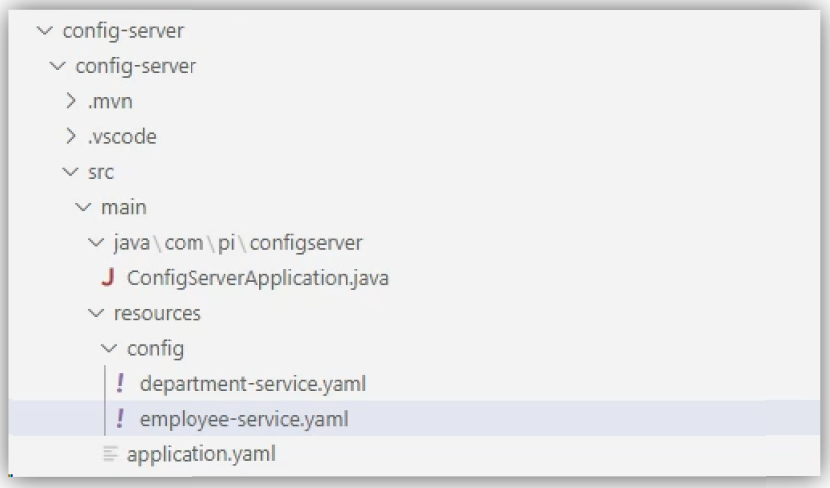


152. Et voici le résultat :



153. Changez le contenu du fichier (**application.yaml**) pour qu'il soit ainsi :

154. Au niveau du projet (config-server) ajouter un nouveau fichier au niveau de (**src\main\resources\config**) appelé (**employee-service.yaml**)



155. Et lui ajouter le code suivant (j'ai juste modifié le numéro de port par rapport au fichier de configuration du microservice department):

config-server > config-server > src > main > resources > config > ! employee-service.yaml

```yaml
server:
    port: 8082

eureka:
    instance:
        hostname: localhost
    client:
        serviceUrl:
            defaultZone: http://${eureka.instance.hostname}:8761/eureka/
management:
    tracing:
        sampling:
            probability: 1.0
```

156. Allez au fichier (C:\VS Code Projects\microservice\employee-service\src\main\java\com\pi\employeeservice\ EmployeeServiceApplication.java), puis ajouter l'annotation (**@EnableDiscoveryClient**)

EmployeeServiceApplication.java

employee-service > src > main > java > com > pi > employeeservice > EmployeeServiceApplication.java > ...

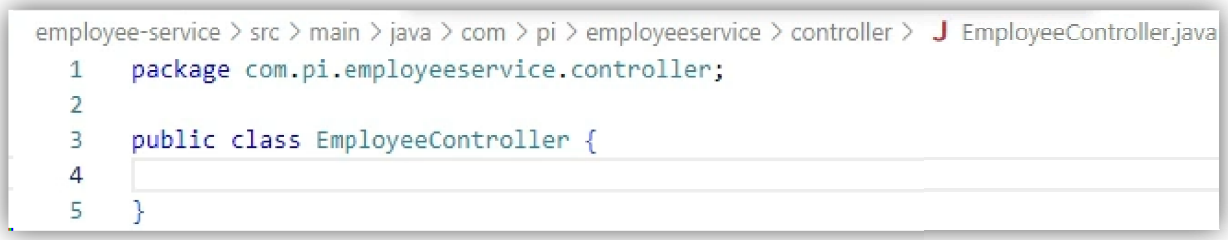
```java
package com.pi.employeeservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class EmployeeServiceApplication {


    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(primarySource:EmployeeServiceApplication.class, args);
    }

}
```

XII. Implémentation du Microservice « Employee »

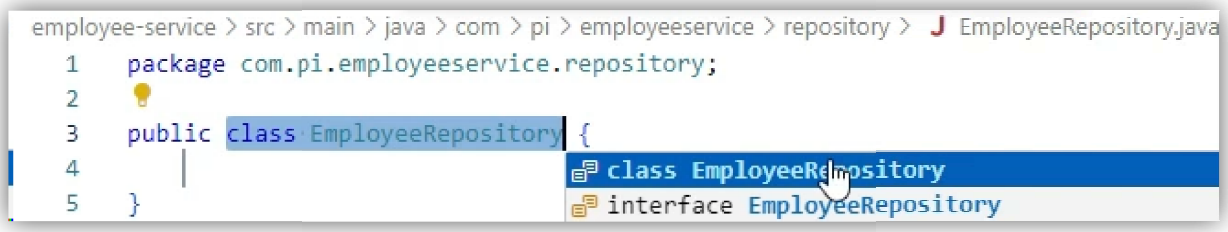
157. Sous le dossier (C:\VS Code Projects\microservice\employee-service\src\main\java\com\pi\employeeservice) créez les répertoires suivants :

- controller ;
- repository ;
- model ;

158. Implémentez la classe (**EmployeeController.java**) pour qu'elle soit comme suit :



159. Au niveau du dossier (repository), ajouter un nouveau fichier appelé (EmployeeRepository.java) :

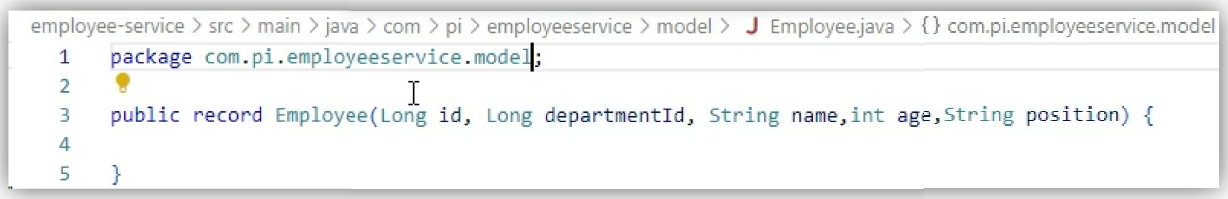


160. Ajouter l'annotation (@Repository) au classe (EmployeeRepository) :

161. Ajouter les annotations : (**@RestController**) et (@RequestMapping("/employee") au classe (**EmployeeController**) :

162. Copiez le modèle (**Employee.java**) du projet (**department-service**), puis coller une copie sous le dossier (**model**) du projet (**employee-service**) écrire le code suivant :

163. Corrigez le nom du package pour qu'il soit ainsi :



164. Allez au fichier appelé (**EmployeeRepository.java**) sous le dossier (**repository**) et écrire le code suivant :

employee-service > src > main > java > com > pi > employeeservice > repository > EmployeeRepository.java

```java
package com.pi.employeeservice.repository;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

import com.pi.employeeservice.model.Employee;

@Repository
public class EmployeeRepository {
    private List<Employee> employees = new ArrayList<>();
    public Employee addEmployee(Employee employee){
        employees.add(employee);
        return employee;
    }
    public Employee findById(Long id){
        return employees.stream().filter(a -> a.id().equals(id))
        .findFirst()
        .orElseThrow();
    }
    public List<Employee> findAll(){
        return employees;
    }
    public List<Employee> findByDepartment(Long departmentId){
        return employees.stream()
        .filter(a -> a.departmentId().equals(departmentId))
        .toList();
    }
}
```

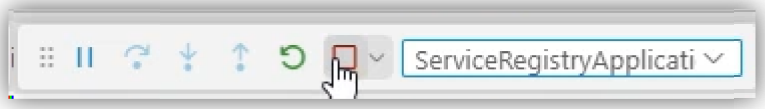
165. Ecrire le code suivant du classe (**EmployeeController.java**) :

EmployeeController.java

employee-service > src > main > java > com > pi > employeeservice > controller > EmployeeController.java > ...

```java
package com.pi.employeeservice.controller;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.pi.employeeservice.model.Employee;
import com.pi.employeeservice.repository.EmployeeRepository;

@RestController
@RequestMapping("/employee")
public class EmployeeController {
    private static final Logger LOGGER = LoggerFactory.getLogger(clazz:EmployeeCont

    @Autowired
    private EmployeeRepository repository;
    @PostMapping
    public Employee add(@RequestBody Employee employee){
        LOGGER.info(format:"Employee add:{}",employee);
        return repository.addEmployee(employee);
    }

    @GetMapping
    public List<Employee> findAll(){
        LOGGER.info(msg:"Employee find");
        return repository.findAll();
    }
    @GetMapping("/{id}")
    public Employee findById(@PathVariable Long id){
        LOGGER.info(format:"Employee find: id={}",id);
        return repository.findById(id);
    }

    @GetMapping("/department/{departmentId}")
    public List<Employee> findByDepartment(@PathVariable Long departmentId){
        LOGGER.info(format:"Employee find: departmentId={}",departmentId);
        return repository.findByDepartment(departmentId);
    }

}
```

166. Arrêter toutes les applications:



167. puis les relancer de nouveau dans l'ordre suivant :

(1) service-Regitry : depuis le fichier :

(SERVICE-REGISTRY\src\main\java\ServiceRegistryApplication.java)

(2) config-server : depuis le fichier :

(CONFIG-SERVER\src\main\java\ConfigServerApplication.java)

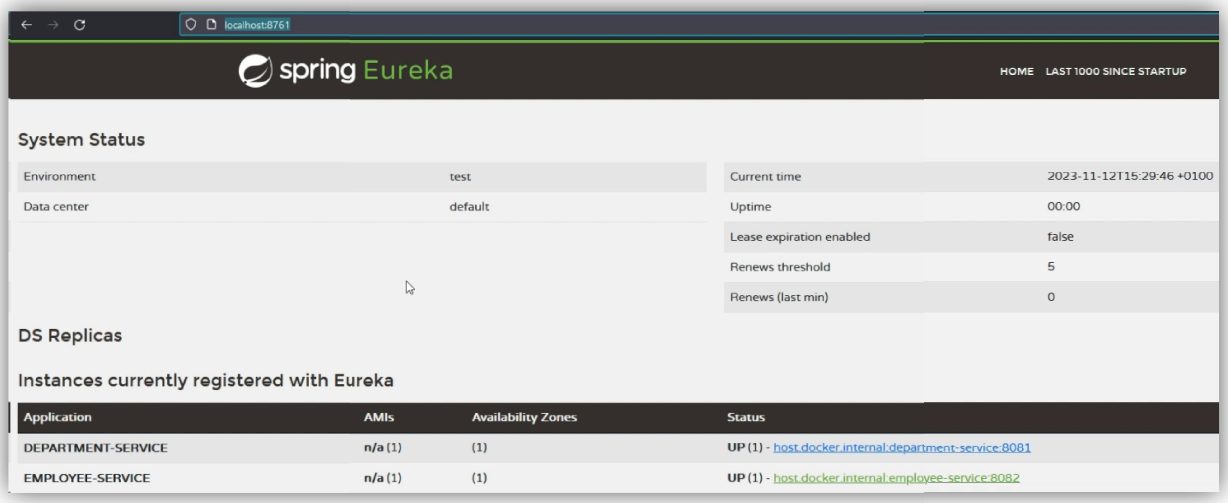
(3) department-service : depuis le fichier :

(DEPARTMENT-SERVICE\src\main\java\DepartmentServiceApplication.java)

(4) employee-service : depuis le fichier :

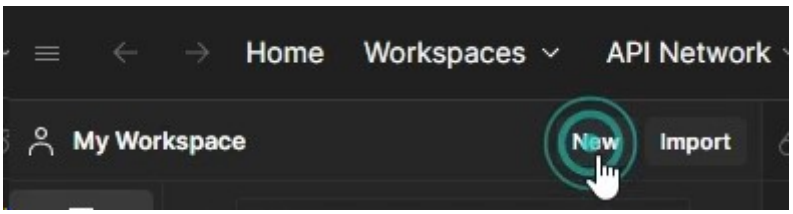
(EMPLOYEE-SERVICE\src\main\java\EmployeeServiceApplication.java)

168. Rafraîchir la page localhost:8761 et notez de la présence de (EMPLOYEE-SERVICE) dans la liste des applications reconnues :

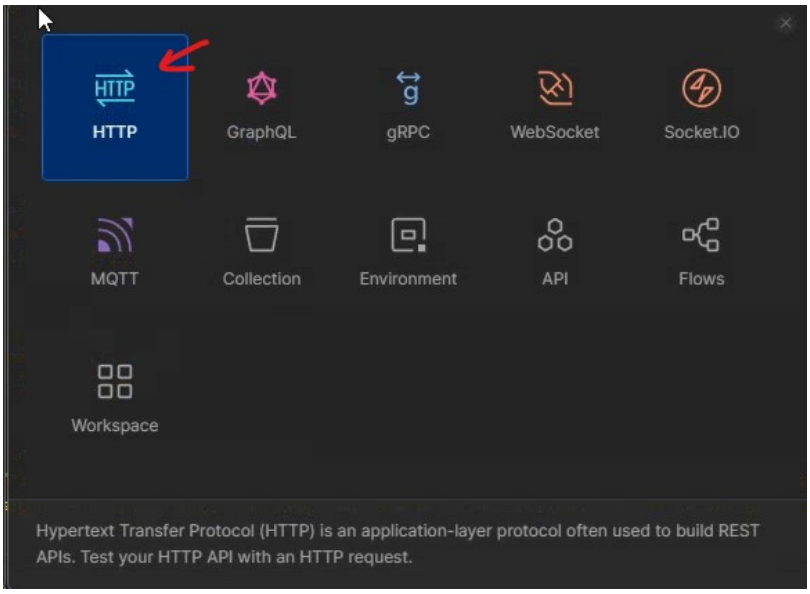


XIII. Test du Microservice « Employee »

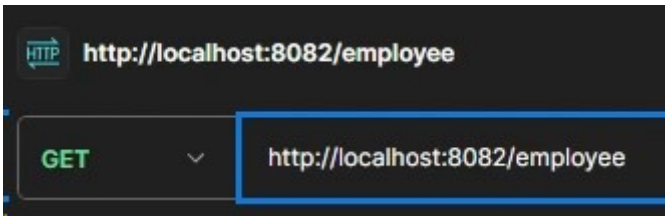
169. Créez une nouvelle connexion:



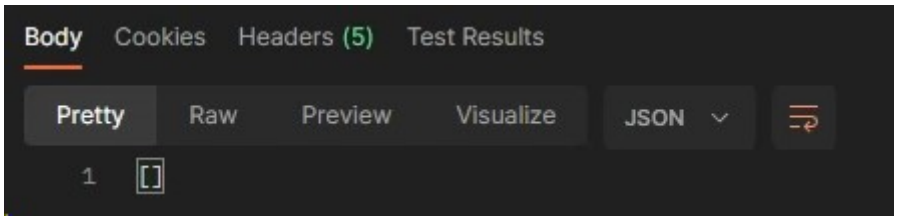
170. Sélectionnez (HTTP):



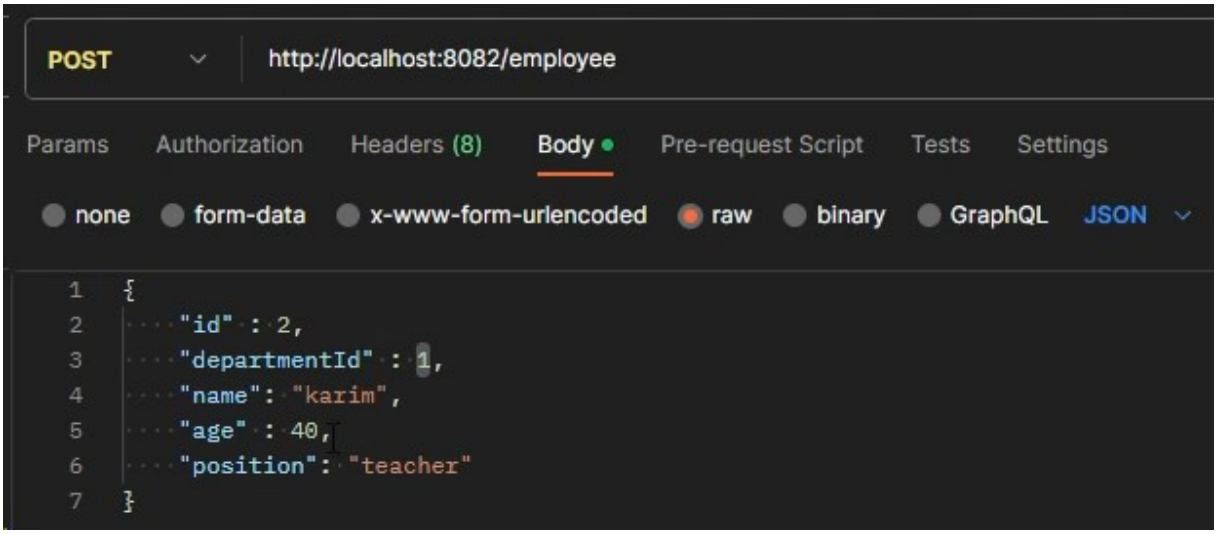
171. Lancez la requête suivante :



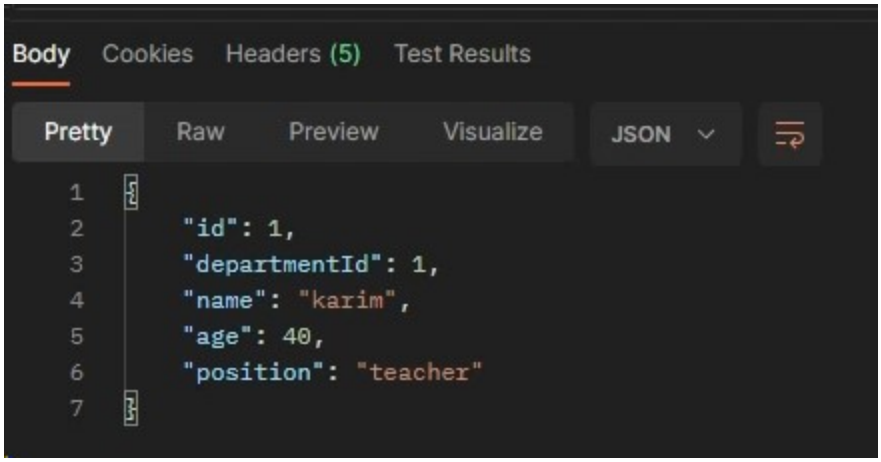
172. Voici le résultat :



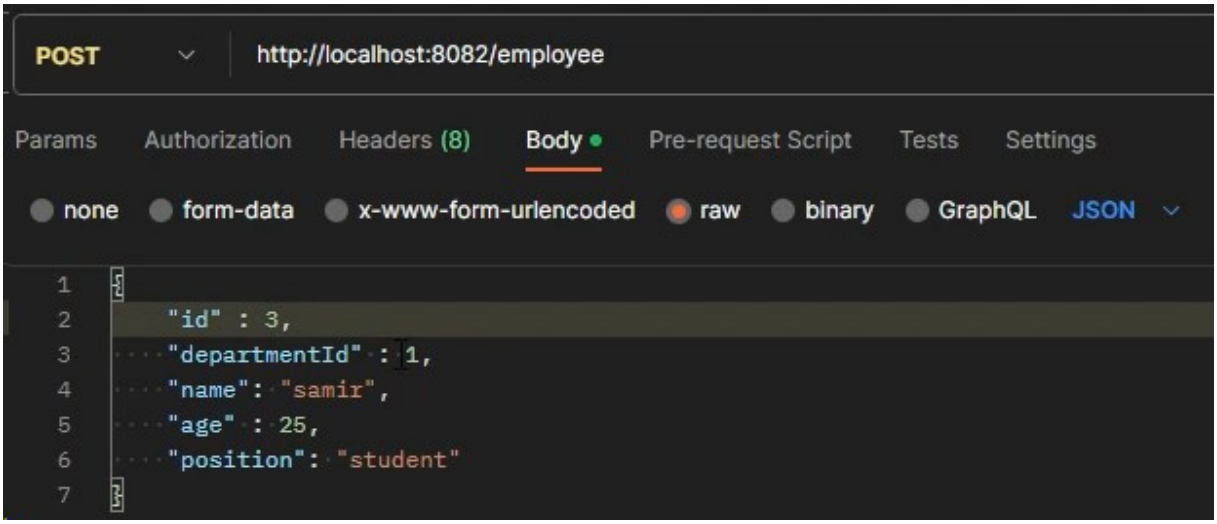
173. Ajoutez un premier employé :



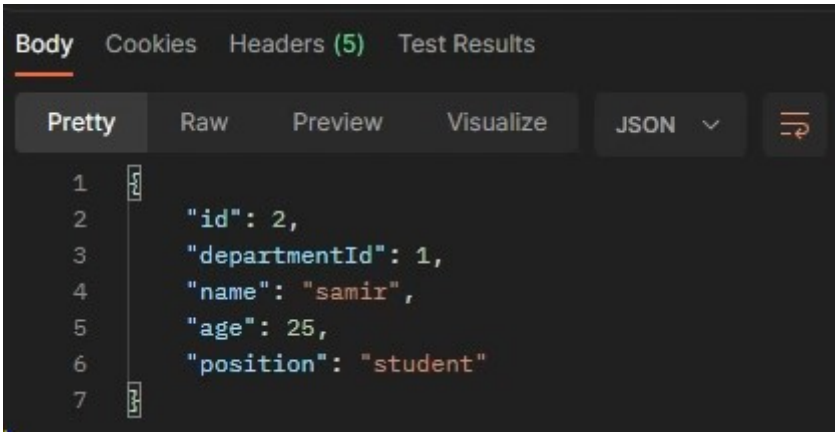
174. Voici le résultat de la dernière requête :

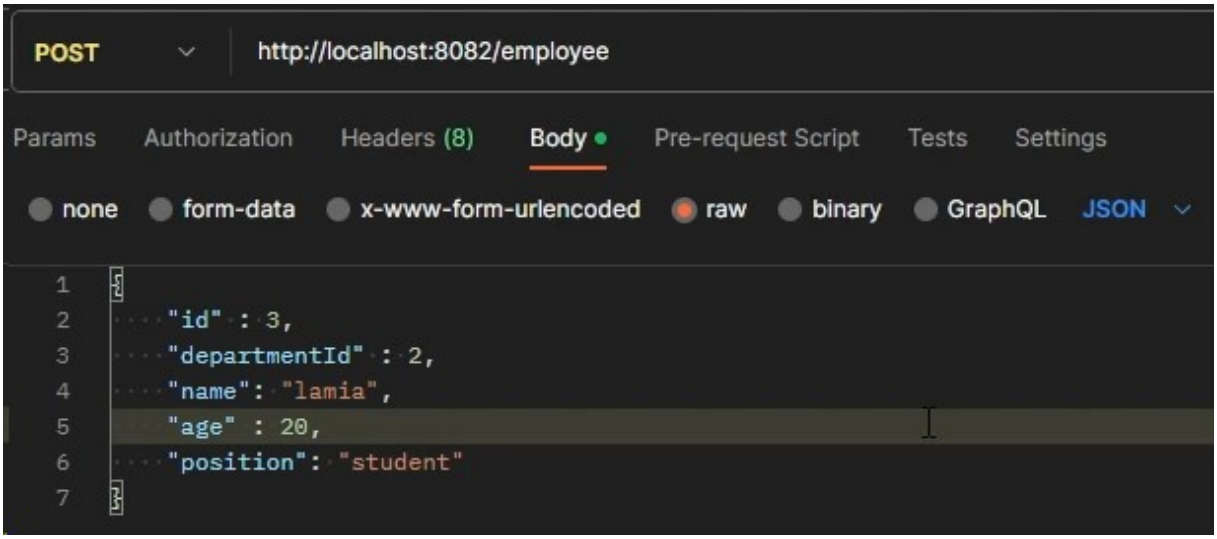


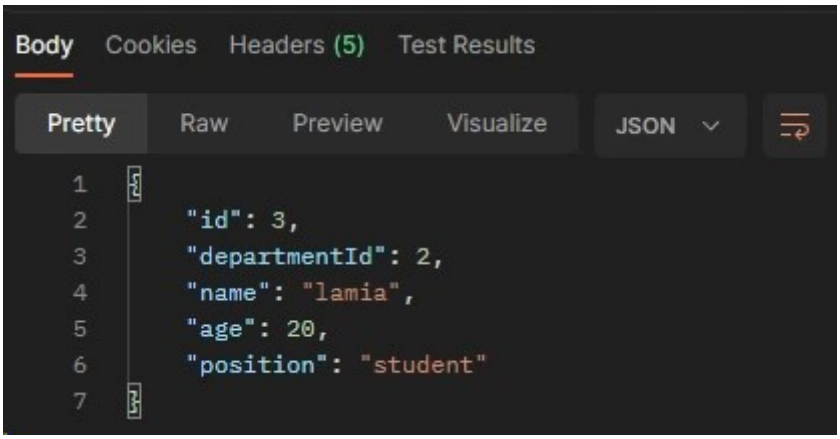
175. Ajoutez un deuxième employé :

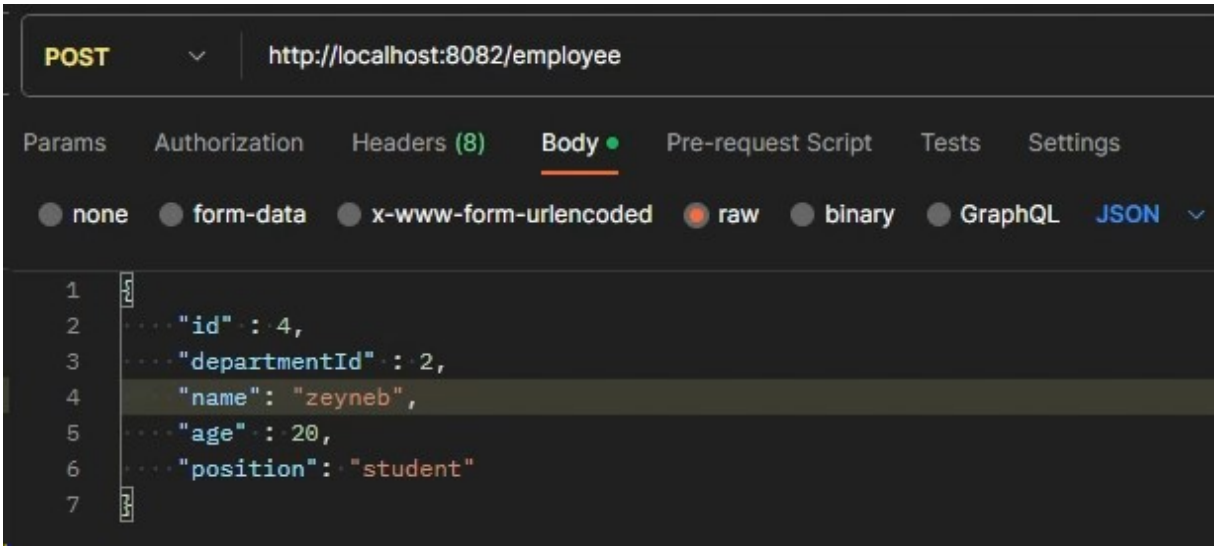


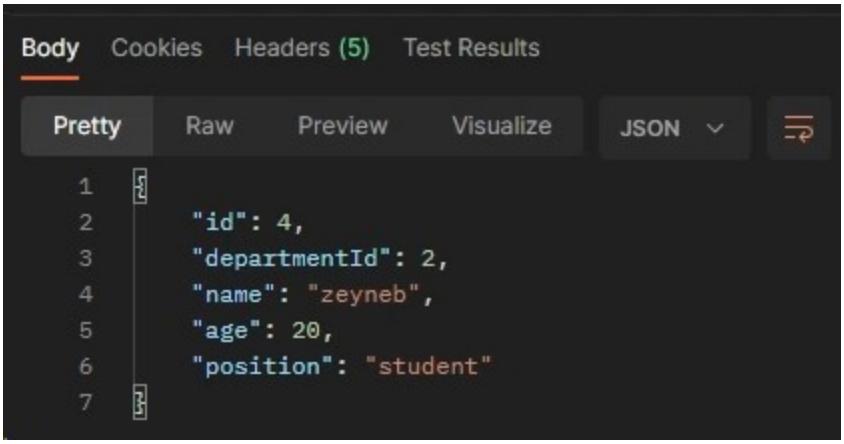
176. Voici le résultat de la dernière requête :



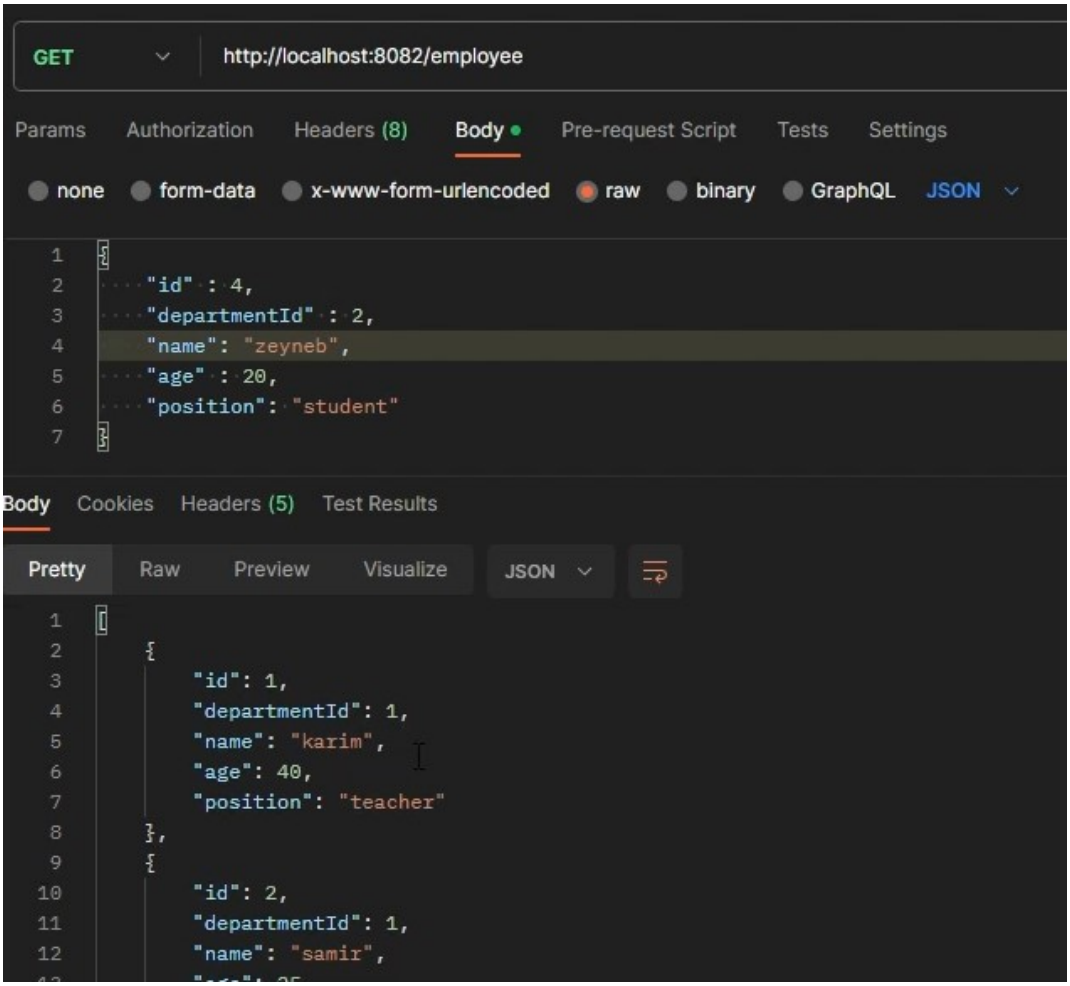




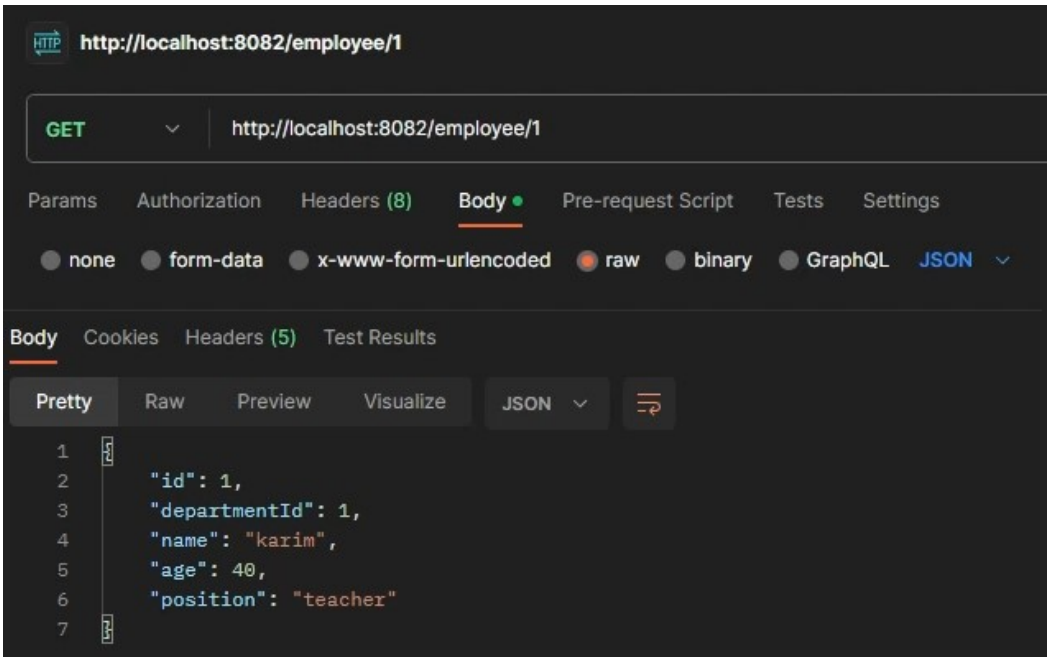

POST
http://localhost:8082/employee
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"id" : 4,
"departmentId" : 2,
"name": "zeyneb",
"age" : 20,
"position": "student"
}


Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 4,
"departmentId": 2,
"name": "zeyneb",
"age": 20,
"position": "student"
}

Lister les employés enregistrés :

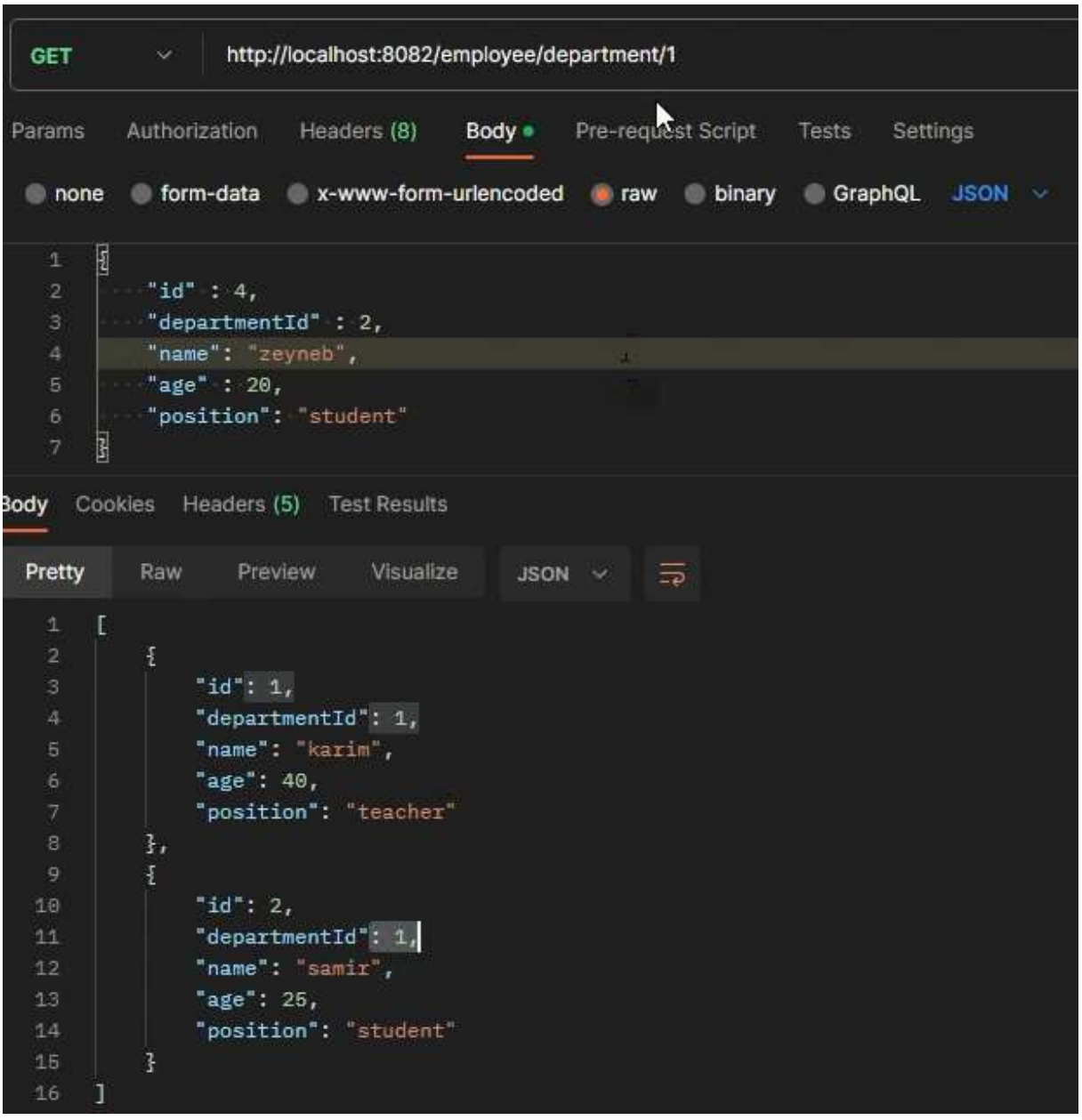


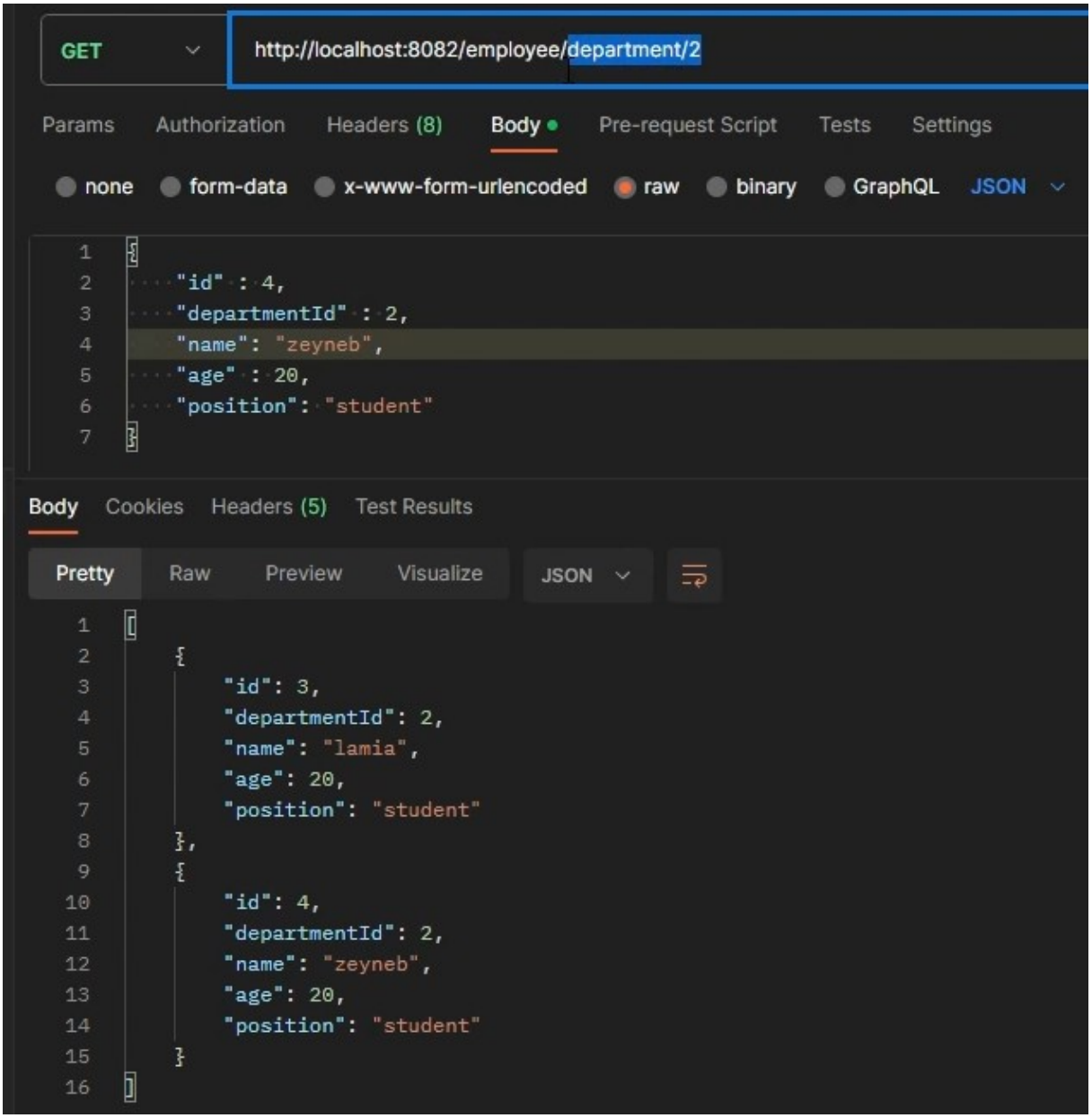
Lister l'employée de id(1) :

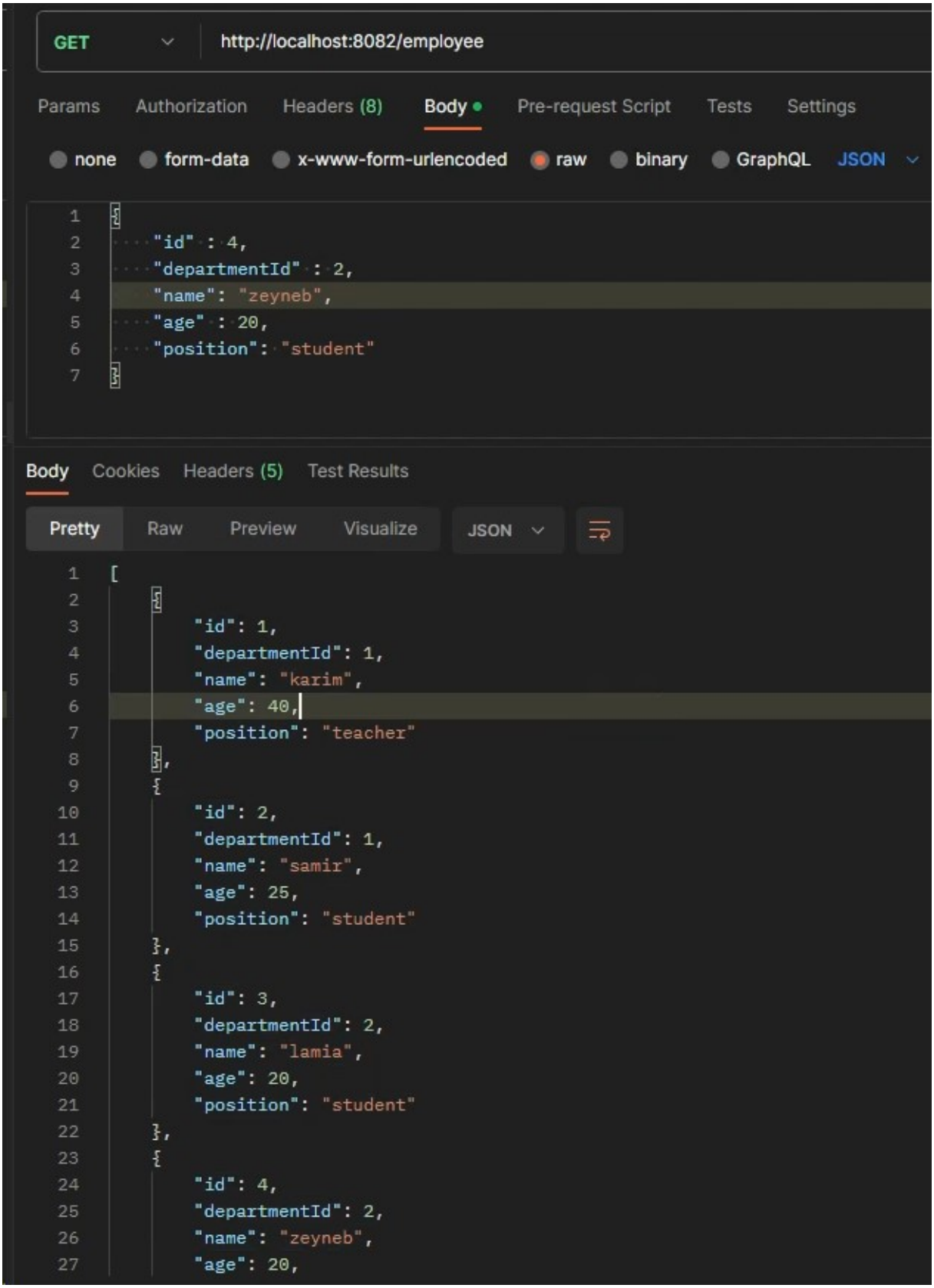


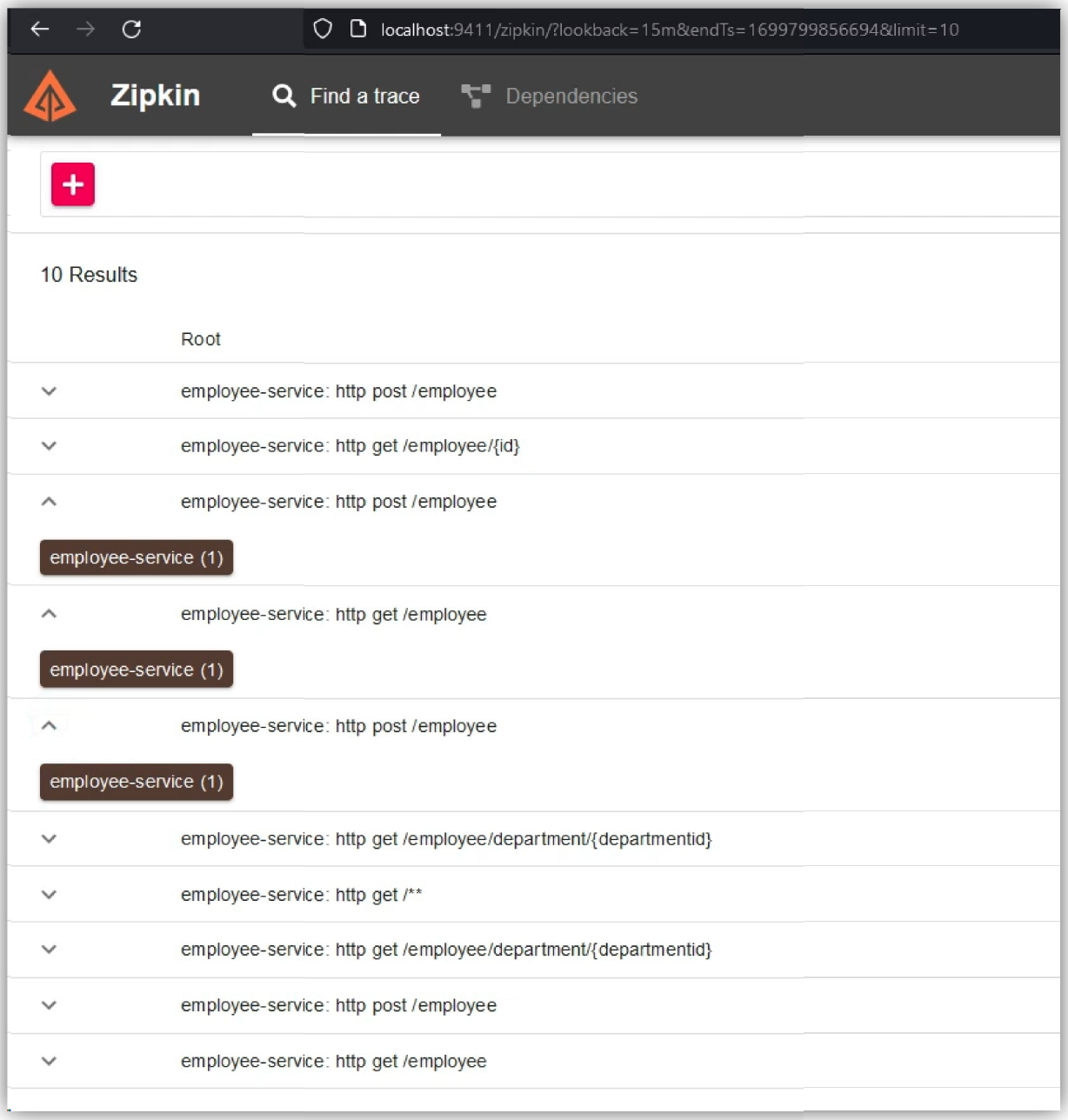
177. Appelez le test de la fonction suivante avec postman :

```java
@GetMapping("/department/{departmentId}")
public List<Employee> findByDepartment(@PathVariable Long departmentId){
    LOGGER.info(format:"Employee find: departmentId={}",departmentId);
    return repository.findByDepartment(departmentId);
}
```



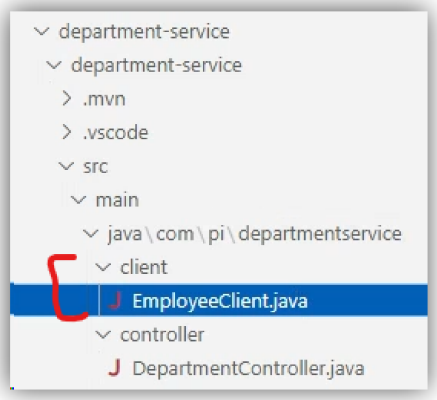

GET
http://localhost:8082/employee/department/2
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"id" : 4,
"departmentId" : 2,
"name": "zeyneb",
"age" : 20,
"position": "student"
}
Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 3,
"departmentId": 2,
"name": "lamia",
"age": 20,
"position": "student"
},
{
"id": 4,
"departmentId": 2,
"name": "zeyneb",
"age": 20,
"position": "student"
}
]


GET
http://localhost:8082/employee
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"id" : 4,
"departmentId" : 2,
"name": "zeyneb",
"age" : 20,
"position": "student"
}
Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"departmentId": 1,
"name": "karim",
"age": 40,
"position": "teacher"
},
{
"id": 2,
"departmentId": 1,
"name": "samir",
"age": 25,
"position": "student"
},
{
"id": 3,
"departmentId": 2,
"name": "lamia",
"age": 20,
"position": "student"
},
{
"id": 4,
"departmentId": 2,
"name": "zeyneb",
"age": 20,


localhost:9411/zipkin/?lookback=15m&endTs=1699799856694&limit=10
Zipkin
Find a trace
Dependencies
10 Results
Root
employee-service: http post /employee
employee-service: http get /employee/{id}
employee-service: http post /employee
employee-service (1)
employee-service: http get /employee
employee-service (1)
employee-service: http post /employee
employee-service (1)
employee-service: http get /employee/department/{departmentid}
employee-service: http get /**
employee-service: http get /employee/department/{departmentid}
employee-service: http post /employee
employee-service: http get /employee

XIV. Implémentation de la relation entre le microservice (department-service) et (employee-service)

178. Sous le dossier (C:\VS Code Projects\microservice\department-service\department-service\src\main\java\com\pi\departmentservice) créez le répertoire (**client**), puis à l'intérieur de ce dernier créez un nouveau fichier appelé (**EmployeeClient.java**) :



179. Déclarer ce fichier en tant que (**interface**), puis ajoutez l'annotation (**@HttpExchange**) à cette dernière et saisir le code suivant :

department-service > department-service > src > main > java > com > pi > departmentservice > client > EmployeeClient.java

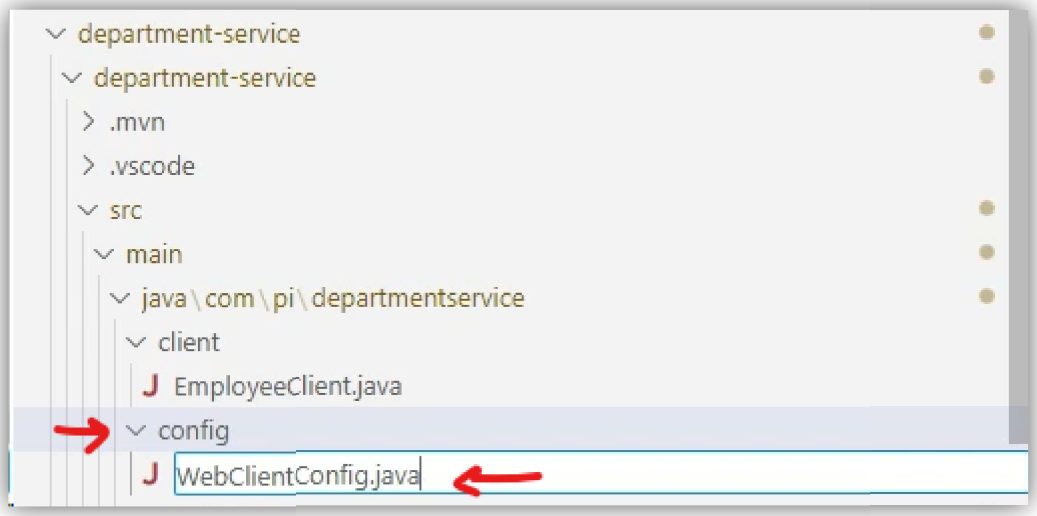
```java
package com.pi.departmentservice.client;

import java.util.List;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.service.annotation.GetExchange;
import org.springframework.web.service.annotation.HttpExchange;
import com.pi.departmentservice.model.Employee;

@HttpExchange
public interface EmployeeClient {
    @GetExchange("/employee/department/{departmentId}")
    public List<Employee> findByDepartment(@PathVariable Long departmentId);
}

```

180. Ajoutez la dépendance (**Spring Reactive Web – Web**) ayant la déclaration suivante au fichier (**pom.xml**) au niveau de la balise (**dependencies**) du projet (**department-service**) :

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

181. Relancez le microservice (department-service) :

182. Créez un nouveau répertoire appelé (config) sous (C:\VS Code Projects\microservice\department-service\department-service\src\main\java\com\pi\departmentservice), puis créez un nouveau fichier appelé (**WebClientConfig.java**) à l'intérieur de ce dernier :



183. Implémentez le code suivant :

WebClientConfig.java

department-service > department-service > src > main > java > com > pi > departmentservice > config > WebClientConfig.java > ...

```java
package com.pi.departmentservice.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.loadbalancer.reactive.LoadBalancedExchangeFilterFunction;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.reactive.function.client.WebClient;
import org.springframework.web.reactive.function.client.support.WebClientAdapter;
import org.springframework.web.service.invoker.HttpServiceProxyFactory;
import com.pi.departmentservice.client.EmployeeClient;

@Configuration
public class WebClientConfig {
    @Autowired
    private LoadBalancedExchangeFilterFunction filterFunction;

    @Bean
    public WebClient employeeWebClient(){
        return WebClient.builder()
        .baseUrl(baseUrl:"http://employee-service")
        .filter(filterFunction)
        .build();
    }
    @Bean
    public EmployeeClient employeeClient(){
        HttpServiceProxyFactory httpServiceProxyFactory
        = HttpServiceProxyFactory
        .builder(WebClientAdapter.forClient(employeeWebClient()))
        .build();
        return httpServiceProxyFactory.createClient(serviceType:EmployeeClient.class);
    }
}

```

184. Allez au fichier (**DepartmentController.java**) du projet (**department-service**) puis ajouter à la fin la fonction suivante :

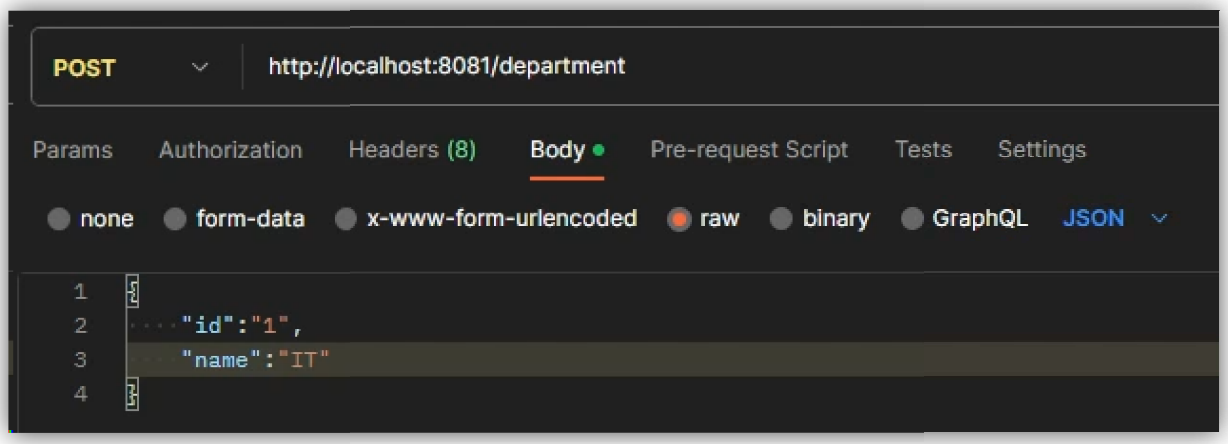
department-service > department-service > src > main > java > com > pi > departmentservice > controller > DepartmentController.java > Language Support for Java(

```java
    @GetMapping("/{id}")
    public Department findById(@PathVariable Long id){
        LOGGER.info(format:"Department find: id={}",id);
        return repository.findById(id);
    }
     @GetMapping("/with-employees")
    public List<Department> findAllWithEmployees(){
        LOGGER.info(msg:"Department find");
        List<Department> departments
        = repository.findAll();
        departments.forEach(department -> department.setEmployees(employeeClient.findByDepartment(department.getId())));
        return departments;
    }
}
```

185. Relancez le microservice (**department-service**) :

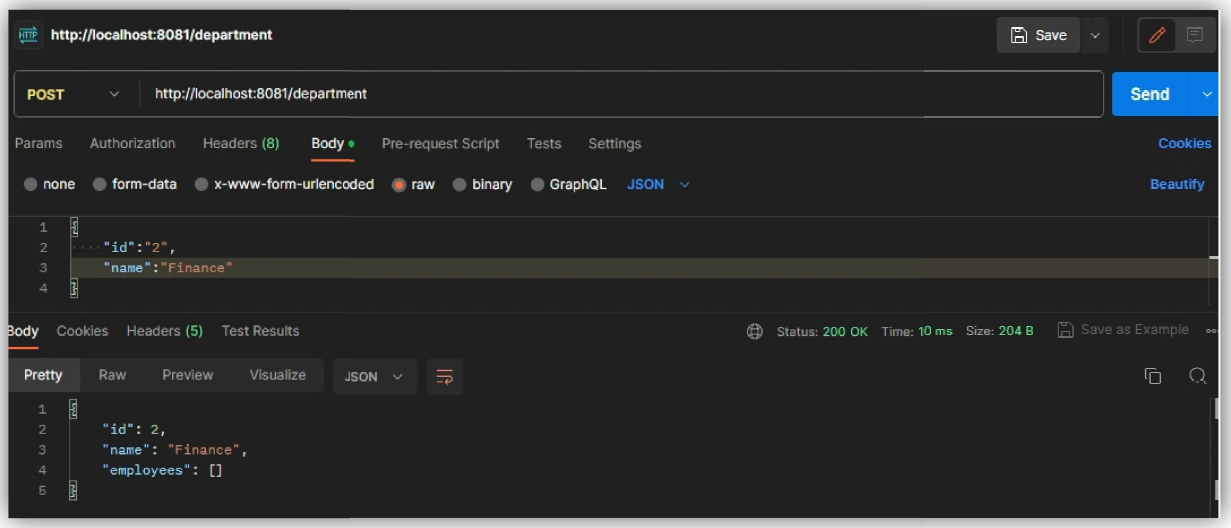
186. Lancez Postman

187. Sélectionnez la méthode (**POST**), puis cliquez sur l'onglet (**Body**), ensuite (**raw**) et sélectionnez (**JSON**) comme format d'échange :

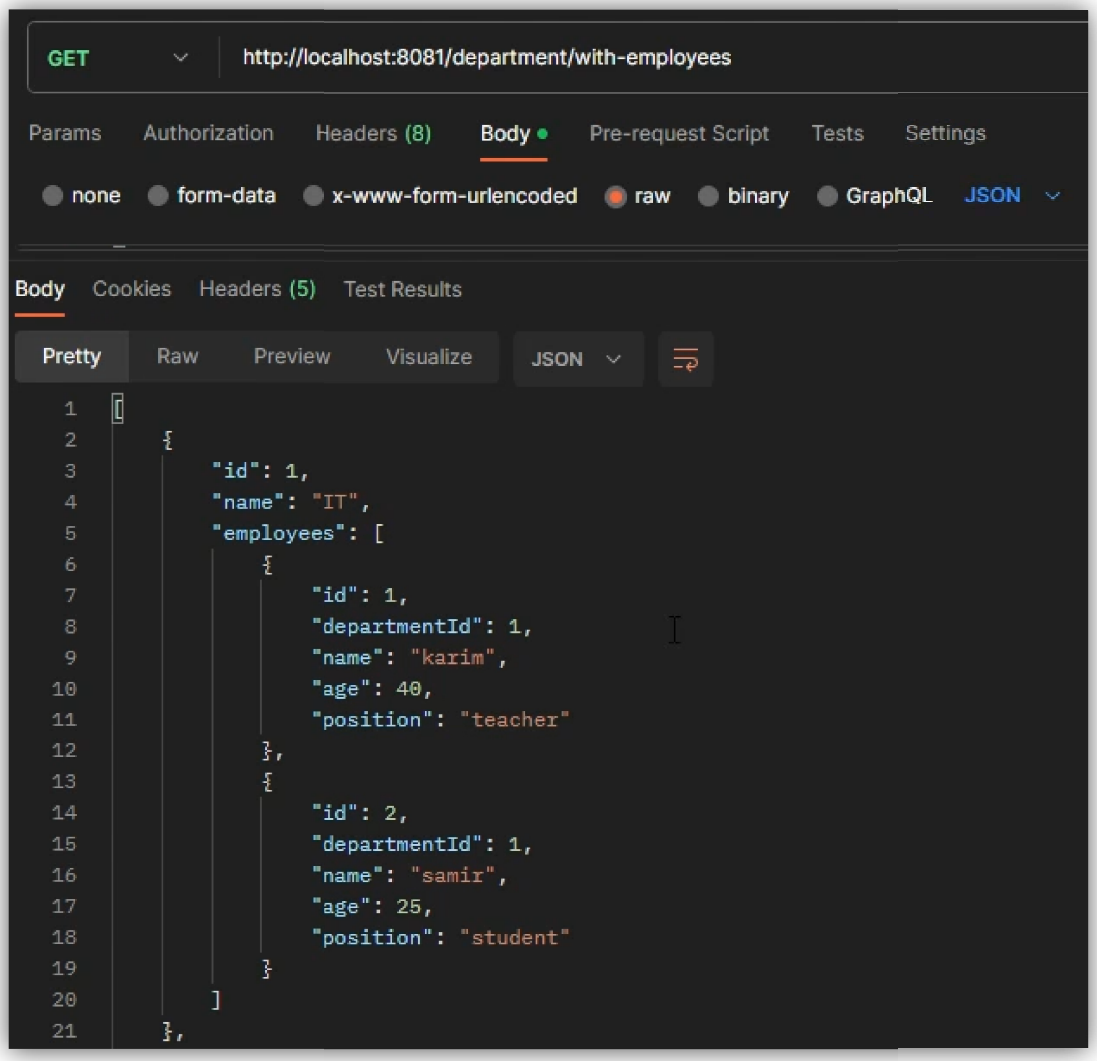


188. Cliquez sur (**SEND**)

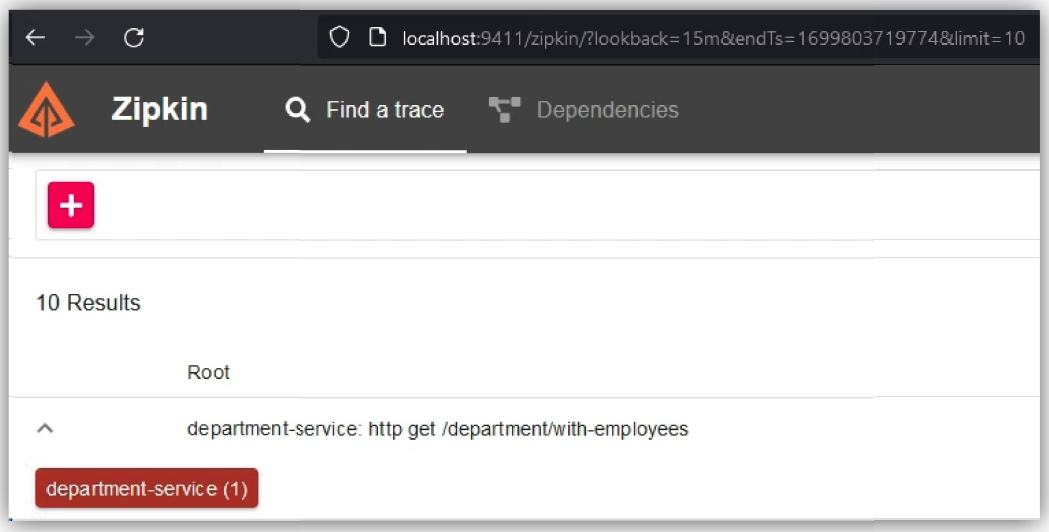
189. Ajoutez un deuxième département de id (2) et name (Finance) puis cliquez sur (SEND) :



190. Tester le nouveau chemin qui implique la relation entre department-service et employee-service :

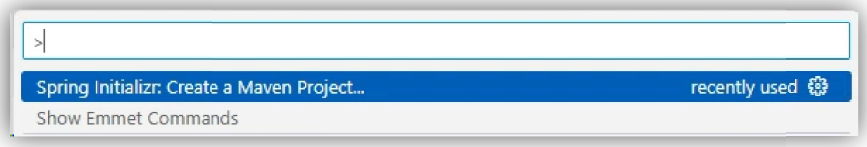


191. Voici le résultat sous (ZIPKIN) :

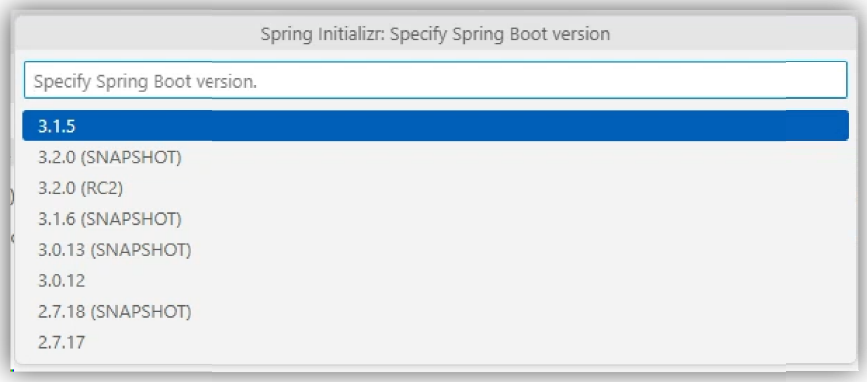


XV. Création du passerelle API

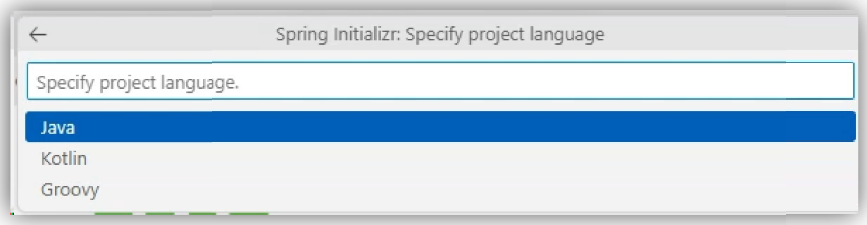
192. Appuyer **sur Ctrl + Shift + P** afin de lancer la palette des commandes.
193. Cliquez sur (**Spring Initializr : Create a Maven Project** …):



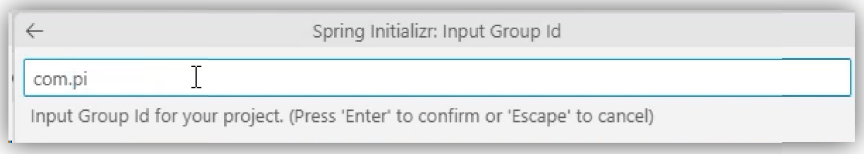
194. Sélectionnez la version (**3.1.5**)



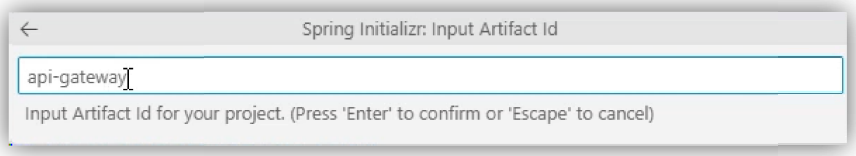
195. Sélectionnez le langage de programmation (**Java**)



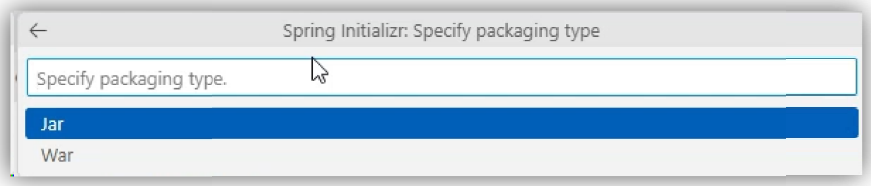
196. Choisir comme nom de package (**com.pi**)



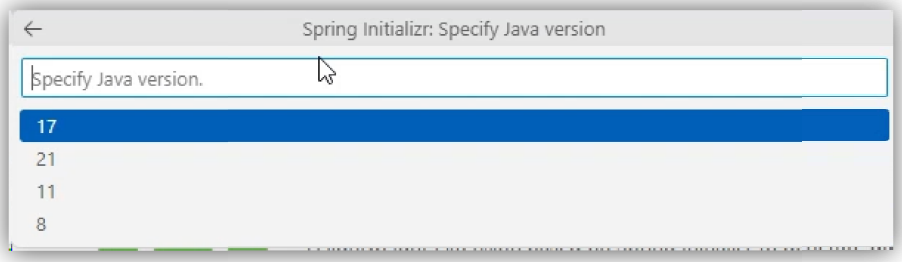
197. Indiquez le nom de projet (**api-gateway**):



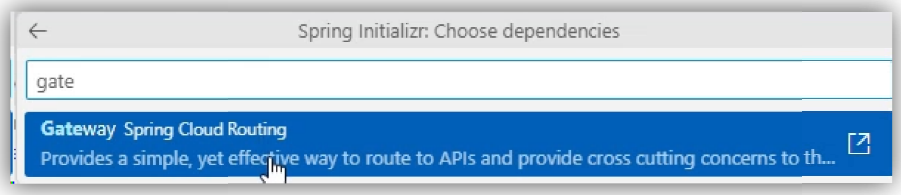
198. Choisir comme type de package (**Jar**)



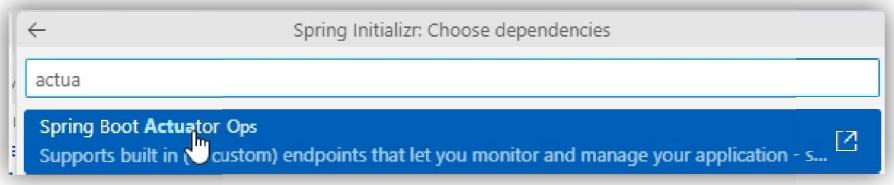
199. Choisir la version de Java Development Kit (**17**)



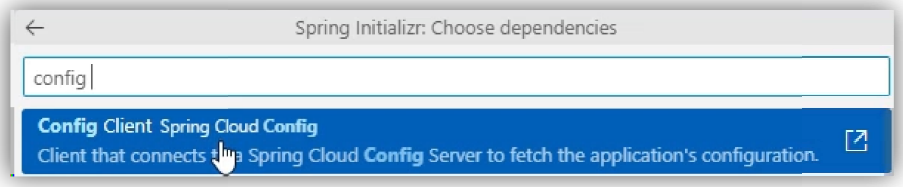
200. Ajoutez la dépendance (**Gateway – Spring Cloud Routing**)



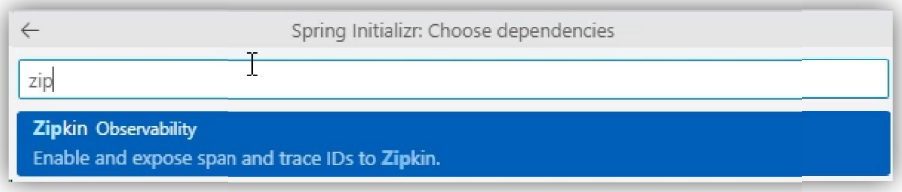
201. Ajoutez la dépendance (**Spring Boot Actuator – Ops**)



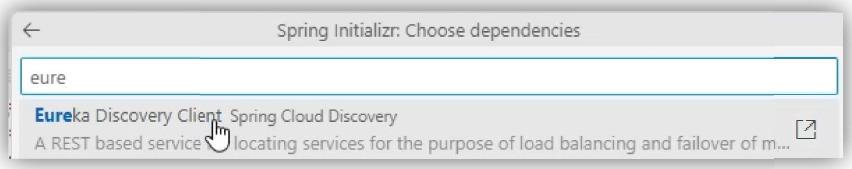
202. Ajoutez la dépendance (**Config Client – Spring Cloud Config**)



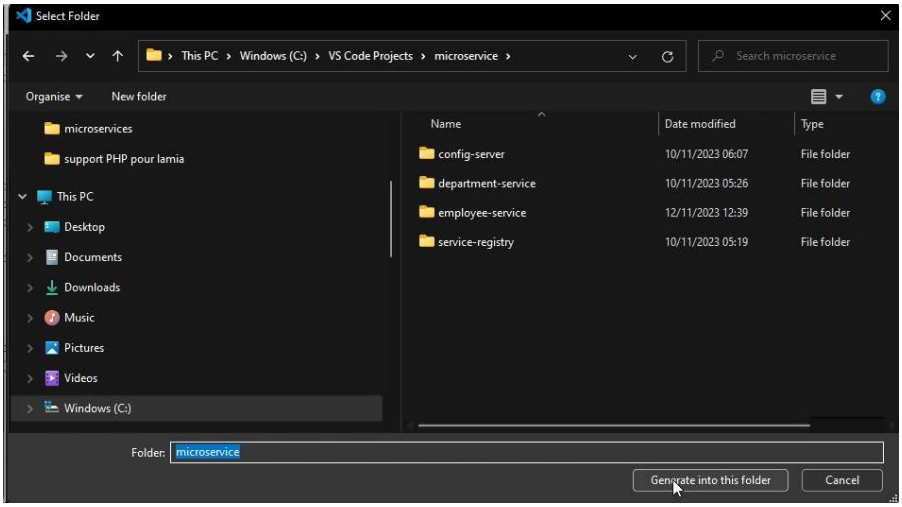
203. Ajoutez la dépendance (**Zipkin – Observability**)



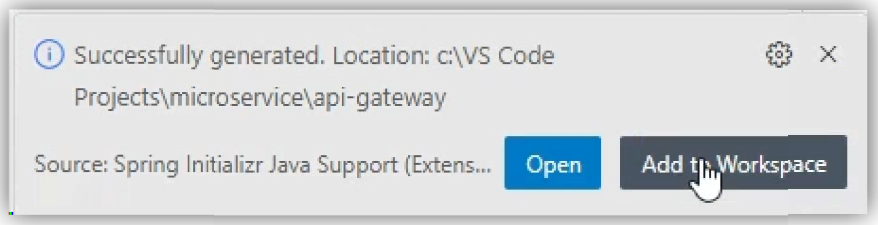
204. Ajoutez la dépendance (**Eureka Discovery Client – Spring Cloud Discovery**)



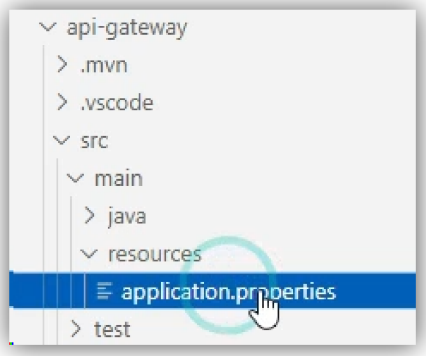
205. Créez un nouveau répertoire sous (**C:\VS Code Projects\microservice**) et appelez le (employee-service) puis cliquez sur (**Generate into this folder**)



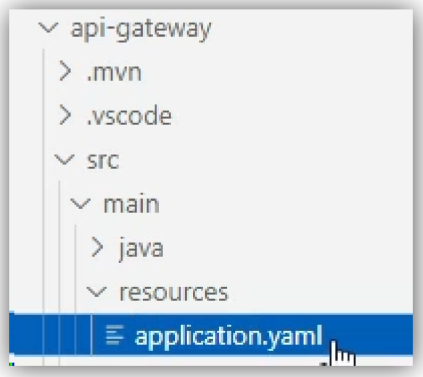
206. Ajouter ce dernier projet à l'espace de travail :



207. Changez l'extension du fichier (application.properties) pour qu'il devienne (**application.yaml**) :



208. Et voici le résultat :

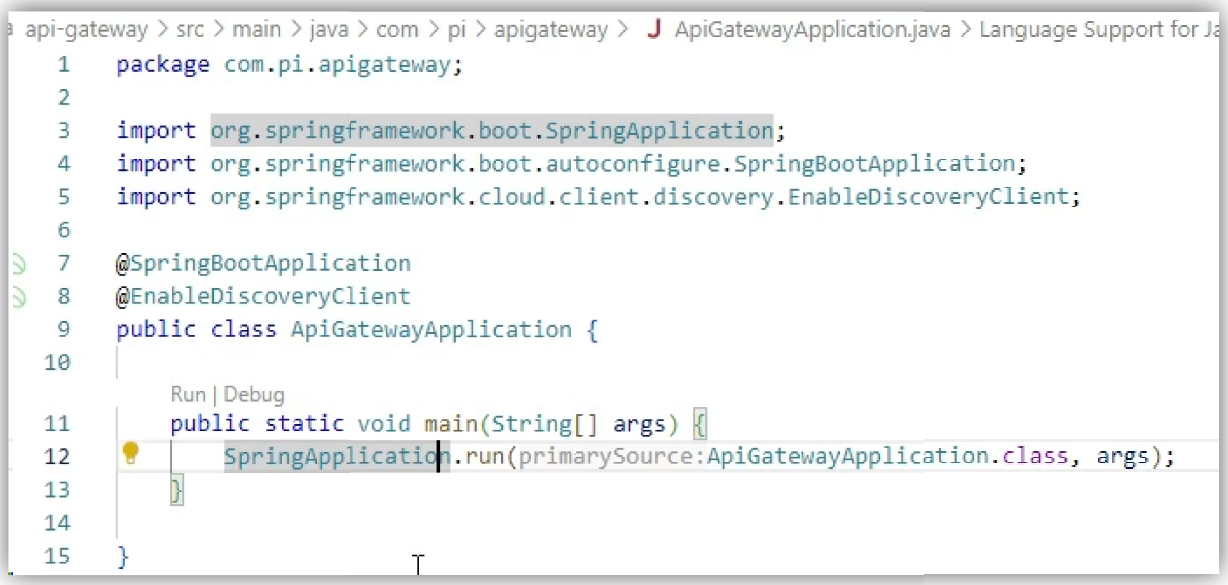


209. Changez le contenu du fichier (**application.yaml**) pour qu'il soit ainsi :

api-gateway > src > main > resources > application.yaml

```yaml
server:
    port: 8060

eureka:
    client:
        serviceUrl:
            defaultZone: http://localhost:8761/eureka/

management:
    tracing:
        sampling:
            probability: 1.0

spring:
    application:
        name: api-gateway
    config:
        import: "optional:configserver:http://localhost:8088"
    cloud:
        gateway:
            routes:
                - id: employee-service
                  uri: lb://employee-service
                  predicates:
                    - Path=/employee/**
                - id: department-service
                  uri: lb://department-service
                  predicates:
                    - Path=/department/**
```

210. Allez au fichier (C:\VS Code Projects\microservice\api-gateway\src\main\java\com\pi\apigateway\ ApiGatewayApplication.java), puis ajouter l'annotation (**@EnableDiscoveryClient**)



211. Arrêter toutes les applications:

212. puis les relancer de nouveau dans l'ordre suivant :

(1) service-Regitry : depuis le fichier :

(SERVICE-REGISTRY\src\main\java\ServiceRegistryApplication.java)

(2) config-server : depuis le fichier :

(CONFIG-SERVER\src\main\java\ConfigServerApplication.java)

(3) department-service : depuis le fichier :

(DEPARTMENT-SERVICE\src\main\java\DepartmentServiceApplication.java)

(4) employee-service : depuis le fichier :

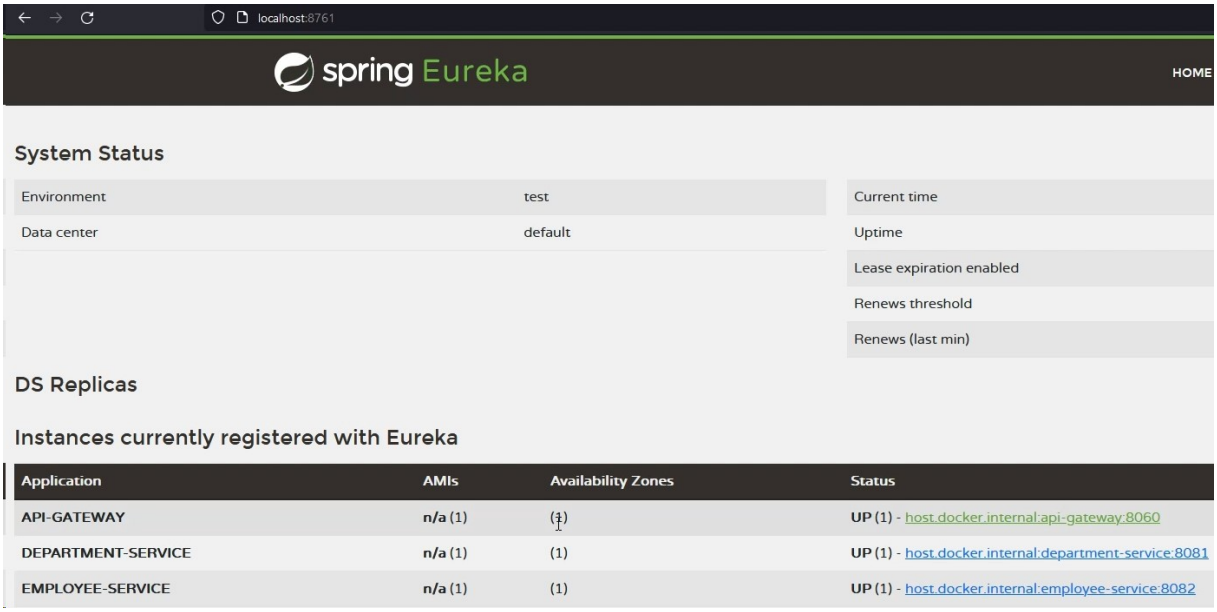
(EMPLOYEE-SERVICE\src\main\java\EmployeeServiceApplication.java)

(5) api-gateway : depuis le fichier :

(API-GATEWAY\src\main\java\ApiGatewayApplication.java)

213. Rafraîchir la page **localhost:8761** et notez de la présence de (API-GATEWAY) dans la liste des applications reconnues :

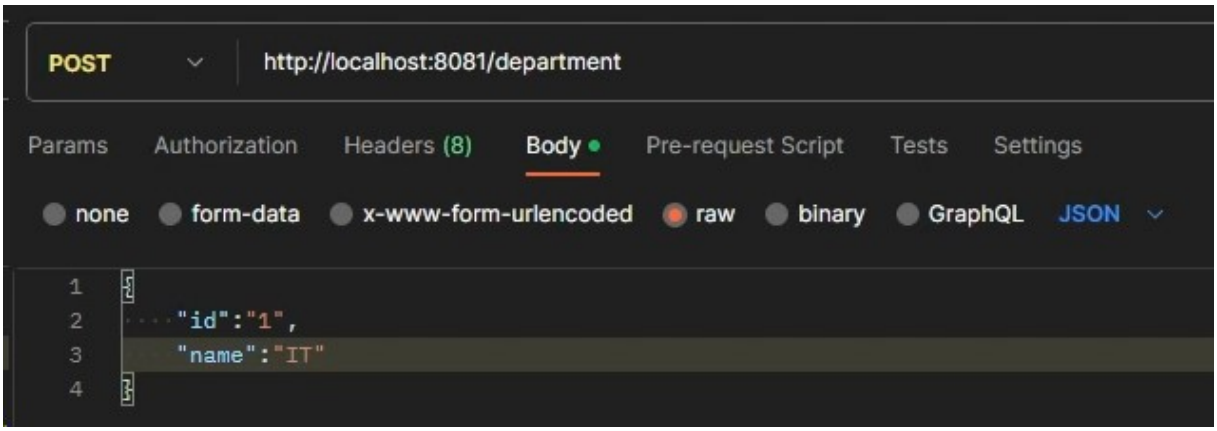
y



XVI. Test du Passerelle API

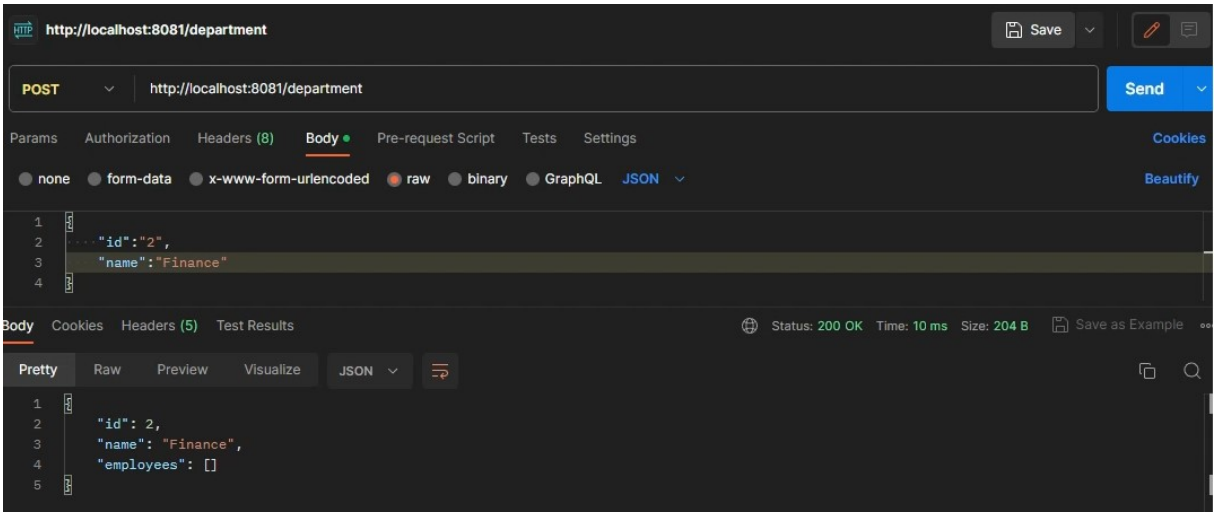
214. Lancez **Postman**

215. Sélectionnez la méthode (**POST**), puis cliquez sur l'onglet (**Body**), ensuite (**raw**) et sélectionnez (**JSON**) comme format d'échange :

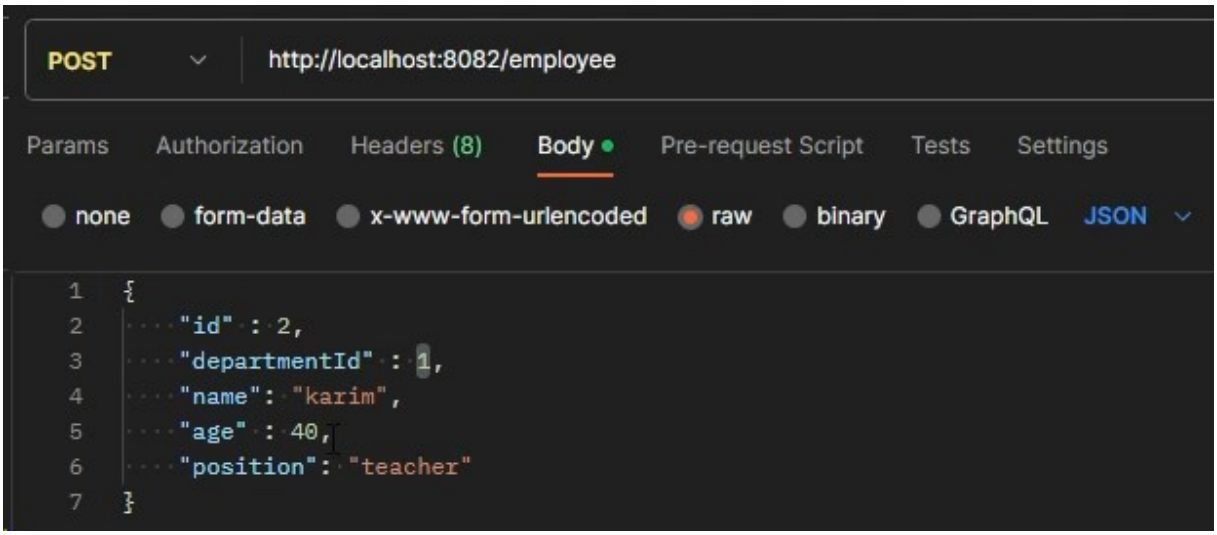


216. Cliquez sur (**SEND**)

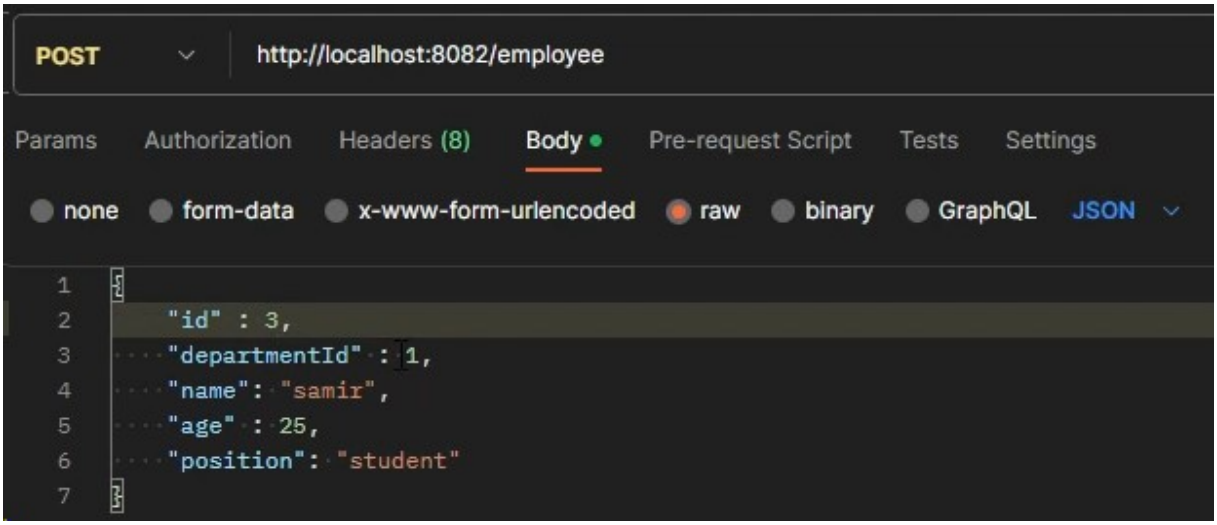
217. Ajoutez un deuxième département de id (2) et name (Finance) puis cliquez sur (SEND) :



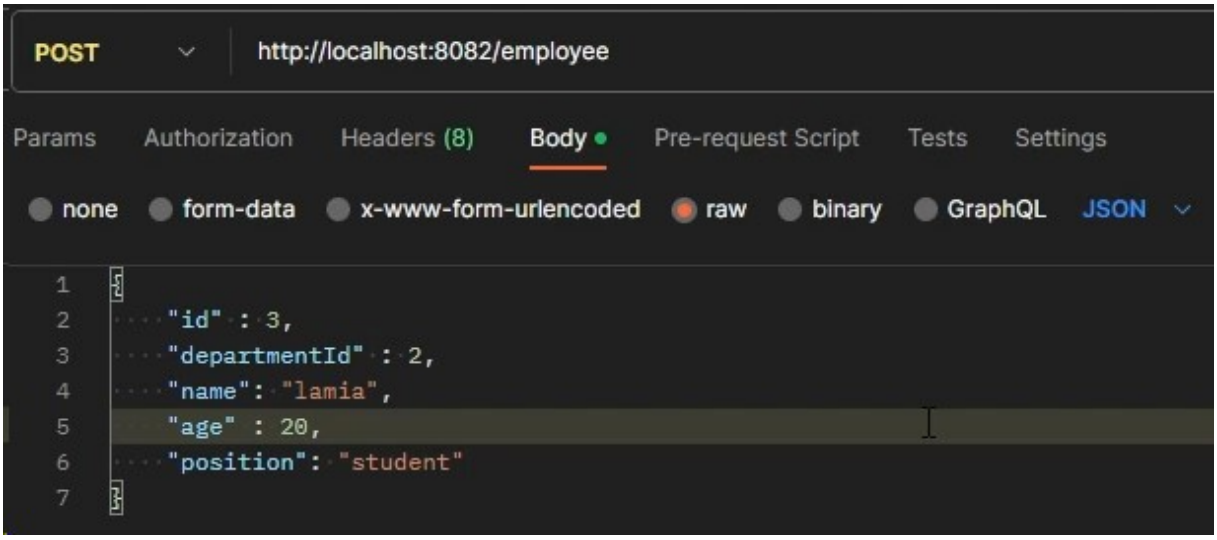
218. Ajoutez un nouvel employé :



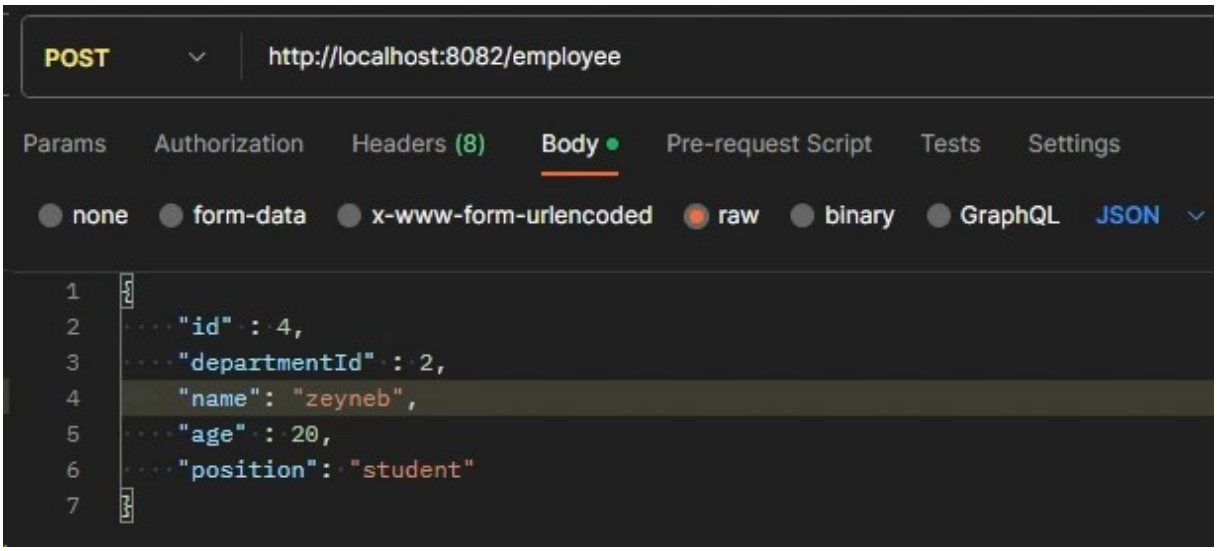
219. Ajoutez un deuxième employé :



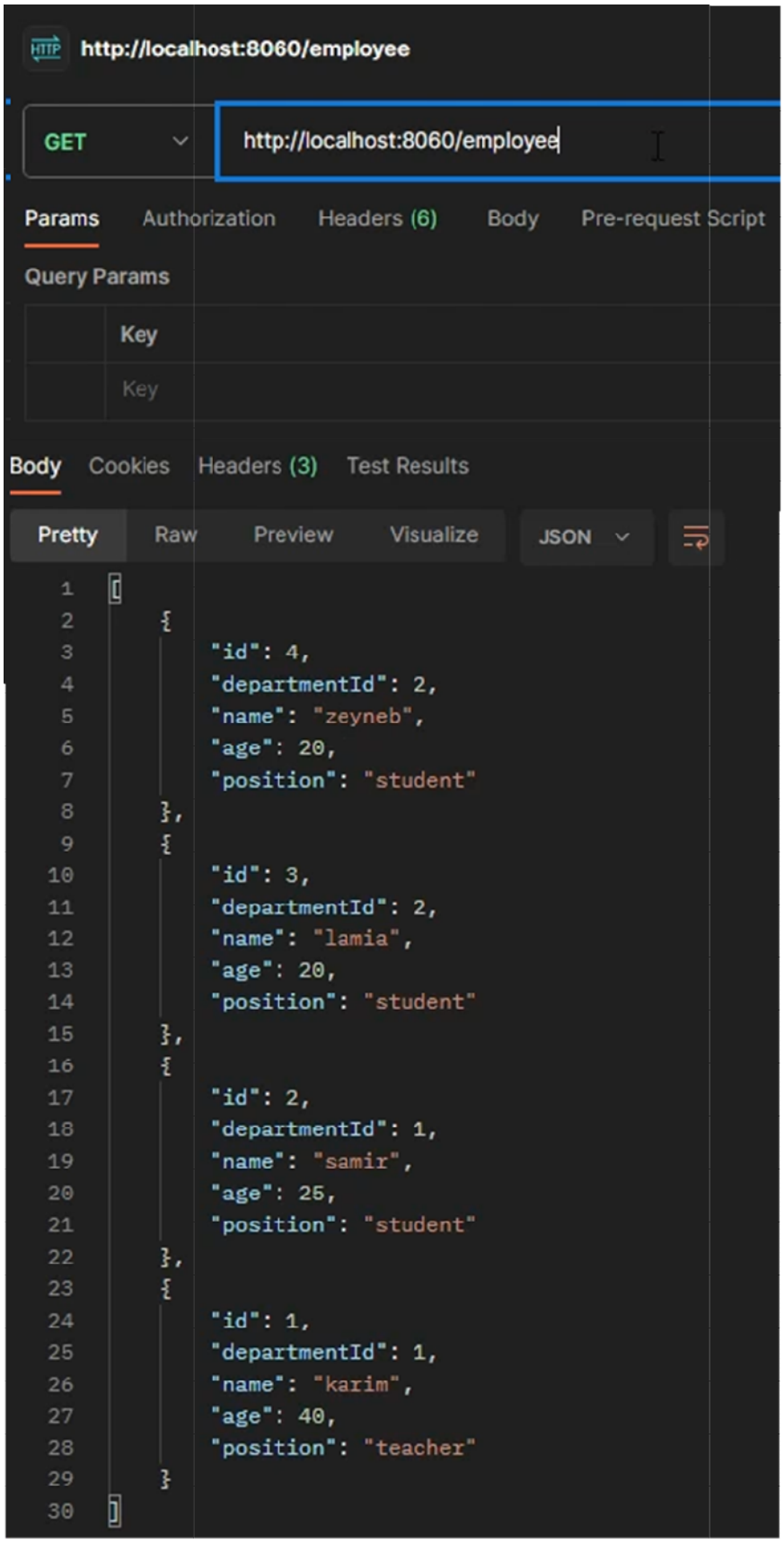
220. Ajoutez un troisième employé :



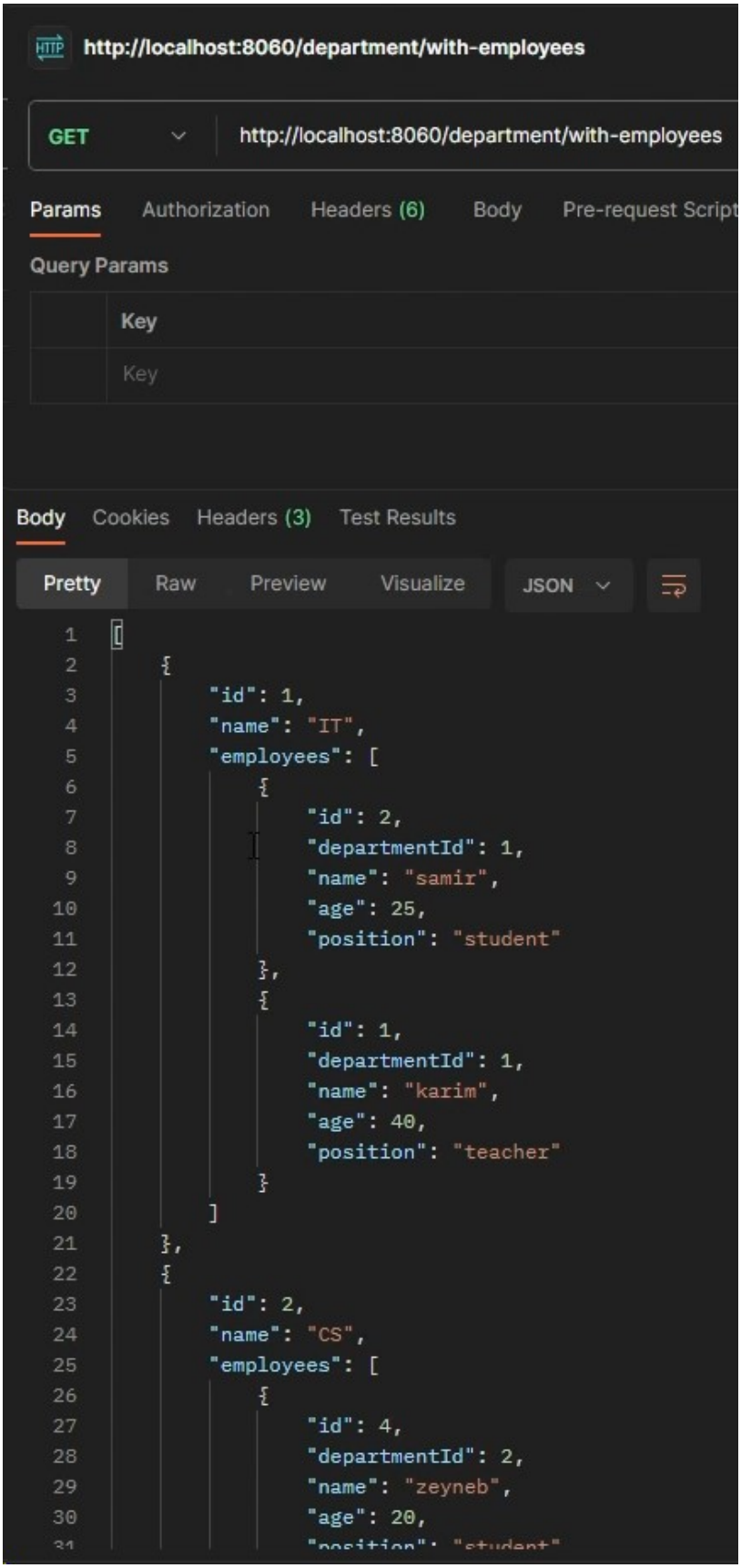
221. Ajoutez un quatrième employé :



222. Lister les employés enregistrés à partir de la passerelle (**port : 8060**) :



223. Lister les employés groupés par département à partir de la passerelle (**port : 8060**) :


http://localhost:8060/department/with-employees
GET
http://localhost:8060/department/with-employees
Params
Authorization
Headers (6)
Body
Pre-request Script
Query Params
Key
Key
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"name": "IT",
"employees": [
{
"id": 2,
"departmentId": 1,
"name": "samir",
"age": 25,
"position": "student"
},
{
"id": 1,
"departmentId": 1,
"name": "karim",
"age": 40,
"position": "teacher"
}
]
},
{
"id": 2,
"name": "CS",
"employees": [
{
"id": 4,
"departmentId": 2,
"name": "zeyneb",
"age": 20,