

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

I. Objectifs du TP :

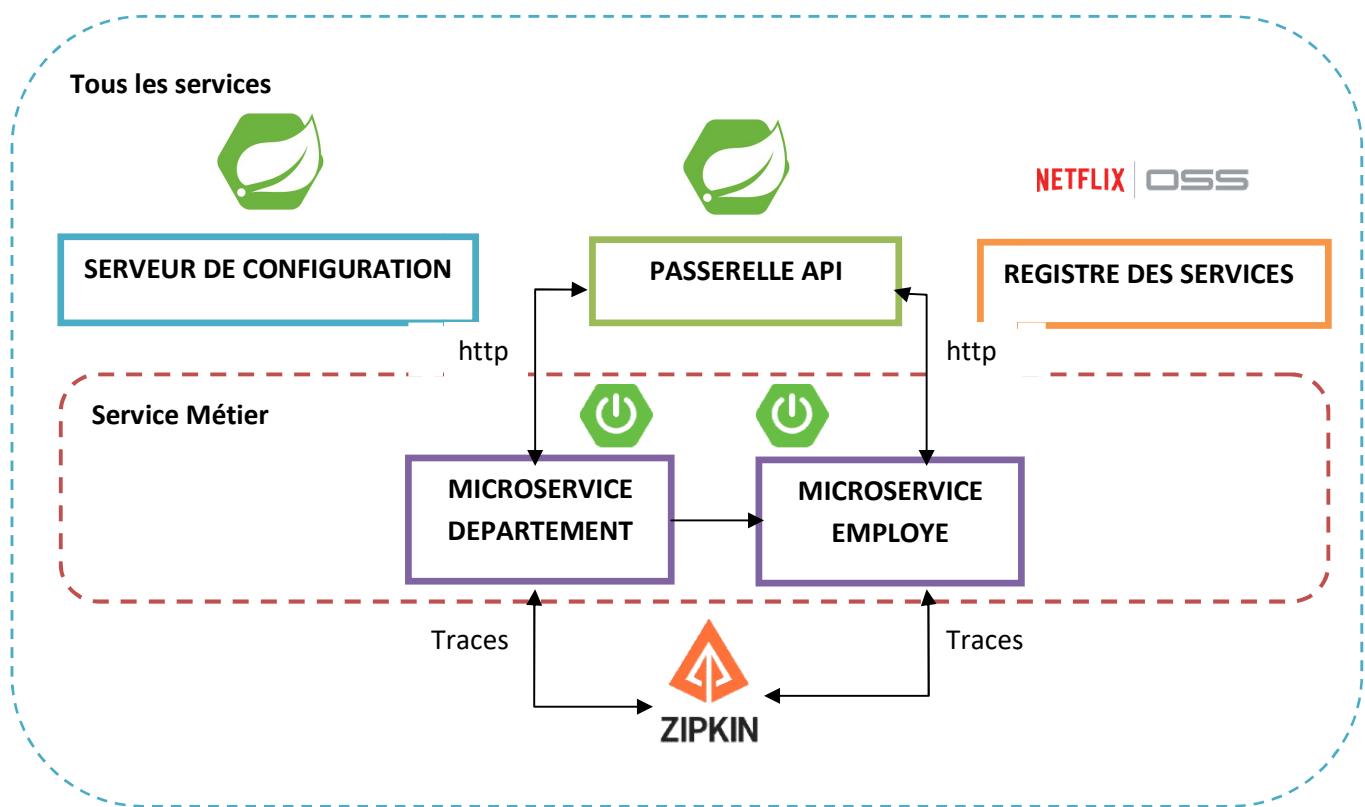
- Mettre en place une architecture microservices avec SpringBoot 3.
- Tester les microservices implémentés à l'aide d'un outil de test.

II. Pré-requis :

- Installation de Postman.

III. Description du projet :

Notre projet aura l'architecture suivante :

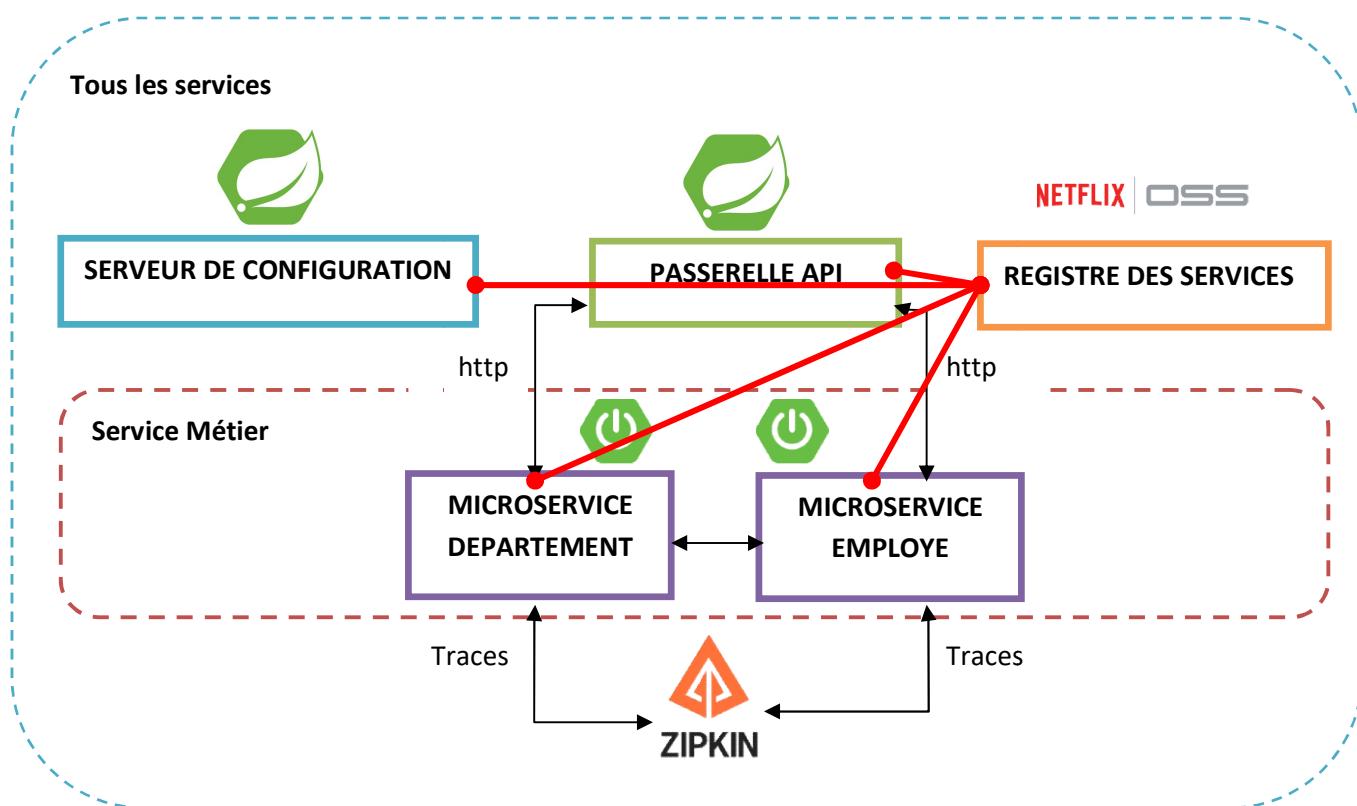


- Chaque microservice aura son propre objectif et service :
 - Le microservice département va gérer uniquement les départements :
 - Récupérer les détails d'un département
 - Ajouter un nouveau département
 - Mettre à jour les infos d'un département
 - ...

- 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.

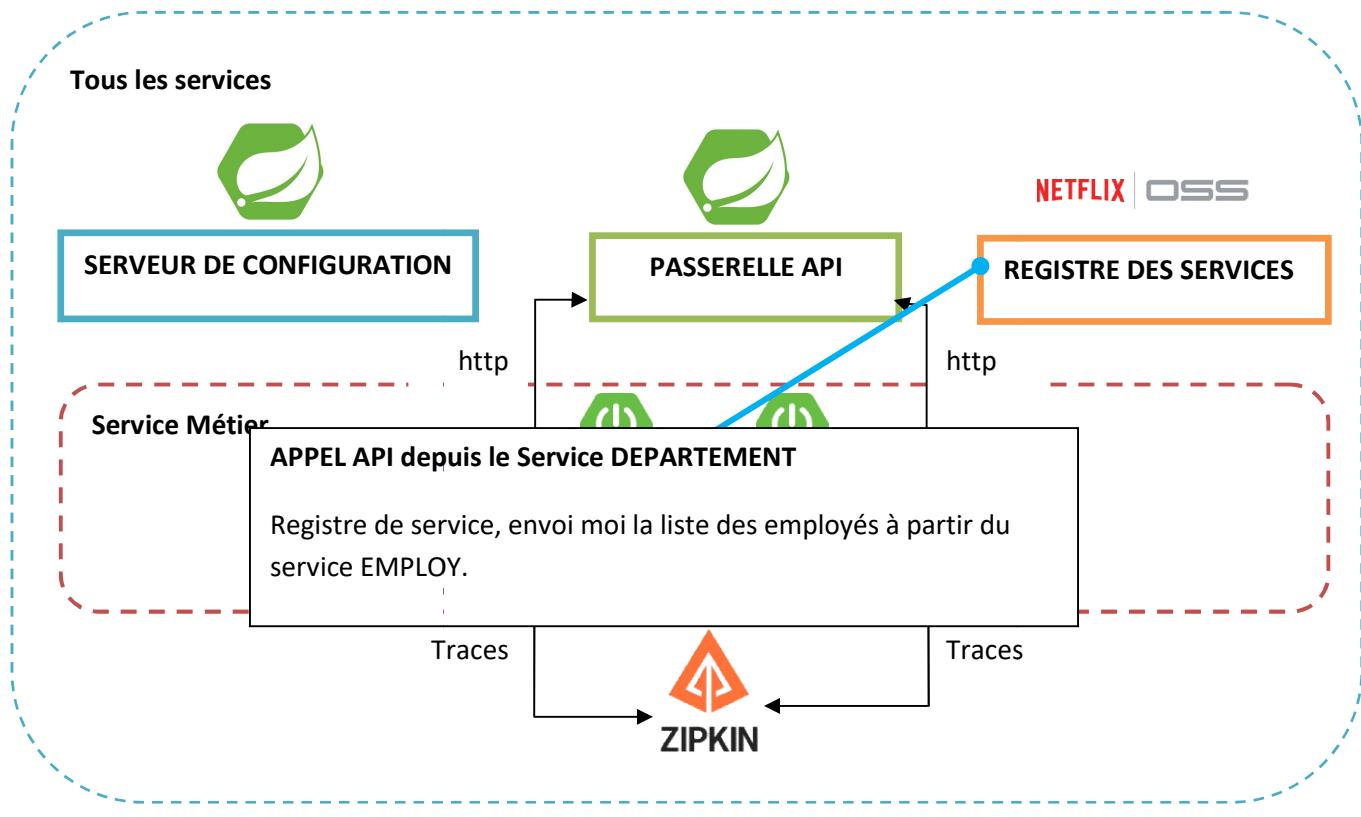


Note importante : avec les microservices on peut mettre à l'échelle (scale-up ou scale-down) un microservice bien déterminé sans affecter les autres.

Donc si j'aurai quatre instances du microservice Employé 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 Employé, 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 EMPLOYEE, 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.

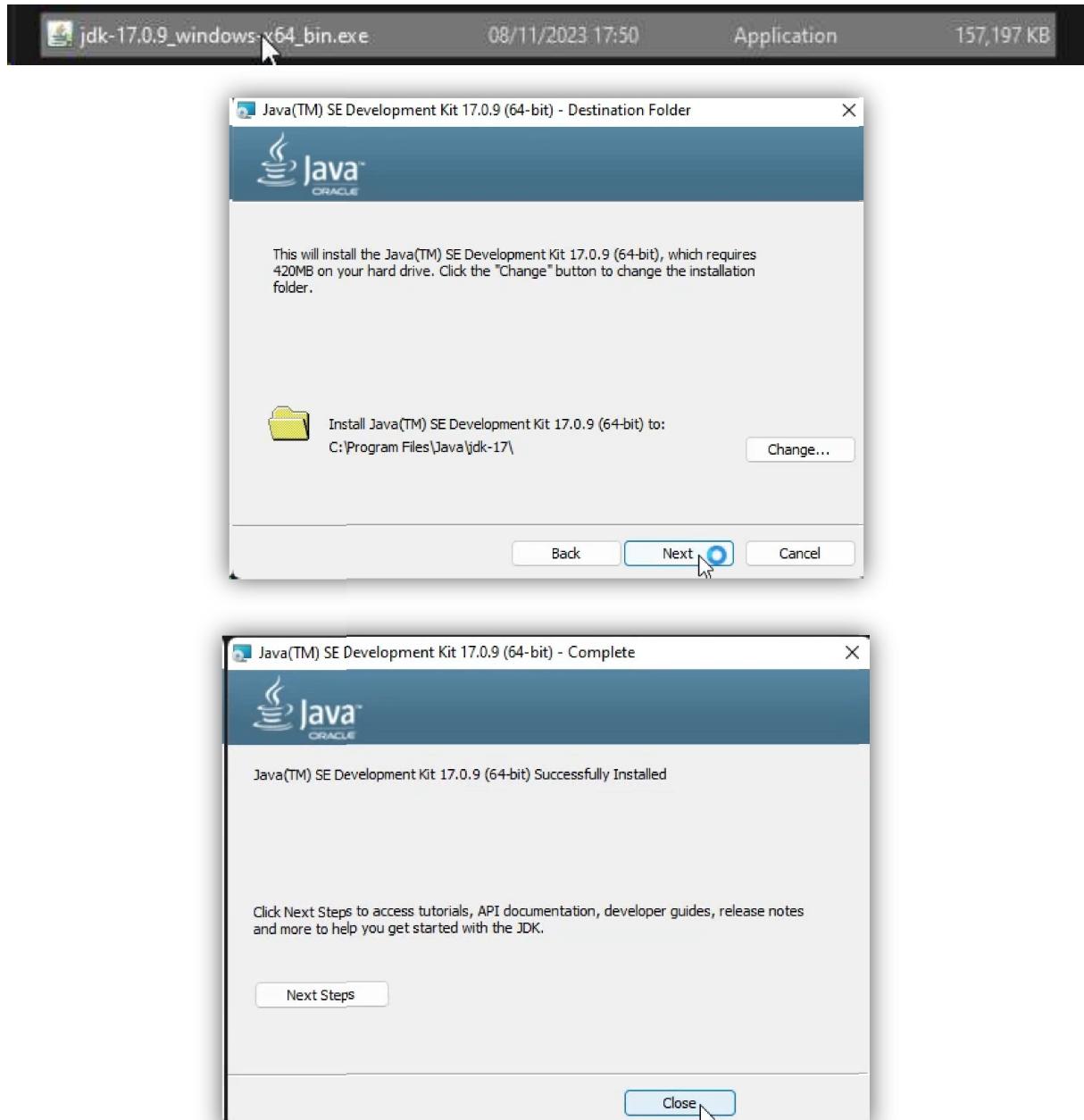
e) Étapes d'implémentation d'une architecture Microservice :

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 ;

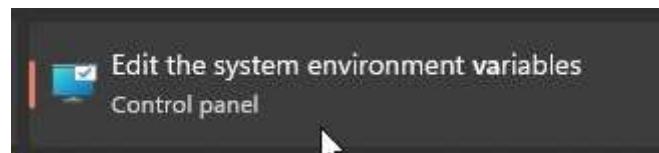
10. Connecter les deux microservices au registre des services ;
11. Test de la communication entre le microservice DEPARTEMENT et le microservice EMPLOYE en utilisant Postman ;
12. Création de la passerelle API ;
13. Test du passerelle API en utilisant Postman.

IV. Crédit du projet :

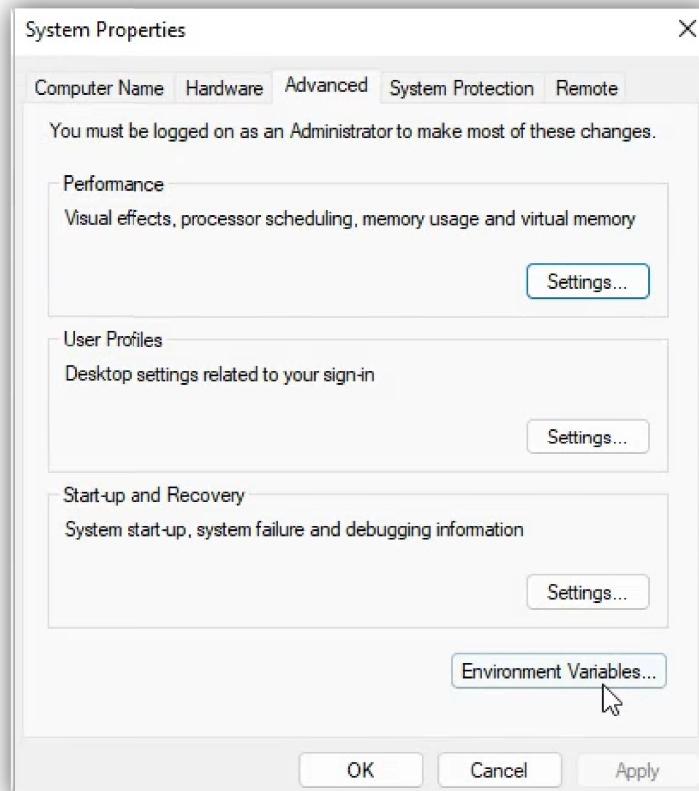
1. Installez jdk-17 :



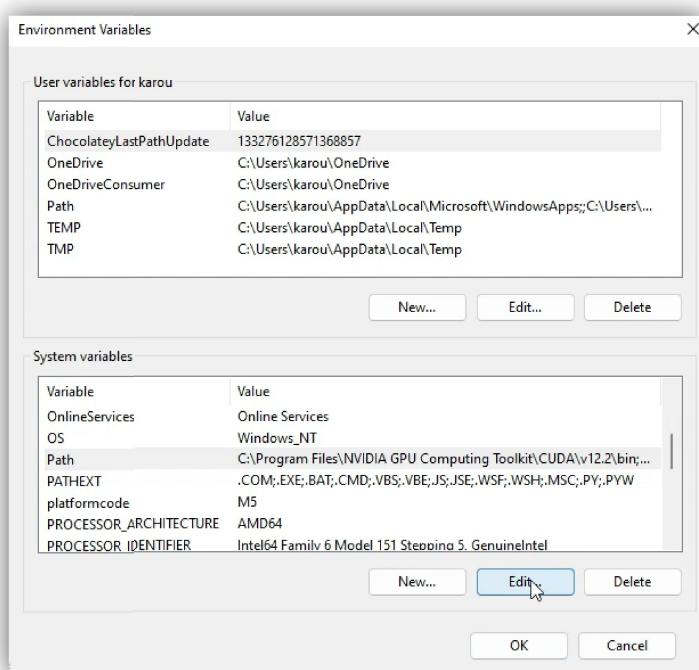
2. Ouvrir la fenêtre dédiée à l'édition des variables d'environnement :



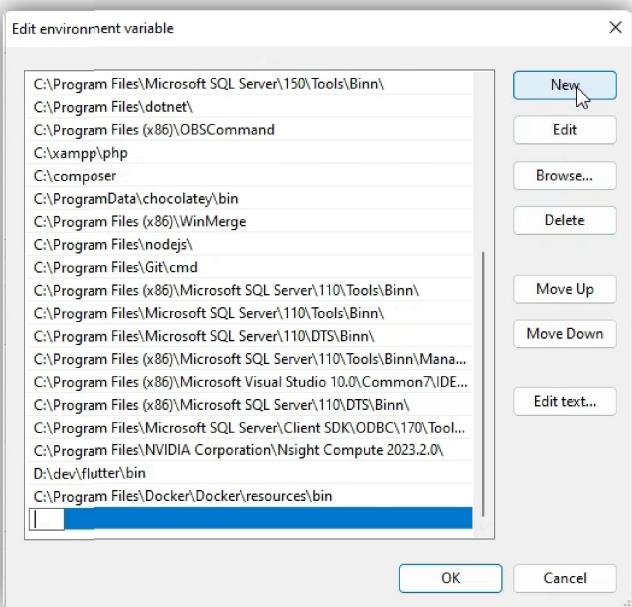
3. Cliquez sur (Environment Variables...) :



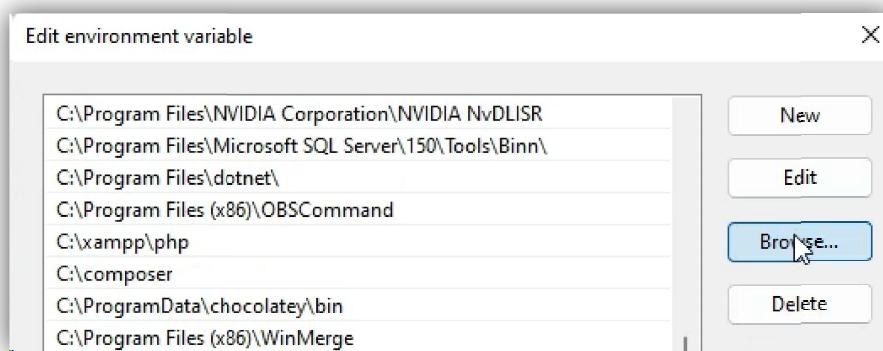
4. Cherchez la variable système (Path)



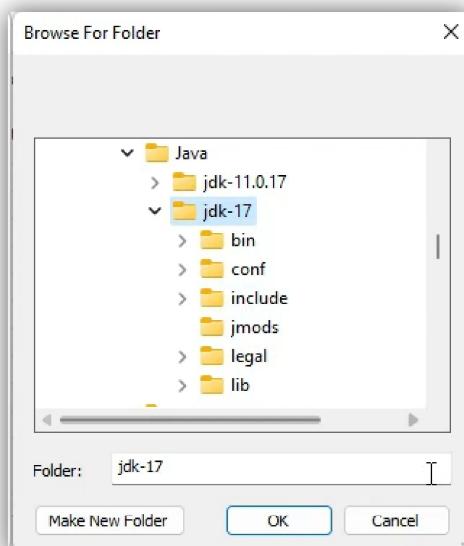
5. Créez une nouvelle variable :



6. Cliquez sur le bouton (Browse)



7. Sélectionnez le dossier (jdk-17) :

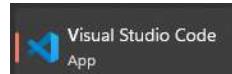


Et voici le résultat :



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

9. Lancez Microsoft Visual Studio Code :

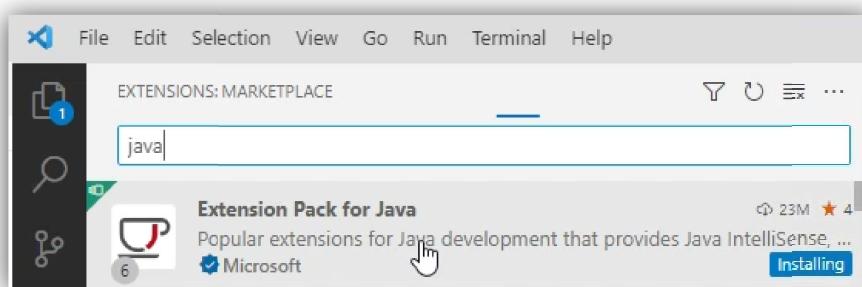


10. Ouvrez l'onglet d'extensions :

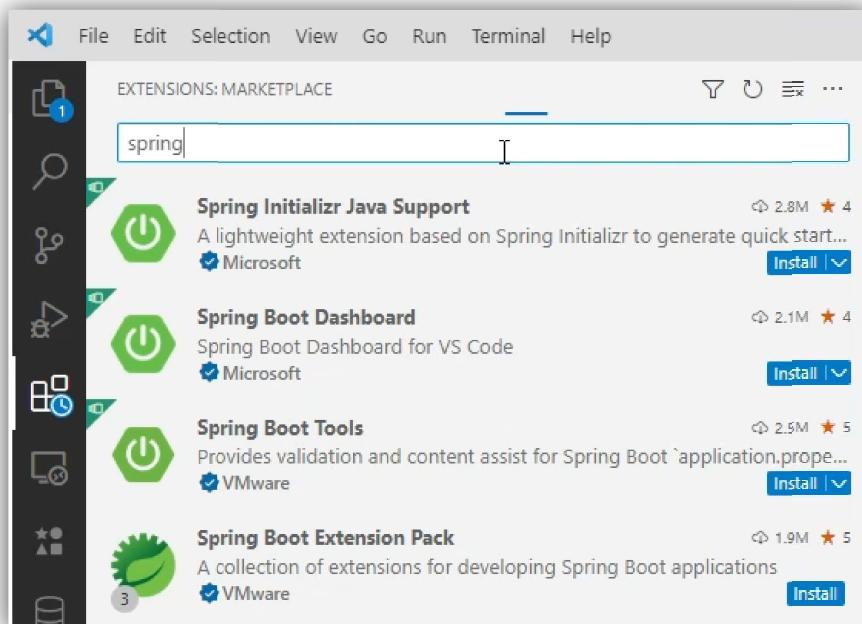


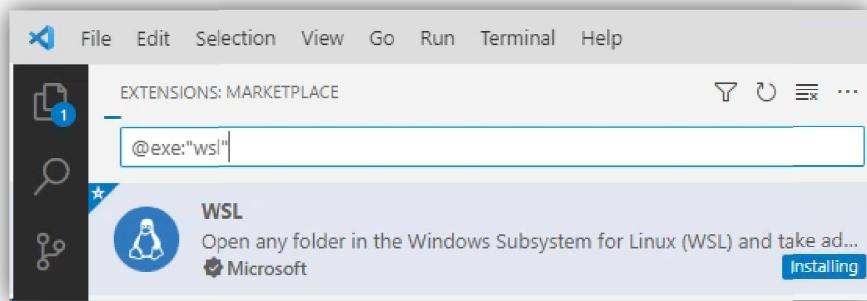
11. Installez les extensions suivantes :

Extension Pack for Java



Extensions Spring Boot suivantes : (quatre extensions)



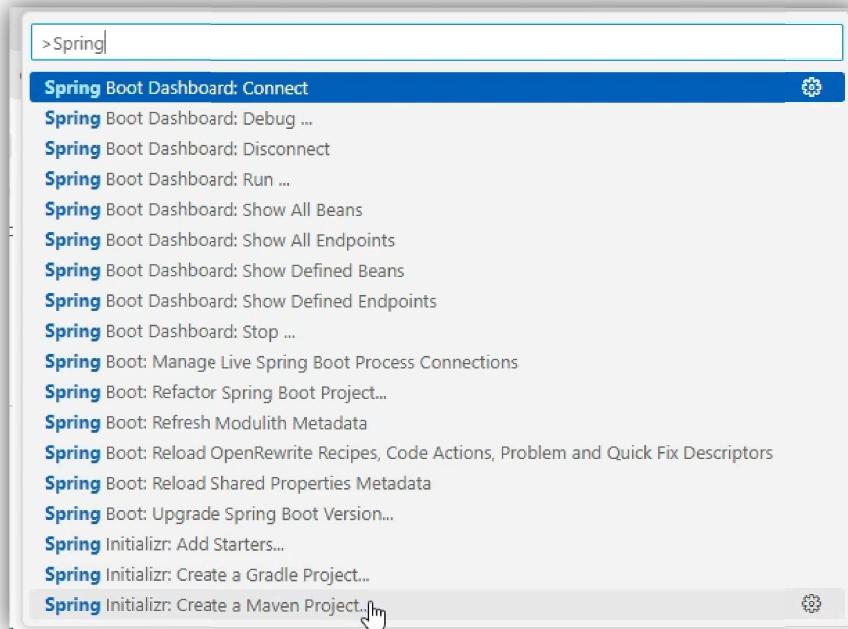
Extension wsl:

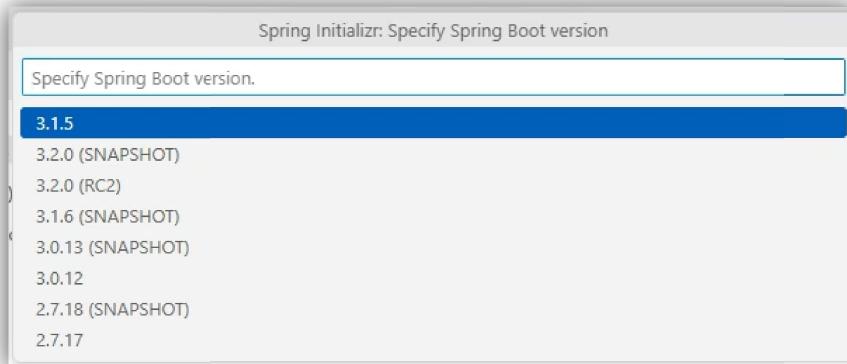
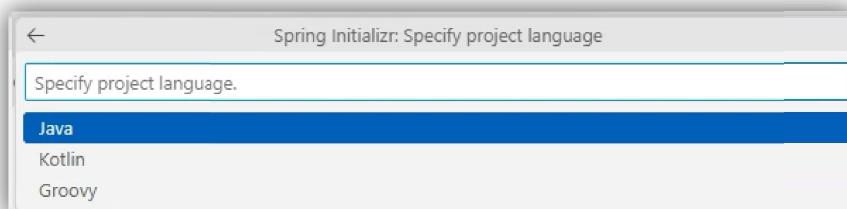
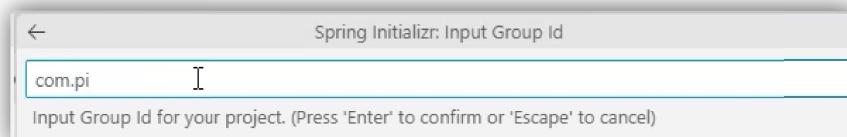
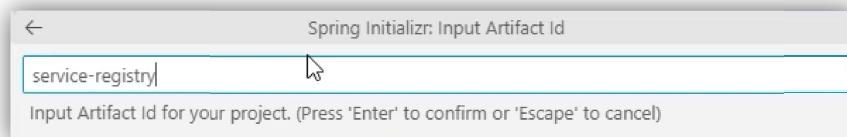
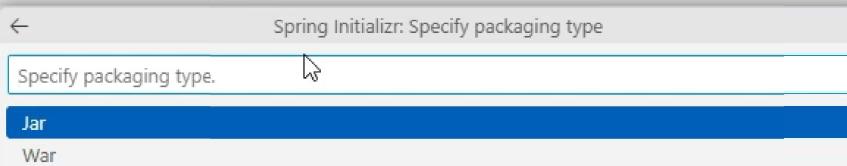
12. Redémarrez Microsoft Visual Studio Code

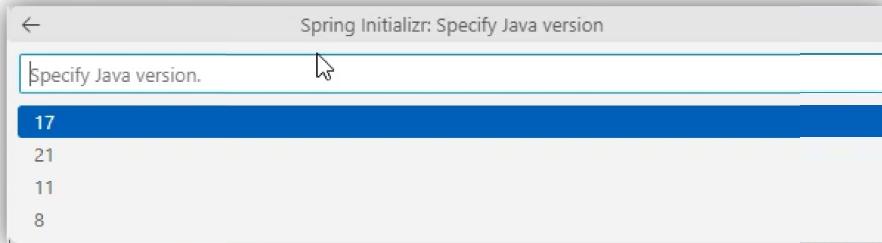
V. Création du Serveur Eureka

13. Appuyer sur Ctrl + Shift + P afin de lancer la palette des commandes.

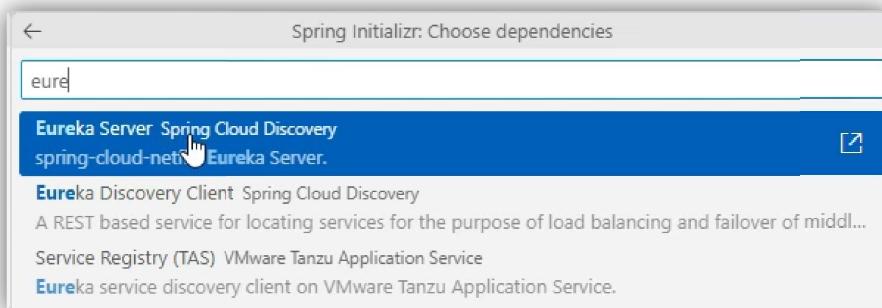
14. Cliquez sur (Spring Initializr : Create a Maven Project ...):



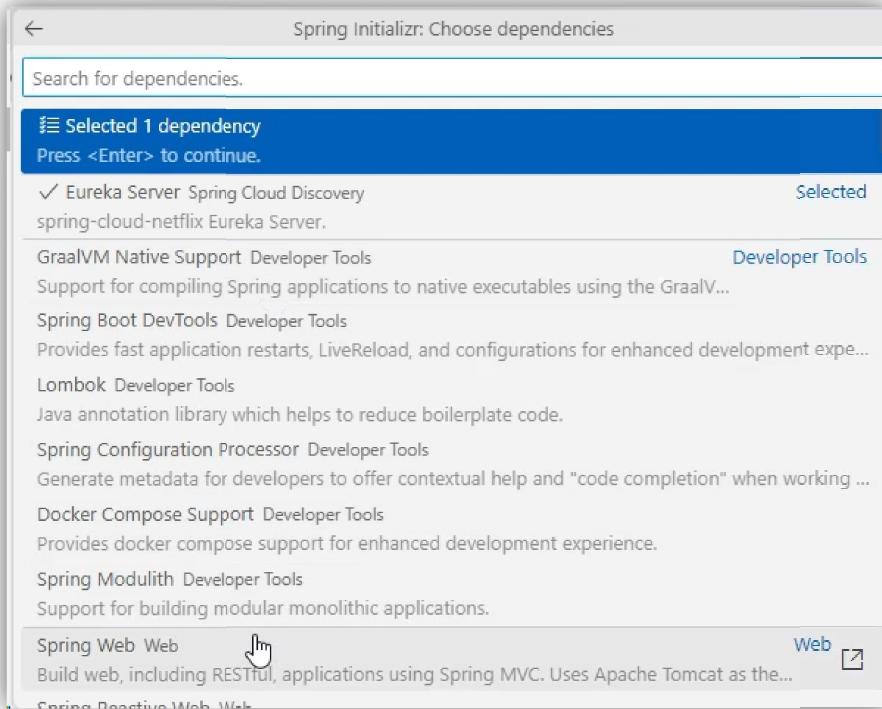
15. Sélectionnez la version (3.1.5**)****16. Sélectionnez le langage de programmation (**Java**)****17. Choisir comme nom de package (**com.pi**)****18. Indiquez le nom de projet (**service-registry**):****19. Choisir comme type de package (**Jar**)****20. Choisir la version de Java Development Kit (**17**)**



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

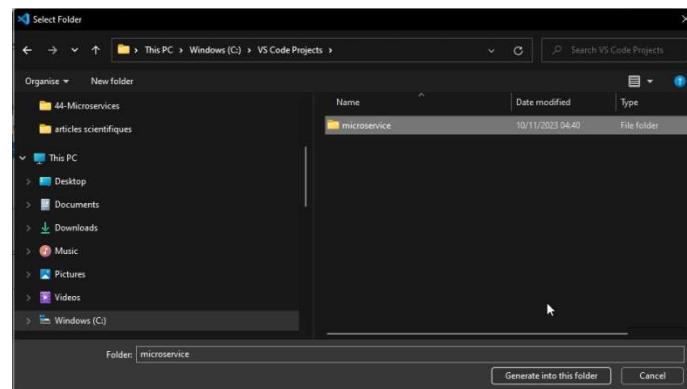


22. Ajoutez la dépendance (**Spring Web – Web**)

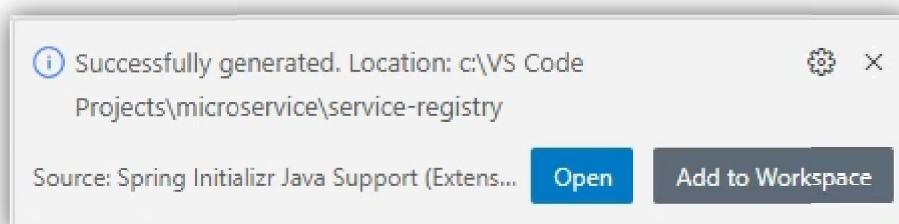


23. Sous le lecteur (C:), créer le chemin suivant (**VS Code Projects\microservice**)

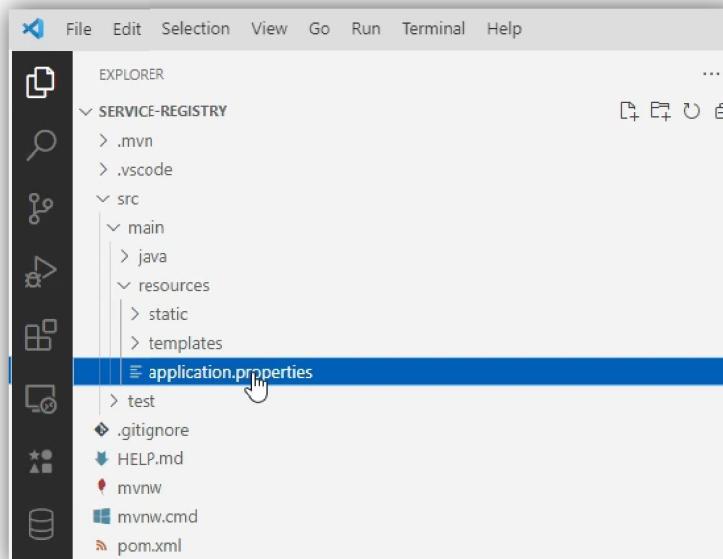
24. Sélectionnez le dossier (**microservice**) puis cliquez sur (**Generate into this folder**) :

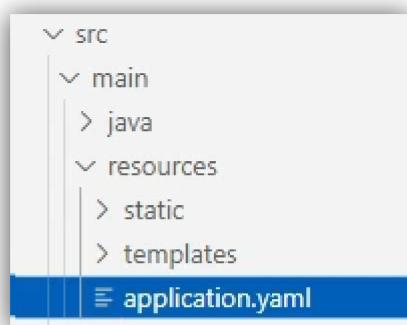
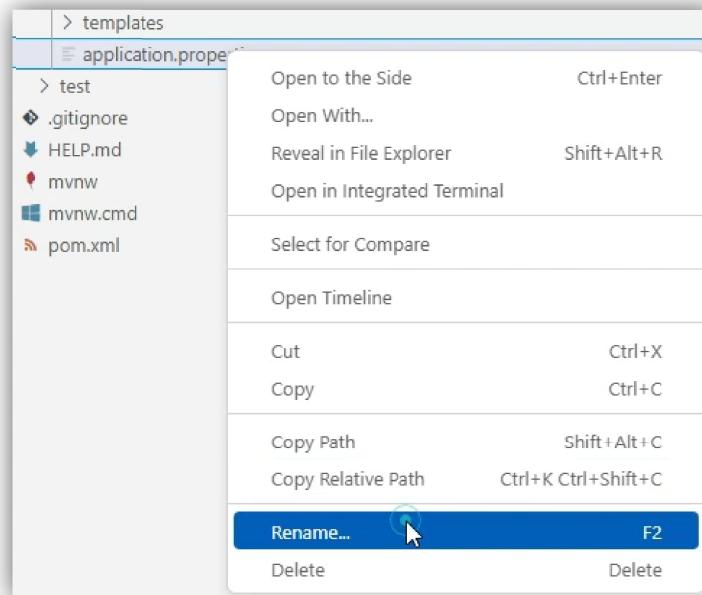


25. Cliquez sur (Open):



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

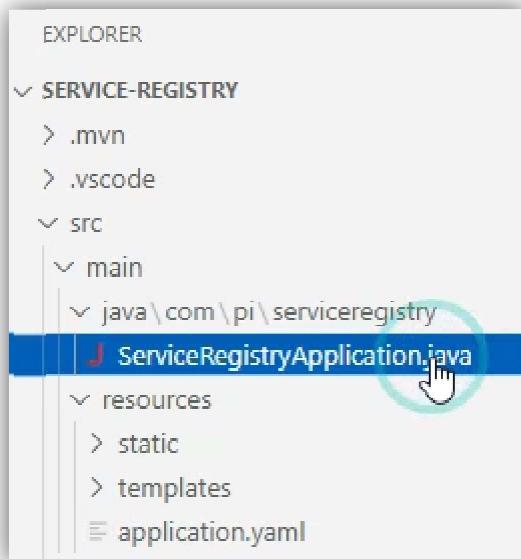
```

src > main > java > com > pi > serviceregistry > J 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
  
```

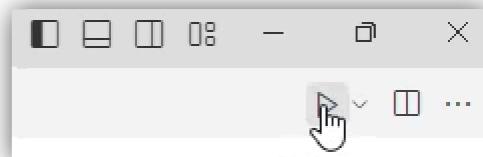
28. Se déplacer au fichier (**application.yaml**), et écrire le code suivant :

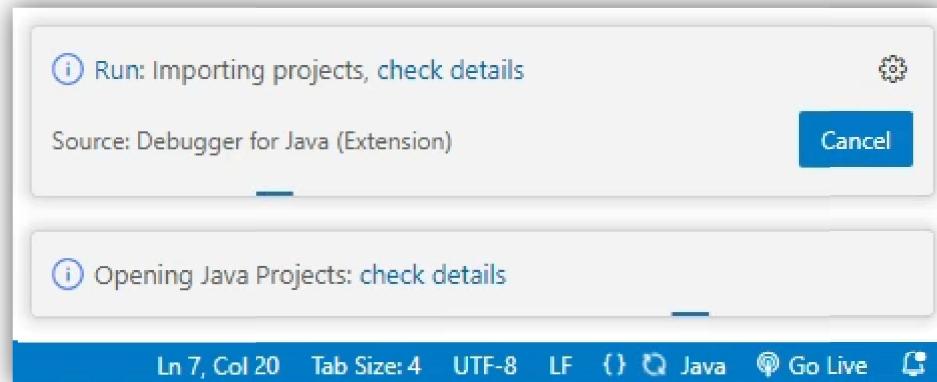
```
src > main > resources > application.yaml
1 server:
2   port: 8761
3 spring:
4   application:
5     name: service-registry
6 eureka:
7   instance:
8     hostname: localhost
9   client:
10    register-with-eureka: false
11    fetch-registry: false
12    serviceUrl:
13      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
14
```

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



30. Puis, Exécuter la fonction main :

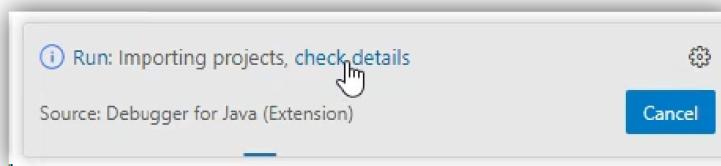




31. Affichez le terminal :



32. Cliquez sur (check details) :

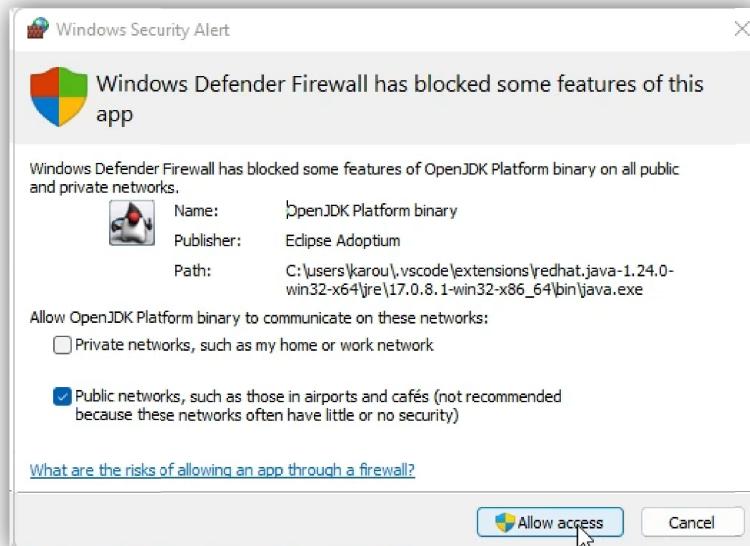


33. Et voici l'avancement de l'importation du projet Maven :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ROBOT DOCUMENTATION ROBOT OUTPUT
9f2c2b87 LookupJDKToolchainsJob [Done]
7b27d521 Importing Maven project(s) - 7MB/10MB (72%) https://repo.maven.apache.org/maven2/org/bouncycastle/bcprov-jdk18on/1.73/bcprov-jdk18on-1.73.jar
...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ROBOT DOCUMENTATION ROBOT OUTPUT
9f2c2b87 LookupJDKToolchainsJob [Done]
7b27d521 Resolve plugin dependency [Done]
99307ee Synchronizing projects [Done]
67cb1f51 Synchronizing projects [Done]
d340ce47 Validate documents [Done]
6a5a9215 Building [Done]
0530ab0b Send Classpath Notifications [Done]
35f53552 Send Classpath Notifications [Done]
9ab38c89 Building [Done]
b4052fee Publish Diagnostics [Done]
40979f36 Background task - 0%
ed1b65bd Refreshing workspace - 0% Refreshing '/service-registry'.
```

A screenshot of the Eclipse IDE interface, showing the "TERMINAL" tab selected. The terminal window displays the progress of a Maven project import, including tasks like "LookupJDKToolchainsJob [Done]", "Importing Maven project(s)", and "Building [Done]".

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



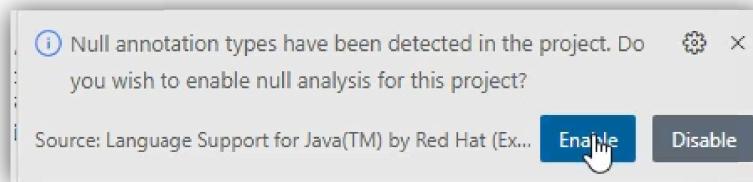
```
PS C:\VS Code Projects\microservice\service-registry> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '@C:\Users\karou\AppData\Local\Temp\cp_w4p8co1czxc64xieob7b2sy.argfile' 'com.pi.serviceregistry.ServiceRegistryApplication'

```
 \ \ /-
 (()\-----\O-----\ \ \ \ \
 \
 :: Spring Boot :: (v3.1.5)
  ```

2023-11-10T05:20:07.964+01:00  INFO 23916 --- [           main] c.p.s.ServiceRegistryApplication      : Starting ServiceRegis
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**

System Status

| Environment | test    | Current time             | 2023-11-10T05:20:21 +0100 |
|-------------|---------|--------------------------|---------------------------|
| Data center | default | Uptime                   | 00:00                     |
|             |         | Lease expiration enabled | false                     |
|             |         | Renews threshold         | 1                         |
|             |         | Renews (last min)        | 0                         |

DS Replicas

Instances currently registered with Eureka

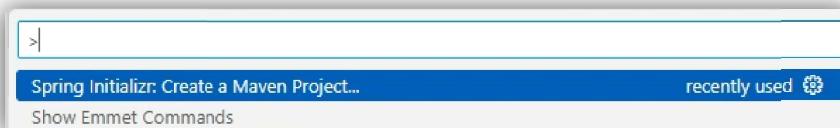
| Application            | AMIs | Availability Zones | Status |
|------------------------|------|--------------------|--------|
| No instances available |      |                    |        |

General Info

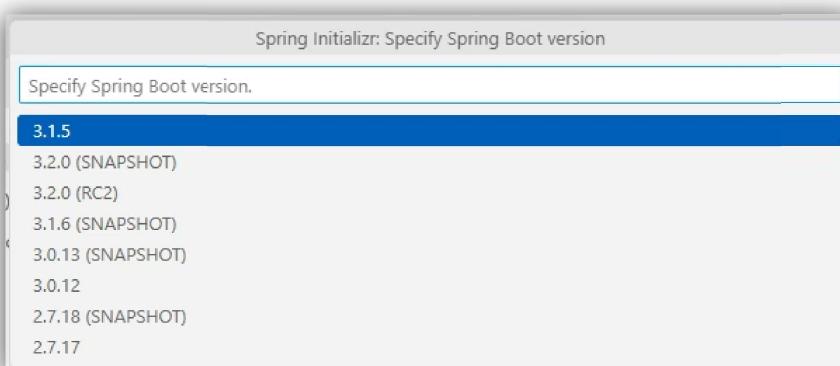
| Name                 | Value      |
|----------------------|------------|
| total-avail-memory   | 96mb       |
| num-of-cpus          | 12         |
| current-memory-usage | 58mb (60%) |
| server-uptime        | 00:00      |
| registered-replicas  |            |
| unavailable-replicas |            |
| available-replicas   |            |

## VI. Création du Microservice « Department »

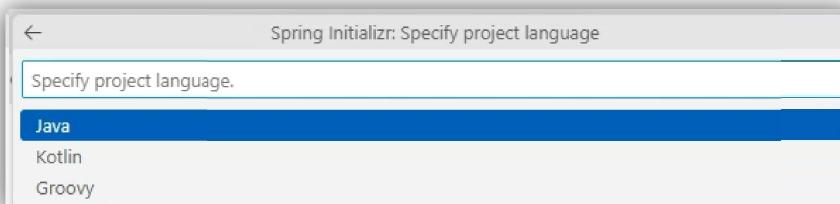
37. Appuyer sur Ctrl + Shift + P afin de lancer la palette des commandes.
38. Cliquez sur ( Spring Initializr : Create a Maven Project ...):



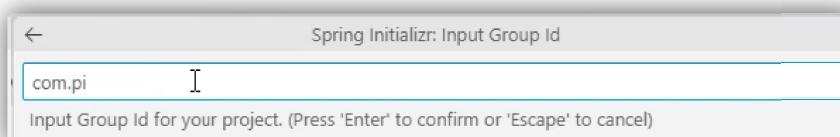
39. Sélectionnez la version (3.1.5)



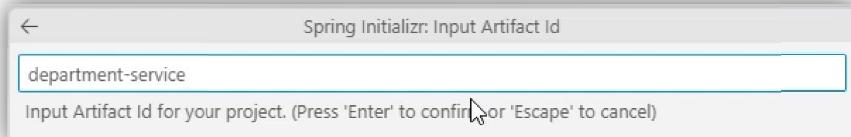
40. Sélectionnez le langage de programmation (Java)



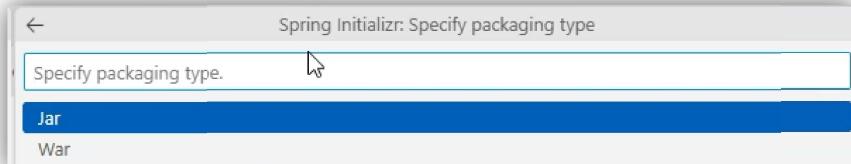
41. Choisir comme nom de package (com.pi)



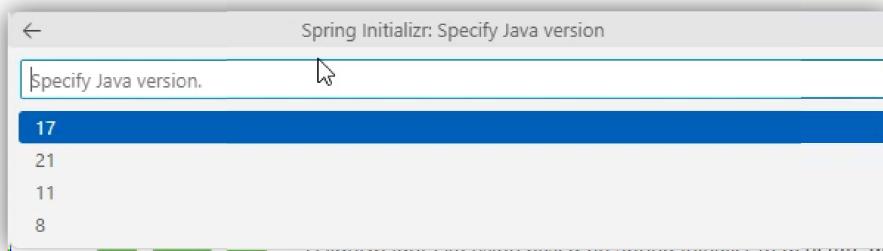
42. Indiquez le nom de projet (department-service):



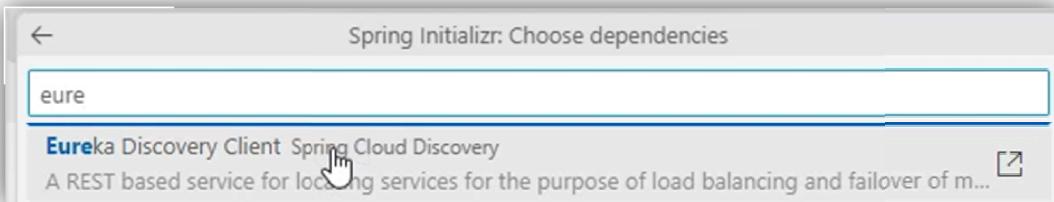
#### 43. Choisir comme type de package (Jar)



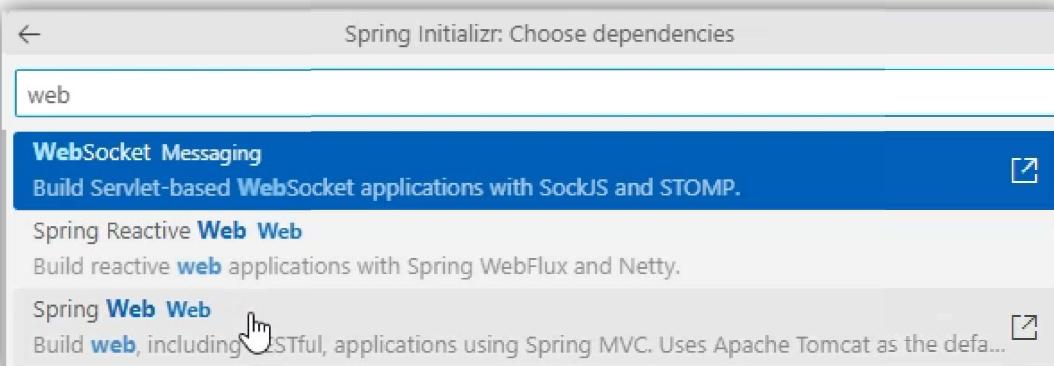
#### 44. Choisir la version de Java Development Kit (17)



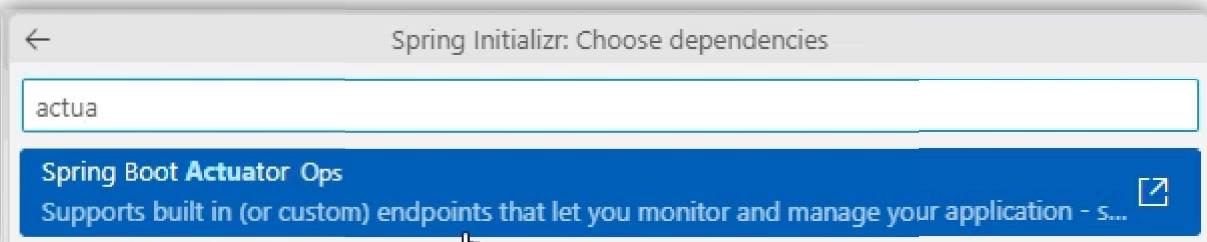
#### 45. Ajoutez la dépendance (Eureka Discovery Client – Spring Cloud Discovery)



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

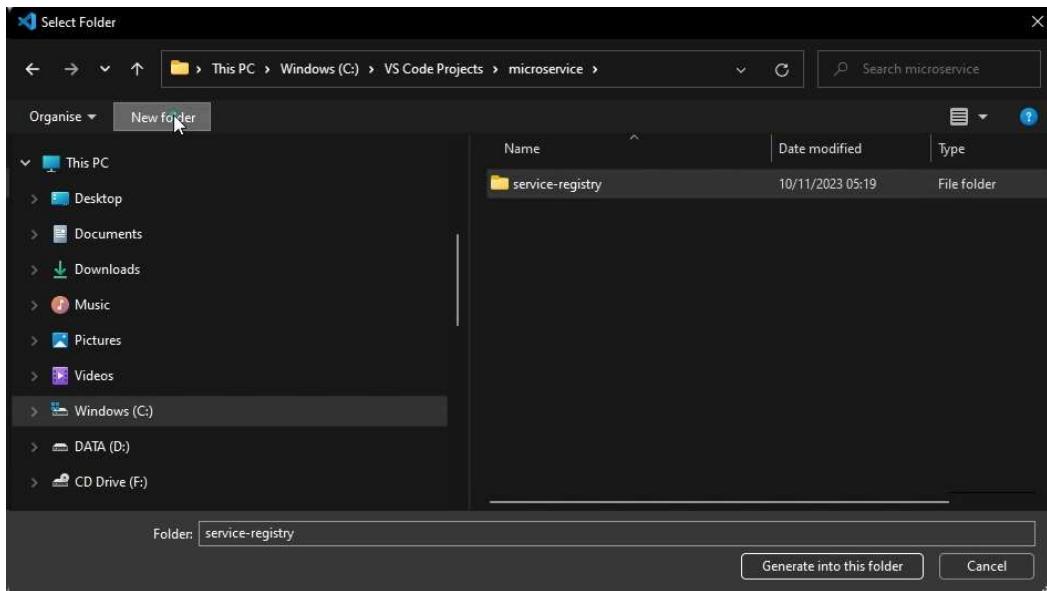


#### 47. Ajoutez la dépendance (Spring Boot Actuator – Ops)

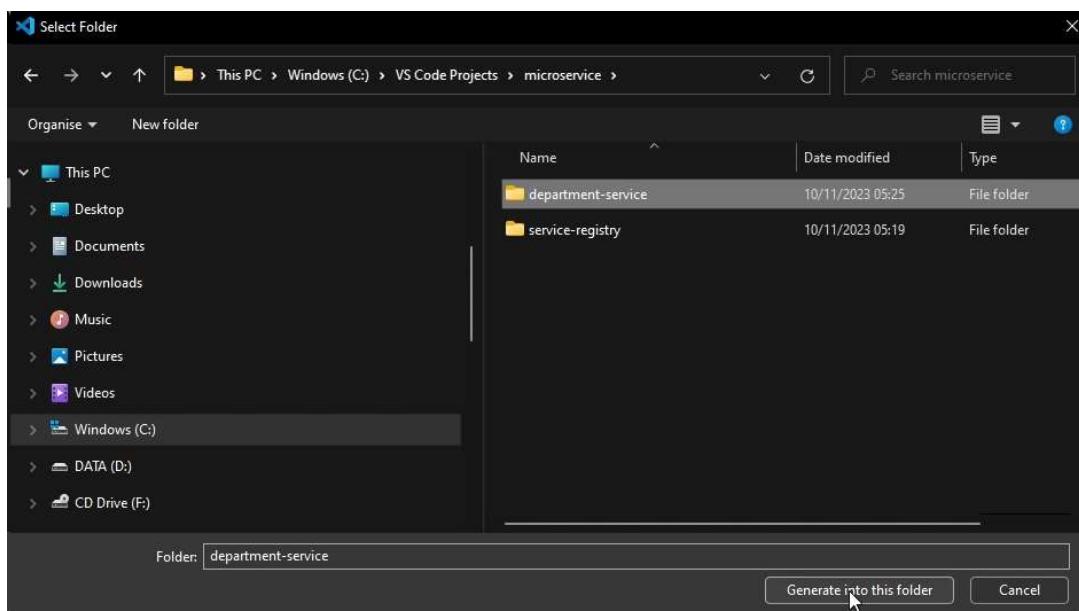


48. Appuyer sur (Entrée).

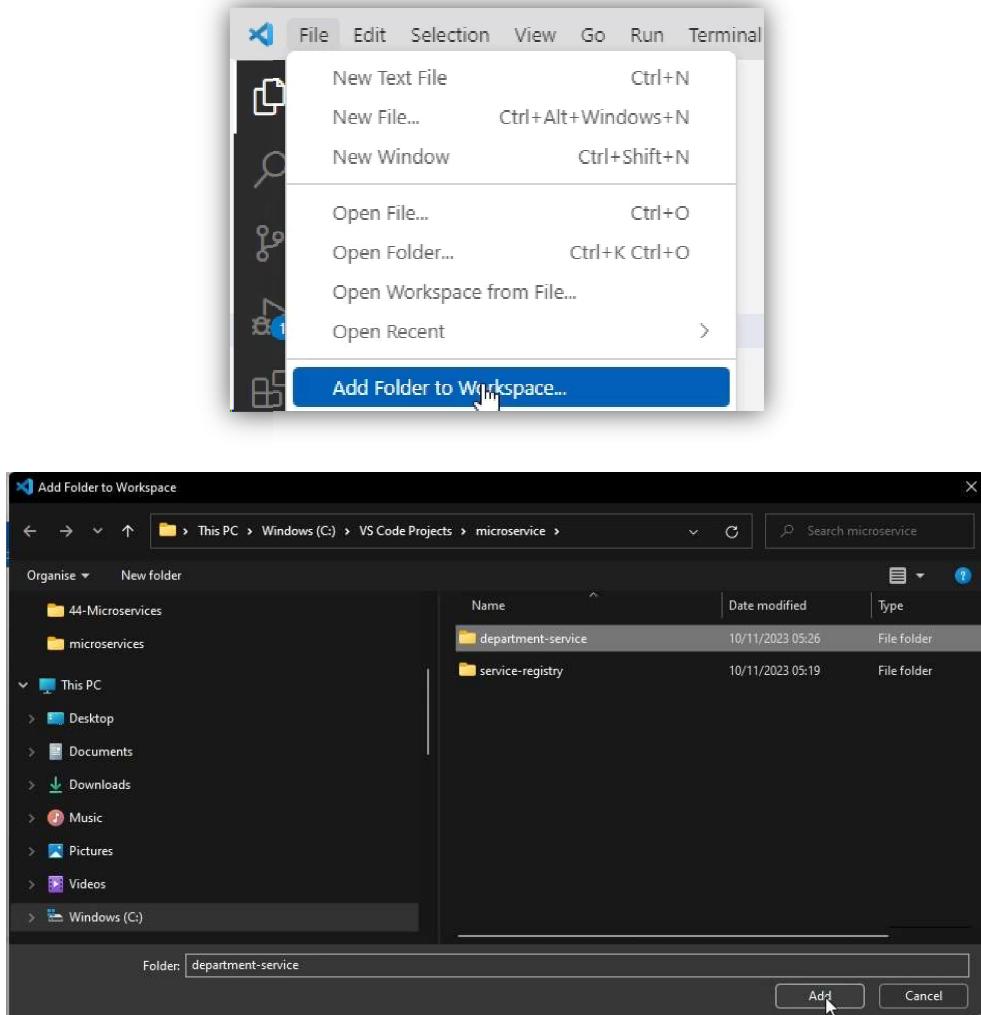
49. Créez un nouveau répertoire sous (C:\VS Code Projects\microservice) et appelez le (department-service)



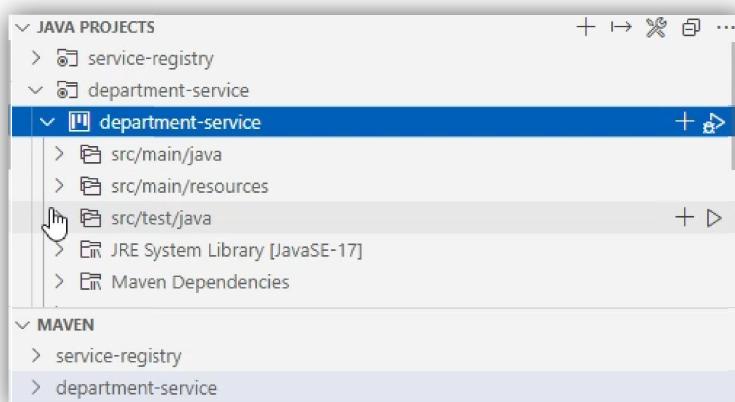
50. Sélectionnez le répertoire (department-service) puis cliquez sur (Generate into this folder)



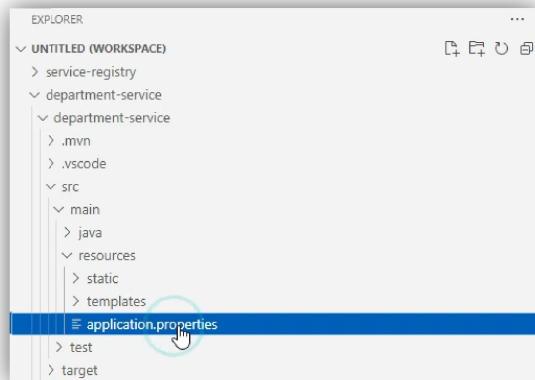
51. Depuis le projet (service-registry), ajoutez projet (department-service) à l'espace de travail:



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)

```

server:
 port: 8081
spring:
 application:
 name: department-service
eureka:
 instance:
 hostname: localhost
 client:
 serviceUrl:
 defaultZone: http://${eureka.instance.hostname}:8761/eureka/

```

The screenshot shows the VS Code editor with the 'application.yaml' file open. The file contains configuration for a Spring Boot application to register with an Eureka server. It specifies port 8081, the application name as 'department-service', and the Eureka registration details.

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

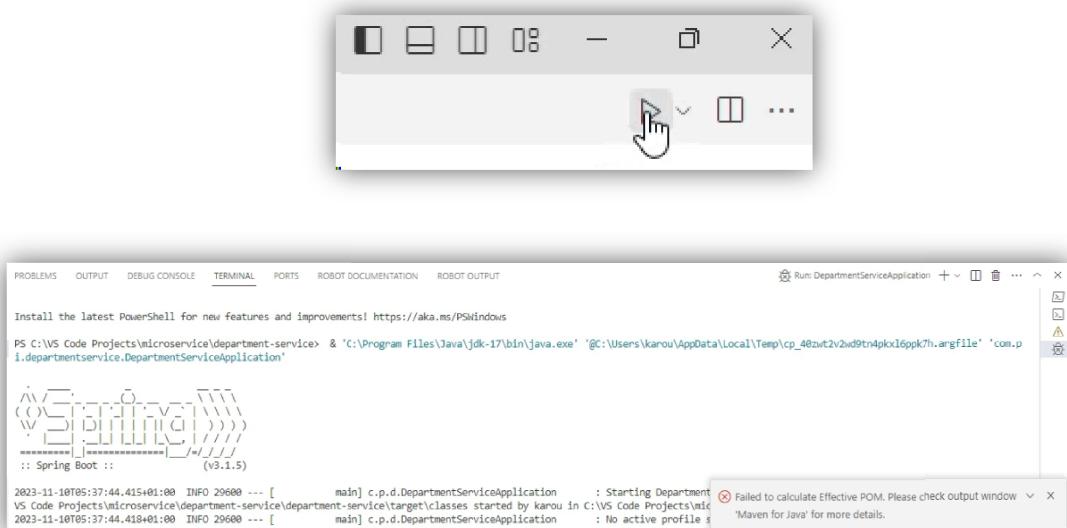
```

package com.pi.departmentservice;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
@SpringBootApplication
@EnableDiscoveryClient
public class DepartmentServiceApplication {
 public static void main(String[] args) {
 SpringApplication.run(DepartmentServiceApplication.class, args);
 }
}

```

The screenshot shows the VS Code editor with the 'DepartmentServiceApplication.java' file open. The code defines a Spring Boot application with an annotation that enables the discovery client feature, allowing it to register with the Eureka service.

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



The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following Spring Boot startup logs:

```

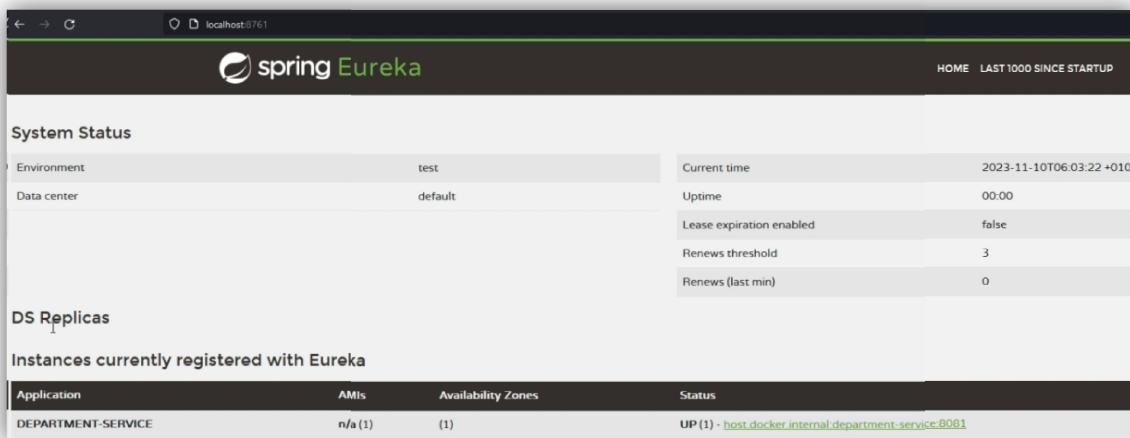
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\VS Code Projects\microservice\department-service> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '@C:\Users\karou\AppData\Local\Temp\cp_40zwt2v2id9tn4pkx16ppk7h.argfile' 'com.p.d.departmentservice.DepartmentServiceApplication'

.\\
(())
:: Spring Boot ::

2023-11-10T05:37:44.415+01:00 INFO 29600 --- [main] c.p.d.DepartmentServiceApplication : Starting DepartmentServiceApplication in 0.099 seconds (JVM running for 0.100)
2023-11-10T05:37:44.418+01:00 INFO 29600 --- [main] c.p.d.DepartmentServiceApplication : No active profile set, falling back to default profiles: 'Maven for Java' for more details.

```

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



The screenshot shows the Spring Eureka dashboard at [localhost:8761](http://localhost:8761). The interface includes sections for System Status and DS Replicas.

**System Status**

| Environment | test    | Current time             | 2023-11-10T06:03:22 +0100 |
|-------------|---------|--------------------------|---------------------------|
| Data center | default | Uptime                   | 00:00                     |
|             |         | Lease expiration enabled | false                     |
|             |         | Renew threshold          | 3                         |
|             |         | Renews (last min)        | 0                         |

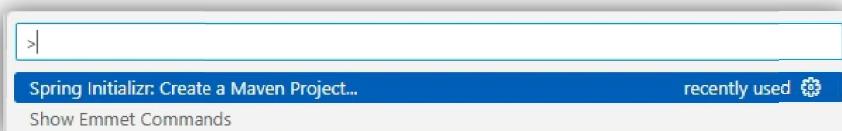
**DS Replicas**

Instances currently registered with Eureka

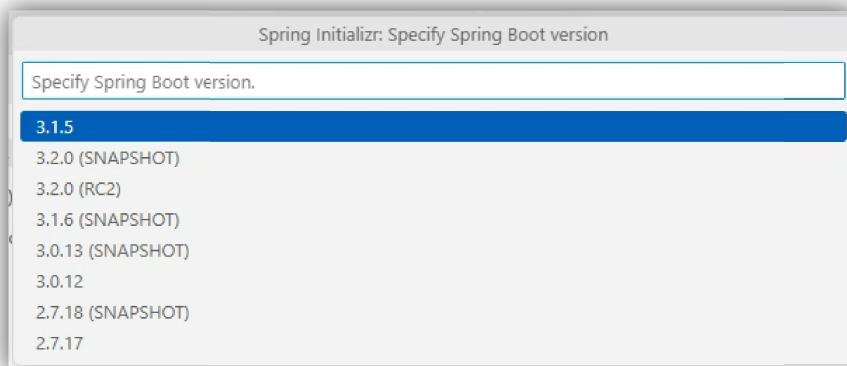
| Application        | AMIs    | Availability Zones | Status                                                |
|--------------------|---------|--------------------|-------------------------------------------------------|
| DEPARTMENT-SERVICE | n/a (1) | (1)                | UP (1) - host.docker.internal:department-service:8081 |

## VII. Création du Serveur de Configuration

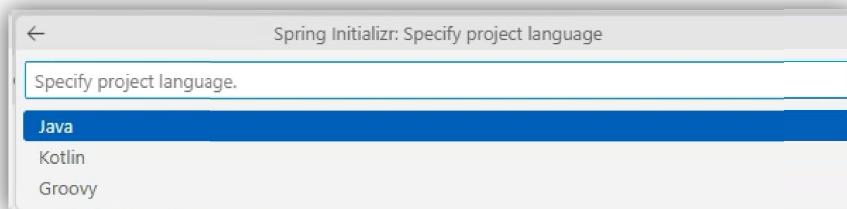
58. Lancez une nouvelle instance de Microsoft Visual Studio Code.
59. Appuyer sur **Ctrl + Shift + P** afin de lancer la palette des commandes.
60. Cliquez sur (**Spring Initializr : Create a Maven Project ...**):



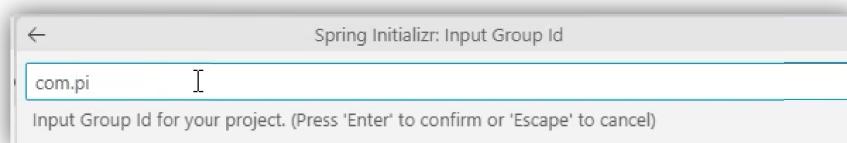
61. Sélectionnez la version (**3.1.5**)



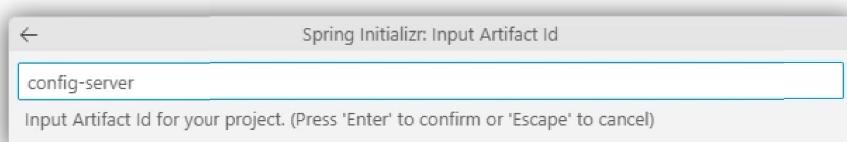
62. Sélectionnez le langage de programmation (**Java**)

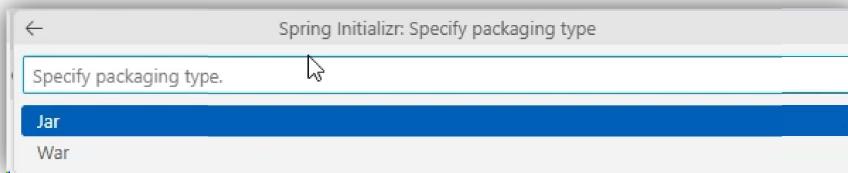
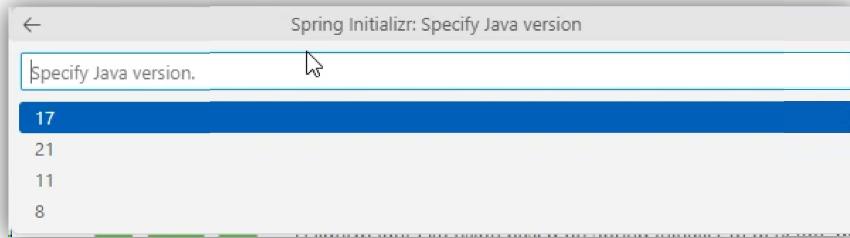
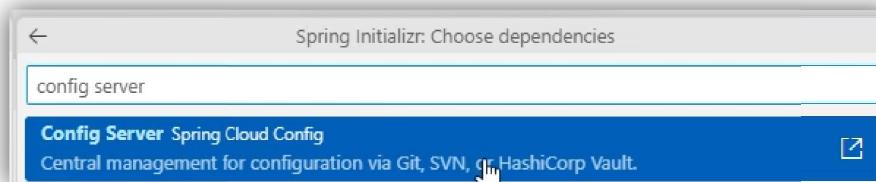
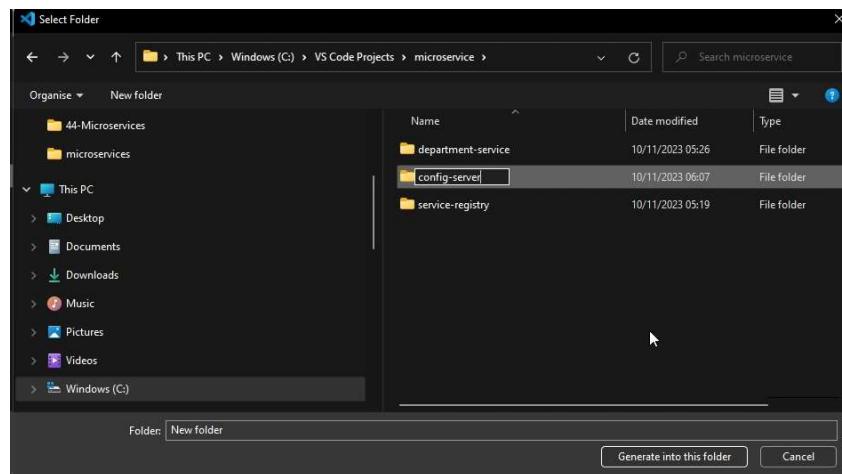


63. Choisir comme nom de package (**com.pi**)

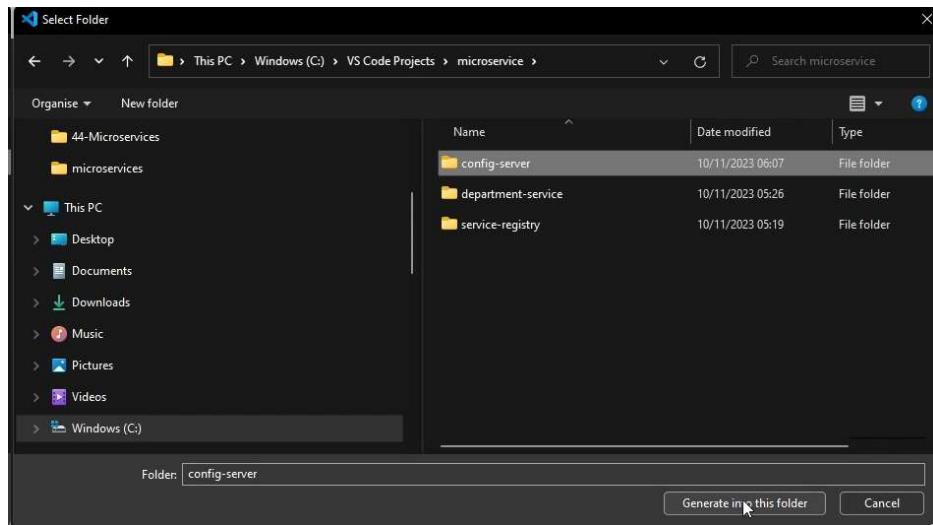


64. Indiquez le nom de projet (**config-server**):

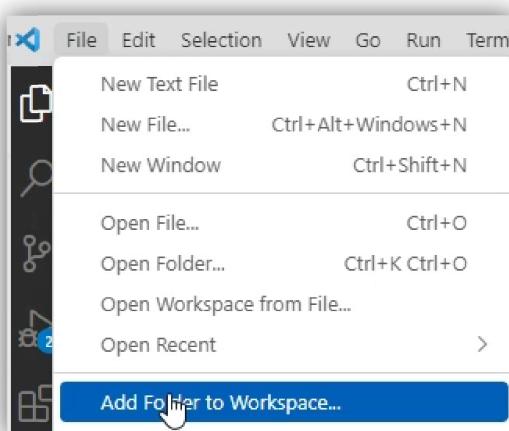


**65. Choisir comme type de package (Jar)****66. Choisir la version de Java Development Kit (17)****67. Ajouter la dépendance (Config Server – Spring Cloud Config)****68. Créer un nouveau répertoire appelé (config-server) sous (C:\VS Code Projects\microservice), puis le sélectionner.**

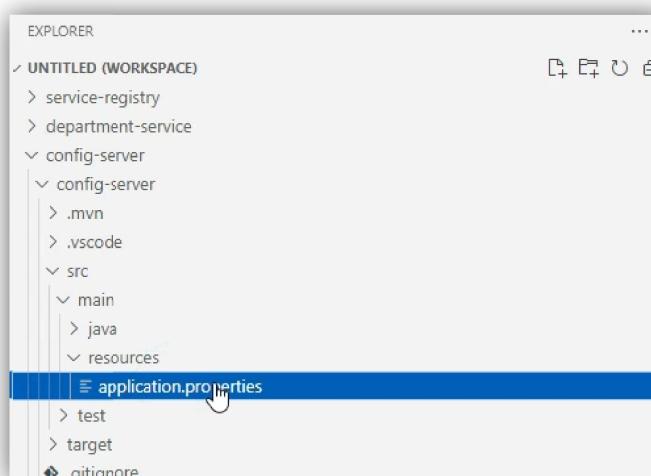
69. Cliquez sur le bouton (**Generate into this folder**) :



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 :



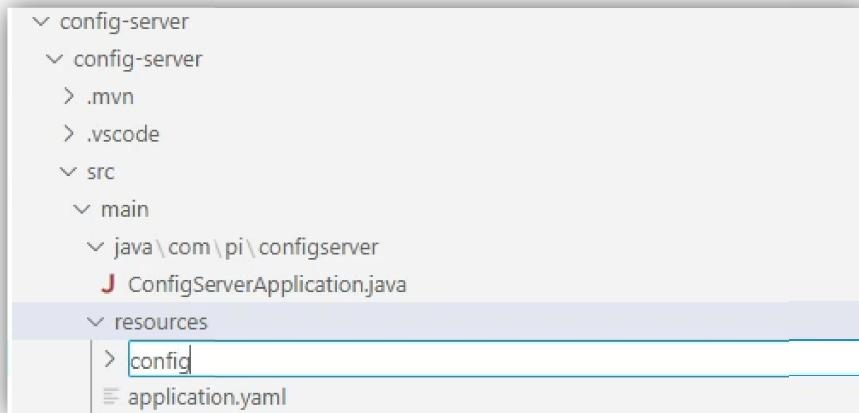
73. Ajoutez l'annotation (`@EnableConfigServer`) au niveau du fichier (`ConfigServerApplication.java`) :

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

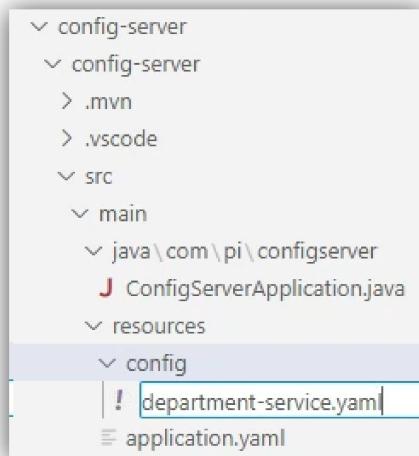
74. Se déplacer au fichier (**application.yaml**), et écrire le code suivant :

```
server:
 port: 8088
spring:
 profiles:
 active: native
```

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



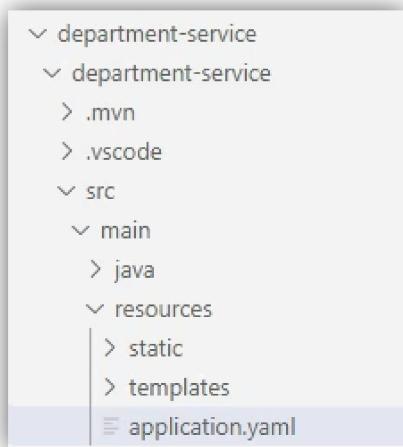
76. Créez un fichier appelé (department-service.yaml) sous (resources\config\)



77. Ecrivez le code suivant :

```
config-server > config-server > src > main > resources > config > ! department-service.yaml
1 server:
2 port: 8081
3
4 eureka:
5 instance:
6 hostname: localhost
7 client:
8 serviceUrl:
9 defaultZone: http://${eureka.instance.hostname}:8761/eureka/
10
```

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



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

```

department-service > department-service > src > main > resources > application.yaml
1 spring:
2 application:
3 name: department-service
4 config:
5 import: "optional:configserver:http://localhost:8088"
6

```

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

```

17 <java.version>17</java.version>
18 <spring-cloud.version>2022.0.4</spring-cloud.version>
19 </properties>
20 <dependencies>
21 <dependency>
22 <groupId>org.springframework.boot</groupId>
23 <artifactId>spring-boot-starter-actuator</artifactId>
24 </dependency>
25 <dependency>
26 <groupId>org.springframework.boot</groupId>
27 <artifactId>spring-boot-starter-web</artifactId>
28 </dependency>
29 <dependency>
30 <groupId>org.springframework.cloud</groupId>
31 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
32 </dependency>
33 <dependency>
34 <groupId>org.springframework.cloud</groupId>
35 <artifactId>spring-cloud-starter-config</artifactId>
36 </dependency>
37 <dependency>
38 <groupId>org.springframework.boot</groupId>
39 <artifactId>spring-boot-starter-test</artifactId>
40 <scope>test</scope>
41 </dependency>
42
43

```

81. Arrêter toutes les applications:



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



**(1) service-Registry** : 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 :

| Application        | AMIs    | Availability Zones | Status                                                |
|--------------------|---------|--------------------|-------------------------------------------------------|
| DEPARTMENT-SERVICE | n/a (1) | (1)                | UP (1) - host.docker.internal:department-service:8081 |

## 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

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\windows\system32> wsl.exe --status
Default Version: 2

The Windows Subsystem for Linux kernel can be manually updated with 'wsl --update', but automatic updates cannot occur due to your system settings.
To receive automatic kernel updates, please enable the Windows Update setting: 'Receive updates for other Microsoft products when you update Windows'.
For more information please visit https://aka.ms/wsl2kernel.

The WSL 2 kernel file is not found. To update or restore the kernel please run 'wsl --update'.
PS C:\windows\system32> wsl --update
Installing: Windows Subsystem for Linux
The requested operation requires elevation.

```

Redémarrez votre PC en cas de problèmes.

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\windows\system32> wsl --update
Installing: Windows Subsystem for Linux
[= 2.0%]

```

```

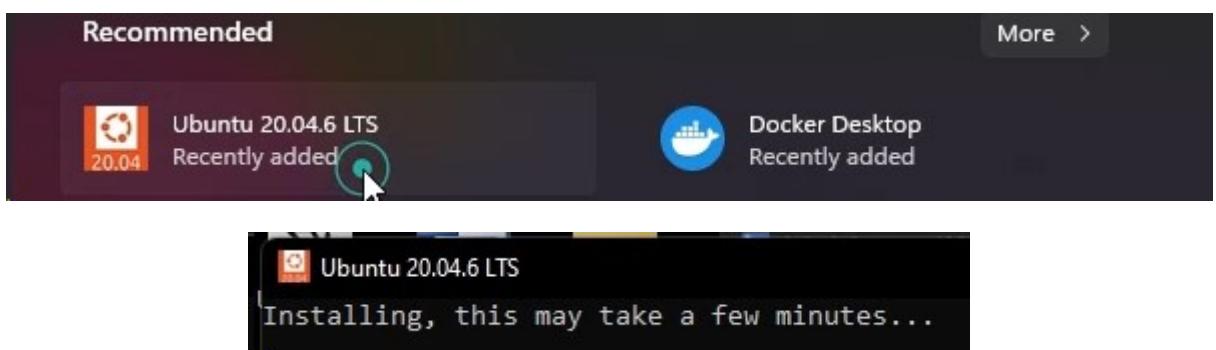
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\windows\system32> wsl --update
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
PS C:\windows\system32>

```

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%
 Swap usage: 0% IPv4 address for eth0: 172.29.235.163

 Expanded Security Maintenance for Applications is not enabled.

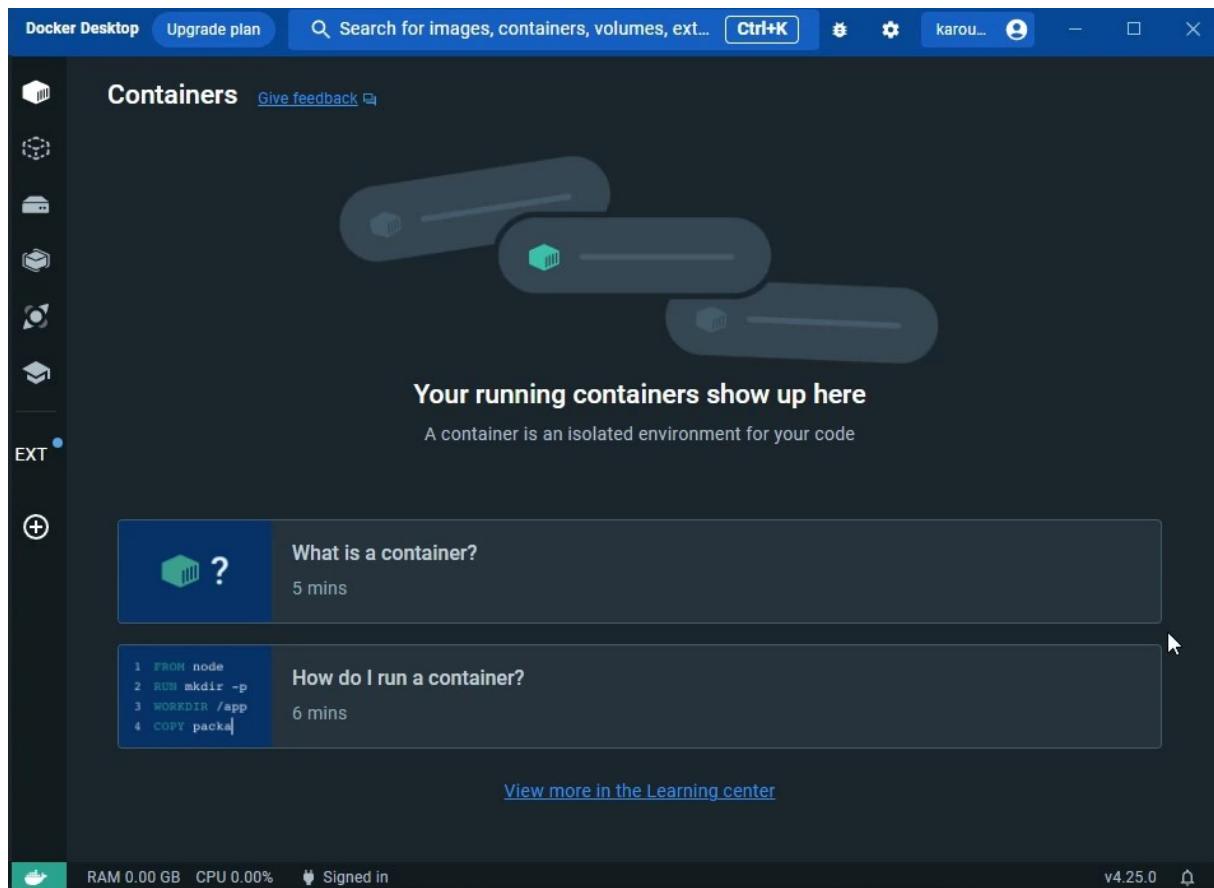
 0 updates can be applied immediately.

 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 :

Google

zipkin server docker

All Videos Images News More Tools

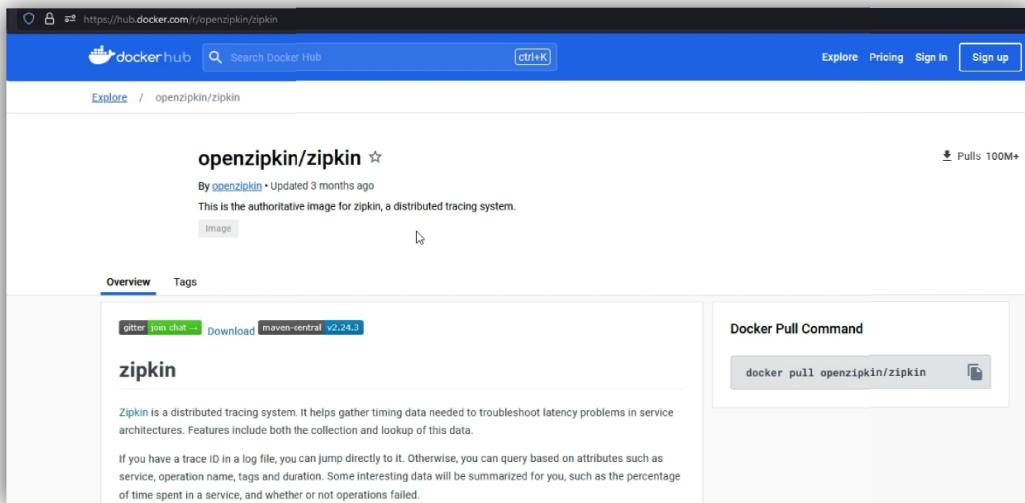
About 425,000 results (0.27 seconds)

Docker Hub  
<https://hub.docker.com/r/openzipkin/zipkin> ::

[openzipkin/zipkin - Docker Image](#)

Zipkin is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in service architectures.

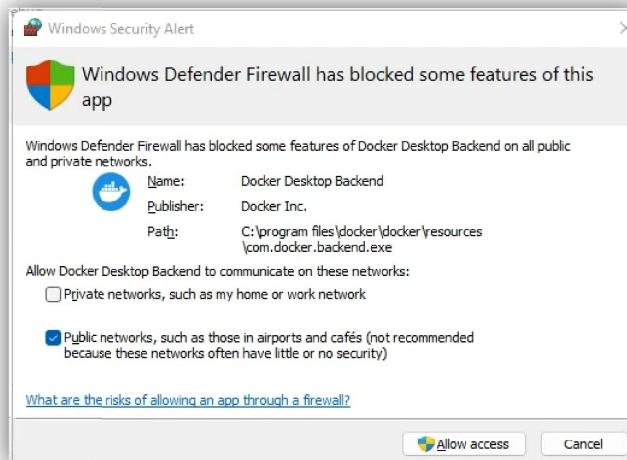
85. Cliquez sur le lien ci-dessus pour accéder à l'image du docker du serveur ZIPKIN :



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

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ROBOT DOCUMENTATION ROBOT OUTPUT
○ PS C:\VS Code Projects\microservice\department-service> docker run -d -p 9411:9411 openzipkin/zipkin
Unable to find image 'openzipkin/zipkin:latest' locally
latest: Pulling from openzipkin/zipkin
c48dcabaa86e: Downloading [=====] 783.2kB/3.398MB
dccbaad6de45: Downloading [=] 187.8kB/3.412MB
bc3c1a0a5c4d: Download complete
1a0aee5c7a5a: Waiting
8517ac596bba: Waiting
fc9312ba5268: Waiting
1c8346113c13: Waiting
e92464172d9f: Waiting
17bdba0ce7cd: Waiting
```

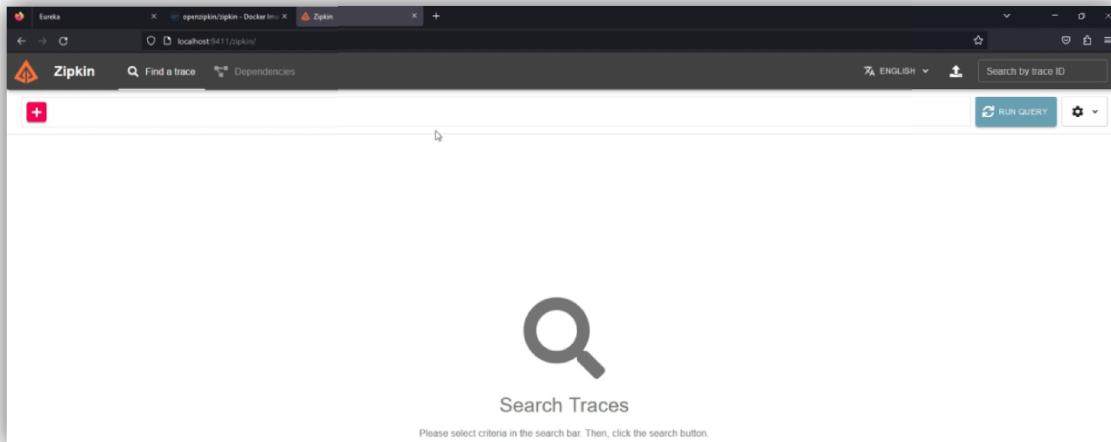
87. Autorisez l'accès :



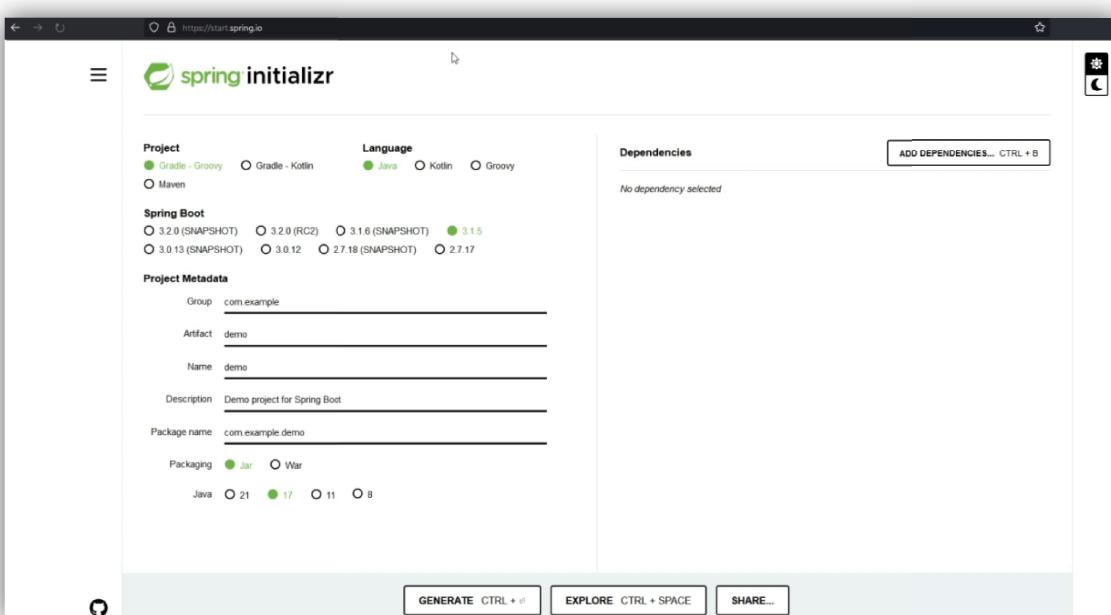
88. Testez l'adresse suivante :

localhost:9411/zipkin/

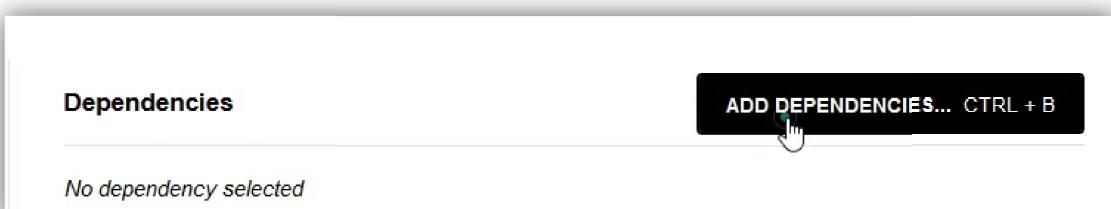
89. Voici le résultat :



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



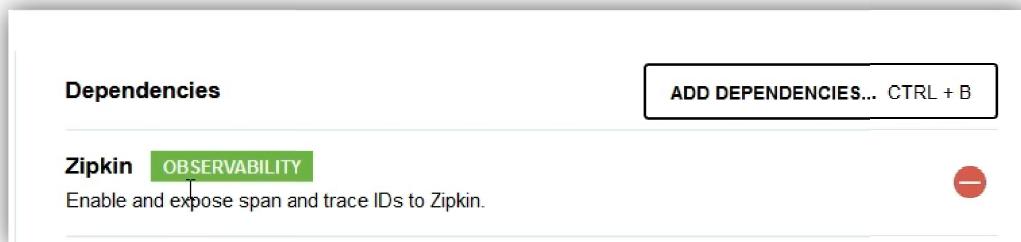
91. Cliquez sur (ajouter dépendances) :



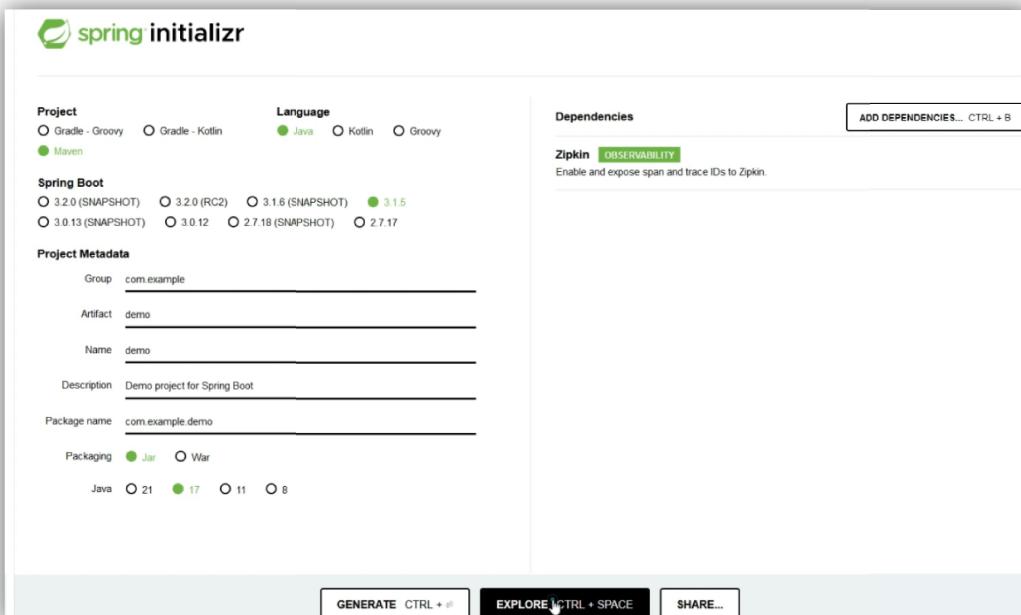
92. Saisir (zipkin) puis cliquez sur le lien de la dépendance:



93. Vous sauriez redirigés vers ceci :



94. Cliquez sur (Explore) :



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

[DOWNLOAD](#) [COPY](#)

```

21 <groupId>org.springframework.boot</groupId>
22 <artifactId>spring-boot-starter-actuator</artifactId>
23 </dependency>
24 <dependency>
25 <groupId>io.micrometer</groupId>
26 <artifactId>micrometer-tracing-bridge-brave</artifactId>
27 </dependency>
28 <dependency>
29 <groupId>io.zipkin.reporter2</groupId>
30 <artifactId>zipkin-reporter-brave</artifactId>
31 </dependency> ↴

32 <dependency>
33 <groupId>org.springframework.boot</groupId>
34 <artifactId>spring-boot-starter-test</artifactId>
35 <scope>test</scope>
36 </dependency>
37 </dependencies>

39 <build>
40 <plugins>
41 <plugin>
42 <groupId>org.springframework.boot</groupId>
43 <artifactId>spring-boot-maven-plugin</artifactId>
44 <configuration>
45 <image>
46 <builder>paketobuildpacks/builder-jammy-base:latest</builder>
47 </image>
48 </configuration>
49 </plugin>

```

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

UN TITLED (WORKSPACE)

- > service-registry
- < department-service
 < department-service
 > .mvn
- > .vscode
- < src
 < main
 < java \ com \ pi \ departmentservice
 J DepartmentServiceApplication.java
 > resources
- > test
- > target
- .gitignore
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml

> config-server

department-service > department-service > pom.xml

```

17 <java.version>17</java.version>
18 <spring-cloud.version>2022.0.4</spring-cloud.version>
19 </properties>
20 <dependencies>
21 <dependency>
22 <groupId>org.springframework.boot</groupId>
23 <artifactId>spring-boot-starter-actuator</artifactId>
24 </dependency>
25 <dependency>
26 <groupId>org.springframework.boot</groupId>
27 <artifactId>spring-boot-starter-web</artifactId>
28 </dependency>
29 <dependency>
30 <groupId>org.springframework.cloud</groupId>
31 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
32 </dependency>
33 <dependency>
34 <groupId>org.springframework.cloud</groupId>
35 <artifactId>spring-cloud-starter-config</artifactId>
36 </dependency>
37 <dependency>
38 <groupId>io.micrometer</groupId>
39 <artifactId>micrometer-tracing-bridge-brave</artifactId>
40 </dependency>
41 <dependency>
42 <groupId>io.zipkin.reporter2</groupId>
43 <artifactId>zipkin-reporter-brave</artifactId>
44 </dependency>

```

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
1 server:
2 | port: 8081
3
4 eureka:
5 | instance:
6 | hostname: localhost
7 | client:
8 | serviceUrl:
9 | defaultZone: http://${eureka.instance.hostname}:8761/eureka/
10 management:
11 | tracing:
12 | sampling:
13 | 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 :

```
department-service > department-service > src > main > java > com > pi > departmentservice > model > J Employee.java > E Employee
1 package com.pi.departmentservice.model;
2
3 public class Employee {
4 |
5 }
6
 class Employee
 interface Employee
 enum Employee
 record Employee()
 abstract class Employee
 @interface Employee
```

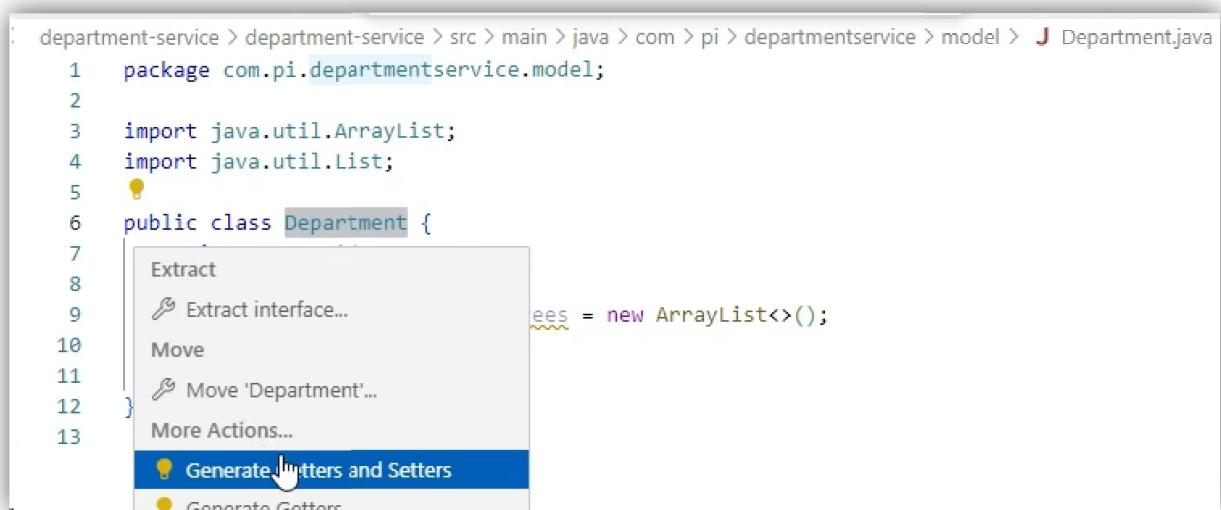
102. Modifiez le code pour qu'il soit comme suit :

```
department-service > department-service > src > main > java > com > pi > departmentservice > model > J Employee.java > E Employee
1 package com.pi.departmentservice.model;
2
3 public record Employee(Long id, Long departmentId, String name,int age, String position) {
4
5 }
6
```

## 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)

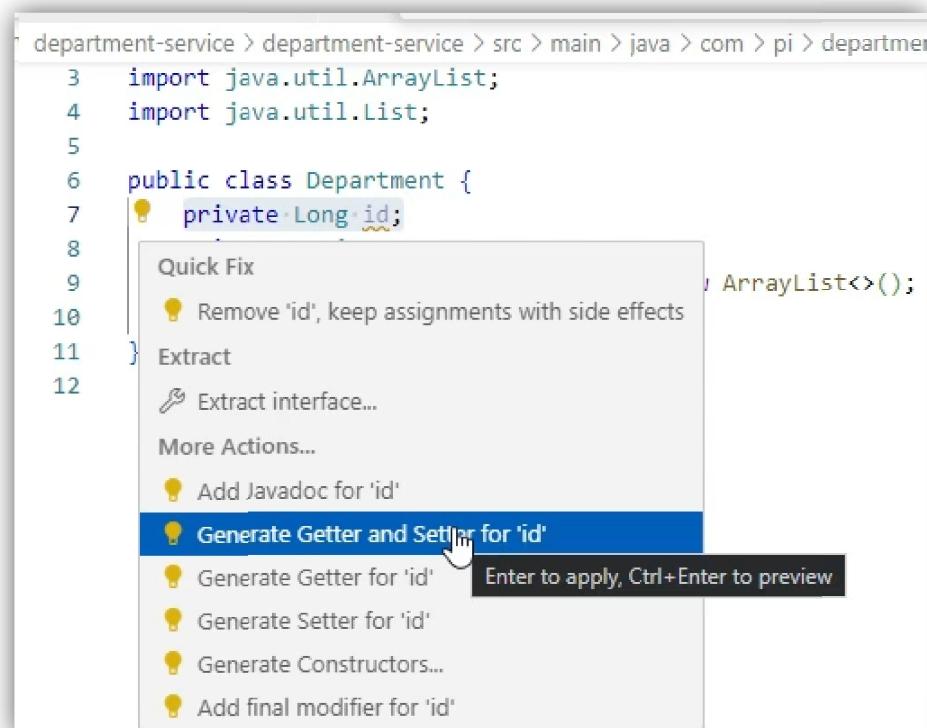


104. Voici le résultat :

```
department-service > department-service > src > main > java > com > pi > departmentservice > model > 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<>();
10 public Long getId() {
11 return id;
12 }
13 public void setId(Long id) {
14 this.id = id;
15 }
16 public String getName() {
17 return name;
18 }
19 public void setName(String name) {
20 this.name = name;
21 }
22 public List<Employee> getEmployees() {
23 return employees;
24 }
25 public void setEmployees(List<Employee> employees) {
26 this.employees = employees;
27 }
}
```

## Deuxième méthode :

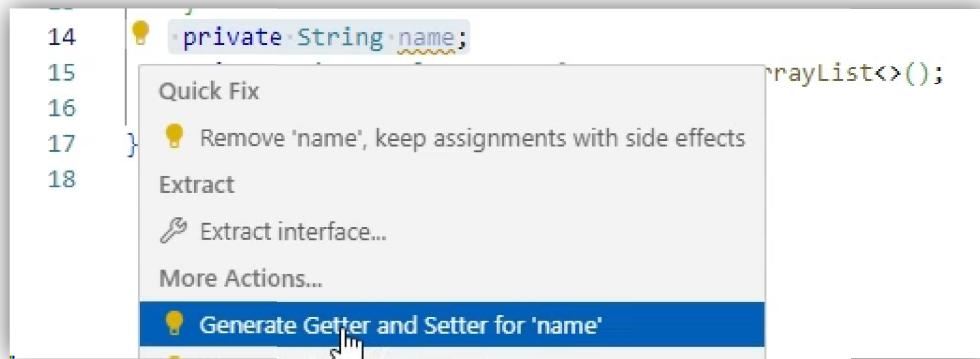
Allez au fichier (Department.java), sélectionnez la ligne qui contient le id, puis click droit pour lancer le menu contextuel ensuite cliquez sur (Generate Getter and Setter for « id »)



Et voici le résultat :

```
department-service > department-service > src > main > java > com > pi > departmentservice > model > J Department.java
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Department {
7 private Long id;
8 public Long getId() {
9 return id;
10 }
11 public void setId(Long id) {
12 this.id = id;
13 }
14 private String name;
15 private List<Employee> employees = new ArrayList<>();
16
17 }
18
```

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



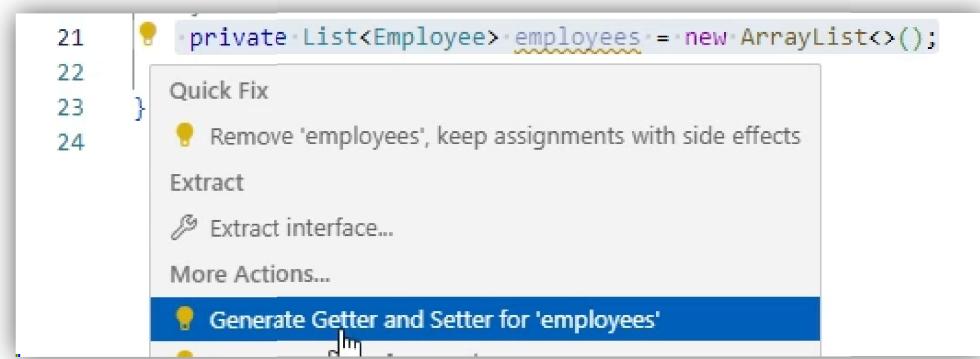
```
14 private String name;
15
16 }
17
18 }
```

Quick Fix

- Remove 'name', keep assignments with side effects
- Extract
- Extract interface...
- More Actions...

Generate Getter and Setter for 'name'

De même pour employees :



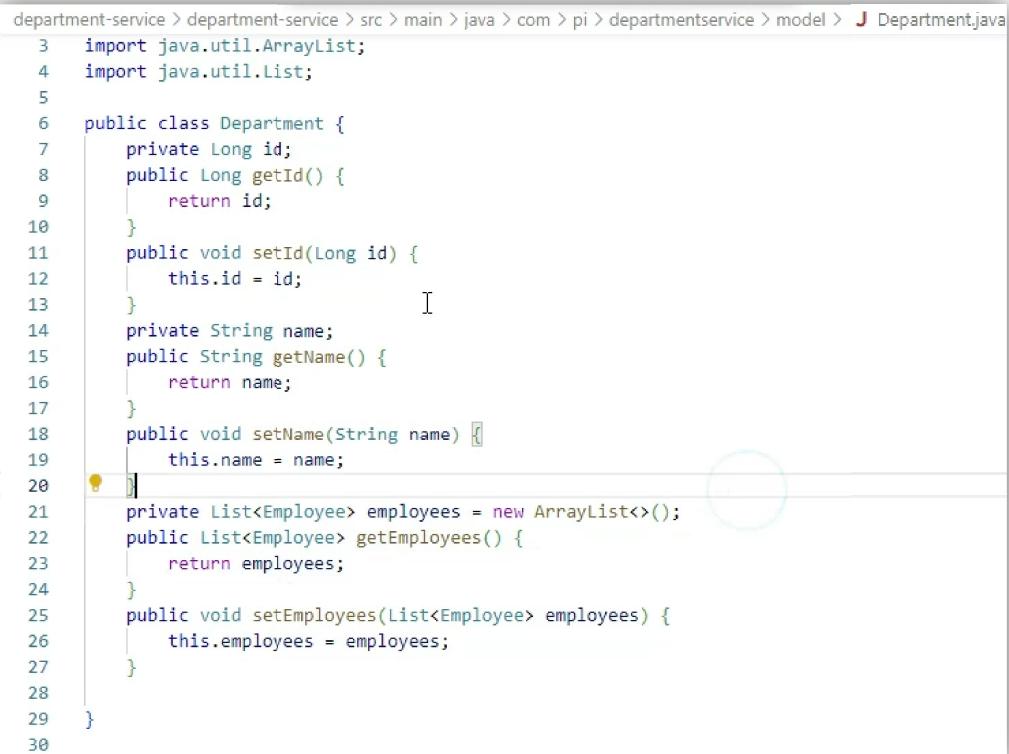
```
21 private List<Employee> employees = new ArrayList<>();
22
23 }
24
```

Quick Fix

- Remove 'employees', keep assignments with side effects
- Extract
- Extract interface...
- More Actions...

Generate Getter and Setter for 'employees'

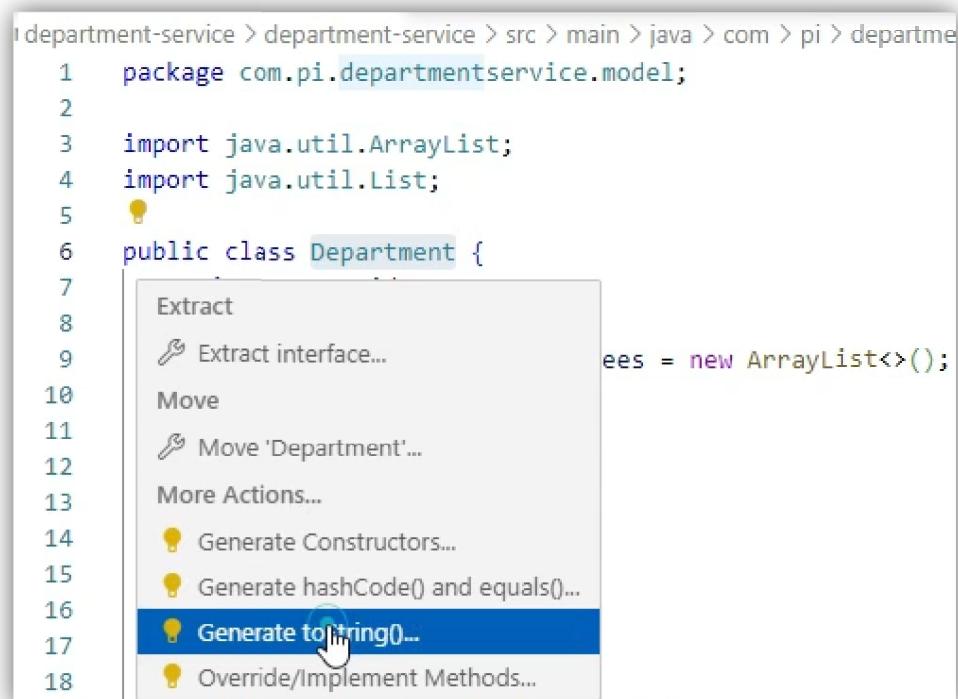
Et voici le code final après génération des Getters et Setters :



```
department-service > department-service > src > main > java > com > pi > departmentservice > model > J Department.java
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Department {
7 private Long id;
8 public Long getId() {
9 return id;
10 }
11 public void setId(Long id) {
12 this.id = id;
13 }
14 private String name;
15 public String getName() {
16 return name;
17 }
18 public void setName(String name) {
19 this.name = name;
20 }
21 private List<Employee> employees = new ArrayList<>();
22 public List<Employee> getEmployees() {
23 return employees;
24 }
25 public void setEmployees(List<Employee> employees) {
26 this.employees = employees;
27 }
28
29 }
30
```

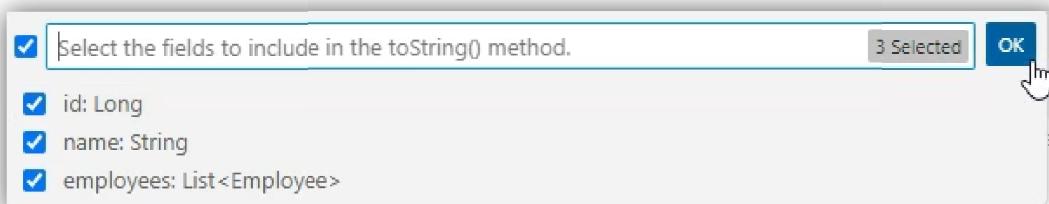
### Génération de la méthode `toString()` :

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



```
|department-service > department-service > src > main > java > com > com > pi > departme
1 package com.pi.departmentservice.model;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Department {
7
8 Extract
9 ↗ Extract interface...
10 Move
11 ↗ Move 'Department'...
12 More Actions...
13 ↗ Generate Constructors...
14 ↗ Generate hashCode() and equals()...
15 ↗ Generate toString()...
16 ↗ Generate hashCode() and equals()...
17 ↗ Override/Implement Methods...
18 }
```

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

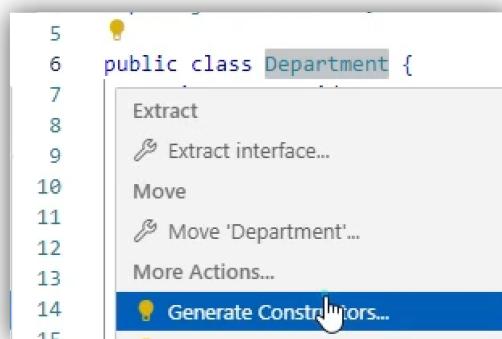


107. Et voici le résultat :

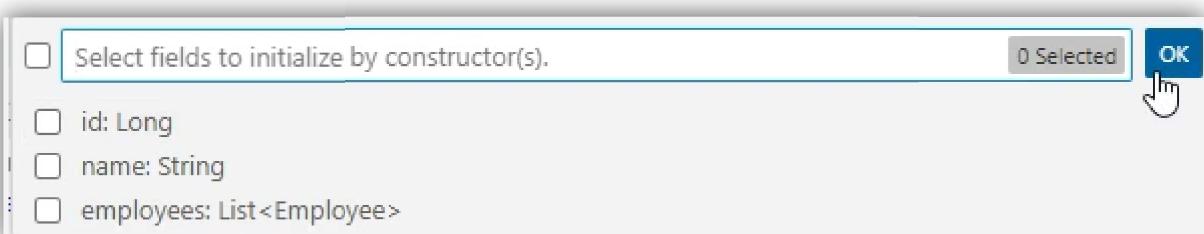
```
28 @Override
29 public String toString() {
30 return "Department [id=" + id + ", name=" + name + ", employees=" + employees + "]";
31 }
```

## Génération des constructeurs :

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



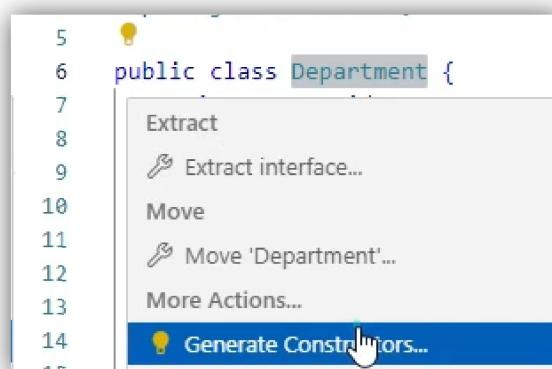
109. Puis créez un constructeur vide (gardez la sélection vide):



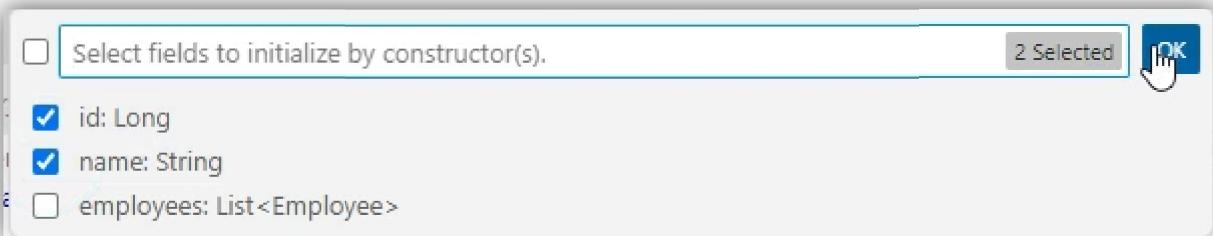
110. Voici le résultat de la génération:

```
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 :



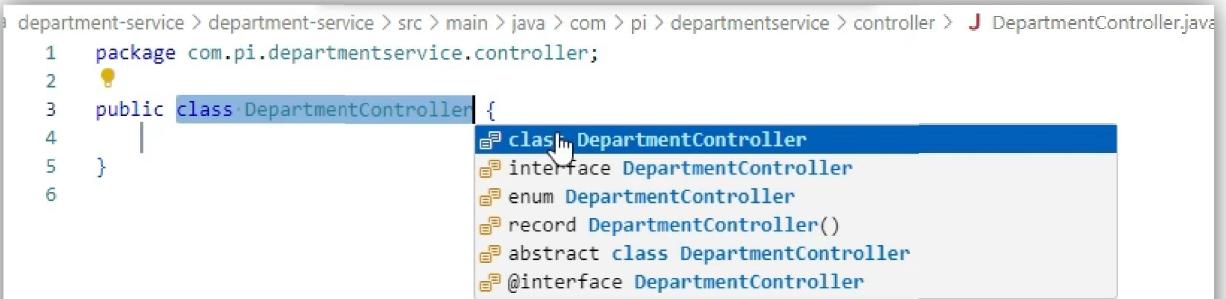
112. Sélectionnez les attributs « id » et « name », puis cliquez sur (OK) :



113. Et voici le résultat :

```
6 public class Department {
7 private Long id;
8 private String name;
9 private List<Employee> employees = new ArrayList<>();
10 public Department(Long id, String name) {
11 this.id = id;
12 this.name = name;
13 }
```

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



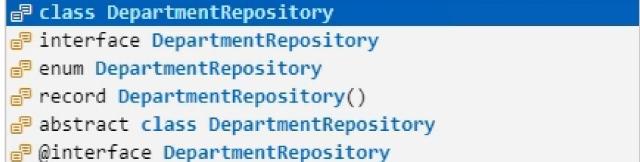
115. Ajouter l'annotation (@RestController) et (@RequestMapping("/department")):

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

```
1 package com.pi.departmentservice.controller;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 @RequestMapping("/department")
8 public class DepartmentController {
```

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

```
department-service > department-service > src > main > java > com > pi > departmentservice > repository > J DepartmentRepository.java
1 package com.pi.departmentservice.repository;
2
3 public class DepartmentRepository {
4 |
5 }
6
```



The screenshot shows the IntelliJ IDEA code editor with a code completion dropdown open over the class definition. The dropdown title is 'class DepartmentRepository'. It lists several items under this title: 'interface DepartmentRepository', 'enum DepartmentRepository', 'record DepartmentRepository()', 'abstract class DepartmentRepository', and '@interface DepartmentRepository'.

117. Ajouter l'annotation (@Repository):

```
: department-service > department-service > src > main > java > com > pi > departmentservice > repository > J DepartmentRepository.java
1 package com.pi.departmentservice.repository;
2
3 import org.springframework.stereotype.Repository;
4
5 @Repository
6 public class DepartmentRepository {
7
8 }
9
```

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

```
department-service > department-service > src > main > java > com > pi > departmentservice > repository > J DepartmentRepository.java
1 package com.pi.departmentservice.repository;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.springframework.stereotype.Repository;
7
8 import com.pi.departmentservice.model.Department;
9
10 @Repository
11 public class DepartmentRepository {
12
13 private List<Department> departments = new ArrayList<>();
14 public Department addDepartment(Department department){
15 departments.add(department);
16 return department;
17 }
18 public Department findById(Long id){
19 return departments.stream().filter(department -> department.getId().equals(id))
20 .findFirst()
21 .orElseThrow();
22 }
23 public List<Department> findAll(){
24 return departments;
25 }
26 }
27
```

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

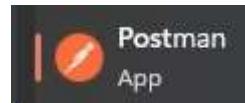
```
J DepartmentController.java 1 ●
department-service > department-service > src > main > java > com > pi > departmentservice > controller > J DepartmentController.java >
14 import com.pi.departmentservice.repository.DepartmentRepository;
15
16 @RestController
17 @RequestMapping("/department")
18 public class DepartmentController {
19 private static final Logger LOGGER = LoggerFactory.getLogger(clazz:DepartmentController.class);
20 @Autowired
21 private DepartmentRepository repository;
22 @PostMapping
23 public Department add(@RequestBody Department department){
24 LOGGER.info(format:"Department add:{}",department);
25 return repository.addDepartment(department);
26 }
27 @GetMapping
28 public List<Department> findAll(){
29 LOGGER.info(msg:"Department find");
30 return repository.findAll();
31 }
32 @GetMapping("/{id}")
33 public Department findById(@PathVariable Long id){
34 LOGGER.info(format:"Department find: id={}",id);
35 return repository.findById(id);
36 }
37 }
```

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



## X. Test du Microservice « Department »

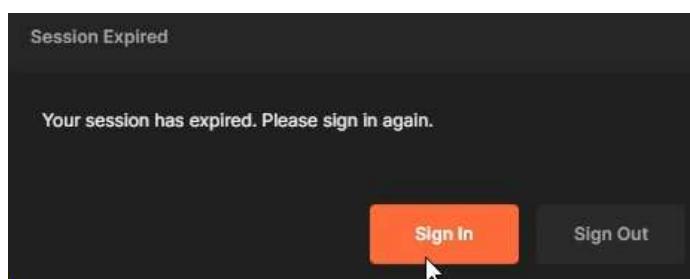
121. Afin de tester notre microservice (**Department**) on va lancer l'application (**Postman**) :



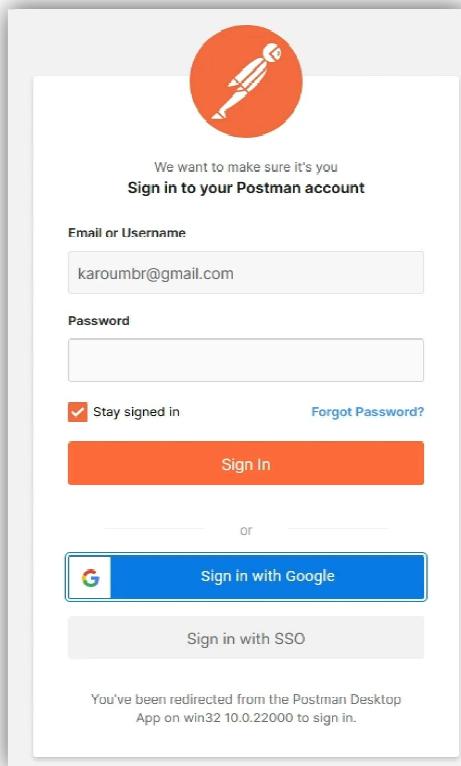
122. En cas de besoin, autorisez postman pour se connecter à Internet :



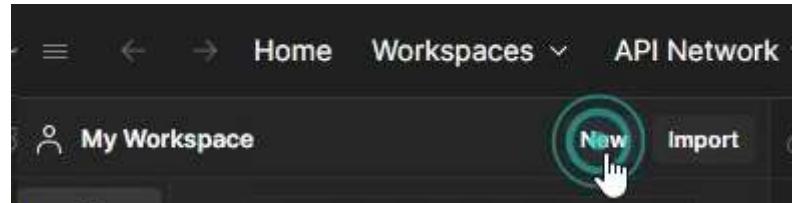
123. Connectez-vous à votre compte sur (**Postman**) :



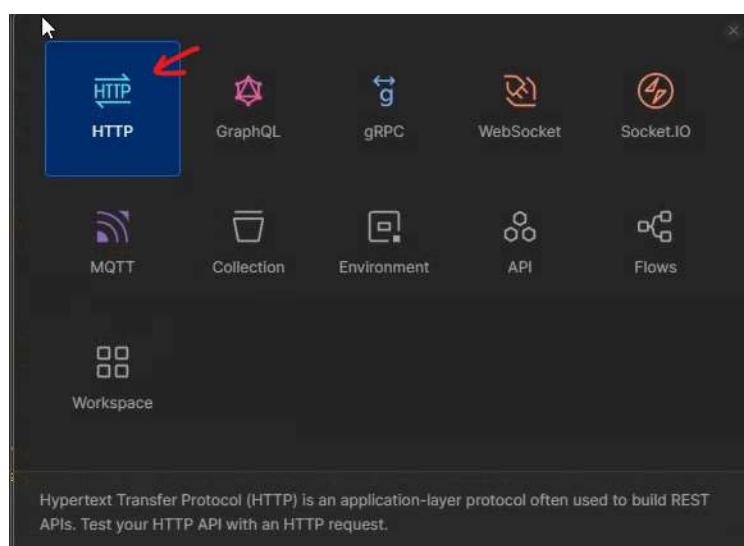
124. Entrez vos données d'authentification :



125. Créez une nouvelle connexion:



126. Sélectionnez (HTTP):



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

The screenshot shows the Postman interface. At the top, there's a header bar with the URL "http://localhost:8081/department". Below it, a search bar contains "http://localhost:8081/department" with a blue border around it. To the right of the search bar are "Save" and "Send" buttons. Underneath the search bar, there's a dropdown menu set to "GET" and a "Send" button. The main workspace is currently empty.

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

The screenshot shows the Postman interface after sending a GET request. The status bar at the bottom indicates "Status: 200 OK Time: 112 ms Size: 166 B". The "Body" tab is selected, showing an empty JSON object: {}.

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

The screenshot shows the Postman interface with a POST request. The "Body" tab is selected, and the "raw" and "JSON" buttons are highlighted. The JSON body is defined as follows:

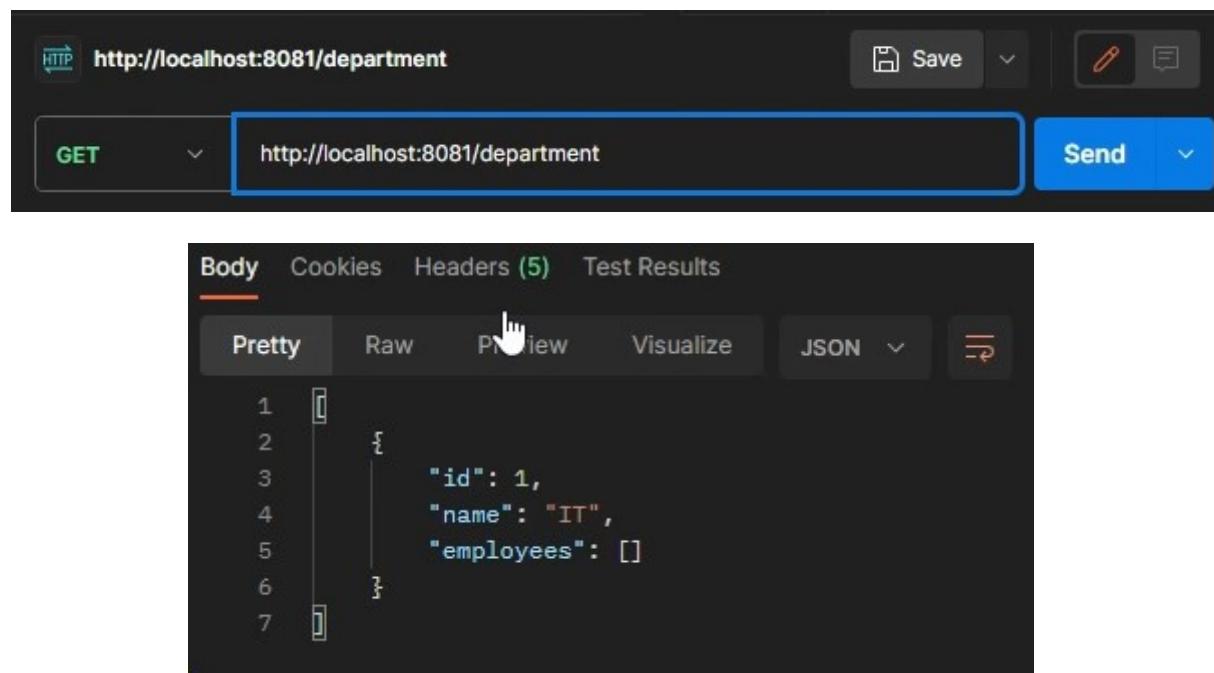
```
1 {
2 "id": "1",
3 "name": "IT"
4 }
```

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

The screenshot shows the Postman interface after sending a POST request. The "Body" tab is selected, and the "Pretty" tab is active, displaying the JSON response:

```
1 {
2 "id": 1,
3 "name": "IT",
4 "employees": []
5 }
```

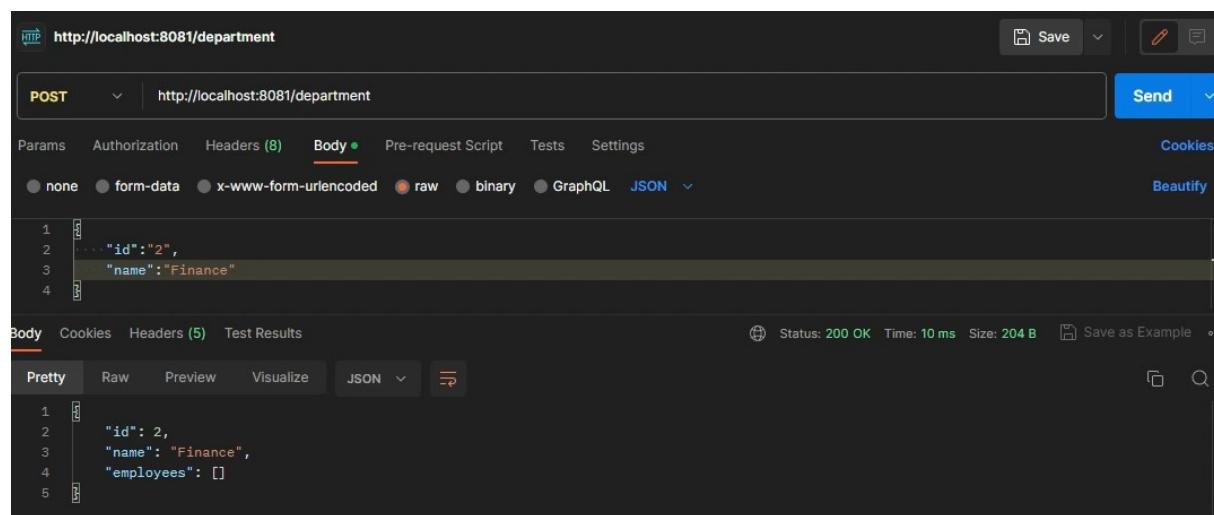
131. Testez la liste des départements par la méthode (GET) :



The screenshot shows the Postman interface. At the top, it says "HTTP" and "http://localhost:8081/department". Below that, there's a dropdown set to "GET" and a URL input field containing "http://localhost:8081/department", which is highlighted with a blue border. To the right of the URL is a "Send" button. The main area is titled "Body" and contains a JSON response:

```
1 []
2 {
3 "id": 1,
4 "name": "IT",
5 "employees": []
6 }
7 []
```

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

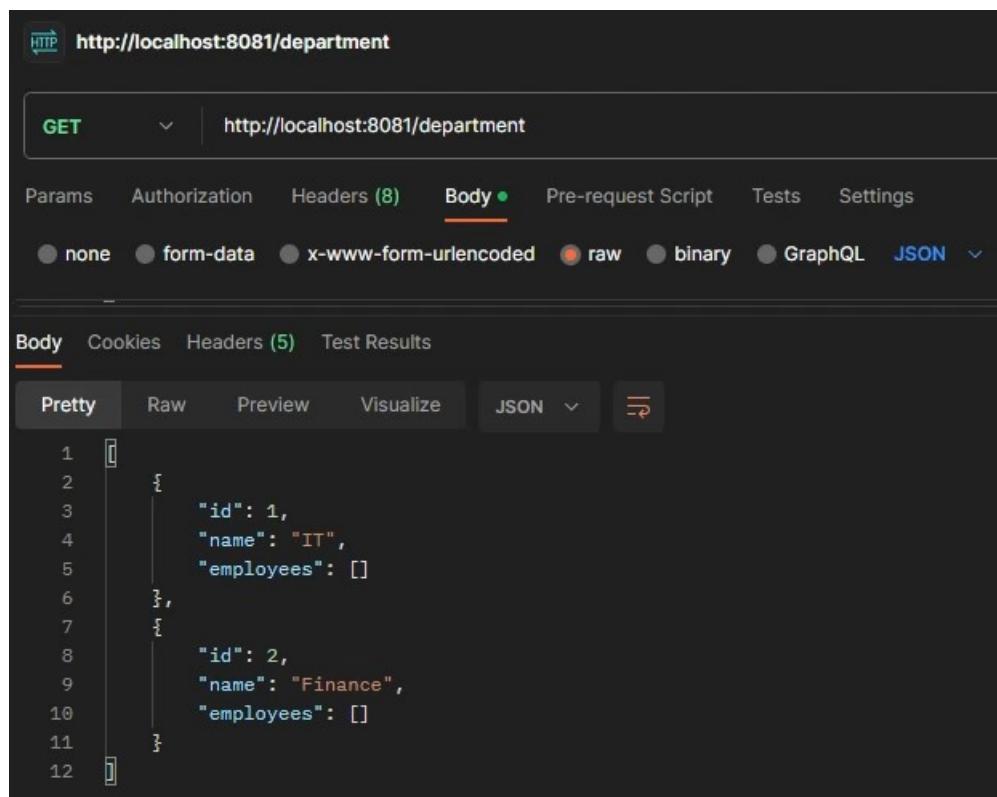


The screenshot shows the Postman interface. At the top, it says "HTTP" and "http://localhost:8081/department". Below that, there's a dropdown set to "POST" and a URL input field containing "http://localhost:8081/department". To the right of the URL is a "Send" button. The "Body" tab is selected, showing the following JSON payload:

```
1 { "id": 2,
2 "name": "Finance"
3 }
```

Below the body, the "Test Results" section shows the response details: Status: 200 OK, Time: 10 ms, Size: 204 B. There is also a "Save as Example" button.

133. Testez de nouveau la méthode (GET) :



The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `http://localhost:8081/department`. Below this, a navigation bar includes tabs for `GET`, `Params`, `Authorization`, `Headers (8)`, `Body` (which is currently selected), `Pre-request Script`, `Tests`, and `Settings`. Under the `Body` tab, there are several options: `none`, `form-data`, `x-www-form-urlencoded`, `raw` (selected), `binary`, `GraphQL`, and `JSON`. The `JSON` tab is also highlighted. Below the tabs, there are buttons for `Pretty`, `Raw`, `Preview`, and `Visualize`. The `Pretty` button is selected. The main content area displays the JSON response:

```
1 []
2 {
3 "id": 1,
4 "name": "IT",
5 "employees": []
6 },
7 {
8 "id": 2,
9 "name": "Finance",
10 "employees": []
11 }
12]
```

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

The screenshot shows the Postman interface with a GET request to `http://localhost:8081/department/2`. The Body tab is selected, showing a JSON response:

```

1
2 "id": 2,
3 "name": "Finance",
4 "employees": []
5

```

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

The screenshot shows the Zipkin UI at `localhost:9411/zipkin/`. A trace for `department-service: http post /department` is expanded, showing several spans. One span is selected, showing its details:

| Start Time                             | Spans | Duration |
|----------------------------------------|-------|----------|
| a few seconds ago (11/12 11:27:33.430) | 1     | 16.597ms |

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

The screenshot shows the Zipkin UI for the same trace, focusing on a specific span for the `http post /department` action. The annotations section shows:

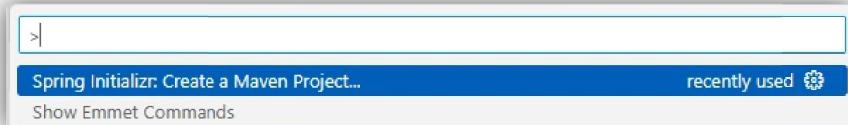
- Annotations** table:
 

|     |                                           |          |          |
|-----|-------------------------------------------|----------|----------|
| 0ms | 5.532ms                                   | 11.005ms | 16.597ms |
| 0ms | department-service: http post /department |          |          |
- Tags** table:
 

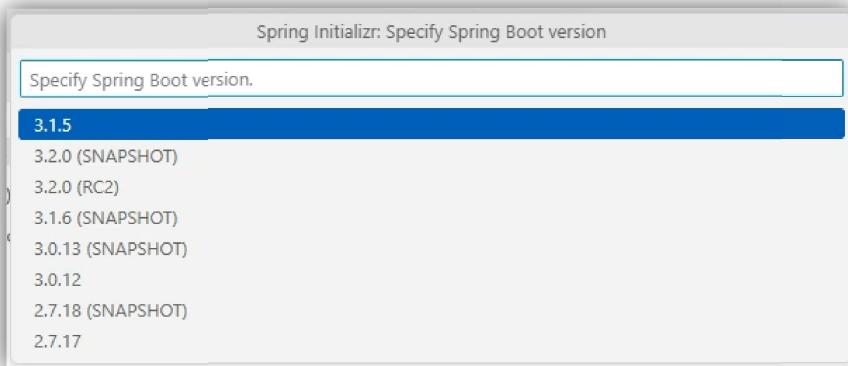
|           |             |
|-----------|-------------|
| exception | none        |
| http.url  | /department |
| method    | POST        |
| outcome   | SUCCESS     |
| status    | 200         |
| url       | /department |

## XI. Création du Microservice « Employee »

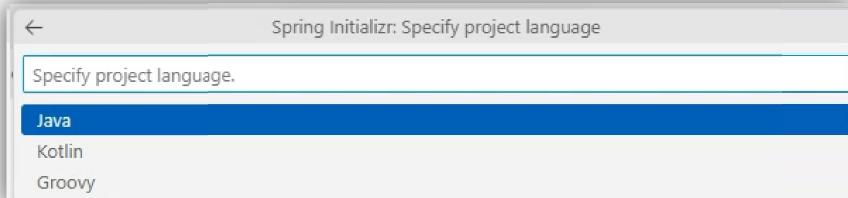
137. Appuyer sur **Ctrl + Shift + P** afin de lancer la palette des commandes.
138. Cliquez sur (**Spring Initializr : Create a Maven Project ...**):



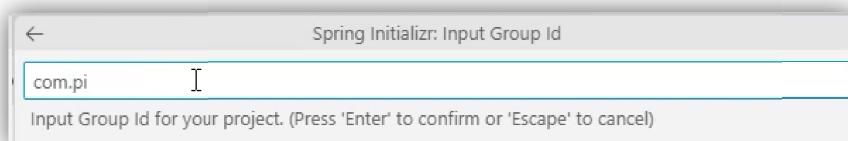
139. Sélectionnez la version (**3.1.5**)



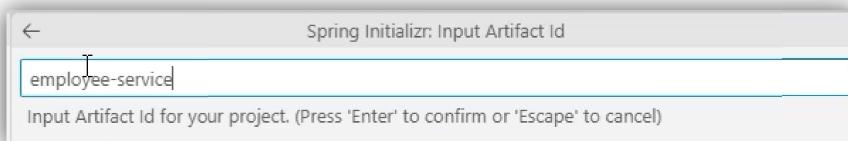
140. Sélectionnez le langage de programmation (**Java**)



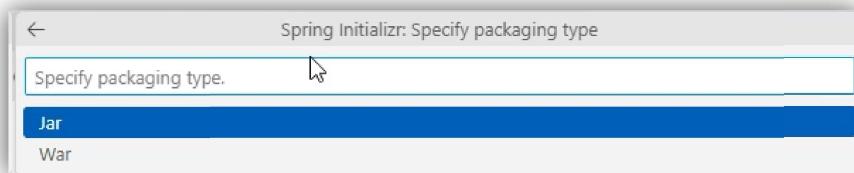
141. Choisir comme nom de package (**com.pi**)



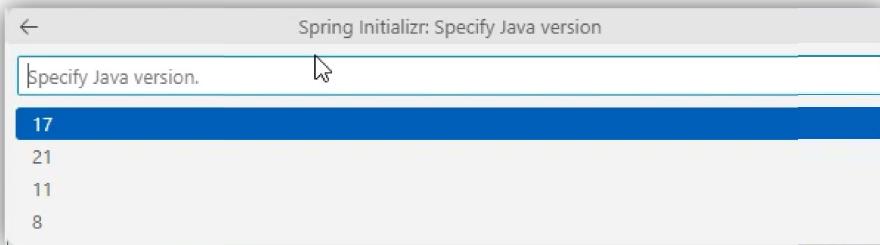
142. Indiquez le nom de projet (**employee-service**):



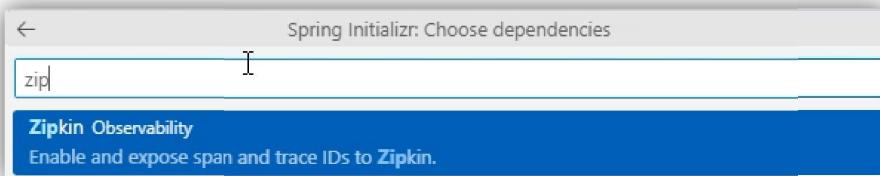
143. Choisir comme type de package (**Jar**)



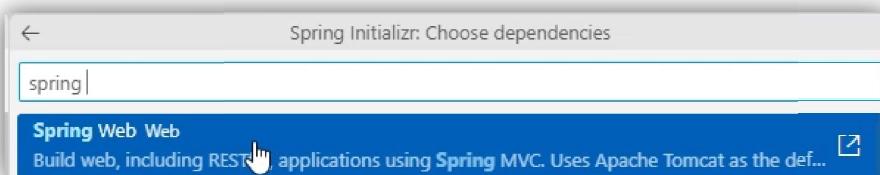
144. Choisir la version de Java Development Kit (**17**)



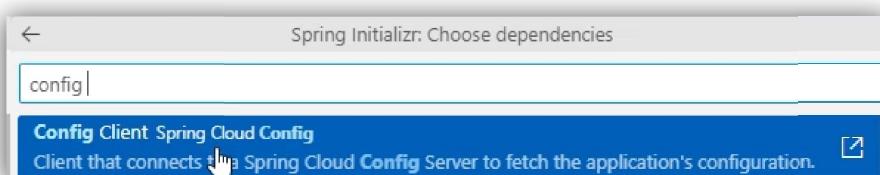
145. Ajoutez la dépendance (**Zipkin – Observability**)



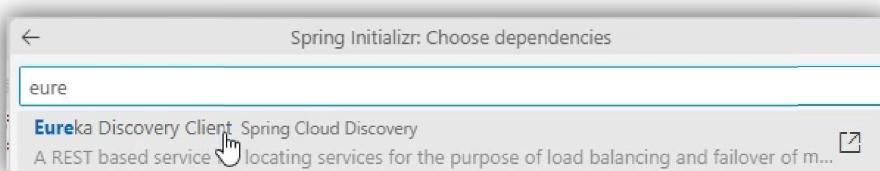
146. Ajoutez la dépendance (**Spring Web – Web**)



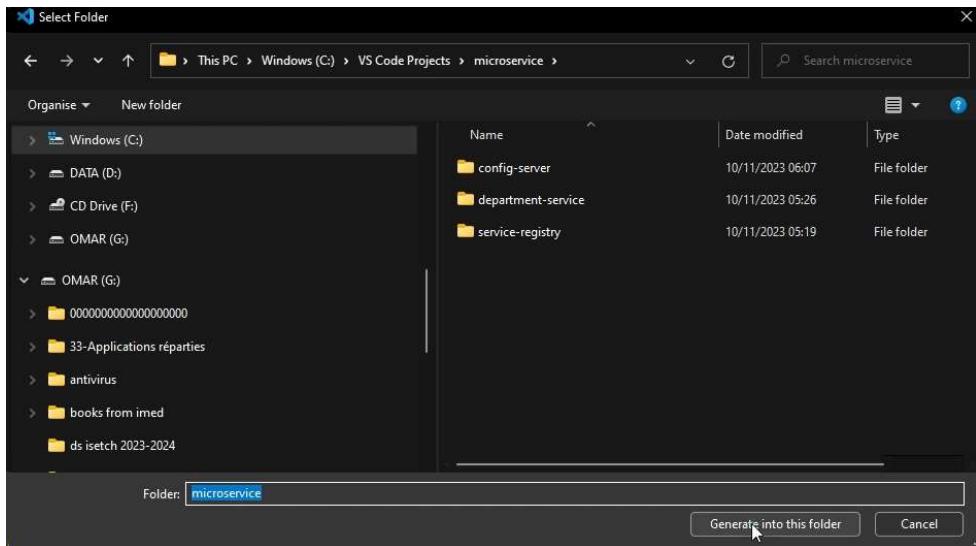
147. Ajoutez la dépendance (**Config Client – Spring Cloud Config**)



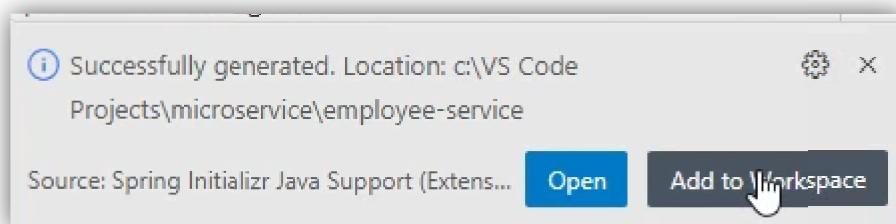
148. Ajoutez la dépendance (**Eureka Discovery Client – Spring Cloud Discovery**)



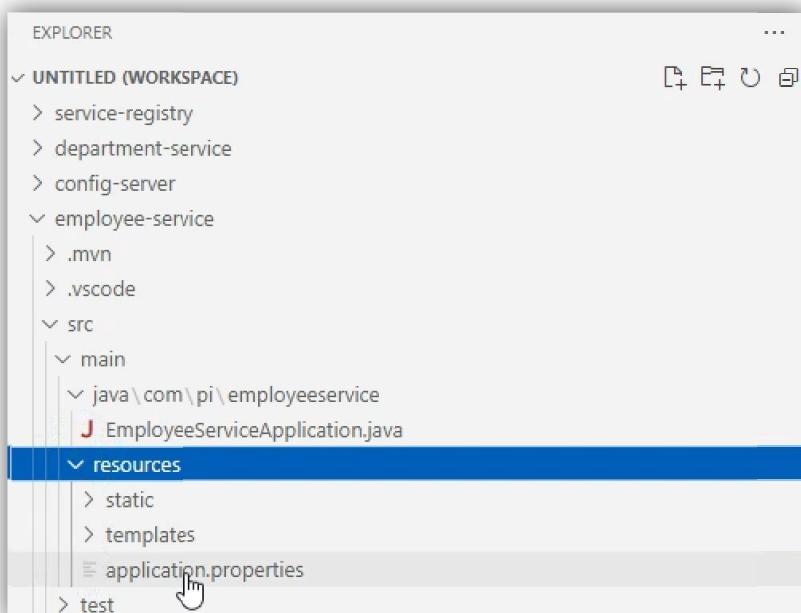
149. Créez un nouveau répertoire sous (**C:\VS Code Projects\microservice**) et appelez le (**employee-service**) puis cliquez sur (**Generate into this folder**)



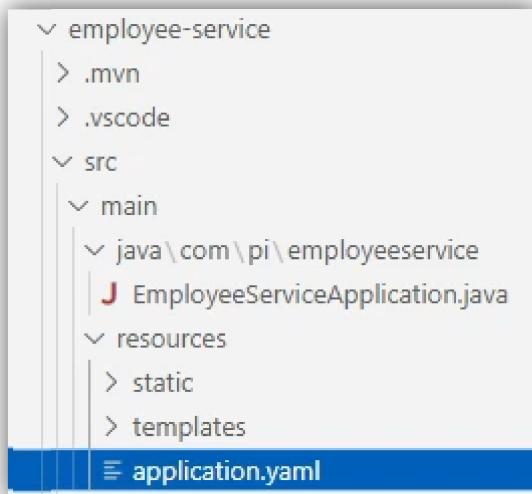
150. Ajouter ce dernier projet à l'espace de travail :



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



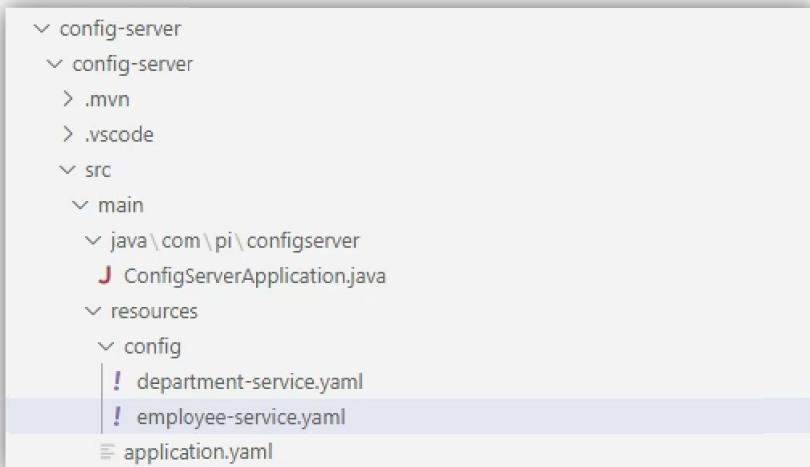
152. Et voici le résultat :



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

```
employee-service > src > main > resources > application.yaml
1 spring:
2 application:
3 | name: employee-service
4 | config:
5 | import: "optional:configserver:http://localhost:8088"
6
```

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
1 server:
2 | port: 8082
3
4 eureka:
5 | instance:
6 | | hostname: localhost
7 | client:
8 | serviceUrl:
9 | | defaultZone: http://${eureka.instance.hostname}:8761/eureka/
10 management:
11 tracing:
12 sampling:
13 | 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**)

```
J EmployeeServiceApplication.java ×
employee-service > src > main > java > com > pi > employeeservice > J EmployeeServiceApplication.java > ...
1 package com.pi.employeeservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6
7 @SpringBootApplication
8 @EnableDiscoveryClient
9 public class EmployeeServiceApplication {
10
11
12 public static void main(String[] args) {
13 SpringApplication.run(EmployeeServiceApplication.class, args);
14 }
15
16 }
17
```

## 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 :

```
employee-service > src > main > java > com > pi > employeeservice > controller > J EmployeeController.java
1 package com.pi.employeeservice.controller;
2
3 public class EmployeeController {
4
5 }
```

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

```
employee-service > src > main > java > com > pi > employeeservice > repository > J EmployeeRepository.java
1 package com.pi.employeeservice.repository;
2
3 public class EmployeeRepository {
4
5 }
```



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

```
employee-service > src > main > java > com > pi > employeeservice > repository > J EmployeeRepository.java
1 package com.pi.employeeservice.repository;
2
3 import org.springframework.stereotype.Repository;
4
5 @Repository
6 public class EmployeeRepository {
7
8 }
```

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

```
employee-service > src > main > java > com > pi > employeeservice > controller > EmployeeController.java
 1 package com.pi.employeeservice.controller;
 2
 3 import org.springframework.web.bind.annotation.RequestMapping;
 4 import org.springframework.web.bind.annotation.RestController;
 5
 6 @RestController
 7 @RequestMapping("/employee")
 8 public class EmployeeController {
 9 ...
10 }
```

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 :

```
employee-service > src > main > java > com > pi > employeeservice > model > Employee.java > {} com.pi.departmentservice.model
 1 package com.pi.departmentservice.model;
 2
 3 public record Employee(Long id,[Long departmentId, String name,int age,String position) {
 4
 5 }
```

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

```
employee-service > src > main > java > com > pi > employeeservice > model > Employee.java > {} com.pi.employeeservice.model
 1 package com.pi.employeeservice.model;
 2
 3 public record Employee(Long id, Long departmentId, String name,int age,String position) {
 4
 5 }
```

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
1 package com.pi.employeeservice.repository;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.springframework.stereotype.Repository;
7
8 import com.pi.employeeservice.model.Employee;
9
10 @Repository
11 public class EmployeeRepository {
12 private List<Employee> employees = new ArrayList<>();
13 public Employee addEmployee(Employee employee){
14 employees.add(employee);
15 return employee;
16 }
17 public Employee findById(Long id){
18 return employees.stream().filter(a -> a.id().equals(id))
19 .findFirst()
20 .orElseThrow();
21 }
22 public List<Employee> findAll(){
23 return employees;
24 }
25 public List<Employee> findByDepartment(Long departmentId){
26 return employees.stream()
27 .filter(a -> a.departmentId().equals(departmentId))
28 .toList();
29 }
30 }
```

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

```
J EmployeeController.java ●
employee-service > src > main > java > com > pi > employeeservice > controller > J EmployeeController.java > ...
1 package com.pi.employeeservice.controller;
2 import java.util.List;
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PathVariable;
8 import org.springframework.web.bind.annotation.PostMapping;
9 import org.springframework.web.bind.annotation.RequestBody;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RestController;
12 import com.pi.employeeservice.model.Employee;
13 import com.pi.employeeservice.repository.EmployeeRepository;
14
15 @RestController
16 @RequestMapping("/employee")
17 public class EmployeeController {
18 private static final Logger LOGGER = LoggerFactory.getLogger(EmployeeController.class);
19
20 @Autowired
21 private EmployeeRepository repository;
22 @PostMapping
23 public Employee add(@RequestBody Employee employee){
24 LOGGER.info(format:"Employee add:{}" ,employee);
25 return repository.addEmployee(employee);
26 }
27
28 @GetMapping
29 public List<Employee> findAll(){
30 LOGGER.info(msg:"Employee find");
31 return repository.findAll();
32 }
33 @GetMapping("/{id}")
34 public Employee findById(@PathVariable Long id){
35 LOGGER.info(format:"Employee find: id={} " ,id);
36 return repository.findById(id);
37 }
38
39 @GetMapping("/department/{departmentId}")
40 public List<Employee> findByDepartment(@PathVariable Long departmentId){
41 LOGGER.info(format:"Employee find: departmentId={} " ,departmentId);
42 return repository.findByDepartment(departmentId);
43 }
44 }
45 }
```

166. Arrêter toutes les applications:



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



**(1) service-Registry** : 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 :

The screenshot shows the Spring Eureka dashboard at localhost:8761. The top navigation bar includes links for Home, Documentation, and GitHub. The main content area is divided into sections: System Status, DS Replicas, and Instances currently registered with Eureka.

**System Status**

| Environment | test    | Current time             | 2023-11-12T15:29:46 +0100 |
|-------------|---------|--------------------------|---------------------------|
| Data center | default | Uptime                   | 00.00                     |
|             |         | Lease expiration enabled | false                     |
|             |         | Renews threshold         | 5                         |
|             |         | Renews (last min)        | 0                         |

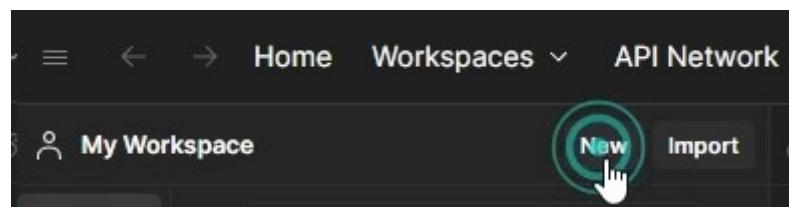
**DS Replicas**

Instances currently registered with Eureka

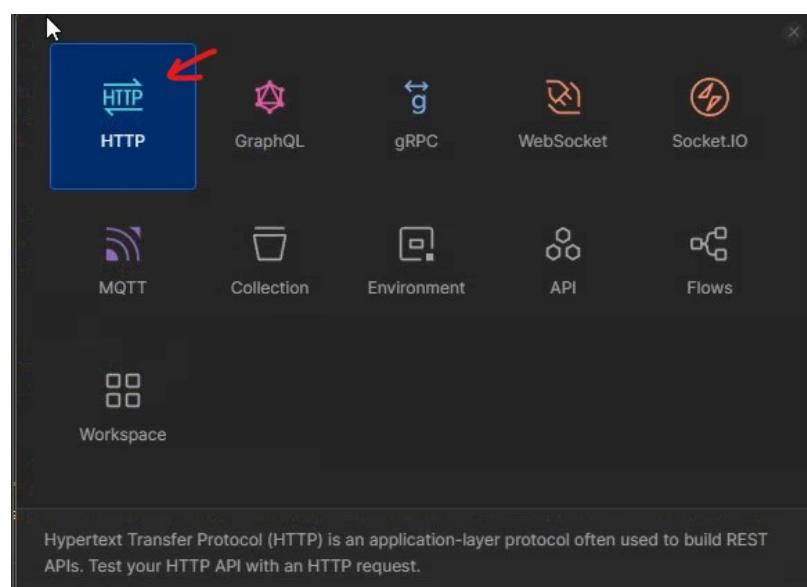
| Application        | AMIs    | Availability Zones | Status                                                                |
|--------------------|---------|--------------------|-----------------------------------------------------------------------|
| DEPARTMENT-SERVICE | n/a (1) | (1)                | UP (1) - <a href="#">host.docker.internal.department-service.8081</a> |
| EMPLOYEE-SERVICE   | n/a (1) | (1)                | UP (1) - <a href="#">host.docker.internal.employee-service.8082</a>   |

### XIII. Test du Microservice « Employee »

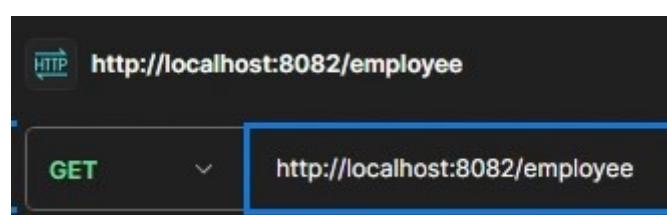
169. Créez une nouvelle connexion:



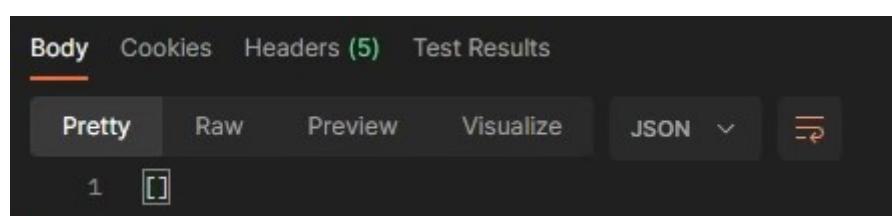
170. Sélectionnez (HTTP):



171. Lancez la requête suivante :



172. Voici le résultat :



173. Ajoutez un premier employé :

The screenshot shows the Postman interface with a POST request to `http://localhost:8082/employee`. The Body tab is selected, showing a JSON payload:

```
1 {
2 "id": 2,
3 "departmentId": 1,
4 "name": "karim",
5 "age": 40,
6 "position": "teacher"
7 }
```

174. Voici le résultat de la dernière requête :

The screenshot shows the Postman interface with the Body tab selected, displaying the response body in Pretty format:

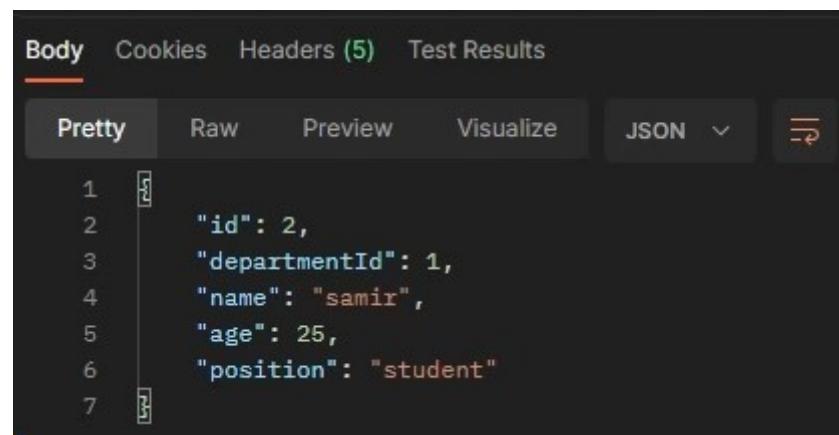
```
1 {
2 "id": 1,
3 "departmentId": 1,
4 "name": "karim",
5 "age": 40,
6 "position": "teacher"
7 }
```

175. Ajoutez un deuxième employé :

The screenshot shows the Postman interface with a POST request to `http://localhost:8082/employee`. The Body tab is selected, showing a JSON payload:

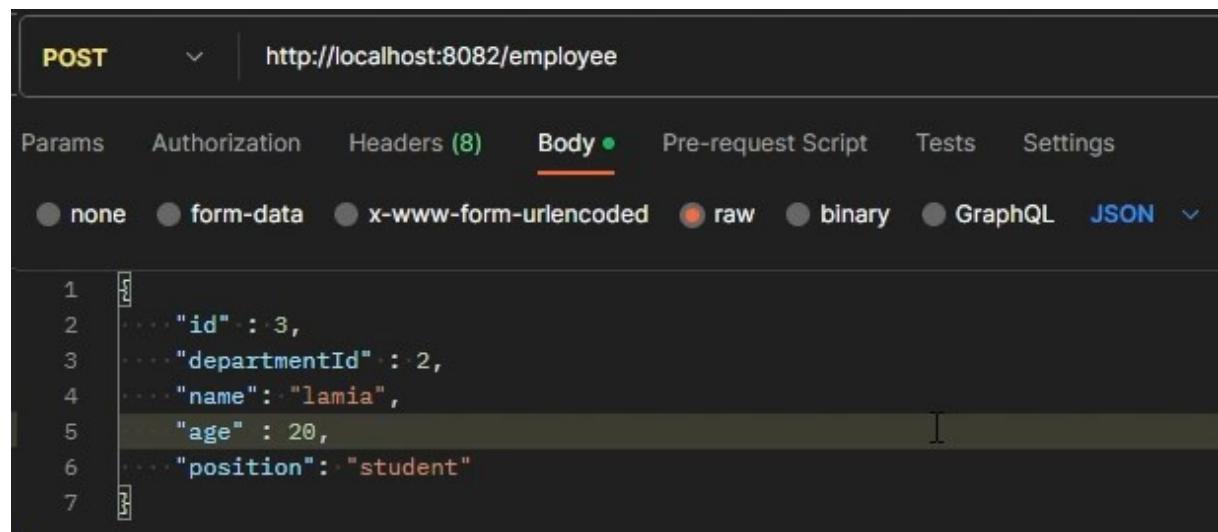
```
1 {
2 "id": 3,
3 "departmentId": 1,
4 "name": "samir",
5 "age": 25,
6 "position": "student"
7 }
```

176. Voici le résultat de la dernière requête :



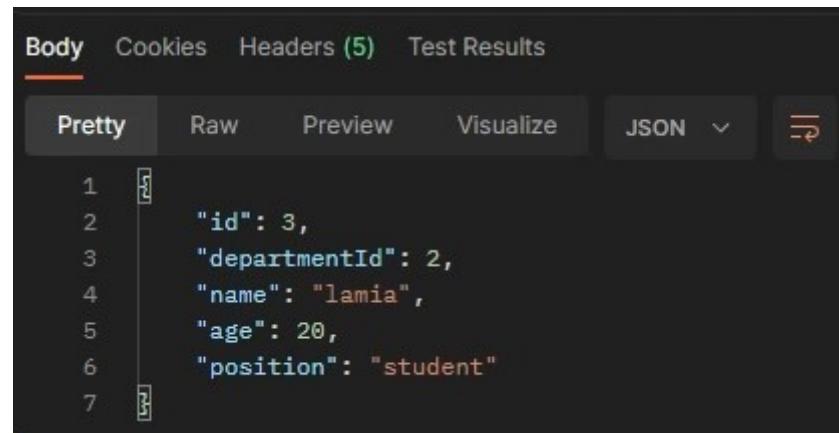
The screenshot shows the Postman interface with the 'Body' tab selected. The response body is displayed in 'Pretty' format:

```
1 {
2 "id": 2,
3 "departmentId": 1,
4 "name": "samir",
5 "age": 25,
6 "position": "student"
7 }
```



The screenshot shows the Postman interface with the 'POST' method selected and the URL 'http://localhost:8082/employee'. The 'Body' tab is selected, and the 'form-data' option is chosen. The request body is displayed in 'Pretty' format:

```
1 {
2 "id": 3,
3 "departmentId": 2,
4 "name": "lamia",
5 "age": 20,
6 "position": "student"
7 }
```



The screenshot shows the Postman interface with the 'Body' tab selected. The response body is displayed in 'Pretty' format:

```
1 {
2 "id": 3,
3 "departmentId": 2,
4 "name": "lamia",
5 "age": 20,
6 "position": "student"
7 }
```

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

```
1 {
2 "id": 4,
3 "departmentId": 2,
4 "name": "zeyneb",
5 "age": 20,
6 "position": "student"
7 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2 "id": 4,
3 "departmentId": 2,
4 "name": "zeyneb",
5 "age": 20,
6 "position": "student"
7 }
```

Lister les employés enregistrés :

The screenshot shows the Postman interface with a GET request to `http://localhost:8082/employee`. The Body tab displays the following JSON response:

```
1 [{ 2 "id": 1, 3 "departmentId": 1, 4 "name": "karim", 5 "age": 40, 6 "position": "teacher" 7 }, 8 { 9 "id": 2, 10 "departmentId": 1, 11 "name": "samir", 12 "age": 25 }]
```

Lister l'employée de id(1) :

The screenshot shows the Postman interface with a GET request to `http://localhost:8082/employee/1`. The Body tab displays the following JSON response:

```
1 [{ 2 "id": 1, 3 "departmentId": 1, 4 "name": "karim", 5 "age": 40, 6 "position": "teacher" 7 }]
```

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

```
@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/1

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 {  
2 "id": 4,  
3 "departmentId": 2,  
4 "name": "zeyneb",  
5 "age": 20,  
6 "position": "student"  
7 }

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 [  
2 {  
3 "id": 1,  
4 "departmentId": 1,  
5 "name": "karim",  
6 "age": 40,  
7 "position": "teacher"  
8 },  
9 {  
10 "id": 2,  
11 "departmentId": 1,  
12 "name": "samir",  
13 "age": 25,  
14 "position": "student"  
15 }  
16 ]

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

```
1 []
2 "id": 4,
3 "departmentId": 2,
4 "name": "zeyneb",
5 "age": 20,
6 "position": "student"
7]
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 []
2 {
3 "id": 3,
4 "departmentId": 2,
5 "name": "lamia",
6 "age": 20,
7 "position": "student"
8 },
9 {
10 "id": 4,
11 "departmentId": 2,
12 "name": "zeyneb",
13 "age": 20,
14 "position": "student"
15 }
16]
```

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

```
1 {
2 "id": 4,
3 "departmentId": 2,
4 "name": "zeyneb",
5 "age": 20,
6 "position": "student"
7 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2 {
3 "id": 1,
4 "departmentId": 1,
5 "name": "karim",
6 "age": 40,
7 "position": "teacher"
8 },
9 {
10 "id": 2,
11 "departmentId": 1,
12 "name": "samir",
13 "age": 25,
14 "position": "student"
15 },
16 {
17 "id": 3,
18 "departmentId": 2,
19 "name": "lamia",
20 "age": 20,
21 "position": "student"
22 },
23 {
24 "id": 4,
25 "departmentId": 2,
26 "name": "zeyneb",
27 "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)**

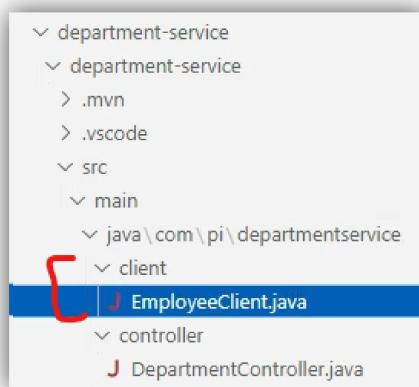
- ^ employee-service: http post /employee

**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
 1 package com.pi.departmentservice.client;
 2
 3 import java.util.List;
 4 import org.springframework.web.bind.annotation.PathVariable;
 5 import org.springframework.web.service.annotation.GetExchange;
 6 import org.springframework.web.service.annotation.HttpExchange;
 7 import com.pi.departmentservice.model.Employee;
 8
 9 @HttpExchange
 10 public interface EmployeeClient {
 11 @GetExchange("/employee/department/{departmentId}")
 12 public List<Employee> findByDepartment(@PathVariable Long departmentId);
 13 }
 14
```

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**) :

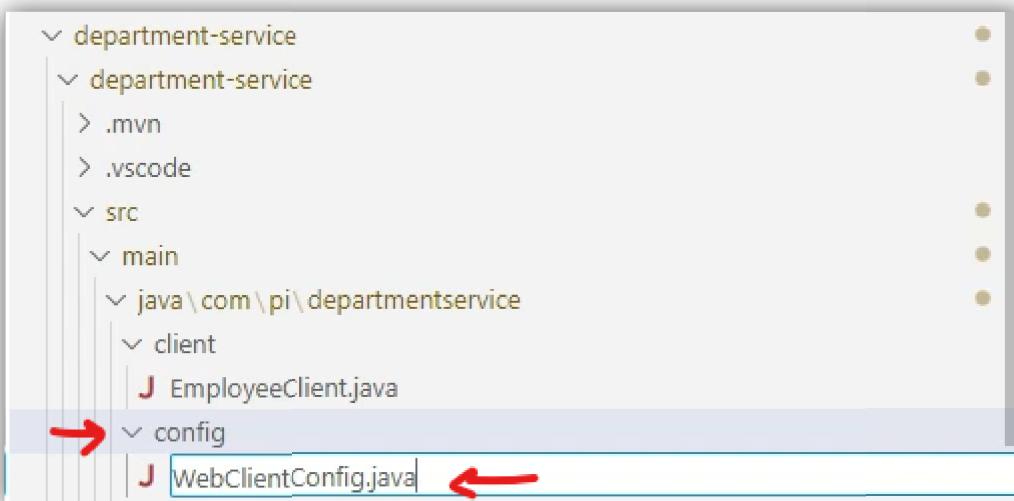
```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

181. Relancez le microservice (department-service) :

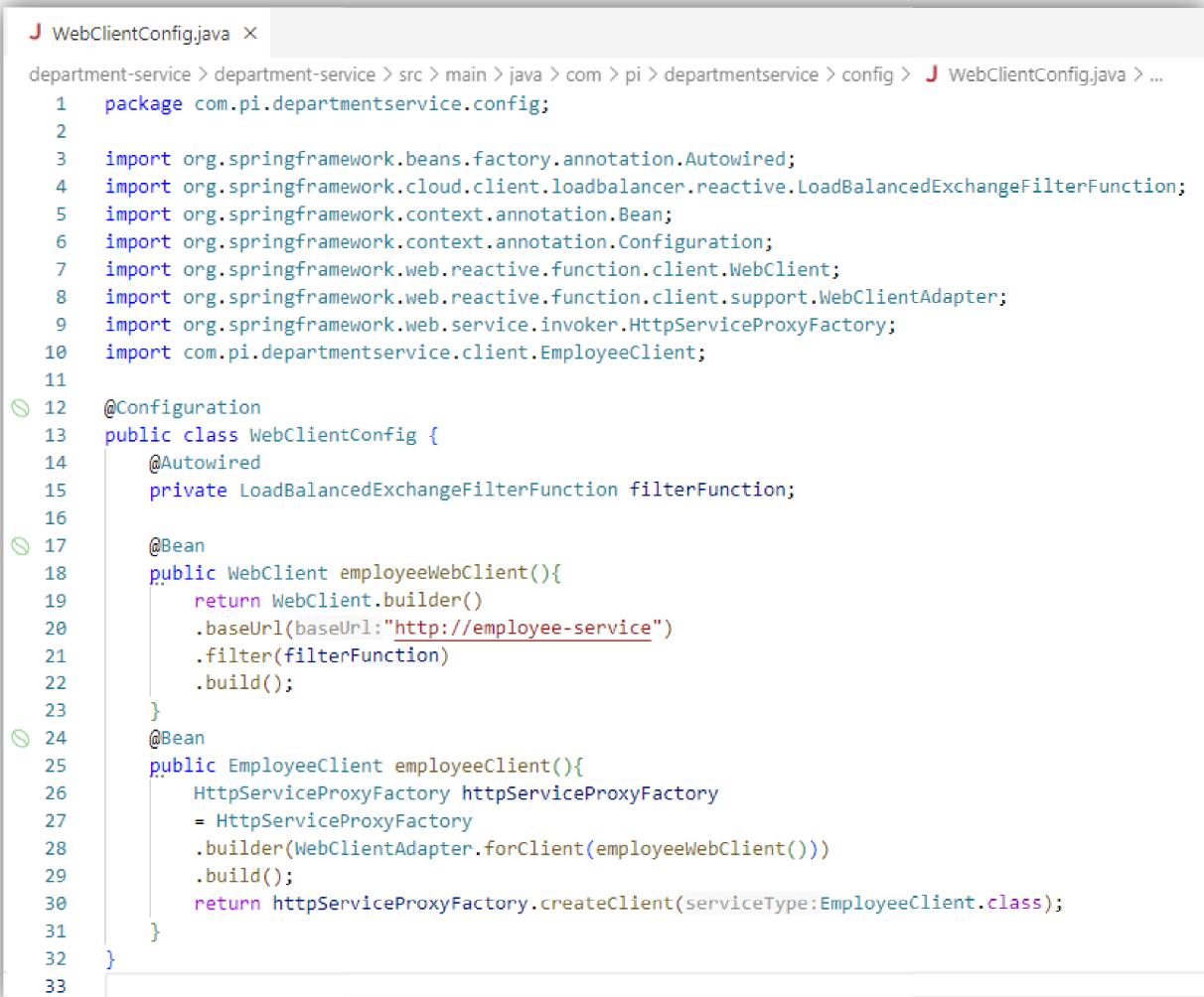


```
> department-service > department-service > src > main > java > com > pi > departmentservice > controller > J DepartmentController.java
❷ 16 @RestController
 17 @RequestMapping("/department")
 18 public class DepartmentController {
 19 private static final Logger LOGGER = LoggerFactory.getLogger(clazz:DepartmentController.class);
 20 @Autowired
 21 private DepartmentRepository repository;
 22 @Autowired
 23 private EmployeeClient employeeClient;
```

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 :



```
J WebClientConfig.java ×
department-service > department-service > src > main > java > com > pi > departmentservice > config > J WebClientConfig.java > ...
1 package com.pi.departmentservice.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.cloud.client.loadbalancer.reactive.LoadBalancedExchangeFilterFunction;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7 import org.springframework.web.reactive.function.client.WebClient;
8 import org.springframework.web.reactive.function.client.support.WebClientAdapter;
9 import org.springframework.web.service.invoker.HttpServiceProxyFactory;
10 import com.pi.departmentservice.client.EmployeeClient;
11
12 @Configuration
13 public class WebClientConfig {
14 @Autowired
15 private LoadBalancedExchangeFilterFunction filterFunction;
16
17 @Bean
18 public WebClient employeeWebClient(){
19 return WebClient.builder()
20 .baseUrl(baseUrl:"http://employee-service")
21 .filter(filterFunction)
22 .build();
23 }
24
25 @Bean
26 public EmployeeClient employeeClient(){
27 HttpServiceProxyFactory httpServiceProxyFactory
28 = HttpServiceProxyFactory
29 .builder(WebClientAdapter.forClient(employeeWebClient()))
30 .build();
31 return httpServiceProxyFactory.createClient(serviceType:EmployeeClient.class);
32 }
33 }
```

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 > J DepartmentController.java > Language Support for Java(...
34 @GetMapping("/{id}")
35 public Department findById(@PathVariable Long id){
36 LOGGER.info(format:"Department find: id={}",id);
37 return repository.findById(id);
38 }
39 @GetMapping("/with-employees")
40 public List<Department> findAllWithEmployees(){
41 LOGGER.info(msg:"Department find");
42 List<Department> departments
43 = repository.findAll();
44 departments.forEach(department -> department.setEmployees(employeeClient.findById(department.getId())));
45 return departments;
46 }
47 }
```

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 :

```
POST http://localhost:8081/department

Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 { "id": "1",
2 "name": "IT"
3 }
4
```

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

```
HTTP http://localhost:8081/department
POST http://localhost:8081/department
Save Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify
none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 { "id": "2",
2 "name": "Finance"
3 }
4
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 10 ms Size: 204 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 { "id": 2,
2 "name": "Finance",
3 "employees": []
4 }
```

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

GET http://localhost:8081/department/with-employees

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 []
2 {
3 "id": 1,
4 "name": "IT",
5 "employees": [
6 {
7 "id": 1,
8 "departmentId": 1,
9 "name": "karim",
10 "age": 40,
11 "position": "teacher"
12 },
13 {
14 "id": 2,
15 "departmentId": 1,
16 "name": "samir",
17 "age": 25,
18 "position": "student"
19 }
20],
21 }
```

191. Voici le résultat sous (ZIPKIN) :

localhost:9411/zipkin/?lookback=15m&endTs=1699803719774&limit=10

**Zipkin** Find a trace Dependencies

+ 10 Results

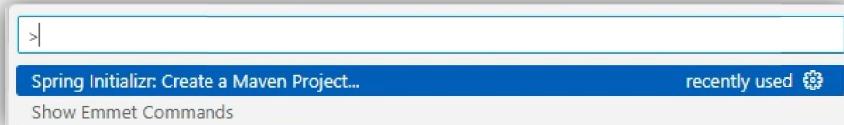
Root

department-service: http get /department/with-employees

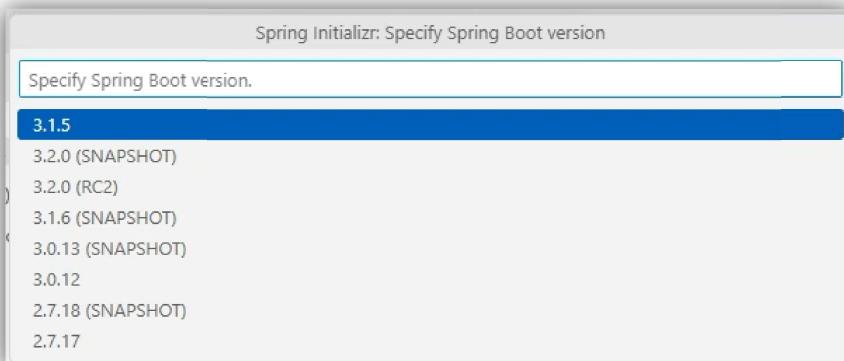
department-service (1)

## XV. Création du passerelle API

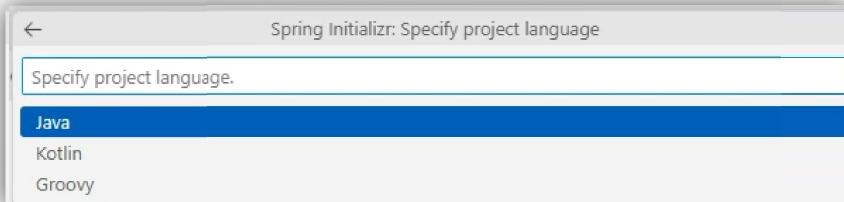
192. Appuyer **sur Ctrl + Shift + P** afin de lancer la palette des commandes.
193. Cliquez sur (**Spring Initializr : Create a Maven Project ...**):



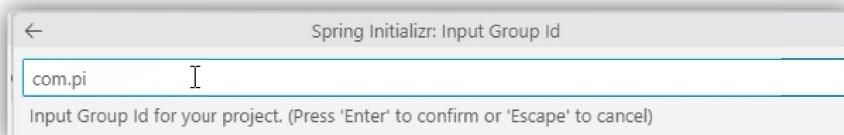
194. Sélectionnez la version (**3.1.5**)



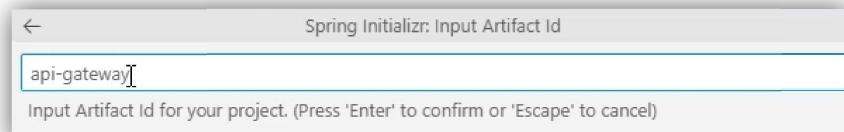
195. Sélectionnez le langage de programmation (**Java**)



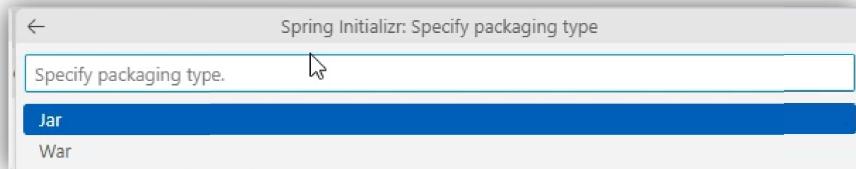
196. Choisir comme nom de package (**com.pi**)



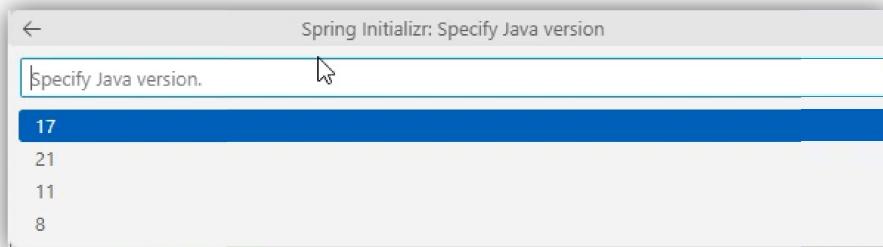
197. Indiquez le nom de projet (**api-gateway**):



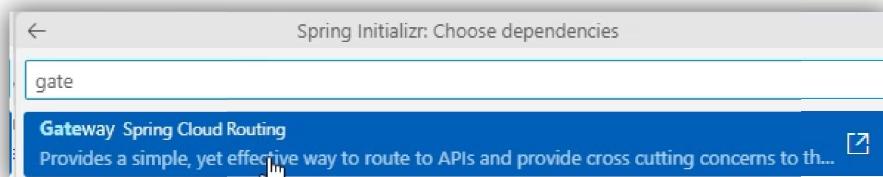
198. Choisir comme type de package (**Jar**)



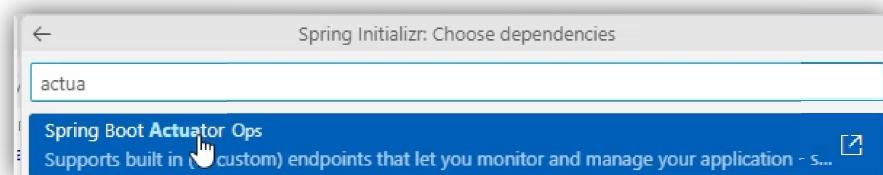
199. Choisir la version de Java Development Kit (17)



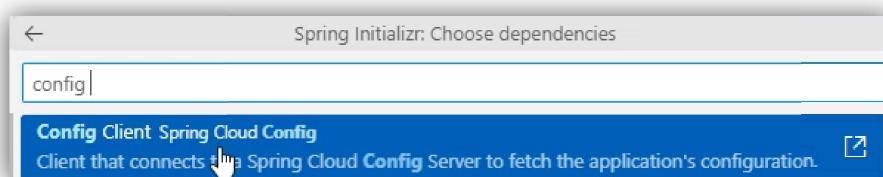
200. Ajoutez la dépendance (**Gateway – Spring Cloud Routing**)



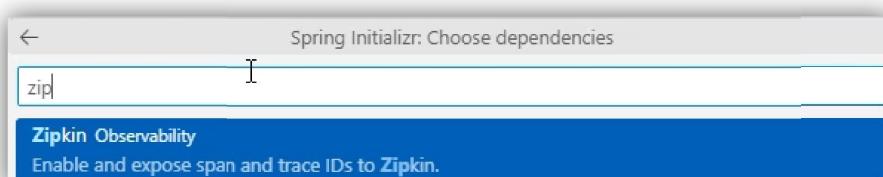
201. Ajoutez la dépendance (**Spring Boot Actuator – Ops**)



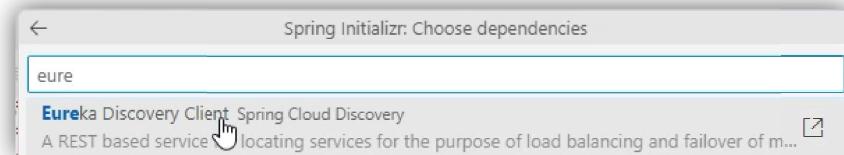
202. Ajoutez la dépendance (**Config Client – Spring Cloud Config**)



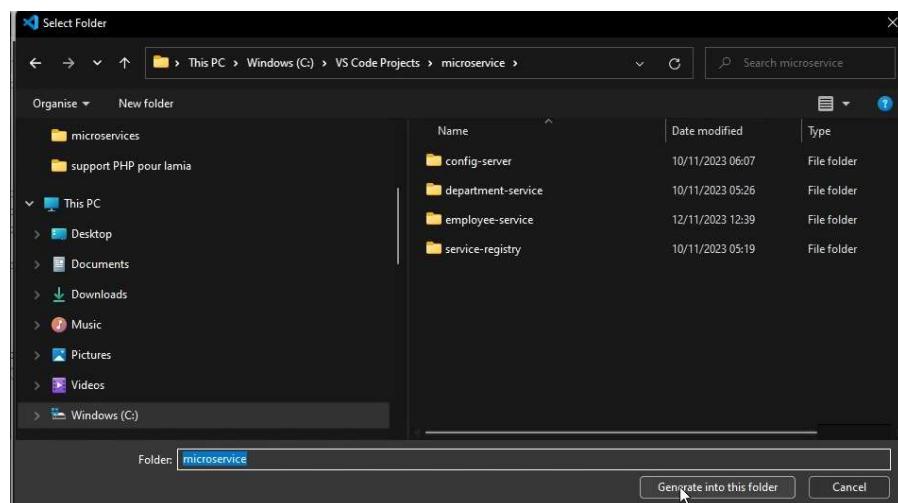
203. Ajoutez la dépendance (**Zipkin – Observability**)



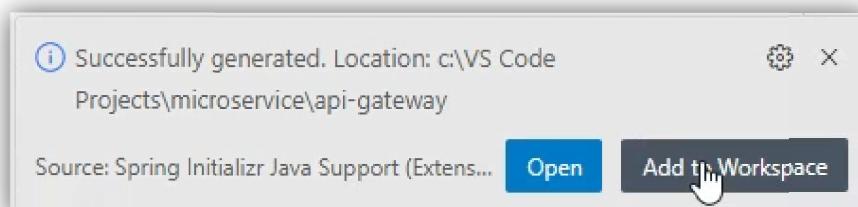
204. Ajoutez la dépendance (**Eureka Discovery Client – Spring Cloud Discovery**)



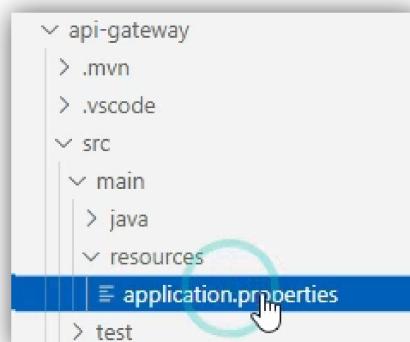
205. Créez un nouveau répertoire sous (**C:\VS Code Projects\microservice**) et appelez le (**employee-service**) puis cliquez sur (**Generate into this folder**)



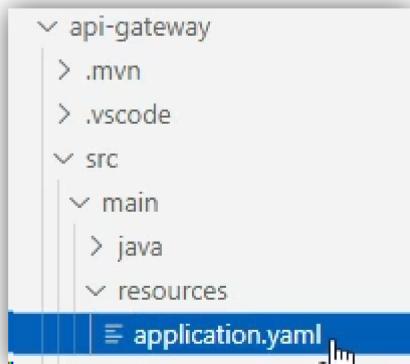
206. Ajouter ce dernier projet à l'espace de travail :



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
1 server:
2 port: 8060
3
4 eureka:
5 client:
6 serviceUrl:
7 defaultZone: http://localhost:8761/eureka/
8
9 management:
10 tracing:
11 sampling:
12 probability: 1.0
13
14 spring:
15 application:
16 name: api-gateway
17 config:
18 import: "optional:configserver:http://localhost:8088"
19 cloud:
20 gateway:
21 routes:
22 - id: employee-service
23 uri: lb://employee-service
24 predicates:
25 - Path=/employee/**
26 - id: department-service
27 uri: lb://department-service
28 predicates:
29 - 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**)



```
1 package com.pi.apigateway;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6
7 @SpringBootApplication
8 @EnableDiscoveryClient
9 public class ApiGatewayApplication {
10
11 public static void main(String[] args) {
12 SpringApplication.run(ApiGatewayApplication.class, args);
13 }
14 }
15 }
```

211. Arrêter toutes les applications:



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



(1) **service-Registry** : 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 :

The screenshot shows the Spring Eureka dashboard at localhost:8761. At the top, there's a header with the Spring logo and 'Eureka'. Below it, a 'System Status' section displays environment information: Environment is 'test', Data center is 'default', Current time shows uptime, lease expiration enabled, renew threshold, and renew statistics. Under 'DS Replicas', a table lists three registered instances: API-GATEWAY, DEPARTMENT-SERVICE, and EMPLOYEE-SERVICE, each with its status (UP) and corresponding port (8060, 8081, 8082).

## 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 :

The screenshot shows a Postman request configuration. The method is set to 'POST' and the URL is 'http://localhost:8081/department'. The 'Body' tab is selected, and the 'raw' option under 'JSON' is chosen. The request body contains the following JSON payload:

```
1 {
2 "id": "1",
3 "name": "IT"
4 }
```

216. Cliquez sur (**SEND**)

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

HTTP <http://localhost:8081/department>

POST <http://localhost:8081/department>

Params Authorization Headers (8) Body **JSON** Pre-request Script Tests Settings Cookies Beautify

Body

```
1 id
2 "id": "2",
3 "name": "Finance"
4
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 10 ms Size: 204 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2 "id": 2,
3 "name": "Finance",
4 "employees": []
5 }
```

218. Ajoutez un nouvel employé :

HTTP <http://localhost:8082/employee>

POST <http://localhost:8082/employee>

Params Authorization Headers (8) Body **JSON** Pre-request Script Tests Settings

Body

```
1 {
2 "id": 2,
3 "departmentId": 1,
4 "name": "karim",
5 "age": 40,
6 "position": "teacher"
7 }
```

219. Ajoutez un deuxième employé :

HTTP <http://localhost:8082/employee>

POST <http://localhost:8082/employee>

Params Authorization Headers (8) Body **JSON** Pre-request Script Tests Settings

Body

```
1 {
2 "id": 3,
3 "departmentId": 1,
4 "name": "samir",
5 "age": 25,
6 "position": "student"
7 }
```

220. Ajoutez un troisième employé :

The screenshot shows a Postman interface with a POST request to `http://localhost:8082/employee`. The 'Body' tab is selected, showing a JSON payload:

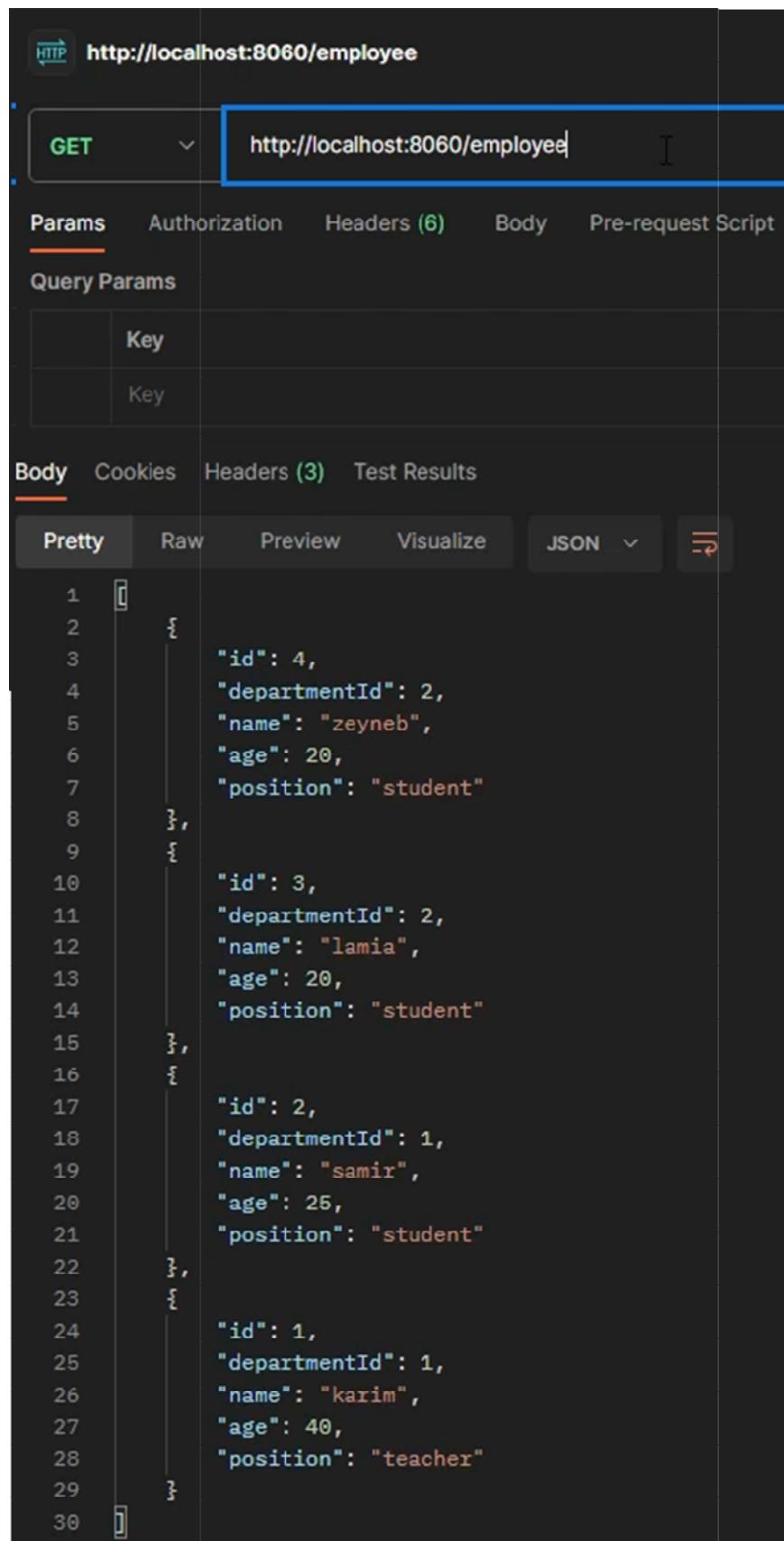
```
1 {
2 "id": 3,
3 "departmentId": 2,
4 "name": "lamia",
5 "age": 20,
6 "position": "student"
7 }
```

221. Ajoutez un quatrième employé :

The screenshot shows a Postman interface with a POST request to `http://localhost:8082/employee`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2 "id": 4,
3 "departmentId": 2,
4 "name": "zeyneb",
5 "age": 20,
6 "position": "student"
7 }
```

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



The screenshot shows a Postman request configuration for a GET request to `http://localhost:8060/employee`. The response body is displayed in a pretty-printed JSON format, showing four employee records:

```
1 [{ 2 "id": 4, 3 "departmentId": 2, 4 "name": "zeyneb", 5 "age": 20, 6 "position": "student" 7 }, 8 { 9 "id": 3, 10 "departmentId": 2, 11 "name": "lamia", 12 "age": 20, 13 "position": "student" 14 }, 15 { 16 "id": 2, 17 "departmentId": 1, 18 "name": "samir", 19 "age": 25, 20 "position": "student" 21 }, 22 { 23 "id": 1, 24 "departmentId": 1, 25 "name": "karim", 26 "age": 40, 27 "position": "teacher" 28 }, 29] 30 }
```

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

HTTP <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 ↻

```
1 []
2 {
3 "id": 1,
4 "name": "IT",
5 "employees": [
6 {
7 "id": 2,
8 "departmentId": 1,
9 "name": "samir",
10 "age": 25,
11 "position": "student"
12 },
13 {
14 "id": 1,
15 "departmentId": 1,
16 "name": "karim",
17 "age": 40,
18 "position": "teacher"
19 }
20],
21 },
22 {
23 "id": 2,
24 "name": "CS",
25 "employees": [
26 {
27 "id": 4,
28 "departmentId": 2,
29 "name": "zeyneb",
30 "age": 20,
31 "position": "student"
32 }
33]
34 }
```