**Micro service Architecture:**

**Naming Registry:** We use Eureka as naming registry where all our services (running in different ports) will be registered here.

**Steps:**

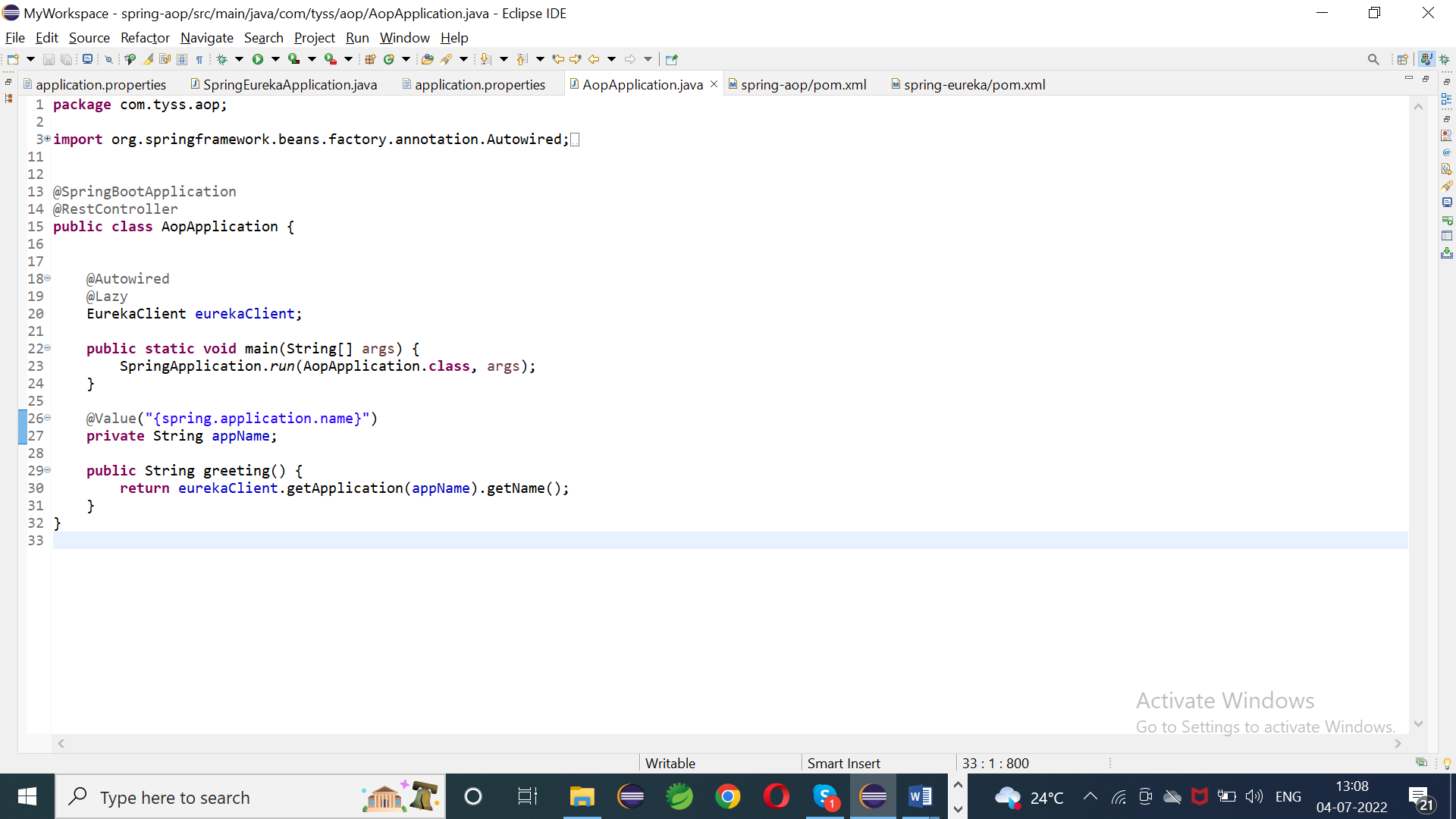
1. Add server side dependency to eureka server application and client side dependency to our other services.
2. Add **@EnableEurekaServer** on server side application. Adding ***@EnableDiscoveryClient*or*@EnableEurekaClient***on client side is optional.
3. server.port= 8761

eureka.client.registerWithEureka = false

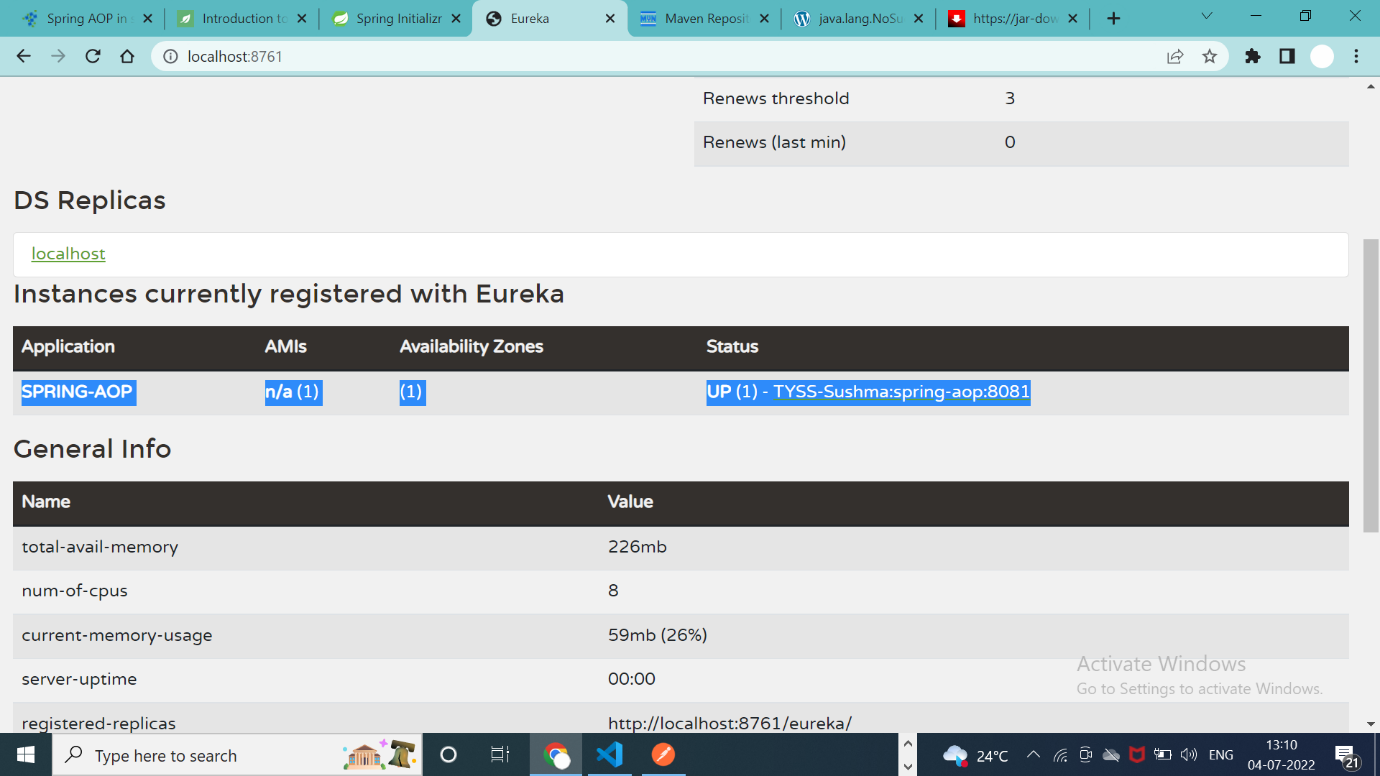
eureka.client.fetchRegistry = false

spring.application.name=spring-eureka

1. Here we're configuring an application port; the default one for ***Eureka***servers is ***8761***. We're telling the built-in ***Eureka******Client*** not to register with itself because our application should be acting as a server.
2. We need to add a piece of code in Client Side application. PFA



1. When the client side application runs, it will be registered to Eureka. We can see all registered services in Eureka dashboard. We can access the dashboard over [***http://localhost:8761/***](http://localhost:8761/)**PFA.**

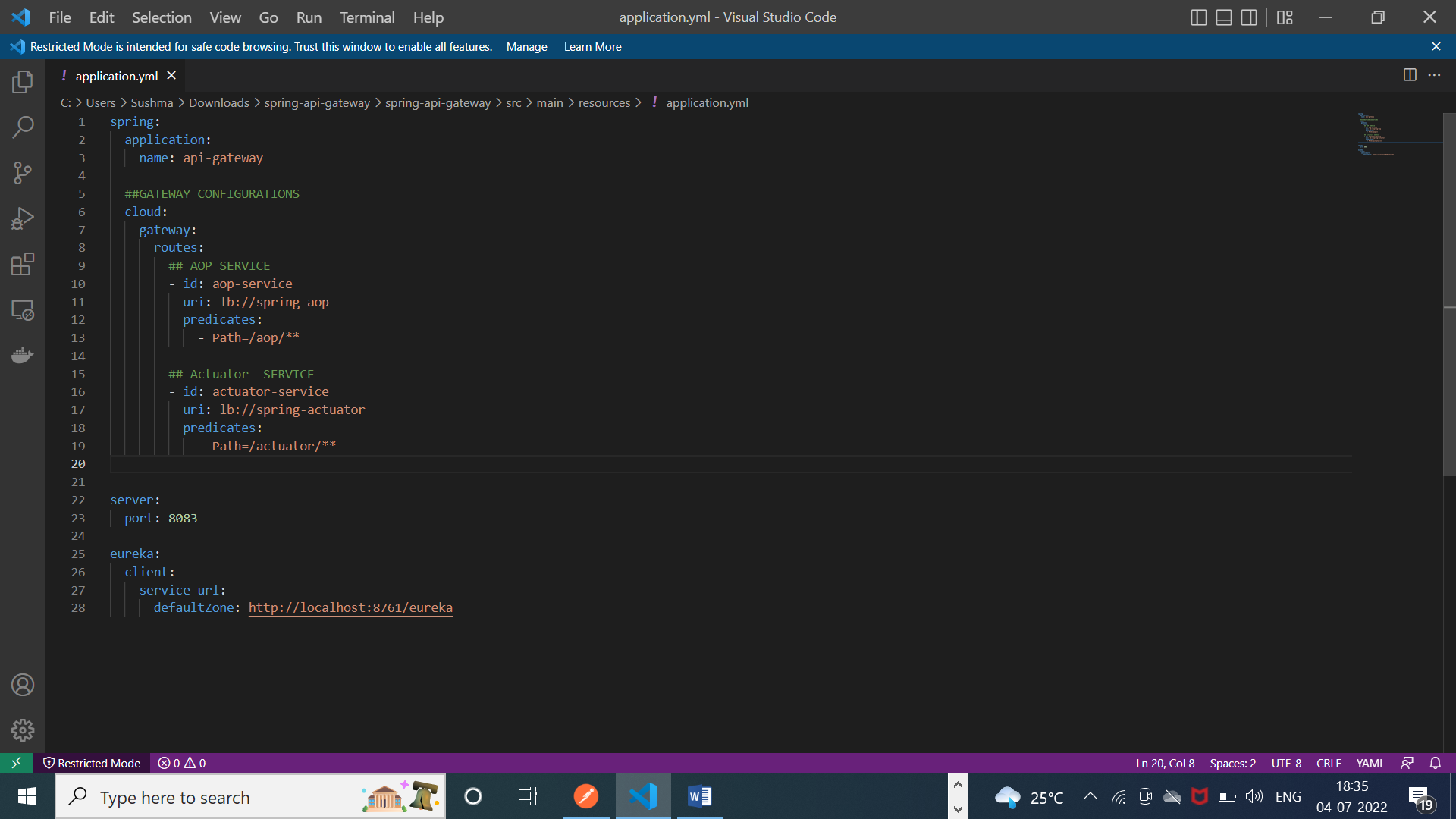
******

**API gateway:**

In a distributed environment, services need to communicate with each other. However, this is inter-service communication. We also have use-cases where a client outside our domain wants to hit our services for the API. So, either we can expose the address of all our micro services which can be called by clients OR we can create a Service Gateway which routes the request to various micro services and responds to the clients.

Steps:

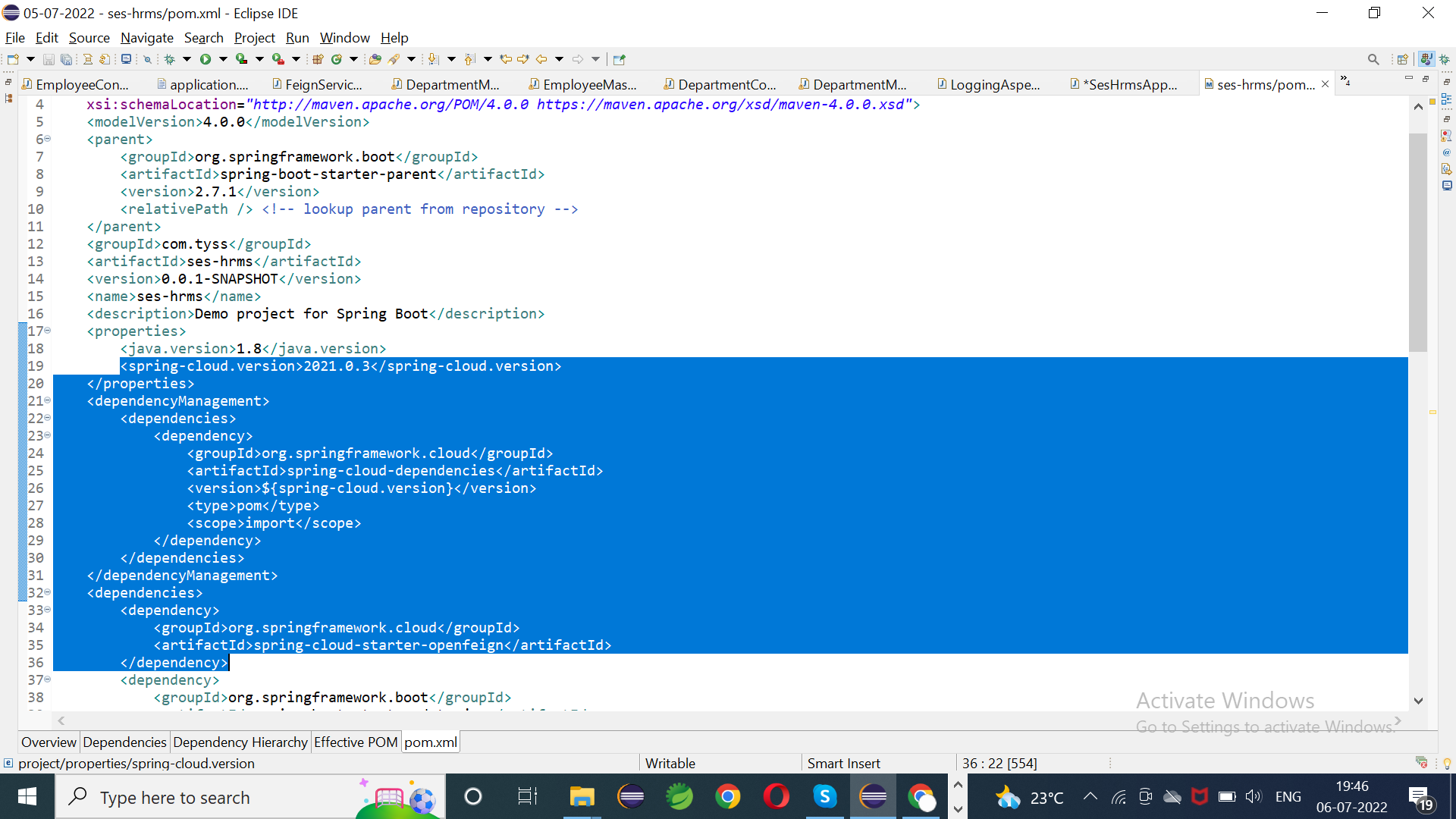
1. API gate way is also client side application, where it has to be registered to Eureka.
2. Add eureka client dependency, annotate the main class with***@EnableEurekaClient.*** Configure the yml file as below PFA.



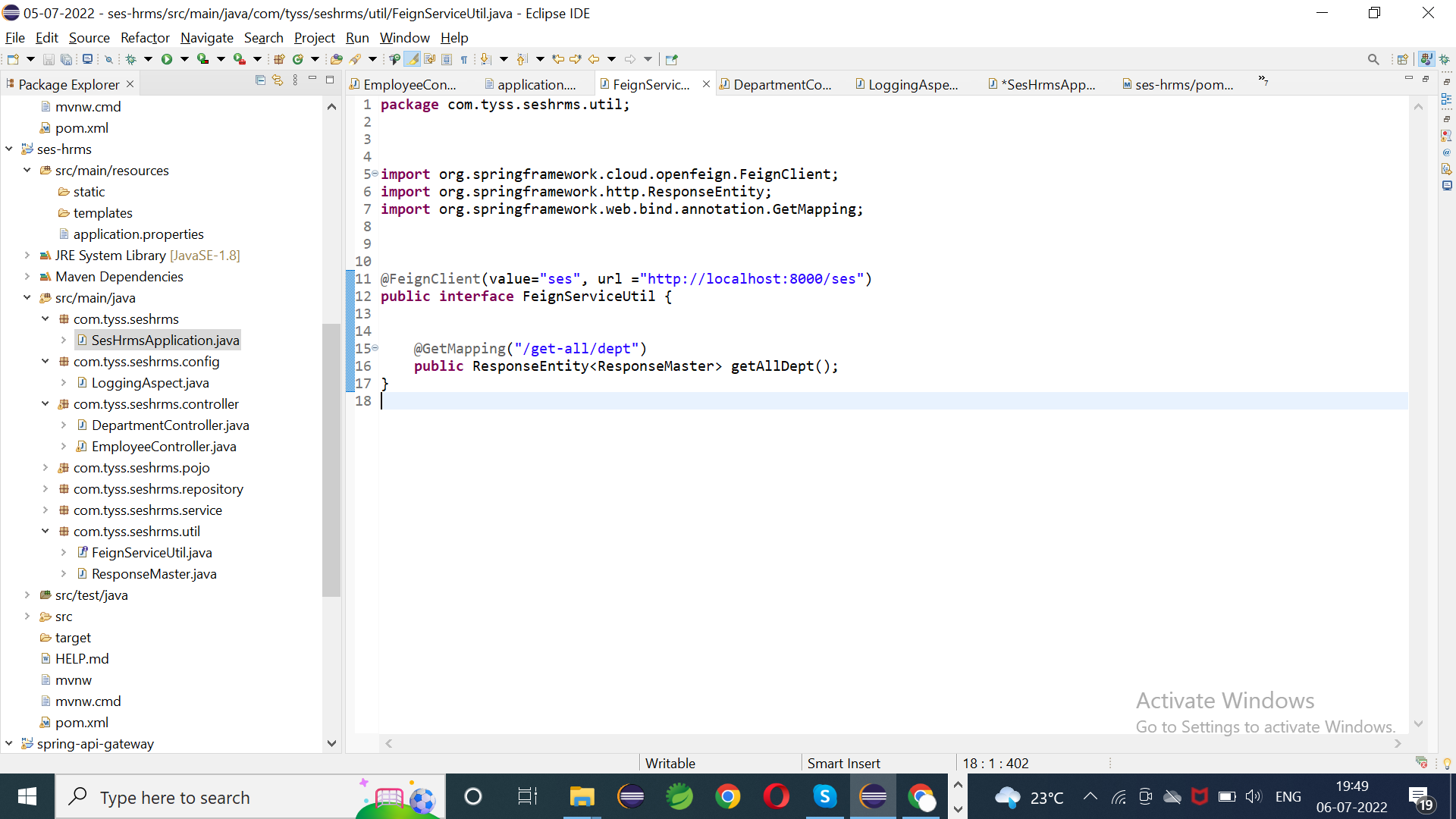
**Feign Client:** It is used for the communication among micro services.

**Steps:**

1. **Add below dependency in client service.**



1. **Annotate the main class of client with @**EnableFeignClients
2. **Create an interface which calls the api of another service.**



**Value** = Application name of another service

**Url** = url with request mapping of another service

1. **Create an api in the client controller**

**Hystrix:** Fault tolerance library through which we can stop delegation from one service to another.

It follows circuit breaker architecture. ***Circuit breaker*** is used to check availability of external service like data base connection, web service call etc.

**Spring Cloud Sleuth**: It is used to generate and attach the trace id, span id to the logs so that these can then be used by tools like Zipkin and ELK for storage and analysis.

The Spring Cloud Sleuth token has,

**applicationName** + **traceId** + **spanId** + **zipkinExportFlag**

**applicationName =** Name of application given in application. properties

**traceId =** This id is given by Sleuth which will be same for all services

**spanId** = This is given by Sleuth which will be same for unit of work (e.g. same within method)

**zipkinExportFlag** = Boolean value which tell, whether span can be extended to Zipkin or not.

Steps:

1. Add below dependencies

[Zipkin-Slueth Dependency.txt](Zipkin-Slueth%20Dependency.txt)

1. Add below code in properties file

spring. zipkin. baseUrl= http://localhost:9411/zipkin/

sleuth. sampler. Probability: 1.0

1. Hit **localhost:9411** in your browser for Zipkin dashboard.

You can see each request details as below.

