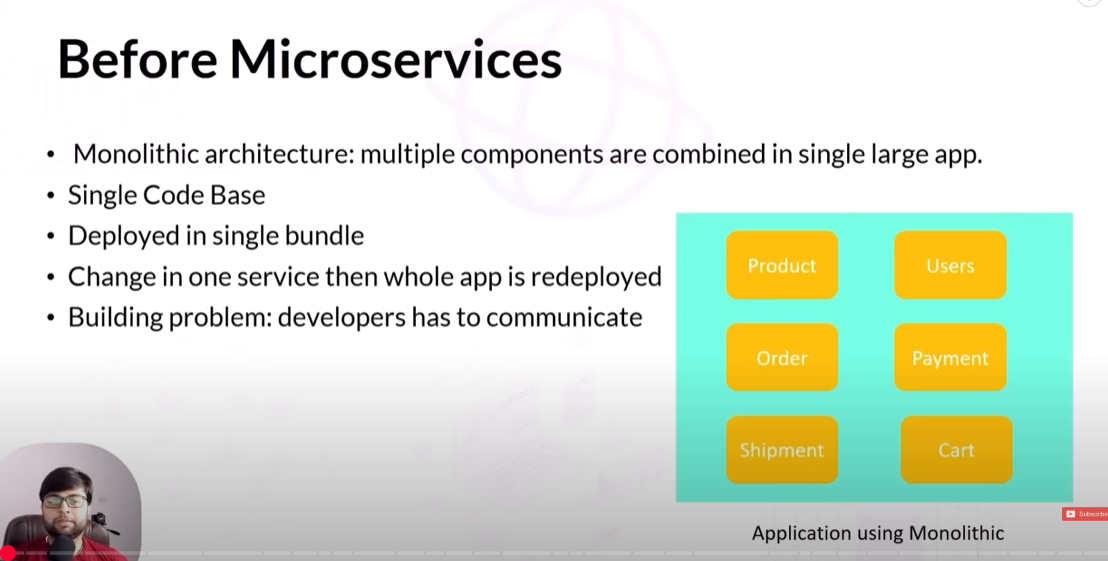
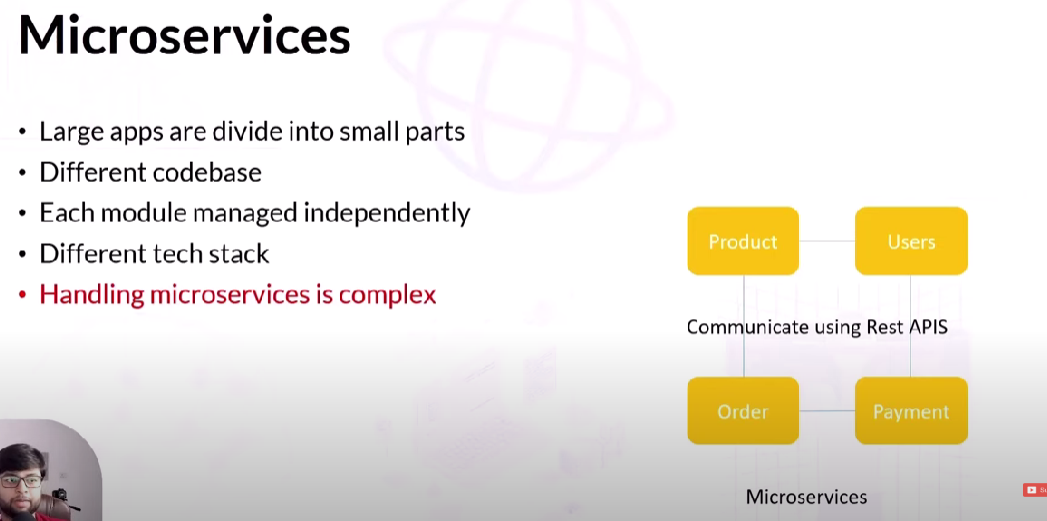
**Microservices Tutorial using Spring Boot in One Video**

**Hotel Rating System Project**





Example: Suppose we need to get information about user (235), what rating about which hotel has given.

User service will communicate with rating service and will find out rating and it belongs to which hotel etc.

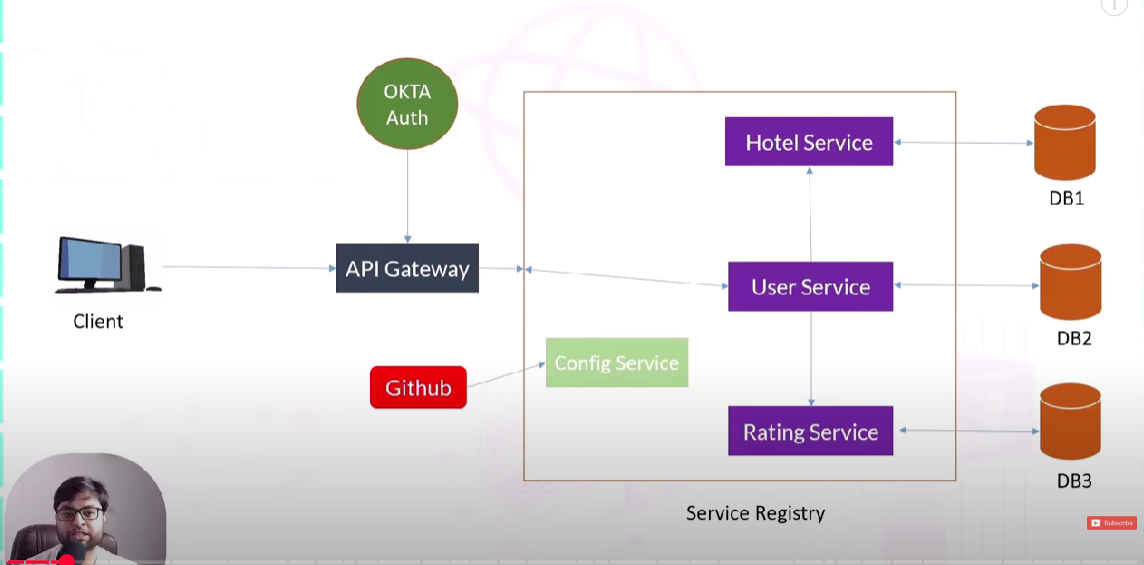
**Config Service:** We can put all the information if any configuration is common among all other 3 microservices **OR** if we want to store any information on the server and we can use in our microservices.

**API Gateway:** All the requests from the client will be sent to API gateway. API gateway will the main gateway from where all the requests will be going to other microservices.

**Authentication:** Third party service (OKTA Auth) with help of Spring security

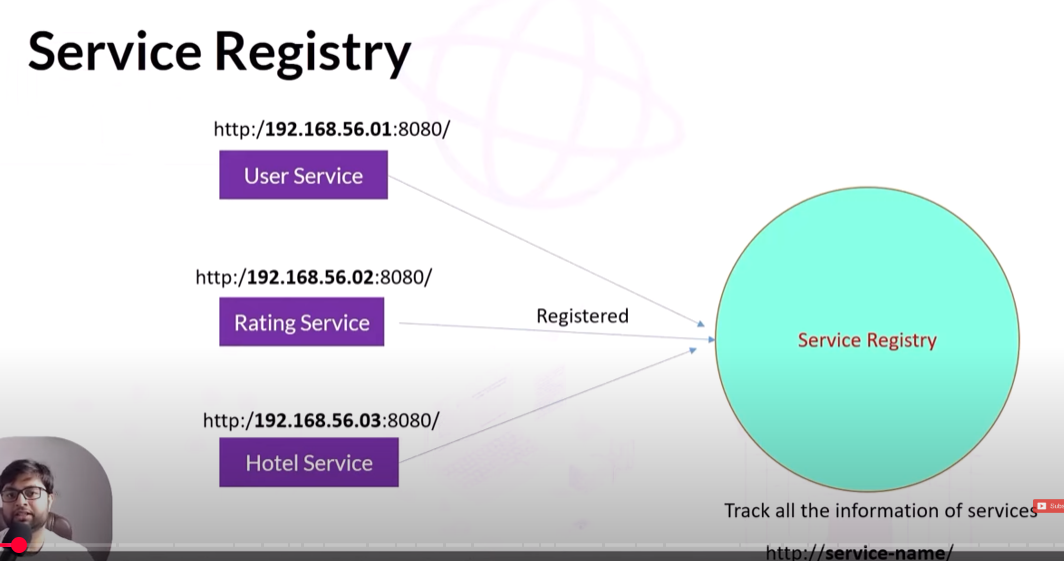
**Service Registry:** We need to register all the microservices here where we can track if any services are up or down.

**Architecture Diagram:**



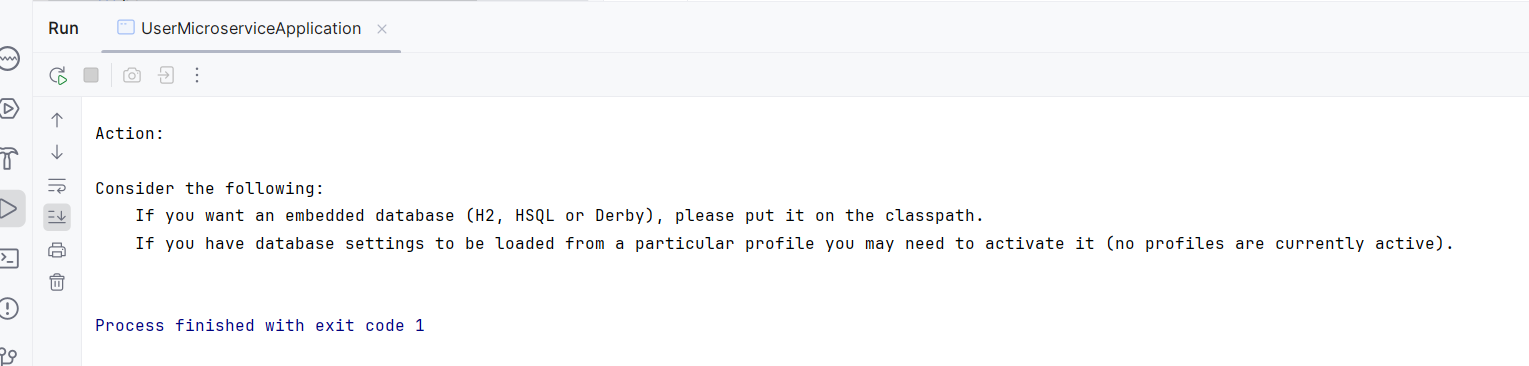
**Benefits of Service Registry:**

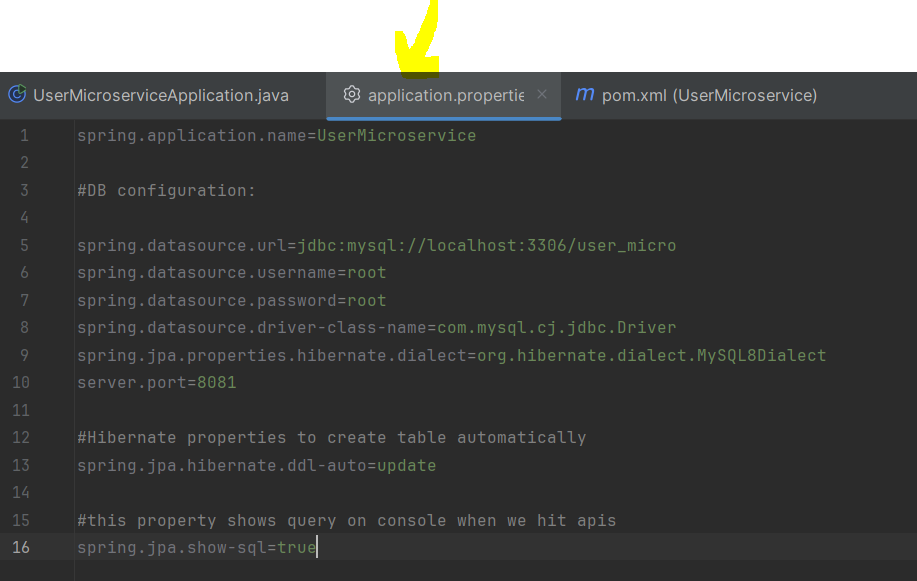
IF all our microservices got registered in Service Registry, we can call services with the help of service name instead of IP address. In future, if IP address gets changes, we can still access the services with help of service name.



**DB Configuration:**

**If we don’t setup DB configuration in our project and runs the application, it will throw below error**





**Entity Class:**

1.Make userID as primary key (@ID annotation)

**2.**

**Repo:**

**To perform DB operation on entity, we will create repository (an interface)**

**Service class:**

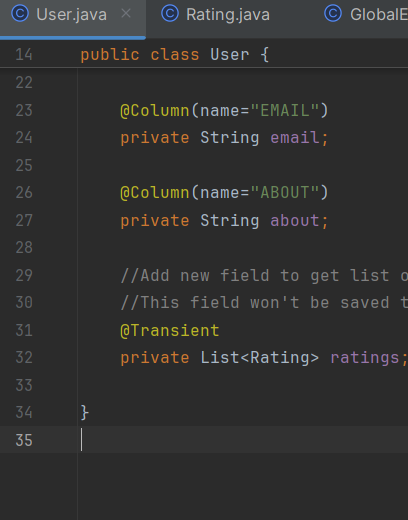
**Impl class:**

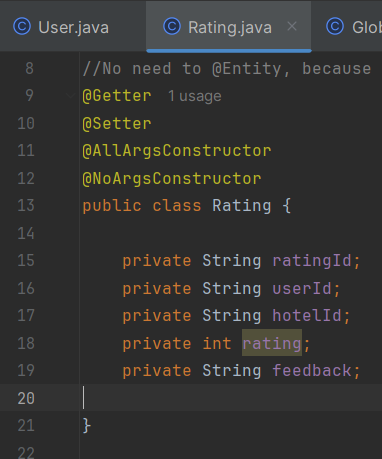
**Exception Handling:**

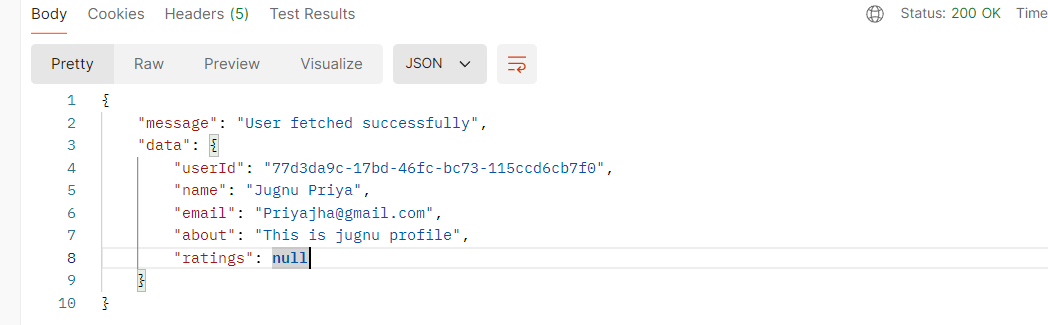
**Controller:**

**When you fire (/user) from request mapping, controller class will load up and active.**

**ResponseEntity: we want to send status as well.**

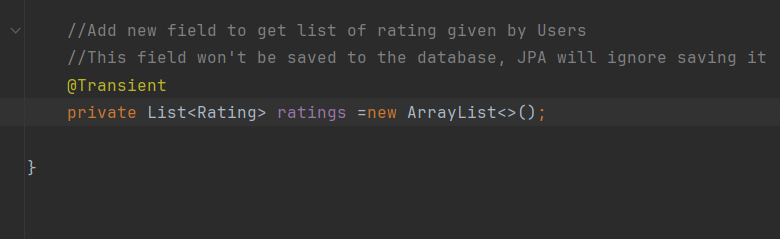


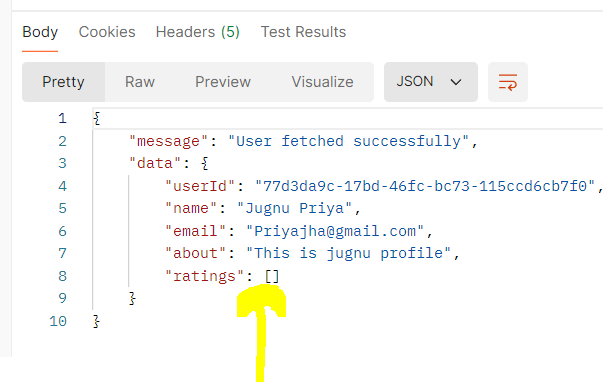




**Currently, getting null in rating. We will fetch rating information from other Rating microservices.**

**Created blank arrayList**





**Hotel Microservice:**

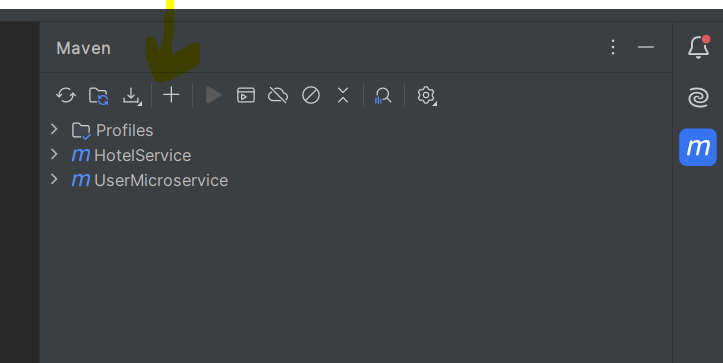
**Created another microservices “Hotel Microservice”.**

**Launch Hotel microservice in same window of IntelliJ instead separate window**

**Steps to open in same window:**

**Go to IntelliJ**

**Click on Maven (right side)**



**Choose Hotel service, pom.xml parent directory and ok.**

**In impl, to do all the operations we need repo.**

**@Autowired**

**Private HotelRepository hotelrepository;**

**NOTE:::::::**

**Now, I am going to create another microservice "Hotel micro" where I have also GlobalExceptionHandler but thing is I dont want to create another class like APIResponse like I did in user micro,, I want to use Map class now in ResponseEntity, how to do..**



**Rating Microservice:**

**1.Created Rating microservice project using spring initializer**

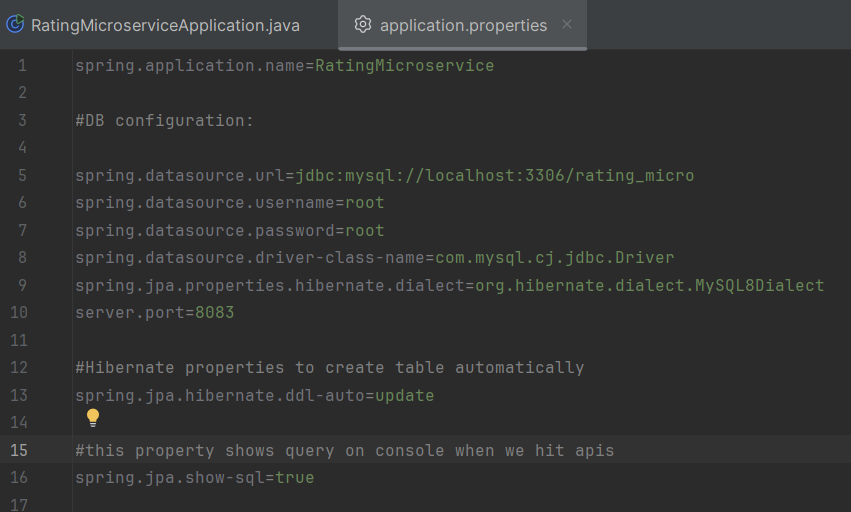
**2.we included important dependencies like (web, Lombok, jpa, sql)**

**3.downlaod zip file and extract**

**4.Go to IntelliJ>>right side Maven option>>add new project**

**5.We can see Rating microservice opened in the same window(User, Hotel)**

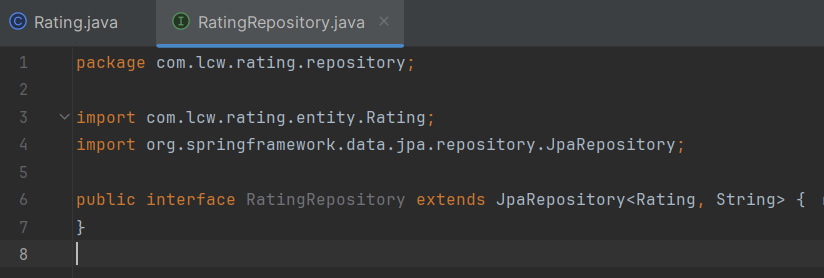
**DB Configuration:**



**Entity Class:**



**Repository:**



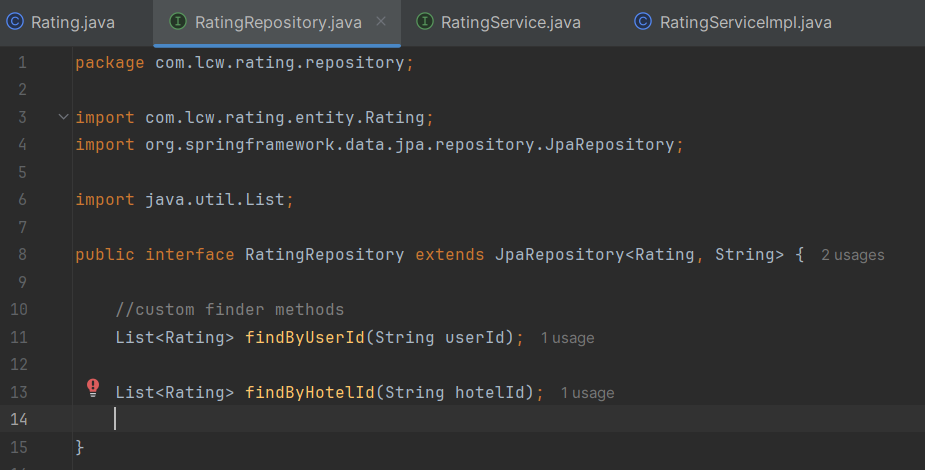
**NOTE: When we work with relational DB, we have to extend JPARepository and when we work with NoSQL DB, we extend MongoRepository**

**Service:**

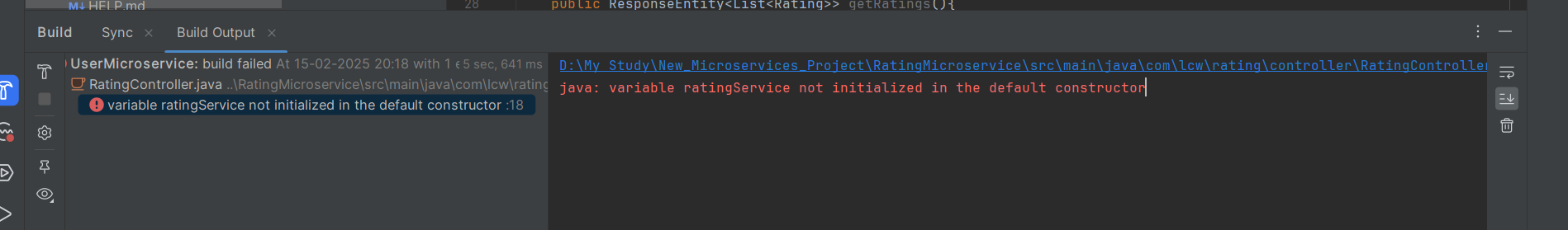
**Impl:**

**Custom finder methods:**

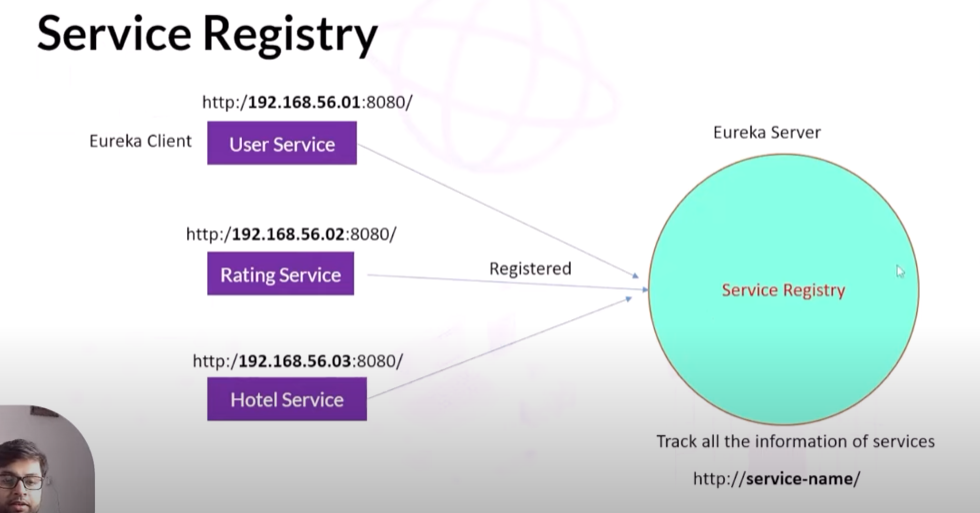
**Syntax: what we want to return and then method name**



**Lombok Version Issue:**

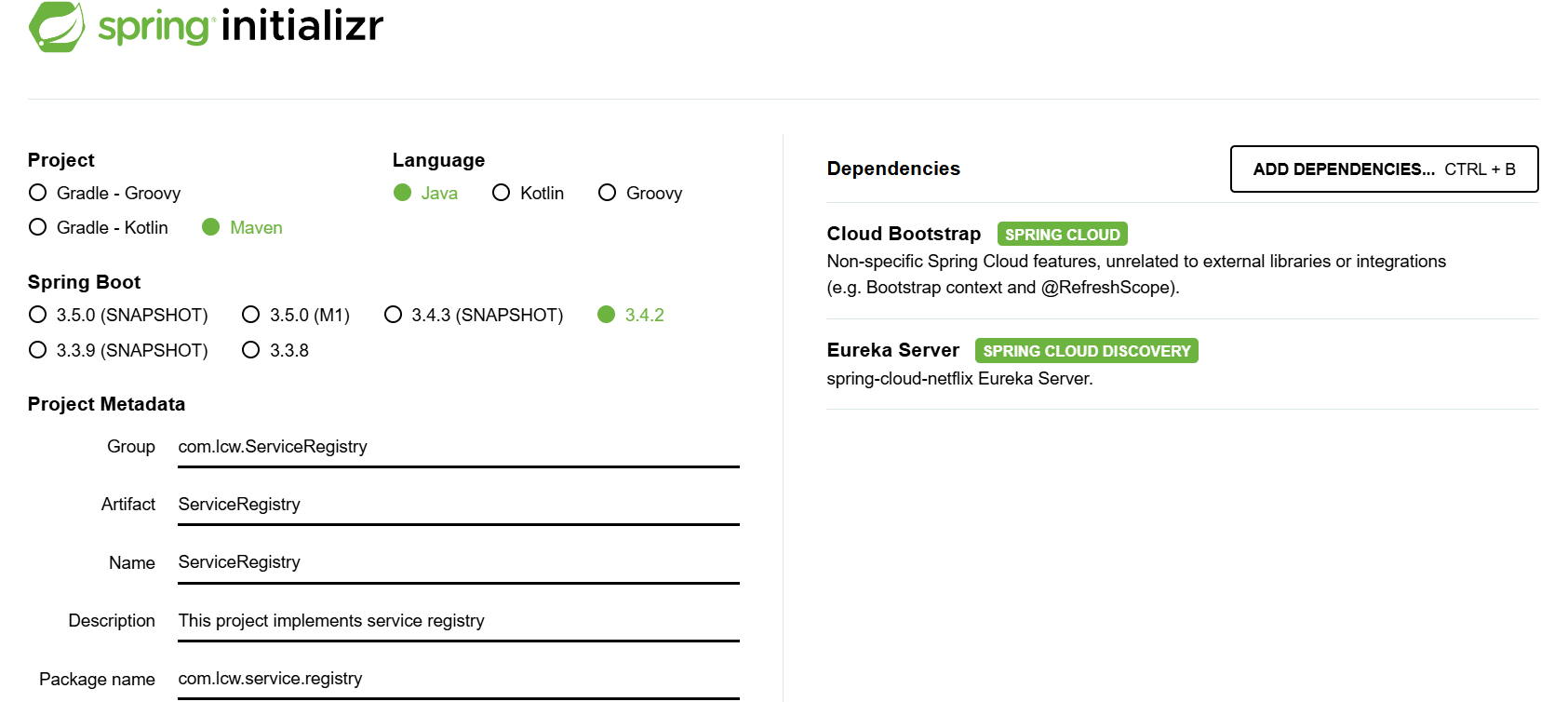


**Service Registry:**



**Project Setup:**

1.Create service registry project from spring initializer(As per below dependency). We don’t want spring web dependency in service registry project



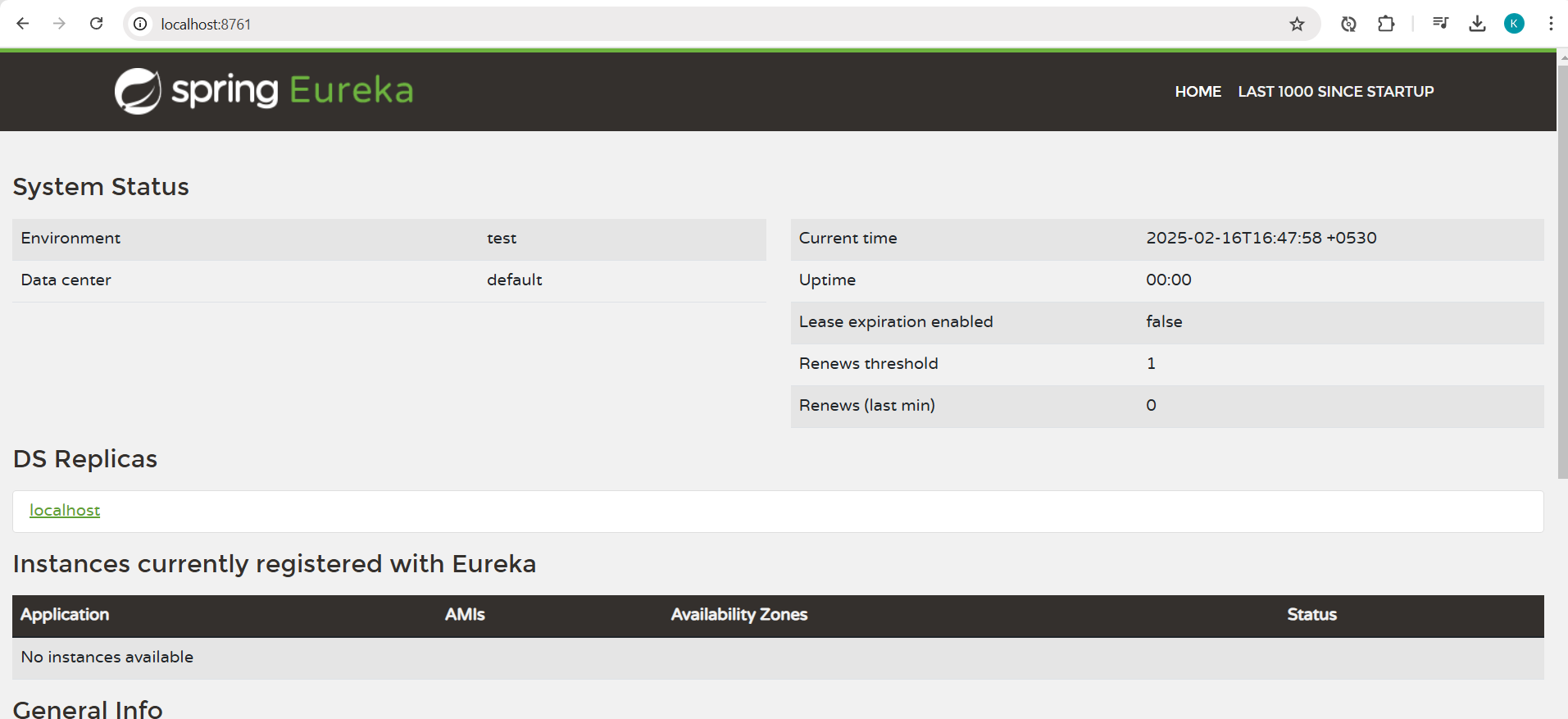
* **Generate the project and download Zip file**
* **Unzip and open project in IDE.**
* **Eureka server: It is given by Netflix**

2.Secondly, In the main application class, add @EnableEurekaServer so that all other microservices will be registered here.

3.We don’t want this service registry to be registered as a client. (Khud se self registar na ho jaye).



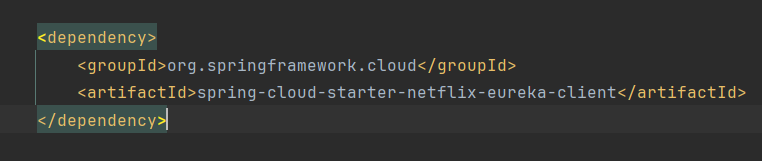
4.Start the eureka server and see the Eureka Dashboard.



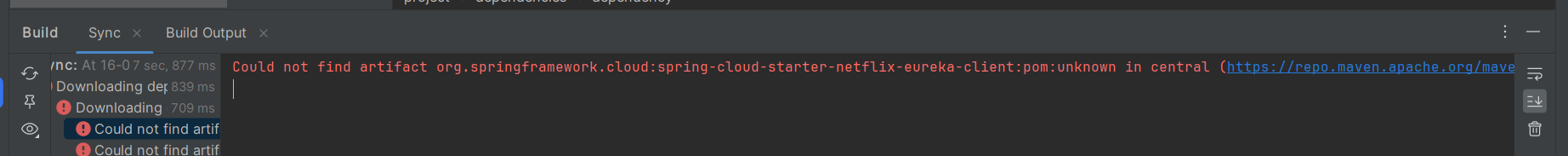
In the above screenshot of Eureka Server, we can observe it says: No instance available, that means no microservices has been registered yet.

**Register Microservices with Service Registry:**

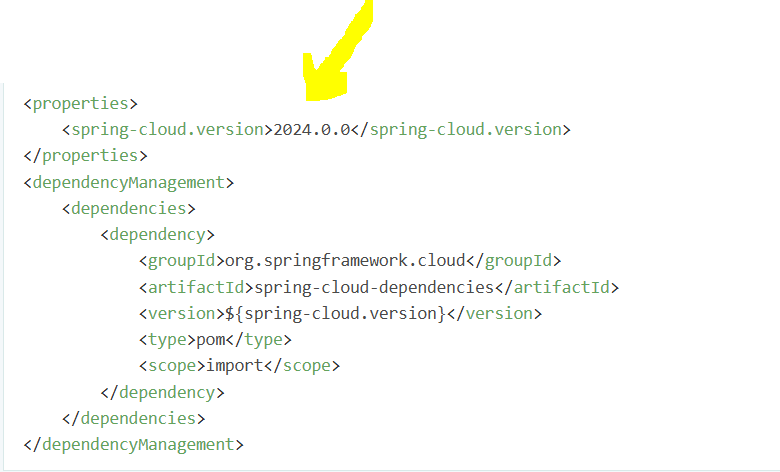
**1.Add Eureka client dependency in pom.xml (User microservice)**



**2.Now, If we reload the User microservice we will get compilation error as below**

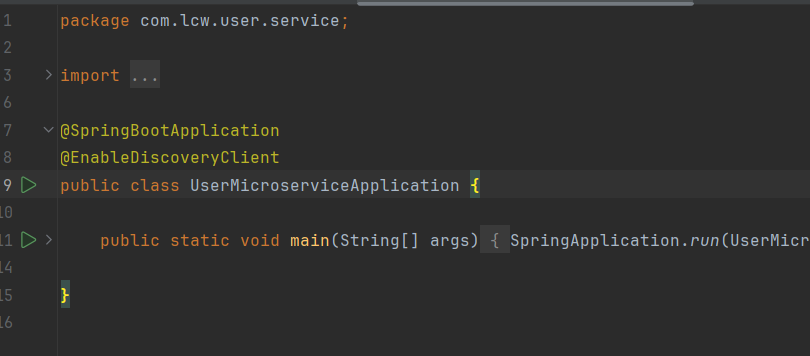


**3.** **For Spring Boot 3.x.x, the recommended approach is to use Spring Cloud Discovery Client in dependency management along with compatible spring cloud version as well.**



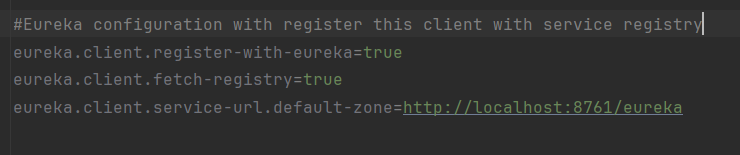
**4.We will enable Eureka client from main application of User Microservice.** **It allows your Spring Boot application to register itself with a Service Registry (like Eureka).**

**5.We will add @EnableDiscoveryClient in main application of user service. But this is optional in spring boot 3.4.0 version**

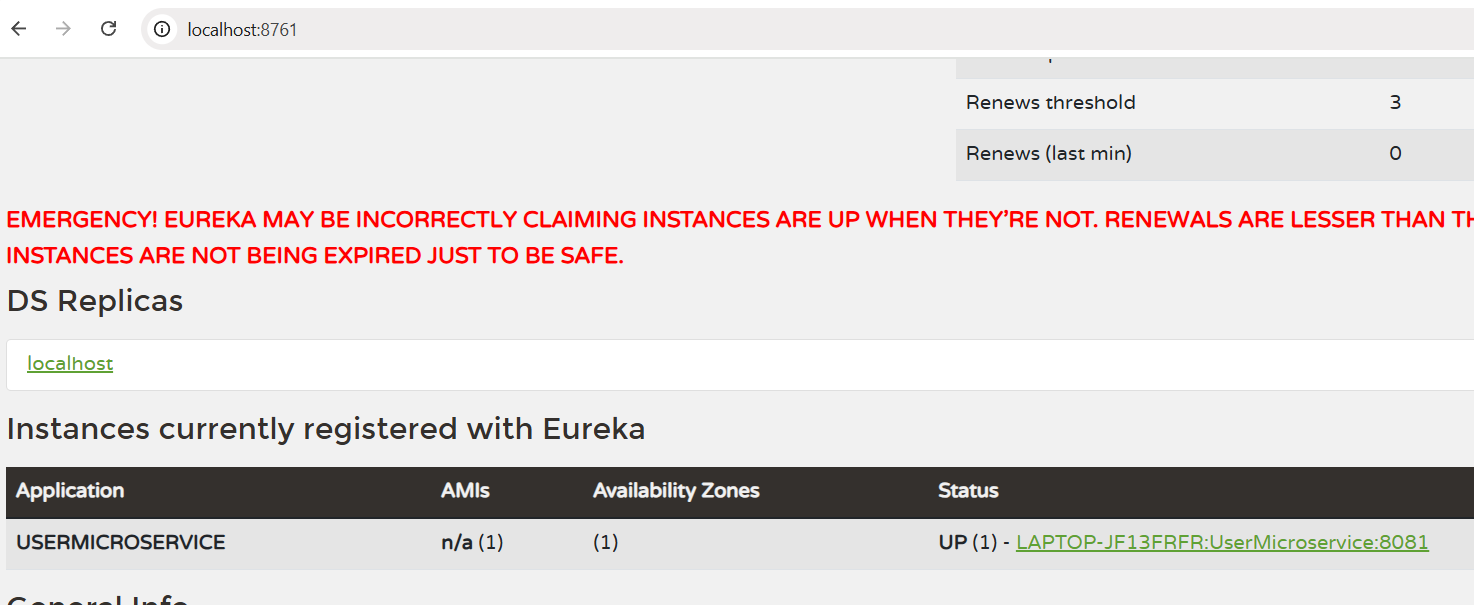


5.We will add below configuration in properties file of user service.

**Eureka.client.service-url.default-zone (it will use this url to register to service registry)**

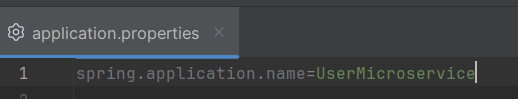


6.Run the Client(User Microservice) and can see that it has been registered successfully to Service Registry.

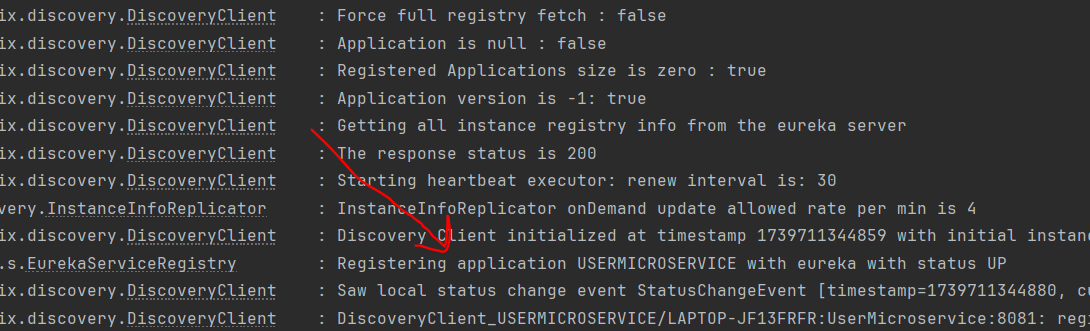


7.We can see appication(UserMicroservice) name is populating on the service registry(8761)

8.Application name comes from spring.application.name properties

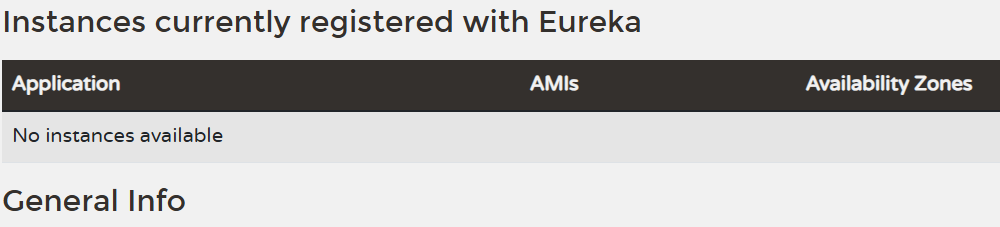


9.We can also see from the logs, eureka client(UserMicroservice) has been registered

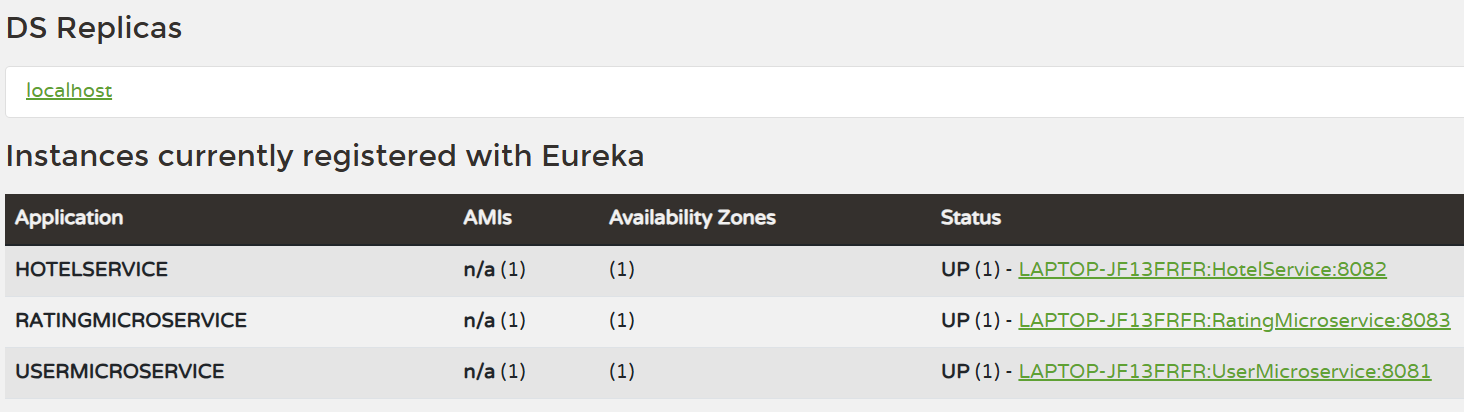


**If Service is Down:**

**Suppose we stopped eureka client(UserMicroservice) application and can see that it is not populating now on service registry.**



**NOTE: We can repeat the above steps to register other microservices as well to Service Registry.**



**Interservice Communication between microservice:**

**1.Using RestTemplate**

**2.Using Feign client**

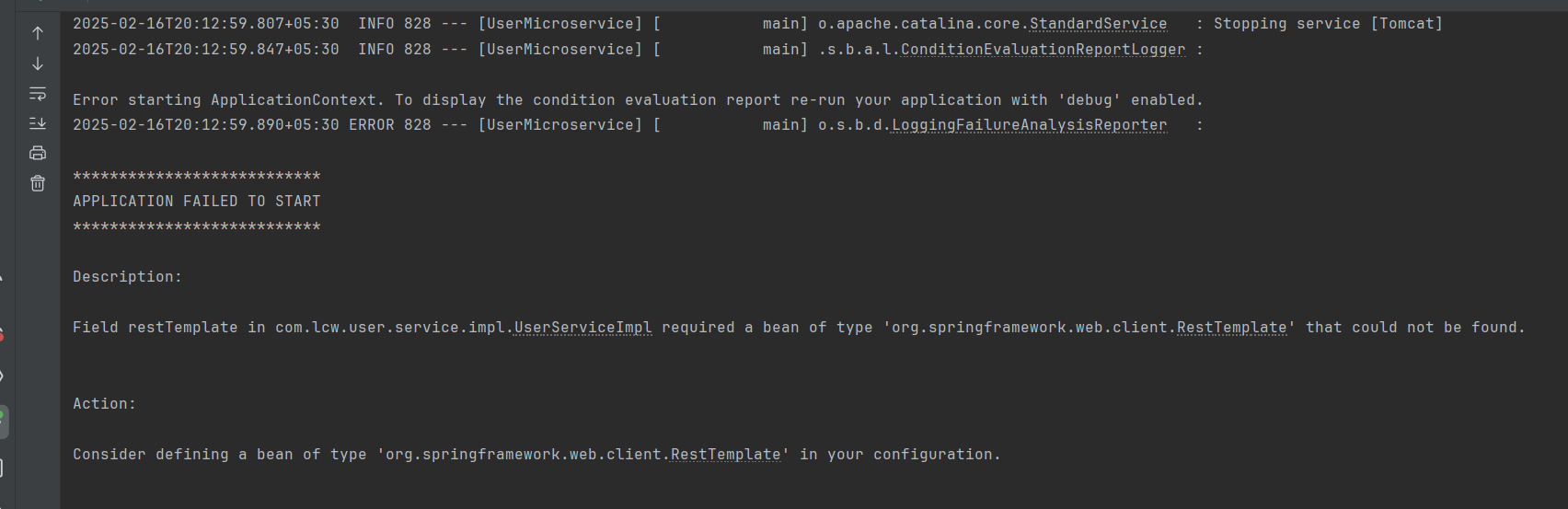
**Using RestTemplate:**

**Description:**

**Field restTemplate in com.lcw.user.service.impl.UserServiceImpl required a bean of type 'org.springframework.web.client.RestTemplate' that could not be found.**

**Action:**

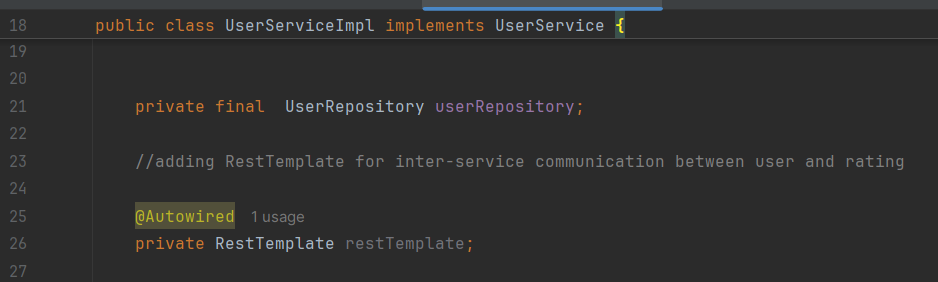
**Consider defining a bean of type 'org.springframework.web.client.RestTemplate' in your configuration.**

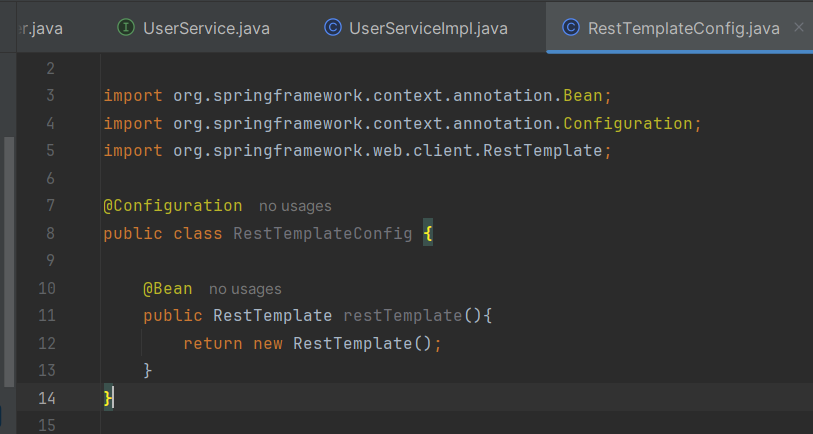


**Solution:**

We need to create a RestTemplate bean in a configuration class, so Spring can inject it into your service. We should be having bean of Resttemplate to Spring container.

We can create Bean either in Main class (@SpringBootApplication) or in separate new configuration class.



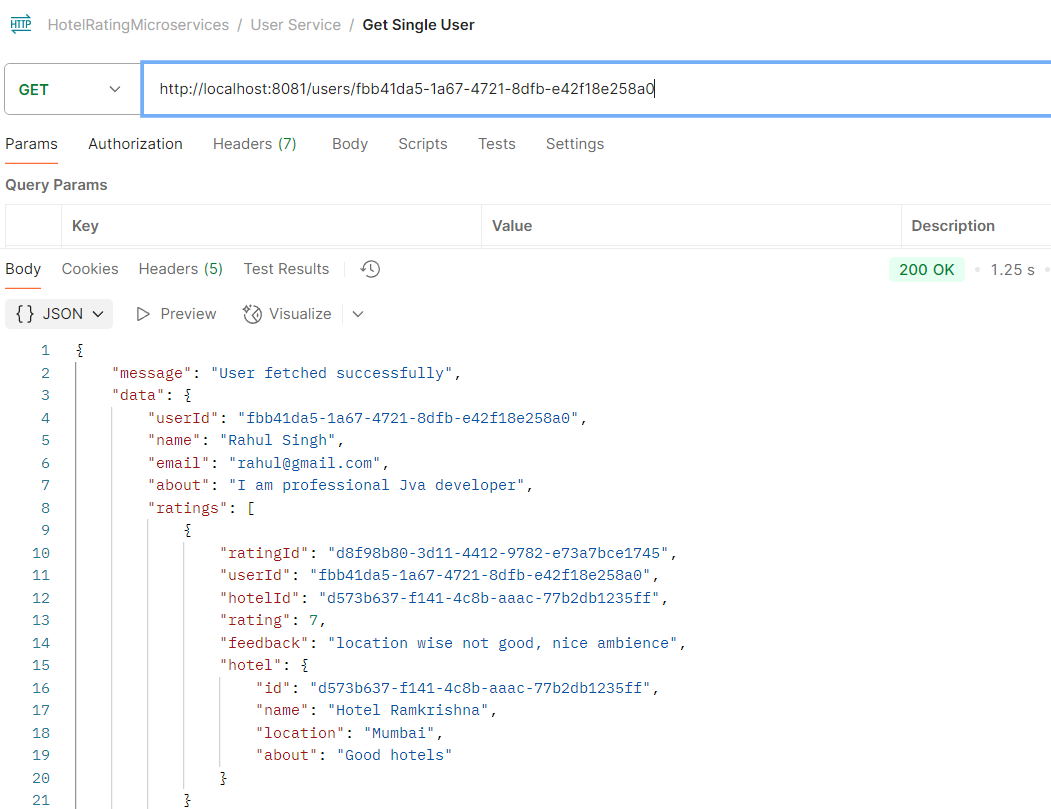


**We will be calling the below API from RatingController in userserviceimpl to get the list of Ratings given by specific user:**

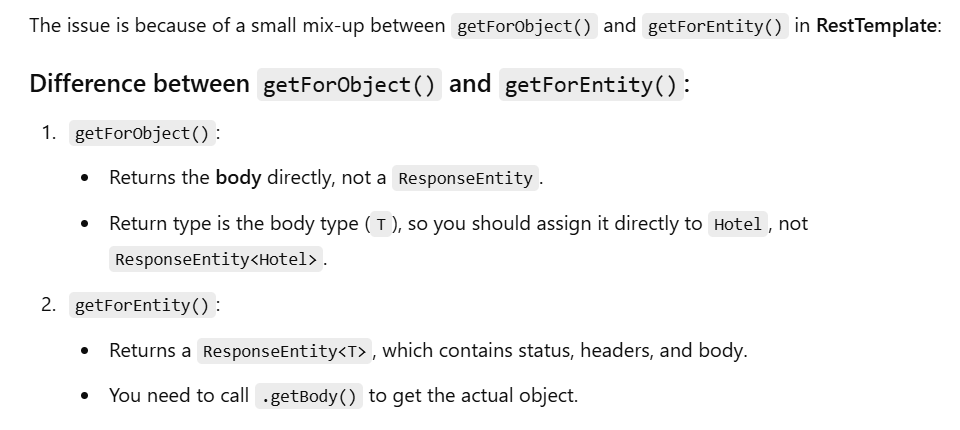
**//get all ratings by userid  
@GetMapping("/users/{userId}")  
public ResponseEntity<List<Rating>> getRatingsByUserId(@PathVariable String userId){  
 return ResponseEntity.*ok*(ratingService.getRatingByUserId(userId));  
}**



**class java.util.LinkedHashMap cannot be cast to class com.lcw.user.service.entity.Rating (java.util.LinkedHashMap is in module java.base of loader 'bootstrap'; com.lcw.user.service.entity.Rating is in unnamed module of loader 'app')**



**We can use getForObject() method as well from resttemplate instead of getForEntity. But there is small difference between them:**



**Hardcoded URL:**

**But still our URL is hardcoded means host and IP can be changed (like in PROD, STAGING). Our Job is to provide the URL dynamically.**

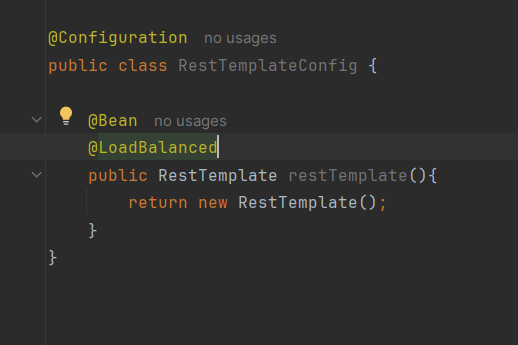


**We can use application(service) Name to call the one microservice to another microservice instead of Host and PORT.**

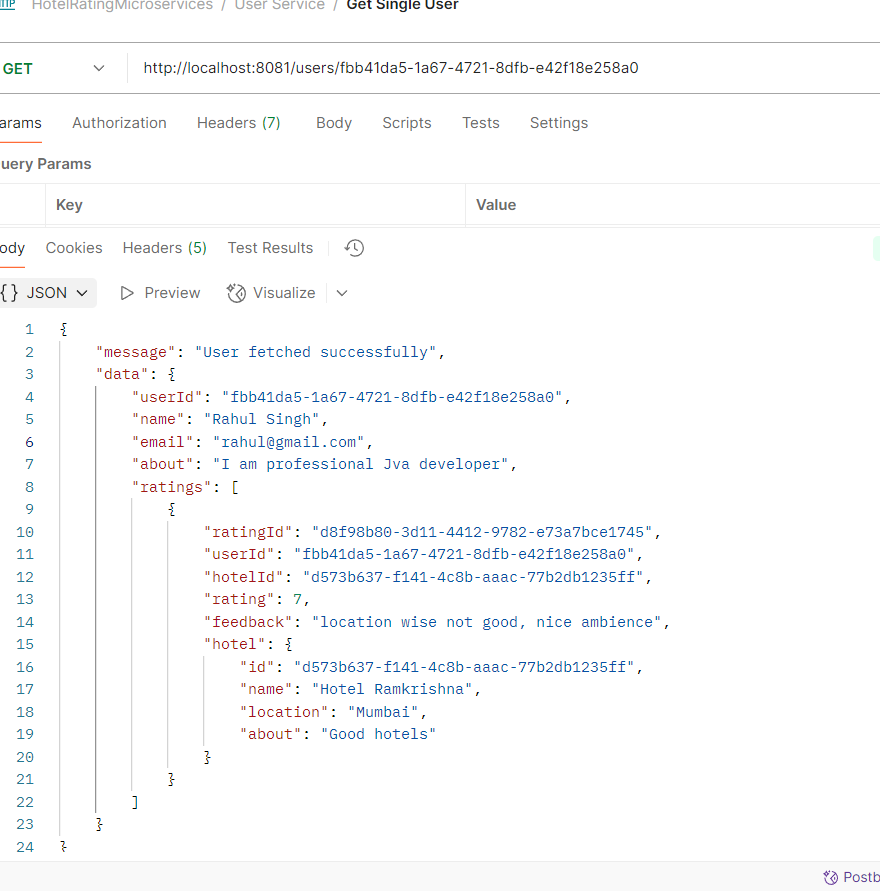
1.Remove Host and port from the URL and use service name which is reflecting on service registry



2.Add @LoadBalanced annotation in configuration class with @Bean. It will distribute the load if there are multiple instances(use service name to make api call)



3.Thats it, within 2 steps we can use service name to call one microservice to another microservice.



**Feign Client:**

It follows declarative approach

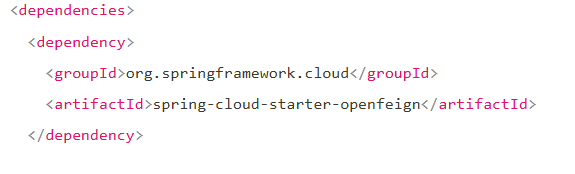
It is developed by Netflix

IF we want to use a Feign, create an interface and annotate it.

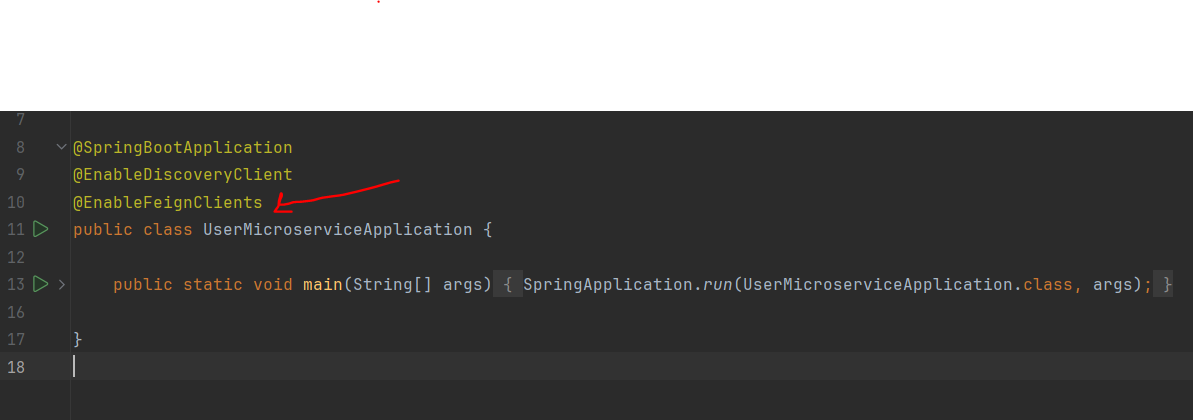
Feign client is also a part of Spring Cloud.

How to Use Feign client step by Step:

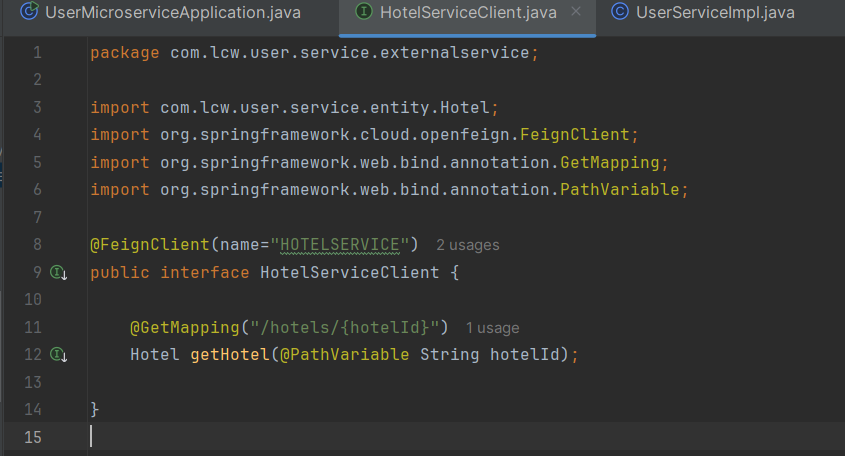
1.Add Open Feign dependency in pom.xml



2.Add @EnableFeignClient annotation in main class application

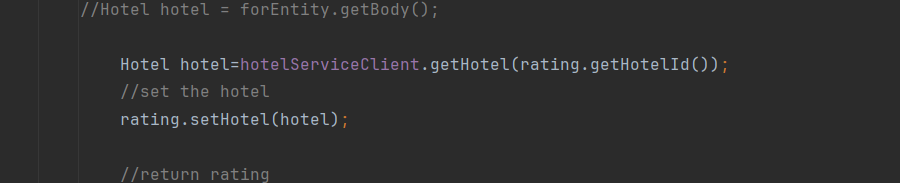


3.Create an interface, annotate it with @FeignClient and provide name of the microservices you want to call



4.Inject newly created service class in impl



5. 

6.Thats it, with this 5 steps we can use Feign client to make microservices call