**Access App via Router**

Implement using Spring Cloud (Eureka) as service registry and Spring Boot as REST Service and the service should be accessed via the router

**Functionalities:**

The Product Application is developed and provided as part of the code skeleton. This application is a rest service application. The name of the application is "**productapp**". You are required to register the given REST service in the eureka server.

The Eureka server is running in port number 8761 and the URL for the same is given below:

<http://localhost:8761/eureka>

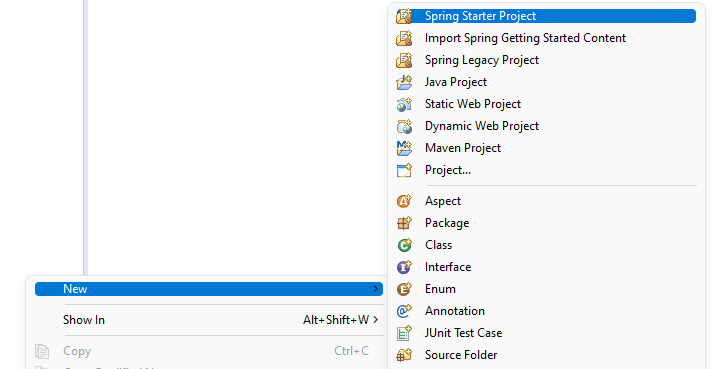
The ProductApp application should be accessed via the Router, you need to develop the Router application do to the same.

In STS create below 3 projects,

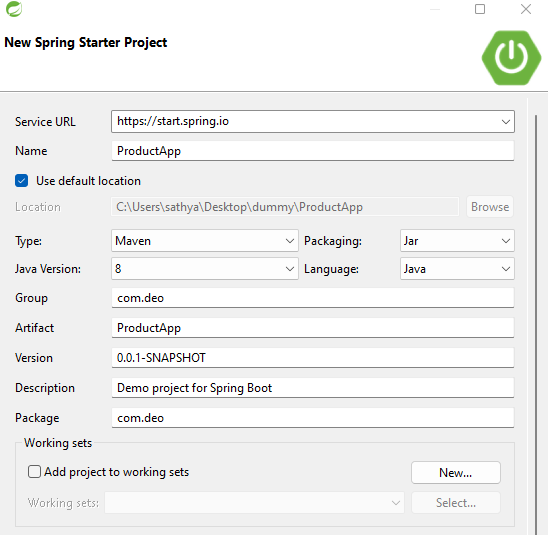
* ProductApp
* Registry – to register the ProductApp application in Eureka Server.
* RouterApp - ProductApp application should be accessed via this RouterApp

**Steps to create a projects in STS:**

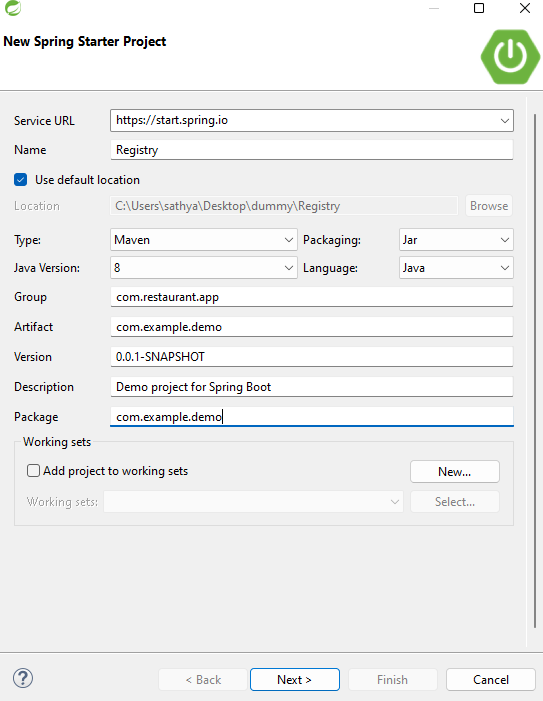
1. Open Spring Tool Suite (STS) and click on File > New > Spring Starter Project. Alternatively, you can also click on the Spring icon on the toolbar and select Spring Starter Project from the drop-down menu.



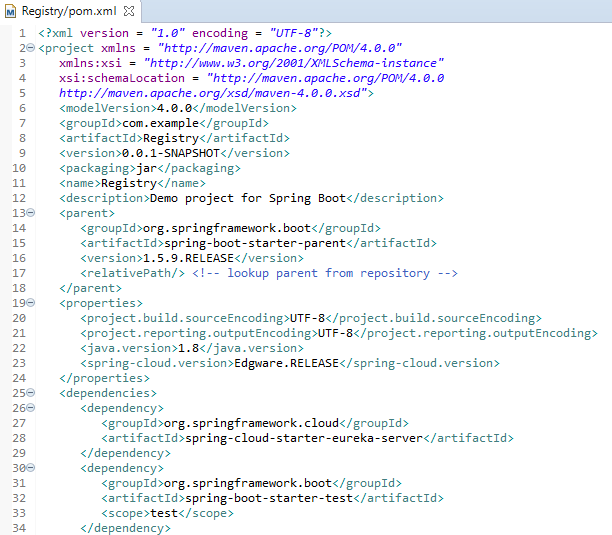
1. In the New Spring Starter Project wizard, you will be prompted to enter the following information:
   1. **Project Name:** Enter a name for your project.
   2. **Group:** Enter the group name for your project.
   3. **Artifact:** Enter the artifact name for your project.

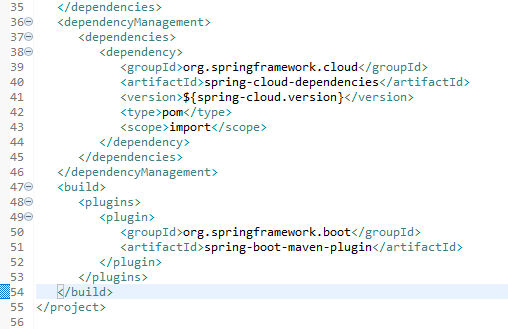


1. Click on the Finish button to create the project. STS will create the project with the specified dependencies and generate the necessary files.
2. Follow the above steps to create other two projects called **Registry and RouterApp.**

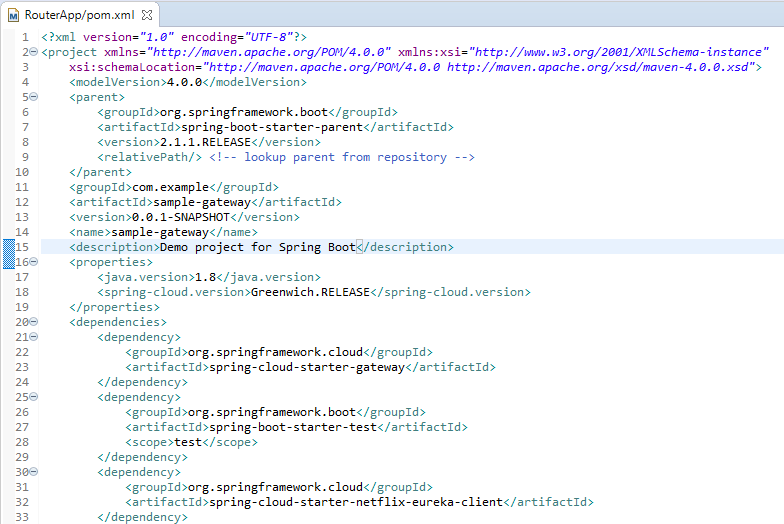


1. Once the project is created, add the below dependency in Registry pom.xml file.



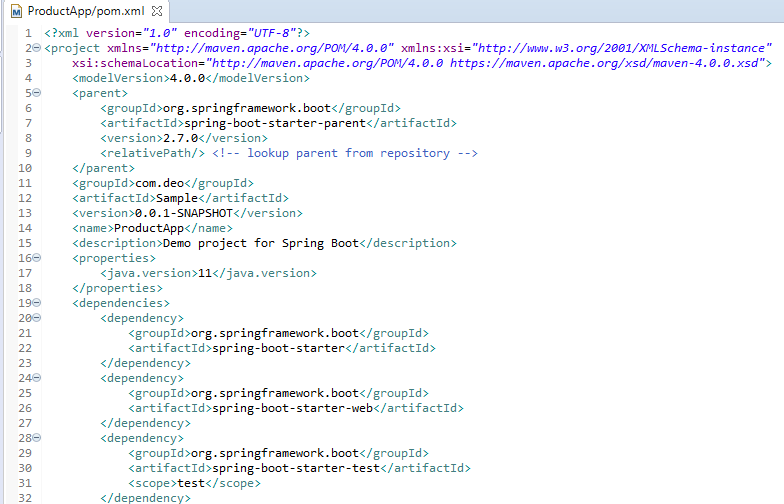


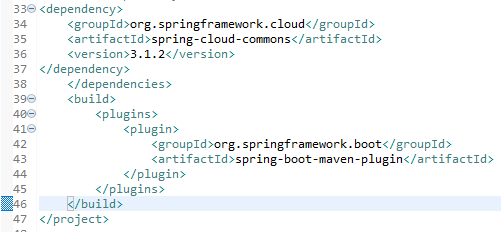
1. Once the project is created, add the below dependency in RouterApp pom.xml file.





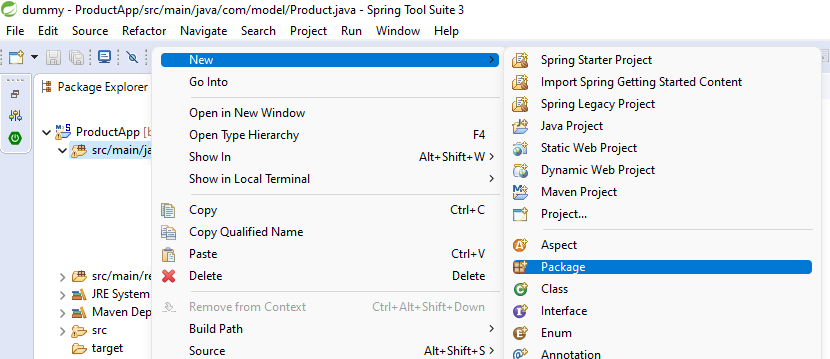
1. Once the project is created, you can open the project in the Package Explorer view and explore the files that have been generated. You can also start coding your application.
2. Once the projects are created, add the below dependency in ProductApp pom.xml file.



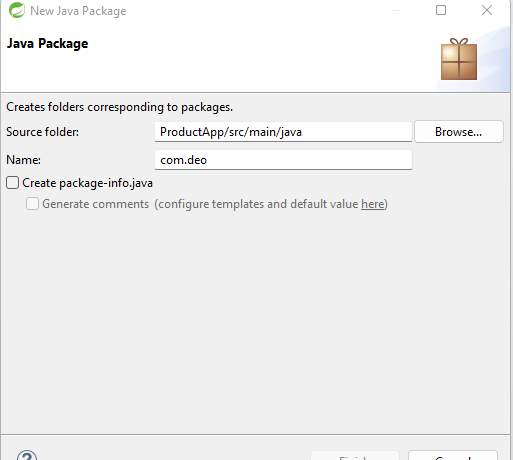


**Requirements:**

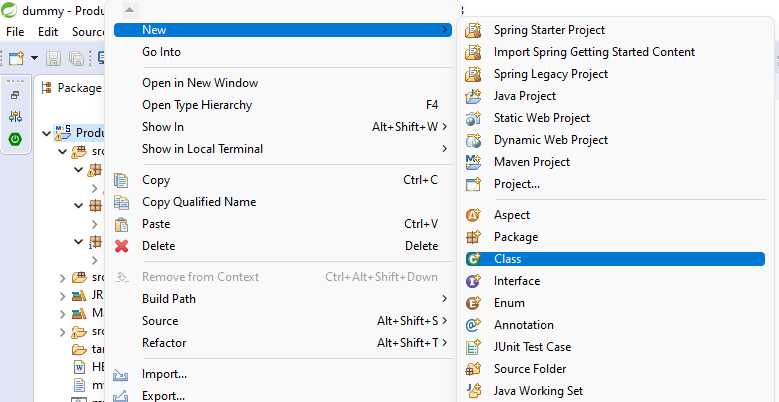
1. Right-click on the ProductApp project in the Package Explorer and select New -> Package.



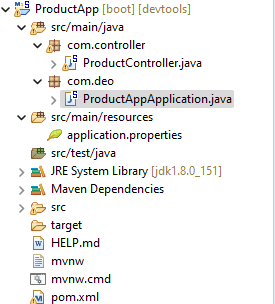
1. In the New Package dialog box, enter the name of the package as **com.deo**



1. Click Finish to create the new package.
2. Once you have created the package, you can create new classes in it by right-clicking on the package and selecting New -> Class.



1. In the New Java Class dialog box, enter the name of the class **ProductAppApplication** and click Finish.
2. Follow the above steps to create other packages and classes required for this project.

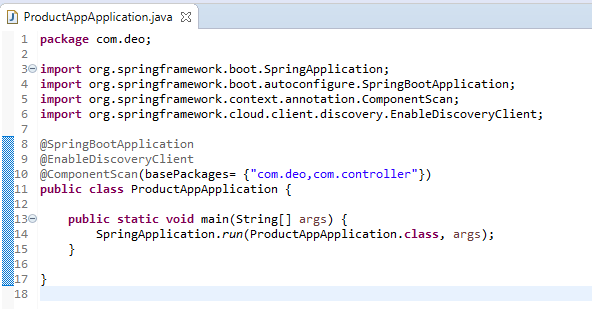


1. In ProductController class include the code which is provided as a part of code skeleton.



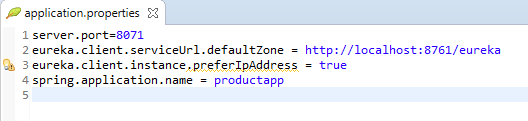
* 1. The **com.controller** package contains the **ProductController** class, which is annotated with **@RestController**. This annotation indicates that this class is responsible for handling HTTP requests and returning responses in a RESTful manner.
  2. The @**GetMapping** annotation on the **getDetails** method specifies that this method should handle **GET** requests to the /**getInfo** endpoint.
  3. Inside the **getDetails** method, the code simply returns a String response that says "This is product application".

1. In **ProductAppApplication** class include the necessary annotation that enables this application to register with a discovery service like Eureka.



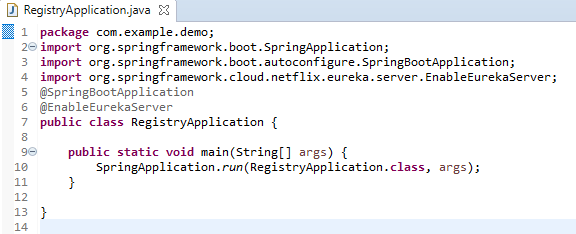
* 1. @**SpringBootApplication** is an annotation that tells Spring Boot to scan this class and all of its sub-packages for components and configuration.
  2. @**ComponentScan** is an annotation that specifies the package(s) that Spring should scan to find the components that will be used by this application.
  3. @**EnableDiscoveryClient** is an annotation that enables this application to register with a discovery service like Eureka or Consul, which can be used for service discovery and load balancing.

1. In **application.properties** file specify the server port number, application name and sets the URL for the Eureka server that this application will register with.



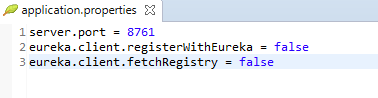
* 1. **"server.port=8071",** this line sets the server port to 8071. This means that the application will listen for incoming HTTP requests on port 8071.
  2. **"eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka",** this line sets the URL for the Eureka server that this application will register with. Eureka is a service discovery and registration server that enables microservices to find and communicate with each other. eureka.client.serviceUrl.defaultZone specifies the URL of the default Eureka server, which is running on the local machine on port 8761.
  3. **"spring.application.name=ProductApp",** this line sets the name of the application. The name is used as a identifier for the application when it registers with the Eureka server. In this case, the application name is set to "ProductApp".

1. In **Registry** project, **RegistryApplication** class is a Java code for a Spring Boot application that serves as a Eureka server. Include required annotation that enables this application to function as a Eureka server.



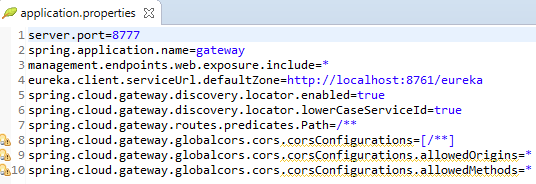
* 1. @**SpringBootApplication** is an annotation that tells Spring Boot to scan this class and all of its sub-packages for components and configuration.
  2. @**EnableEurekaServer** is an annotation that enables this application to function as a Eureka server. Eureka is a service registry and discovery server used for microservices architecture.

1. Configuration details needs to be added in application.properties file.



* 1. "**server.port=8761**", this line sets the server port to 8761. This means that the application will listen for incoming HTTP requests on port 8761.
  2. "**eureka.client.registerWithEureka=false**", this line disables automatic registration of the application with the Eureka server. When this property is set to false, the application will not register itself with the Eureka server.
  3. "**eureka.client.fetchRegistry=false**", this line disables fetching the registry from the Eureka server. When this property is set to false, the application will not attempt to fetch the registry from the Eureka server.

1. In **RouterApp** project, **RouterAppApplication** class is a Java code for a Spring Boot application that serves as a gateway. Include required annotation that enables this application to function as a cloud gateway.
2. @**SpringBootApplication**: This line is a Spring Boot annotation that indicates that this is a Spring Boot application.
3. @**EnableDiscoveryClient**: This line is a Spring Cloud annotation that enables service discovery in this Spring Boot application. It is provided by the Spring Cloud framework and is used to register the application with a service registry.
4. public class RouterAppApplication {: This line defines a public class called RouterAppApplication.
5. public static void main(String[] args) {: This line defines the main method for this application.
6. SpringApplication.run(RouterAppApplication.class, args);: This line starts the Spring Boot application by running the run method of the SpringApplication class. It takes two arguments: the class that contains the main method (RouterAppApplication.class), and the command line arguments (args).
7. Include the following code in application.properties file.

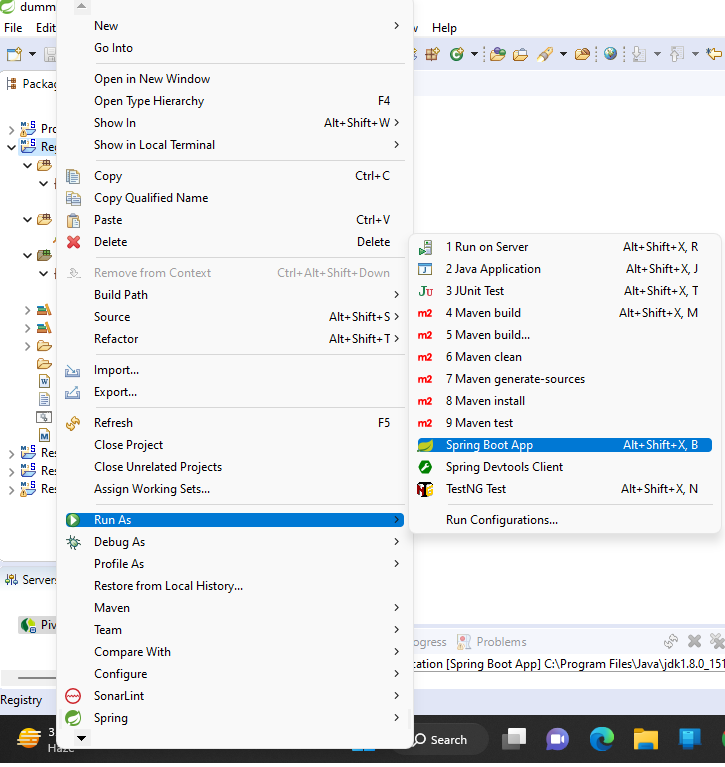


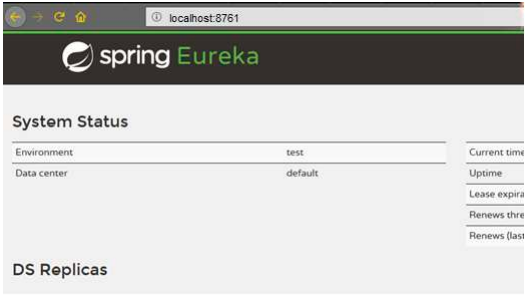
1. server.port=8777: This line sets the port number that the Gateway will listen on to 8777.
2. spring.application.name=gateway: This line sets the name of the Gateway application to "gateway".
3. management.endpoints.web.exposure.include=\*: This line exposes all management endpoints over the web.
4. eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka: This line sets the URL of the Eureka server that the Gateway will use for service discovery. In this case, it is set to <http://localhost:8761/eureka>.
5. spring.cloud.gateway.discovery.locator.enabled=true: This line enables service discovery for the Gateway using the Spring Cloud Gateway's Discovery Locator.
6. spring.cloud.gateway.routes.predicates.Path=/\*\*: This line specifies that all incoming requests should be handled by the Gateway.

**Steps to execute the projects**

**Project 1: Registry -** This project should be the Eureka-Server acting as the registry. Run this server in port 8761

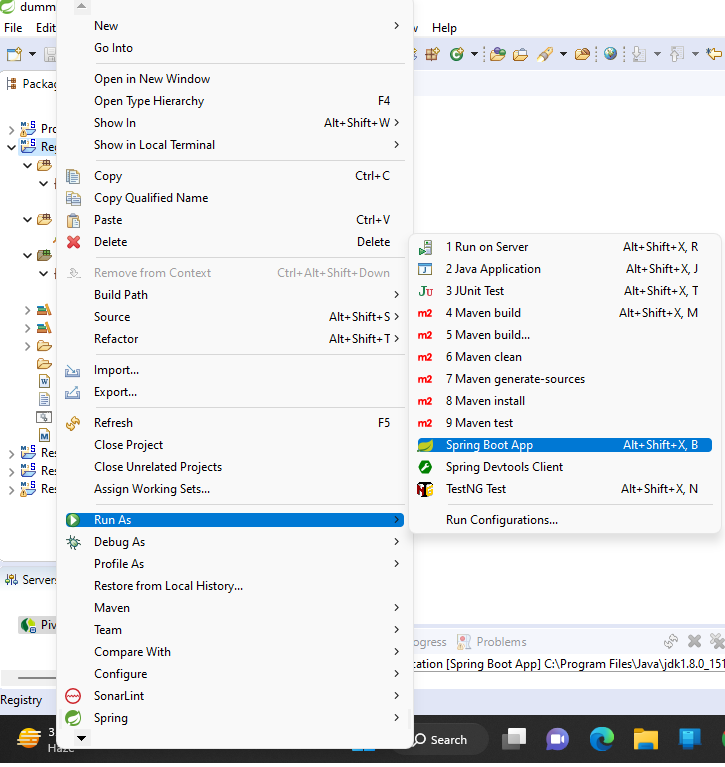
1. **Execute the Registry** 
   1. Right click on the Registry project -> click Run As -> Click Spring Boot App

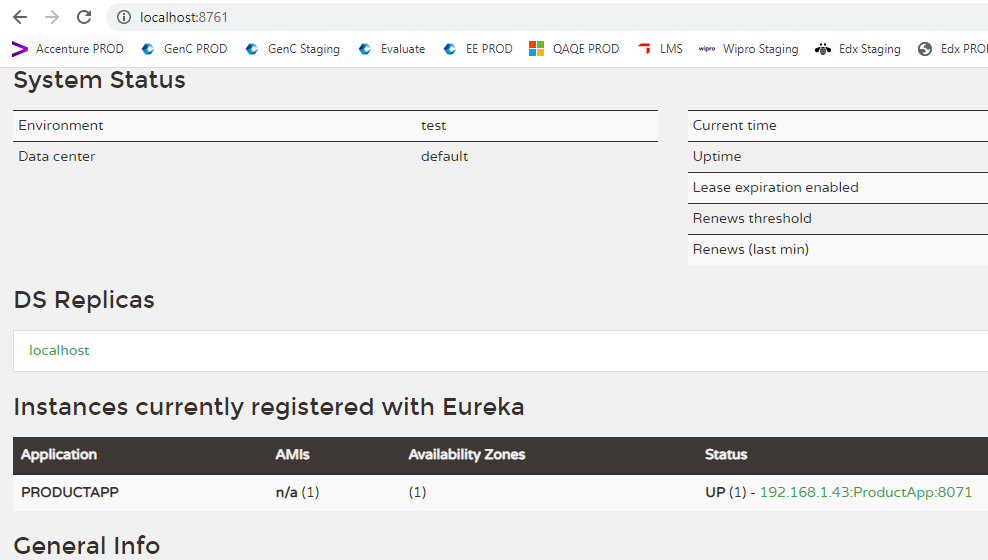
****



**Project 2:** This project should be a spring boot application containing all the REST services implemented as per the requirements stated in the case study. These services must be registered with Eureka-Server. Run the services in port 8071.Services should get automatically registered with the Eureka Registry After the service is registered,

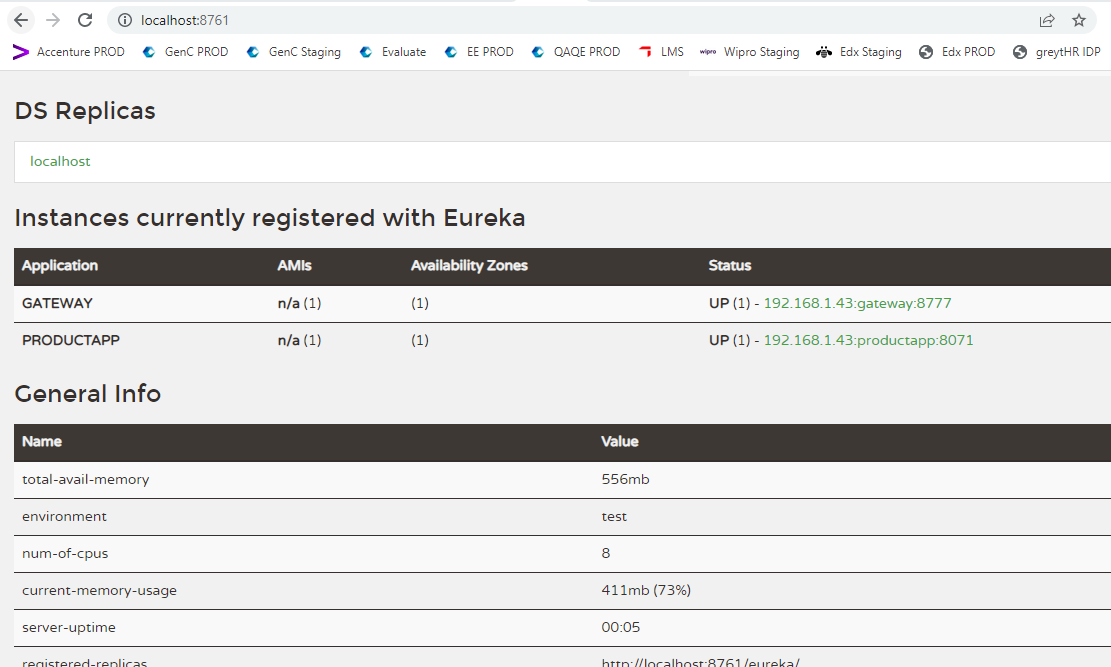
1. **Execute the ProductApp**
   1. Right click on the Registry project -> click Run As -> Click Spring Boot App

****

****

**Project 3:**

This must be a Spring cloud gateway project that contains the routing implementation to the actual services running in port 8071. Run this in port 8777

****

**Important Note:** In Tekstac platform the Eureka server is already running in port number 8760. So you can just drag and drop only the ProductApp and RouterApp applications to platform.

**Output:**

You can use postman to check the correctness of the code.

