**Register App in Spring Cloud**

Implement using Spring Cloud (Eureka) as a service registry and Spring Boot for REST Service. This application is a rest service application.

**Functionalities:**

The Product Application is developed and provided as part of the code skeleton. 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 8760 and the URL for the same is given below:

**http://webapps.tekstac.com:8760/eureka/**

Help them to automate the above process by developing a Rest Service using Maven and registering the given REST service in the eureka server.

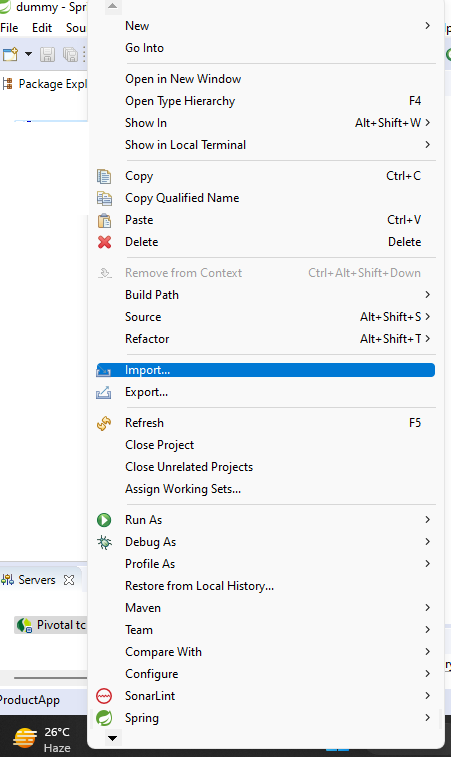
In STS create the 2 following projects,

* ProductApp
* Registry – to register the ProductApp application in the Eureka Server.

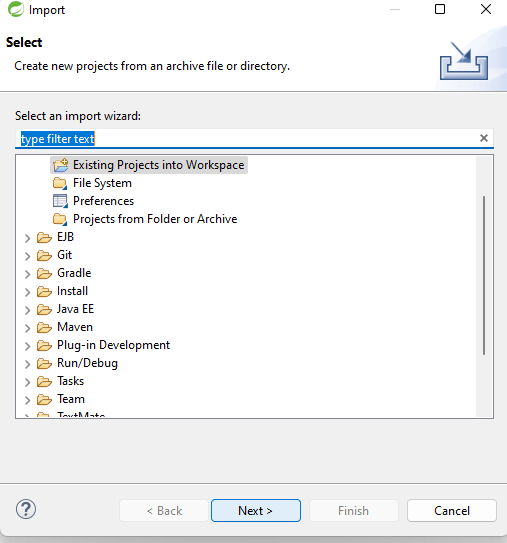
**Note:**

If the code skeleton is provided as a part of question description, you can just download and import it in STS as shown below,

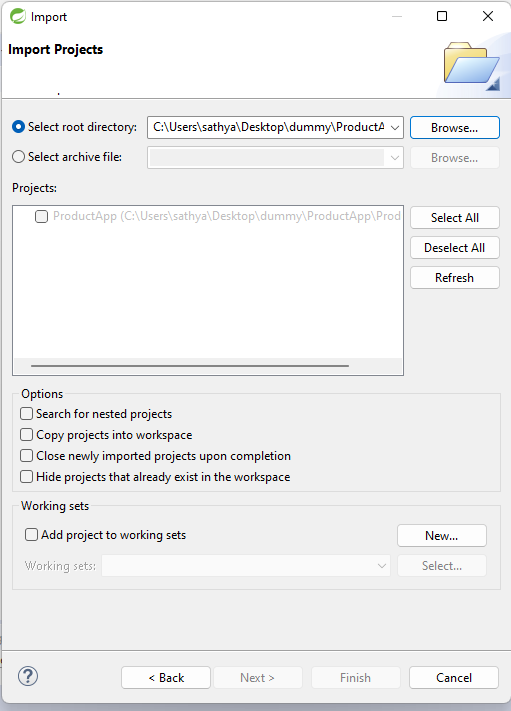
* Right click on package explorer and select import



* Select -> existing projects into workspace



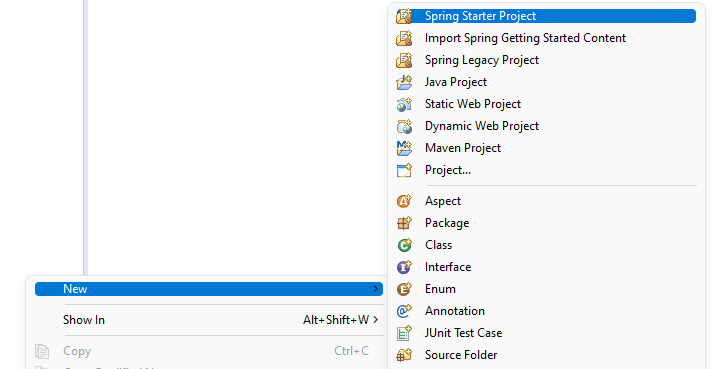
* Choose the project -> click finish



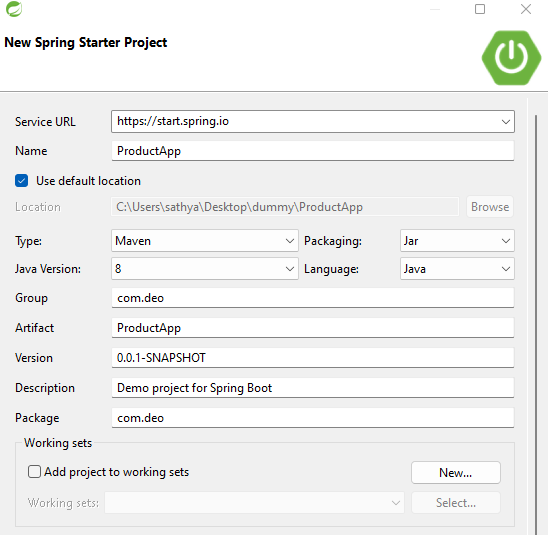
**If code skeleton is not provided, create new project as shown below**

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

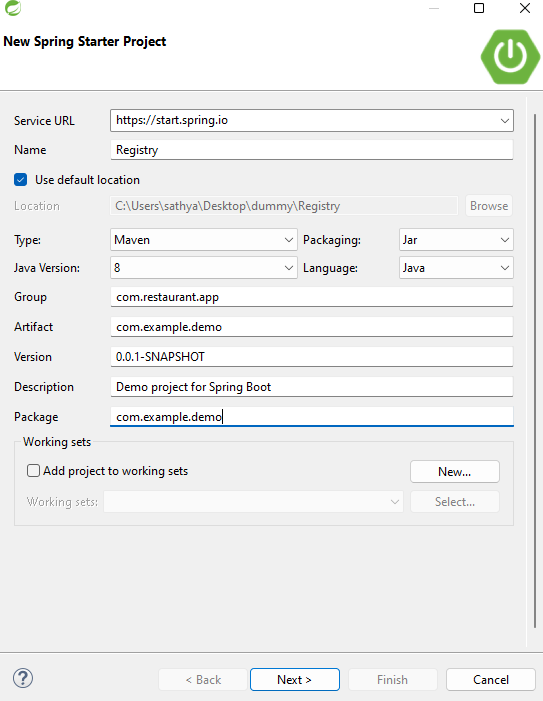
1. Open Spring Tool Suite (STS) and click on File > New > Spring Starter Project. Alternatively, you can also click on the Spring icon on the toolbar and select Spring Starter Project from the drop-down menu.



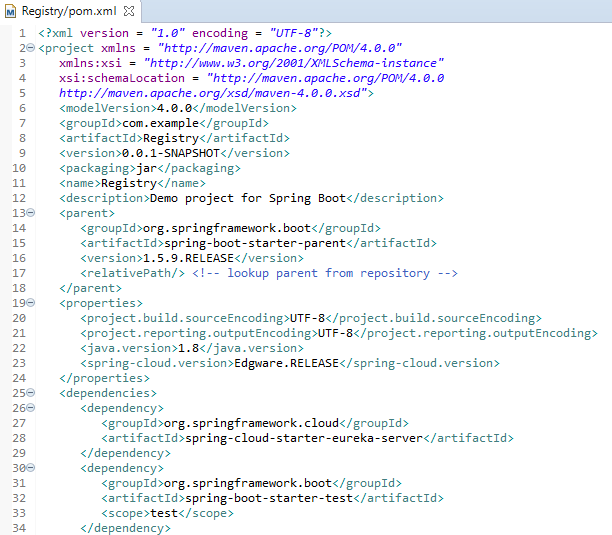
1. In the New Spring Starter Project wizard, you will be prompted to enter the following information:
   1. **Project Name:** Enter a name for your project.
   2. **Group:** Enter the group name for your project.
   3. **Artifact:** Enter the artifact name for your project.

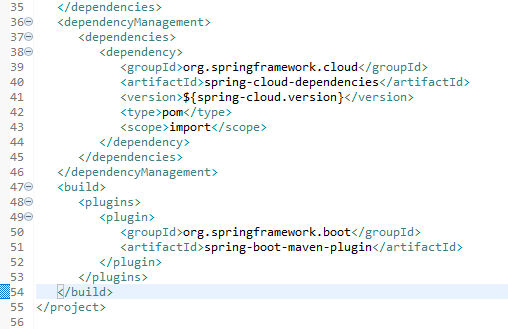


1. Click on the Finish button to create the project. STS will create the project with the specified dependencies and generate the necessary files.
2. Follow the above steps to create another project called **Registry.**

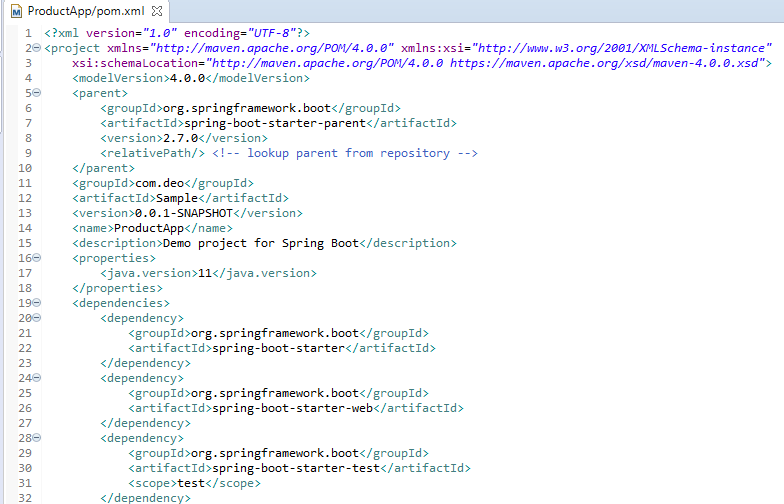


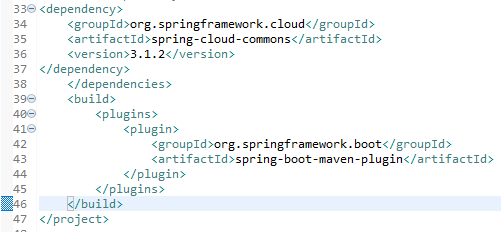
1. Once the project is created, add the following dependency in the Registry pom.xml file.





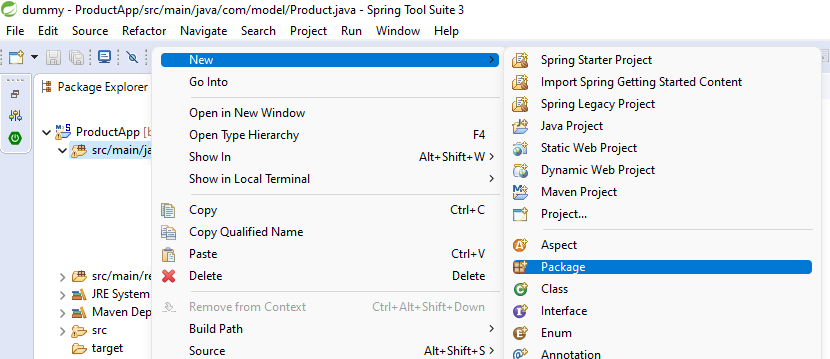
1. Once the project is created, you can open the project in the Package Explorer view and explore the files that have been generated. You can also start coding your application.
2. Once the projects are created, add the given dependency in the ProductApp pom.xml file.



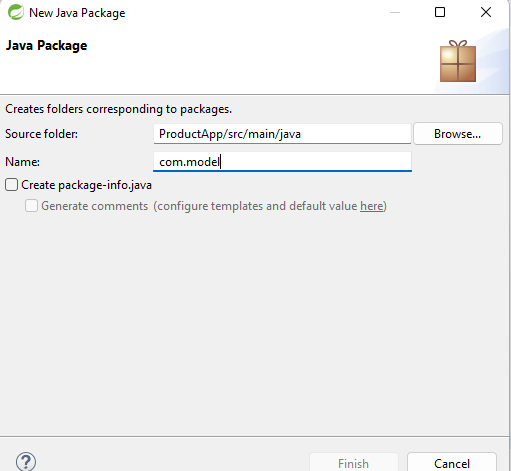


**Requirements:**

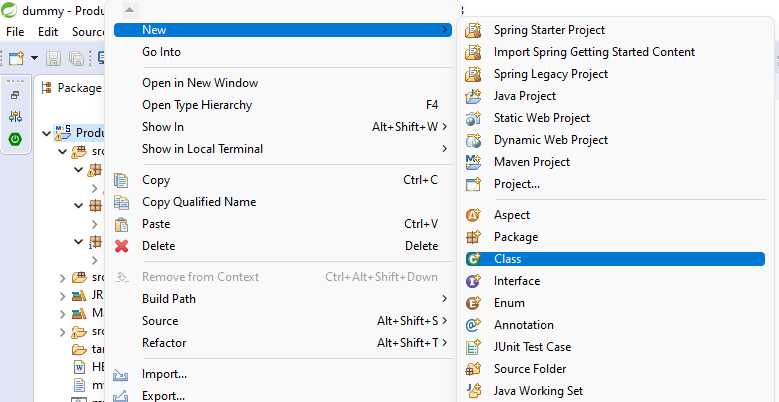
1. Right-click on the ProductApp project in the Package Explorer and select New -> Package.



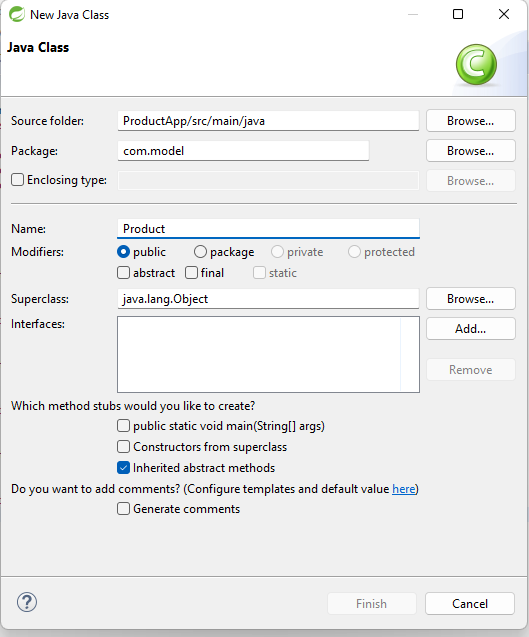
1. In the New Package dialog box, enter the name of the package as **com.model**



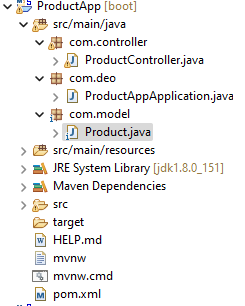
1. Click Finish to create the new package.
2. Once you have created the package, you can create new classes in it by right-clicking on the package and selecting New -> Class.



1. In the New Java Class dialog box, enter the name of the class **Product** and click Finish.

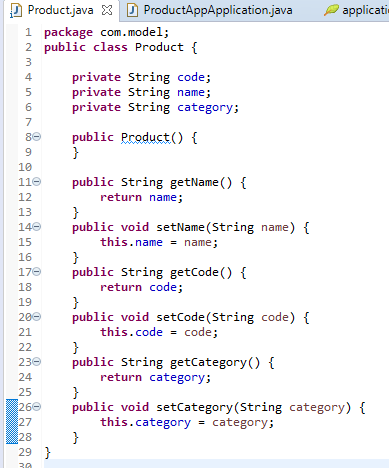


1. Follow the above steps to create other packages and classes required for this project.

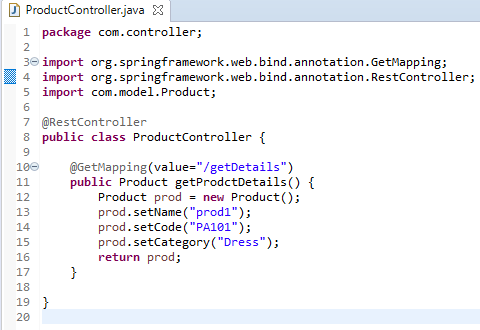


1. Once the **Product** class is created, include the following attributes. Include the required constructor, getter and setter methods.

|  |  |
| --- | --- |
| **Attributes** | **DataType** |
| code | String |
| name | String |
| category | String |

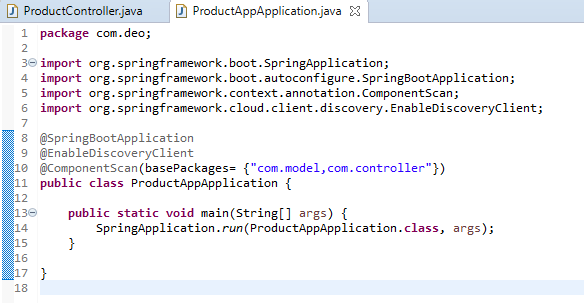


1. In the ProductController class include the code which is provided as a part of the code skeleton.



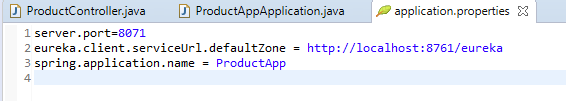
* 1. The **com.controller** package contains the **ProductController** class, which is annotated with **@RestController**. This annotation indicates that this class is responsible for handling HTTP requests and returning responses in a RESTful manner.
  2. The **ProductController** class contains a single method named **getProdctDetails**(), which is annotated with **@GetMapping(value="/getDetails").** This annotation specifies that this method should handle HTTP GET requests to the /getDetails endpoint.
  3. Inside the **getProdctDetails**() method, a new instance of the Product class is created, and its properties are set using the **setName**(), **setCode**(), and **setCategory**() methods. Finally, the Product object is returned as a response to the HTTP request.

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



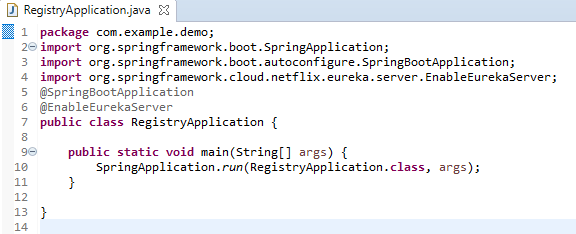
* 1. @**SpringBootApplication** is an annotation that tells the Spring Boot to scan this class and all of its sub-packages for components and configuration.
  2. @**ComponentScan** is an annotation that specifies the package(s) that Spring should scan to find the components that will be used by this 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 the **application.properties** file specify the server port number, and application name and set the URL for the Eureka server that this application will register with.



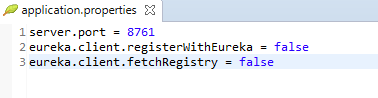
* 1. **"server.port=8071",** this line sets the server port to 8071. This means that the application will listen for incoming HTTP requests on port 8071.
  2. **"eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka",** this line sets the URL for the Eureka server that this application will 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 an identifier for the application when it registers with the Eureka server. In this case, the application name is set to "ProductApp".

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



* 1. @**SpringBootApplication** is an annotation that tells the 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 need to be added in the application.properties file.

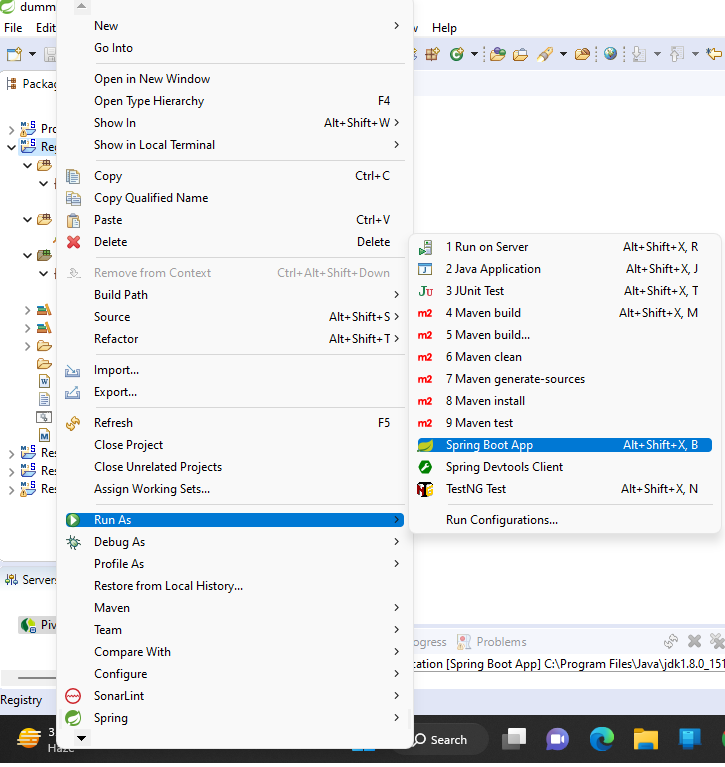


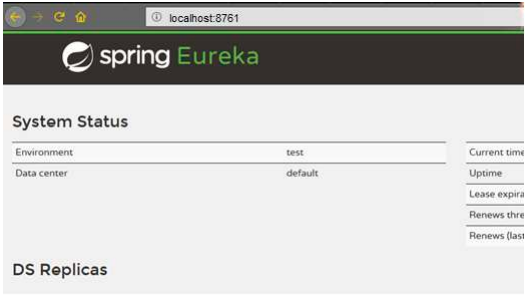
* 1. "**server.port=8761**", this line sets the server port to 8761. This means that the application will listen for incoming HTTP requests on port 8761.
  2. "**eureka.client.registerWithEureka=false**", this line disables automatic registration of the application with the Eureka server. When this property 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.

**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

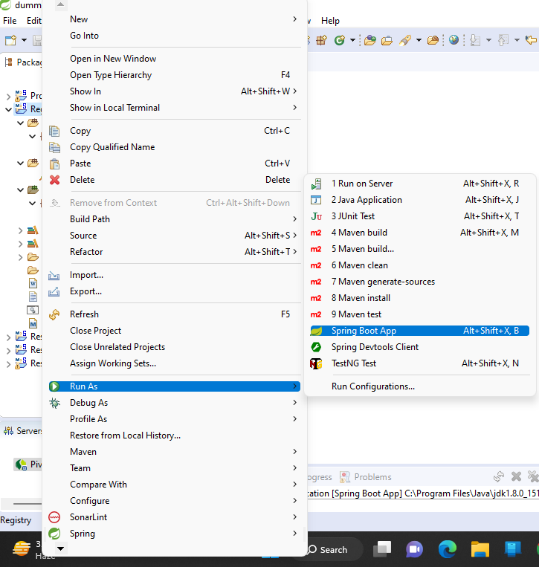
1. **Execute the Registry** 
   1. Right click on the Registry project -> click Run As -> Click Spring Boot App

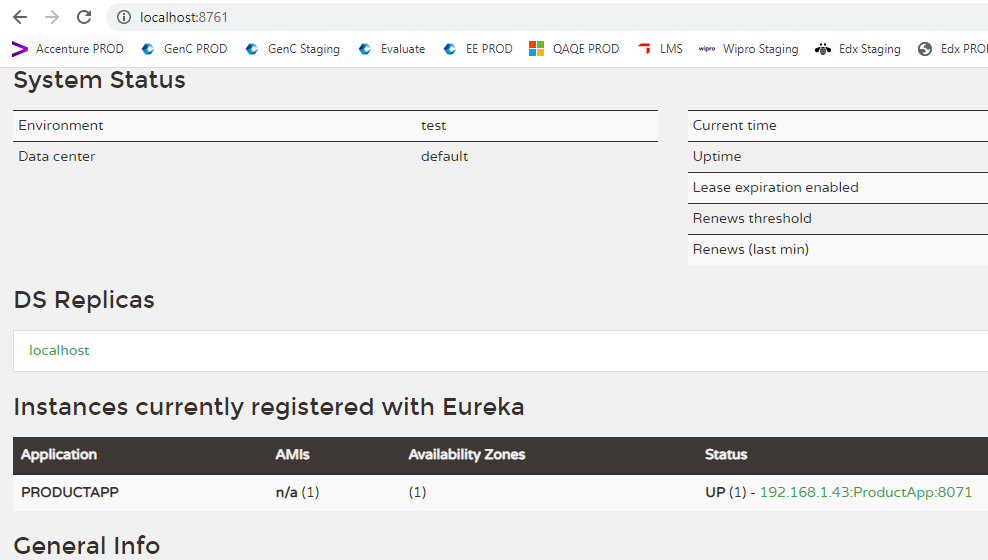
****



**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

****

****

**Important Note:** In the Tekstac platform the Eureka server is already running in port number 8760. So you can just drag and drop only the ProductApp application to the platform by modifying the application.properties file as shown below,

