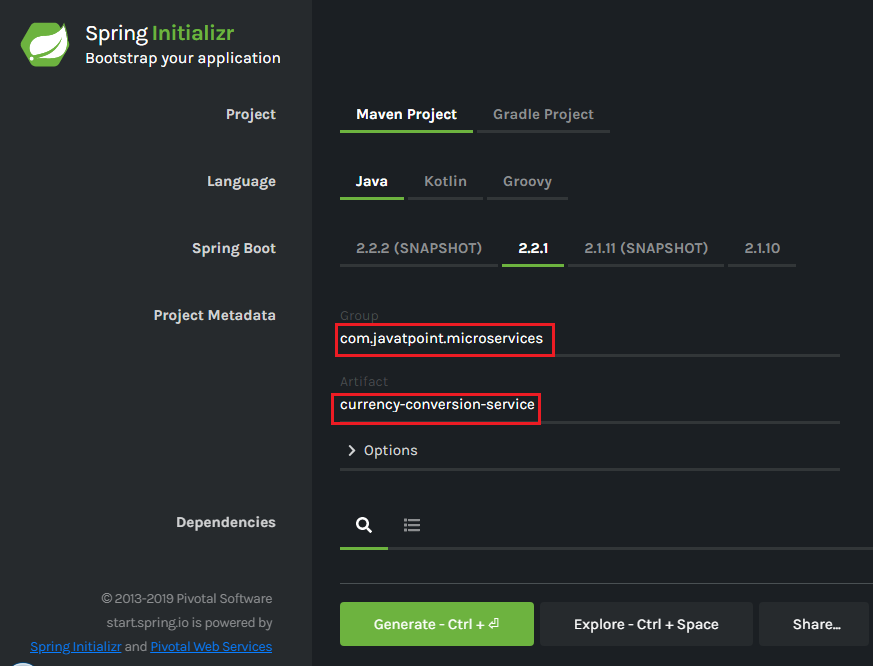
Setting up Currency Conversion Microservice

In the previous section, we have created currency-exchange-service. Now we will create a currency-conversion-service that talks to currency-exchange-service.

**Step 1:**Open the browser and type https://start.spring.io/.

* Provide the Group name **javatpoint.microservice** and Artifact **currency-conversion-service.**
* Add the dependencies: **Spring web, DevTools, Actuator,**and **Config Client**.
* Click on the **Generate** It downloads the created project.



**Step 2**: Import the downloaded project in **Spring Tool Suite (STS)**.

File -> Import -> Existing Maven Projects -> Next -> Browse -> Select the project -> Finish.

It takes some time to import the project.

**Step 3:** Open the **application.properties** file and configure the **application name** and **port**number.

**application.properties**

1. spring.application.name=currency-conversion-service
2. server.port=8100

The currency-conversion-service runs on port **8100**.

Setting up Currency Conversion Microservice

In the next section, we will create a service that talks to the currency-exchange-service.

Creating a Service for currency-conversion-service

In the previous section, we have used EUR to INR that returns what the **conversionMultiple**is. The currency-exchange-service tells what is the exchange value when we convert currency from EUR to INR.

In this section, we will create CurrencyCalculationService. It defines a lot of functionality related to conversion.

We will create a service currency-converter that accepts two path parameters "**from**" and "**to**". It also accepts the quantity (amount which we want to convert).

Let's create a currency-conversion-service.

**Step 1:** Create a class with the name **CurrencyConversionController**.

**Step 2:**Add an annotation **@RestController.**

**Step 3:**Create a **GetMapping**.

**CurrencyConversionController.java**

1. **package** com.javatpoint.microservices.currencyconversionservice;
2. **import** java.math.BigDecimal;
3. **import** org.springframework.web.bind.annotation.GetMapping;
4. **import** org.springframework.web.bind.annotation.PathVariable;
5. **import** org.springframework.web.bind.annotation.RestController;
6. @RestController
7. **public** **class** CurrencyConversionController
8. {
9. @GetMapping("/currency-converter/from/{from}/to/{to}/ quantity/{quantity}") //where {from} and {to} represents the column
10. //return a bean back
11. **public** CurrencyConversionBean convertCurrency(@PathVariable String from, @PathVariable String to, @PathVariable BigDecimal quantity)
12. {
13. **return** **new** CurrencyConversionBean(1L, from,to,BigDecimal.ONE, quantity,quantity,0 );
14. }
15. }

**Step 4:**Create a class with the name **CurrencyConversionBean** and define the following fields:

1. **private** Long id;
2. **private** String from;
3. **private** String to;
4. **private** BigDecimal ConversionMultiple;
5. **private** BigDecimal quantity;
6. **private** BigDecimal totalCalculatedAmount;
7. **private** **int** port;

**Step 5:** Generate **Getters** and **Setters**.

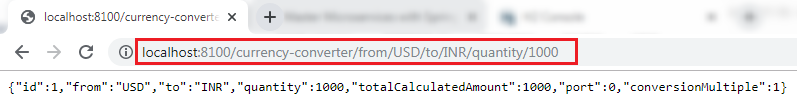
**Step 6:** Generate **constructor** and also create a **default** constructor.

**CurrencyConversionBean.java**

1. **package** com.javatpoint.microservices.currencyconversionservice;
2. **import** java.math.BigDecimal;
3. **public** **class** CurrencyConversionBean
4. {
5. **private** Long id;
6. **private** String from;
7. **private** String to;
8. **private** BigDecimal ConversionMultiple;
9. **private** BigDecimal quantity;
10. **private** BigDecimal totalCalculatedAmount;
11. **private** **int** port;
12. //default constructor
13. **public** CurrencyConversionBean()
14. {
15. }
16. //creating constructor
17. **public** CurrencyConversionBean(Long id, String from, String to, BigDecimal conversionMultiple, BigDecimal quantity, BigDecimal totalCalculatedAmount, **int** port)
18. {
19. **super**();
20. **this**.id = id;
21. **this**.from = from;
22. **this**.to = to;
23. ConversionMultiple = conversionMultiple;
24. **this**.quantity = quantity;
25. **this**.totalCalculatedAmount = totalCalculatedAmount;
26. **this**.port = port;
27. }
28. //creating setters and getters
29. **public** Long getId()
30. {
31. **return** id;
32. }
33. **public** **void** setId(Long id)
34. {
35. **this**.id = id;
36. }
37. **public** String getFrom()
38. {
39. **return** from;
40. }
41. **public** **void** setFrom(String from)
42. {
43. **this**.from = from;
44. }
45. **public** String getTo()
46. {
47. **return** to;
48. }
49. **public** **void** setTo(String to)
50. {
51. **this**.to = to;
52. }
53. **public** BigDecimal getConversionMultiple()
54. {
55. **return** ConversionMultiple;
56. }
57. **public** **void** setConversionMultiple(BigDecimal conversionMultiple)
58. {
59. ConversionMultiple = conversionMultiple;
60. }
61. **public** BigDecimal getQuantity()
62. {
63. **return** quantity;
64. }
65. **public** **void** setQuantity(BigDecimal quantity)
66. {
67. **this**.quantity = quantity;
68. }
69. **public** BigDecimal getTotalCalculatedAmount()
70. {
71. **return** totalCalculatedAmount;
72. }
73. **public** **void** setTotalCalculatedAmount(BigDecimal totalCalculatedAmount)
74. {
75. **this**.totalCalculatedAmount = totalCalculatedAmount;
76. }
77. **public** **int** getPort()
78. {
79. **return** port;
80. }
81. **public** **void** setPort(**int** port)
82. {
83. **this**.port = port;
84. }
85. }

**Step 7:** Restart the application and type the following URI in the browser:

**http://localhost:8100/currency-converter/from/USD/to/INR/quantity/1000**



In the above response, "**from,"** "**to,"** and "**quantity"** variables picked up from the path. We have hardcoded the other variables.

In the next step, from the currency-conversion-service, we will call the currency-exchange-service. We will also determine what the **conversion multiple** is, and will use that amount (**conversion multiple**) to calculate the **total** amount. We will also use the port that comes in the response.