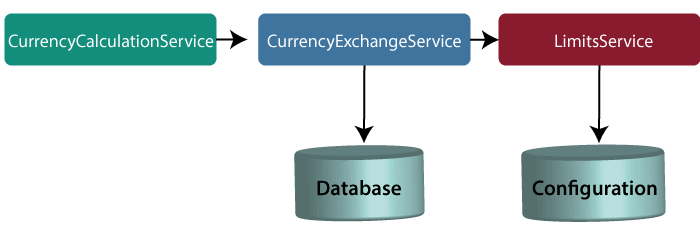
Introduction to Currency Conversion and Currency Exchange Service

In this section, we will create a couple of microservices: **CurrencyCalculationService** and **CurrencyExchangeService**.



*Note: In this tutorial, we have quoted currency conversion service as a currency calculation service. Both the services have the same meaning, so don't be confused.*

Let's understand the functionality of these services.

In the above figure, the CurrencyExchangeService uses JPA to talk to the database and returns the exchange value of the specific currency. For example, USD to INR conversion.

When we invoke CurrencyExchangeService, we need to pass two parameters: **from**(convert from), and **to** (convert to). For example, if we want to convert currency from **USD** to **INR**.

Consider the URL **http://localhost:8000/currency-exchange/from/USD/to/INR**. It retunes the following response:

1. {
2. id: 101,
3. from: "USD",
4. to: "INR",
5. conversionMultiple: 72,
6. port: 8000
7. }

The currency exchange service will return what the conversion multiple is. The conversion multiple means **1 USD** is equal to **72 INR**. The currency converter service uses a currency exchange service. Suppose the currency converter service wants to convert 100 USD to INR. So it will call the currency exchange service and will convert the specified amount that we have provided in the path parameter. For example:

**http://localhost:8100/currency-converter/from/USD/to/INR/quantity/100**

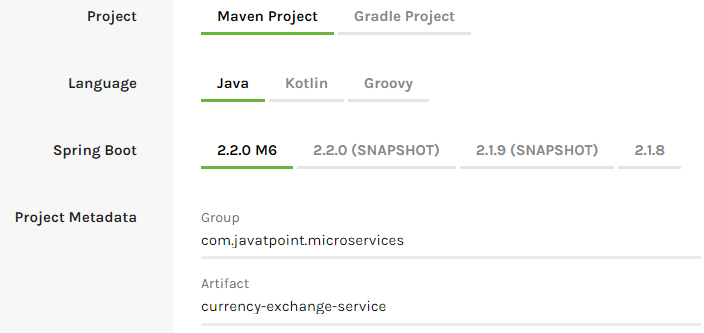
1. {
2. Id: 101,
3. from: "USD",
4. to: "INR",
5. conversionMultiple: 72,
6. quantity: 100
7. totalCalculatedAmount: 7200,
8. port: 8000
9. }

We will implement these two services in our example using Spring Cloud.

Setting up a currency-exchange-service

**Step 1:**Open the spring initializer http://start.spring.ios.

**Step 2:**Select the **Project**: Maven Project, **Language:**Java, and Spring Boot version **2.2.0 M6**or above. Provide the **Group name** and **Artifact ID.**We have provided**com.javatpoint.microservices**and **currency-exchange-service,**for group name and Artifact id respectively.



**Step 3:**Add the dependencies **Web, DevTools, Actuator,**and **Config Client**.

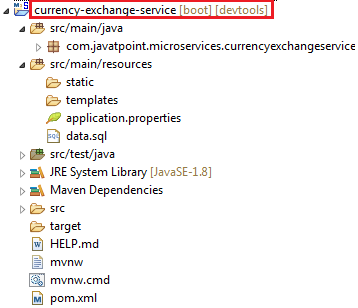
**Step 4:**Click on the **Generate Project** button. It will download the **zip** file of the project.

**Step 5: Extract** it in the local disk.

**Step 6: Import** the project.

Click on File menu-> Import -> Existing Maven Projects -> Next -> Browse ->Select the project ->Finish

It takes some time to import. When the project import is done, it shows the following project directory. Do not consider the data.sql file in the directory, because we will create it later.



**Step 7:**Open the **application.properties** file and configure the **application name** and **port** number.

**application.properties**

1. spring.application.name=currency-exchange-service.
2. server.port=8000

When we run the currency-exchange-service, it runs but does not perform any service. In the next step, we will implement code in the currency-exchange-service.

Hardcoded the currency-exchange-service

Now we will create a service that converts the currency from USD to INR.

**Step 1:**Create a class file (REST Controller) with the name **CurrencyExchangeController** in the package **com.javatpoint.microservices.currencyexchangeservice.**

**CurrencyExchangeController.java**

1. **package** com.javatpoint.microservices.currencyexchangeservice;
2. **import** java.math.BigDecimal;
3. **import** org.springframework.boot.SpringApplication;
4. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
5. **import** org.springframework.web.bind.annotation.GetMapping;
6. **import** org.springframework.web.bind.annotation.PathVariable;
7. **import** org.springframework.web.bind.annotation.RestController;
8. @SpringBootApplication
9. @RestController
10. **public** **class** CurrencyExchangeController
11. {
12. @GetMapping("/currency-exchange/from/{from}/to/{to}")       //where {from} and {to} are path variable
13. **public** ExchangeValue retrieveExchangeValue(@PathVariable String from, @PathVariable String to)  //from map to USD and to map to INR
14. {
15. **return** **new**  ExchangeValue(1000L, from, to, BigDecimal.valueOf(65));
16. }
17. }

**Step 2:**Create a class file with the name **ExchangeValue.**

**ExchangeValue.java**

1. **package** com.javatpoint.microservices.currencyexchangeservice;
2. **import** java.math.BigDecimal;
3. **public** **class** ExchangeValue
4. {
5. **private** Long id;
6. **private** String from;
7. **private** String to;
8. **private** BigDecimal conversionMultiple;
10. **public** ExchangeValue()
11. {
12. }
13. //generating constructor using fields
14. **public** ExchangeValue(Long id, String from, String to, BigDecimal conversionMultiple) {
15. **super**();
16. **this**.id = id;
17. **this**.from = from;
18. **this**.to = to;
19. **this**.conversionMultiple = conversionMultiple;
20. }
21. //generating getters
22. **public** Long getId()
23. {
24. **return** id;
25. }
26. **public** String getFrom()
27. {
28. **return** from;
29. }
30. **public** String getTo()
31. {
32. **return** to;
33. }
34. **public** BigDecimal getConversionMultiple()
35. {
36. **return** conversionMultiple;
37. }
38. }

**Step 3:**Run the **CurrencyExchangeServiceApplication.java.**It runs on the port **8000** that we have configured in the application.properties file.

We get the following response on the browser:

1. {
2. id: 101,
3. from: "USD",
4. to: "INR",
5. conversionMultiple: 72,
6. port: 8000
7. }

Setting up Dynamic port in the Response

The CurrencyExchangeService determines the exchange value of the currency. The CurrencyCalculationService uses the CurrencyExchangeService to determine the value of one currency in other currency. We will create multiple instances of the **CurrencyExchangeService** later in next topic.

At present, the service is running on port **8000**. Later we will run it on port **8001, 8002,** and so on. In the next step, we will set a port to the currency-exchange-service.

**Step 1:** Open the **ExchangeValue.java** file and add a **port**variable. Generate getters and setters for the port variable only.

**ExchangeValue.java**

1. **package** com.javatpoint.microservices.currencyexchangeservice;
2. **import** java.math.BigDecimal;
3. **public** **class** ExchangeValue
4. {
5. **private** Long id;
6. **private** String from;
7. **private** String to;
8. **private** BigDecimal conversionMultiple;
9. **private** **int** port;
10. **public** ExchangeValue()
11. {
12. }
13. //generating constructor using fields
14. **public** ExchangeValue(Long id, String from, String to, BigDecimal conversionMultiple) {
15. **super**();
16. **this**.id = id;
17. **this**.from = from;
18. **this**.to = to;
19. **this**.conversionMultiple = conversionMultiple;
20. }
21. //generating getters
22. **public** **int** getPort() {
23. **return** port;
24. }
25. **public** **void** setPort(**int** port) {
26. **this**.port = port;
27. }
28. **public** Long getId()
29. {
30. **return** id;
31. }
32. **public** String getFrom()
33. {
34. **return** from;
35. }
36. **public** String getTo()
37. {
38. **return** to;
39. }
40. **public** BigDecimal getConversionMultiple()
41. {
42. **return** conversionMultiple;
43. }
44. }

We have already configured the application name and port number in the application.properties file, so need not to configure again.

Now pick up port number from the environment.

**Step 3**: Open the **CurrencyExchangeController.java** and get the property of the environment.

**CurrencyExchangeController.java.**

1. **package** com.javatpoint.microservices.currencyexchangeservice;
2. **import** java.math.BigDecimal;
3. **import** org.springframework.beans.factory.annotation.Autowired;
4. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
5. **import** org.springframework.core.env.Environment;
6. **import** org.springframework.web.bind.annotation.GetMapping;
7. **import** org.springframework.web.bind.annotation.PathVariable;
8. **import** org.springframework.web.bind.annotation.RestController;
9. @SpringBootApplication
10. @RestController
11. **public** **class** CurrencyExchangeController
12. {
13. @Autowired
14. **private** Environment environment;
15. @GetMapping("/currency-exchange/from/{from}/to/{to}") //where {from} and {to} are path variable
16. **public** ExchangeValue retrieveExchangeValue(@PathVariable String from, @PathVariable String to)  //from map to USD and to map to INR
17. {
18. //taking the exchange value
19. ExchangeValue exchangeValue= **new** ExchangeValue (1000L, from, to, BigDecimal.valueOf(65));
20. //picking port from the environment
21. exchangeValue.setPort(Integer.parseInt(environment.getProperty("local.server.port")));
22. **return** exchangeValue;
23. }
24. }

When we refresh the browser, the URL changes to: **http://localhost:8000/currency-exchange/from/USD/to/INR**.

1. {
2. id: 1000,
3. from: "USD",
4. to: "INR"
5. conversionMultiple: 65,
6. port: 8000
7. }

At present **CurrencyExchangeServiceApplication** is running on port **8000**.

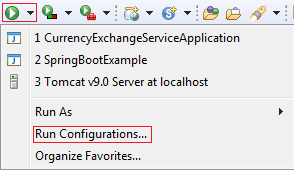
Now we will run **CurrencyExchangeServiceApplication** on a different port number. For this, we have to change the port in the **application.properties**file from 8000 to 8001, 8002, etc. whichever we want.

Suppose we want to create two instances of the **CurrencyExchangeServiceApplication**. For this, we have to set port externally.

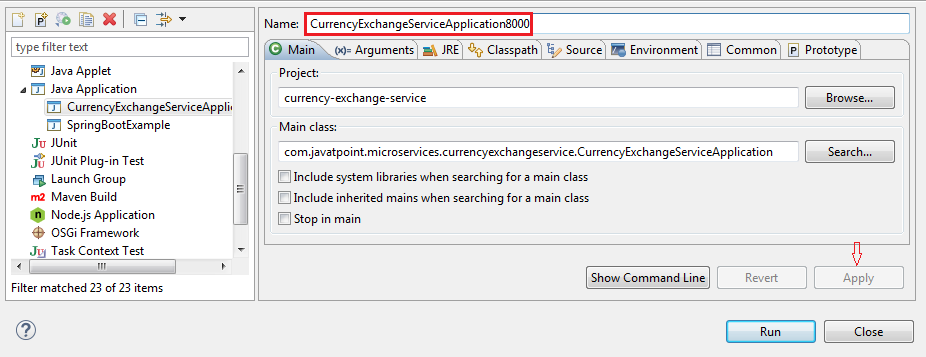
Let's create an instance of the **CurrencyExchangeServiceApplication**that runs on the port **8001**.

**Step 1:**Right-click on the project -> Run As -> Run Configurations.

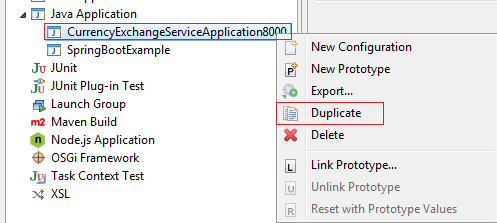
Or click on the highlighted symbol -> Run Configurations.



**Step 2: Rename**the**CurrencyExchangeServiceAppication** to**CurrencyExchangeServiceAppication8000**and click on the**Apply**button.



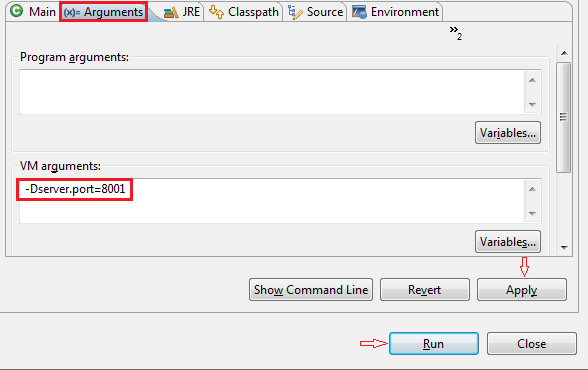
**Step 3:**Right-click on the **CurrencyExchangeServiceApplication8000**-> Duplicate.



It generates the duplicate file of **CurrencyExchangeServiceApplication8000.**We will run it on port **8001.**

**Step 4:**Click on the **Arguments** tab and write **–Dserver.port=8001**in the **VM arguments** text box. Click on the **Apply**and **Run** button, respectively.

*Note: Whatever value we are passing in the VM arguments, it overwrites the configuration of the application.properties file.*



After clicking on the **Run**button, it starts running on port **8001**.

**Step 5:**Change the port number in the URL **http://localhost:8001/currency-exchange/from/USD/to/INR** and press enter key. We get the following response:

1. {
2. id: 1000,
3. from: "USD",
4. to: "INR",
5. conversionMultiple: 65,
6. port: 8001
7. }

Now we have two instances of **CurrencyExchangeServiceApplication**that are running on two different ports **8000** and **8001**.

[Click here to download currency-exchange-service](https://d2jdgazzki9vjm.cloudfront.net/tutorial/microservices/download/currency-exchange-service.zip)