**Micro Services Problem statement**

**Part-1 – Micro Services creation**

1. Create following 4 micro service applications
   1. Product-Catalog-Service
   2. Price-Service
   3. Cart-Service
   4. Order-Service
2. Configure following port number for above created applications
   1. Product-Catalog-Service : 8081
   2. Price-Service : 8082
   3. Cart-Service : 8083
   4. Order-Service : 8084

Add following business scenarios for the above-mentioned micro services

1. **Product-Catalog-Service** : Create a RestController for adding a single product as JSON, or for adding multiple products as Array of JSON objects. Details for Product object properties are given below.

Table/Collection name : **Product**

Properties:

@Id

@GeneratedValue

private int id;

private String name;

private double price;

private String description;

Table/Collection name: **Review**

Properties:

@Id

@GeneratedValue

private int reviewId;

private int stars;

private String author;

private String body;

@ManyToOne

@JoinColumn(name = "id")

private Product product;

Now Create end point url to retrieve Product and Reviews information

Finding all products

**GET** request: http://localhost:8081/api/products

[

{

"id": 1,

"name": "Pen",

"price": 25,

"description": "Red Ink"

},

{

"id": 2,

"name": "Mobile",

"price": 16000,

"description": "Samsung A7 Mobile"

}

]

Finding specified product. (ex: finding a product whose id is 1)

**GET** request: http://localhost:8081/api/products/1

{

"id": 1,

"name": "Pen",

"price": 25,

"description": "Red Ink"

}

Finding all products by product name.(ex: any product having a letter ‘e’)

**GET** request: http://localhost:8081/api/products/byName/e

[

{

"id": 1,

"name": "Pen",

"price": 25,

"description": "Red Ink"

},

{

"id": 2,

"name": "Mobile",

"price": 16000,

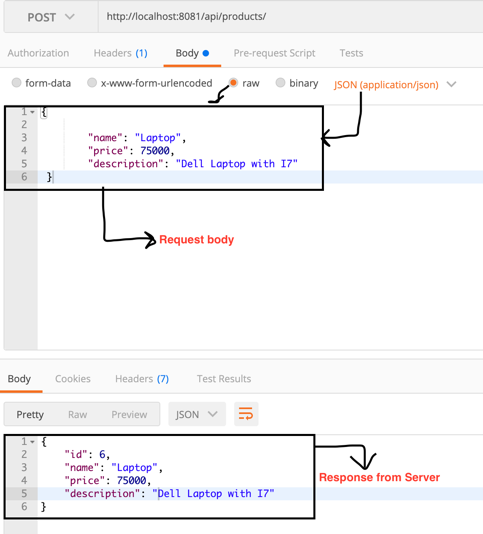
"description": "Samsung A7 Mobile"

}

]

Posting a single product

**POST** request : <http://localhost:8081/api/products>



Posting more than a product

**POST** request: http://localhost:8081/api/products/all



Finding a product with Reviews

**GET** request: http://localhost:8081/api/products/2/reviews

[

{

"stars": 5,

"author": "Praveen",

"body": "Very Good Mobile",

"product": {

"id": 2,

"name": "Mobile",

"price": 16000,

"description": "Samsung A7 Mobile"

},

"id": 3

},

{

"stars": 4,

"author": "Ozvitha",

"body": "Poor Display",

"product": {

"id": 2,

"name": "Mobile",

"price": 16000,

"description": "Samsung A7 Mobile"

},

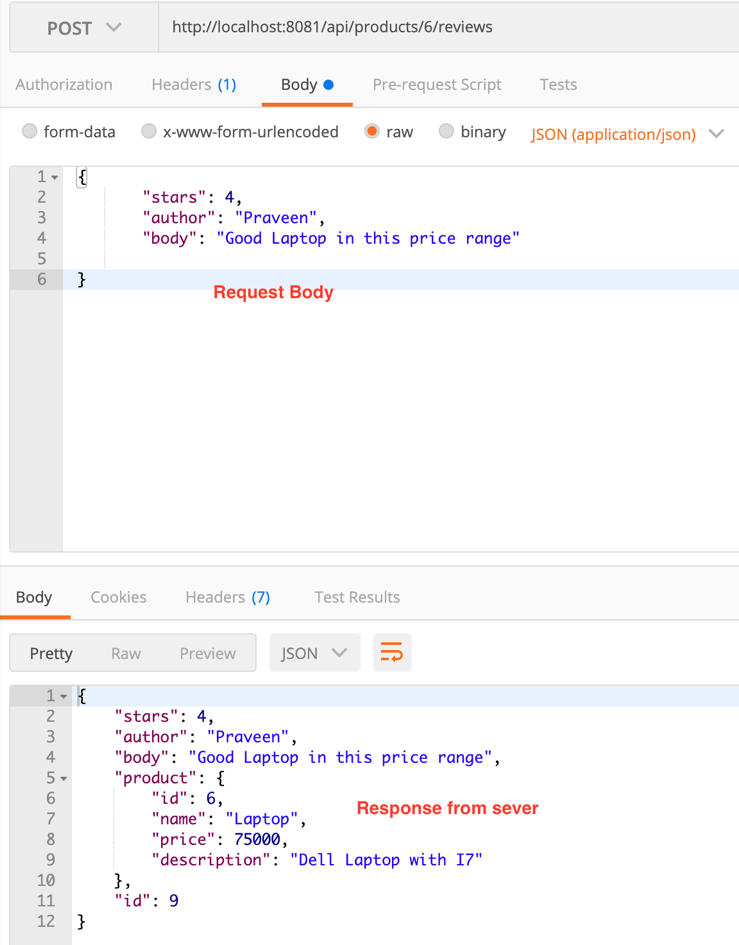
"id": 4

}

]

Posting a review for a specified product

**POST** request : <http://localhost:8081/api/products/6/reviews>



**Part-2 – Config Server, Microservices communication**

**Create a Config server with the following configuration.**

server.port=8888

spring.cloud.config.server.git.uri=https://github.com/ctsjava/osp-git.git

management.security.enabled=false

spring.application.name=spring-config-server-app

management.endpoints.web.exposure.include=\*

1. **Price-Service:** Add the following features for this service

Make sure database configuration for this service should be maintained by ConfigServer from github repository

Price-service application properties file

spring.application.name=price-service

server.port=8082

ConfigServer application properties file

server.port=8888

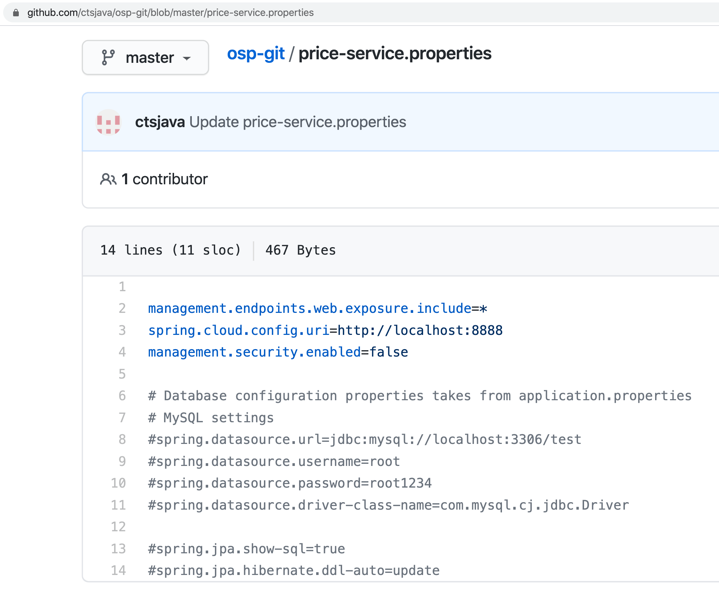
spring.cloud.config.server.git.uri=https://github.com/ctsjava/osp-git.git

management.security.enabled=false

spring.application.name=spring-config-server-app

management.endpoints.web.exposure.include=\*

price-service application database properties from github



Finding price for a given Product

**GET** request: http://localhost:8082/api/price/2

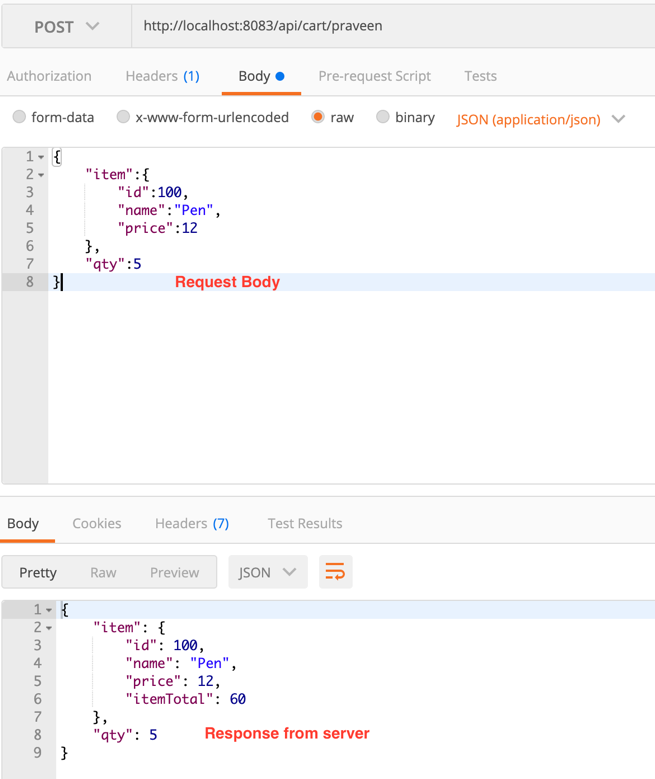
Should return price of product id 2

1. **Cart-Service:**  Add the following capabilities for this service

Add a Product Item to the cart with following end point url

**POST** request <http://localhost:8083/api/cart/praveen> (Here Praveen is the logged user )

Note: Passing price here is optional. You should get the price for the product id 100 from the Price-Service application. (You may use RestTemplate or FeignClient as per your convenience for micro services intercommunications)



1. **Order-Service** : Add the following capabilities for the order-service application

You have already added a Item in the cart. Now lets make an order for the items which are available in the cart only. You can not order any items which are not available in the cart. PFB the end point URL for ordering an Item or Items. Please make sure that you are passing only user name who added Product in the cart.

**POST** request. <http://localhost:8084/api/orders/praveen>

You should pull the cart items for the user Praveen from the Cart-Service application. Use Feign client or RestTemplate for the Microservices inter-communications.

When the above url is posted, Return the following response from the server and delete the ordered items from the cart for the user Praveen ion the Cart-Service application.

{

"id": 13,

"date": "2020-08-03T23:32:31.443",

"amount": 60,

"user": "praveen"

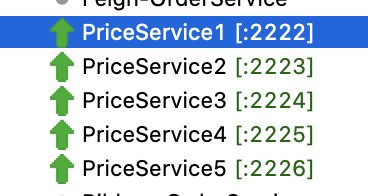
}

**Part-3 – Load Balancing with Ribbon**

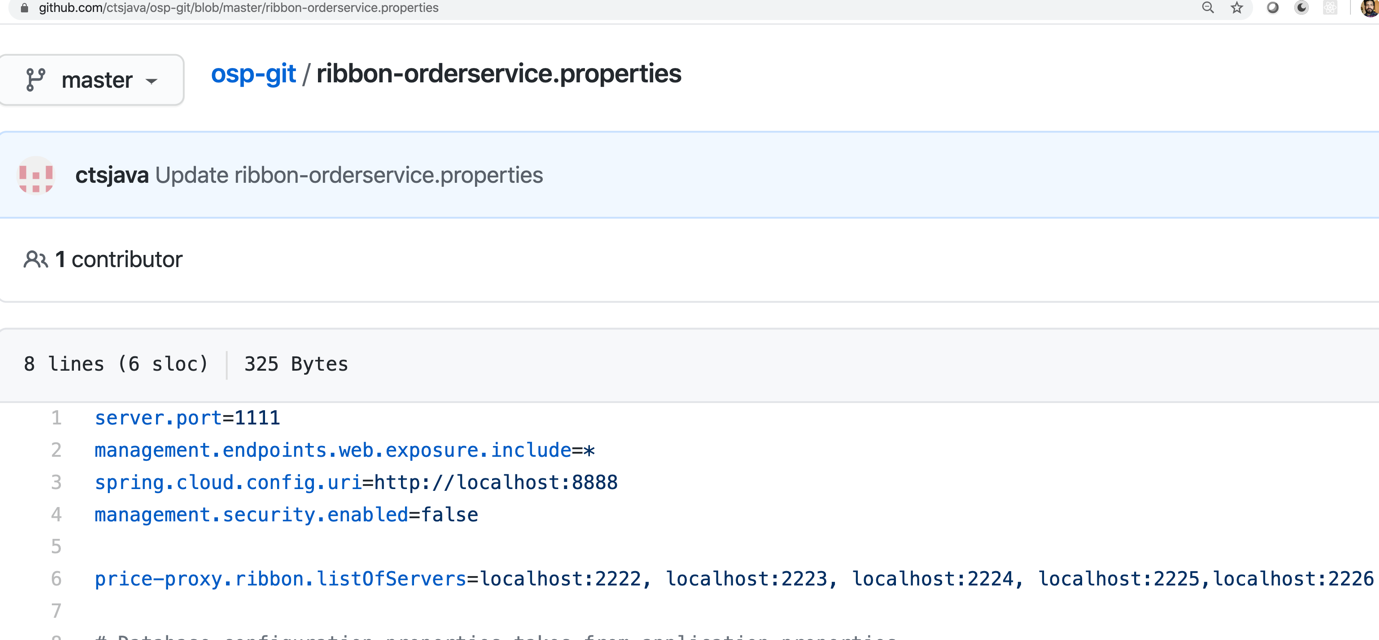
Create Ribbon Service application for the Order-Service with following configuration in the application.properties file.

**spring.application.name=ribbon-orderservice**

Now create 5 different Price-Service Instances with port numbers 2222, 2223,2224,2225 and 2226 as following diagram.



Now add Ribbon-Order-Service application properties in the github config server like following diagram

Please note that you have configured Above created 5 instances of Price-Service applications port numbers with Ribbon for load balancing.

server.port=1111

management.endpoints.web.exposure.include=\*

spring.cloud.config.uri=http://localhost:8888

management.security.enabled=false

price-proxy.ribbon.listOfServers=localhost:2222, localhost:2223, localhost:2224, localhost:2225,localhost:2226