**Coding Challenge Checkout**

Problem

Implement the code for a checkout system with a pricing schema in a supermarket. This exercise is not about framework know-how, but in order to make reviewing easier for us, please use:

1. Java
2. Maven or Gradle for the implementation.

This exercise describes pricing schemes and requirements for enhance ability for a system that calculates the total cost of a shopping cart based on a set of pricing rules. The items in a supermarket are identified by using Stock Keeping Units or SKUs. For this, we will use individual letters of the alphabet (e.g., A, B, C) and the items are priced individually. In addition to that some items are multi priced: buy n of them and they will cost you y Euro. For example: Item 'A' costs 40 Cent individually, but this week there is a special offer: buy three 'A's and they will cost you 1 Euro.

|  |  |  |
| --- | --- | --- |
| SKU | Unit Price | Special Price |
| A | 40 | 20% discount for 1 after 6 |
| B | 50 | Free shipping on 4 |
| C | 20 | After 10 buy 1 free |
| D | 10 | After 8 buy 2 free |

Check out should accept products with their stock keeping units in any order with their checkout quantity. Checkout should operate the pricing rules (special discounts) before sending us back the actual total.  
  
Solution

Before going into solution, I would like to list down the tech stack used in the development.  
This project is built for **idealo** internet GmbH, by keeping in mind the flexibility, resilience, fault-tolerance and S**O**LID principle.  
 **Tech Stack**

* **Spring cloud (for microservices)**
* **Spring Boot 2**
* **Spring Boot start for JPA**
* **Eureka Server (Discovery server)**
* **Eureka Client (Microservices)**
* **Netflix openFeign (Declarative Http Client)**
* **Netflix Ribbon (Load balancer used by openFeign)**
* **Netflix Hystrix (Circuit breaker used by openFeign)**
* **MapStruct**
* **Swagger parser**
* **Apache common language 3**
* **Junit5 (for unit tests)**
* **Junit4 (for integration tests)**
* **Mockito (used in both unit and integration tests)**
* **Apache Derby (embedded database)**
* **H2 (memory database for testing)**
* **Openpojo (for injected method testing)**

**Note:**No external (server) database is used for this project, instead serverless derby is used, so that reviewer will not need any external configurations or Database setup during reviewing. H2 is only use while testing, and to make it more simple application.properties override is provided in test/resources.

**Design pattern**

I used couple of design pattern during development of this assignment.

* **SOL**ID (**SOL,** three are used from SOLID principle)
* **Factory pattern**
* **Initialization on demand**
* **Builder pattern**

**Common Object Creation**

There are some common objects which are across the microservices, instead of creating them on multiple places and having code redundancy, I used the help of some great features of OPEN API with Swagger parser dependency.

Objects with API definitions are define in api.yaml file under resource directory of every microservice. Swagger plugin is configured in each project pom.xml in a way that it will deploy objects under target folder which will be available during runtime.

**Testing**

Application is build using TDD approach so there will be a lot of unit test. Two types of test are added, as define unit test and other are integrations tests. Both can be fine for each application in separate folders. Integration test classes are ended with postfix ‘IT’ as a convention of Junit 5.

**Note**

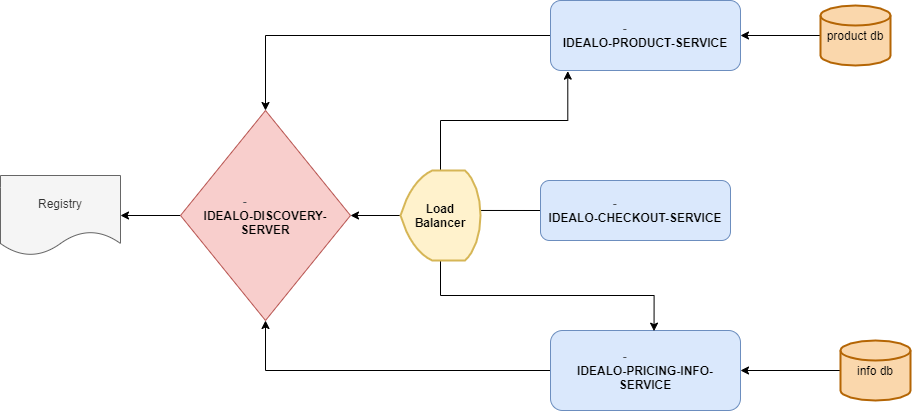
Unit test are completely developed using Junit 5, while integrations tests are developed using Junit 4.

**Microservices**Application consists of four microservices, including one registry service.

* **IDEALO-DISCOVERY-SERVER**
* **IDEALO-CHECKOUT-SERVICE**
* **IDEALO-PRODUCT-SERVICE**
* **IDEALO-PRICING-INFO-SERVICE**

**Design**

Now let’s have a look on system design of checkout challenge in a nutshell. Discover server is responsible for discovering all microservices and registered them in Registry. In the case registered services will not send heartbeats to server, they will be removed from Registry. I am using Netflix Eureka server for discovery server, which is allow to do **client-side discovery**. All the services first registered to eureka server and the ask for other services if they need to communicate, server provide them route, which is also called **Fetch Registry**. For in depth knowledge of how microservices work and how they communicate. What happened if discovery server goes down, how to make it Highly available by initiating it as server as cluster infrastructure you can refer to my article below  
  
**https://www.linkedin.com/pulse/microservices-using-spring-boot-cloud-aqib-javed-/**  
Lets see our checkout design below,



**IDEALO-DISCOVERY-SERVER**

It is a simple discovery server, maintaining the minimal configuration for discovery server Eureka by Netflix. Remember that this server should run before starting other services. Other services will not come up if we did not start IDEALO-DISCOVERY-SERVER.

**Note:**This server is single instance (non-cluster) for assignment purpose, but we can scale that to cluster for high availability with some very simple YAML configuration.  
  
   
**IDEALO-CHECKOUT-SERVICE (orchestrator)**

This is check out service which act as orchestrator, which is responsible for receiving request for checkouts, where we can send request with SKU and check out quantity. This service will verify if we have a valid product, and then apply the pricing rule for this product if any active against SKU and send us total checkout price result.  
  
This service is an orchestrator service which communicate two more microservices to validate product and get its active pricing rules and the prepare the final response. This service does not have any database, in fact it’s just receiving request, communicate with concern microservices and send back us response. This service using Netflix open Feign which is a declarative Http client, and perform load balancing as well by default round robin algorithm. It also supports fault tolerance by providing support of circuit breaker (Hystrix).  
  
**API Usage**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **PATH** | **USAGE** |
| POST | IDEALO-CHECKOUT-SERVICE/checkout/check | Checkout request for products |

**Https Status**

* 200 OK: The request has succeeded
* 400 Bad Request: The request could not be understood by the server
* 404 Not Found: The requested resource cannot be found
* 500 Internal Server Error: The server encountered an unexpected condition

**IDEALO-PRODUCT-SERVICE**

Idealo product service is a completely autonomous, standalone, independent microservice.  
This service is responsible for handling product information in its own database. We can add, remove, update, read products by using IDEALO-PRODUCT-SERVICE exposed API’s.

This service not only taking care of our stock, but also maintain all product details like name of product, brand name, country name where product manufactured etc. But most importantly it contains SKU of a particular product.

**API Usage**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **PATH** | **USAGE** |
| POST | IDEALO-PRODUCT-SERVICE/product/add | Add request for a new product |
| GET | IDEALO-PRODUCT-SERVICE/product/all | Getting all product request |
| POST | IDEALO-PRODUCT-SERVICE/product/all | Getting all product by provided SKU list. |
| POST | IDEALO-PRODUCT-SERVICE/product/drop | Drop products by providing SKU list |

**Https Status**

* 200 OK: The request has succeeded
* 400 Bad Request: The request could not be understood by the server
* 404 Not Found: The requested resource cannot be found
* 500 Internal Server Error: The server encountered an unexpected condition

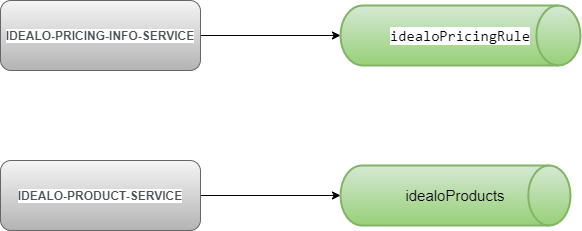
**IDEALO-PRICING-INFO-SERVICE**

This is the service which is responsible to maintain pricing rules against a SKU, and also maintain the logic of applying pricing rules which we see later. This service is flexible to maintain and create pricing rules as much as you want. It will not only help us to create pricing rules but also, we can update, and modify those rules on a very abstract level and finally we can also deactivate them.  
  
This service will load the corresponding pricing rule against SKU if it has any or if not, then a special pricing rule NO will apply. After fetching pricing rule, service will apply those rules in a very abstraction form. I use one of a very famous technique from SOLID pattern, **Open for extension and close for modification.** Product it self do not know how to apply pricing rule, in fact check in request also do not know that, but a pricing rule category define its logic and execute that on run time.

**API Usage**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **PATH** | **USAGE** |
| POST | IDEALO-PRICING-INFO-SERVICE /pricing/rule | Applying pricing rule for given request |
| POST | IDEALO-PRICING-INFO-SERVICE /pricing/rule | Adding new pricing rule |
| GET | IDEALO-PRICING-INFO-SERVICE /pricing/rule | Getting all pricing rule |
| POST | IDEALO-PRICING-INFO-SERVICE /pricing/rule | Getting all pricing rule by given SKU |
| POST | IDEALO-PRICING-INFO-SERVICE /pricing/drop | Dropping pricing rule by given SKU |

**Databases** I am using apache derby embedded database for application and H2 mem database for testing. Each microservice have its own database with is complete isolated from other microservices.  
So both services storing details on the base of SKU, like product details will be saved with SKU, and meanwhile we need to define and store pricing info in info service, associated with a valid SKU for product.  
  
This can be a possibility for product API, that to define product which is not associated with any pricing rule, so in that case when pricing rule API try to fetch pricing rule for product which is not registered in pricing rule API it will consider as rule type **N.**



**Request / Responses**

Let’s have a look for request against these services. These are sample request and response json for each API for the developed application for your ease, in the case you want to run and test yourself.

**Note:** Please run - IDEALO-DISCOVERY-SERVER first if you want to run and test application in real time.

**IDEALO-CHECKOUT-SERVICE/checkout/check**

**Request**

***{***

***"checkoutInfo":[***

#### **{"sku":"PKCAQ","checkoutQuantity":"1"}**

***]***

***}***

**Response**

***{***

***"total": 550,***

***"ruleInfoResponse": [***

***{***

***"sku": "PKCAQ",***

***"checkoutQuantity": 1,***

***"unitPrice": 500,***

***"shipping": 50,***

***"specialPrice": 550,***

***"detailedMessage": "Product PKCAQ have offer to buy 3 products get 1***

***products free"***

***}***

***]***

***}***

**IDEALO-PRODUCT-SERVICE/product/add**

**Request**

***{***

***"sku":"AXMLA",***

***"productId":"PROD\_1",***

***"productName":"TEST",***

***"madeIn":"china",***

***"brand": "ABC",***

***"unitPrice": 500,***

***"quantity": 20,***

***"supplierName":"acc",***

***"isActive":"true"***

***}***

**Response**

***{***

***"sku": "AXMLAX",***

***"productId": "PROD\_X",***

***"productName": "TEST",***

***"detailMessage": "Product save successfully."***

***}***

**IDEALO-PRODUCT-SERVICE/product/all [Get API]**

**Request N/A**

**Response**

***{***

***"productDetailResponseList": [***

***{***

***"sku": "AXMLAX",***

***"productId": "PROD\_X",***

***"productName": "TEST",***

***"madeIn": "china",***

***"brand": "ABC",***

***"unitPrice": 500,***

***"shipping": null,***

***"quantity": 20,***

***"supplierName": "acc",***

***"isActive": true***

***}***

***]***

***}***

**IDEALO-PRODUCT-SERVICE/product/all [Post API]**

**Request**

***{***

***"checkoutInfo":[***

***{***

***"sku": "PKCAQ",***

***"checkoutQuantity": "1"***

***}***

***]***

***}***

**Response**

***{***

***"productShortResponseList": [***

***{***

***"sku": "PKCAQ",***

***"productId": "COBB-S1",***

***"quantity": 100,***

***"unitPrice": 500,***

***"shipping": 50,***

***"detailedMessage": null***

***}***

***]***

***}***

**IDEALO-PRODUCT-SERVICE/product/drop**

**Request**

***{***

***"skuList":[***

***"PKCAQ"***

***]***

***}***

**Response**

***{***

***"detailMessage": "Products dropped successfully"***

***}***

**IDEALO-CHECKOUT-SERVICE/pricing/rule**

**Request**

***{***

***"ruleInfoRequest": [***

***{***

***"sku": "PKCAQ",***

***"unitPrice":"30",***

***"shipping":"5",***

***"checkoutQuantity":4***

***}***

***]***

***}***

**Response**

***{***

***"total": 95,***

***"ruleInfoResponse": [***

***{***

***"sku": "PKCAQ",***

***"checkoutQuantity": 4,***

***"unitPrice": 30,***

***"shipping": 5,***

***"specialPrice": 95,***

***"detailedMessage": "Product PKCAQ have offer to buy 3 products get 1***

***products free"***

***}***

***]***

***}***

**IDEALO-CHECKOUT-SERVICE/pricing/add**

**Request**

***{***

***"ruleInfos" : [***

***{***

***"sku":"RKCAQ",***

***"itemsFromApply":"3",***

***"discount":"1",***

***"ruleType":"BNGN",***

***"isActive":"true"***

***}***

***]***

***}***

**Response**

***{***

***"message": "1 number of rules are successfully inserted"***

***}***

**IDEALO-CHECKOUT-SERVICE/pricing/all [Get API]**

**Request N/A**

**Response**

***{ "ruleInfos": [***

***{***

***"sku": "PKCAQ",***

***"itemsFromApply": 3,***

***"discount": 1,***

***"ruleType": "BNGN",***

***"isActive": true***

***},***

***{***

***"sku": "PKCAB",***

***"itemsFromApply": 10,***

***"discount": 2,***

***"ruleType": "BNGN",***

***"isActive": true***

***}***

***]***

***}***

**IDEALO-CHECKOUT-SERVICE/pricing/all [Post API]**

**Request**

***{***

***"skuList": [***

***"PKCAQ"***

***]***

***}***

**Response**

***{"ruleInfos": [***

***{***

***"sku": "PKCAQ",***

***"itemsFromApply": 3,***

***"discount": 1,***

***"ruleType": "BNGN",***

***"isActive": true***

***}***

***]***

***}***

**IDEALO-CHECKOUT-SERVICE/pricing/drop**

**Request**

***{***

***"skuList" : [***

***"PKCAQ"***

***]***

***}***

**Response**

***{***

***"detailMessage": "Rules drop successfully"***

***}***