Rakuten Deutschland

Coding Test

Table of Contents

Introduction

Congratulations! You are one step closer of joining the Rakuten Team!

In this step you are invited to show up your coding skills. Please read cautiously the instructions to make sure that you understood the requirements before start coding.

During the test you are going to develop a fictional scenario, Rakuten.de does not follow the specifications provided in this test.

Please do not share this test. If you are going to host your code in Github, Bitbucket or a similar code hosting solution, please use a private repository.

Customer Engagement Program

At Rakuten.de we have special programs to keep our customers happy and buying from us.

The Customer Engagement Program, a.k.a. CEP is one of them. CEP is a program to give customers points that they can redeem on their next orders.

Program details

Points Balance

The points are accumulated in the Customer's Points Balance, which works like a "wallet" to keep the customer's Points;

Calculation Rules

The rules for the program are the following:

- After submitting an order, it is added to the customer's Points Balance 1 Point for each Euro payed in products. Considering that, it is right to assume that the default rate for calculating Points x Euros is 1 x 1;
- The Marketing team can create special Campaigns, in which there are custom Points x Euros rates for specific products. The Campaigns are valid in between a range of dates defined by the Marketing team;
- If the customer returns an item or the full order, the points received for the cancelled items are subtracted from the customer's Points Balance;

Functional Requirements

Campaigns

The Marketing team must be able to create special Campaigns providing the following Campaign data:

1. name
2. startDate
3. endDate
4. points (per Euro)
5. products (list of products selected to the campaign)

Important:

- All of those data are required.
- The minimum start date for creating a new campaign is Current Date + 1, to avoid on-the-fly calculation changes and allow for caching;
- The endDate obviously must be after the startDate;
- Points (Points x Euros rate) must be bigger than the default rate;
- It is required at least one product in the products list;

Points calculation

The following data will be provided to calculate and accumulate the points related to an order:

- order
  - customerEmail
  - createdAt
  - items
    - product 1
      - id
      - name
      - price
    - product 2
      - id
      - name
      - price
    - product n…
      - id
      - name
      - price

For each order, must be calculated the total quantity of points to receive and the result must be added to the Customer's Points Balance.

The points calculation must be done for each order item, looking for active campaigns including the product in the order creation date and, if found, must be used the custom Points x Euros rate defined in the campaign for the specified items.

Order cancelation

In case of order cancelation, the points calculation transaction must be deleted, and the related points must be subtracted from the Customer's Points Balance.

Get Customer's Points Balance

The customer must be able to get the current Points Balance to check the quantity of points available for using in the next orders.

Technical requirements

The Development Unit uses the Microservices architectural model for developing the ecosystem.

For this feature, two new Microservices should be created, the Campaign API and the Points API.

Both Microservices will run in a Containers Orchestration Environment, Kubernetes, so that they need to be able to run as Containers. Please provide the Dockerfile prepared for running your new APIs as containers.

The responsible Solution Architect modeled the APIs in Swagger Hub. The attached Open API 2.0 Json (Swagger) files can be used to understand the expected features and to generate the model classes in your preferred language.

The Quality Assurance team prepared some tests to check the behavior of the new APIs. The attached Postman collections can be used to test if the Microservices are working properly.

Expected Artifacts

- **Campaign API** and **Points API** ready to run in Containers;
    - The Applications must respond in a Web Context:
        - http://localhost:8080/campaignapi
        - http://localhost:8080/pointsapi
    - The Microservice **Points API** must reference the Microservice **Campaign API** using the address provided in the Environment Variable CAMPAIGN_API_ADDRESS(export CAMPAIGN_API_ADDRESS= http://localhost:8080/campaignapi)
    - The Microservices must load the Database Connection configuration from an Environment Variable named DB_CONNECTION;
    - The applications must have a working **Dockerfile**
- The database must run in a Container. Please provide the Docker Image used to setup the database in your local environment and the required configuration;
- Minimum project documentation (e.g. README.md) with
    - Build instructions
    - Configuration instructions
    - Relevant information
- All features described in the Swagger specification and Postman collections working;
- Proper Unit tests coverage;

Expected Mindset

- Clean Code;
- Well Structured Architecture;
- Code secure by design;
- Smart technology selection in terms of
    - Cloud Native
    - Building scalable and resource efficient artifacts
    - Dependencies chosen
    - Productivity

Conclusion

Take your time to do the test and reach us out if you have any question.
After finishing please zip your project code and documentation and send it through e-mail.

Good luck!