Exercise 01 - Order Process Changes

Required Services

The following services are involved and have to be started before the final exercise validation:

- CustomerSrv (`http://localhost:8000`)
- NotificationSrv (`http://localhost:8010`)
- OrderProcessSrv (`http://localhost:8020`)
- OrderSrv (`http://localhost:8030`)
- ProductSrvFacade (`http://localhost:8040`)
- ProductSrv (`http://localhost:8050`)
- Apache Zookeeper (`localhost:2181`, starts automatically in the provided Ubuntu VM)
- Apache Kafka (`localhost:9092`, starts automatically in the provided Ubuntu VM)

Description

The `OrderProcessSrv` is responsible for orchestrating the process of creating a new order, which can be invoked via `POST http://localhost:8020/order-process (experiment.webshop.orderprocess.resources.OrderProcessResource)`. The following JSON payload is an example for a new order request:

```
{
   "customerId": 1,
   "items": [
      {
         "productId": 1,
         "amount": 2
      },
      {
         "productId": 2,
         "amount": 1
      }
   ]
}
```

The current process looks as follows (see the `createOrderProcess()` method in the `experiment.webshop.orderprocess.resources.OrderProcessResource` class):

1. Use the provided `customerId` to refresh and check the customer's credit rating (1-6) by invoking the `CustomerSrv` via `GET http://localhost:8000/customers/{id}/credit-rating-check (experiment.webshop.customers.resources.CustomerResource)`. Ratings of 4 or worse are rejected, ratings of 1-3 are accepted. An example response may look as follows:

```
{
    "customerId": 1,
```

```
    "rating": 6
}
```

2. If the credit rating check was successful, the availability of the requested items is checked via the `ProductSrv`. For each item, `GET http://localhost:8050/products/{id}/availability?amount={amount} (experiment.webshop.products.resources.ProductResource)` is invoked. A product counts as `available`, if at least 3 copies would remain in stock after fullfilling the new order. An example response may look as follows:

```
{
    "productId": 1,
    "availableAmount": 6,
    "requestedAmount": 3
}
```

3. If all requested items are available in the necessary capacity, the order is created and stored by invoking the `OrderSrv` via `POST /orders` with the original order payload.

After some research, the sales team has decided that this process should now be adjusted and extended.

Tasks

All changes have to be performed in the `OrderProcessSrv`, more precisely within the `createOrderProcess()` method of the `experiment.webshop.orderprocess.resources.OrderProcessResource` class.

1. **Change the credit rating validation logic.** From now on, ratings of 1-4 should be accepted and ratings from 5-6 should be rejected. In short, the worst allowed rating should be increased from 3 to 4.
2. **Change the product availability validation logic.** From now on, at least 2 copies of the ordered product have to remain in stock after fulfilling the new order for the product to count as `available`. In short, the minimal remaining amount should be decreased from 3 to 2.
3. **Add a new final process step.** After successful ordering, the `NotificationSrv` should be invoked to send a marketing mail with similar products to the customer via `POST http://localhost:8010/marketing-mails`. Use the provided Jersey `restClient` instance for this. You can copy and adapt one of the existing invocations from the same method (e.g. the credit rating check). Instead of `get()`, invoke the `post()` method of a created `request` (see below). An example payload (`experiment.webshop.orders.api.MarketingMailRequest`) is also provided below (`order` will of course be the newly created order).

```
POST request example:

Invocation.Builder request = restClient.target(notificationSrvUrl).request();
request.post(Entity.json(marketingMailRequest), BaseResponse.class);
```

```
MarketingMailRequest example:

{
    "type": "SIMILAR_PRODUCTS_MAIL",
    "order": {
        "id": 6,
        "created": 1525695805999,
        "status": "NEW",
        "customerId": 6,
        "items": [
            {
                "productId": 1,
                "amount": 2
            },
            {
                "productId": 2,
                "amount": 1
            }
        ]
    }
}
```

Validation

When you are finished with all tasks, make sure all required services (see Required Services) and the exercise validation UI is up and running (if not, execute `exercise-validation/build-and-run-validation-ui.sh`) and then navigate to `http://localhost:5001` (**it is important to start from this page, because it will determine which version you are working on**). Click on `Exercise 01` and then on `Start Validation`. If every check is successful (`status: true`), pause your stopwatch and notify an experiment admin to write down your time.