Complaint System

Overview

You are tasked with building a complaint management system for an innovative online market system. The system should be designed to receive a complaint from buyers in the market.
The market handles a very high load of transactions, and we're expecting a relatively high load of complaints as well. As such, your system should be robust, fault tolerant and designed to scale (no autoscaling will be required at this point).

System requirements

You will be building a REST service that will receive user complaints. The complaints will contain the following information:

- User ID (UUID)
- Subject
- Complaint (text describing the user's issue)
- Associated purchase UUID (optional)

You will need to store the complaint information in persistent storage. In addition, every generated complaint should retrieve (and store in the DB) additional information from our systems to capture a snapshot at the time of the complaint for the customer success specialist to review.
Note that these are REST services controlled by a different team in our product and you're expected to make REST calls to fetch data from them. A mock implementation of the following APIs has been provided for you for integration testing (instructions will be provided below). The services have not been completed yet, and it's unknown at this time what the SLA for these calls will be.

1. Full information about the user from our user management service (/users/{id}). Sample response below:

```
{
   "Id":"a93adc57-4d59-4a9d-85c6-b5d48d99101d",
   "fullName":"John Doe",
   "emailAddress":"johndoe@test.com",
   "physicalAddress":"Test Lane 1 Los Angeles"
}
```

2. Information about the associated purchase from our purchases service (/purchases/{id}). Sample response below:

```
{
	"id":"f256c996-6dcb-40cf-8dce-a11fa9bcab6b",
	"userId":"a93adc57-4d59-4a9d-85c6-b5d48d99101d",
	"productId":"4ac9da0b-66eb-415c-9153-a59ec0b3c3fe",
	"productName":"Book",
	"pricePaidAmount":13.2,
	"priceCurrency":"USD",
	"discountPercent":0.1,
	"merchantId":"71e234d2-dc65-41ff-bada-9d08d82fc6d1",
	"purchaseDate":"2020-11-20T22:00:00.000+00:00"
}
```

In the future, more information from additional sources may be required, please plan the system accordingly.

Assignment requirements

1. Create a REST endpoint for complaint creation as described above. Since this endpoint interacts with the customer directly, it should be kept as error-free as possible (e.g. unavailability of the additional data shouldn't keep us from registering the complaint). This API call should not exceed 200 ms at most.
   Example request:

   ```
   {
      “userId” : “14b28cf0-7a0d-11ec-90d6-0242ac120003”,
      “subject” : “The seller never sent my item!”,
      “complaint” : “I made a purchase and the item hasn’t shipped. It’s been over a week.
   Please help!”,
      “purchaseId” : “14b28cf0-7a0d-11ec-90d6-0242ac120003”
   }
   ```

   You should make provisions for retrieving data from the external services above, while keeping in mind that you have to keep to a strict SLA (as mentioned above), and in the future there might be requirements for retrieval of data from additional sources, while the SLA for your API will remain unchanged (since it’s an end-customer interaction).
   There’s no need to implement the external services yourself.

2. Create a REST endpoint for complaint case retrieval. This will be used to provide the information to our customer success specialists. Make sure that the additional information (as described above) is present and retrieved in this API.
   This API call should not exceed 2 seconds.

3. The system should be built in Java, using Maven and SpringBoot. Consider best practices in REST API design, clean code, naming conventions, testability, failure handling etc.
4. Feel free to add any additional components that may be required to support the requirements (such as additional microservices, queues etc.).
5. Be prepared to present the design and explain your design decisions.
6. The submission should include the final version of your code and any instructions that may be required to run it.

Instructions for running the mock service

You can simply run the mock service jar (e.g. via running "java -jar craft-mock.jar" in your terminal). The mock service exposes HTTP port 8081 and will respond to the following APIs:

1. /users/{id}
   IDs that will return a value from this API:
   a93adc57-4d59-4a9d-85c6-b5d48d99101d
   a872d86a-c7cb-48b7-b5d9-f218d6845405
   dcb6a039-b0fc-49dd-b5de-58856f66727d
   bbbb080d-cffa-46d0-aa22-786c35d1a35b
   f22dff3f-06cf-49fe-97ec-bf7afe9a7fdb
   72dddc34-f058-4d31-b370-e88f772ea8e8
   a90f6dd7-b74b-4be6-9065-daa1a92ba7ab
   a227c2bd-358a-4587-95f0-61fb63678952
   8145b0d6-feb2-4ff6-8546-c0a5eece6f82
2. /purchases/{id}
   IDs that will return a value from this API:
   f256c996-6dcb-40cf-8dce-a11fa9bcab6b
   21d5dbe2-8369-459d-a955-a6b4f19b4d53
   340d04f5-4241-4cc3-bfb4-861c5c552891
   30de333b-d7da-4906-b382-1a8ff59556ff
   4933d1ce-9ca7-4640-ba17-e442057c44f1
   e11cc042-80f4-4a0b-b580-d1eedf6b153c
   57cd80e4-ed5b-4d7e-a39d-15d039cb2069
   0797d85e-ae2e-482a-ae38-26ab83a8947e
   c2437271-ec14-40fe-92cc-22284ab3a25f
   4b8d0786-a2e8-45cf-8d89-faa390c098df
   c7f07884-2faf-4c6e-8a51-c6256812d73b
   af0676a6-542f-44c6-ae9d-1c0da4c4f1b3