Cypress Take Home Exercise

Mastercard Decision Management Program Take Home Exercise

Thanks for your interest and application for this role at Mastercard, we would like to move forward with the process by having you complete a take home exercise that we can use for collaborative discussion in the next technical interview round. Please read through the instructions and submit your exercise back to the recruiting team as a zip file or hosted in a location of your preference. We look forward to what you're able to come up with!

Goal:
Build a new proof of concept API in Java that functions as a simple credit card fraud detection service.

Description:
We will build a service that has 1 endpoint takes in several parameters as inputs, calls another service to get information regarding the transaction, and then applies some business logic before returning a decision to approve or decline the transaction.

Endpoints:
Create an endpoint named "/analyzeTransaction" that takes in a json object of 1 transaction with properties for card number and transaction amount.

JSON Schema for the request body:


```
{
   "title":"root",
   "type":"object",
   "required": [
   "transaction"
   ],
   "properties": {
      "transaction": {
         "title": "transaction",
         "type": "object",
         "required": [
            "cardNum",
            "amount"
         ],
         "properties": {
            "cardNum": {
               "title": "cardnum",
               "type": "integer",
               "minimum": 1000000000000000,
               "maximum": 9999999999999999,
               "examples": [
                  5026840000000001
               ],
               "default": 9999999999999999
            },
            "amount": {
               "title": "amount",
               "type": "number",
               "minimum": 0.00,
               "examples": [
                  3.99
               ],
               "default": 0.00
            }
         }
      }
   }
}
```


Example transaction object where the card number is 5206840000000001 and amount is $3.99:


```
{

   "transaction": {
      "cardNum":5206840000000001,
      "amount":3.99
   }
}
```

External service call:
For this exercise, we have an imaginary external microservice built by another team in our organization that returns how many times a card has been used in each of the past 7 days. This service can be called using the following endpoint:

http://www.randomnumberapi.com/api/v1.0/random?min=0&max=12&count=7

We will use the information provided by this service in the logic stated in the following section to make informed decisions on fraudulent activity in transactions.

Example response where the card was used 4,6,10,12,1,12,and 2 times for a total of 47 times in the past 7 days: [4, 6, 10, 12, 1, 12, 2]

Business logic:
Data scientists in our organization have come up with the following fraud scenarios that need to be implemented in this proof of concept service that we're building.

- If the amount of a transaction is over $50,000.00 decline the transaction.
- If the card has been used over 60 times in the last 7 days, decline the transaction.
- If the card has been used under 35 times in the last 7 days, decline the transaction if the (transaction amount/times used in last 7 days) > 500. (E. g. Decline if transaction amount is $9000 and the card has been used 2 times in the last 7 days. 9000/2 = 4500)
- Approve all other transactions.

Response JSON object:
Return a transaction analysis JSON object in the response with the following information:

- *Card Number*
- *Transaction Amount*
- *Information on if the transaction was approved or declined*
- *Number of times the card has been used in the last 7 days*

Other requirements:

- Our service should be able to handle and respond to input errors.
- The service should log the following underlined transaction information for debugging - card number (for privacy of users for this service, obfuscate the middle 8 digits of card numbers, e.g. log 52068400000000000001 as 5206********0001), amount, and number of times card was used in last 7 days.

If anything isn't clear, use a comment in the code or in the README to state any assumptions.

Build and packaging:
Candidate should bundle this service akin to a production ready application:

Build – pom.xml if Maven project

Tests – unit tests (integration would be nice since it calls external API) with coverage report, load-test even better

Documentation – README/Javadoc/swagger

Monitoring – endpoint to show basic stats (if using Spring, can easily include lots of info)

Packaging – if wrapped around in a container, would be nice, if deployable to cluster like k8s, even better!

Send the packaged code back to the recruiting team (we recommend a github link) and be ready to discuss the implementation details during the next interview round.

PDF version

Confirm Selections in List

Retention Labels

- ☐ Corporate Security Incidents and Monitoring
- ☐ Corporate Security Permissions Access and User Requests
- ☐ Corporate Security Policy and Procedures
- ☐ Customer and External Interactions
- ☐ Information Technology MC Software Design and Business Continuity
- ☐ Information Technology Ops and Technical Standards

- ☐ Intellectual Property Patents and Filings
- ☐ Legal Regulatory and Compliance
- ☐ Marketing and Sales Campaign Management
- ☐ Marketing and Sales Strategy and Intelligence
- ☐ Money Service Business and Cross Border Services
- ☐ Not an Official Record
- ☐ Privacy Incident Management
- ☐ Privacy Standards and Compliance
- ☐ Project Management
- ☐ Real Estate Business Management
- ☐ Real Estate Facility Operations
- ☐ Real Estate Property and Land Management