TNT Digital – Back-End Software Developer Assessment

This is the TNT back-end software developer assessment, the purpose of which is to give us insight in your technical abilities, development approach and general technical working habits. We view your performance on this assessment as indicative of the work you will deliver as a back-end developer at TNT.

The assessment consists of an assignment to prepare beforehand. The assessment will be concluded by an in-person discussion of your solution. We advise to develop the stories in-order, since they built on each other, but feel free to do them in any order you prefer. Also feel free to time-box yourself as long as the stories you finish are delivered in a complete and end-to-end fashion. We expect one working solution that covers both stories.

The assignment is implementing the 'API queuing service', described below. Read the case carefully, and approach it as you would a regular project. Consider aspects such as robustness, maintainability, testing and user experience.

The only requirement is that the assignment should be implemented in Java (so not Scala, Go, etc.). You're free to choose any framework you see fit. Any API dependencies can be mocked away by running a stub HTTP server, or any other method of choice.

We ask you to treat this assessment as confidential so we can apply the scenarios to future candidates. Your solution will not be kept after the assessment and will not be used by TNT, but we would appreciate it if you would not place this into the public domain e.g. upload it to GitHub.

Good luck with the assignment!

The Context: API Queuing Service

TNT is building a brand new application that has to interface with several APIs, and you are tasked with building the queue for communicating with these external APIs. The product owner has split this into two user stories, each with their own requirements.

There are 3 different external APIs that our service has to interface with. Each of the APIs accepts a query parameter `?q=` that can accept multiple queries, split using a comma delimiter. If the same value is present multiple times in the query, the response will only contain it once. Each of the APIs provides requests and responses in their own unique manner as shown below. Expect all APIs to always return `200 OK` responses with well-formed input, or `503 UNAVAILABLE` when they are unavailable.

The Shipments API

Accepts one or more 9-digit order numbers (comma separated) and returns a set of products (envelope, box or pallet) equivalent to the last digit. For example: order number `109347263` will return a set of 3 different products. E.g.:

```
GET http://<domain>/shipments?q=109347263,123456891
200 OK
content-type: application/json
{
  "109347263": ["box", "box", "pallet"],
  "123456891": ["envelope"]
}
```

The Track API

Accepts one or more 9-digit order numbers (comma separated) and returns one of the following tracking statuses: `NEW, IN TRANSIT, COLLECTING, COLLECTED, DELIVERING, DELIVERED`.

```
GET http://<domain>/track?q=109347263,123456891
200 OK
content-type: application/json
{
  "109347263": "NEW",
  "123456891": "COLLECTING"
}
```

The Pricing API

Accepts one or more ISO-2 country codes (comma separated) and returns a randomized floating number between 1 and 100.

```
GET http://<domain>/pricing?q=NL,CN
200 OK
content-type: application/json
{
  "NL": 14.242090605778
  "CN": 20.503467806384
}
```

The Stories

AQS-1: As TNT, I want to be able to query all services in order to provide

information to our users.

Implement this service - no endpoint is required, a simple method is fine - that accepts a collection of API requests to Pricing, Track and/or Shipments. Each API request object consists of an API identifier and the query for that API. Upon calling, the different calls should be forwarded to the individual APIs. Only upon receiving all responses should the complete set of responses be returned to the caller.

AQS-2: as TNT, I want service calls to be throttled and bulked into 1 request per respective API to prevent services from being overloaded.

To prevent overloading the APIs with query calls we would like to queue calls per API endpoint. All incoming requests for each individual API should be kept in a queue and be forwarded to the API as soon as a cap of 5 calls for an individual API is reached.

If the cap for a specific API is reached a single request will be sent using the `q` parameter with 5 comma delimited values.

Example: if there is a caller querying each API and the queue of the Pricing API holds 4 requests, the next request to the Pricing API will trigger the actual bulk request to be made. Each API will have its own queue.

Only upon receiving a response from all API endpoints that were queried should the original service request be responded to.

Our current implementation has one major downside; the caller will not receive a response to its requests if the queue cap for a specific service is not reached. This is acceptable for now.

Well done!

Congratulations on finishing the assessment!