

TNT Digital - Back-End Software Developer Assessment

Intro

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. Consequently, later stories should not break the functionality already implemented in the earlier stories.

Please keep a log of the significant design decisions you make during development. A brief sentence per decision is enough, but use your own judgement on the level of detail. These points can be further discussed during the in-person interview.

We feel that it should be possible to complete the assignment in a normal working day (8 hours) of hands-on work.

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 make it available to the general public e.g. upload it to a public GitHub repository.

Good luck with the assignment!

Assignment

The assignment is implementing the 'API aggregation service', described below. Read the case carefully, and approach it as you would a regular project. Consider aspects such as robustness, maintainability, and automated testing. Deliver a code quality that you consider acceptable for submitting a pull request for your team members to review for inclusion in a production service.

The only technical requirement is that the assignment should be implemented in Java (so not Scala, Kotlin, Go, etc.). You're free to choose any framework you see fit.

The assignment includes the use of existing back-end services. We provide implementation of those services as a Docker image which is available at Docker Hub <https://hub.docker.com/r/xyzassessment/backend-services>.

Please provide instructions on running your completed assignment in the Readme. Part of the evaluation will be done by running a pre-built test suite against your implementation.

The API Aggregation Service Context

TNT is building a brand new application that has to interface with several APIs, and you are tasked with building an aggregation service for consolidating the interface with these external APIs into a single endpoint that can handle multiple logical requests in one network request. The product owner has split this into three 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.

These back-end services are delivered with an SLA guaranteeing a response time of at most 5 seconds for at least 99 percent of the requests.

An implementation of all 3 of these APIs is available at Docker Hub
<https://hub.docker.com/r/xyzassessment/backend-services>.

The Shipments API

Accepts one or more 9-digit order numbers (comma separated) and returns a list 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://<host>:8080/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://<host>:8080/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 randomised floating number between 1 and 100.

```
GET http://<host>:8080/pricing?q=NL,CN
200 OK
content-type: application/json
{
  "NL": 14.242090605778
  "CN": 20.503467806384
}
```

The API Aggregation Service Contract

The contract that the new application will use to access the aggregation API has already been defined.

The API accepts three different parameters to specify the values to be passed to the individual backing APIs. It returns the consolidated results in a Json object. For cases where the backing api fails to return a good result, due to either error or timeout, the field will still be included in the returned object, but the value will be 'null'.

```
GET http://<host>:8080/aggregation?
pricing=NL,CN&track=109347263,123456891&shipments=109347263,123456891
200 OK
content-type: application/json
{
  "pricing": {
    "NL": 14.242090605778
    "CN": 20.503467806384
  }
  "track": {
    "109347263": null,
    "123456891": "COLLECTING"
  }
}
```

```

    "shipments": {
      "109347263": ["box", "box", "pallet"],
      "123456891": null
    }
  }
}

```

The SLA for the aggregation service is to respond within 10 seconds for the 99th percentile, although that becomes mostly relevant in the final story, and can be ignored in the first 2 stories.

The Stories

AS-1: As TNT, I want to be able to query all services in a single network call to optimise network traffic.

Implement the interface described above that accepts a collection of API requests to Pricing, Track and/or Shipments. For each request to the aggregation API, 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.

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

To prevent overloading the APIs with query calls we would like to consolidate 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.

Out of scope for this story is dealing with calls that remain in the queue due to not submitting exact multiples of 5 for each API.

AS-3: as TNT, I want service calls to be scheduled periodically even if the queue is not full to prevent overly-long response times.

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. To solve this, we want the service queues to also be sent out within 5 seconds of the oldest item being inserted into the queue. In case of the cap being reached within these 5 seconds, the timer should be reset to zero.

This will allow us to meet the 10 second SLA for requests to the aggregation service.

Checklist

- ☐ AS-1 implemented
- ☐ AS-2 implemented
- ☐ AS-3 implemented
- ☐ Readme with run instructions
- ☐ Document with design decisions

Well done! Congratulations on finishing the assignment!