FedEx backend software developer assignment

Intro

This is the FedEx backend assessment, the purpose of which is to give us insight into 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 backend developer at FedEx.

The assessment consists of an assignment to prepare beforehand. The assessment will be concluded by an in-person discussion of your solution.

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 FedEx, but we would appreciate it if you would not make it available to the general public, e.g. you should not upload it to a public GitHub repository.

Good luck with the assignment!

Assignment

Your assignment is to implement the `API Aggregation Service`, described below. Read the requirements 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 backend services. We provide implementation of those services as a Docker image which is available at Docker Hub https://hub.docker.com/r/qwkz/backend-services/tags

To run the backend services locally (to use them to implement the solution), use the following command: `docker container run --publish 4000:4000 qwkz/backend-services:latest`.

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

The API Aggregation Service

FedEx 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.

There are 3 different external APIs that our service has to interface with. 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 a well-formed output, or `503 Unavailable` when they are unavailable.

These backend services will respond in 5 seconds most of the time, but sometimes they can take a lot of time to respond.

The Shipments API

Accepts a 9-digit order number and returns a list of products (ENVELOPE, BOX or PALLET) for that order.

```
GET http://127.0.0.1:4000/shipment-products?orderNumber=109347263

200 OK
Content-Type: application/json

["BOX", "BOX", "PALLET"]
```

The Track API

Accepts a 9-digit order number and returns one of the following tracking statuses: `NEW`, `IN_TRANSIT`, `COLLECTING`, `COLLECTED`, `DELIVERING`, `DELIVERED`.

```
GET http://127.0.0.1:4000/track-status?orderNumber=109347263

200 OK
Content-Type: application/json

"IN_TRANSIT"
```

The Pricing API

Accepts an ISO 3166-1 alpha-2 country code and returns the base pricing for that country:

```
GET http://127.0.0.1:4000/pricing?countryCode=NL

200 OK
Content-Type: application/json

14.242090605778
```

The API Aggregation Service Contract

This is the contract of the API that you need to implement.

The aggregation service must respond within 5 seconds for the 99th percentile.

The API accepts three different parameters to specify the values to be passed to the individual backend 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 should not be included in the returned object.

```
GET
http://127.0.0.1:8080/aggregation?shipmentsOrderNumbers=987654321,123456789&trackOrder
Numbers=987654321,123456789&pricingCountryCodes=NL,CN

200 OK
Content-Type: application/json

{
  "shipments": {
    "987654321": ["BOX", "BOX", "PALLET"]
  },
  "track": {
    "123456789": "COLLECTING"
  },
  "pricing": {
    "NL": 14.242090605778
    "CN": 20.503467806384
  }
}
```

Note how `123456789` is missing from the `shipments` field, and also how `987654321` is missing from the `track` field. This is because either the backend responded with an error, or didn't respond in time.

All 3 query parameters (`shipmentsOrderNumbers`, `trackOrderNumbers`, and `pricingCountryCodes`) are optional. If any of them is missing, the aggregation API should not do any request to the corresponding backend service, and the response should have an empty map. For example, in the extreme case where all 3 query parameters are missing, the aggregation API should return this:

```
GET http://127.0.0.1:8080/aggregation

200 OK
Content-Type: application/json

{
  "shipments": { },
  "track": { },
  "pricing": { }
}
```

To do

Implement the API described above that accepts a collection of API requests to Pricing, Track and/or Shipments backend APIs. The aggregation API should call the backend APIs as needed.

If any of the backend API calls doesn't complete in time for us to satisfy the 5 seconds SLA of the aggregation API, they need to be aborted, and the corresponding key should not be added to the aggregation response.

Checklist

- [ ] aggregation API implemented
- [ ] aggregation API automated tests implemented
- [ ] readme with instructions on how to run your solution provided
- [ ] document with design decisions provided

Well done! Congratulations on finishing the assignment!