Please make sure the backend service is configurable.

For example, to run your implementation with the gradle wrapper:

```
> SERVER_PORT=8080 BACKEND_SERVICE=http://localhost:4000 ./gradlew bootRun
```

Or docker:

```
> docker run -p 8080:8080 -e BACKEND_SERVICE=http://localhost:4000 my-image
```

You don't have to use these names, please just make sure the readme shows how both aspects can be set from the command line.

Evaluation

We look both at your code by manually reviewing it, and running a pre-built test suite against your implementation.

The API Aggregation Service Context

[redacted] 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 backend 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: pricing, track and shipments. These parameters are all optional and could be missing. 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 be mostly becomes relevant in de final story, and can be ignored in the first 2 stories.

The Stories

AS-1: As [redacted] 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 [redacted] 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 per API and be forwarded as soon as a cap of 5 values 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:

- This request only triggers a call to the Pricing API `/aggregation?pricing=NL,CN,CH,GB,DE&track=117347282,109347263,123456891`
- As soon as this request arrives, a call to the Track API is triggered as well: `/aggregation?track=219389201,813434811`

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 exactly multiples of 5 for each API.

AS-3: [redacted], 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 value 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!

Generated Tue Nov 21 12:57:54 UTC 2023