1 Java Technical Test

This is the initial test for a Java developer role.

For this exam, we have defined the REST APIs for an application you will need to build.

Data can be stored in-memory so do not worry about connecting to a database or a fancy caching solution, you are encouraged to use a data structure that is built into the Java Collections library.

You are also welcome to include other 3rd party libraries as you seem fit.

We would ideally expect 80% unit test coverage and it is optional whether you want to write any additional automated tests such as integration tests you are welcome to however it is not a requirement to do so.

Please write “production quality” code and add comments when you feel it is not clear on its own.

Every solution must:

- Be written in JAVA 8.
- Be compiled and packaged using maven.
- Must contain Spring Boot opensource libraries.
- Have a “README.md” file that explains how to compile, run, and test it.

1.1 Java Exam - Consumer IoT

Write a Java Spring Boot microservice to expose a simple REST API to report on the location of IoT tracking devices batch loaded from a csv file.

data.csv file provided

There are three parts to this assignment. Each of the three parts should be thought about as a minimally viable product (MVP) providing a fully working solution that is well tested, clearly documented and comprised of production quality code that is also performant.

Please use Java 8+ with lambda expressions/streams where appropriate.

Feel free to modify the data file to include more scenarios in it; we will be testing the controller with a larger file to ensure that all the boundary conditions and unhappy paths are handled appropriately.

1.1.1 Part 1 MVP: IoT batch data loading service

Assumptions: both IoT location tracking devices generate data that was sent from the device to the backend platform in a specific location. It is not yet loaded on the memory.

Please write a service and REST controller capable of batch loading the data provided in the attached csv file (see data.csv) and matching the following endpoint:

URL Path	`iot/event/v1/`
HTTP Method	`POST`
Body	`{ "filepath": "C:/path/to/data.csv" }`

SUCCESSFUL RESPONSE:

HTTP Status	`200 OK`
Body	`{ "description": "data refreshed" }`

The data only has to be loaded in memory and use of a database or cache is not required.

Every time this endpoint is called the data should completely replace whatever was previously there with the contents of the csv file.

Please use Swagger to cleanly document and test this endpoint.

Unhappy Paths

1. If the path to the data.csv is incorrect or the file cannot be found please return the following message:
   (No data present)

HTTP Status	404 `Not Found`
Body	`{ "description": "ERROR: no data file found" }`

2. All other exceptions should return 500 with the message "ERROR: A technical exception occurred". You are also free to append more details to a technical exception if you think it would be informative to the user and/or for debugging.
   (Misc. technical exception)

HTTP Status	500 `Internal Server Error`
Body	`{ "description": "ERROR: A technical exception occurred" }`

1.1.2 Part 2 MVP+1: Report device details and location

Extend your microservice form Part 1 to include an additional REST endpoint that will report on a device's location given the product id and an optional timestamp parameter.

The timestamp lookup should return a single set of data (aka a row from the csv file) matching the DateTime and ProductId fields provided in the csv; if the timestamp provided is not a complete match, then it should return the data that is closest to it in the past for that particular ProductId. If no tstmp param is provided, then the value should default to the current UTC time. All datetimes should relate to UTC (although you are free to make this configurable provided the default configuration is set to UTC and milliseconds).

Please build this API to match following endpoint definition:

URL Path	`iot/event/v1?ProductId=WG11155638&tstmp=1582605137000`
HTTP Method	`GET`

SUCCESSFUL RESPONSE:

HTTP Status	`200 OK`
Body	`{ "id":"WG11155638", "name":"CyclePlusTracker", "datetime":"25/02/2020 04:31:17", "long":"51.5185", "lat":"-0.1736", "status":"Active", "battery":"Full", "description":"SUCCESS: Location identified." }`

As this is a customer-facing product, you will need to format the data to be user-friendly.

1. Differentiate between the two products supported: one is a CyclePlusTracker (product id starts with 'WG') which is used on bicycles the other is a GeneralTracker (product id starts with '69').
2. Airplane mode enables users to not broadcast their location. If AirplaneMode is on, then the GPS data will not be available but should still return a 200 and a description "SUCCESS: Location not available: Please turn off airplane mode" as well as the other details about this product. Only long and lat should be blank. If AirplaneMode is switched off again then the GPS data should be present and the SUCCESS message should reappear.
3. The status flag will be 'Active' for both devices as long as lat and long data is present. When no GPS data is available, e.g., Airplane mode is on, then the status should read 'Inactive'.
4. Battery life is also reported as being 'Full' if it is greater than or equal to 98%, 'High' 60% or higher, 'Medium' for 40% or higher, 'Low' for 10% or higher and 'Critical' if below 10%.

Unhappy Paths

1. If the AirplaneMode is off and there is no GPS data available in the csv file the status should be 400:

   (GPS not reported)

HTTP Status	400

Body	<pre>{
	  "description": "ERROR: Device could not be located"
	}</pre>

2. If the user provides an Id that is not present in the csv file, then a 404 should result:

   (ID not found)

HTTP Status	404

Body	<pre>{
	  "description": "ERROR: Id <insert productId> not found"
	}</pre>

1.1.3 Part 3 MVP+2: Dynamic activity-tracking

Following the release of 1.1.2 users of the CyclePlusTracker have requested an easy way to determine if they are actively cycling or whether they are resting or stopped at a traffic light, etc. The design team and architects have suggested modifying the functionality of the status field to do this. Basically, they would like to have the status remain in Inactive as long as three consecutive sets of GPS coordinates are exactly the same. If the device is just starting up and there are not enough readings to compute three GPS coordinates then they would like to see the Status read 'N/A' instead.

NB These changes should only apply to the CyclePlusTracker whilst the already implemented functionality for the GeneralTracker should remain the same as in part 1.1.2.