**JVM Coding Challenge**

This challenge consists in designing and implementing a new system intended to be used in an e-commerce website to present some products. This simple system is composed of 2 services, each of them exposing a number of specific REST endpoints.

The design of the services must take into account the need to cope with high volume and low latency. Moreover, it is essential to consider the resiliency of the system including, but not limited to, the high availability of all its components.

**Basic architecture principles**

- **Reusability**: design the components of the system in a way that makes them reusable.

- **Correctness**: always consider data security and integrity

- **Security**: take into account general IT security principles

- **Cost**: the solution should treat core operational database as a costly resource that needs to be used economically.

- **Design for failure**: your solution needs to be resilient, and exceptions handled correctly

**Requirements**

With the information given you should develop the following 2 micro-services:

* **Review Service:**
  1. implement CRUD operations for the resource:
  2. **/review/{product\_id}** , (e.g. AB1234), and the response is a JSON with the following data: **Product ID, Average Review Score, Number of Reviews.**
  3. In order to protect the service and prohibit data tampering, authentication is needed to protect the write operations.
  4. Choose any datastore for persisting the data – that can be easily deployed or installed with the application. The datastore should contain seeded data for a few products
* **Product Service:**
  1. The service will expose the resource **/product/{product\_id}**, only supporting GET.
  2. The response should be an aggregation of a live API (for the purpose of this coding challenge you can use the real public API provided by https://www.adidas.co.uk/api/products/{product\_id})
  3. Sample product IDs: M20324, AC7836, C77154, BB5476, B42000, etc.

NOTE: by **aggregation**, it means that the JSON in the response should contain the fields retrieved from the example live product API + the fields retrieved by the Review service

**Primary tech stack**

* Java + Spring Boot 2.X (for REST API)
* Database (any of your choice, SQL or NoSQL)
* Docker

**Expected outcome**

* Develop this application with a microservice approach: all services should run independently.
* Write API tests for your endpoints.
* Include a dockerized version that can be run with a simple *docker-compose* command (necessary Dockerfiles to be also included in the solution delivered)
* The coding challenge should build (both maven and gradle accepted) and run locally in any machine with a JDK, Docker and some standard tools
* The system will be delivered as a link to a public Git (Github, GitLab, etc) repo that we can simply pull and build

**BONUS**: Create a test integration suite for both services.