AMARO

BACK-END CHALLENGE

AMARO is a digitally native fashion brand that sells the latest trends at disruptive prices. AMARO sells omnichannel on amaro.com and in digitally immersive brick-and-mortar Guide Shops.

The objective of this test is to evaluate the knowledge of JAVA and Programming Logic.

Rating criteria:
The project will be evaluated by the AMARO development team, which will be based on the following criteria:

** Organization **: Folder and file structure, separation of concerns;
** Maintainability **: Readability, comments, simplicity, modularity;
** Implementation **: Logical implementation, use of object orientation and design patterns;
** Performance **: Execution and / or response time;
** Quality **: Development and execution of unit tests;
** Documentation **: Installation and execution instructions; internal working descriptions;

JAVA and Programming Logic

1) Similarity between products

Requirements:

- Create an application that can run locally exposing 2 REST APIs.
- Create unit tests
- Send the application through some public repository (GitHub, Bitbucket or others)

Consider a payload in the JSON format, which contains a list of 20 products, where each product has three attributes:

- "id", which is an integer and serves as the unique identifier of the product
- "name", which is a string with the product name
- "tags", which is an array of strings with product characteristics

Example:

```
{
      "id": 58345,
      "name": "Saia Maxi Chiffon Saint",
      "tags": ["balada", "metal", "delicado", "descolado"]
}
```

a) Tags Vector

Consider that the total number of possible tags that can characterize a product is 20.

With their respective array indexes, they are:

0: neutro

1: veludo

2: couro

3: basics

4: festa

5: workwear

6: inverno

7: boho

8: estampas

9: balada

10: colorido

11: casual

12: liso

13: moderno

14: passeio

15: metal

16: viagem

17: delicado

18: descolado

19: elastano

The first part of this exercise consists of developing an API that receives a payload of products and for each product creates a vector of tags of dimension 20, where each position of the array corresponds to a characteristic and has a value equal to 1 if the product has the characteristic, otherwise the value is equal to zero.

For example, if a product has all 20 possible characteristics, its tag vector will contain only numbers 1, such as:

```
(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
```

On the other hand, if the product has only the characteristics "neutral" (index 0), "veludo" (index 1) and "balada" (index 9), its tag vector will be:

```
(1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0)
```

In this first part of the exercise, develop an API that receives the following payload (available here: https://goo.gl/86o0pL) and returns payload that contains the same products, but with an additional attribute, which is the vector of tags of the product.

For example, each product in the return payload should have the tagsVector attribute, as follows:


```
{
	"id": 58345,
	"name": "Saia Maxi Chiffon Saint",
	"tags": ["balada", "metal", "delicado", "descolado"],
	"tagsVector": [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0]
}
```


b) Similar Product Finder

In this second part of the exercise, a search API for similar products will be developed for a given input product. The API should receive as input the id (the unique identifier of the product) and the payload generated in the previous exercise, which contains the array of tags (tagsVector) in each product.
For the product supplied in the input (through the id), the API should fetch the product in the input payload of the search engine) (hint: index the products by the id in the deserialization operation of the input payload, so as not to go through the entire payload in search for the product) and calculate the similarity between the input product and all other products, and return a list with the three most similar products.
The similarity between two products can be calculated taking into account the Euclidean distance between their tag vectors, and the similarity is inversely proportional to the distance.
That is, we can define the similarity S between two products p1 and p2, which have vectors of tags v1 and v2 of dimension N, such as:

S = 1/(1 + D), where

D = sqrt((v1[0] - v2[0])^2 + (v1[1] - v2[1])^2 + .. + (v1[N-1] - v2[N-1])^2) is the distance between the vectors v1 and v2

Search result example:


```
[
	{
		"id": 54354,
		"name": "Saia Maxi Chiffon Saint",
		"similarity": 0.78
	},
	{
		"id": 32432,
		"name": "Saia Lápis Cetim Stretch",
		"similarity": 0.12
	},
	{
		"id": 15083,
```

```
            "name": "Saia Zíper Deepa",
            "similarity": 0.05
        }
]
```


Note 1: The above similarity values serve only as an example and do not represent the values that should be obtained with the supplied products.

Note 2: Do not use external libraries for the Euclidean distance calculation.