Warehouse API – Spring boot

In this challenge, you are part of a team that is building a warehouse platform. One requirement is for a REST API service to manage products using Spring boot. You will need to add functionality to add and show product information as well as list all the products in the system. The team has come up with a set of requirements, including the API format, response codes, and the structure for the queries you must implement.

The definitions and detailed requirements list follow. You will be graded on whether your application performs data retrieval and manipulation based on given use cases exactly as described in the requirements.

Each product has the following structure:

- *id*: the unique ID of the product [int]
- *name*: the name of the product [String]
- *description*: the description of the product [String]
- *vendor*: the product vendor name [String]
- *price*: the price of the product [int]
- *stock*: the number of product items in the warehouse [int]
- *currency*: the currency of the product (either EUR or USD) [String]
- *image_url*: the URL of the product image [String]
- *sku*: the product stock-keeping unit (SKU) must be unique [String]

Sample product JSON

```
{
   "name": "Pen",
   "description": "Black Ball Point",
   "vendor": "Parker",
   "price": 1010,
   "stock": 5,
   "currency": "EUR",
   "image_url": "https://via.placeholder.com/150",
   "sku": "SKU001"
}
```

APIs

The following APIs need to be implemented:

1. Adding a new product- **POST** request should be created to add a new restaurant. The API endpoint would be /product/add. The request body contains the details of the product. The product should have a unique id. HTTP response should be 201. If some data validations failed and the data is not inserted into the collection, then you should send a response code of **400**.

2. Updating product's price and stock -it can be updated by **PUT** request to endpoint /product/{id}. The request body would contain price and stock. If id validations failed and the data is not updated into the collection, then you should send a response code of 400.

3. Getting all products which are greater than or equal to the given rating - **GET** request to endpoint /product/get should return the entire list of products in that warehouse. The HTTP response code should be 200. If no product exists with this condition, return a response code of 400.

4. Get the products according to given value- **GET** request to endpoint /product/vendor with request param ->value should return products as per the given value (vendor). The HTTP response code should be 200. If no restaurant exists, return a response code of 400.

5. Deleting product by id - **DELETE** request to endpoint /product/{id} should delete the corresponding product. The HTTP response code should be 200. If id validations failed and the data is not deleted from the collection, then you should send a response code of 400.

Instructions

- The install file will get installed automatically. Incase if it is not installed, kindly install the required dependencies by running '**bash install.sh**' from the project folder.
- For running the application use '**mvn spring-boot:run**'.
- For testing the application use '**mvn clean test**'.
- If you are getting port already in use error, open terminal and execute '**fuser -k 8080/tcp**'. If you find difficult doing this, go to application.properties and change server.port=8080 to any other port.
  E.g., server.port=8082
- After completing the hands-on, submit the test.