**Items Management Application**

# Overview

###### In this exercise, you are required to complete an incomplete application that manages **items for sale**.

The system contains the following items types:

* T-Shirt
* Beer
* Scotch Bottle
* Basketball
* Banana
* **More items can be added in the future...**

And every Items has the following properties:

* Item ID
* Item type
* Price
* daysToExpire - the number of days untill the item exprires.

## Items Update Policy

* The system updates every item once a day
* This update changes the price and daysToExpire for each item, based on a pre-defined rule\* associated with the item type.

(\*) these pre-defined rules are described later

## Implementation Notes:

This exercise is implemented as a Java Spring Boot application, built with Gradle (a build tool similar to Maven)

The excercise contains Unit Tests and Rest Api Tests which you will need to implement and run to make the system work properly.

All the existing test currently fail, and your task is to make them work, while implementing parts of the system logic along the way.

Note that the existing tests only cover the most basic scenarios, and you are free to add more tests as needed to make sure the system works correctly.

# Tasks

Please complete the tasks below.

You are free to work locally or to commit the code to a Git repository of your choice. Whatever you feel comfortable with.

### Guideline and Limitations

When working on the tasks below:

* you are allowed change existing code and add new code (including new classes or packages)
* you are NOT allowed to change the ItemService interface.

## Before you start:

* Open the Project using your preferred IDE
* If you don’t have Gradle installed will need to install it from <https://gradle.org/>
* Run the application by running **ItemApplication** class.   
  You should see the following output at the Console output in your IDE:

|  |
| --- |
| Current items: [Item{id=1, type='T\_SHIRT', daysLeftToExpire=10, price=20}, Item{id=2, type='SCOTCH\_BOTTLE', daysLeftToExpire=2, price=0}, Item{id=3, type='BEER', daysLeftToExpire=5, price=7}, Item{id=4, type='BASKETBALL', daysLeftToExpire=0, price=50}] |

## Task 1 – Implement ItemsController (part 1)

Implement these methods in the existing ItemsController class:

* findAllItems
* getItem
* createItem

**Note:** at this point, there is no need to implement the following methods, as these methods will be implemented in Task 5.

* dailyUpdateItems
* setItemRule

#### Definition of Done:

All the relevant tests in *ItemRestIntegrationTest* should pass successfully.

**Important**: that before running these rest tests to work, you must launch the ItemAppliction.

## Task 2 – Implement logic for updating items on a daily basis

### Every item should be updated on a daily basis, where the logic for each update is determined according to a predefined rule that is associated with each item type.

Each item can be set with one of the following update rules

1. **FixedValue**Price and daysToExpire do not change
2. **GainsValueWithAge**Price increases by 1 every day, and daysToExpire does not change
3. **LosesValueWithAge**Both Price and daysToExpire decreases by 1 every day
4. **LosesValueWithAgeQuickly**Price decreases by 3 every day, and daysToExpire deceases by 1 every day

In addition, the following is always true, so matter which Update Rule is associated with an item type:

1. Item price is never negative
2. Item price is never higher then 50
3. Items that loses value, loses value twice as fast once they expire (daysToExpire = 0)

Next, implement the code that associates item types to the predefined rules defined above, when the application starts.

To do that you will need to implement the *setItemRule* method in *InMemoryItemService* class

Here is the association between item types and predefined rules:

* + T-Shirt: **FixedValue**
  + Beer: **LosesValueWithAge**
  + Scotch Bottle: **GainsValueWithAge**
  + Basketball: **FixedValue**
  + Banana: **QuicklyLosesValueWithAge**

#### Definition of Done:

Unit tests that demonstrate the update rules logic defined for the different item types work as excepted.