

## 1. MyRetail REST API - Objectives

MyRetail RESTful service offers the following features

1. Retrieve the Product and Price information for a given product ID
2. Ability to modify the price information in the database

## 2. Frameworks/Tool/Technologies:

1. Spring Boot Rest Template: to expose REST Services, and to develop a REST client to consume the product information from external API.
2. MongoDB: To store the product price information.
3. Docker: To build and manage the deployment artifacts and to create/manage the MongoDB
4. Mockito: Unit Testing
5. PostMan : Unit testing
6. Maven: Build Configuration
7. GIT: Source configuration

\*\*\*\*\*Services:

## 3. Services - Implementation

### 3.1 Get Product Information:

The consumer can do a GET request at the path `"/products/{id}"` for a product detail to `"redsky.target.com"` and retrieves the product description, and is appended to the available price and name information. For a product with product id `'13860428'`, the sample JSON output is as shown below

```
{"id":13860428,"name":"The Big Lebowski (Blu-ray  
(Widescreen)","current_price":{"value": 13.49,"currency_code":"USD"}}
```

### 3. 2. Update Product Price in MongoDB:

The price information could be updated with this API. The user/client application can do a PUT request with input similar to the response received in GET and should be able to modify the price in the datastore. The request is done at the same path `"/products/{id}"`

####Sample Input: JSON Body - `{"value": 15.67,"currency_code":"USD"}`

## 4. Instructions to setup the environment

- Clone the code from the git repository

The API can be deployed in dev environment with IntelliJ/Eclipse or commandline, or as a Docker container

### 4.1 Dev Environment:

- Install Maven, Docker
- Docker: `'docker-compose -f docker-mongo.yml up -d'` (to install MongoDB as docker container), or setup a standalone MongoDB
- maven clean package - to create Jar file for REST endpoints
- execute: `mvn Sprint-boot:run.` to bring up the server

```
o.s.c.support.DefaultLifecycleProcessor : Starting beans in phase 0
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8081 (http
org.mongodb.driver.connection : Opened connection {connectionId{localValue:2, serverValue:155}} to localhost:27017
c.e-product.ProductPriceApiApplication : Started ProductPriceApiApplication in 29.735 seconds (JVM running for 30.392)
```

- or run the API in docker container  
`docker-compose up -d`

## 5 Testing Results

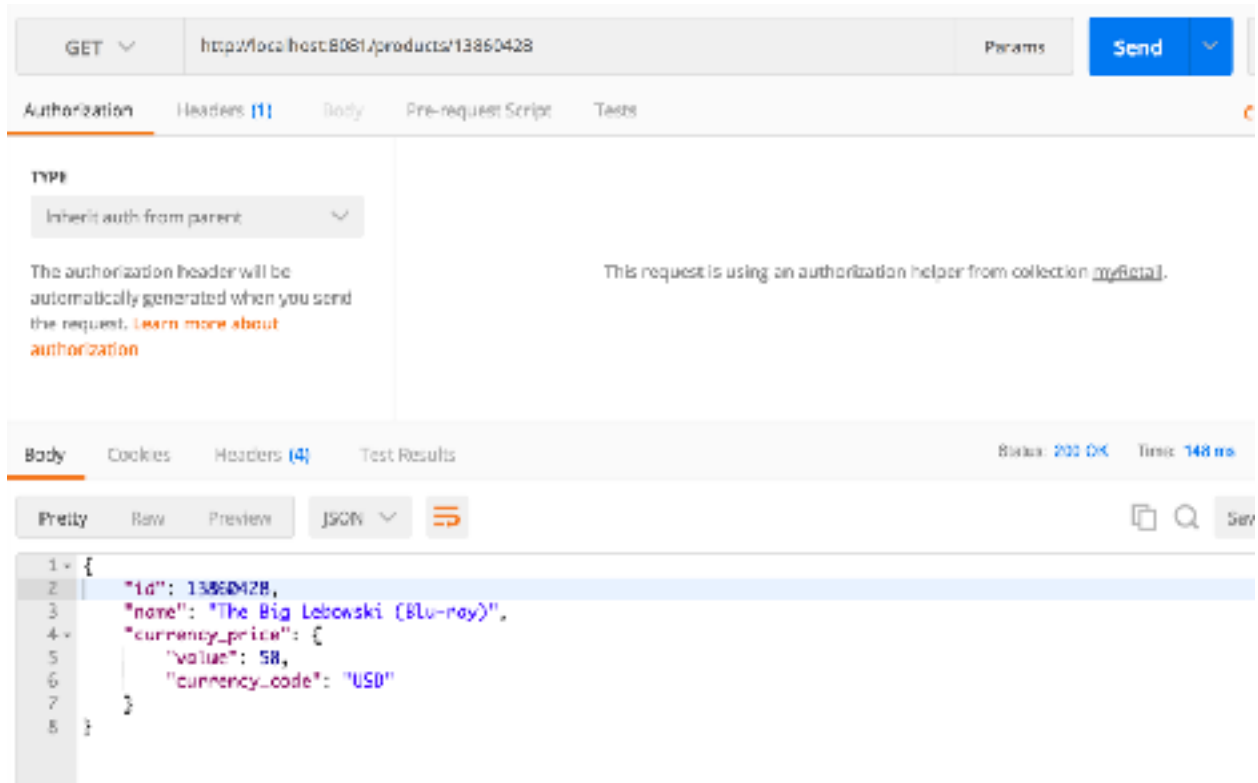
The testcases are implemented using 'mockito' framework under the folder `'src\test\java\ '`.

The test cases can be executed by running the command `'mvn test'`

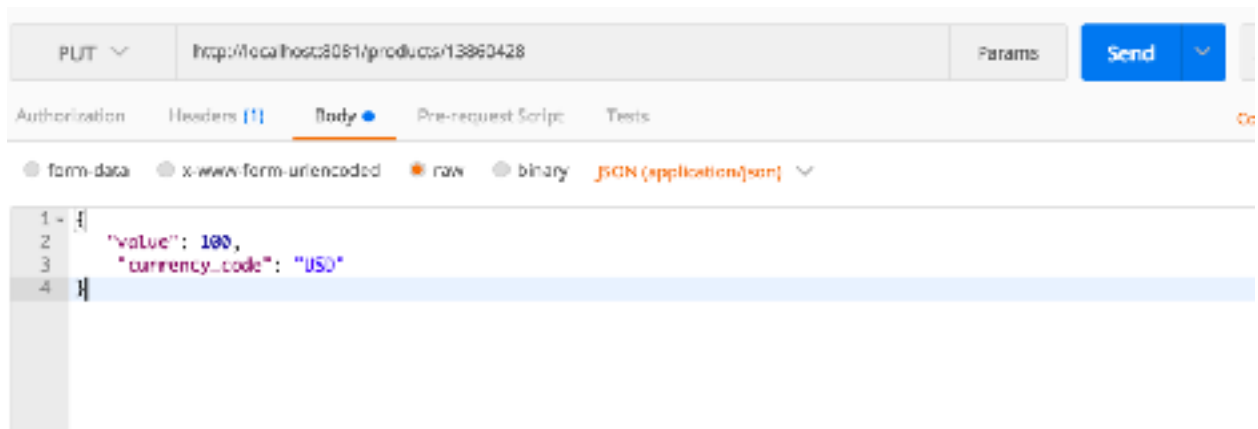
## ^^^PostMan UI:

Attached the Test results using Postman UI

### 1. GET Product Request



### 2. PUT: to update the price for a given product id



## 2.1 Updated Price: Response of the PUT API

The screenshot shows a REST client interface with a PUT request to `http://localhost:8081/products/13860428`. The request body is a JSON object: `{ "value": 100, "currency_code": "USD" }`. The response status is 200 OK, and the response body is a JSON object: `{ "id": 13860428, "name": "The Big Lebowski (Blu-ray)", "currency_price": { "value": 100, "currency_code": "USD" } }`.

```
PUT http://localhost:8081/products/13860428
```

```
{
  "value": 100,
  "currency_code": "USD"
}
```

Status: 200 OK Time: 829 ms

```
{
  "id": 13860428,
  "name": "The Big Lebowski (Blu-ray)",
  "currency_price": {
    "value": 100,
    "currency_code": "USD"
  }
}
```

## 2.2 Updated Data from the GET product API

The screenshot shows a REST client interface with a GET request to `http://localhost:8081/products/13860428`. The response status is 200 OK, and the response body is a JSON object: `{ "id": 13860428, "name": "The Big Lebowski (Blu-ray)", "currency_price": { "value": 100, "currency_code": "USD" } }`.

```
GET http://localhost:8081/products/13860428
```

Status: 200 OK Time: 181 ms

```
{
  "id": 13860428,
  "name": "The Big Lebowski (Blu-ray)",
  "currency_price": {
    "value": 100,
    "currency_code": "USD"
  }
}
```

## Current Automated Test Coverage:

