MyRetail Product Details Services

# High-Level Technical Specification

Version 1.0

01/06/2017

Table of Contents

[1. Use Case 1](#_Toc471500883)

[2. Technical Architecture 2](#_Toc471500884)

[2.1 Technology Stack 2](#_Toc471500885)

[2.2 Design Overview 2](#_Toc471500886)

[3. REST Service Specification 3](#_Toc471500887)

[3.1 Product Price Get Service 3](#_Toc471500888)

[3.2 Product Price Update Service 3](#_Toc471500889)

[4. Deployment Instructions 5](#_Toc471500890)

[4.1 MongoDB Container 5](#_Toc471500891)

[4.2 Build Code 5](#_Toc471500892)

[4.3 Code Container 6](#_Toc471500893)

[5. Service Test 7](#_Toc471500894)

[5.1 Product Price GET Service 7](#_Toc471500895)

[5.2 Update Product Price PUT Service 8](#_Toc471500896)

## Use Case

RESTful services need to be developed for myRetail Products. Service will retrieve Product and Price Details by Product ID. Also Product Price can be updated by passing the updated Product Price.

Product Details will be GET service and will be accessible by /products/{id}. This service will return Product Details in JSON format as below:

{"id":13860428,"name":"The Big Lebowski (Blu-ray) (Widescreen)","current\_price":{"value": 13.49,"currency\_code":"USD"}}

Product name and other details can be fetched from external service using the below URL:

<http://redsky.target.com/v1/pdp/tcin/13860428?excludes=taxonomy,price,promotion,bulk_ship,rating_and_review_reviews,rating_and_review_statistics,question_answer_statistics>

Product Price Details will be fetched from NoSQL store and merged with the result of the above HTTP Request before returning.

Product Price update will be PUT service and will be accessible by /products/{id} containing below JSON request:

{"id":13860428,"name":"The Big Lebowski (Blu-ray) (Widescreen)","current\_price":{"value": 13.49,"currency\_code":"USD"}}

This request will update the Product Price in NoSQL store.

## Technical Architecture

### Technology Stack

Following technologies will be used to develop and deploy the services:

* Java 1.7
* Spring BOOT 1.4.3
* Mockito
* Junit
* MongoDB
* Maven 3.3
* Tomcat 8
* Docker

### Design Overview

Spring REST will be used to build the services. Controller class will call Service Layer and Service will call DAO layer to get the Product Price data from MongoDB. Service Layer will also call the external link using RestTemplate to get the Product Details. After getting Product Details and Price, Service Layer will merge the details and return to Controller.

High level Class Details are given below:

* Controller Layer

ProductDetailsController: Controller class which contains GET service productPriceById and PUT service updateProductPrice.

* Service Layer

ProductDetailsServiceImpl: Service layer class which contains method processProductPriceDetails and updateProductPriceDetails. These methods call DAO layer to get and update Product Price details. This class also contain fetchProductDetails() to call external service to get Product details.

* DAO Layer

ProductDetailsDAOImpl: DAO class which calls MongoDB to get and update the Product data.

* Utility Layer

ProductDetailsUtil: Contains utility method to merge product details from external service and MongoDB.

## REST Service Specification

### Product Price Get Service

|  |  |
| --- | --- |
| Title | Get Product Price |
| URL | /products/{id}. E.g. http://<server>:<port>/products/13860419 |
| Method | GET |
| Data Params | NA |
| URL Params | Path param id is required. Must be valid Integer. E.g /products/13860419 |
| Success Response | Code: 200  Content: {  "id": 13860419,  "name": "The Rolling Stones: Some Girls - Live in Texas '78",  "productPrice": {  "value": 18.09,  "currencyCode": "USD"  }  } |
| Error Response | Code: 404 Not Found |

### Product Price Update Service

|  |  |
| --- | --- |
| Title | Update Product Price |
| URL | /products/{id}. E.g. http://<server>:<port>/products/13860419 |
| Method | PUT |
| Data Params | {  "id": 13860419,  "name": "The Rolling Stones: Some Girls - Live in Texas '78",  "productPrice": {  "value": 19.09,  "currencyCode": "USD"  }  } |
| URL Params | Path param id is required. Must be valid Integer. E.g /products/13860419 |
| Success Response | Code: 200  Content: {  "id": 13860419,  "name": "The Rolling Stones: Some Girls - Live in Texas '78",  "productPrice": {  "value": 19.09,  "currencyCode": "USD"  }  } |
| Error Response | Code: 404 Not Found |

## Deployment Instructions

Code has been uploaded in GITHUB in the URL below:

<https://github.com/arnabkarmakar2008/productDetails>

### MongoDB Container

Get MongoDB image from DockerHub. Run MongoDB container with name **myRetailDB**. Create DB **myretail** and collection **productprice** inside the container. Insert sample rows in the collection as below:

db.productprice.insert({\_id:13860428,price:100.13,code:”USD”})

db.productprice.insert({\_id:13860421,price:10.13,code:”USD”})

db.productprice.insert({\_id:13860422,price:9.13,code:”USD”})

db.productprice.insert({\_id:13860423,price:18.13,code:”USD”})

db.productprice.insert({\_id:13860424,price:109.14,code:”USD”})

db.productprice.insert({\_id:13860425,price:101.13,code:”USD”})

db.productprice.insert({\_id:13860426,price:102.13,code:”USD”})

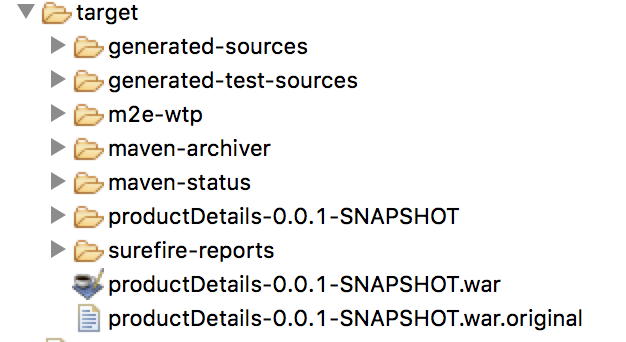
db.productprice.insert({\_id:13860427,price:103.13,code:”USD”})

### Build Code

Clone or download the code from GIT. In the code download directory, execute the below Maven command to package the code:

**mvn –U clean install**

Above will create WAR file productDetails-0.0.1-SNAPSHOT.war in target directory.



### Code Container

Copy WAR file productDetails-0.0.1-SNAPSHOT.war and Dockerfile in a directory. Run the below docker command to create the image:

**docker build --rm --no-cache -t myretail:1.0 .**

myretail image will be created. Now run the image by linking MongoDB container **myRetailDB.**

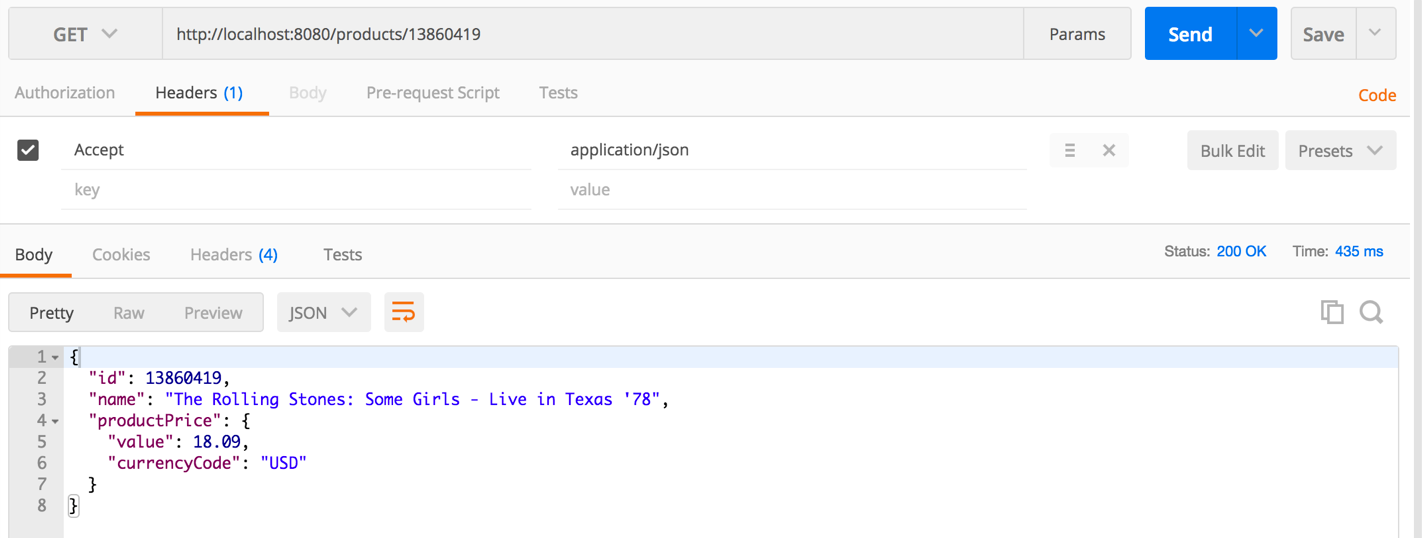
**docker run -p 8080:8080 -e spring.profiles.active=prod -e spring.data.mongodb.uri=**[**mongodb:// myRetailDB:27017/myretail**](mongodb://mymongo:27017/myretail) **-v /logs:/logs —link myRetailDB: myRetailDB —name myretail myretail:1.0**

## Service Test

Use SOAPUI or POSTMAN to test the services.

### Product Price GET Service

Use the URL like http:<host>:<port>/products/138604190 and Method=GET in PostMan/SOAPUI. Set Header Accept=application/json.



Service should return HTTP Status 200 and Product Price in JSON as below:

{

"id": 13860419,

"name": "The Rolling Stones: Some Girls - Live in Texas '78",

"productPrice": {

"value": 18.09,

"currencyCode": "USD"

}

}

### Update Product Price PUT Service

Use the URL like http:<host>:<port>/products/138604190 and Method=PUT in PostMan/SOAPUI. Set Header Accept=application/json and Content-Type=application/json. Set Body as below:

{

"id": 13860419,

"name": "The Rolling Stones: Some Girls - Live in Texas '78",

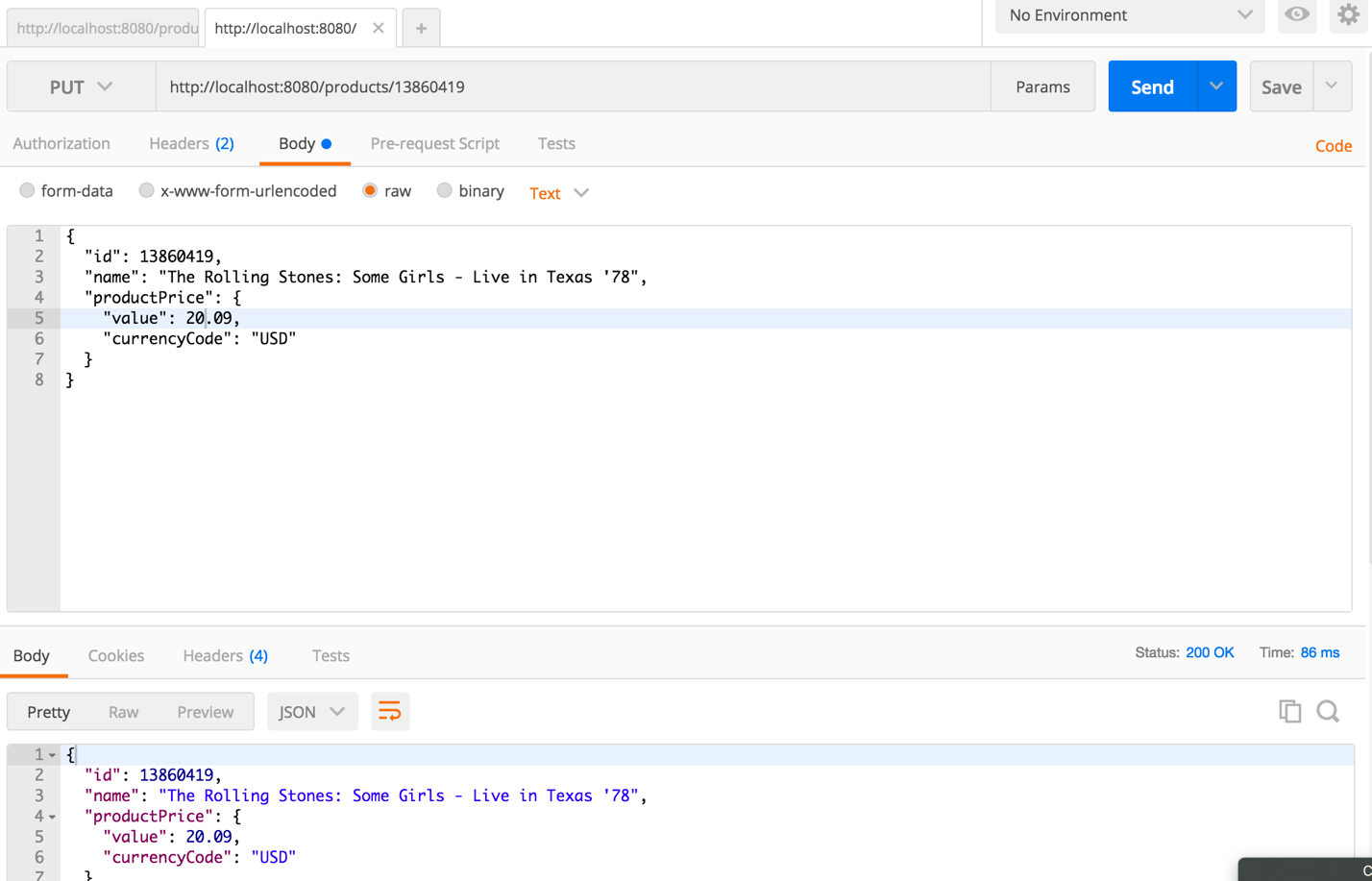
"productPrice": {

"value": 20.09,

"currencyCode": "USD"

}

}



Service should return HTTP Status 200 and Product details as passed in the request.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*