myRetail RESTful service

myRetail is a rapidly growing company with HQ in Richmond, VA and over 200 stores across the east coast. myRetail wants to make its internal data available to any number of client devices, from myRetail.com to native mobile apps.

The goal for this exercise is to create an end-to-end Proof-of-Concept for a products API, which will aggregate product data from multiple sources and return it as JSON to the caller.

Your goal is to create a RESTful service that can retrieve product and price details by ID. The URL structure is up to you to define, but try to follow some sort of logical convention.

Build an application that performs the following actions:

* Responds to an HTTP GET request at /products/{id} and delivers product data as JSON (where {id} will be a number.
* Example product IDs: 15117729, 16483589, 16696652, 16752456, 15643793)
* Example response: {"id":13860428,"name":"The Big Lebowski (Blu-ray) (Widescreen)","current\_price":{"value": 13.49,"currency\_code":"USD"}}
* Performs an HTTP GET to retrieve the product name from an external API. (For this exercise the data will come from redsky.target.com, but let’s just pretend this is an internal resource hosted by myRetail)
* Example: <http://redsky.target.com/v2/pdp/tcin/13860428?excludes=taxonomy,price,promotion,bulk_ship,rating_and_review_reviews,rating_and_review_statistics,question_answer_statistics>
* Reads pricing information from a NoSQL data store and combines it with the product id and name from the HTTP request into a single response.
* BONUS: Accepts an HTTP PUT request at the same path (/products/{id}), containing a JSON request body similar to the GET response, and updates the product’s price in the data store.

Solution

MyRetail RESTful webservice is implemented basically to provide following 2 services.

1. Provides product information (Name, Price and Currency) for a valid product ID
2. Updates price information for a valid Product ID.

These services are implemented using HTTP GET and PUT methods respectively. In order to get Product name, this service also makes REST web service call to MyRetail service. This service is built with Spring Boot with Maven dependency and Mockito for unit testing.

Prerequisites:

1. Eclipse Java EE IDE with Spring Tool Suite Plugin
2. Spring Data JPA
3. MySQL data store
4. Postman client (Google Chrome browser plugin) for testing.

Install MySQL data store:

Download and install MySQL data store as per the instructions available [here](https://dev.mysql.com/doc/refman/5.7/en/windows-installation.html).

Once installed, make sure the server is running on port number 3306, set database username, password as root and admin respectively. If you wish to use different details, you change configuration in application.properties file in the project resources folder.

Install Package:

Run the below command to install the application on server under the target folder.

java -jar MyRetailService-0.0.1-SNAPSHOT.jar

Test the webservices using Postman client:

Install Postman plugin to Google Chrome browser to test following scenarios.

1. GET Service: [<http://localhost:8080/products/13860428>]
2. Select GET option in Postman, then use the URL mentioned above, set Content-Type as application/json in the header and hit Send button.
3. Response should be received as follows.

{

"productID": 13860428,

"productName": "The Big Lebowski (Blu-ray)",

"priceInfo": {

"price": 18.99,

"currencyCode": "USD"

}

}

1. PUT Service: [<http://localhost:8080/products/13860428>]
2. Select PUT option in Postman, then use the URL mentioned above and hit, use following Request Body and hit Send button.

{

"productID": 13860428,

"productName": "The Big Lebowski (Blu-ray) (Widescreen)",

"priceInfo": {

"price": 18.99,

"currencyCode": "USD"

}

}

1. Response should be received as follows.

{

"value": 200,

"message": "Price has been updated successfully"

}

Testcases:

Use the command (*mvnw clean install*) to execute the test cases in the root directory of the application.

Screenshots:

Test screens are included in the Test Screens folder.