**Test Cases – Restful API Implementation – Online Store**

All tests have been performed using the Postman Extension for Google Chrome. All input for POST and PUT requests have to be given in JSON Format. Otherwise the API will return:

{

"error": "Input not in correct format"

}

# Database online-store

## Table structure for table products

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Null** | **Default** |
| ***id*** | int(11) | No |  |
| name | varchar(50) | No |  |
| Description | Text | Yes |  |
| sku | varchar(50) | No |  |
| salePrice | float | No |  |
| regularPrice | float | No |  |
| created | datetime | No |  |
| modified | datetime | No |  |

### User Access

Only administrators can access the API. I have made a user with privileges to test the authorization functionality. The administrator user and password to access the API are:

Username: ‘kanishk’

Password: ‘password’

Test Case 1:

username: random

password: random

Output:

Please login with administrator credentials

Test Case 2:

username: kanishk2

password: password2

Output:

Only Administrators are allowed.

Test Case 3:

username:kanishk

password: password

Output:

Successfully Logged In

### Listing Products

Various query parameters can be given to retrieve the results intended in a particular way. Any of the query parameters can be combined in any combination to retrieve the result.

Test Case 1:

GET http://localhost/online-store/products

Output:

The first ten products with all fields order by their id are returned. Ten records are returned since by default ten resources are fetched unless you pass a per\_page parameter in the query. And by default resources are ordered by id.

Conclusion: Result as expected.

Test Case 2:

GET http://localhost/online-store/products?fields=name,sku&page=1&per\_page=5

Output:

Only name and sku is returned for each product. The first five records are skipped and the next five are returned since per\_page parameter is given as 5 and page parameter is specified as 2. And by default resources are ordered by id.

Conclusion: Result as expected.

Test Case 3:

GET http://localhost/online-store/products?sku=testsku

Output:

Those records are returned which have sku=testsku.

Conclusion: Result as expected.

Test Case 4:

GET http://localhost/online-store/productsdf

Output:

STATUS 400

Content-Type: application/json

{

"error": “Bad Request”

}

Conclusion: Result as expected since ‘products’ is misspelled.

### Listing a product

Test Case 1:

GET http://localhost/online-store/products/1

Output:

STATUS 200

Content-Type: application/json

{

"id": "1",

"name": "testting\_product",

"description": "test desc",

"sku": "testsku",

"salePrice": "10",

"regularPrice": "20",

"created": "2016-06-07 12:38:00",

"modified": "0000-00-00 00:00:00"

}

Conclusion: Result as expected.

Test Case 2:

GET http://localhost/online-store/products/1?fields=name,sku

Output:

STATUS 200

Content-Type: application/json

{

"name": "testting\_product",

"sku": "testsku"

}

Conclusion: Result as expected since only the fields requested are returned.

Test Case 3:

GET http://localhost/online-store/products/20

Output:

STATUS 404

Content-Type: application/json

{

"error": “resource not found”

}

Conclusion: Result as expected since the resource with id = 20 does not exist

Test Case 4:

GET http://localhost/online-store/productsdfg/20

Output:

STATUS 400

Content-Type: application/json

{

"error": “Bad Request”

}

Conclusion: Result as expected since ‘products’ is misspelled.

### Inserting a Product

POST method is only allowed on a collection. If it is attempted on a resource, it will always return “Method not allowed on a resource”.

Test Case 1:

POST http://localhost/online-store/products

Content-Type: application/json

{

"name": "Test product 4",

"description": "Test product 4 desc",

"sku": "test\_pr\_4",

"salePrice": "32",

"regularPrice": "35"

}

Output:

STATUS 201

Content-Type: application/json

{

"id": "6",

"name": "Test product 89",

"description": "Test product 89 desc",

"sku": "test\_pr\_89",

"salePrice": "32",

"regularPrice": "35",

"created": "2016-06-08 17:54:28",

"modified": "0000-00-00 00:00:00"

}

Conclusion: Result as expected.

Test Case 2:

POST http://localhost/online-store/products/5/

Content-Type: application/json

{

"name": "Test product 5",

"description": "Test product 5 desc",

"sku": "test\_pr\_5",

"salePrice": "522",

"regularPrice": "540"

}

Output:

STATUS 400

Content-Type: application/json

{

"error": "Method Not Allowed on a Resource"

}

Conclusion: Result as expected since the POST Method is not allowed on a Resource.

### Updating a Product

Update method is only allowed on a single resource. If it is attempted on a collection, it will always return “Method not allowed on a collection”.

Test Case 1:

PUT http://localhost/online-store/products/4

Content-Type: application/json

{

"id": "4",

"name": "Test product 41",

"description": "Test product 4 desc",

"sku": "test\_pr\_4",

"salePrice": "323",

"regularPrice": "353"

}

Output:

STATUS 200

Content-Type: application/json

{

"id": "4",

"name": "Test product 41",

"description": "Test product 4 desc",

"sku": "test\_pr\_4",

"salePrice": "323",

"regularPrice": "353",

"created": "2016-06-07 18:08:18",

"modified": "2016-06-07 18:23:03"

}

Conclusion: Result as expected. The resource gets updated.

Test Case 2:

PUT http://localhost/online-store/products/20

Content-Type: application/json

{

“id”:20,

"name": "changes\_name",

"description": "Changes Desc",

"sku": "changed\_sku"

}

Output:

STATUS 404

Content-Type: application/json

{

"error": "resource not found"

}

Conclusion: Result as expected since there is no product with id = 20

Test Case 3:

PUT http://localhost/online-store/products/1

Content-Type: application/json

{

"id":2,

"name": "changes\_name",

"description": "Changes Desc",

"sku": "changed\_sku"

}

Output:

STATUS 400

Content-Type: application/json

{

"error": “Bad Request”

}

Conclusion: Result as expected since the id in the URL(1) does not match the id in the JSON Input(2)

Test Case 4:

PUT http://localhost/online-store/products

Content-Type: application/json

{

"id":2,

"name": "changes\_name",

"description": "Changes Desc",

"sku": "changed\_sku",

"salePrice": "456",

"regularPrice": "467"

}

Output:

STATUS 400

Content-Type: application/json

{

"error": “Method not allowed on a collection”

}

Conclusion: Result as expected since the PUT Method is not allowed on a collection.

### Deleting a Product

Delete method is only allowed on a single resource. If it is attempted on a collection, it will always return “Method not allowed on a collection”.

Test Case 1:

DELETE http://localhost/online-store/products/3

Output:

STATUS 200

Content-Type: application/json

{

"deleted": true

}

Conclusion: Result as expected. The resource gets deleted

Test Case 2:

DELETE http://localhost/online-store/products/20

Output:

STATUS 404

Content-Type: application/json

{

"error": “Resource not found”

}

Conclusion: Result as expected since there is no prosuct with id = 20

Test Case 3:

DELETE http://localhost/online-store/products

Output:

STATUS 404

Content-Type: application/json

{

"error": “Method not allowed on a collection”

}

Conclusion: Result as expected since the DELETE method is not allowed on a collection.