

E-mail: sdfasiullah924@gmail.com

Contact: 7097450534

SportyShoes.com

RESTful Spring Boot E-Commerce portal with JPA Data Base:

In this project, I developed Restful Spring Boot E-Commerce portal named SportyShoes.com that uses internal memory database i.e., Spring H2 Data Base to store all the information about the products, users who purchased it & this information can be fetched up by date & Customer name as well.

Here, Admin can manage all the products by categorizing them as brand, color, size & price.

Admin can see the purchase list filtered by date & category.

Tools:

1. Agile-Scrum-Jira
2. Spring Tool Suite | 4
3. Spring boot initializer
4. Postman
5. Git
6. GitHub

Steps for Execution:

1. Clone the project from the repository

GitHub Link: https://github.com/SyedFasiHussaini/Phase3_Project_SportyShoes.com.git

As I have developed the RESTful API, it can be run on postman tool.

2. I have used SWAGGER in my spring boot RESTful API project so that the project documentation can be well understood and can easily be verified.
3. URL for the SWAGGER documentation:

<https://localhost:9200/swagger-ui.html>

4. All the endpoints and example data will be present in that link and an option to try it out, Put the data into which will be in JSON format and simply try it out.

<u>Headers</u>	<u>End-Point</u>
1. Get/Post/Put to get, add & update all Shoes-----	/shoes
2. Get Shoes by Category:	
i) Get shoes by brand -----	/shoes/brand/{brand}
ii) Get shoes by color -----	/shoes/color/{color}
iii) Get shoes by size -----	/shoes/size/{size}
iv) Get shoes by price -----	/shoes/price/{price}
3. Get/Delete shoes by Id-----	/shoes/{id}
4. Post/Put to add & update user-----	/user
5. Get to get all users info. -----	/users
6. Get Users by:	
i) Get users by userId -----	/user/{userId}
ii) Get users by username-----	/user/username/{userName}
iii) Get users by userAge -----	/user/userAge/{userAge}
7. Get/Delete users by id -----	/user/{userId}

Screen Shots of the output result:

1. Database Storing all the different versions of products

The screenshot shows the H2 Console interface with the following details:

- Top Bar:** Phase3-Project-SportyShoes.com, WhatsApp, java - Consider defining a bean, H2 Console.
- Header:** localhost:9200/h2/login.do?jsessionid=ae66df11b08c5cc378d42ef46d5c089a, Auto commit checked, Max rows: 1000, Auto complete Off, Auto select On.
- Left Sidebar:** Database structure tree with nodes: jdbc:h2:~/test, CART, CART_SHOES, SHOES, USER, USER_SHOES, INFORMATION_SCHEMA, Sequences, Users. Version: H2 1.4.200 (2019-10-14).
- Central Area:** SQL statement input field containing "SELECT * FROM SHOES". Buttons: Run, Run Selected, Auto complete, Clear.
- Result Area:** A table titled "SELECT * FROM SHOES;" showing the following data:

PRODUCT_ID	BRAND	COLOR	PRICE	SIZE
1	Fastrack	White	8000	8
2	Reebok	White	7500	7
3	Nike	Black	7500	8
4	puma	Black	7500	8
5	Adidas	Black	7500	8
6	Fastrack	Blue	7500	8
7	Adidas	Black	7000	7

(7 rows, 8 ms)

Bottom Bar: Type here to search, system icons, ENG, 00:50, 24-08-2020.

2. Get method that fetches all the product details shown above in h2 DB in JSON format

The screenshot shows the Postman application window. In the top navigation bar, there are links for File, Edit, View, Help, New, Import, Runner, and My Workspace. A sign-in button is also present. The main workspace shows a search bar with 'Filter' and a list of recent requests under 'History'. One request, 'GET localhost:9200/shoes', is selected. The 'Body' tab is active, displaying a JSON response:

```
[{"productId":1,"brand":"Fastrack","color":"White","size":8,"price":8000}, {"productId":2,"brand":"Reebok","color":"White","size":7,"price":7500}, {"productId":3,"brand":"Nike","color":"Black","size":8,"price":7500}, {"productId":4,"brand":"Puma","color":"Black","size":8,"price":7500}, {"productId":5,"brand":"Adidas","color":"Black","size":8,"price":7500}, {"productId":6,"brand":"Fastrack","color":"Blue","size":8,"price":7500}, {"productId":7,"brand":"Adidas","color":"Black","size":7,"price":7000}]
```

The status bar at the bottom indicates 'Status: 200 OK', 'Time: 52 ms', and 'Size: 661 B'. The taskbar at the bottom of the screen shows various application icons.

3. Get method that fetches all the products by brand

The screenshot shows the Postman application window. The 'History' tab is selected in the sidebar. A request 'GET localhost:9200/shoes/brand/Fastrack' is selected. The 'Body' tab is active, displaying a JSON response:

```
1  [
2    {
3      "productId": 1,
4      "brand": "Fastrack",
5      "color": "white",
6      "size": 8,
7      "price": 8000
8    },
9    {
10      "productId": 6,
11      "brand": "Fastrack",
12      "color": "blue",
13      "size": 8,
14      "price": 7500
15    }
16  ]
```

The status bar at the bottom indicates 'Status: 200 OK', 'Time: 23 ms', and 'Size: 310 B'. The taskbar at the bottom of the screen shows various application icons.

4. Get Method that fetches based on price:

The screenshot shows the Postman application interface. In the top navigation bar, there are buttons for 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and a user icon. The main header displays 'My Workspace' and 'Invite'. On the left sidebar, under 'History', there is a list of API requests. A recent request is highlighted: 'GET localhost:9200/shoes/price/7500'. The main workspace shows an 'Untitled Request' with a 'GET' method selected. The URL is 'localhost:9200/shoes/price/7500'. Below the URL, tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible. The 'Body' tab is active, showing a JSON response with six products. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 19 ms', and 'Size: 517 B'. The Windows taskbar at the bottom shows various pinned icons and the date '24-08-2020'.

5. Get Method that fetches based on color:

This screenshot is identical to the one above, showing the Postman interface with a successful GET request to 'localhost:9200/shoes/price/7500'. The main workspace shows an 'Untitled Request' with a 'GET' method selected, URL 'localhost:9200/shoes/color/Black', and a JSON response containing six products. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 40 ms', and 'Size: 445 B'. The Windows taskbar at the bottom shows various pinned icons and the date '24-08-2020'.

6. User and Shoes detail added to h2 DB

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for 'File', 'Edit', 'View', and 'Help'. Below the navigation bar, there are buttons for '+ New', 'Import', 'Runner', and a dropdown for 'My Workspace' with an 'Invite' button. On the right side of the header, there are icons for 'No Environment', 'Sign In', and settings.

The main workspace is titled 'Untitled Request'. It shows a 'POST' request to 'localhost:9200/user'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "date": "2020-08-23T19:33:28.373Z",
3   "shoes": [
4     {
5       "brand": "Reebok",
6       "color": "white",
7       "price": 7500,
8       "productId": 8
9     }
]
```

Below the body, the response status is shown as 'Status: 200 OK' with a green checkmark, 'Time: 74 ms', and 'Size: 326 B'. There are buttons for 'Send', 'Save', 'Cookies', and 'Code'.

The left sidebar lists recent requests and responses. The 'History' tab is selected, showing the following log:

- POST localhost:9200/user
- GET localhost:9200/shoes/color/Black
- GET localhost:9200/shoes/price/7500
- GET localhost:9200/shoes/price/8000
- GET localhost:9200/shoes/price/7000
- GET localhost:9200/shoes/brand/Fastrack
- GET localhost:9200/shoes
- POST localhost:9200/user
- POST localhost:9200/user
- POST localhost:9200/user

At the bottom of the screen, there is a Windows taskbar with various pinned icons and system status indicators.

7. User and product details fetched:

This screenshot shows the Postman application interface again. The top navigation bar and workspace are identical to the previous screenshot.

The main workspace is titled 'Untitled Request'. It shows a 'GET' request to 'localhost:9200/users'. The 'Body' tab is selected, showing a JSON response:

```
1 [
2   {
3     "date": "2020-08-23T19:33:28.373Z",
4     "shoes": [
5       {
6         "brand": "Reebok",
7         "color": "white",
8         "price": 7500,
9         "productId": 8,
10        "size": 9
11      }
12    ]
13  ]
14 ]
```

Below the body, the response status is shown as 'Status: 200 OK' with a green checkmark, 'Time: 62 ms', and 'Size: 328 B'. There are buttons for 'Send', 'Save', 'Cookies', and 'Code'.

The left sidebar lists recent requests and responses. The 'History' tab is selected, showing the following log:

- GET localhost:9200/users
- GET localhost:9200/user
- POST localhost:9200/user
- GET localhost:9200/shoes/color/Black
- GET localhost:9200/shoes/price/7500
- GET localhost:9200/shoes/price/8000
- GET localhost:9200/shoes/price/7000
- GET localhost:9200/shoes/brand/Fastrack
- GET localhost:9200/shoes
- POST localhost:9200/user

At the bottom of the screen, there is a Windows taskbar with various pinned icons and system status indicators.

8. Documentation that shows status 200 ok for delete method

DELETE /shoes/{id} deleteShoesById

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	<input type="text" value="6"/>	id	path	integer

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	OK		
204	No Content		
401	Unauthorized		
403	Forbidden		

Curl

```
curl -X DELETE --header 'Accept: */*' 'http://localhost:9200/shoes/6'
```

Request URL

```
http://localhost:9200/shoes/6
```

Request Headers

```
{ "Accept": "*/*" }
```

Request Headers

```
{ "Accept": "*/*" }
```

Response Body

```
no content
```

Response Code

```
200
```

Response Headers

```
{ "connection": "keep-alive", "content-length": "0", "date": "Sun, 23 Aug 2020 19:43:36 GMT", "keep-alive": "timeout=60", "content-type": null }
```

GET /shoes/{id} getShoesById

Response Class (Status 200)

OK