

Project: SportyShoes.com

Developer: Siddharth Basu

- Project details:

SportyShoes is aiming to digitise a project SportyShoes.com which is a online ecommerce system capable of performing operations such as managing products for example adding, deleting, updating, retrieving on various conditions etc. The admin of the site can also define and redefine categories the shoes belong to. Also the admin of the site can manage all the users of the ecommerce website and also get all the purchases done on the site filtered on various parameters such as by product, category, user etc. The admin requires a login username and password for performing all these activities. The admin can change his/her username anytime he wants and can logout as well.

**Currently the system performs admin specific functionalities (as mentioned in the problem statement), a sample get all products endpoint is defined as “/products” which requires no admin login.

**All the admin REST endpoints have the prerequisite that an admin MUST first log in (“admin/login”). All admin endpoints start with “/admin”.

- Github Link:

https://github.com/siddharth-basu98/SportyShoes_SpringBootApplication

- Admin username and password (POST request to “localhost:XXXX(default 8080)/admin/login with following body)

```
{  
  "username": "root"  
  "password": "root"  
}
```

- After running SpringBoot App, to access swagger API documentation:

<http://localhost:8080/swagger-ui.html>

- Initial Data Entered via data.sql file in application.properties

How to run the app

Step 1:

If using HTTPS, run the following command on the terminal to clone the Sporty Shoes App made with SpringBoot, https://github.com/siddharth-basu98/SportyShoes_SpringBootApplication.git, otherwise if using SSH, git@github.com:siddharth-basu98/SportyShoes_SpringBootApplication.git

Step 2:

Using any IDE of choice, open the cloned project in the workspace as a SpringBoot project with Maven. The IDE upon reading the pom.xml file will download all the dependencies. Currently the app runs on the default embedded TOMCAT server port of 8080, if in case the port needs to be changed, add the following code in the “application.properties” file “**server.port=XXXX**”, where XXXX is any port which is free on your system.

Step 3:

The app used the in memory database H2 to store, retrieve, update data, the initial data is entered in the file “**data.sql**” in the “**application.properties**” file, to see the tables generated in H2, open **localhost:XXXX/h2** on your system, and make sure the JDBC url is “**jdbc:h2:mem:testdb**”. Upon pressing connect, all the tables with the initial data can be viewed.

Step 4:

The application for REST API documentation used **Swagger**. One can also test the APIs directly from swagger. To access the swagger API documentation page, visit “**localhost:XXXX/swagger-ui.html**” on your browser to see the complete list of API endpoints defined in the system.

Step 5:

The app majorly as required were to perform admin related tasks, **so to access any API starting with /admin which is most of them as it was the requirement, you first need to login by sending a POST request to “localhost:XXXX/admin/login”**, The request body for the admin login will be by default,

```
{  
  "username": "root"  
  "password": "root"  
}
```

You may change the password or even add new admin later on by first logging in with the above credentials.

Step 6:

Enjoy testing out all the APIs endpoints.

SPRING BOOT CONCEPTS AND TECHNIQUES USED:

- **Annotations:** Various annotations such as @RestController, @Service, @Repository, @Entity, @Autowired, @Configuration etc were used heavily or bean creation and other corresponding purposes.
- **Embedded Database H2:** The selected database for the system is the in memory small scale embedded database H2.
- **Embedded Server:** The application runs on the embedded server on default port 8080. However if the port needs to be changed add in the "application.properties" file "**server.port=XXXX**", where XXXX is any port which is free on your system.
- **Spring Data JPA:** The database operations are performed using the Spring Data JPA starter with SpringBoot. The Repository CRUD functions are all written using Spring Data JPA.
- **OneToMany andManyToOne (mapped by) annotations:** The relationships between the entities are defined using these annotations.
- **@Query and Joins:** Few complicated queries requiring a join between two entity tables, a @Query annotation and a **custom query written in JPQL** was used.
- **CASCADE delete and update:** For entities having a join with other entities, if any of these entities is changed or deleted, it cascades its effect on the dependant entities as well.
- **LAZY Fetching:** To reduced DB load, the fetch type for some services is set to a LAZY fetching.
- **Exception handling via ResponseStatusException:** The Repsonse Status exception method is used to throw errors upon getting bad request, or a request not found etc. Appropriate HTTP Status Codes and personalised messages are sent for each of these exceptions.

CORE JAVA CONCEPTS USED:

- **Layered Architecture and separation of concerns:** Constructed the application following the layered architecture by dividing the app into a controller layer for communicating with the API consumers, A model layer containing the structure of the entities we need to store, a service or business layer in between them to launch functions for altering or retrieving data from the DB, or else perform non DB manipulations, lastly a repository to actually manipulate and access the data stores.
- **Exception Handling:** Created the application using checked exceptions by developing an exception layer by extending the Exception class and throwing exceptions throughout the application to prevent the abnormal abortion of the program.
- **Interfaces:** Constructed the interfaces for the DAO and Service layers and instantiated model objects through the interfaces as a design principle.
- **Classes and Objects:** The class is a template consisting of data members and methods against which many objects of its type are instantiated. In our case a single file is saved as an object and has many data members.
- **Lambda functions:** The comparators in the application were designed using Lamda expressions to make the code more functional. The predicates inside the filter method were also written in lambda notation.
- **Data Hiding:** Most of the data members defined in the classes were defined as private with appropriate setters and getters so that any alien function isn't able to change its values from outside without proper access.
- **Throws and throw:** These keywords were aptly used in method definitions and inside functions to indicate the possibility of throwing an exception and actually throwing one if certain conditions are met.
- **Agile:** The agile framework was followed while doing the project by dividing the work to be done in sprints and implementing user stories in a phase wise manner.
- **Git and Github:** To have version control in our system, git and github was used.

Application Models/Entities:

1) Category

category_id	int (PRIMARY KEY)
name	String

2) Product

product_id	int (PRIMARY KEY)
name	String
brand	String
price	int
category_id	FOREIGN KEY References CATEGORY

3) User

user_id	int (PRIMARY KEY)
firstname	String
lastname	String
age	int
emailID	String
password	String

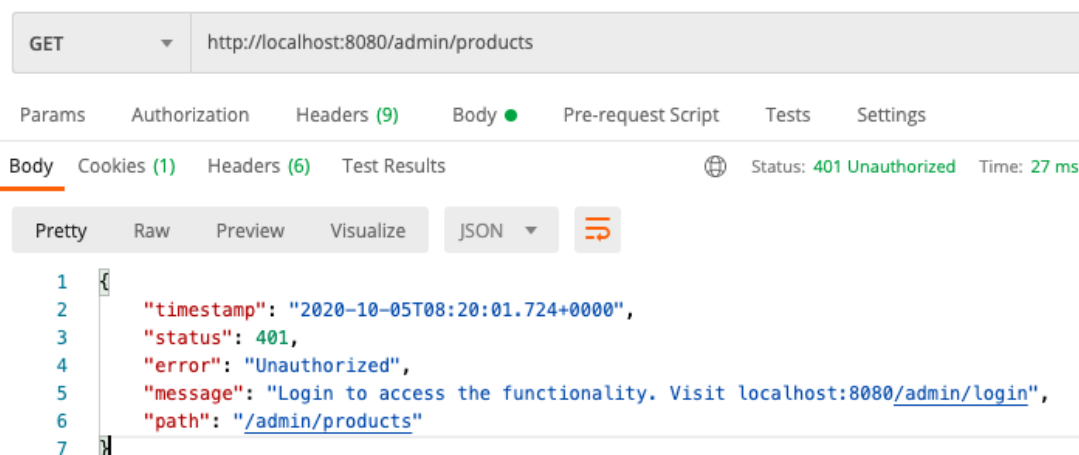
4) Purchase

purchase_id	int (PRIMARY KEY)
quantity	int
date	LocalDateTime
user_id	FOREIGN KEY References USER
product_id	FOREIGN KEY References PRODUCT

5) Admin

username	String (PRIMARY KEY)
password	String

NOTE: All services mentioned from heron require logging in as a prerequisite. Otherwise this **UNAUTHORISED** error is thrown. So must login to start the session.



so to access any API starting with `/admin` which is most of them as it was the requirement, you first need to login by sending a POST request to `localhost:XXXX/admin/login`, The request body for the admin login will be by default,

```
{
  "username": "root"
  "password": "root"
}
```

SERVICES OFFERED AT THE VARIOUS CONTROLLERS:

1) Category

1)	Adding a category	POST	/admin/category
2)	Updating a category	PUT	/admin/category
3)	Deleting a category	DELETE	/admin/category/{category_id}
4)	Get all categories	GET	/admin/categories
5)	Get category by ID	GET	/admin/category/{category_id}
6)	Get category by Name	GET	/admin/category/name/{category_name}

- POST /admin/category => Adding a category

This endpoint is used to add or create a category by the admin. Only the category name needs to be entered, the category ID is auto generated using the TableGenerator with initial value as 4. Returns the added category.

The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/admin/category
- Body Type:** JSON
- Request Body:**

```
1 {  
2   "name": "golf shoes"  
3 }
```
- Response Body:**

```
1 {  
2   "categoryid": 5,  
3   "name": "golf shoes"  
4 }
```
- Status:** 200 OK
- Time:** 113 ms

- **PUT /admin/category => Updating an existing category**

This endpoint is used to update a category by the admin. The category ID of the one you want to update and the new category name desired needs to be entered. Returns the updated category.

The screenshot displays a REST client interface. At the top, the method is set to 'PUT' and the URL is 'http://localhost:8080/admin/category'. Below this, several tabs are visible: 'Params', 'Authorization', 'Headers (9)', 'Body' (which is selected and highlighted with an orange underline), 'Pre-request Script', 'Tests', and 'Settings'. Under the 'Body' tab, there are radio buttons for different body types: 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected and highlighted with an orange circle), 'binary', and 'GraphQL'. To the right of these is a dropdown menu set to 'JSON'. The main area shows a JSON body with the following content:

```
1 {  
2   "categoryid": 5,  
3   "name": "motorsport shoes"  
4 }
```

Below the body editor, there are tabs for 'Body' (selected), 'Cookies (1)', 'Headers (5)', and 'Test Results'. To the right of these tabs, the status is shown as '200 OK' and the time as '39 ms'. At the bottom, there are tabs for 'Pretty' (selected), 'Raw', 'Preview', and 'Visualize'. To the right of these is a dropdown menu set to 'JSON' and a button with a red icon. The main area shows the same JSON body as above, formatted in a 'Pretty' view:

```
1 {  
2   "categoryid": 5,  
3   "name": "motorsport shoes"  
4 }
```


- **DELETE /admin/category/{category_id} => Deleting an existing category**

This endpoint is used to delete a category by the admin. The category ID of the one you want to delete needs to be entered as a Path Variable in the URL. Returns void if successfully deleted.

- Successful deletion of category with ID = 5

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://localhost:8080/admin/category/5
- Body:** The 'Body' tab is selected, showing a single line with the number '1'.
- Response:** The 'Body' tab is selected, showing a single line with the number '1'.
- Status:** 200 OK

- Error thrown as category to be deleted doesn't exist.

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://localhost:8080/admin/category/10
- Body:** The 'Body' tab is selected, showing a single line with the number '1'.
- Response:** The 'Body' tab is selected, showing a JSON response with the following structure:

```
1 {  
2   "timestamp": "2020-10-04T18:29:18.257+0000",  
3   "status": 400,  
4   "error": "Bad Request",  
5   "message": "The Category you want to delete doesn't exist",  
6   "path": "/admin/category/10"  
7 }
```
- Status:** 400 Bad Request
- Time:** 29 ms

- **GET /admin/categories => Get All Categories**

This endpoint is used to retrieve all categories by the admin. No input is required from the user. Returns a List of all the categories.

GET http://localhost:8080/admin/categories

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 16 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "categoryid": 1,
4     "name": "running shoes"
5   },
6   {
7     "categoryid": 2,
8     "name": "tennis shoes"
9   },
10  {
11    "categoryid": 3,
12    "name": "basketball shoes"
13  }
14 ]
```

- **GET /admin/category/{category_id} => Get category by ID**

This endpoint is used to get/retrieve a category using the category ID by the admin. The category ID of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the Category having the ID as entered, otherwise Error 404.

GET http://localhost:8080/admin/category/3

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "categoryid": 3,
3   "name": "basketball shoes"
4 }
```

- **GET /admin/category/name/{category_name} => Get category by Name**

This endpoint is used to get/retrieve a category (list) using the category name by the admin. The category name of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the Category having the name as entered, otherwise Error 404.

- A 200 OK as category with the name “tennis shoes” was found and returned.

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/admin/category/name/tennis shoes`. The response status is 200 OK and the time taken is 23 ms. The response body is displayed in JSON format, showing a single category object with `categoryid: 2` and `name: "tennis shoes"`.

```
1 {
2   {
3     "categoryid": 2,
4     "name": "tennis shoes"
5   }
6 }
```

- Error thrown as there is no category with the name “walking shoes”.

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/admin/category/name/walking shoes`. The response status is 404 Not Found and the time taken is 14 ms. The response body is displayed in JSON format, showing an error object with `timestamp`, `status: 404`, `error: "Not Found"`, `message: "No category found with the given name"`, and `path: "/admin/category/name/walking%20shoes"`.

```
1 {
2   "timestamp": "2020-10-04T18:31:28.761+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No category found with the given name",
6   "path": "/admin/category/name/walking%20shoes"
7 }
```

2) Product

1)	Adding a product	POST	/admin/product/{category_id}
2)	Updating a product	PUT	/admin/product/{category_id}
3)	Deleting a product	DELETE	/admin/product/{product_id}
4)	Get all products	GET	/admin/products
5)	Get product by ID	GET	/admin/product/{product_id}
6)	Get products by name	GET	/admin/product/name/{product_name}
7)	Get products by brand	GET	/admin/product/brand/{product_brand}
8)	Get products by category	GET	/admin/product/category/{category_id}

- POST /admin/product/{category_id}

This endpoint is used to add or create a product by the admin. Product details such as name, brand, price needs to be entered in the request body, the productId is auto generated using the TableGenerator, and lastly the category ID to which the product belongs needs to be passed as a path variable. Returns the final product added.

- 200 OK for product added successfully

POST http://localhost:8080/admin/product/3

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "name": "Nike Kyrie FlyTrap",
3   "brand": "Nike",
4   "price": 7895
5 }
```

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 24 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "productid": 9,
3   "name": "Nike Kyrie FlyTrap",
4   "brand": "Nike",
5   "price": 7895,
6   "category": {
7     "categoryid": 3,
8     "name": "basketball shoes"
9   }
}
```

- 400 Error in case product is added for a category which doesn't exist. Foreign Key Violation.

The screenshot shows a REST client interface. The top bar indicates a POST request to `http://localhost:8080/admin/product/8`. The 'Body' tab is selected, showing a JSON request body:

```
1 {  
2   "name": "Nike Kyrie FlyTrap",  
3   "brand": "Nike",  
4   "price": 7895  
5 }
```

Below the request, the response is shown with a status of 400 Bad Request and a time of 15 ms. The response body is displayed in the 'Pretty' view:

```
1 {  
2   "timestamp": "2020-10-04T08:51:17.316+0000",  
3   "status": 400,  
4   "error": "Bad Request",  
5   "message": "The category doesn't exist. Violates foreign key relation",  
6   "path": "/admin/product/8"  
7 }
```

- **PUT /admin/product/{category_id}**

This endpoint is used to update a product by the admin. Updated product details such as name, brand, price needs to be entered in the request body, the productId is also needs to be entered, and lastly the (updated) category ID to which the product belongs needs to be passed as a path variable. Returns the final product added.


Both price and category changed for the last added product with productId = 9.


PUT http://localhost:8080/admin/product/2

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```
1 {
2   "productid":9,
3   "name": "Nike Kyrie FlyTrap",
4   "brand": "Nike",
5   "price": 7100
6 }
```

Body Cookies (1) Headers (5) Test Results  Status: 200 OK Time: 13 ms

Pretty Raw Preview Visualize JSON ▼ 

```
1 {
2   "productid": 9,
3   "name": "Nike Kyrie FlyTrap",
4   "brand": "Nike",
5   "price": 7100,
6   "category": {
7     "categoryid": 2,
8     "name": "tennis shoes"
9   }
10 }
```


- **DELETE /admin/product/{product_id} => Deleting a product by ID**


This endpoint is used to delete a product by the admin. The product ID of the one you want to delete needs to be entered as a Path Variable in the URL. Returns void if successfully deleted. Else returns product not found for an Error request to delete.

Successfully deleted product with id = 9

DELETE http://localhost:8080/admin/product/9

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

Body Cookies (1) Headers (4) Test Results  Status: 200 OK Time: 34 ms

Pretty Raw Preview Visualize Text ▼ 

```
1
```

- Product to delete with id = 11 doesn't exist in the database.

DELETE http://localhost:8080/admin/product/11

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 12 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T09:01:36.427+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "The product ID to delete doesn't exist",
6   "path": "/admin/product/11"
7 }
```

- GET /admin/products => Get all products

This endpoint is used to retrieve a list all products by the admin. No input is required from the user. Returns a List of all the products. If there are no products in the database, it returns a error that products list size is 0.

GET http://localhost:8080/admin/products

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 26 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "productid": 1,
4     "name": "Nike Revolution",
5     "brand": "Nike",
6     "price": 3734,
7     "category": {
8       "categoryid": 1,
9       "name": "running shoes"
10    }
11  },
12  {
13    "productid": 2,
14    "name": "Adidas Aswee run",
15    "brand": "Adidas",
16    "price": 4200,
17    "category": {
18      "categoryid": 1,
19      "name": "running shoes"
20    }
21  },
22  {
23    "productid": 3,
```

- **GET /admin/product/{product_id} => Get product by ID**

This endpoint is used to get/retrieve a product using the product ID by the admin. The product ID of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the product having the ID as entered, otherwise Error 404 Product not found error.

- Successfully retrieved product with id = 3

GET http://localhost:8080/admin/product/3

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 28 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "productid": 3,
3   "name": "Sketchers Nordic",
4   "brand": "Sketchers",
5   "price": 2800,
6   "category": {
7     "categoryid": 1,
8     "name": "running shoes"
9   }
10 }
```

- Error as product requested with id = 76 doesn't exist.

GET http://localhost:8080/admin/product/76

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 9 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T09:08:13.892+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "The product with the ID requested doesn't exist",
6   "path": "/admin/product/76"
7 }
```


- **GET /admin/product/name/{product_name} => Get product by Name**

This endpoint is used to get/retrieve a product (list) using the product name by the admin. The product name of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the (list of) products having the name as entered, otherwise Error 404 that no products found with the given name.

- Successfully retrieved product with the name "Adidas Asweerun"

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/admin/product/name/Adidas Asweerun`. The response status is **200 OK** with a time of **11 ms**. The response body is displayed in JSON format, showing a single product object:

```
1 {
2   {
3     "productid": 2,
4     "name": "Adidas Asweerun",
5     "brand": "Adidas",
6     "price": 4200,
7     "category": {
8       "categoryid": 1,
9       "name": "running shoes"
10    }
11  }
12 }
```

- Error finding a product with name "Nike Zoom" as product doesn't exist

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/admin/product/name/Nike Zoom`. The response status is **404 Not Found** with a time of **9 ms**. The response body is displayed in JSON format, showing an error object:

```
1 {
2   "timestamp": "2020-10-04T09:13:52.561+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No products found with the given name",
6   "path": "/admin/product/name/Nike%20Zoom"
7 }
```

- **GET /admin/product/brand/{product_brand} => Get product by Brand**

This endpoint is used to get/retrieve a product (list) using the product brand by the admin. The product brand of the ones you want to retrieve needs to be entered as a Path Variable in the URL. Returns the (list of) products having the brand as entered, otherwise Error 404 that no products found with the given brand.

- List of products returned with brand = "Nike"

GET http://localhost:8080/admin/product/brand/Nike

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 33 ms

Pretty Raw Preview Visualize JSON

```
2 {
3   "productid": 1,
4   "name": "Nike Revolution",
5   "brand": "Nike",
6   "price": 3734,
7   "category": {
8     "categoryid": 1,
9     "name": "running shoes"
10  },
11 },
12 {
13   "productid": 5,
14   "name": "Nike Court Lite",
15   "brand": "Nike",
16   "price": 4200,
17   "category": {
18     "categoryid": 2,
19     "name": "tennis shoes"
20  },
21 },
22 {
23   "productid": 6,
24   "name": "Nike Air Jordan 11",
25   "brand": "Nike"
```

- No products found with brand = "Puma"

GET http://localhost:8080/admin/product/brand/Puma

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 28 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T09:20:24.593+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No products found with the given brand",
6   "path": "/admin/product/brand/Puma"
7 }
```

- **GET /admin/product/category/{category_id} => Get product by Category**

This endpoint is used to get/retrieve a product (list) using the Category ID by the admin. The product category of the ones you want to retrieve needs to be entered as a Path Variable in the URL. Returns the (list of) products having the category ID as entered, otherwise Error 404 that no products found with the given category ID.

- List of products with category id = 3 belonging to "basketball shoes"

GET http://localhost:8080/admin/product/category/3

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 15 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "productid": 6,
4     "name": "Nike Air Jordan 11",
5     "brand": "Nike",
6     "price": 9000,
7     "category": {
8       "categoryid": 3,
9       "name": "basketball shoes"
10    }
11  },
12  {
13    "productid": 7,
14    "name": "Asics Jump",
15    "brand": "Asics",
16    "price": 6700,
17    "category": {
18      "categoryid": 3,
19      "name": "basketball shoes"
20    }
21  }
22 ]
```

- Error that no products found for the given category ID.

GET http://localhost:8080/admin/product/category/5

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 10 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T09:24:21.157+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No products found with the given category",
6   "path": "/admin/product/category/5"
7 }
```

3) User

1)	Adding a user	POST	/admin/user
2)	Updating a user	PUT	/admin/user
3)	Deleting a user	DELETE	/admin/user/{user_id}
4)	Get all users	GET	/admin/users
5)	Get user by ID	GET	/admin/user/{user_id}
6)	Get users by FirstName	GET	/admin/users/firstname/{firstname}
7)	Get users by LastName	GET	/admin/users/lastname/{lastname}
8)	Get users by Age	GET	/admin/users/age/{age}
9)	Get user by EmailID	GET	/admin/users/emailid/{emailid}

- POST /admin/user => Adding a user to the system

This endpoint is used to add or create a user by the admin. User details such as firstName, LastName, Age, Email ID, Password need to be entered in the request body, the Userid is auto generated using the TableGenerator. Returns the user added.

The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/admin/user
- Body:** A JSON object containing user details:

```
{  "firstname": "Apurva",  "lastname": "Bhardwaj",  "age": 23,  "emailid": "apurva98@gmail.com",  "password": "apurva98"}
```
- Status:** 200 OK
- Time:** 79 ms
- Response Body:** A JSON object containing the created user details:

```
{  "userid": 5,  "firstname": "Apurva",  "lastname": "Bhardwaj",  "age": 23,  "emailid": "apurva98@gmail.com",  "password": "apurva98"}
```

- **PUT /admin/user => Updating a user of the system**

This endpoint is used to update a user by the admin. Updated product details such as firstname, lastname, age, email ID, password needs to be entered in the request body, the userId also needs to be entered. Returns the final updated user

- Both Age and email ID are updated for the user added with the add user request.

The screenshot displays a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:8080/admin/user
- Body Type:** raw (selected)
- Body Content (JSON):**

```
1 {  
2   "userid": 5,  
3   "firstname": "Apurva",  
4   "lastname": "Bhardwaj",  
5   "age": 25,  
6   "emailid": "bhardwaj98@gmail.com",  
7   "password": "apurva98"  
8 }
```
- Response Status:** 200 OK
- Response Time:** 32 ms
- Response Body (Pretty):**

```
1 {  
2   "userid": 5,  
3   "firstname": "Apurva",  
4   "lastname": "Bhardwaj",  
5   "age": 25,  
6   "emailid": "bhardwaj98@gmail.com",  
7   "password": "apurva98"  
8 }
```

- **DELETE /admin/user/{user_id} => Updating a user of the system**


This endpoint is used to delete a user by the admin. The user ID of the one you want to delete needs to be entered as a Path Variable in the URL. Returns void if successfully deleted. Else returns user not found as an Error for the request to delete.


- Successfully deleted user with id = 5

Untitled Request

DELETE http://localhost:8080/admin/user/5

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (4) Test Results  Status: 200 OK Time: 17 ms


Pretty Raw Preview Visualize Text 


1

- 404 error user not found when trying to delete a user which doesn't exist.

DELETE http://localhost:8080/admin/user/10

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results  Status: 404 Not Found Time: 19 ms

Pretty Raw Preview Visualize JSON 

```
1 {  
2   "timestamp": "2020-10-04T11:42:19.373+0000",  
3   "status": 404,  
4   "error": "Not Found",  
5   "message": "No user found for the given ID",  
6   "path": "/admin/user/10"  
7 }
```

- **GET /admin/users => Get all users**

This endpoint is used to retrieve a list all users by the admin. No input is required from the user. Returns a List of all the users. If there are no users registered in the database, it returns an error that user list size is 0.

- The list of all users registered in the system.

GET http://localhost:8080/admin/users

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 17 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "userid": 1,
4     "firstname": "Siddharth",
5     "lastname": "Basu",
6     "age": 22,
7     "emailid": "siddharthbasu98@gmail.com",
8     "password": "siddharth98"
9   },
10  {
11    "userid": 2,
12    "firstname": "Saphal",
13    "lastname": "Patro",
14    "age": 23,
15    "emailid": "saphal98@gmail.com",
16    "password": "saphal98"
17  },
18  {
19    "userid": 3,
20    "firstname": "Vaibhav",
21    "lastname": "Harit",
22    "age": 23,
23    "emailid": "vharit98@gmail.com",
```

- **GET /admin/user/{user_id} => Get user by ID**

This endpoint is used to get/retrieve a user using the user ID by the admin. The user ID of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the user having the ID as entered, otherwise Error 404 user not found error.

- Successfully retrieved user with id = 1

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/admin/user/1
- Response Status:** 200 OK
- Response Time:** 15 ms
- Response Body (JSON):**

```
{  "userid": 1,  "firstname": "Siddharth",  "lastname": "Basu",  "age": 22,  "emailid": "siddharthbasu98@gmail.com",  "password": "siddharth98"}
```

- No user found with id = 23 hence error.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/admin/user/23
- Response Status:** 404 Not Found
- Response Time:** 14 ms
- Response Body (JSON):**

```
{  "timestamp": "2020-10-04T11:48:15.821+0000",  "status": 404,  "error": "Not Found",  "message": "No user found for the given ID",  "path": "/admin/user/23"}
```


- **GET /admin/users/firstname/{firstname} => Get users by First Name**

This endpoint is used to get/retrieve a user (list) using the user's first name by the admin. The user's first name of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the (list of) users having the first name as entered, otherwise Error 404 that no users found with the given first name.

- Successfully retrieved user with first name = "Siddharth"

GET http://localhost:8080/admin/users/firstname/Siddharth

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 14 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "userid": 1,
4     "firstname": "Siddharth",
5     "lastname": "Basu",
6     "age": 22,
7     "emailid": "siddharthbasu98@gmail.com",
8     "password": "siddharth98"
9   }
10 ]
```

- Error retrieving user with first name = "John"

GET http://localhost:8080/admin/users/firstname/John

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 14 ms S

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T11:55:56.110+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "There are currently no users with the requested first name in the system",
6   "path": "/admin/users/firstname/John"
7 }
```

- **GET /admin/users/lastname/{lastname} => Get users by last Name**

This endpoint is used to get/retrieve a user (list) using the user's last name by the admin. The user's last name of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the (list of) users having the last name as entered, otherwise Error 404 that no users found with the given last name.

- Successfully retrieved user with last name = "Basu"

GET http://localhost:8080/admin/users/lastname/Basu

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 16 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "userid": 1,
4     "firstname": "Siddharth",
5     "lastname": "Basu",
6     "age": 22,
7     "emailid": "siddharthbasu98@gmail.com",
8     "password": "siddharth98"
9   }
10 ]
```

- Error retrieving user with last name = "Gregory"

GET http://localhost:8080/admin/users/lastname/Gregory

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 16 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T11:58:51.781+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "There are currently no users with the requested last name in the system",
6   "path": "/admin/users/lastname/Gregory"
7 }
```

- **GET /admin/users/age/{age} => Get users by age**

This endpoint is used to get/retrieve a user (list) using the user's age by the admin. The user's age of the ones you want to retrieve needs to be entered as a Path Variable in the URL. Returns the (list of) users having the age as entered, otherwise Error 404 that no users found with the given age.

- Successfully retrieved user with age = "23"

GET http://localhost:8080/admin/users/age/23

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 10 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "userid": 2,
4     "firstname": "Saphal",
5     "lastname": "Patro",
6     "age": 23,
7     "emailid": "saphal98@gmail.com",
8     "password": "saphal98"
9   },
10  {
11    "userid": 3,
12    "firstname": "Vaibhav",
13    "lastname": "Harit",
14    "age": 23,
15    "emailid": "vharit98@gmail.com",
16    "password": "vaibhav98"
17  }
18 ]
```

- Error as no users found with age = 29

GET http://localhost:8080/admin/users/age/29

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 18 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T12:03:51.386+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "There are currently no users with the requested age in the system",
6   "path": "/admin/users/age/29"
7 }
```

- **GET /admin/users/emailid/{emailid} => Get users by email ID**

This endpoint is used to get/retrieve a user using the user's email id by the admin. The user's email of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the user having the email ID as entered, otherwise Error 404 that no users found with the given email ID.

- Successfully retrieved user with emailID = "siddharthbasu98@gmail.com"

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/admin/users/emailid/siddharthbasu98@gmail.com`. The response status is 200 OK and the time taken is 88 ms. The response body is displayed in JSON format, showing the details of the user with the specified email ID.

```
{
  "userid": 1,
  "firstname": "Siddharth",
  "lastname": "Basu",
  "age": 22,
  "emailid": "siddharthbasu98@gmail.com",
  "password": "siddharth98"
}
```

- Error retrieving user with emailID as something which is not registered.

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/admin/users/emailid/aparna66@gmail.com`. The response status is 404 Not Found and the time taken is 96 ms. The response body is displayed in JSON format, indicating that the user was not found and providing a message and the requested path.

```
{
  "timestamp": "2020-10-04T12:09:57.894+0000",
  "status": 404,
  "error": "Not Found",
  "message": "There are currently no users with the requested emailID in the system",
  "path": "/admin/users/emailid/aparna66@gmail.com"
}
```

4) Purchase

1)	Adding a purchase	POST	/admin/purchase/{user_id}/{product_id}
2)	Updating a purchase	PUT	/admin/purchase/{user_id}/{product_id}
3)	Deleting a purchase	DELETE	/admin/purchase/{purchase_id}
4)	Get all purchases	GET	/admin/purchases
5)	Get purchase by purchase ID	GET	/admin/purchase/{purchase_id}
6)	Get purchases by Date	GET	/admin/purchase/date/{date}
7)	Get purchase by user ID	GET	/admin/purchase/user/{user_id}
8)	Get purchase by product ID	GET	/admin/purchase/product/{product_id}

- **POST /admin/purchase/{user_id}/{product_id} => Adding a purchase**

This endpoint is used to add or create a purchase by the admin. Purchase details such as purchase date, purchase quantity needs to be entered in the request body, the purchase ID is auto generated using the TableGenerator, and lastly the user ID who has purchased the product, and the product ID of the product purchased needs to be passed as a path variable. Returns the final purchase added.

- 200 OK for purchase added successfully

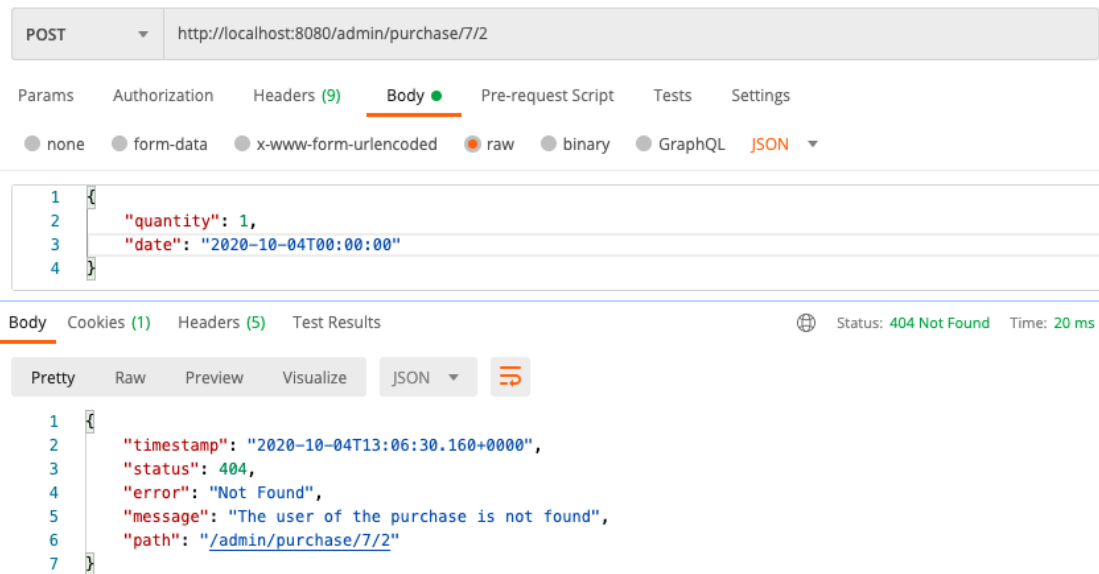
The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/admin/purchase/1/2
- Body Type:** raw (JSON)
- Request Body:**

```
1 {  
2   "quantity": 1,  
3   "date": "2020-10-04T00:00:00"  
4 }
```
- Status:** 200 OK, Time: 155 ms
- Response Body (JSON):**

```
1 {  
2   "purchaseid": 8,  
3   "quantity": 1,  
4   "date": "2020-10-04T00:00:00",  
5   "user": {  
6     "userid": 1,  
7     "firstname": "Siddharth",  
8     "lastname": "Basu",  
9     "age": 22,  
10    "emailid": "siddharthbasu98@gmail.com",  
11    "password": "siddharth98"  
12  },  
13   "product": {  
14     "productid": 2,  
15     "name": "Adidas Asweerun",  
16     "brand": "Adidas",  
17     "price": 4200,  
18     "category": {  
19       "categoryid": 1,  
20       "name": "running shoes"  
21     }  
22   }  
23 }
```

- Error adding a purchase for a user who doesn't exist.



- PUT /admin/purchase/{user_id}/{product_id} => Updating a purchase

This endpoint is used to update a purchase by the admin. Purchase details such as purchase date, purchase quantity needs to be entered in the request body, additionally the purchase ID which needs to be updated has to be added. The user ID who has purchased the product, and the product ID of the product purchased needs to be passed as a path variable. Returns the updated purchase added.

- Updated purchase ID 8 which was added in the previous request. Both quantity and the product purchased is updated. Returns 200 OK.

POST

http://localhost:8080/admin/purchase/1/3

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON ▼

```
1 {
2   "purchaseid": 8,
3   "quantity": 2,
4   "date": "2020-10-04T00:00:00"
5 }
```

Body

Cookies (1)

Headers (5)

Test Results

🌐

Status: 200 OK

Time: 54 ms

Pretty

Raw

Preview

Visualize

JSON ▼

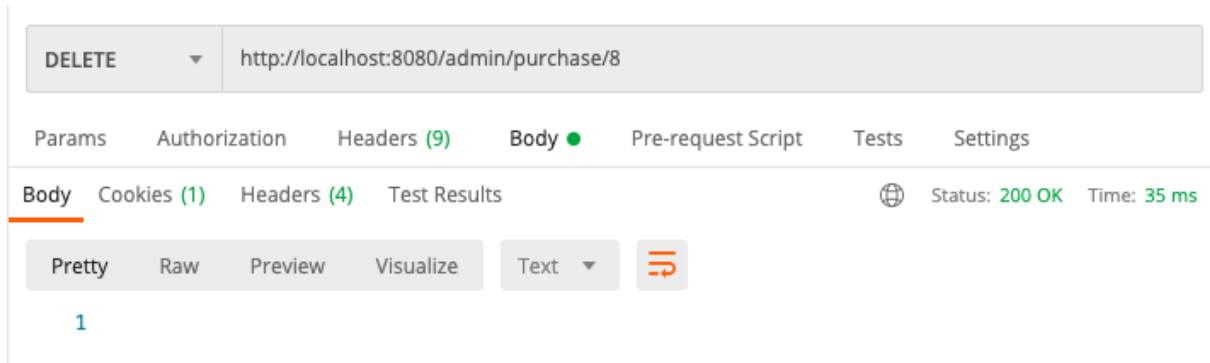
🔍

```
1 {
2   "purchaseid": 8,
3   "quantity": 2,
4   "date": "2020-10-04T00:00:00",
5   "user": {
6     "userid": 1,
7     "firstname": "Siddharth",
8     "lastname": "Basu",
9     "age": 22,
10    "emailid": "siddharthbasu98@gmail.com",
11    "password": "siddharth98"
12  },
13  "product": {
14    "productid": 3,
15    "name": "Sketchers Nordic",
16    "brand": "Sketchers",
17    "price": 2800,
18    "category": {
19      "categoryid": 1,
20      "name": "running shoes"
21    }
22  }
23 }
```

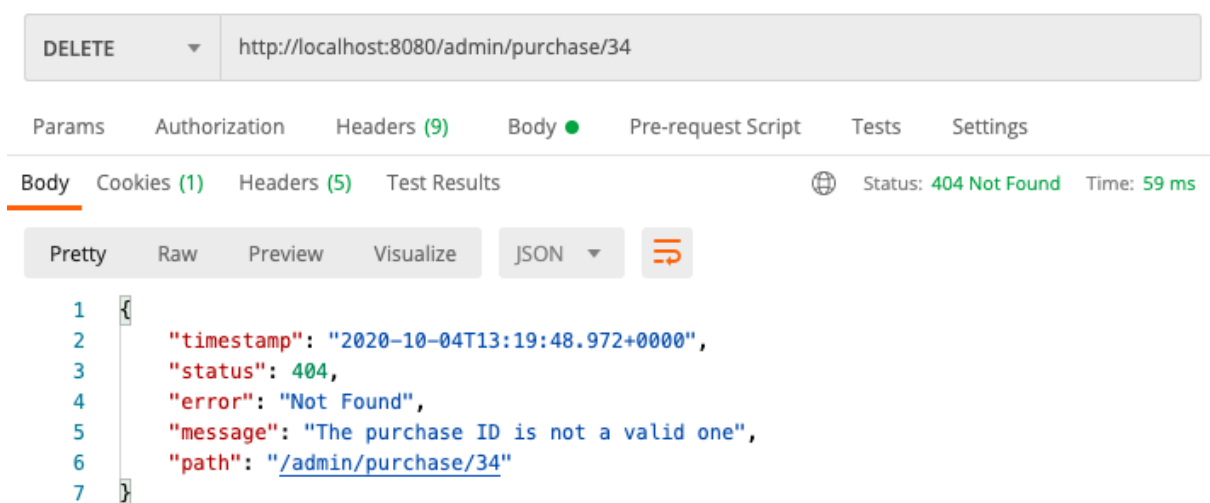

- **DELETE /admin/purchase/{purchase_id} => Deleting a purchase by ID**

This endpoint is used to delete a purchase by the admin. The purchase ID of the one you want to delete needs to be entered as a Path Variable in the URL. Returns void if successfully deleted. Else returns purchase not found for an Error request to delete.

- 200 OK successfully deleted purchase with ID = 8



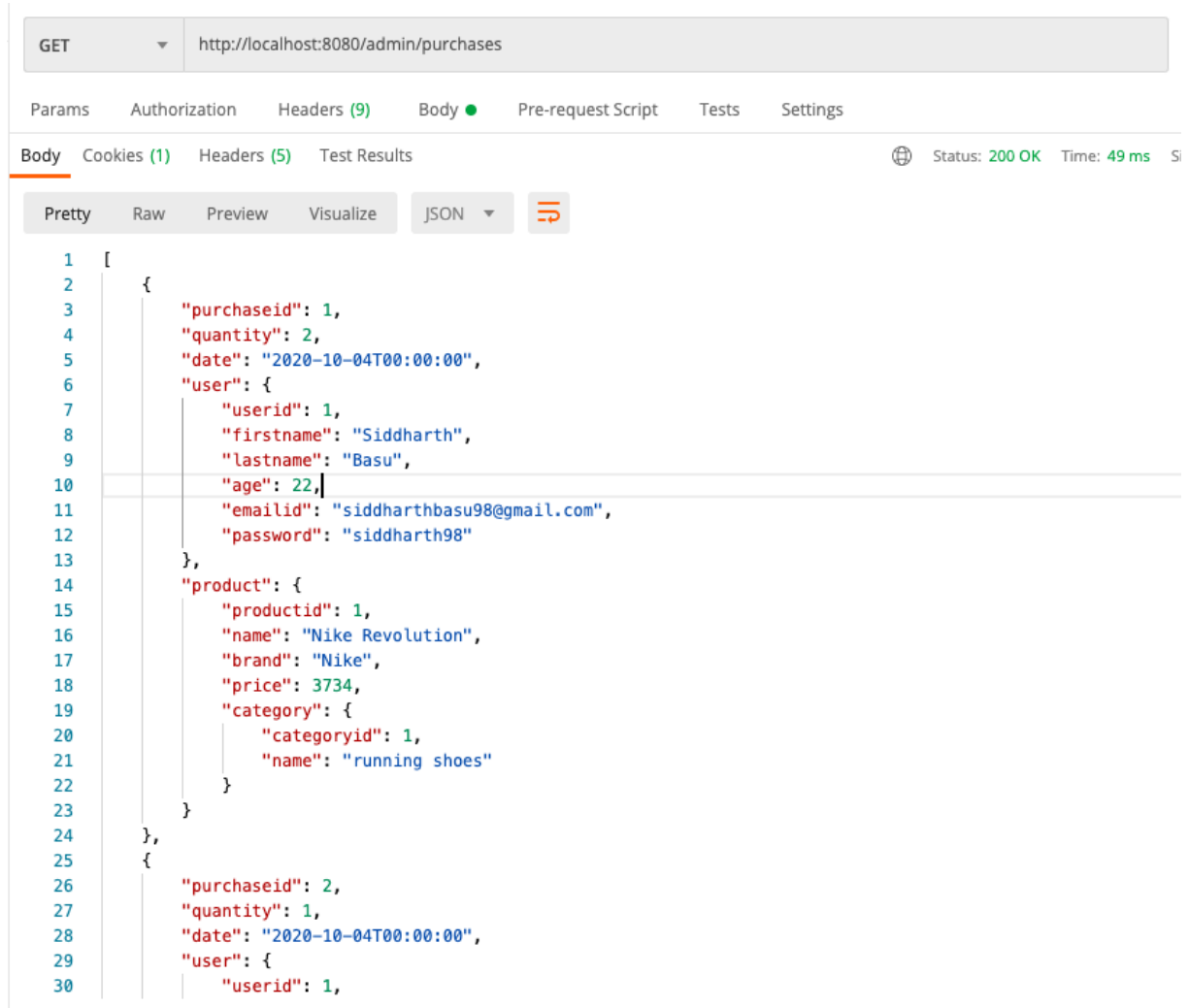
- Error with deleting an invalid purchase



- **GET /admin/purchases => Get all purchases**

This endpoint is used to retrieve a list of all purchases. No input is required from the user. Returns a List of all the purchases. If there are no purchases registered in the database, it returns an error that user list size is 0.

- The list of all purchases registered in the system.



```
1  [
2    {
3      "purchaseid": 1,
4      "quantity": 2,
5      "date": "2020-10-04T00:00:00",
6      "user": {
7        "userid": 1,
8        "firstname": "Siddharth",
9        "lastname": "Basu",
10       "age": 22,
11       "emailid": "siddharthbasu98@gmail.com",
12       "password": "siddharth98"
13     },
14     "product": {
15       "productid": 1,
16       "name": "Nike Revolution",
17       "brand": "Nike",
18       "price": 3734,
19       "category": {
20         "categoryid": 1,
21         "name": "running shoes"
22       }
23     }
24   },
25   {
26     "purchaseid": 2,
27     "quantity": 1,
28     "date": "2020-10-04T00:00:00",
29     "user": {
30       "userid": 1,
```

- Fetches the user and product details as well as per the Relationships defined.

- GET /admin/purchase/{purchase_id} => Get purchase by purchase ID

This endpoint is used to get/retrieve a purchase using the purchase ID by the admin. The purchase ID of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the purchase having the ID as entered, otherwise Error 404 purchase not found error.

- Successfully retrieved purchase with id = 1

GET http://localhost:8080/admin/purchase/2

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 31 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "purchaseid": 2,
3   "quantity": 1,
4   "date": "2020-10-04T00:00:00",
5   "user": {
6     "userid": 1,
7     "firstname": "Siddharth",
8     "lastname": "Basu",
9     "age": 22,
10    "emailid": "siddharthbasu98@gmail.com",
11    "password": "siddharth98"
12  },
13  "product": {
14    "productid": 6,
15    "name": "Nike Air Jordan 11",
16    "brand": "Nike",
17    "price": 9000,
18    "category": {
19      "categoryid": 3,
20      "name": "basketball shoes"
21    }
22  }
23 }
```

- Error retrieving purchase with an invalid ID

GET http://localhost:8080/admin/purchase/52

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 11 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T13:35:45.530+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "The purchase with the requested ID is not found",
6   "path": "/admin/purchase/52"
7 }
```

- **GET /admin/purchase/date/{date} => Get list of purchases done on a date**

This endpoint is used to get/retrieve a list of purchases done on a particular date. The date in strict **YYYY:MM:DD** format of the ones you want to retrieve needs to be entered as a Path

Variable in the URL. Returns the list of purchases having the date of purchase as entered, otherwise Error 404 purchase list not found error. If invalid date, appropriate error is thrown.

- 200 OK for list of purchases done on 2020-10-05 (5th October, 2020)

GET http://localhost:8080/admin/purchase/date/2020-10-05

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 50 ms Si

Pretty Raw Preview Visualize JSON

```
17     "brand": "Nike",
18     "price": 3734,
19     "category": {
20       "categoryid": 1,
21       "name": "running shoes"
22     }
23   },
24 },
25 {
26   "purchaseid": 2,
27   "quantity": 1,
28   "date": "2020-10-05T00:00:00",
29   "user": {
30     "userid": 1,
31     "firstname": "Siddharth",
32     "lastname": "Basu",
33     "age": 22,
34     "emailid": "siddharthbasu98@gmail.com",
35     "password": "siddharth98"
36   },
37   "product": {
38     "productid": 6,
39     "name": "Nike Air Jordan 11",
40     "brand": "Nike",
41     "price": 9000,
42     "category": {
43       "categoryid": 3,
44       "name": "basketball shoes"
45     }
46   }
47 }
```

- Error not a valid date entered as it mentions 13th month (invalid)

GET http://localhost:8080/admin/purchase/date/2019-13-05

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (4) Test Results Status: 400 Bad Request Time: 49 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-05T08:06:38.504+0000",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "Enter the date in a valid YYYY:MM:DD format",
6   "path": "/admin/purchase/date/2019-13-05"
7 }
```

- Error for no purchases found on 2019-09-05 (Empty list)

```
GET http://localhost:8080/admin/purchase/date/2019-09-05

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings
Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 57 ms
Pretty Raw Preview Visualize JSON {
1  {
2    "timestamp": "2020-10-05T08:06:59.947+0000",
3    "status": 404,
4    "error": "Not Found",
5    "message": "No purchases found for the given date",
6    "path": "/admin/purchase/date/2019-09-05"
7  }
```

- **GET /admin/purchase/user/{user_id} => Get list of purchases done by the user**

This endpoint is used to get/retrieve a list of purchases done by a user using the user ID by the admin. The user ID of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the list of purchases having the user ID as entered, otherwise Error 404 purchase list not found error.

- Successfully retrieved purchase list for user with id = 2

GET http://localhost:8080/admin/purchase/user/1

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 27 ms

Pretty Raw Preview Visualize JSON ↕

```
1 {
2   "purchaseid": 1,
3   "quantity": 2,
4   "date": "2020-10-04T00:00:00",
5   "user": {
6     "userid": 1,
7     "firstname": "Siddharth",
8     "lastname": "Basu",
9     "age": 22,
10    "emailid": "siddharthbasu98@gmail.com",
11    "password": "siddharth98"
12  },
13  "product": {
14    "productid": 1,
15    "name": "Nike Revolution",
16    "brand": "Nike",
17    "price": 3734,
18    "category": {
19      "categoryid": 1,
20      "name": "running shoes"
21    }
22  }
23 },
24 {
25   "purchaseid": 2,
26   "quantity": 1,
27   "date": "2020-10-04T00:00:00",
28   "user": {
29     "userid": 1,
30     "firstname": "Siddharth",
31     "password": "siddharth98"
32   }
33 }
```

- Error retrieving purchase for an invalid user ID

GET http://localhost:8080/admin/purchase/user/6

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 9 ms

Pretty Raw Preview Visualize JSON ↕

```
1 {
2   "timestamp": "2020-10-04T13:41:11.446+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No purchases found for the given User",
6   "path": "/admin/purchase/user/6"
7 }
```

- **GET /admin/purchase/product/{product_id} => Get list of purchases done for a given product**

This endpoint is used to get/retrieve a list of purchases done for a product using the product ID by the admin. The product ID of the one you want to retrieve needs to be entered as a Path Variable in the URL. Returns the list of purchases having the product ID as entered, otherwise Error 404 purchase list not found error.

- Successfully retrieved purchase list for product with id = 2

GET http://localhost:8080/admin/purchase/product/4

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 21 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "purchaseid": 6,
4     "quantity": 1,
5     "date": "2020-10-04T00:00:00",
6     "user": {
7       "userid": 3,
8       "firstname": "Vaibhav",
9       "lastname": "Harit",
10      "age": 23,
11      "emailid": "vharit98@gmail.com",
12      "password": "vaibhav98"
13    },
14    "product": {
15      "productid": 4,
16      "name": "Adidas Court Adapt",
17      "brand": "Adidas",
18      "price": 3000,
19      "category": {
20        "categoryid": 2,
21        "name": "tennis shoes"
22      }
23    }
24  }
25 ]
```

- Error retrieving purchases for an invalid product

GET http://localhost:8080/admin/purchase/product/78

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 12 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T13:44:26.738+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "No purchases found for the given Product",
6   "path": "/admin/purchase/product/78"
7 }
```

5) Admin

1)	Login	POST	/admin/login
2)	Logout	GET	/admin/logout
3)	Change Password	POST	/admin/changePassword
4)	Add/Create admin	POST	/admin
5)	Delete Admin	DELETE	/admin/{username}
6)	Get all admins	GET	/admins
7)	Get admin by username	GET	/admin/name/{username}

- POST admin/login => Login for the admin using credentials

This endpoint is used to login the admin using credentials in the POST request providing the “username” and “password” in the request body. If the credentials provided are correct, then it returns the admin object. If the username is incorrect, it returns error as invalid username, if password is incorrect, it returns error as invalid password.

The login created an admin session with the session attribute as the username of the admin. All functions for the admin need to have the session attribute set to the username.

The screenshot displays a REST client interface for a POST request to `http://localhost:8080/admin/login`. The request body is a JSON object: `{ "username": "root", "password": "root" }`. The response status is `200 OK` with a time of `10 ms`. The response body is also a JSON object: `{ "username": "root", "password": "root" }`.

```
POST http://localhost:8080/admin/login

Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 {
2   "username": "root",
3   "password": "root"
4 }
```

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 10 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "username": "root",
3   "password": "root"
4 }
```


- Error : Incorrect username returns admin not found error.

POST http://localhost:8080/admin/login

Params Authorization Headers (9) Body Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▾

```
1 {
2   "username": "rt",
3   "password": "root"
4 }
```

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 10 ms

Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "timestamp": "2020-10-04T15:14:55.265+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "Enter valid admin username to get existing admin. Admin not found",
6   "path": "/admin/login"
7 }
```

- Incorrect password returns incorrect password entered if the username is valid but the password associated with the username is incorrect.

POST http://localhost:8080/admin/login

Params Authorization Headers (9) Body Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▾

```
1 {
2   "username": "root",
3   "password": "rt"
4 }
```

Body Cookies (1) Headers (4) Test Results Status: 400 Bad Request Time: 18 ms

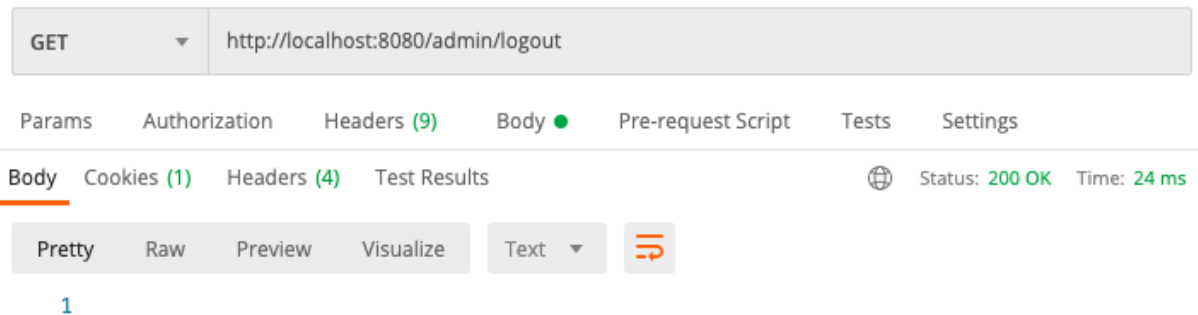
Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "timestamp": "2020-10-04T15:15:20.549+0000",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "Incorrect password entered. Please try again",
6   "path": "/admin/login"
7 }
```

GET admin/logout => Logout and invalidate the admin session

This endpoint is used to logout the admin from the currency session. It invalidates the session and removes the attribute of the session set to the username of the admin who was logged in.

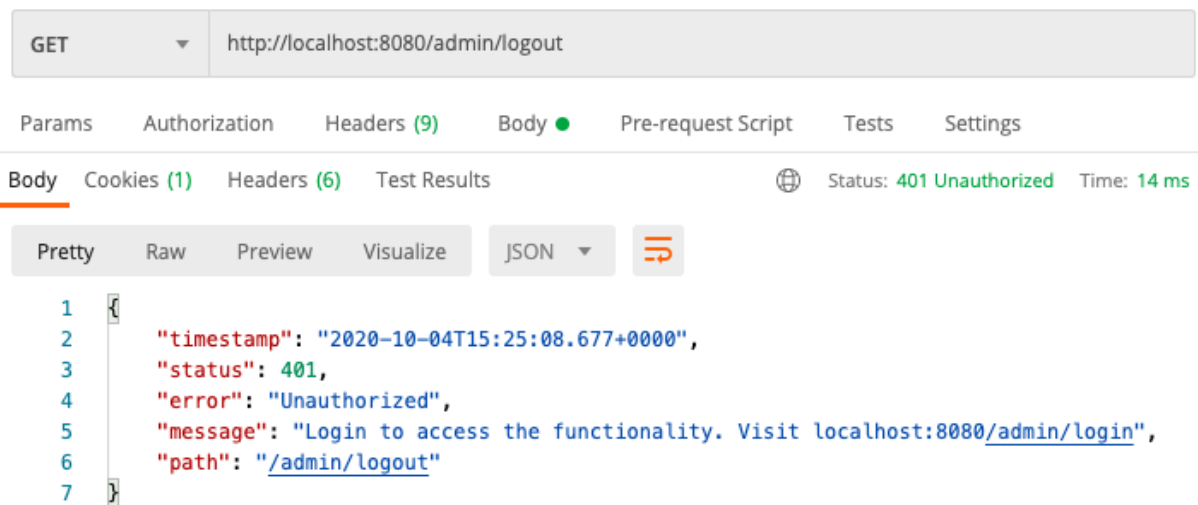
- 200 OK for a successful Logout procedure.



The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8080/admin/logout
- Response Status: 200 OK
- Response Time: 24 ms
- Response Body: 1

- Unauthorised error for firing a logout without logging in first.



The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8080/admin/logout
- Response Status: 401 Unauthorized
- Response Time: 14 ms
- Response Body (JSON):

```
1 {  
2   "timestamp": "2020-10-04T15:25:08.677+0000",  
3   "status": 401,  
4   "error": "Unauthorized",  
5   "message": "Login to access the functionality. Visit localhost:8080/admin/login",  
6   "path": "/admin/logout"  
7 }
```

- **PUT admin/changePassword** => A change password functionality for the currently logged in admin. A PUT request with the “username” and the updated “password” returns the admin with the updated credentials if the change password functionality is successful. Otherwise an error is thrown if admin is changing password for any other admin.
- 200 OK for a successful password change request for the currently logged in admin.

PUT http://localhost:8080/admin/changePassword

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```
1 {
2   "username": "root",
3   "password": "new_root"
4 }
```

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 21 ms

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "username": "root",
3   "password": "new_root"
4 }
```

- Unauthorised error for changing the password of any other admin who doesn't hold the current session.

PUT http://localhost:8080/admin/changePassword

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```
1 {
2   "username": "siddharth",
3   "password": "new_root"
4 }
```

Body Cookies (1) Headers (5) Test Results Status: 401 Unauthorized Time: 8 ms

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "timestamp": "2020-10-04T15:31:50.495+0000",
3   "status": 401,
4   "error": "Unauthorized",
5   "message": "You are unauthorized to change another admin's password",
6   "path": "/admin/changePassword"
7 }
```

- **POST /admin => Adding a admin to the system**

This endpoint is used to add or create an admin by a currently logged in admin. Admin details such as “username” and “password” need to be entered in the request body, Returns the admin added.

POST http://localhost:8080/admin/

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 {  
2   "username": "aparna",  
3   "password": "aparna"  
4 }
```

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 19 ms

Pretty Raw Preview Visualize JSON

```
1 {  
2   "username": "aparna",  
3   "password": "aparna"  
4 }
```

- **DELETE /admin/{username} => Deleting an admin to the system**

This endpoint is used to delete an admin by a currently logged in admin. Admin details such as “username” and “password” need to be entered in the request body, Returns an empty message if successful, otherwise an error exception.

DELETE http://localhost:8080/admin/aparna

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

Body Cookies (1) Headers (4) Test Results Status: 200 OK Time: 30 ms

Pretty Raw Preview Visualize Text

```
1
```

DELETE http://localhost:8080/admin/snbasu

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (4) Test Results Status: 400 Bad Request Time: 11 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T17:15:23.895+0000",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "Enter valid admin username to delete existing admin",
6   "path": "/admin/snbasu"
7 }
```

- **GET /admins => Returns a list of all the registered admins in the system**

This endpoint is used to get a list of all the admins in the system. If the number of admins is zero, an error is returned.

GET http://localhost:8080/admins

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 16 ms

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "username": "root",
4     "password": "new_root"
5   },
6   {
7     "username": "siddharth98",
8     "password": "sid98"
9   }
10 ]
```

- **GET /admin/name/{username} => Get admin by username**

This endpoint returns the admin by the username. The username for which the admin needs to be found out is required to be passed as a path variable and it returns the admin details. If admin doesn't exist for the given username, an error is thrown.

- 200 OK for a successful retrieval of existing admin

GET http://localhost:8080/admin/name/siddharth98

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 19 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "username": "siddharth98",
3   "password": "sid98"
4 }
```

- Invalid Admin username exception for an admin who doesn't exist.

GET http://localhost:8080/admin/name/apurva

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body Cookies (1) Headers (5) Test Results Status: 404 Not Found Time: 10 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-10-04T18:20:29.301+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "Enter valid admin username to get existing admin. Admin not found",
6   "path": "/admin/name/apurva"
7 }
```

- 6) ****Customer** (Not in the problem statement, has only basic non admin functionality, doesn't require login)

1)	Get all products	GET	/products
2)	Get all categories	GET	/categories
3)	Get products by Category	GET	/product/category/{category_id}

- **GET /products => Get all products**

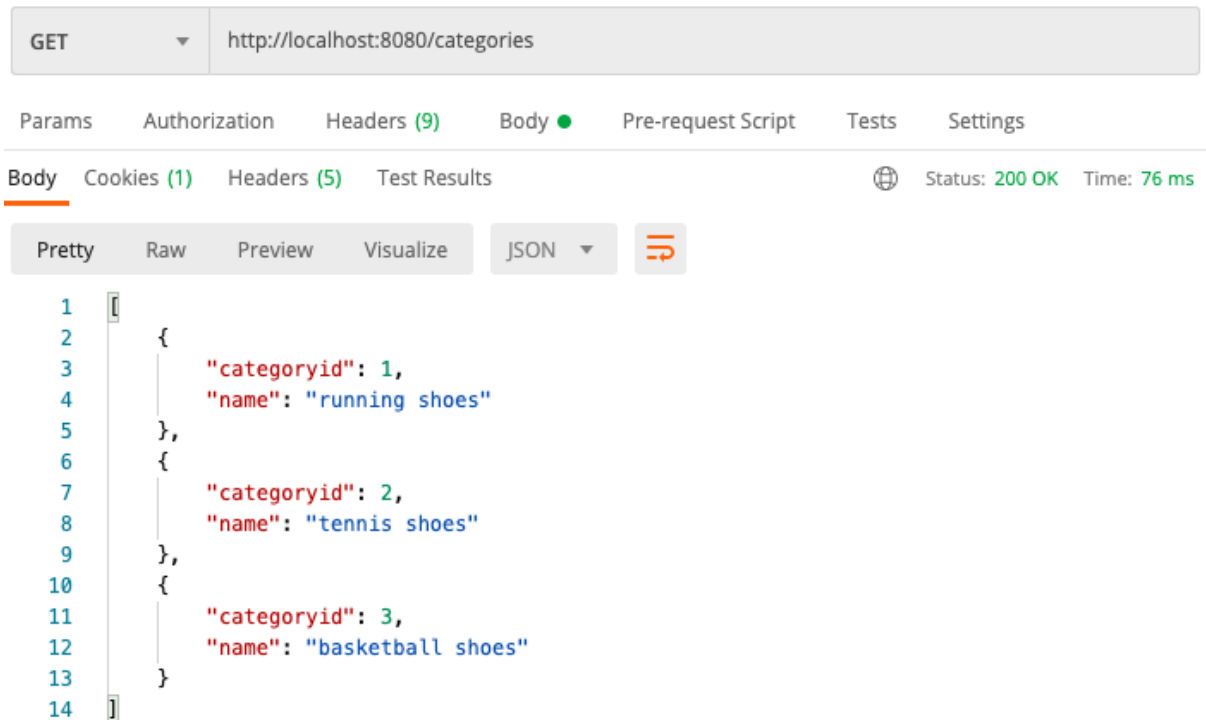
This endpoint is used to retrieve a list of all products. No input is required from the user. Returns a List of all the products. If there are no products in the database, it returns a error that products list size is 0.

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/products`. The response is a JSON array of two product objects, each with a category reference.

```
1  [
2    {
3      "productid": 1,
4      "name": "Nike Revolution",
5      "brand": "Nike",
6      "price": 3734,
7      "category": {
8        "categoryid": 1,
9        "name": "running shoes"
10     }
11   },
12   {
13     "productid": 2,
14     "name": "Adidas Asweerun",
15     "brand": "Adidas",
16     "price": 4200,
17     "category": {
18       "categoryid": 1,
19       "name": "running shoes"
20     }
21   },
22   {
23     "productid": 3,
```

- **GET /categories => Returns a list of all the categories to the user**

This endpoint is used to retrieve all categories by the admin. No input is required from the user. Returns a List of all the categories.



- **GET /product/category/{category_id} => Get product by Category**

This endpoint is used to get/retrieve a product (list) using the Category ID. The product category of the ones you want to retrieve needs to be entered as a Path Variable in the URL. Returns the (list of) products having the category ID as entered, otherwise Error 404 that no products found with the given category ID.

GET

http://localhost:8080/product/category/2

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

Body

Cookies (1)

Headers (5)

Test Results

⌐

Status: 200 OK

Time: 13 ms


Pretty

Raw

Preview

Visualize

JSON ▾



1

[

2

{

3

"productid": 4,

4

"name": "Adidas Court Adapt",

5

"brand": "Adidas",

6

"price": 3000,

7

"category": {

8

"categoryid": 2,

9

"name": "tennis shoes"

10

}

11

},

12

{

13

"productid": 5,

14

"name": "Nike Court Lite",

15

"brand": "Nike",

16

"price": 4200,

17

"category": {

18

"categoryid": 2,

19

"name": "tennis shoes"

20

}

21

}

22

]

