Java FSD Training 2022

Phase - 2 Project: sportyshoes.com

Developer details:

- Name: Aditya Saini
- Phone: +91 9991552342
- Email: aditya.fr@prolim.com

Introduction

This specification document describes the various aspect of the Phase2-project of the course JFSD full stack training. The document contains the following things:

- Project and developer details
- Java concepts used in the project
- Generic features of the developed product
- API implementations of all the features
- Links to the GitHub repository to verify the project completion
- Conclusion on enhancing the application and defining the USPs

Project Description

The task was to develop a prototype version of a website called sportyshoes.com. The functionality of the project can be narrowed down to building the following features:

- Manage the products in the store, including categorizing them
- Browse the list of users who have signed up and be able to search users
- See purchase reports filtered by date and category

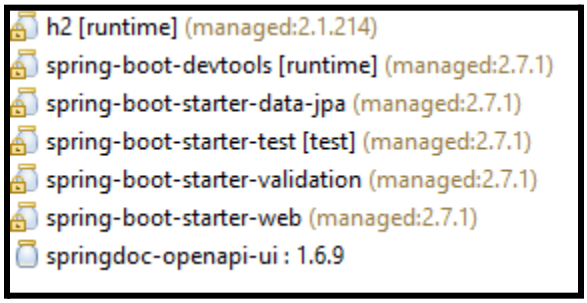
Product features

After a thorough reading of the initial project description, we have developed the following features for the initial prototype version of the website:

1) **The website admin can:**
    a) See all existing administrator configurations
    b) Update their password
2) **With regards to the signed up users, the admin can:**
    a) See all signed up users
    b) Search users filtered by their
        i) Unique id
        ii) Name
        iii) Shoe size
        iv) Mobile Number
        v) Email address
3) **With regards to the products, the admin can:**
    a) See all existing shoes in the inventory
    b) Search shoes filtered by their
        i) Unique id
        ii) Color
        iii) Category
        iv) Price
    c) Manage products by
        i) Adding new shoes
        ii) Update existing shoes
        iii) Deleting existing shoes
            - Note: If any shoe to be deleted exists in any purchase report, the purchase report will also get deleted.
4) **With regards to the purchase reports, the admin can:**
    a) Search purchase reports filtered by purchase date and category
    b) Create new purchase reports

Programming environment

- Language: Java 11
- IDE: Spring tools Suite v4.14.1
- Dependencies(Please refer pom.xml for more details):



- Operating System: Windows
- Version control system: Git
- Project hosting service: Github

Java concepts used in this project

- The OOPs paradigm was used extensively in the project:
    - The repositories were implemented using Java interfaces and extended from **JpaRepository** to ease the process of querying data.
    - The service classes were also divided into interfaces and then inheritance-based implementations.
    - All the variables were made private and were accessible through getters and setters to promote data security.
- The **Spring Boot framework** was used to build this project because of its ease of use and practicality in developing real-world projects.
- The **Java Persistence API(JPA)** coupled with an **in-memory H2 database** was used extensively to ease up the CRUD operations of the project.
    - **On the Model side, JPA annotation(s) like**

        - @Entity and @Table were used to map Java objects to tables in the H2 database.
        - @Id and @Column were used to separate the regular columns from the primary key column.
        - @GeneratedValue was used for autogenerating primary keys.
        - @ManytoOne and @JoinColumn were used to create foreign key-based many-to-one relationships between Java objects.
        - @OnDelete was used to implement the assumption that the parent entity also gets deleted upon the deletion of the child entity.
    - **On the Service and Repository side, JPA annotation(s) like**
        - @Service was used to denote the Service classes.
        - @Autowired was used in the service implementation classes to create repository beans.
    - **On the Controller side, JPA annotation(s) like**
        - @RestController was used to denote the Controller classes.
        - @Autowired was used in the controller classes to create Service beans.
        - @GetMapping, @PutMapping, @PostMapping, and @DeleteMapping were used to implement GET, PUT, POST, and DELETE requests, respectively.
        - @PathVariable annotation was used for accessing the values sent by the admin into the corresponding API.
        - @RequestBody was used in POST and PUT requests for obtaining the corresponding object bodies.
- Exception handling was also done for all four entities to capture edge cases using the **RunTimeException** class.
- The **Swagger-UI** tool was also used to generate easy-to-use reference documentation of all the implemented REST APIs.

Project Implementation

The application was broken down into 4 Java classes, and each class was further abstracted into an H2 database table as given below:

<table>
<tr><th>S no.</th><th>Java class</th><th>Table name</th><th>Description</th><th>Columns</th></tr>
<tr><td>1.</td><td>Admin.java</td><td>admintbl</td><td>Contains details of the website administrators</td><td>
<table>
<tr><th>FIELD</th><th>TYPE</th><th>NULL</th><th>KEY</th><th>DEFAULT</th></tr>
<tr><td>ID</td><td>BIGINT</td><td>NO</td><td>PRI</td><td>NULL</td></tr>
<tr><td>PASSWORD</td><td>CHARACTER VARYING(255)</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>USERNAME</td><td>CHARACTER VARYING(255)</td><td>YES</td><td></td><td>NULL</td></tr>
</table>
</td></tr>
<tr><td>2.</td><td>User.java</td><td>usertbl</td><td>Contains details of all the signed-in users</td><td>
<table>
<tr><th>FIELD</th><th>TYPE</th><th>NULL</th><th>KEY</th><th>DEFAULT</th></tr>
<tr><td>ID</td><td>BIGINT</td><td>NO</td><td>PRI</td><td>NULL</td></tr>
<tr><td>EMAIL</td><td>CHARACTER VARYING(255)</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>MOBILE</td><td>CHARACTER VARYING(255)</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>NAME</td><td>CHARACTER VARYING(255)</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>SIZE</td><td>DOUBLE PRECISION</td><td>YES</td><td></td><td>NULL</td></tr>
</table>
</td></tr>
<tr><td>3.</td><td>Shoe.java</td><td>shoetbl</td><td>Contains details of all the shoes available on the website</td><td>
<table>
<tr><th>FIELD</th><th>TYPE</th><th>NULL</th><th>KEY</th><th>DEFAULT</th></tr>
<tr><td>ID</td><td>BIGINT</td><td>NO</td><td>PRI</td><td>NULL</td></tr>
<tr><td>CATEGORY</td><td>CHARACTER VARYING(255)</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>COLOR</td><td>CHARACTER VARYING(255)</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>PRICE</td><td>DOUBLE PRECISION</td><td>YES</td><td></td><td>NULL</td></tr>
</table>
</td></tr>
<tr><td>4.</td><td>PurchaseReport.java</td><td>purchasetbl</td><td>Contains details of all purchases. Has many-to-one relationships with shoetbl and purchasetbl</td><td>
<table>
<tr><th>FIELD</th><th>TYPE</th><th>NULL</th><th>KEY</th><th>DEFAULT</th></tr>
<tr><td>ID</td><td>BIGINT</td><td>NO</td><td>PRI</td><td>NULL</td></tr>
<tr><td>DATE</td><td>DATE</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>SHOE_ID</td><td>BIGINT</td><td>YES</td><td></td><td>NULL</td></tr>
<tr><td>USER_ID</td><td>BIGINT</td><td>YES</td><td></td><td>NULL</td></tr>
</table>
</td></tr>
</table>

For the given schema, we inserted some initial values into all the tables using the **data.sql** file in the project. The initial entries are as given below:

<table>
<tr><th>S no.</th><th>Table name</th><th>Records</th></tr>
<tr><td>1.</td><td>admintbl</td><td>
<table>
<tr><th>ID</th><th>PASSWORD</th><th>USERNAME</th></tr>
<tr><td>1</td><td>admin@123</td><td>admin</td></tr>
<tr><td>2</td><td>aditya@456</td><td>aditya</td></tr>
</table>
</td></tr>
<tr><td>2.</td><td>usertbl</td><td>
<table>
<tr><th>ID</th><th>EMAIL</th><th>MOBILE</th><th>NAME</th><th>SIZE</th></tr>
<tr><td>1</td><td>aditya@yahoo.com</td><td>9991552342</td><td>Aditya</td><td>10.5</td></tr>
<tr><td>2</td><td>shreya@gmail.com</td><td>1559293942</td><td>Shreya</td><td>7.0</td></tr>
<tr><td>3</td><td>rohan@bing.com</td><td>9416643282</td><td>Rohan</td><td>9.5</td></tr>
<tr><td>4</td><td>sarika@outlook.com</td><td>8877678768</td><td>Sarika</td><td>8.0</td></tr>
</table>
</td></tr>
<tr><td>3.</td><td>shoetbl</td><td>
<table>
<tr><th>ID</th><th>CATEGORY</th><th>COLOR</th><th>PRICE</th></tr>
<tr><td>1</td><td>sneakers</td><td>red</td><td>10000.0</td></tr>
<tr><td>2</td><td>tennis</td><td>white</td><td>7000.0</td></tr>
<tr><td>3</td><td>cricket</td><td>red</td><td>12000.0</td></tr>
<tr><td>4</td><td>running</td><td>blue</td><td>7000.0</td></tr>
<tr><td>5</td><td>tennis</td><td>yellow</td><td>12000.0</td></tr>
</table>
</td></tr>
<tr><td>4.</td><td>purchasetbl</td><td>
<table>
<tr><th>ID</th><th>DATE</th><th>SHOE_ID</th><th>USER_ID</th></tr>
<tr><td>1</td><td>2022-01-16</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>2022-01-16</td><td>1</td><td>4</td></tr>
<tr><td>3</td><td>2022-09-16</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>2021-09-08</td><td>3</td><td>2</td></tr>
<tr><td>5</td><td>2021-07-08</td><td>3</td><td>4</td></tr>
<tr><td>6</td><td>2018-12-20</td><td>2</td><td>2</td></tr>
<tr><td>7</td><td>2022-07-16</td><td>1</td><td>3</td></tr>
<tr><td>8</td><td>2018-12-20</td><td>2</td><td>3</td></tr>
</table>
</td></tr>
</table>

API implementation details

The product features were divided into a set of 17 different REST APIs as described in the following four sections:

1. **Section - A:** 2 APIs describing the Admin login related functionalities mentioned in Point 1) in the Product Features section.
2. **Section - B:** 6 APIs describing the User related functionalities mentioned in Point 2) in the Product Features section.
3. **Section - C:** 8 APIs describing the Shoe related functionality mentioned in Point 3) in the Product Features section.
4. **Section - D:** 1 API describing the Purchase Report-related functionality mentioned in Point 4) in the Product Features section.

Note: The user interaction with the APIs was done through the Postman utility.

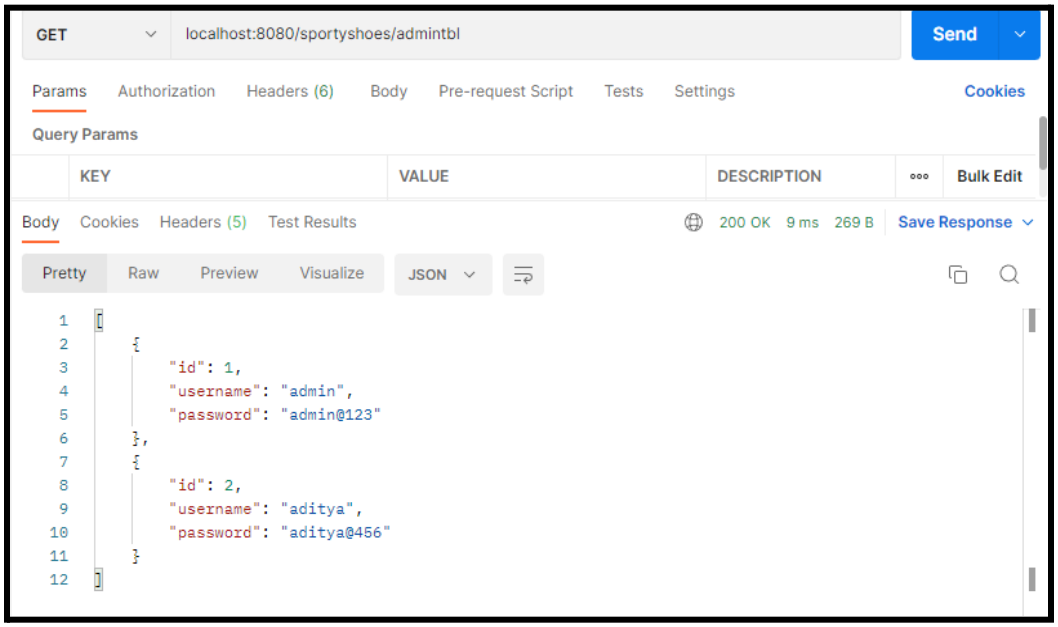
Section - A: Admin table requests

The following table describes all the APIs created for the Admin table described by the name *"admintbl".*

S no.	HTTP Method	Functionality	URI	Status Code
A.1	GET	Display all entries of Admin table	localhost:8080/sportyshoes/admintbl	200
A.2	PUT	Update the password of an entry in the Admin table	localhost:8080/sportyshoes/admintbl/{username}	204

A.1) Display all entries of Admin table

- **API end point:** localhost:8080/sportyshoes/admintbl
- **Postman response:**

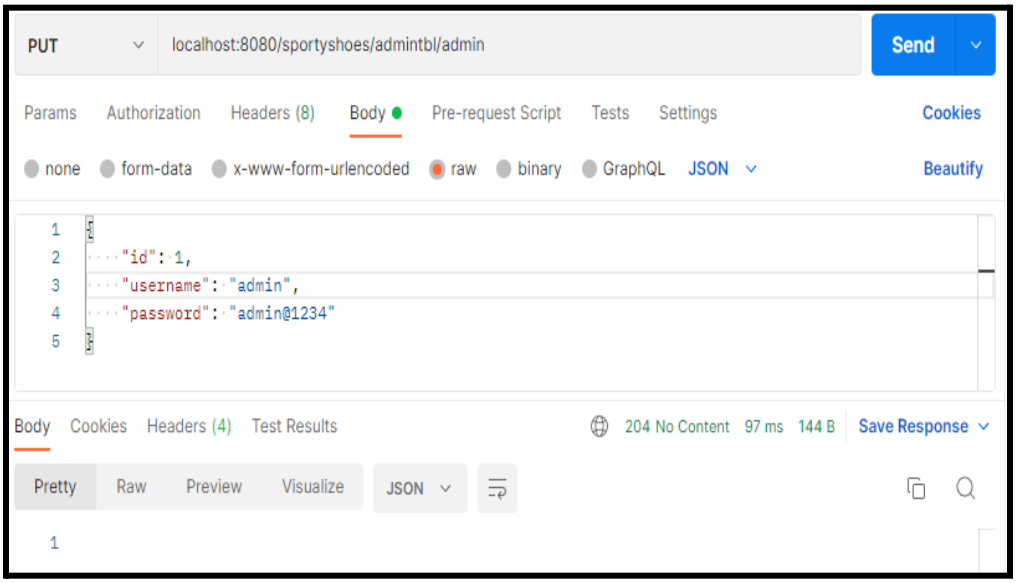


- **JSON response:**

```
[
    {
        "id": 1,
        "username": "admin",
        "password": "admin@123"
    },
    {
        "id": 2,
        "username": "aditya",
        "password": "aditya@456"
    }
]
```

A.2) Update password for given username

- **API end point:** localhost:8080/sportyshoes/username
- **Postman response:**



- **Updated *admintbl*:**

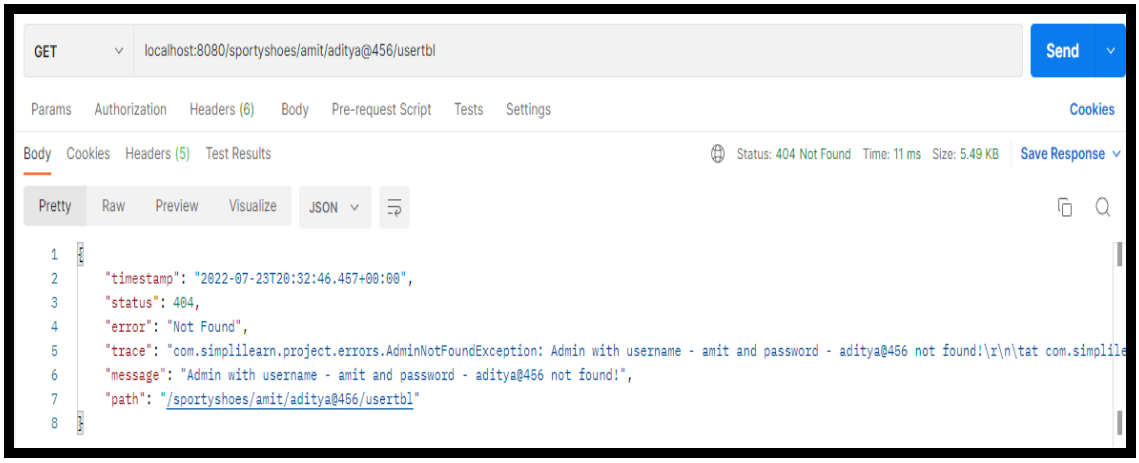
Before	After

ID	PASSWORD	USERNAME
1	admin@123	admin
2	aditya@456	aditya

ID	PASSWORD	USERNAME
1	admin@1234	admin
2	aditya@456	aditya

Here onwards, we'll be using the following admin credentials for testing the remaining APIs:

- **Admin username:** aditya
- **Admin password:** aditya@456
- Note: The API throws a RunTime Exception called **AdminNotFoundException** with 404 status code in case an **admin with the specified username and password combination doesn't exist.** For example: Querying a sample API(API 1 from the Section: User table requests) for **a nonexistent username "amit"** results in the following(Note that the error trace has been shortened for readability):



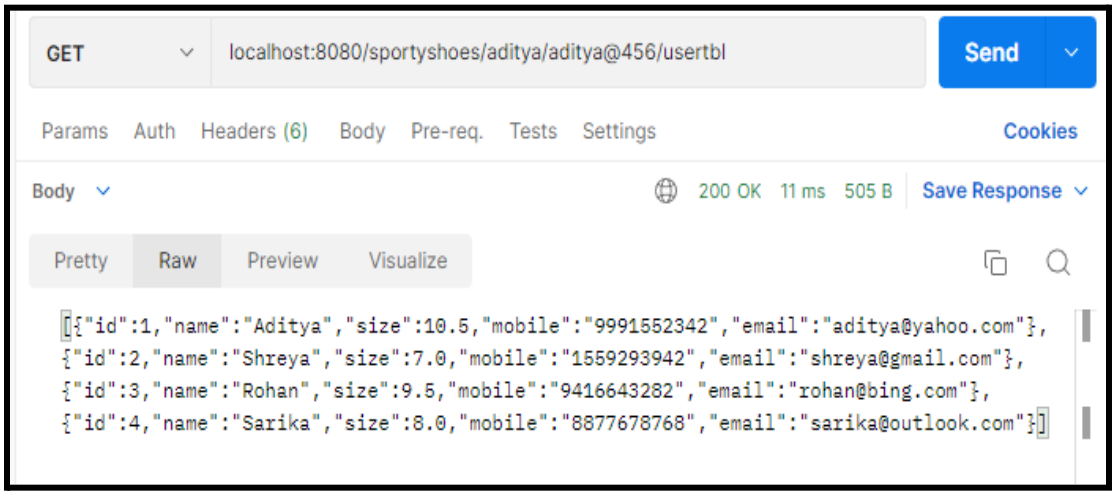
Section - B: User table requests

The following table describes all the APIs created for the User table described by the name *"usertbl".*

S no.	HTTP Method	Functionality	URI	Status Code
B.1	GET	Display all entries of users in *usertbl*	localhost:8080/sportyshoes/{username}/{password}/usertbl	200
B.2	GET	Search users in *usertbl* on basis of id	localhost:8080/sportyshoes/{username}/{password}/usertbl/id/{id}	200
B.3	GET	Search users in *usertbl* on basis of name	localhost:8080/sportyshoes/{username}/{password}/usertbl/name/{name}	200
B.4	GET	Search users in *usertbl* on basis of size	localhost:8080/sportyshoes/{username}/{password}/usertbl/size/{size}	200
B.5	GET	Search users in *usertbl* on basis of mobile number	localhost:8080/sportyshoes/{username}/{password}/usertbl/mobile/{mobile}	200
B.6	GET	Search users in *usertbl* on basis of email address	localhost:8080/sportyshoes/{username}/{password}/usertbl/email/{email}	200

B.1) Display all entries of users in *usertbl*

- **API end point:** localhost:8080/sportyshoes/{username}/{password}/usertbl
- **Postman response:**

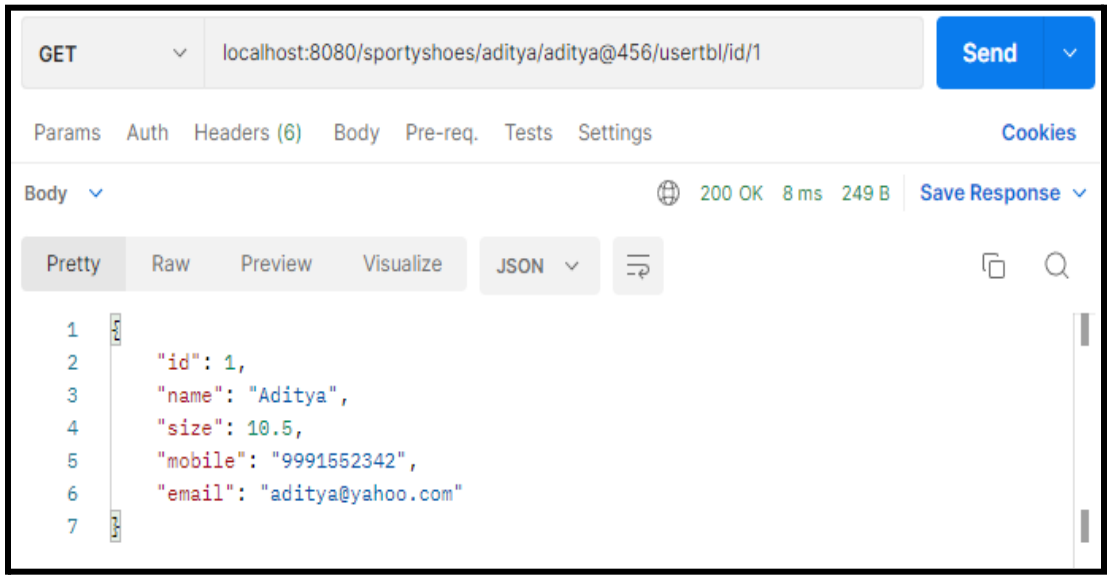


- **Beautified JSON response:**

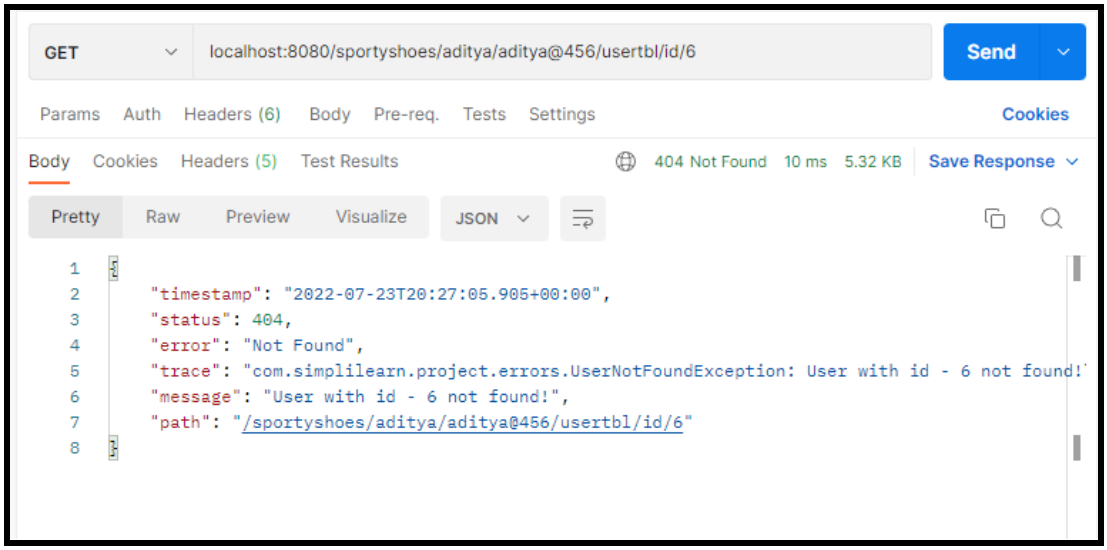

```
[
    {
        "id": 1,
        "name": "Aditya",
        "size": 10.5,
        "mobile": "9991552342",
        "email": "aditya@yahoo.com"
    },
    {
        "id": 2,
        "name": "Shreya",
        "size": 7.0,
        "mobile": "1559293942",
        "email": "shreya@gmail.com"
    },
    {
        "id": 3,
        "name": "Rohan",
        "size": 9.5,
        "mobile": "9416643282",
        "email": "rohan@bing.com"
    },
    {
        "id": 4,
        "name": "Sarika",
        "size": 8.0,
        "mobile": "8877678768",
        "email": "sarika@outlook.com"
    }
]
```

B.2) Search users in *usertbl* on basis of id

- **API end point:** localhost:8080/sportyshoes/{username}/{password}/usertbl/id/{id}
- **Postman response(for id = 1):**

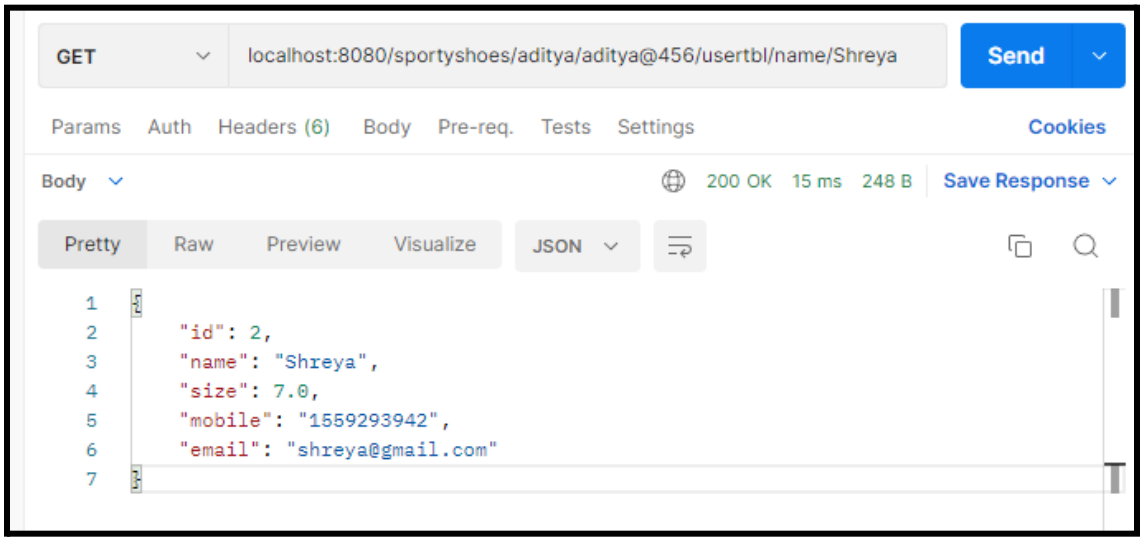


- **Note:** The API throws a RunTime Exception called **UserNotFoundException** with 404 status code in case **a user with specified id doesn't exist.** For example: Querying the API for a **nonexistent id = 6** results in the following(Note that the error trace has been shortened for readability):

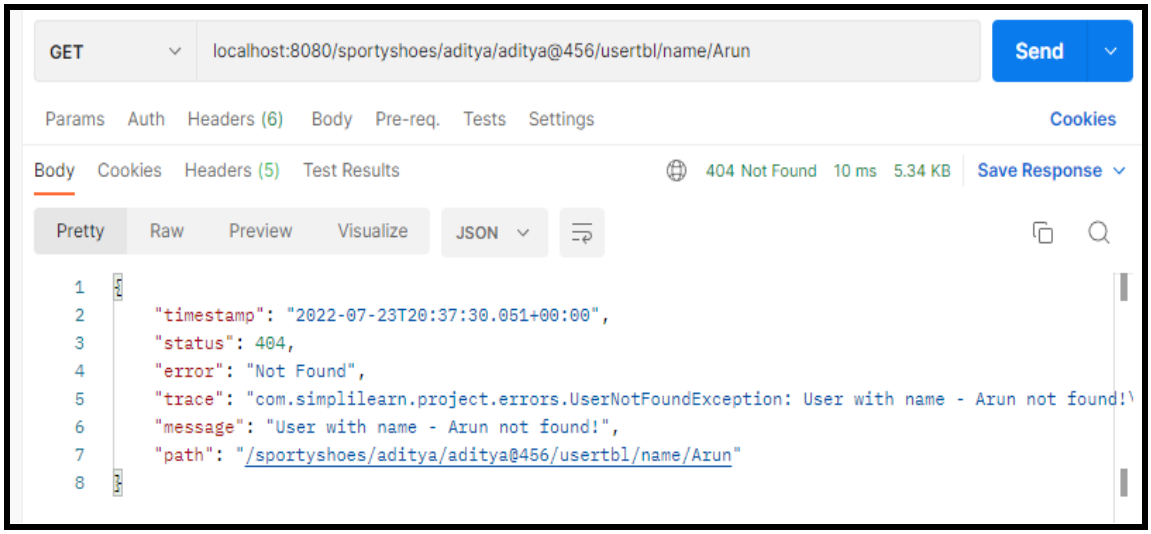


B.3) Search users in *usertbl* on the basis of name

- **API endpoint:**
  localhost:8080/sportyshoes/{username}/{password}/usertbl/name/{name}
- **Postman response(for name = "Shreya"):**

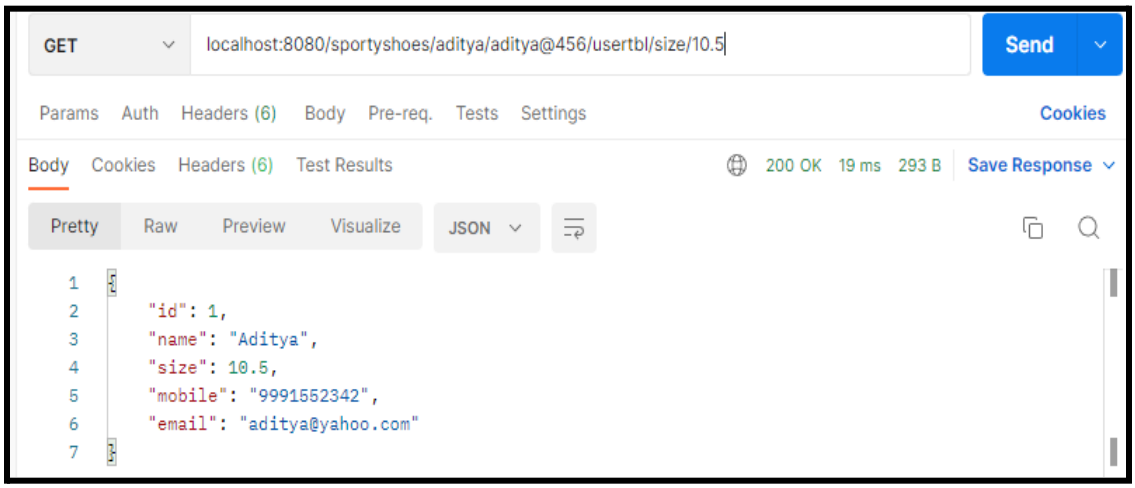


- **Note:** The API throws a RunTime Exception called **UserNotFoundException** with 404 status code in case **a user with the specified name doesn't exist.** For example: Querying the API for a **nonexistent name = "Arun"** results in the following(Note that the error trace has been shortened for readability):

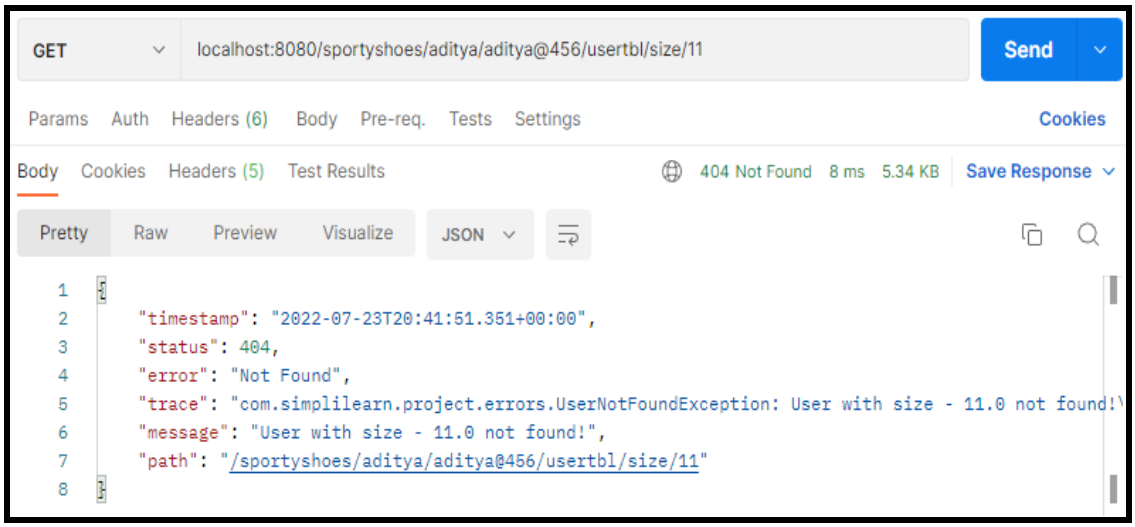


B.4) Search users in *usertbl* on the basis of size

- **API endpoint:**
  localhost:8080/sportyshoes/{username}/{password}/usertbl/size/{size}
- **Postman response(for size = 10.5):**

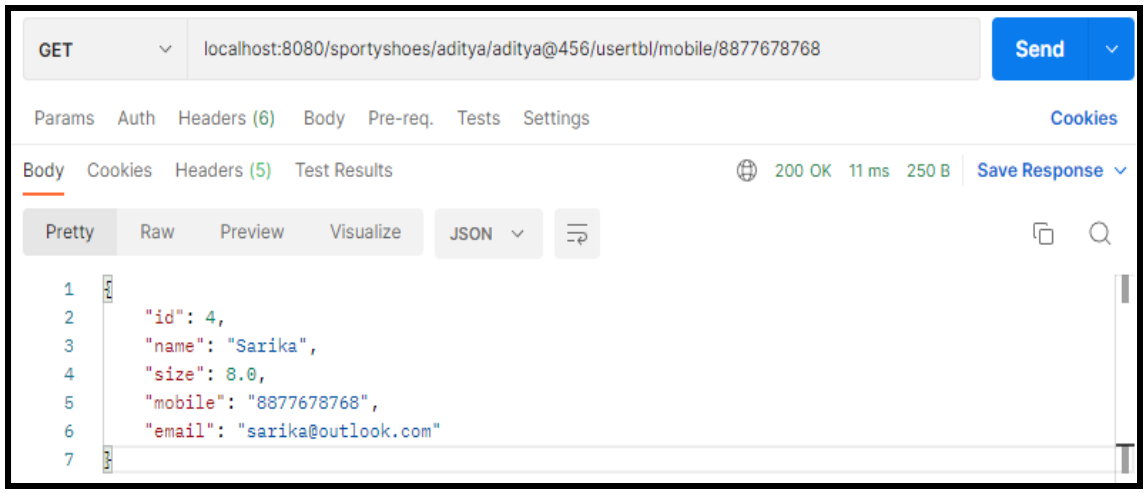


- **Note:** The API throws a RunTime Exception called **UserNotFoundException** with 404 status code in case **a user with specified size doesn't exist.** For example: Querying the API for a **nonexistent size = 11** results in the following(Note that the error trace has been shortened for readability):

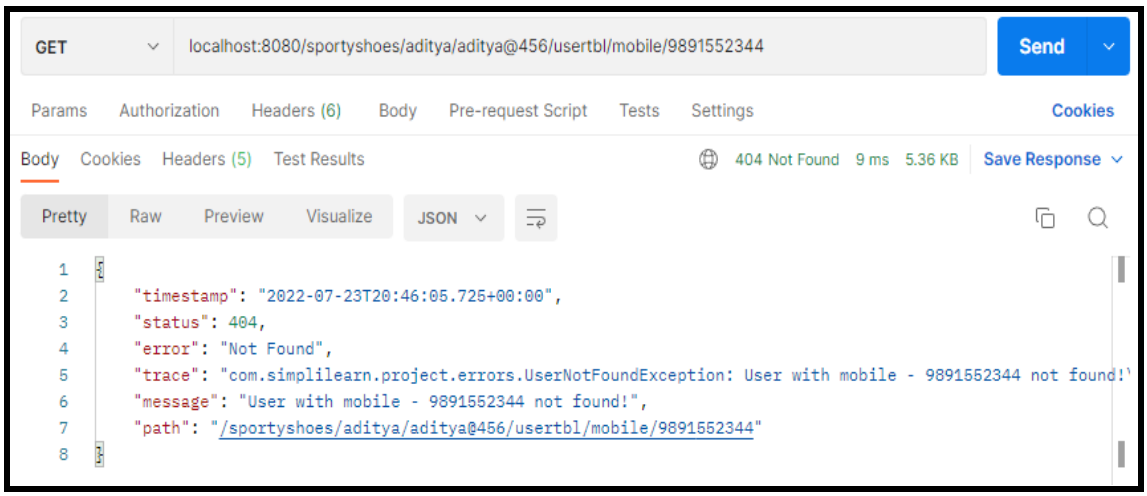


B.5) Search users in *usertbl* on the basis of mobile number

- **API endpoint:**
  localhost:8080/sportyshoes/{username}/{password}/usertbl/mobile/{mobile}
- **Postman response(for mobile = 8877678768):**

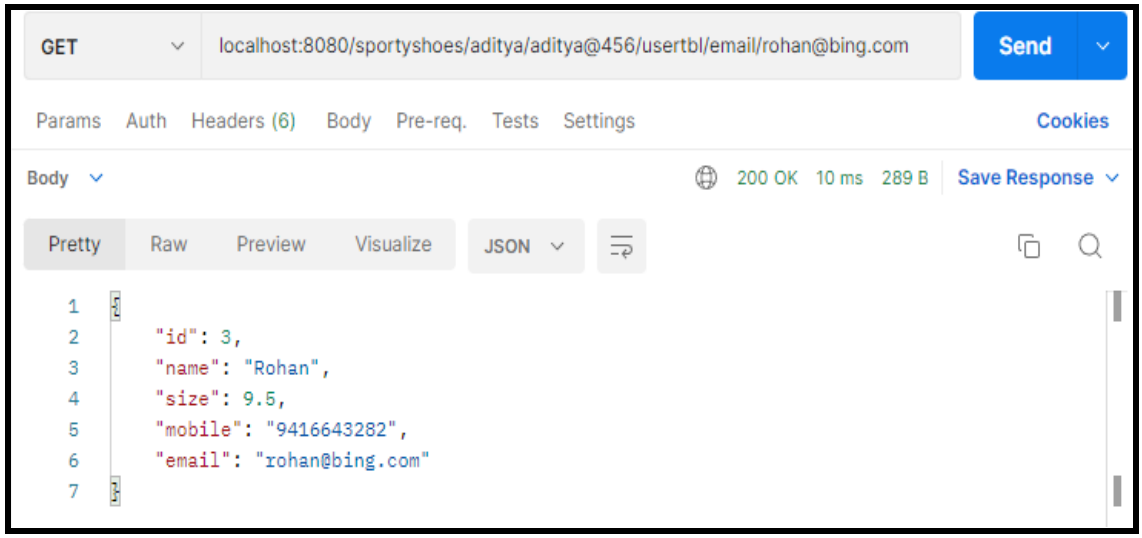


- **Note:** The API throws a RunTime Exception called **UserNotFoundException** with 404 status code in case **a user with a specified mobile number doesn't exist.** For example: Querying the API for a **mobile = 9891552344** results in the following(Note that the error trace has been shortened for readability):

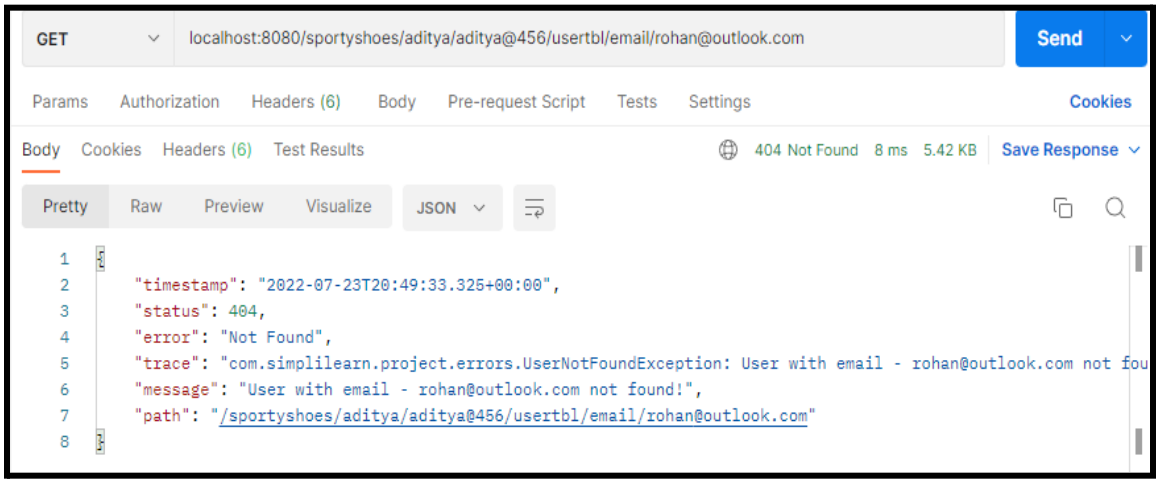


B.6) Search users in *usertbl* on the basis of email address

- **API endpoint:**
  localhost:8080/sportyshoes/{username}/{password}/usertbl/email/{email}
- **Postman response(for email = rohan@bing.com):**



- **Note:** The API throws a RunTime Exception called **UserNotFoundException** with 404 status code in case **a user with specified email doesn't exist.** For example: Querying the API for an **email = rohan@outlook.com** results in the following(Note that the error trace has been shortened for readability):



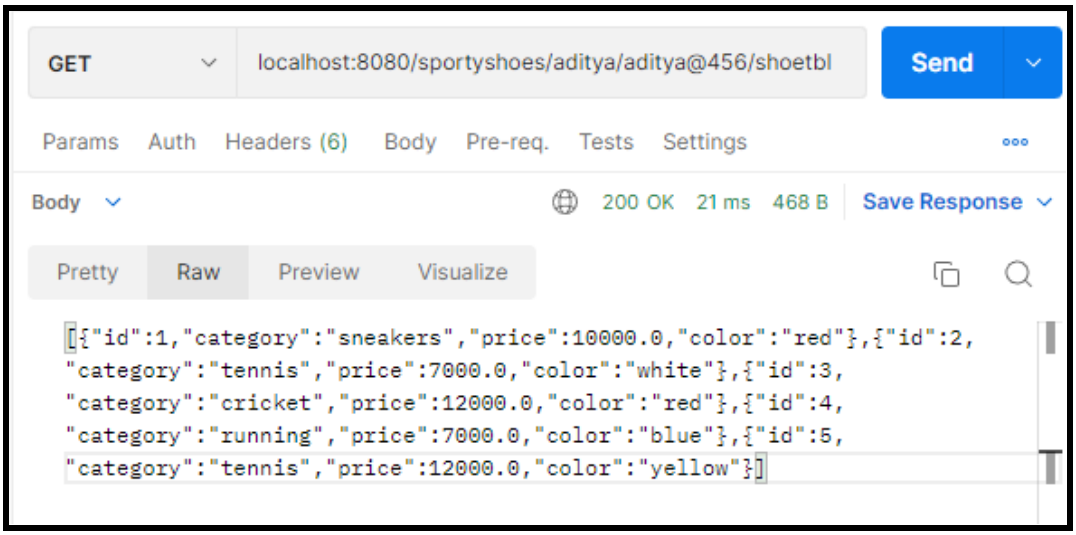
Section - C: Shoe table requests

The following table describes all the APIs created for the Shoe table described by the name *"shoetbl".*

S no.	HTTP Method	Functionality	URI	Status Code
C.1	GET	Display all entries of users in *shoetbl*	localhost:8080/sportyshoes/{username}/{password}/shoetbl	200
C.2	GET	Search shoes in *shoetbl* on basis of id	localhost:8080/sportyshoes/{username}/{password}/shoetbl/id/{id}	200
C.3	GET	Search shoes in *shoetbl* on basis of color	localhost:8080/sportyshoes/{username}/{password}/shoetbl/color/{color}	200
C.4	GET	Search shoes in *shoetbl* on basis of category	localhost:8080/sportyshoes/{username}/{password}/shoetbl/category/{category}	200
C.5	GET	Search shoes in *shoetbl* on basis of price	localhost:8080/sportyshoes/{username}/{password}/shoetbl/price/{price}	200
C.6	POST	Adding new entry into *shoetbl*	localhost:8080/sportyshoes/{username}/{password}/shoetbl	201
C.7	PUT	Updating properties of an existing entry in *shoetbl*	localhost:8080/sportyshoes/{username}/{password}/shoetbl/id/{id}	204
C.8	DELETE	Deleting existing entry from *shoetbl*	localhost:8080/sportyshoes/{username}/{password}/shoetbl/id/{id}	204

C.1) Display all entries of shoes in *shoetbl*

- **API end point:** localhost:8080/sportyshoes/{username}/{password}/shoetbl
- **Postman response:**



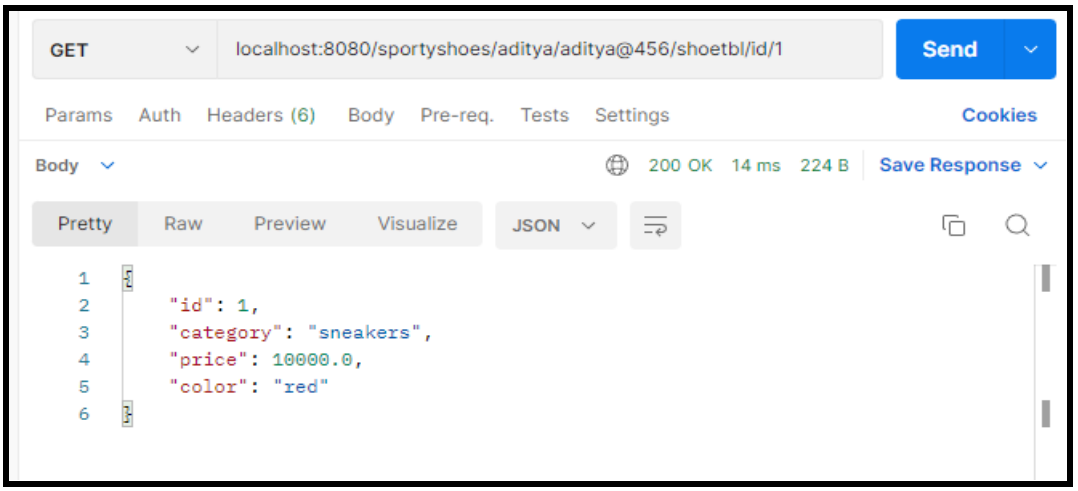
- **Beautified JSON response:**


```
[
    {
        "id": 1,
        "category": "sneakers",
        "price": 10000.0,
        "color": "red"
    },
    {
        "id": 2,
        "category": "tennis",
        "price": 7000.0,
        "color": "white"
    },
    {
        "id": 3,
        "category": "cricket",
        "price": 12000.0,
        "color": "red"
    },
    {
        "id": 4,
        "category": "running",
        "price": 7000.0,
        "color": "blue"
    },
```

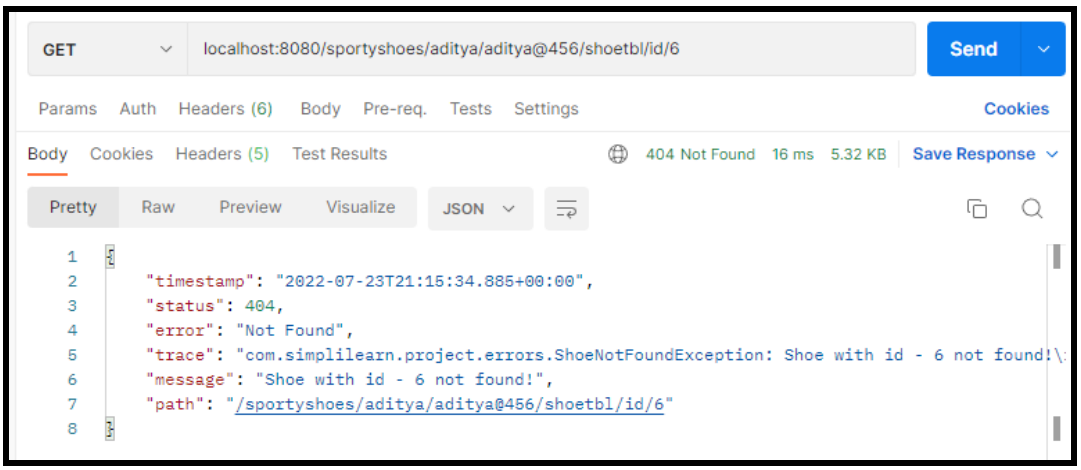
```
    {
        "id": 5,
        "category": "tennis",
        "price": 12000.0,
        "color": "yellow"
    }
]
```


C.2) Search shoes in *shoetbl* on basis of id

- **API end point:** localhost:8080/sportyshoes/{username}/{password}/shoetbl/id/{id}
- **Postman response(for id=1):**

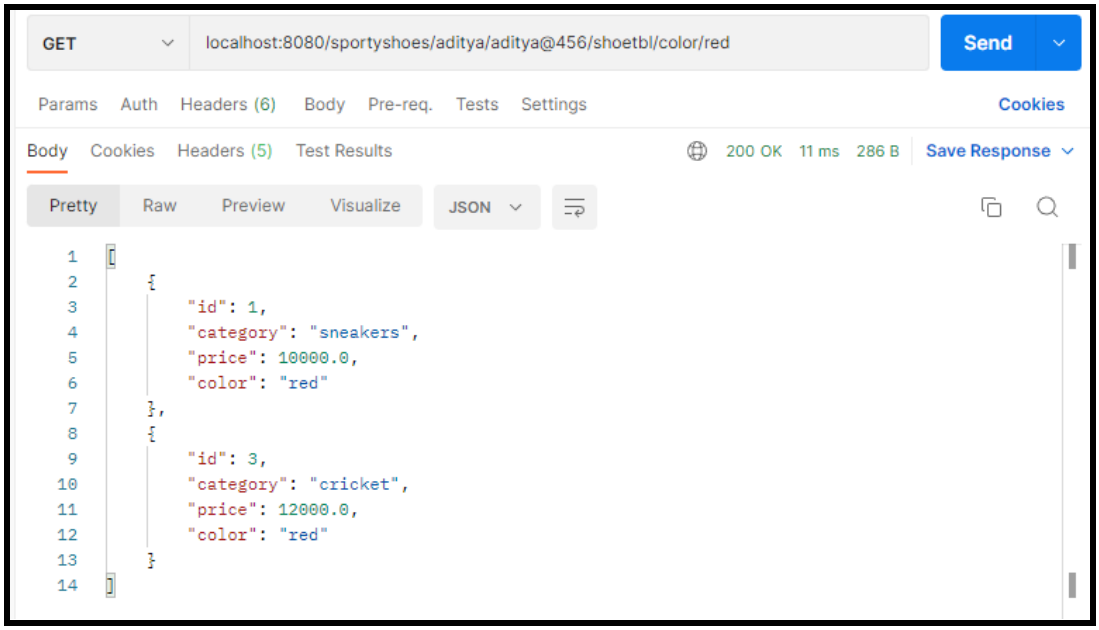


- **Note:** The API throws a RunTime Exception called **ShoeNotFoundException** with 404 status code in case **a shoe with specified id doesn't exist.** For example: Querying the API for a **nonexistent id = 6** results in the following(Note that the error trace has been shortened for readability):

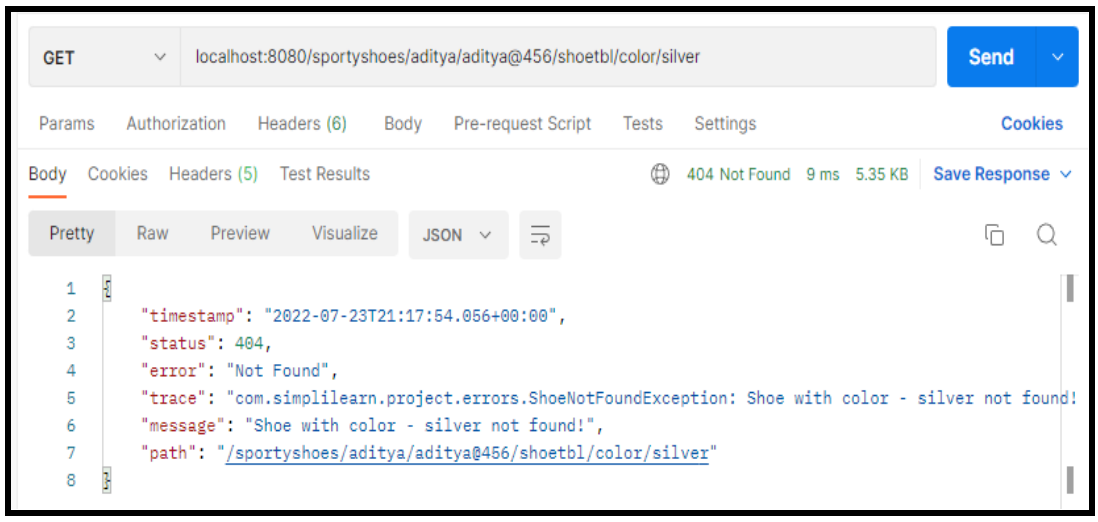


C.3) Search shoes in *shoetbl* on basis of color

- **API end point:**
  localhost:8080/sportyshoes/{username}/{password}/shoetbl/color/{color}
- **Postman response(for color = "red"):**

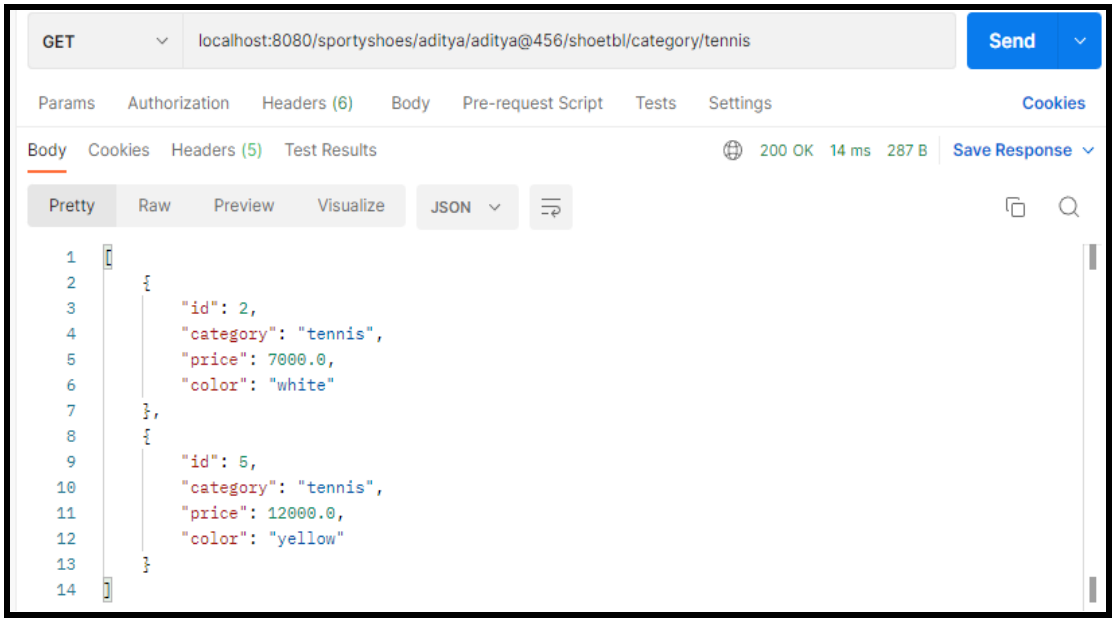


- **Note:** The API throws a RunTime Exception called **ShoeNotFoundException** with 404 status code in case **a shoe with specified color doesn't exist.** For example: Querying the API for a **nonexistent color = "silver"** results in the following(Note that the error trace has been shortened for readability):

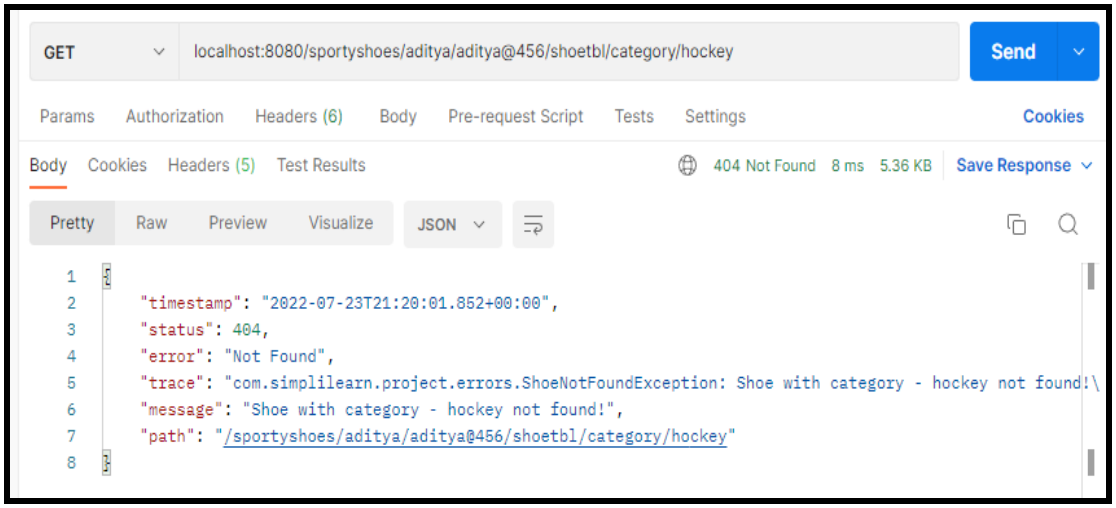


C.4) Search shoes in *shoetbl* on basis of category

- **API end point:**
  localhost:8080/sportyshoes/{username}/{password}/shoetbl/category/{category}
- **Postman response(for category = "tennis"):**

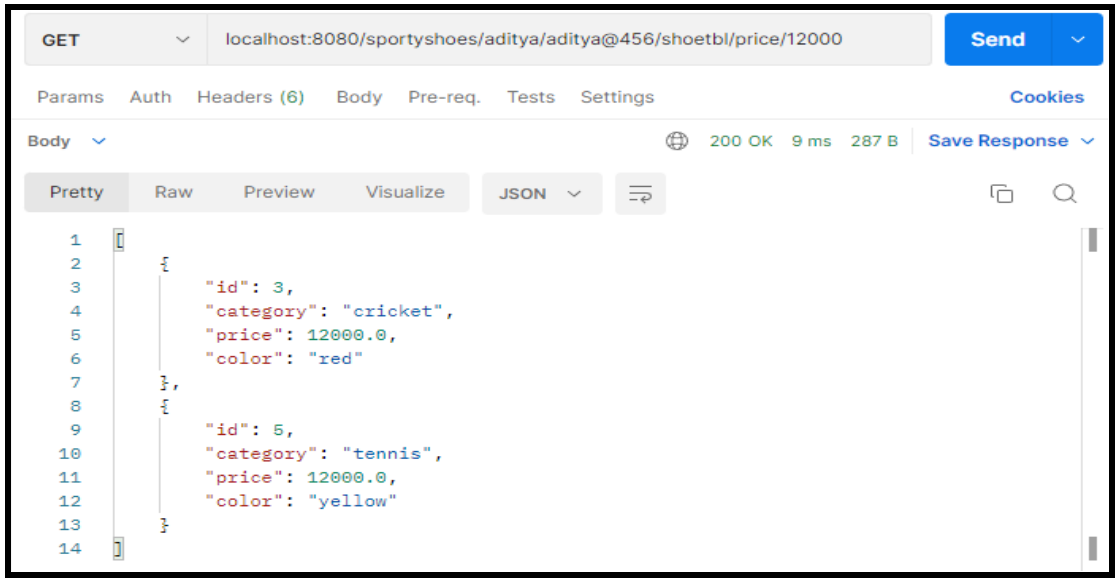


- **Note:** The API throws a RunTime Exception called **ShoeNotFoundException** with 404 status code in case **a shoe with specified category doesn't exist.** For example: Querying the API for a **nonexistent category = "hockey"** results in the following(Note that the error trace has been shortened for readability):

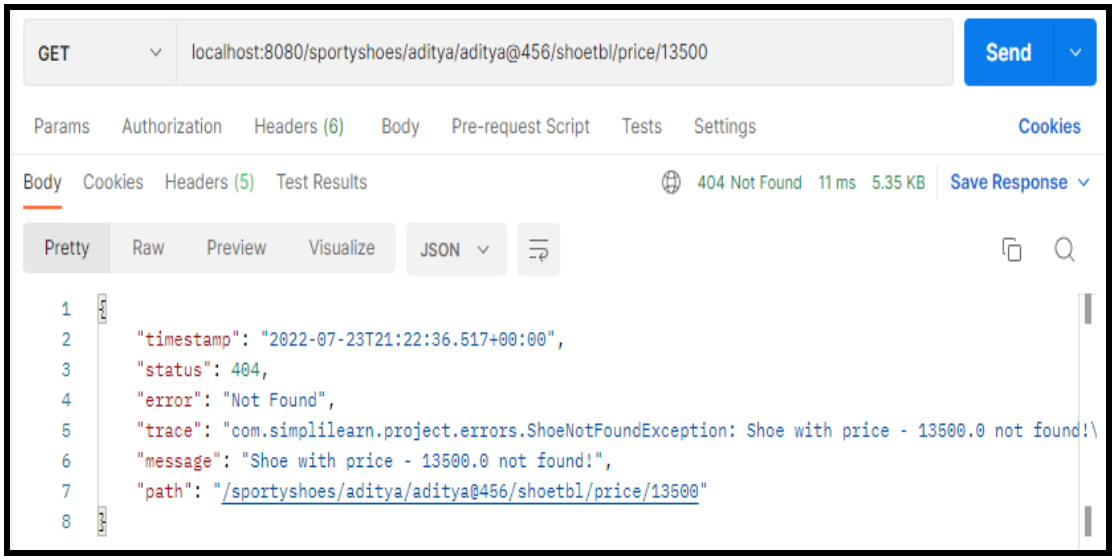


C.5) Search shoes in *shoetbl* on basis of price

- **API end point:**
  localhost:8080/sportyshoes/{username}/{password}/shoetbl/price/{price}
- **Postman response(for price = 12000):**



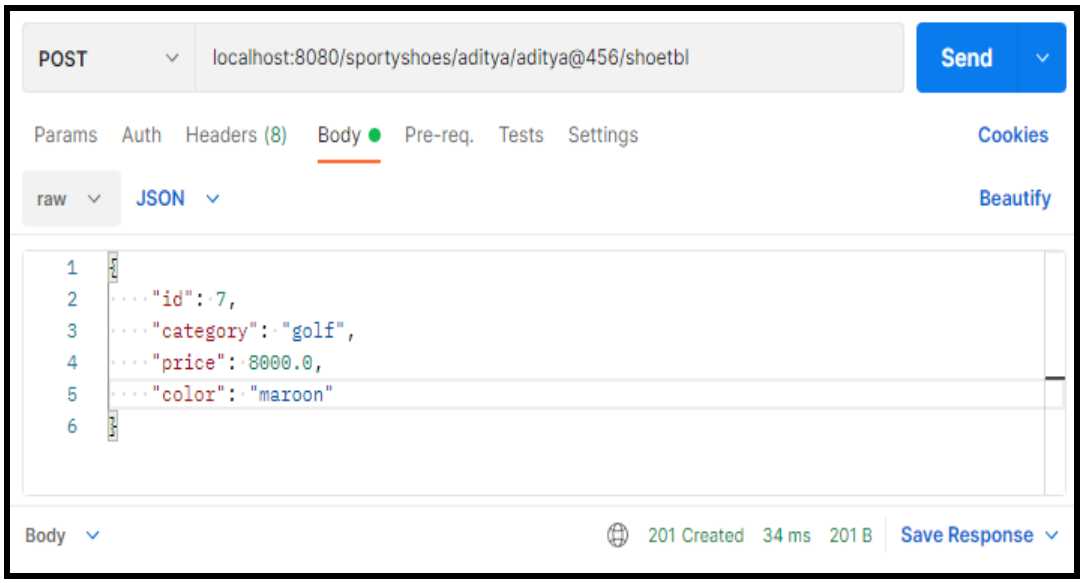
- **Note:** The API throws a RunTime Exception called **ShoeNotFoundException** with 404 status code in case **a shoe with specified price doesn't exist.** For example: Querying the API for a **nonexistent price = 13500** results in the following(Note that the error trace has been shortened for readability):



C.6) Adding new entry into *shoetbl*

- **API end point:** localhost:8080/sportyshoes/{username}/{password}/shoetbl

- **Postman response:**



- **Updated *shoetbl*:**

Before	After

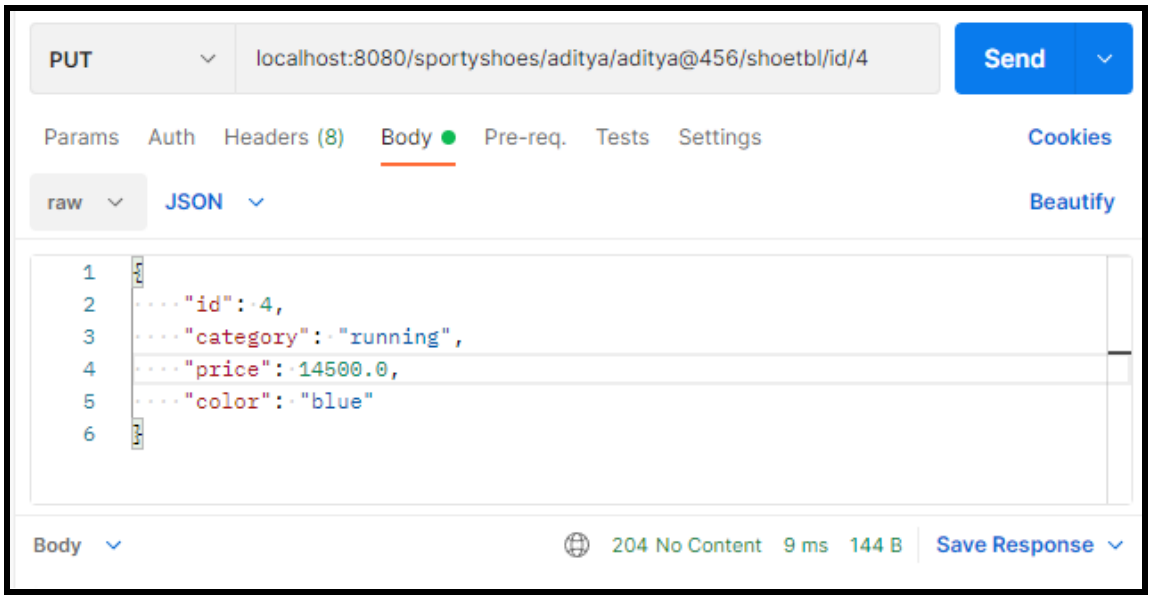
ID	CATEGORY	COLOR	PRICE
1	sneakers	red	10000.0
2	tennis	white	7000.0
3	cricket	red	12000.0
4	running	blue	7000.0
5	tennis	yellow	12000.0

ID	CATEGORY	COLOR	PRICE
1	sneakers	red	10000.0
2	tennis	white	7000.0
3	cricket	red	12000.0
4	running	blue	7000.0
5	tennis	yellow	12000.0
6	golf	maroon	8000.0

C.7) Updating properties of an existing entry in *shoetbl*

- **API end point:** localhost:8080/sportyshoes/{username}/{password}/shoetbl/id/{id}

- **Postman response(for updating price of shoe at id = 4 from 7000 to 14500):**



- **Updated *shoetbl*:**

Before

ID	CATEGORY	COLOR	PRICE
1	sneakers	red	10000.0
2	tennis	white	7000.0
3	cricket	red	12000.0
4	running	blue	7000.0
5	tennis	yellow	12000.0
6	golf	maroon	8000.0

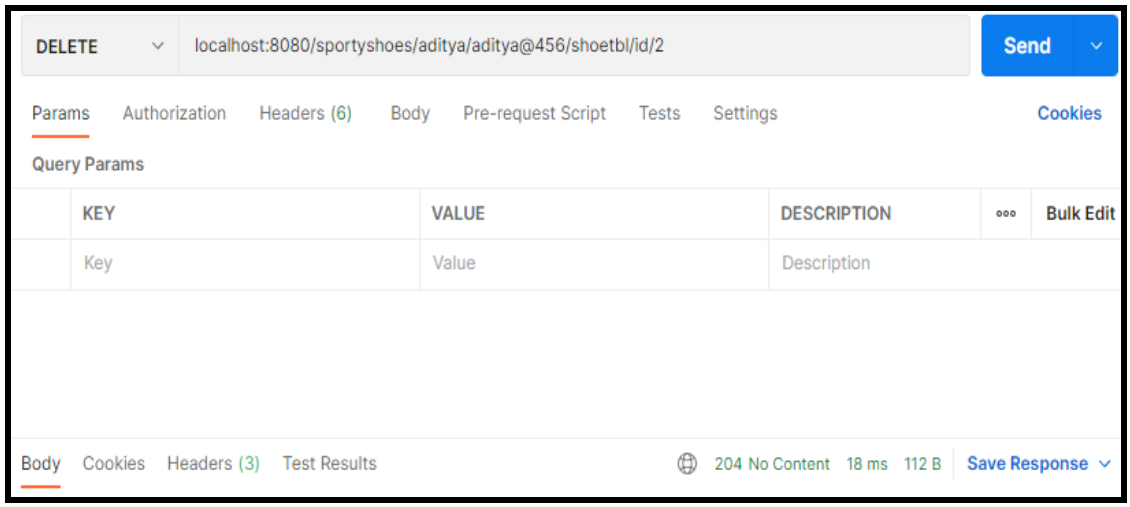
After

ID	CATEGORY	COLOR	PRICE
1	sneakers	red	10000.0
2	tennis	white	7000.0
3	cricket	red	12000.0
4	running	blue	14500.0
5	tennis	yellow	12000.0
6	golf	maroon	8000.0

C.8) Deleting existing entry from *shoetbl*

- **API end point:** localhost:8080/sportyshoes/{username}/{password}/shoetbl/id/{id}

- **Postman response(for deleting id = 2):**



- **Updated *shoetbl*:**

Before

ID	CATEGORY	COLOR	PRICE
1	sneakers	red	10000.0
2	tennis	white	7000.0
3	cricket	red	12000.0
4	running	blue	7000.0
5	tennis	yellow	12000.0
6	golf	maroon	8000.0

After

ID	CATEGORY	COLOR	PRICE
1	sneakers	red	10000.0
3	cricket	red	12000.0
4	running	blue	14500.0
5	tennis	yellow	12000.0
6	golf	maroon	8000.0

- **Updated *purchasetbl:*** Note that we have set an assumption that any parent entry possessing a foreign key relationship to the Shoe entry will also get deleted when a given Shoe entry is elated. In this case, all purchases containing the shoe_id=2 are also deleted.

Before

ID	DATE	SHOE_ID	USER_ID
1	2022-01-16	1	1
2	2022-01-16	1	4
3	2022-09-16	2	3
4	2021-09-08	3	2
5	2021-07-08	3	4
6	2018-12-20	2	2
7	2022-07-16	1	3
8	2018-12-20	2	3

After

ID	DATE	SHOE_ID	USER_ID
1	2022-01-16	1	1
2	2022-01-16	1	4
4	2021-09-08	3	2
5	2021-07-08	3	4
7	2022-07-16	1	3

Section - D: Purchase table requests

The following table describes all the APIs created for the Purchase Reports table described by the name *"purchasetbl".*

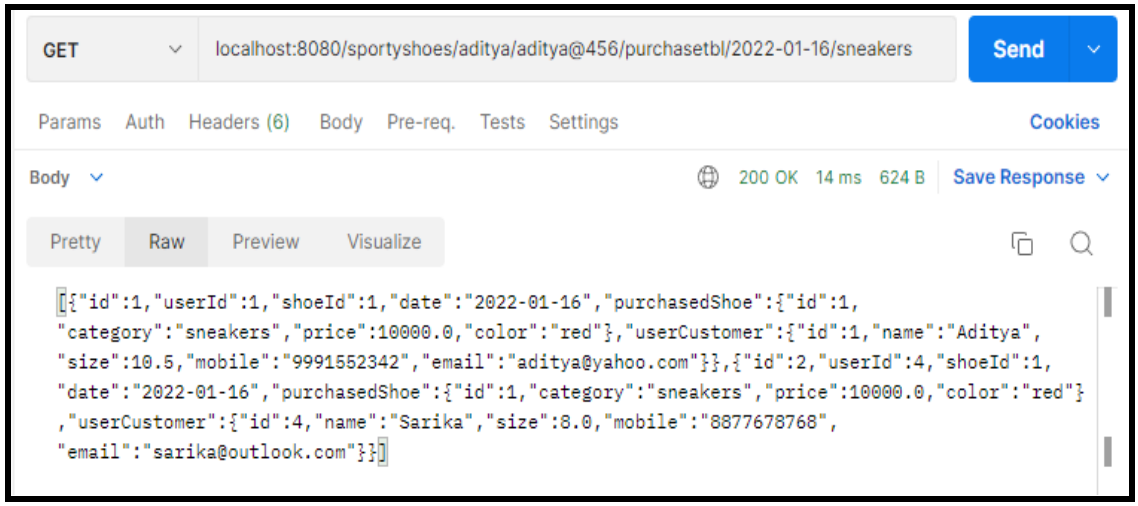
S no.	HTTP Method	Functionality	URI	Status Code
D.1	GET	Search purchase reports filtered by purchase date and shoe category	localhost:8080/sportyshoes/{username}/{password}/purchasetbl/{date}/{category}	200

D.1) Search purchase reports in *purchasetbl* on basis of purchase date and shoe category

- **API end point:**

  localhost:8080/sportyshoes/{username}/{password}/purchasetbl/{date}/{category}

- **Postman response(for date = 2022-01-16 and category = “sneakers”):**

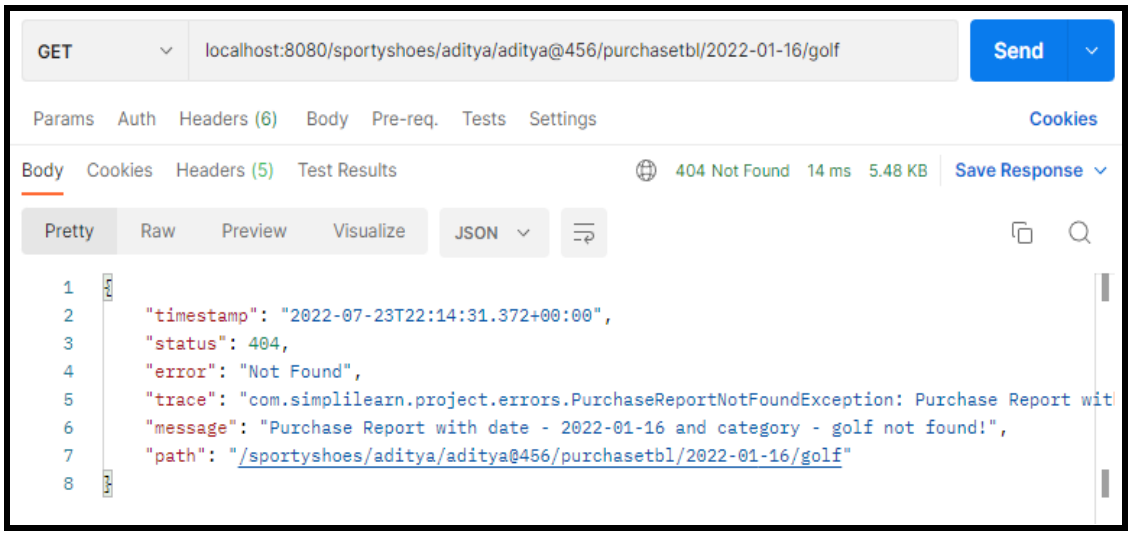


- **Beautified JSON response:**


```
[
    {
        "id": 1,
        "userId": 1,
        "shoeId": 1,
        "date": "2022-01-16",
        "purchasedShoe": {
            "id": 1,
            "category": "sneakers",
            "price": 10000.0,
            "color": "red"
        },
        "userCustomer": {
            "id": 1,
            "name": "Aditya",
            "size": 10.5,
            "mobile": "9991552342",
            "email": "aditya@yahoo.com"
        }
    },
    {
        "id": 2,
        "userId": 4,
```

```
        "shoeId": 1,
        "date": "2022-01-16",
        "purchasedShoe": {
            "id": 1,
            "category": "sneakers",
            "price": 10000.0,
            "color": "red"
        },
        "userCustomer": {
            "id": 4,
            "name": "Sarika",
            "size": 8.0,
            "mobile": "8877678768",
            "email": "sarika@outlook.com"
        }
    }
]
```


- **Note:** The API throws a RunTime Exception called **PurchaseReportNotFoundException** with 404 status code in case **a purchase report with the specified purchase date and category combination doesn't exist.** For example: Querying the API for a **nonexistent entry with date = 2022-01-16 and category = "golf",** results in the following(Note that the error trace has been shortened for readability):



Source code & Documentation

- Apart from the attached .zip file on the Simplilearn portal, the entire project code is hosted on Github at https://github.com/adityasaini70/Prolim-Project2.
- The Swagger UI tool was also used to generate a ready-to-use reference for all the APIs described in the previous sections. For quick testing, the same documentation can be accessed at http://localhost:8080/swagger-ui/index.html after running the Spring Boot Application.

Conclusion

In its current state, the USP of the application is its ease of usage and modular code base. In the future, we can make the project more appealing to the end user by adding a front-end-based layer.