# SportyShoes.com implementation in SpringBoot using JPA coupled with H2 database

## Programmer Details:

**Name: Saad Ahmad**
**Project Name: Simplilearn phase 2 project**
**Github repo: https://github.com/DaasDaham/prolim-project-2**

## Database tables

1. User: It contains the details of each user
2. Shoe: It contains the details of each shoe
3. Purchase: It contains the details of purchases made by users. It links to User and Shoe table through foreign keys.

## Table Specifications

### User Table

| | |
|---|---|
| Id | Autogenerated |
| username | varchar |
| emailAddress | varchar |
| hasSignedUp | bool |

All the above fields are self explanatory except the hasSignedUp column. It refers to the state whether a particular user has signed up on the platform or not.

### Shoe Table

| | |
|---|---|
| Id | Autogenerated |
| price | Double |
| size | Integer |
| category | varchar |
| color | varchar |

**Purchase Table**

| | |
|---|---|
| Id | Autogenerated |
| userId | Foreign key linking to user table |
| shoeId | Foreign key linking to shoe table |
| date | Date of purchase |

## Core tasks

The core tasks that were listed on simplilearn for the project were:
1. Manage products in the store i.e. shoes
2. Browse list of users who have signed up and be able to search users
3. See purchase reports filtered by category and date

All of the above functionalities have been implemented using REST APIs and have the following endpoints.
1. Manage shoes: GET: /saadsportyshoes/shoe_api/id/{id}, /saadsportyshoes/user_api, /saadsportyshoes/shoe_api/color/{color}, /saadsportyshoes/shoe_api/color/{category}, /saadsportyshoes/shoe_api/color/{price}, /saadsportyshoes/shoe_api/color/{size}; POST: /saadsportyshoes/shoe_api; PUT: /saadsportyshoes/shoe_api/id/{id}; DELTE: /saadsportyshoes/shoe_api/id/{id}.
2. Browse users: GET: /saadsportyshoes/user_api/hasSignedUp/{signedUpState}, /saadsportyshoes/user_api/username/{username}, /saadsportyshoes/user_api/username/{emailAddress}, etc.
3. See purchase reports filtered by date and category: GET: /saadsportyshoes/purchase_api/{purchaseDate}/{category}

Apart from the above mentioned rest api’s some other APIs have also been implemented which can be found in the com.simplilearn.project.resources package.

The working of APIs are self explanatory by looking at the function names under the endpoint mapping.

## Java concepts used in the project:

1. Interfaces were heavily used in the project and some interfaces were extended from other interfaces. For example we have various services interfaces with their

implementations as classes. Similarly the repository interfaces extend the JpaRepository<T,D> interface which contains the declarations for method named queries.

2. Annotations from JPA and springboot were extensively used to implement different components. Some examples are @Service, @Autowired, @GetMapping, @Entity, @ManyToOne. These annotations were used in defining the model and its relation to other models, the controllers, services, etc.
3. Exception handling with user defined exceptions.
4. Use of java collection objects such as Lists.
5. SpringBoot framework was used along with JPA and H2. These allow for easy prototyping and testing the APIs with an inbuilt tomcat server.

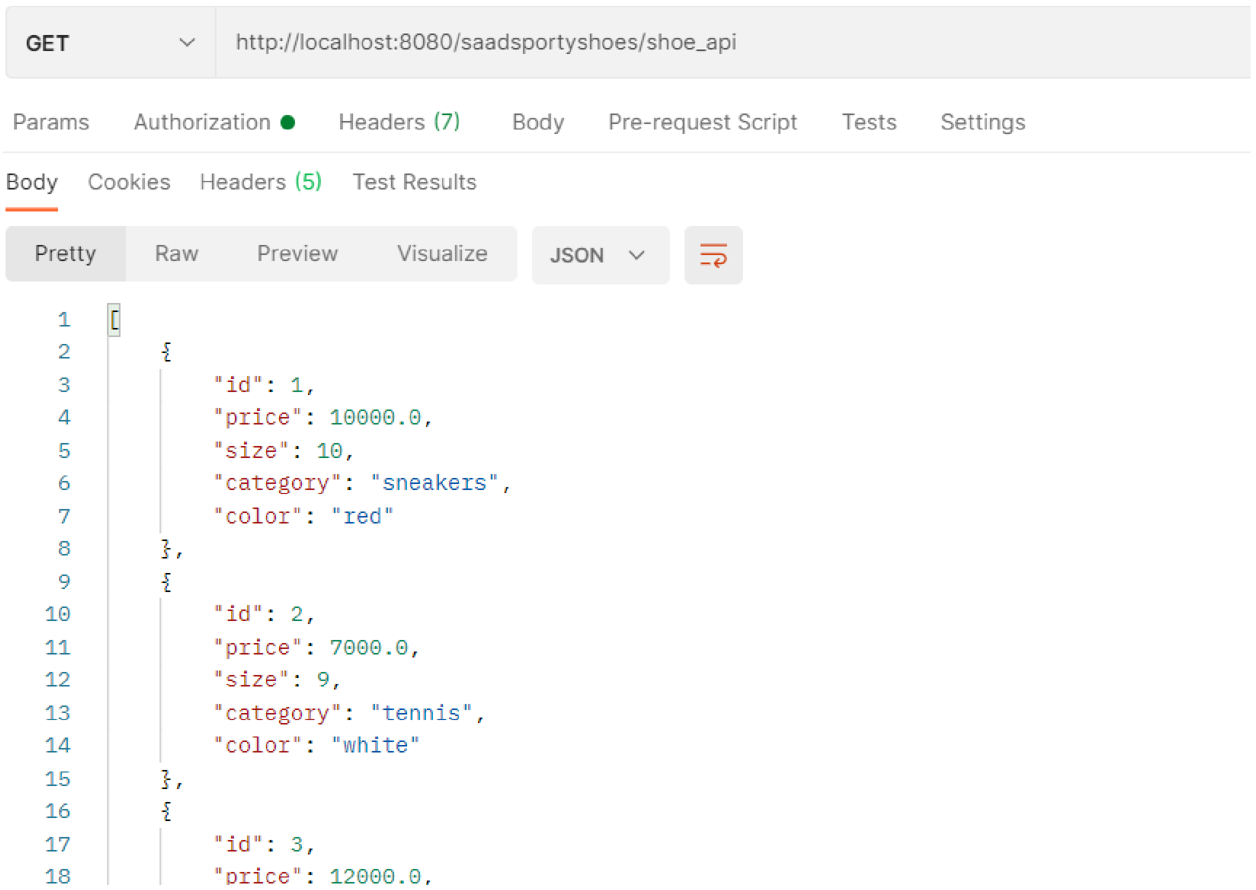
## Initial data entered into the table (put in data.sql file)

```
insert into UserTable(username, emailAddress, hasSignedUp)
values ('Saad','saad@gmail.com', TRUE);
insert into UserTable(username, emailAddress, hasSignedUp)
values ('Shrey','shrey@gmail.com', FALSE);
insert into UserTable(username, emailAddress, hasSignedUp)
values ('Bhavay','bhavay@outlook.com', TRUE);
insert into UserTable(username, emailAddress, hasSignedUp)
values ('Mike','mike@outlook.com.com', TRUE);

insert into ShoeTable(price, size, category, color) values
(10000, 10, 'sneakers', 'red');
insert into ShoeTable(price, size, category, color) values
(7000, 9, 'tennis', 'white');
insert into ShoeTable(price, size, category, color) values
(12000, 11, 'sneakers', 'red');
insert into ShoeTable(price, size, category, color) values
(7000, 7, 'running', 'blue');
insert into ShoeTable(price, size, category, color) values
(12000, 6, 'tennis', 'yellow');
insert into ShoeTable(price, size, category, color) values
(1000, 6, 'running', 'red');

insert into PurchaseTable(userId, shoeId, date) values (3, 1,
'2021-01-11');
```

```
insert into PurchaseTable(userId, shoeId, date) values (2, 2,
'2020-01-12');
insert into PurchaseTable(userId, shoeId, date) values (1, 1,
'2019-09-16');
insert into PurchaseTable(userId, shoeId, date) values (4, 3,
'2017-09-08');
insert into PurchaseTable(userId, shoeId, date) values (4, 2,
'2021-07-04');
insert into PurchaseTable(userId, shoeId, date) values (3, 3,
'2018-12-20');
insert into PurchaseTable(userId, shoeId, date) values (3, 2,
'2021-07-16');
```

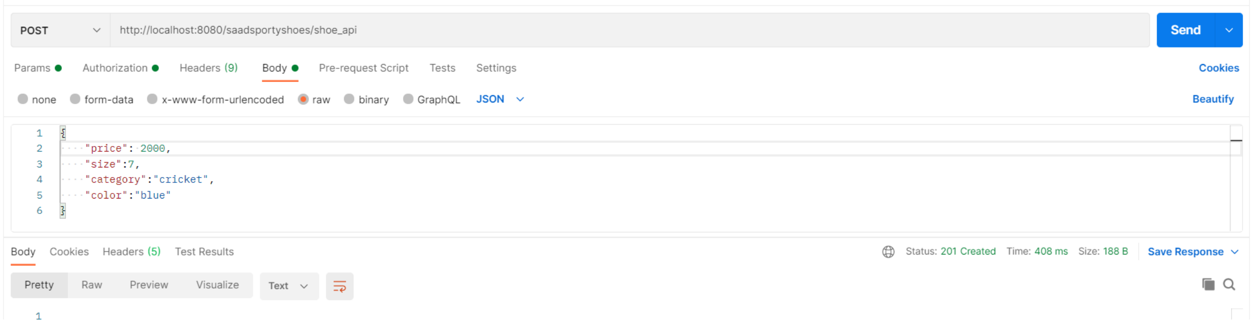
## Some working screenshots

1. **Show all shoes from the Shoe Table. Endpoint:** /saadsportyshoes/shoe_api. **Request type:** GET.



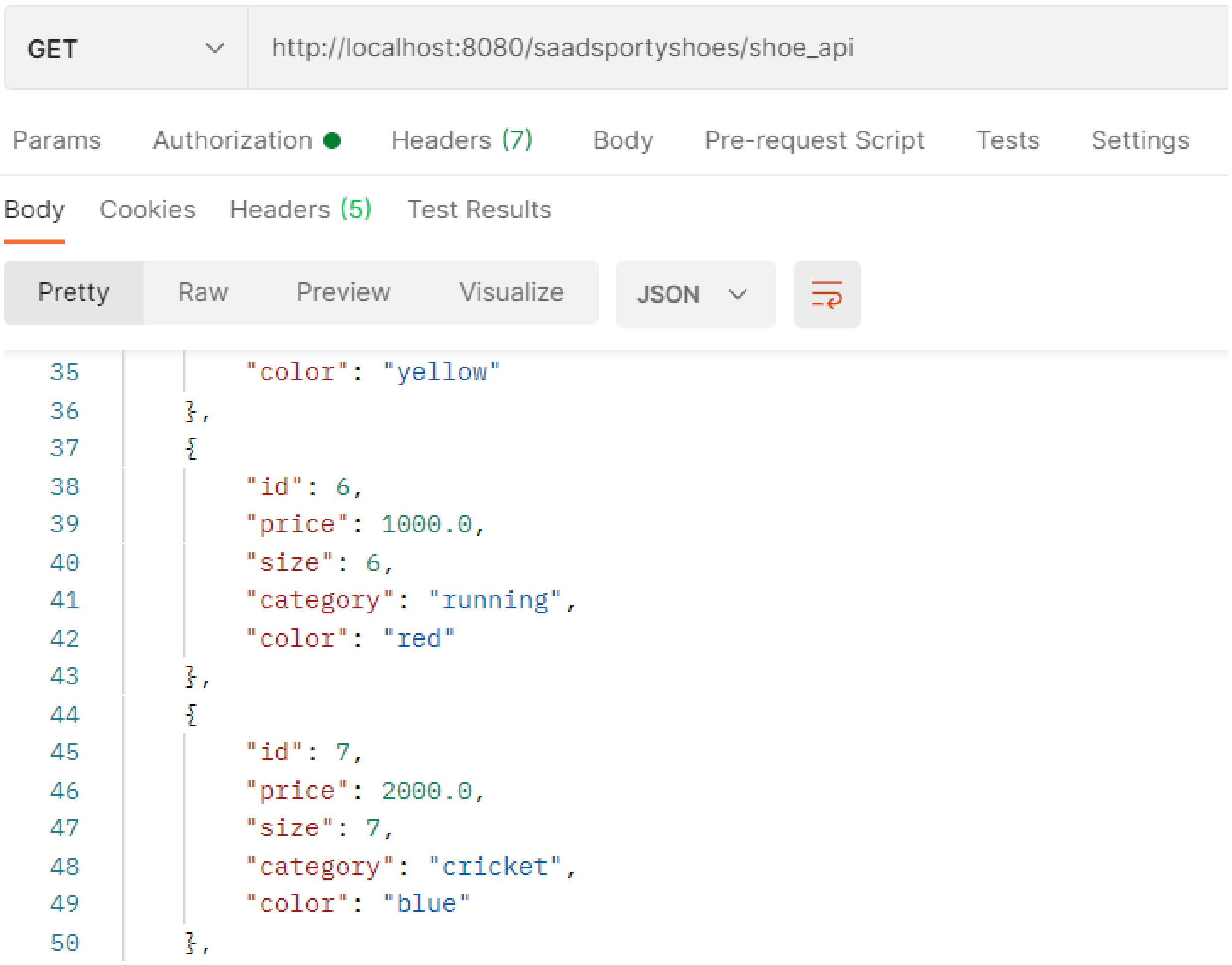
```
[
    {
        "id": 1,
        "price": 10000.0,
        "size": 10,
        "category": "sneakers",
        "color": "red"
    },
    {
        "id": 2,
        "price": 7000.0,
        "size": 9,
        "category": "tennis",
        "color": "white"
    },
    {
        "id": 3,
        "price": 12000.0,
```

2. **Insert a new shoe object into the shoe table. <u>Endpoint</u>:** /saadsportyshoes/shoe_api**. <u>Request type</u>:** POST.



```
{
    "price": 2000,
    "size":7,
    "category":"cricket",
    "color":"blue"
}
```

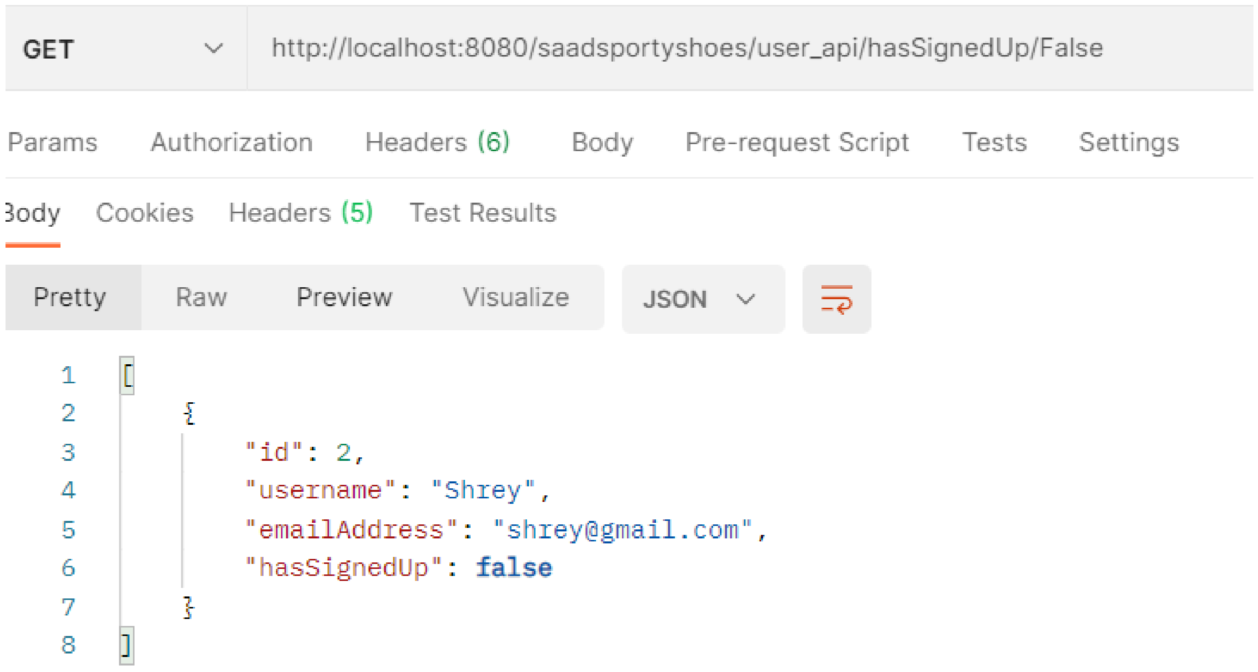
Status: 201 Created Time: 408 ms Size: 188 B



http://localhost:8080/saadsportyshoes/shoe_api

```
            "color": "yellow"
        },
        {
            "id": 6,
            "price": 1000.0,
            "size": 6,
            "category": "running",
            "color": "red"
        },
        {
            "id": 7,
            "price": 2000.0,
            "size": 7,
            "category": "cricket",
            "color": "blue"
        },
```

3. **Show users who have signed up. Endpoint:** /saadsportyshoes/user_api/hasSignedUp/{signedUpState}**. Request type:** GET. The signedUpState needs to be either True or False for the api to work.


GET http://localhost:8080/saadsportyshoes/user_api/hasSignedUp/False

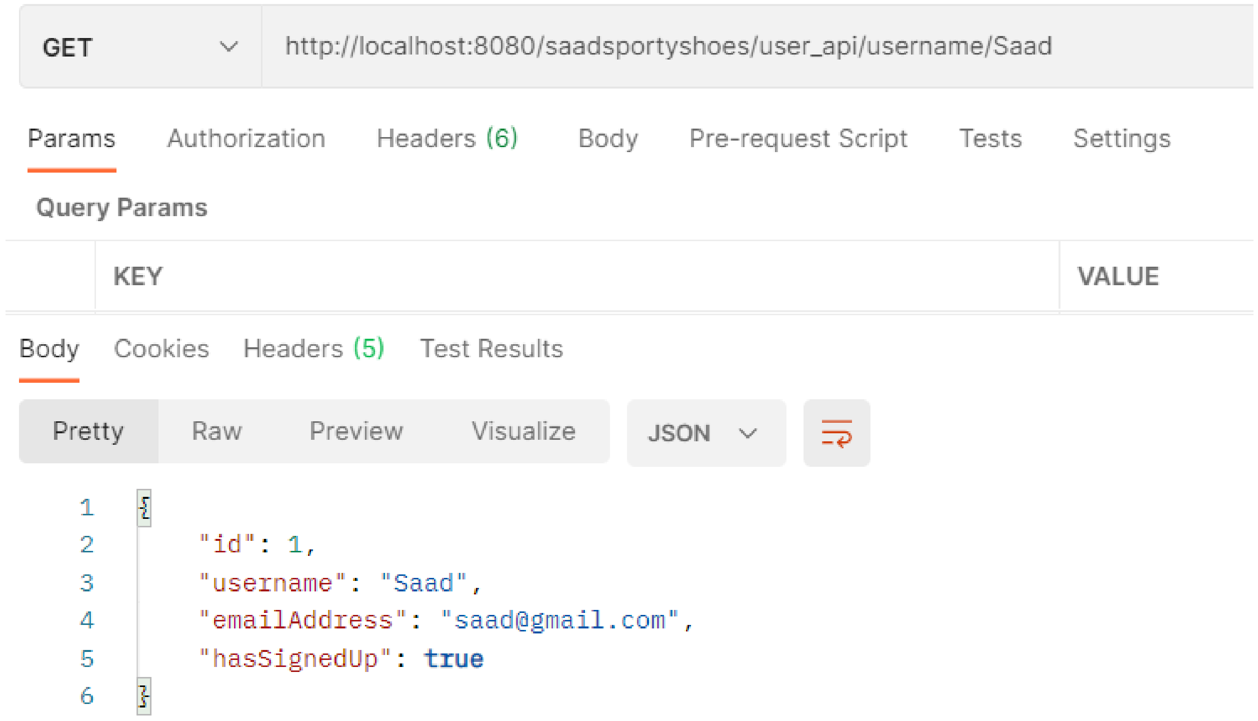
Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```json
[
    {
        "id": 2,
        "username": "Shrey",
        "emailAddress": "shrey@gmail.com",
        "hasSignedUp": false
    }
]
```

4. **Show users who have the username "Saad". Endpoint:** /saadsportyshoes/user_api/username/{username}**. Request type:** GET.


GET http://localhost:8080/saadsportyshoes/user_api/username/Saad

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Query Params

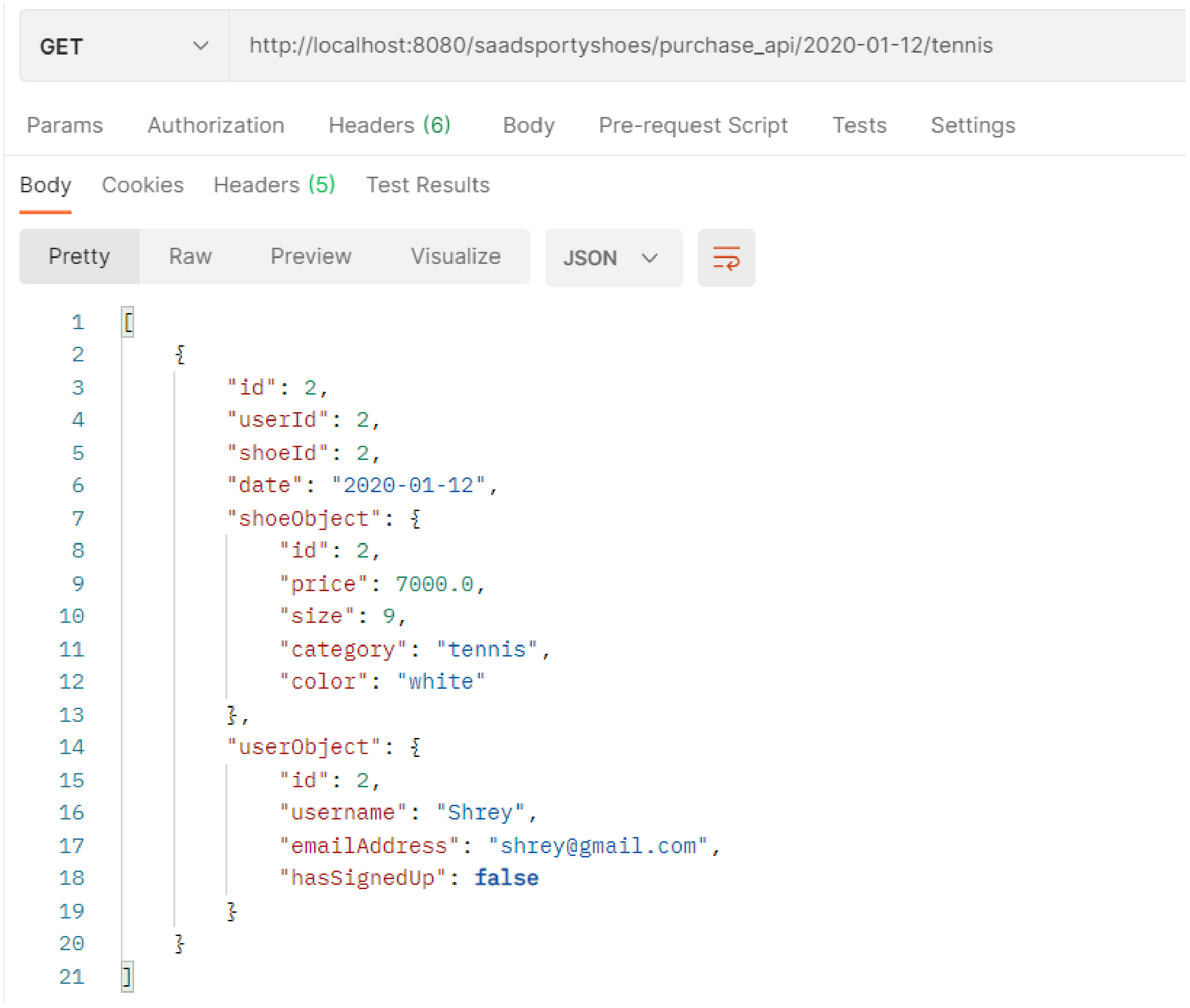
| KEY | VALUE |
|---|---|

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | Visualize | JSON

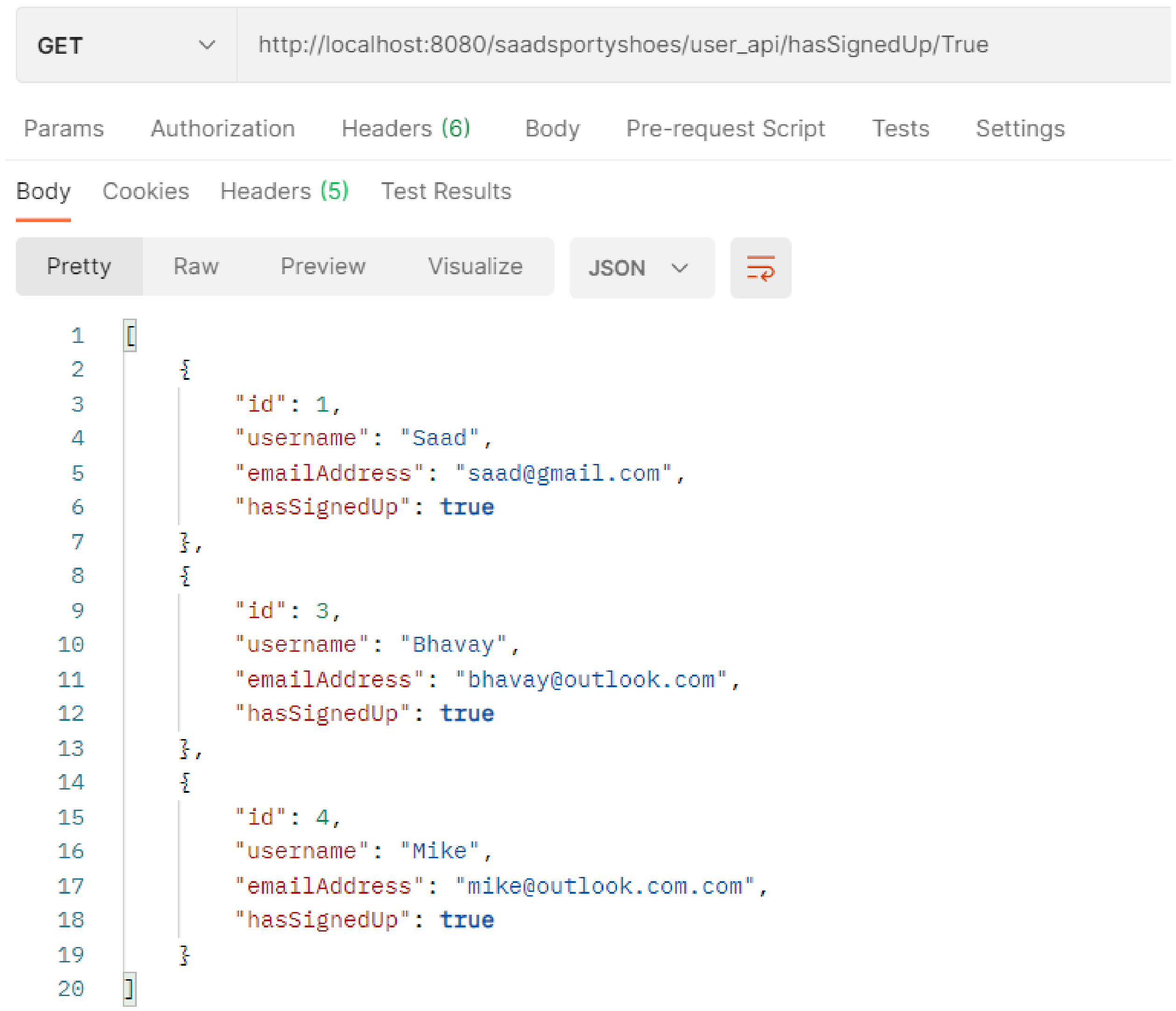
```json
{
    "id": 1,
    "username": "Saad",
    "emailAddress": "saad@gmail.com",
    "hasSignedUp": true
}
```

5. **Show purchases filtered by date and category. <u>Endpoint</u>:** /saadsportyshoes/purchase_api/{purchaseDate}/{category}**. <u>Request type</u>:** GET



```
http://localhost:8080/saadsportyshoes/purchase_api/2020-01-12/tennis
```

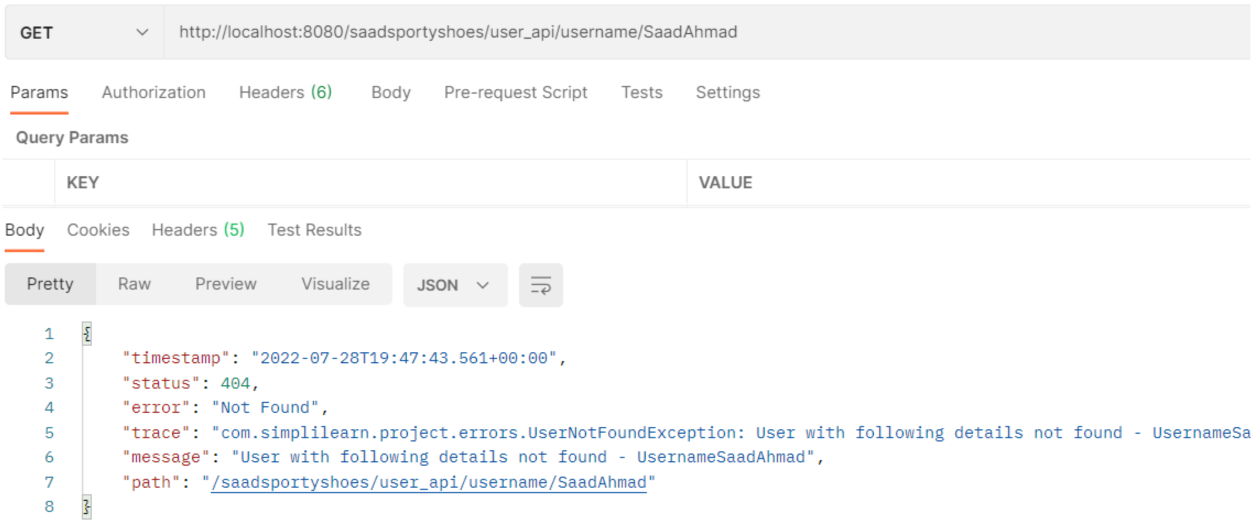
```
[
    {
        "id": 2,
        "userId": 2,
        "shoeId": 2,
        "date": "2020-01-12",
        "shoeObject": {
            "id": 2,
            "price": 7000.0,
            "size": 9,
            "category": "tennis",
            "color": "white"
        },
        "userObject": {
            "id": 2,
            "username": "Shrey",
            "emailAddress": "shrey@gmail.com",
            "hasSignedUp": false
        }
    }
]
```



```
http://localhost:8080/saadsportyshoes/user_api/hasSignedUp/True
```

```
[
    {
        "id": 1,
        "username": "Saad",
        "emailAddress": "saad@gmail.com",
        "hasSignedUp": true
    },
    {
        "id": 3,
        "username": "Bhavay",
        "emailAddress": "bhavay@outlook.com",
        "hasSignedUp": true
    },
    {
        "id": 4,
        "username": "Mike",
        "emailAddress": "mike@outlook.com.com",
        "hasSignedUp": true
    }
]
```

6. **In case of a resource not found (404) exception, I throw a custom exception with custom message. For example in the screenshot below, the “message” key in the response JSON body contains the error message thrown by the custom exception.**



http://localhost:8080/saadsportyshoes/user_api/username/SaadAhmad

```
{
    "timestamp": "2022-07-28T19:47:43.561+00:00",
    "status": 404,
    "error": "Not Found",
    "trace": "com.simplilearn.project.errors.UserNotFoundException: User with following details not found - UsernameSa
    "message": "User with following details not found - UsernameSaadAhmad",
    "path": "/saadsportyshoes/user_api/username/SaadAhmad"
}
```