

START2IMPACT'S FULL STACK DEVELOPMENT MASTER

“Node JS - LookBook”



Github repo: https://github.com/diegoddie/S2I_FullStack_NodeJS

Diego Lauricella

TABLE OF CONTENTS

0 1 **Introduction**

0 2 **Tech Stack**

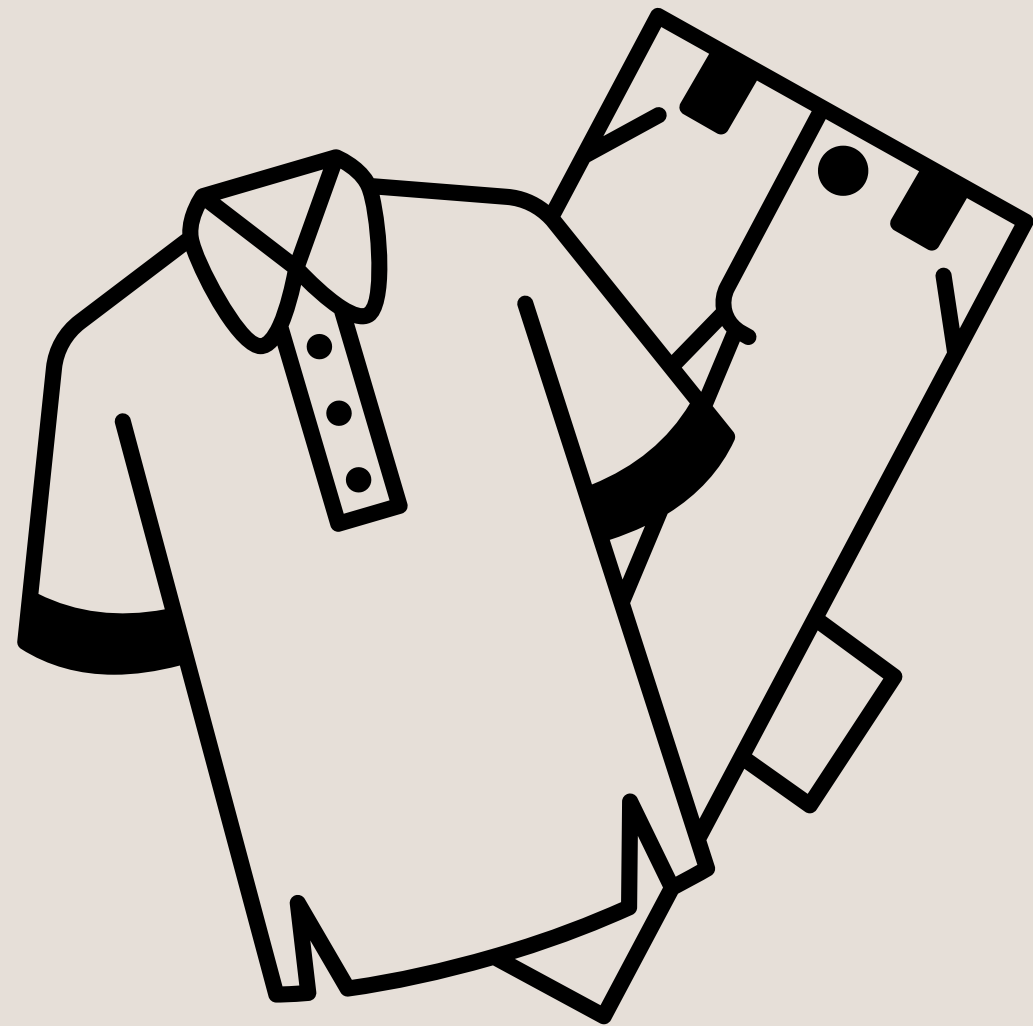
0 3 **Quick Start**

0 4 **API Documentation**

0 5 **Sanitization and Validation**

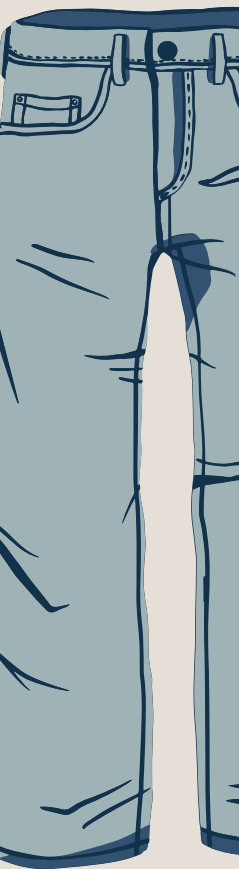
0 6 **Testing**

0 7 **Contacts**



01 - INTRODUCTION

At LookBook, we believe in the power of **sustainability** and **circular economy**. Our goal is to promote the reuse and exchange of pre-loved fashion items. By connecting users and second-hand products in a meaningful way, we contribute to **reduce waste** and embracing a more eco-friendly lifestyle!



02 - TECH STACK

N O D E . J S

A runtime environment for executing JavaScript code on the server-side.

E X P R E S S . J S

A web application framework for building robust and efficient web applications.

E X P R E S S V A L I D A T O R

A set of express.js middlewares and sanitizer functions.

N O D E M O N

A tool that helps develop Node.js based applications by automatically restarting the node application when file changes are detected.

M O N G O D B

A NoSQL database that stores data in flexible, JSON-like documents.

M O N G O O S E

A library for MongoDB and Node.js.

M O N G O D B M E M O R Y S E R V E R

In-memory MongoDB server for testing purposes.

J E S T

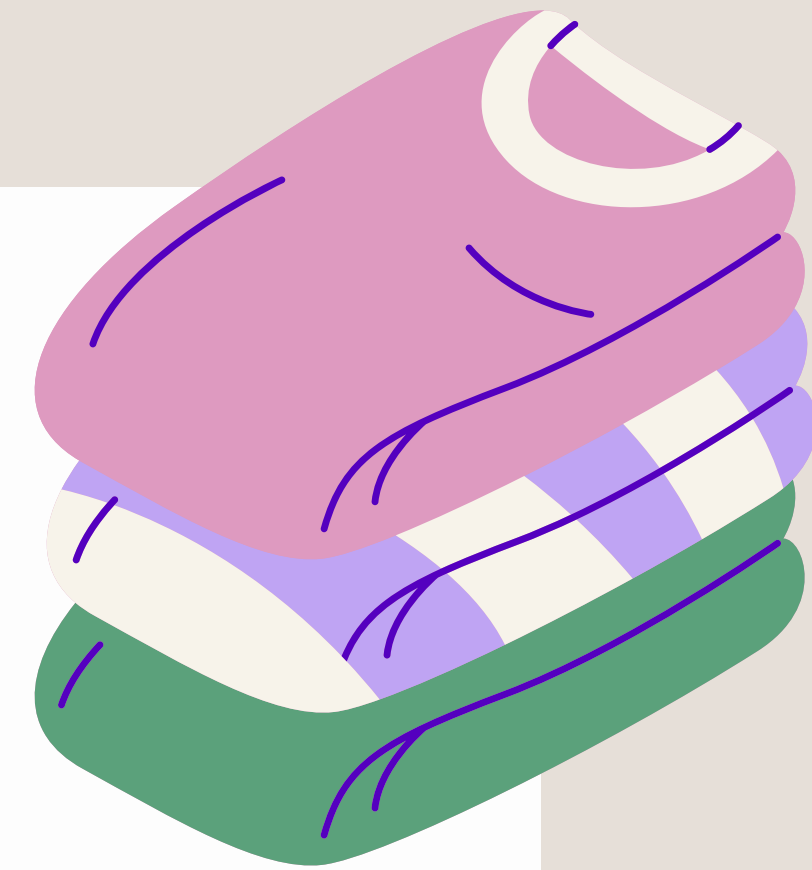
A JavaScript testing framework.

S U P E R T E S T

A library for testing HTTP assertions.

03 - QUICK START

1. Clone the repo
2. Install Dependencies
3. Configure your MongoDB Database via <https://www.mongodb.com/atlas/database>
4. Create a .env file and add your MONGODB_URI
5. npm run dev
6. Use Postman to try all the APIs (section 04 - API Documentation)
7. View Your Data in MongoDB Atlas
8. Open a new terminal and run npm test



04 - API DOCUMENTATION

USER-MODEL

```
const userSchema = new mongoose.Schema({  
  firstName: {  
    type: String,  
    required: true,  
  },  
  lastName: {  
    type: String,  
    required: true,  
  },  
  email: {  
    type: String,  
    required: true,  
    unique: true,  
  },  
});
```

04 - API DOCUMENTATION

GET /users ➡ Returns a list of all users.

```
[
  {
    "_id": "653e9c139f0eda7643e1d537",
    "firstName": "Lebron",
    "lastName": "James",
    "email": "lebron@james.com",
    "__v": 0
  },
  {
    "_id": "654213c35a5d03179fb25afe",
    "firstName": "John",
    "lastName": "Doe",
    "email": "johndoe@example.com",
    "__v": 0
  }
]
```

GET /users/:id ➡ Returns a specific user

GET localhost:3000/users/654214b8d91e102b7ff4fba1

Body Cookies Headers (7) Test Results

Pretty

Raw

Preview

Visualize

JSON



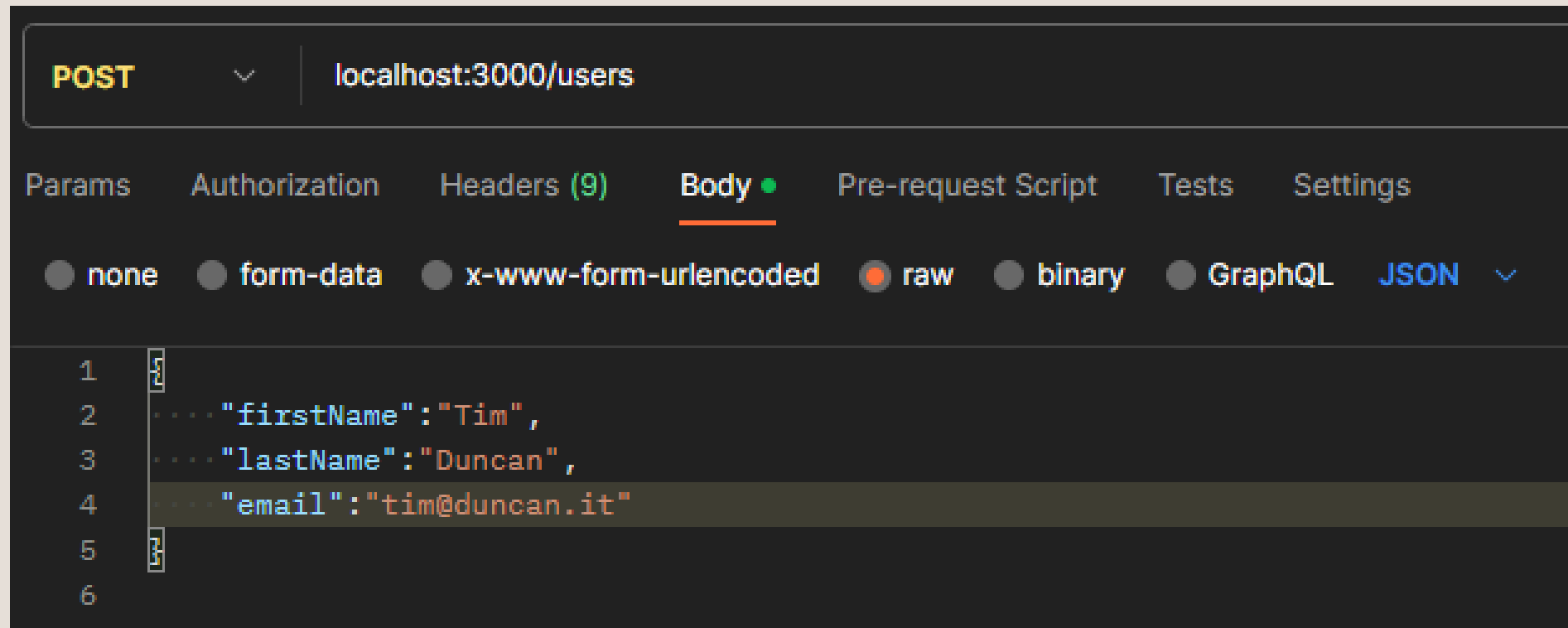
```
1 [
2   "_id": "654214b8d91e102b7ff4fba1",
3   "firstName": "Start",
4   "lastName": "2Impact",
5   "email": "start@lauricell7a.it",
6   "__v": 0
7 ]
```

04 - API DOCUMENTATION

POST /users ➡ Creates a new user

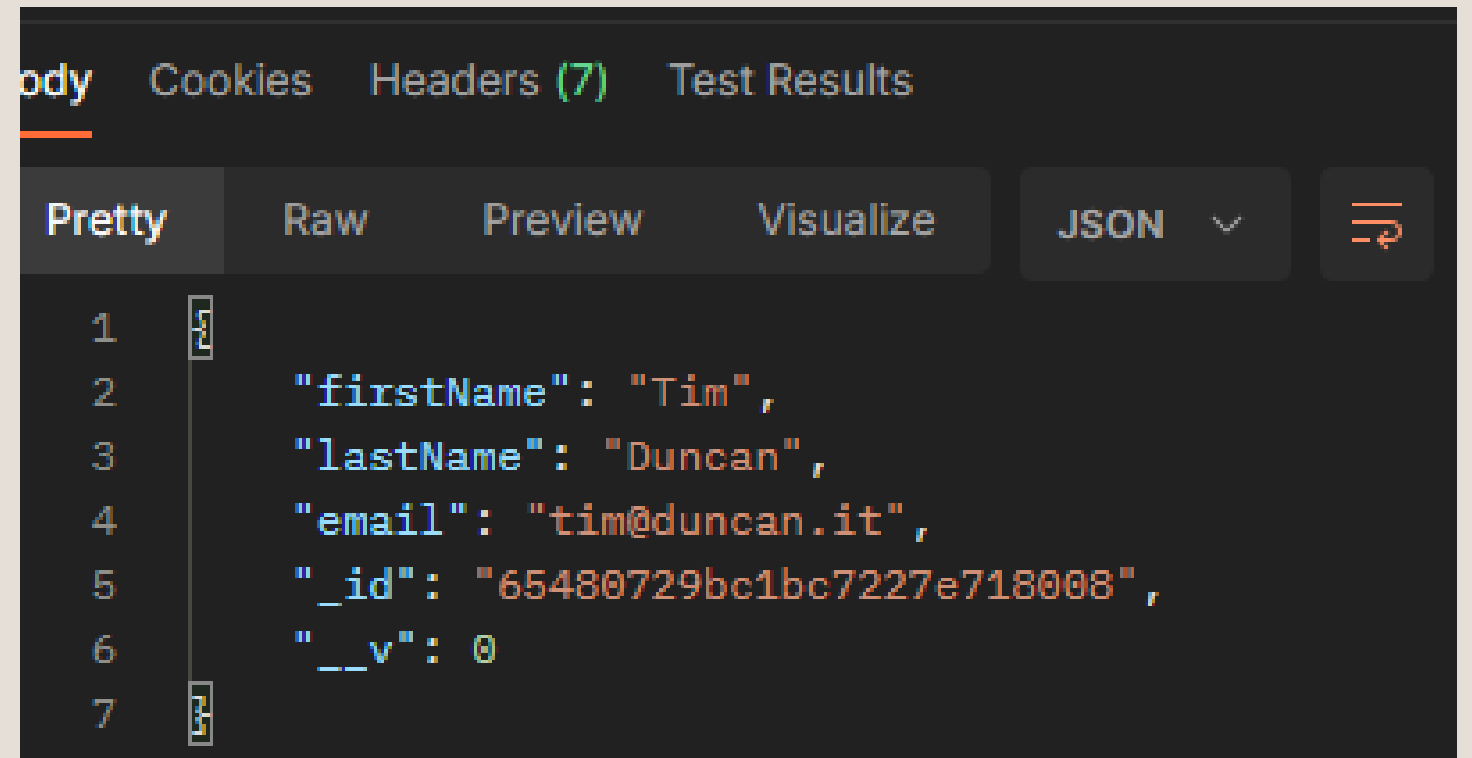
Request Body Fields:

- **firstName**: The first name of the user (required).
- **lastName**: The last name of the user (required).
- **email**: The email address of the user (required). It must be a valid email format and not already in use.



```
POST localhost:3000/users

{
  "firstName": "Tim",
  "lastName": "Duncan",
  "email": "tim@duncan.it"
}
```



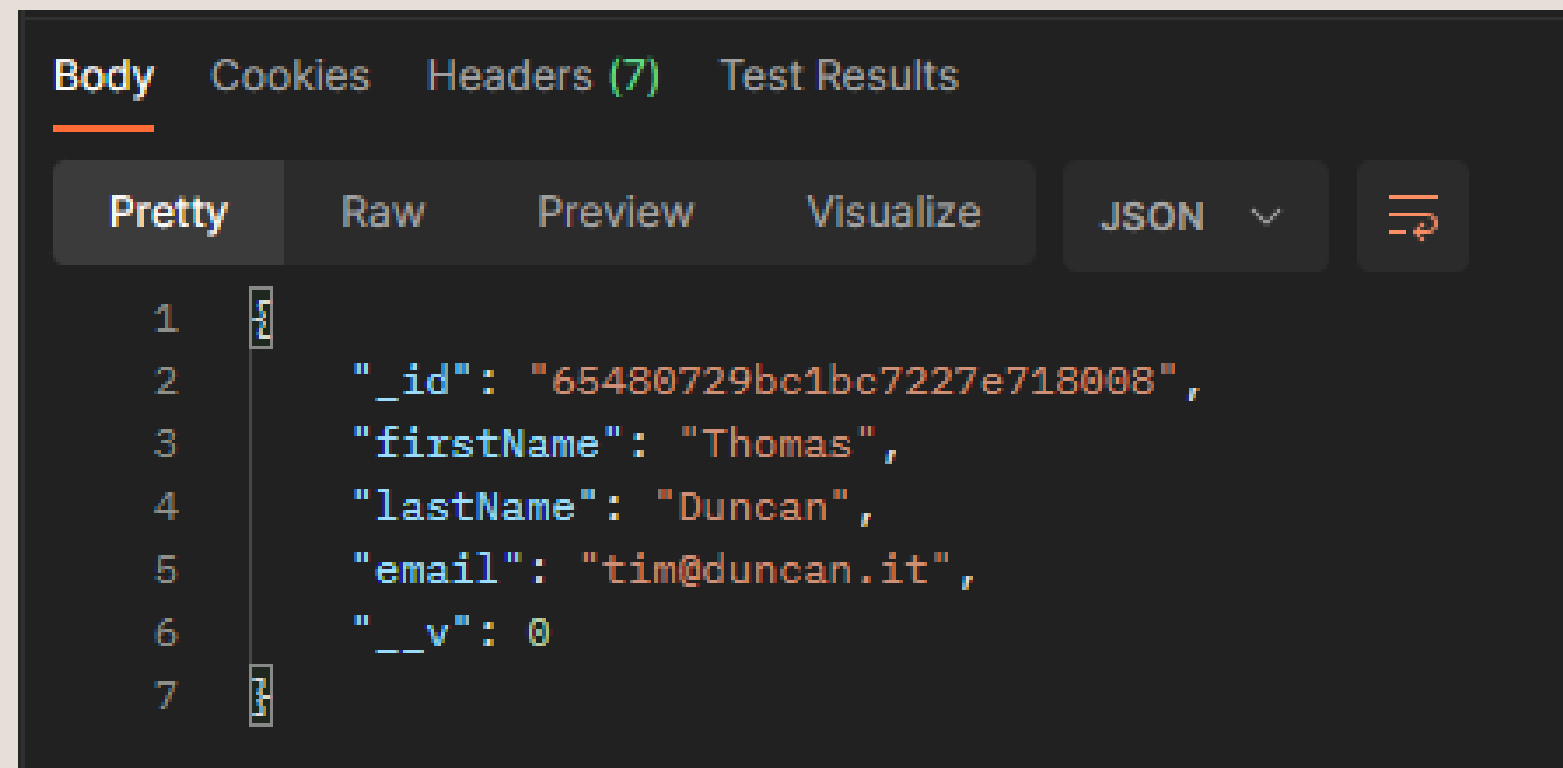
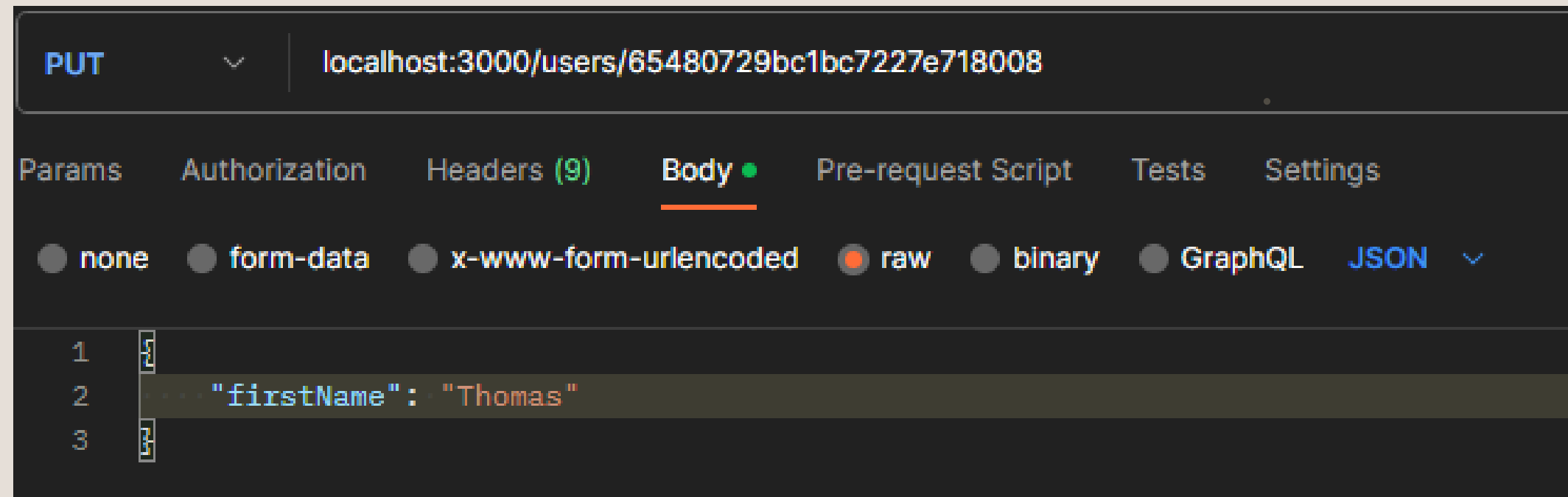
```
body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

{
  "firstName": "Tim",
  "lastName": "Duncan",
  "email": "tim@duncan.it",
  "_id": "65480729bc1bc7227e718008",
  "__v": 0
}
```

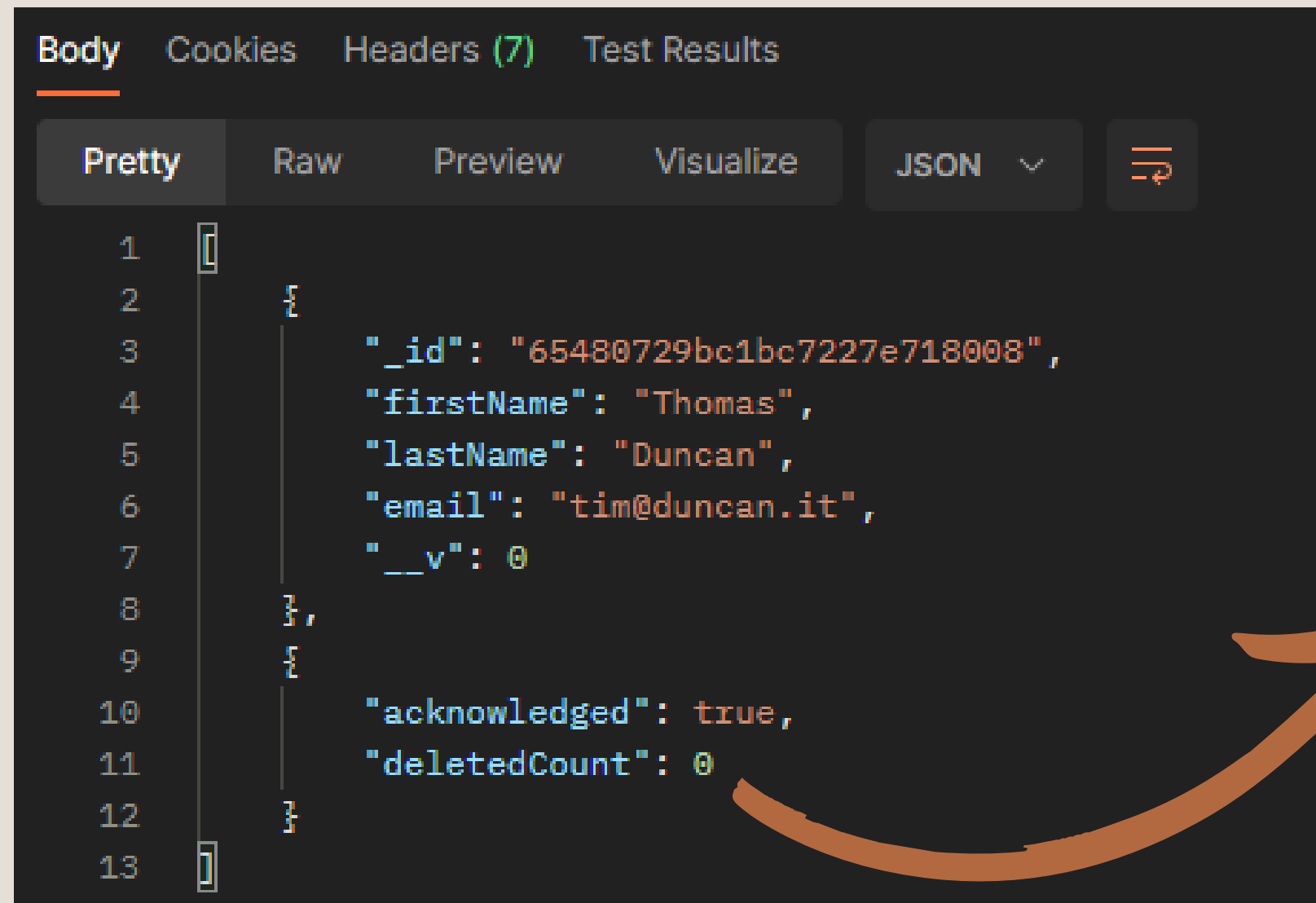
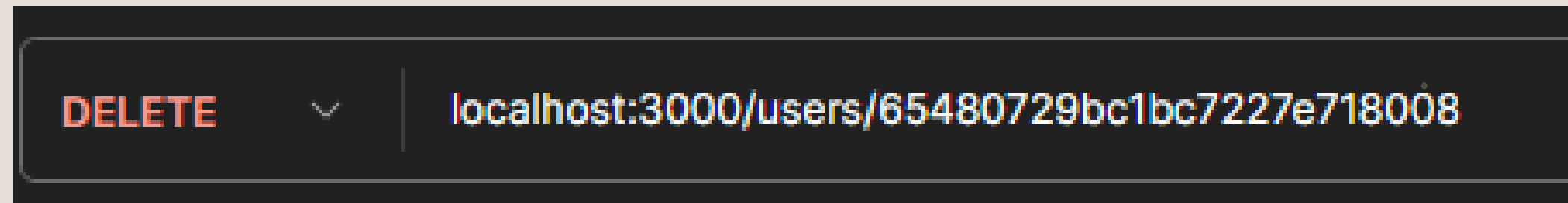

04 - API DOCUMENTATION

PUT `/users/:id` ➡ Updates an existing user based on the ID.



04 - API DOCUMENTATION

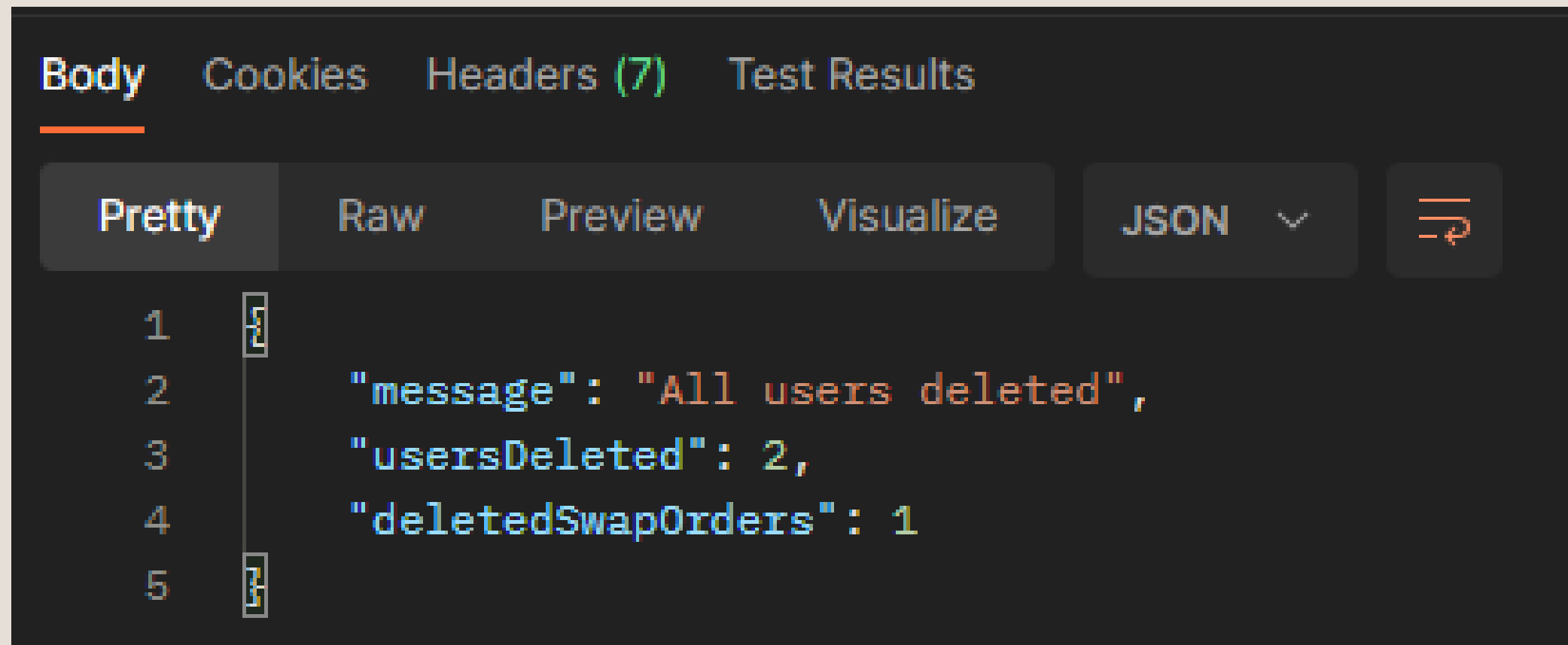
DELETE `/users/:id` ➡ Deletes a specific user



When a user is deleted, all the associated swap orders are automatically removed.

04 - API DOCUMENTATION

DELETE /users ➡ Deletes all users



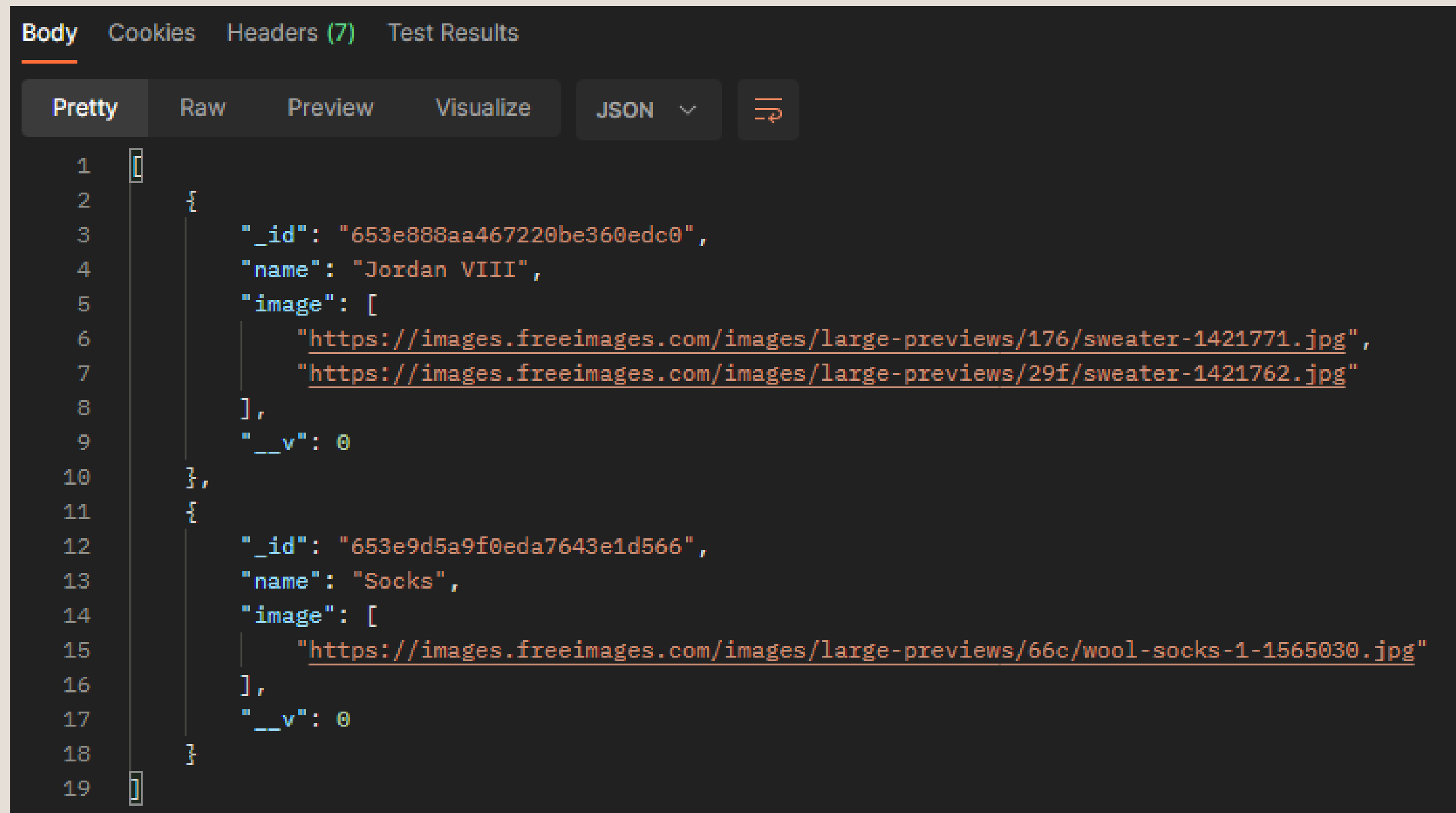
04 - API DOCUMENTATION

PRODUCT-MODEL

```
const productSchema = new mongoose.Schema({  
  name: {  
    type: String,  
    required: true  
  },  
  image: [{  
    type: String,  
    required: true  
  }],  
});
```

04 - API DOCUMENTATION

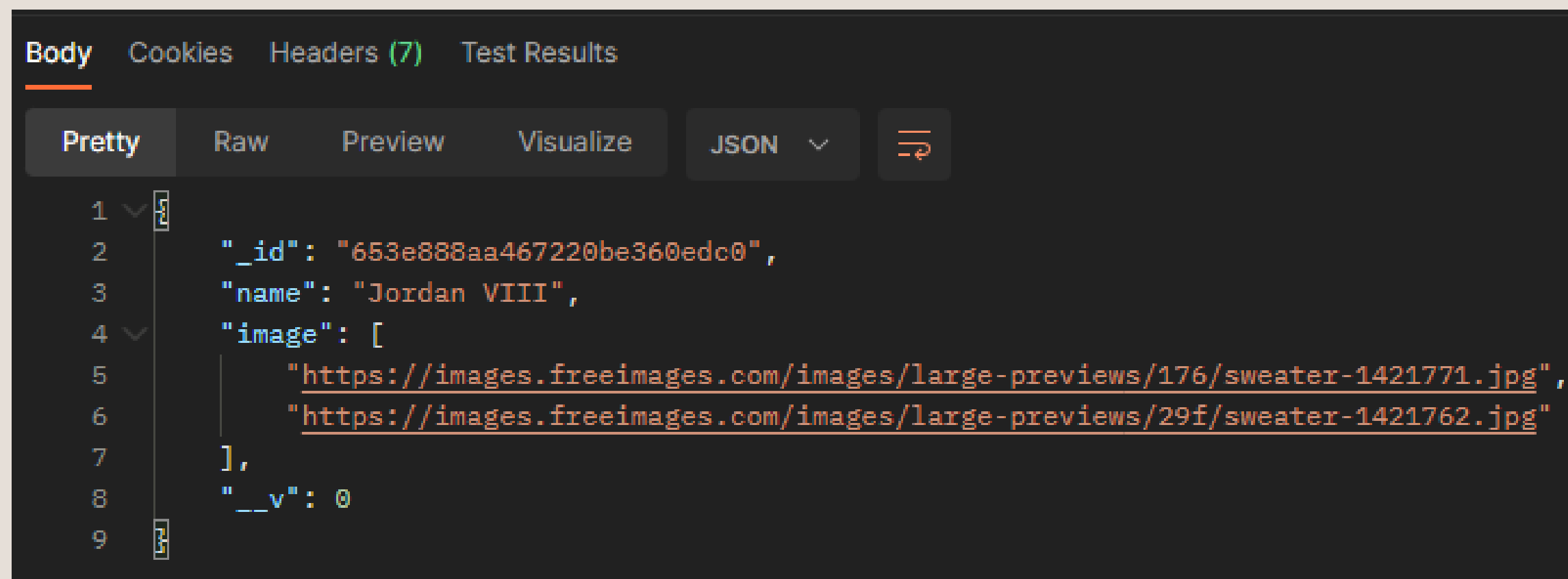
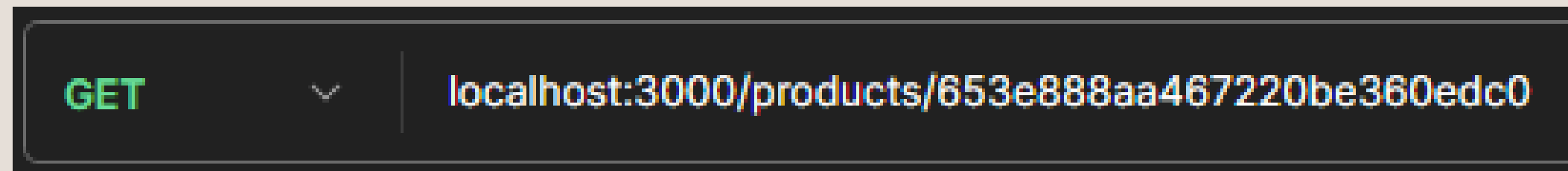
GET /products ➡ Returns a list of all products.



```
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON ↕
1 []
2 {
3   "_id": "653e888aa467220be360edc0",
4   "name": "Jordan VIII",
5   "image": [
6     "https://images.freeimages.com/images/large-previews/176/sweater-1421771.jpg",
7     "https://images.freeimages.com/images/large-previews/29f/sweater-1421762.jpg"
8   ],
9   "__v": 0
10 },
11 {
12   "_id": "653e9d5a9f0eda7643e1d566",
13   "name": "Socks",
14   "image": [
15     "https://images.freeimages.com/images/large-previews/66c/wool-socks-1-1565030.jpg"
16   ],
17   "__v": 0
18 }
19 ]
```

04 - API DOCUMENTATION

GET /products/:id ➡ Returns a specific product.

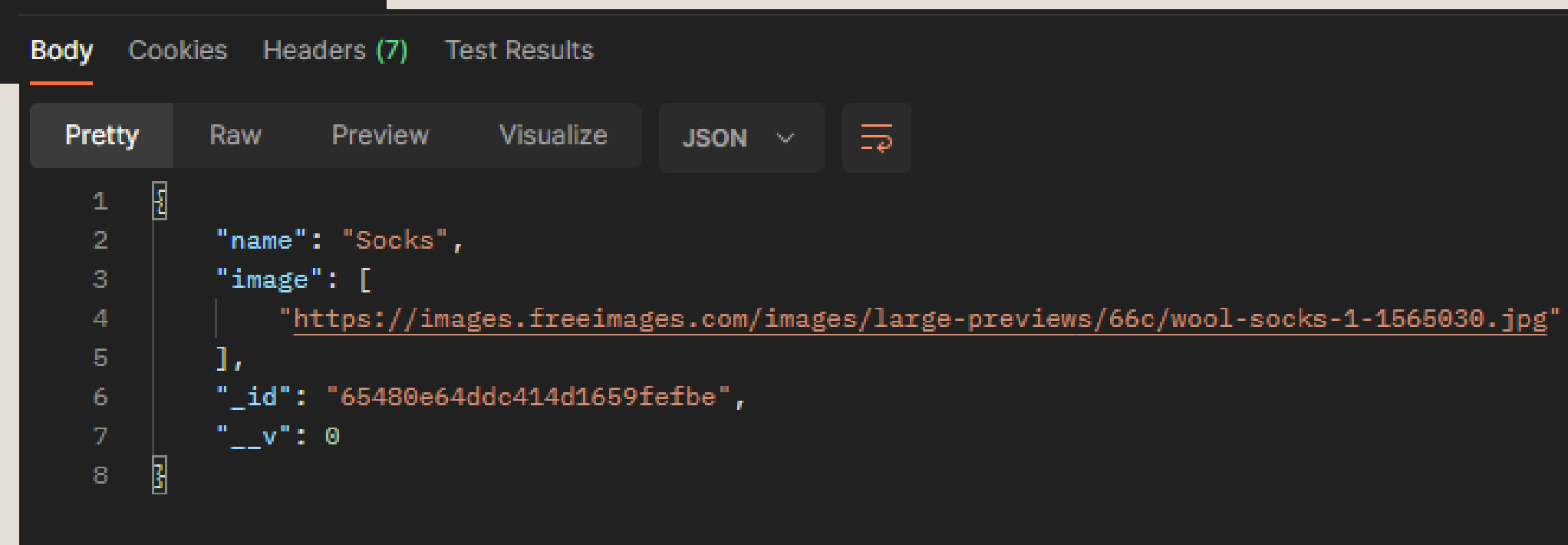
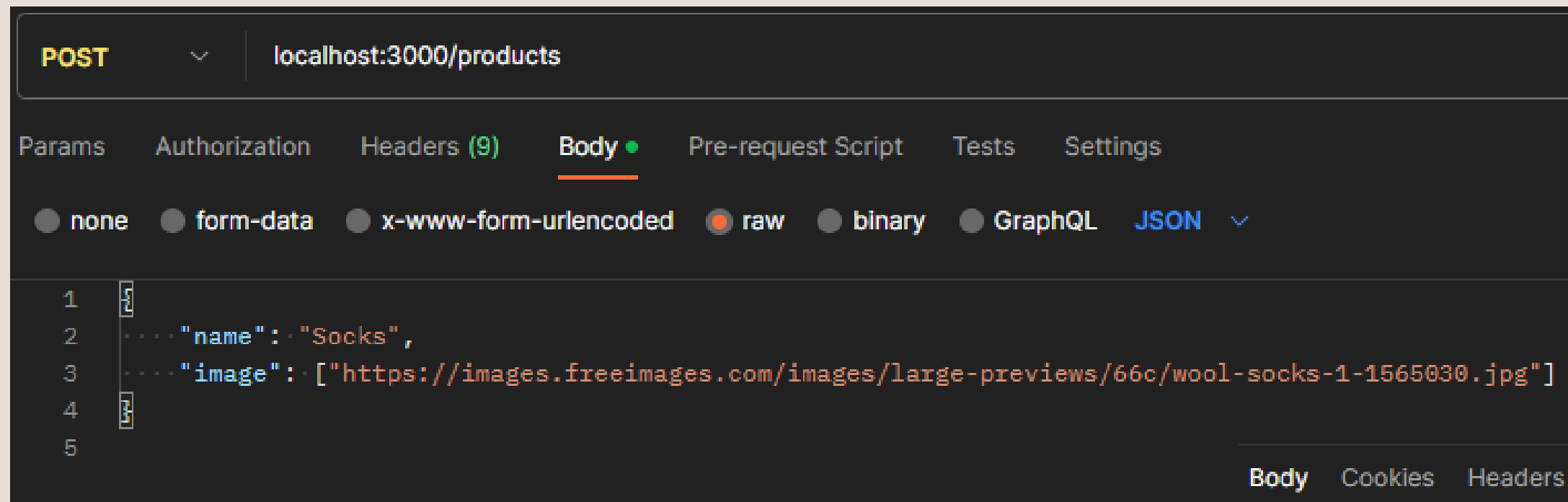


04 - API DOCUMENTATION

POST /products ➡ Creates a new product

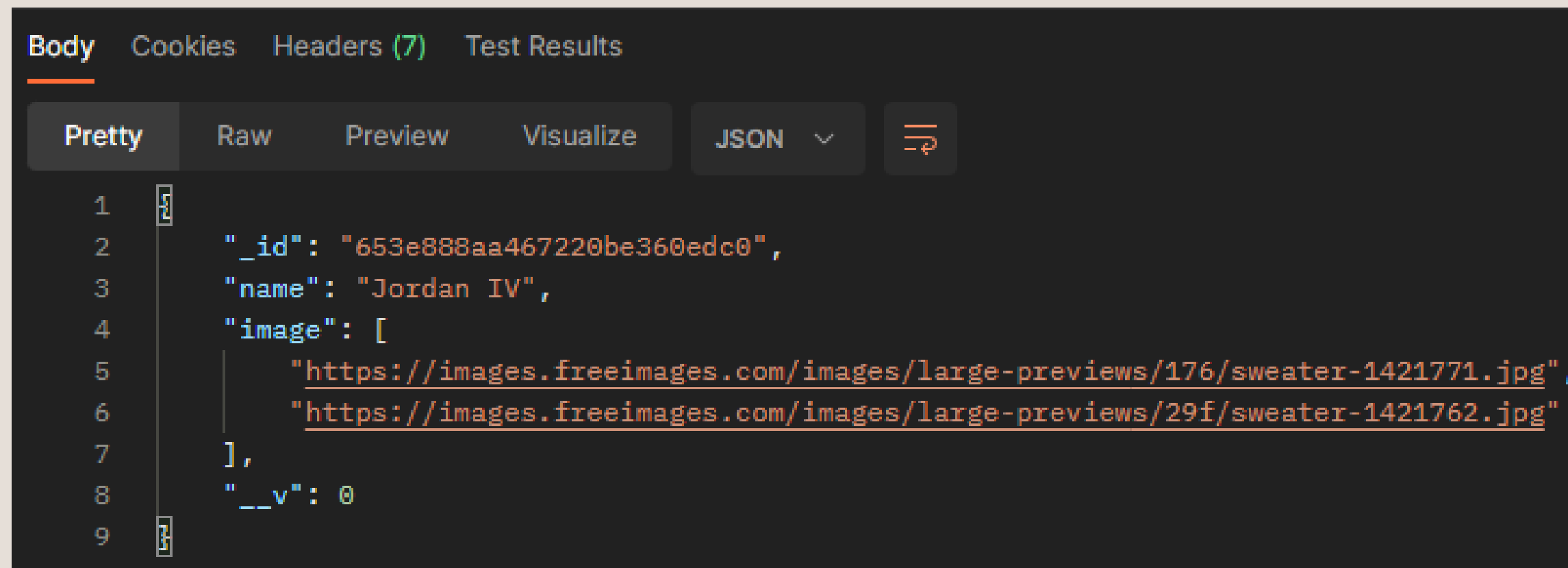
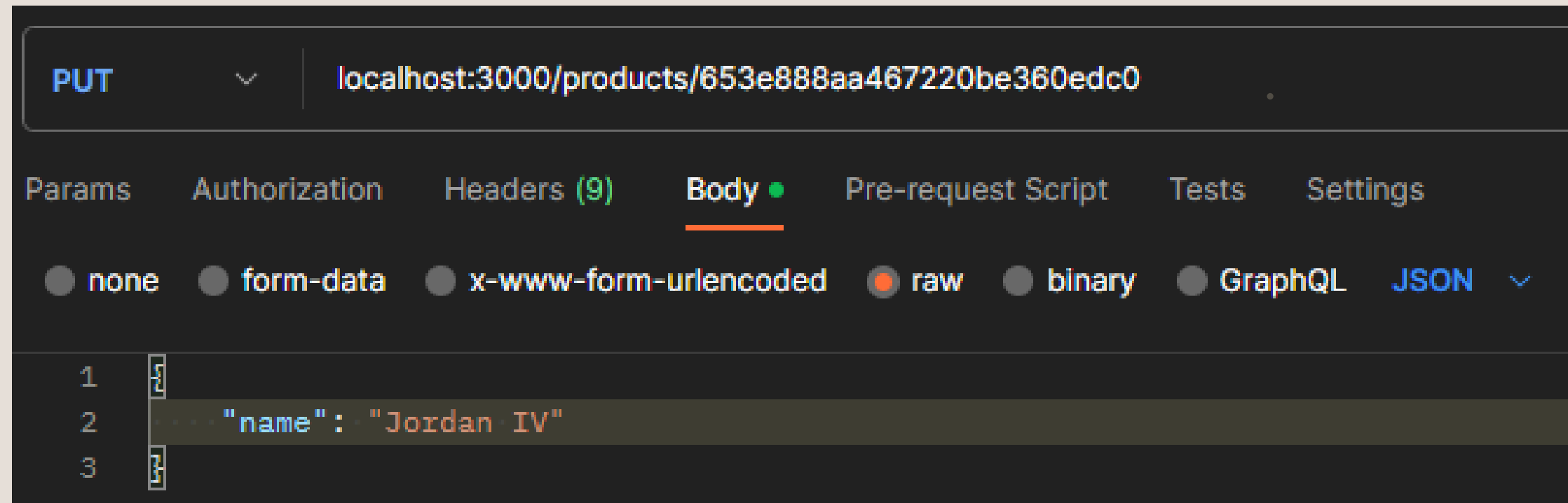
Request Body Fields (both required):

- **name:** The name of the product.
- **image:** An array of URLs for the images (1 URL minimum). Each URL must be a valid HTTP, HTTPS, or FTP link.



04 - API DOCUMENTATION

PUT `/products/:id` ➡ Updates an existing product based on the ID.



04 - API DOCUMENTATION

DELETE `/products/:id` ➡ Deletes a specific product

DELETE

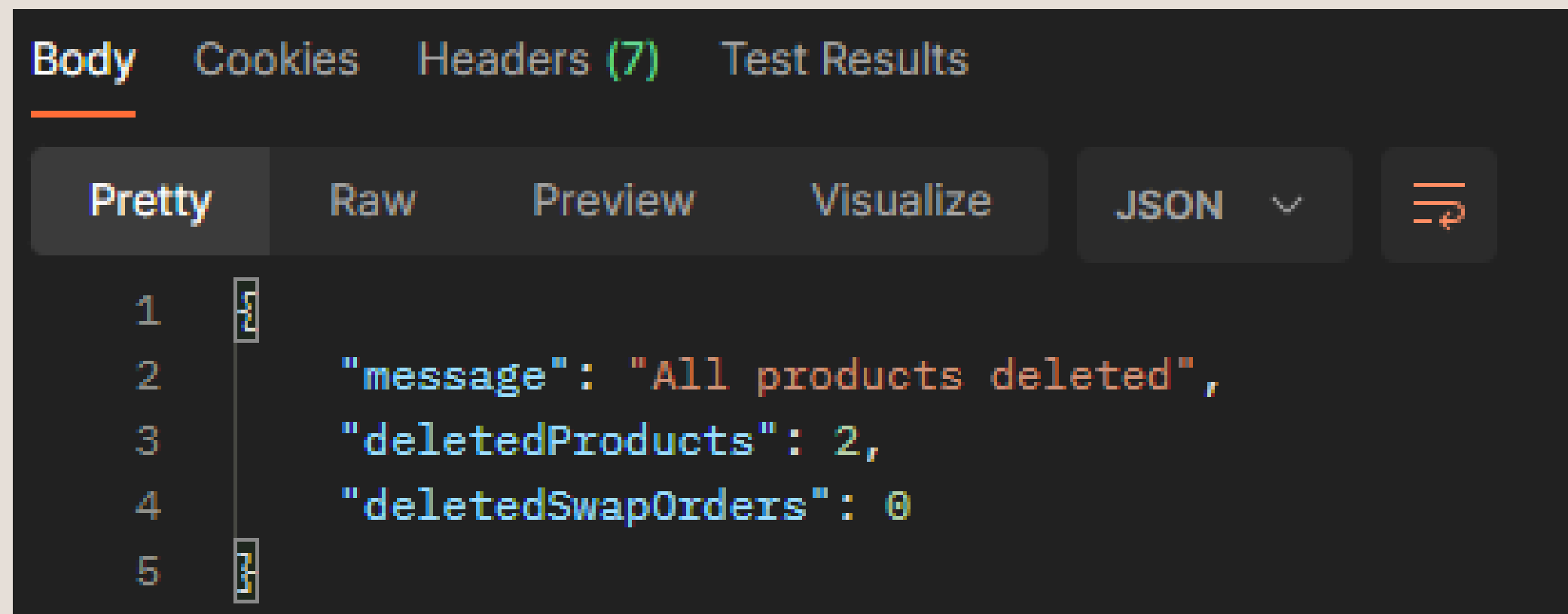
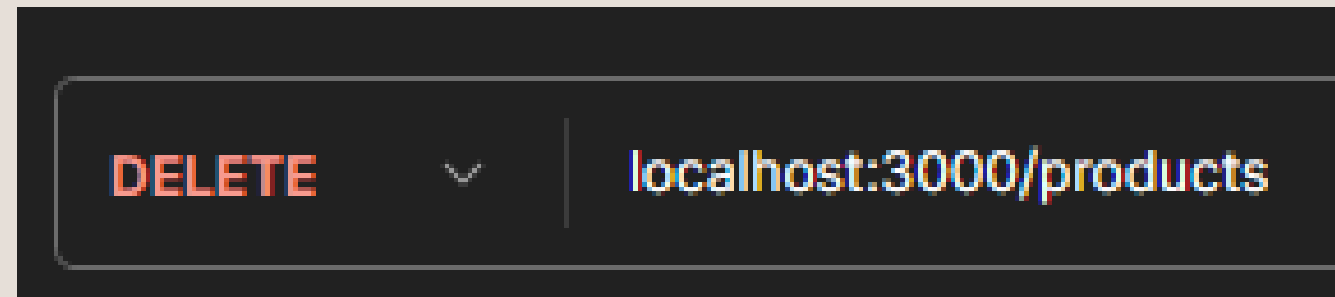
`localhost:3000/products/653e888aa467220be360edc0`

```
Body  Cookies  Headers (7)  Test Results
Pretty  Raw  Preview  Visualize  JSON  [icon]
1  [
2    {
3      "_id": "653e888aa467220be360edc0",
4      "name": "Jordan IV",
5      "image": [
6        "https://images.freeimages.com/images/large-previews/176/sweater-1421771.jpg",
7        "https://images.freeimages.com/images/large-previews/29f/sweater-1421762.jpg"
8      ],
9      "__v": 0
10   },
11   {
12     "acknowledged": true,
13     "deletedCount": 0
14   }
15 ]
```

When a product is deleted, all the associated swap orders are automatically removed.

04 - API DOCUMENTATION

DELETE /products ➡ Deletes all products



04 - API DOCUMENTATION

SWAPORDER-MODEL

```
const swapOrderSchema = new mongoose.Schema({  
  // Define the products field as an array of ObjectIds, referencing the 'Product' model, and it's required  
  products: [{  
    type: mongoose.Schema.Types.ObjectId,  
    ref: 'Product',  
    required: true,  
  }],  
  // Define the users field as an array of ObjectIds, referencing the 'User' model, and it's required  
  users: [{  
    type: mongoose.Schema.Types.ObjectId,  
    ref: 'User',  
    required: true  
  }],  
  // Define the insertionDate field as a Date with a default value of the current date and time  
  insertionDate: {  
    type: Date,  
    default: Date.now  
  },  
});
```

An array of at least 2 product IDs is required.

An array of exactly 2 user IDs is required.

04 - API DOCUMENTATION

GET /swap-orders ➡ Returns a list of all swap orders.

```
{
  "_id": "6548a9f3ddc414d1659fefda",
  "products": [
    {
      "_id": "6548a91fddc414d1659fefc6",
      "name": "Socks",
      "image": [
        "https://images.freeimages.com/images/large-previews/66c/wool-socks-1-1565030.jpg"
      ],
      "__v": 0
    },
    {
      "_id": "6548a92cddc414d1659fefc8",
      "name": "Hoodie",
      "image": [
        "https://images.freeimages.com/images/large-previews/66c/wool-hoodie-1-1565030.jpg"
      ],
      "__v": 0
    }
  ],
  "users": [
    {
      "_id": "6548a9d5ddc414d1659fed5",
      "firstName": "Tim",
      "lastName": "Duncan",
      "email": "tim@duncan.it",
      "__v": 0
    },
    {
      "_id": "6548a9e4ddc414d1659fed7",
      "firstName": "Michael",
      "lastName": "Jordan",
      "email": "mj@23.it",
      "__v": 0
    }
  ],
  "insertionDate": "2023-11-06T08:55:15.698Z",
  "__v": 0
}
```

04 - API DOCUMENTATION

GET /swap-orders/:id ➡ Returns a specific swap order based on the ID.

GET

localhost:3000/swap-orders/6548a9f3ddc414d1659fefda

```
1  {
2    "_id": "6548a9f3ddc414d1659fefda",
3    "products": [
4      {
5        "_id": "6548a91fddc414d1659fefc6",
6        "name": "Socks",
7        "image": [
8          "https://images.freeimages.com/images/large-previews/66c/wool-socks-1-1565030.jpg"
9        ],
10       "__v": 0
11     },
12     {
13       "_id": "6548a92cddc414d1659fefc8",
14       "name": "Hoodie",
15       "image": [
16         "https://images.freeimages.com/images/large-previews/66c/wool-hoodie-1-1565030.jpg"
17       ],
18       "__v": 0
19     }
20   ],
21   "users": [
22     {
23       "_id": "6548a9d5ddc414d1659fefd5",
24       "firstName": "Tim",
25       "lastName": "Duncan",
26       "email": "tim@duncan.it",
27       "__v": 0
28     },
29     {
30       "_id": "6548a9e4ddc414d1659fefd7",
31       "firstName": "Michael",
32       "lastName": "Jordan",
33       "email": "mj@23.it",
34       "__v": 0
35     }
36   ],
37   "insertionDate": "2023-11-06T08:55:15.698Z",
38   "__v": 0
39 }
```

04 - API DOCUMENTATION

GET /swap-orders/user/:id ➡ Returns swap orders associated with a specific user.

GET

localhost:3000/swap-orders/user/6548a9e4ddc414d1659fef7

```
1  {
2    "_id": "6548a9f3ddc414d1659fefda",
3    "products": [
4      {
5        "_id": "6548a91fddc414d1659fefc6",
6        "name": "Socks",
7        "image": [
8          "https://images.freeimages.com/images/large-previews/66c/wool-socks-1-1565030.jpg"
9        ],
10       "__v": 0
11     },
12     {
13       "_id": "6548a92cddc414d1659fefc8",
14       "name": "Hoodie",
15       "image": [
16         "https://images.freeimages.com/images/large-previews/66c/wool-hoodie-1-1565030.jpg"
17       ],
18       "__v": 0
19     }
20   ],
21   "users": [
22     {
23       "_id": "6548a9d5ddc414d1659fef7",
24       "firstName": "Tim",
25       "lastName": "Duncan",
26       "email": "tim@duncan.it",
27       "__v": 0
28     },
29     {
30       "_id": "6548a9e4ddc414d1659fef7",
31       "firstName": "Michael",
32       "lastName": "Jordan",
33       "email": "mj@23.it",
34       "__v": 0
35     }
36   ],
37   "insertionDate": "2023-11-06T08:55:15.698Z",
38   "__v": 0
39 }
```

04 - API DOCUMENTATION

GET /swap-orders/product/:id ➡ Returns swap orders associated with a specific product.

GET

localhost:3000/swap-orders/product/6548a92cddc414d1659fefc8

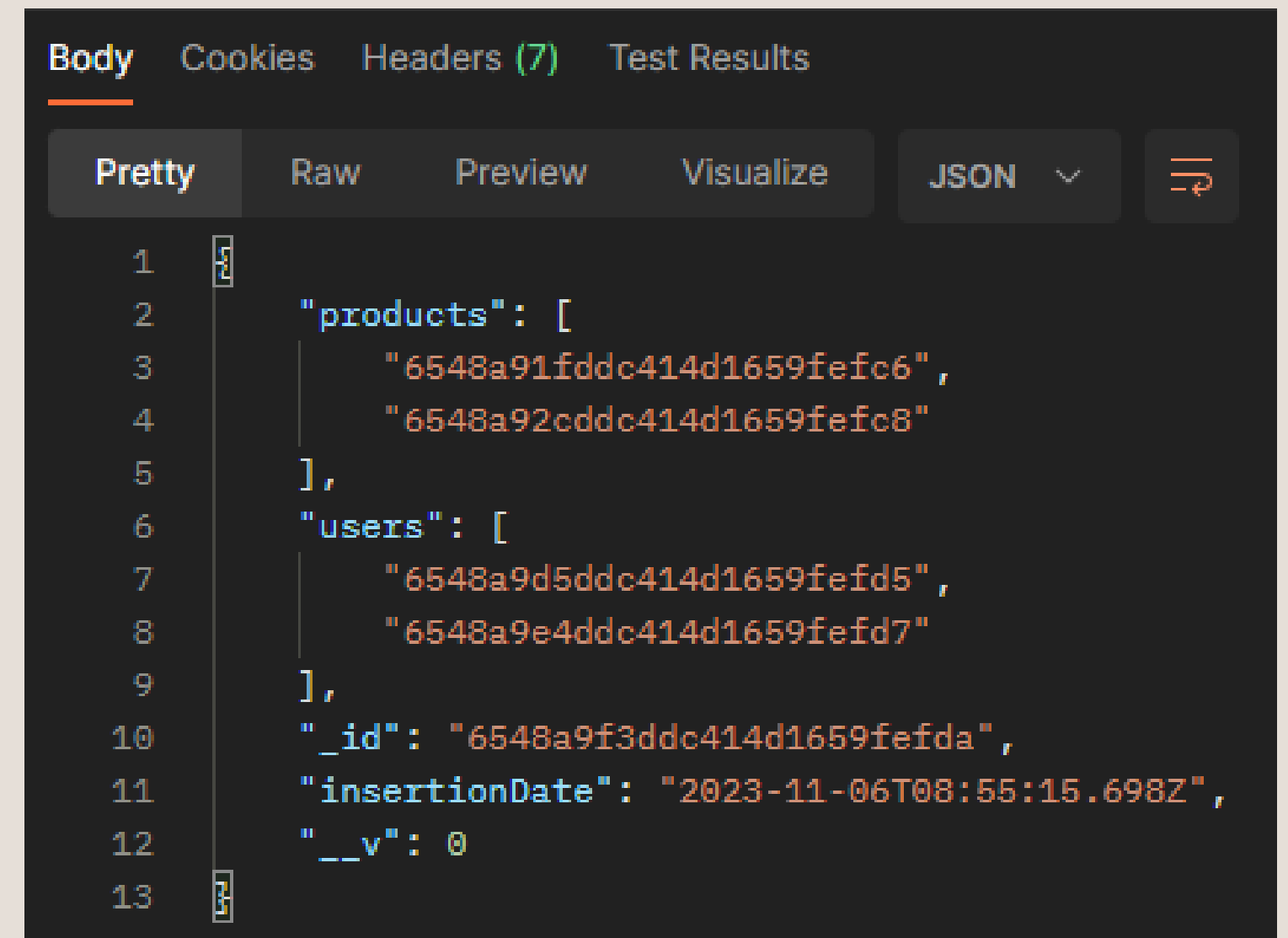
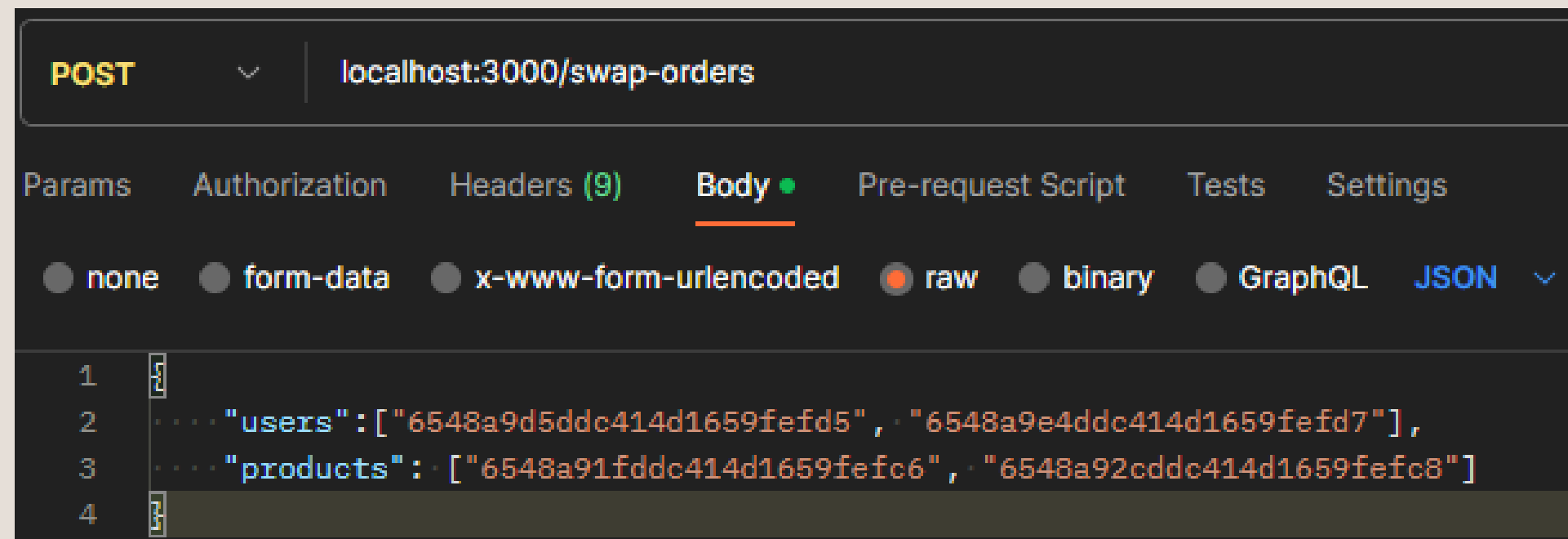
```
1  {
2    "_id": "6548a9f3ddc414d1659fefda",
3    "products": [
4      {
5        "_id": "6548a91fddc414d1659fefc6",
6        "name": "Socks",
7        "image": [
8          "https://images.freeimages.com/images/large-previews/66c/wool-socks-1-1565030.jpg"
9        ],
10       "__v": 0
11     },
12     {
13       "_id": "6548a92cddc414d1659fefc8",
14       "name": "Hoodie",
15       "image": [
16         "https://images.freeimages.com/images/large-previews/66c/wool-hoodie-1-1565030.jpg"
17       ],
18       "__v": 0
19     }
20   ],
21   "users": [
22     {
23       "_id": "6548a9d5ddc414d1659fefd5",
24       "firstName": "Tim",
25       "lastName": "Duncan",
26       "email": "tim@duncan.it",
27       "__v": 0
28     },
29     {
30       "_id": "6548a9e4ddc414d1659fefd7",
31       "firstName": "Michael",
32       "lastName": "Jordan",
33       "email": "mj@23.it",
34       "__v": 0
35     }
36   ],
37   "insertionDate": "2023-11-06T08:55:15.698Z",
38   "__v": 0
39 }
```

04 - API DOCUMENTATION

POST /swap-orders ➡ Creates a new swap order

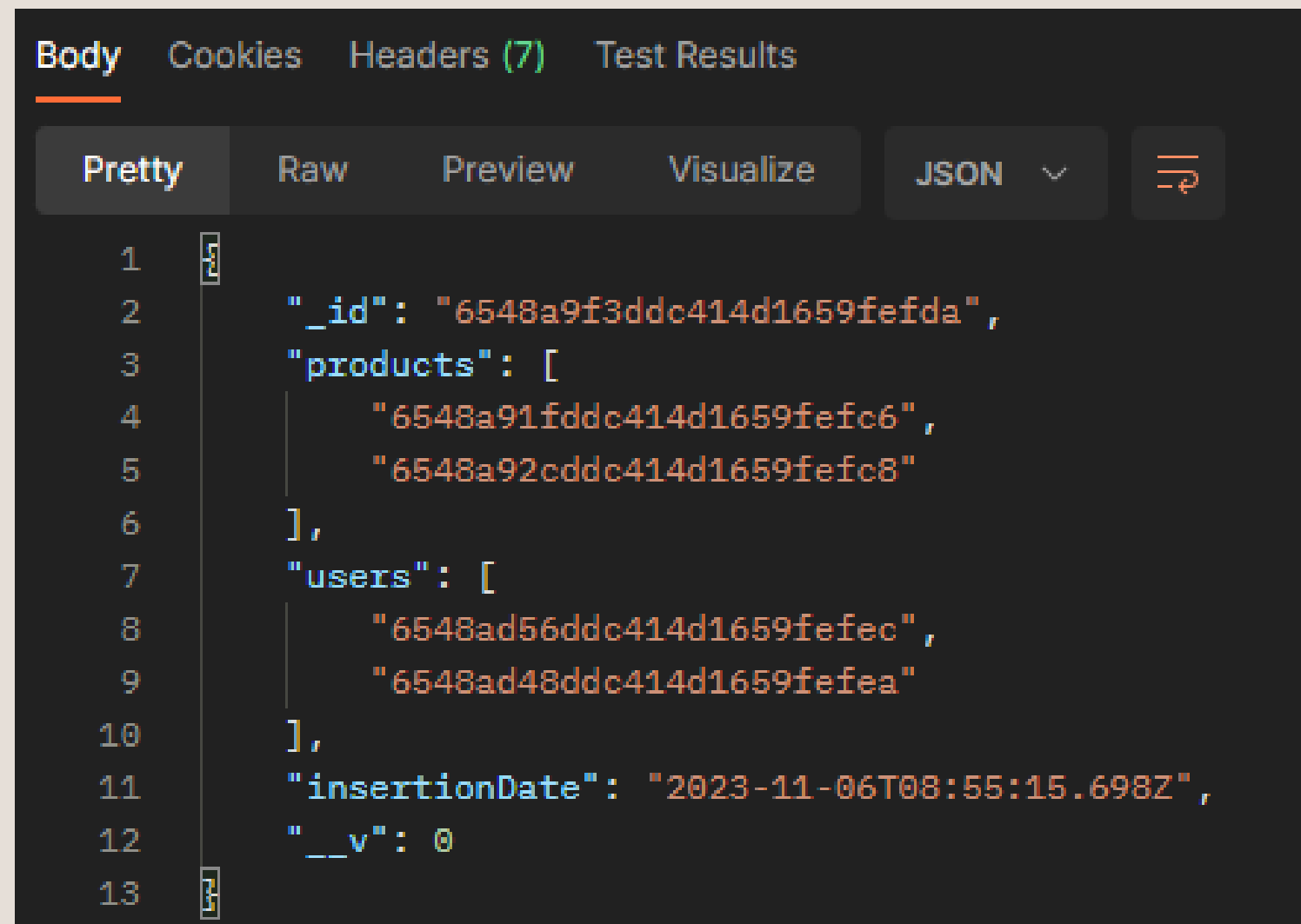
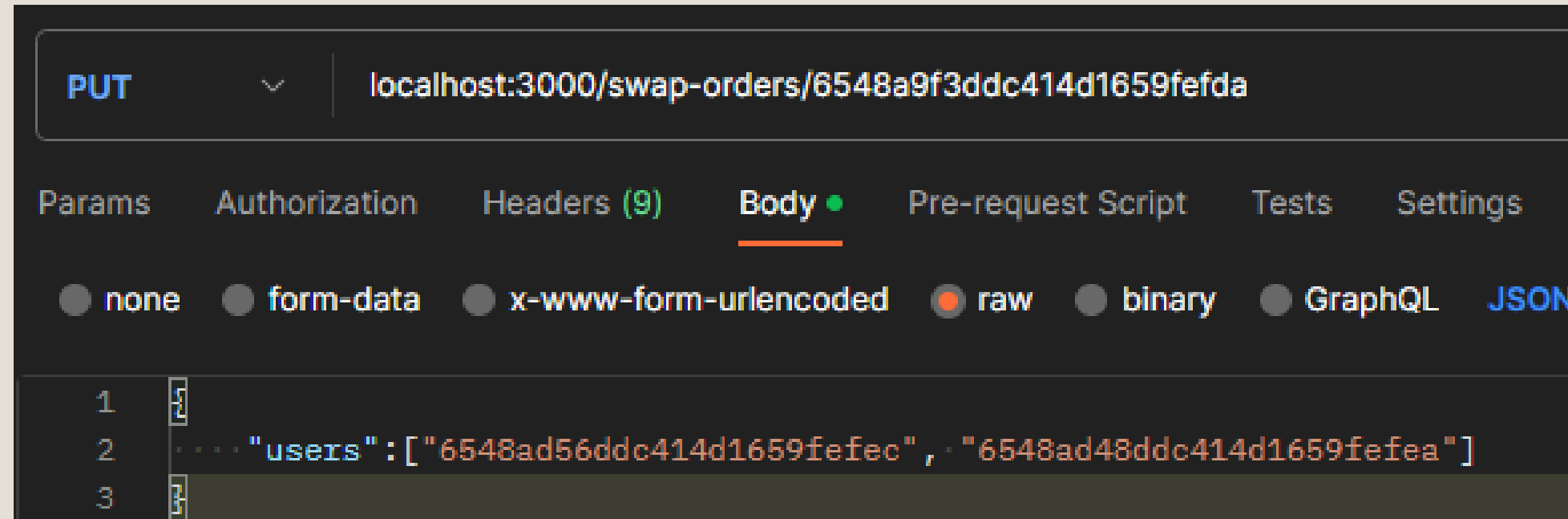
Request Body Fields (both required):

- **products:** An array of product IDs involved in the swap order. At least 2 unique and valid product IDs are required.
- **users:** An array of user IDs involved in the swap order. Exactly 2 unique and valid user IDs are required.



04 - API DOCUMENTATION

PUT /swap-orders/:id ➡ Updates an existing swap order based on the ID.



04 - API DOCUMENTATION

DELETE `/swap-orders/:id` ➡ Deletes a specific swap order based on the ID.

DELETE

`localhost:3000/swap-orders/6548a9f3ddc414d1659fefda`

Body Cookies Headers (7) Test Results

Pretty

Raw

Preview

Visualize

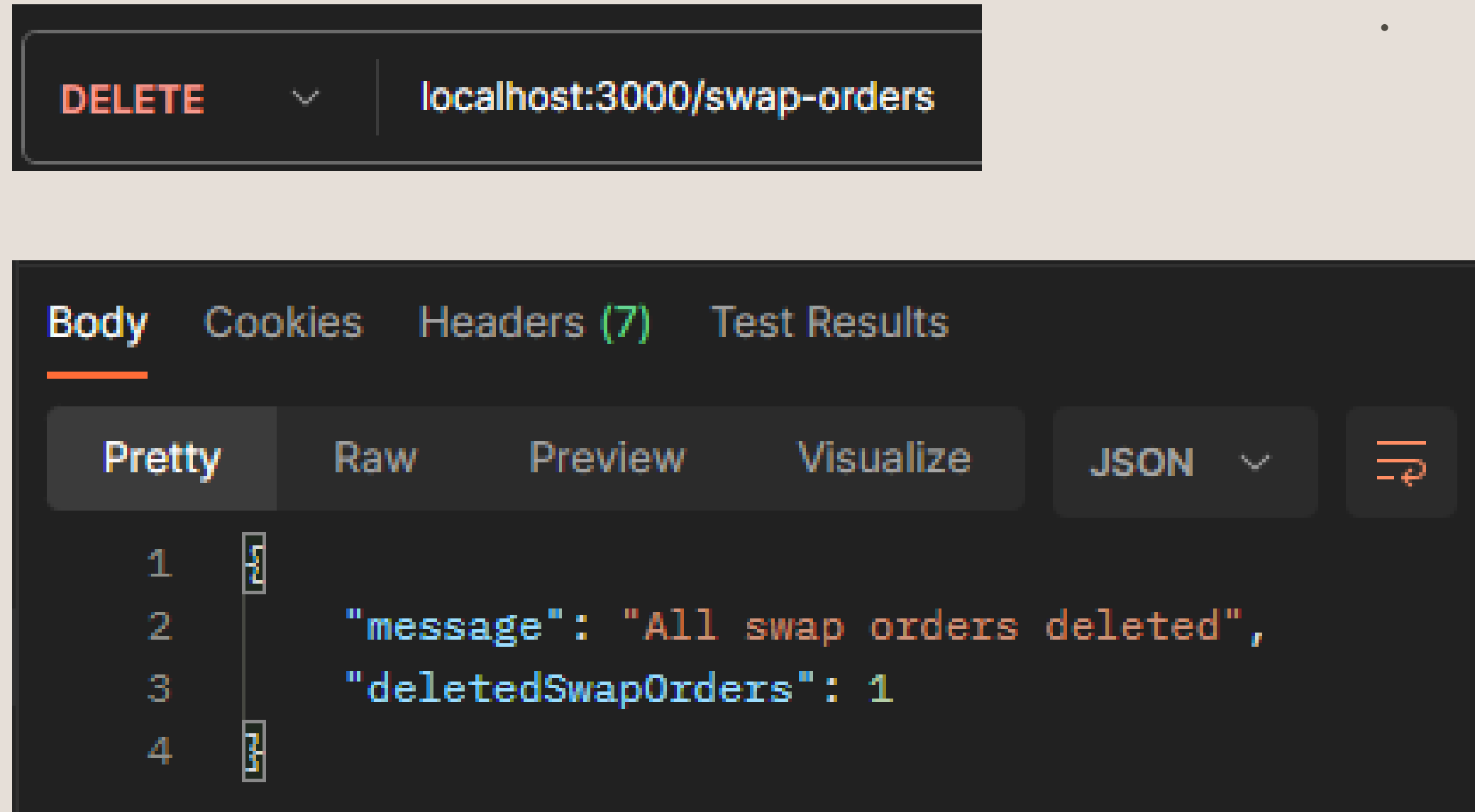
JSON



```
1 {
2   "_id": "6548a9f3ddc414d1659fefda",
3   "products": [
4     "6548a91fddc414d1659fefc6",
5     "6548a92cddc414d1659fefc8"
6   ],
7   "users": [
8     "6548ad56ddc414d1659fefec",
9     "6548ad48ddc414d1659fefea"
10  ],
11  "insertionDate": "2023-11-06T08:55:15.698Z",
12  "__v": 0
13 }
```

04 - API DOCUMENTATION

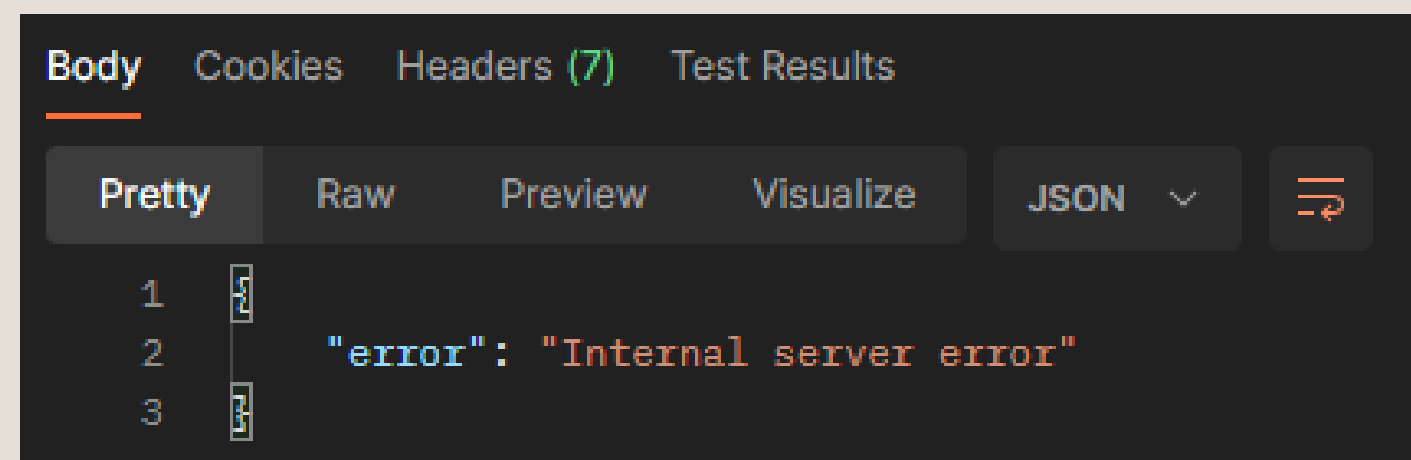
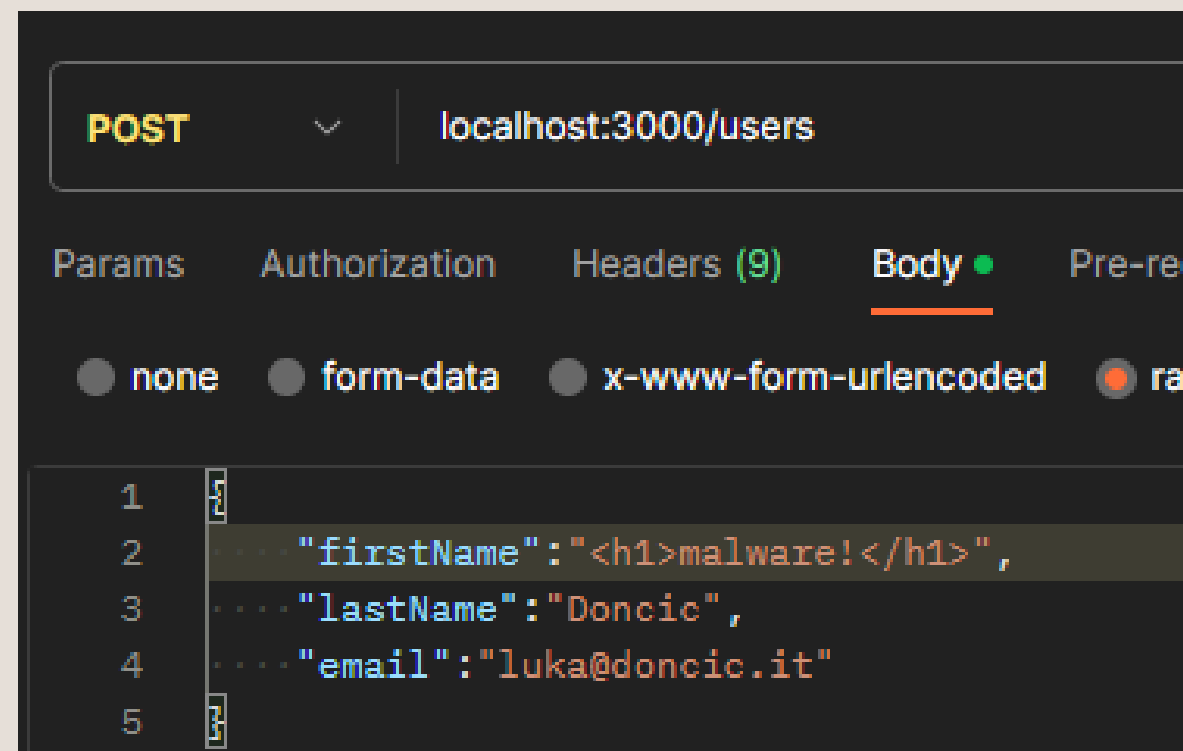
DELETE /swap-orders ➡ Deletes all swap orders



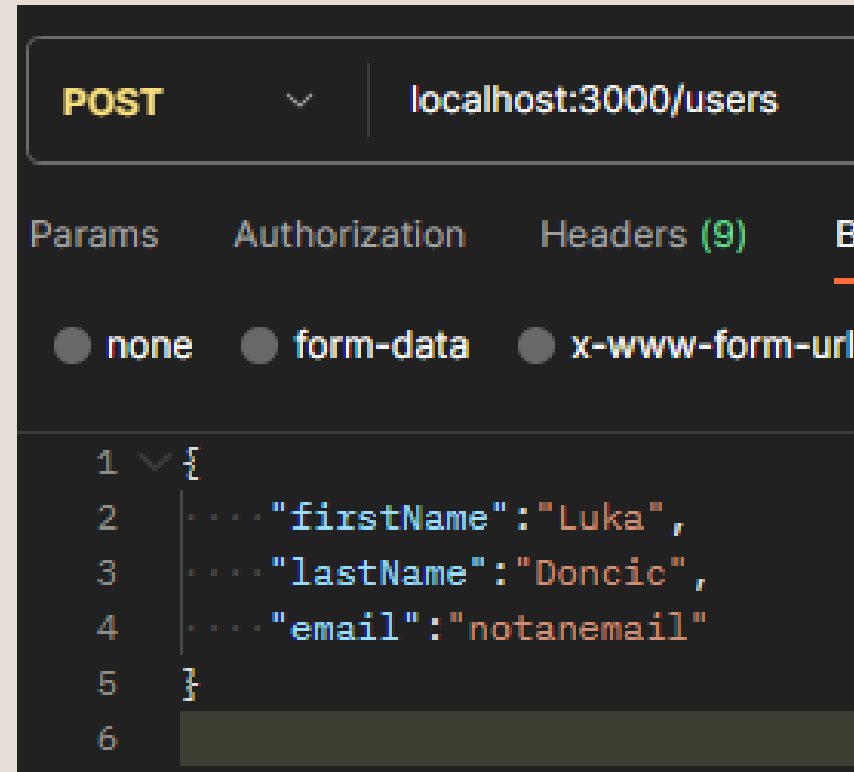
05- SANITIZATION AND VALIDATION

All user inputs are **sanitized** and **validated** to prevent common security issues such NoSQL Injection. This project utilizes **Express Validator** and follows best practices to ensure the safety and security of your data.

```
router.post('/', [
  check('firstName').notEmpty().escape().withMessage('First name cannot be empty'),
  check('lastName').notEmpty().escape().withMessage('Last name cannot be empty'),
  check('email').isEmail().escape().withMessage('Email required'),
], async (req, res) => {
```



05- SANITIZATION AND VALIDATION

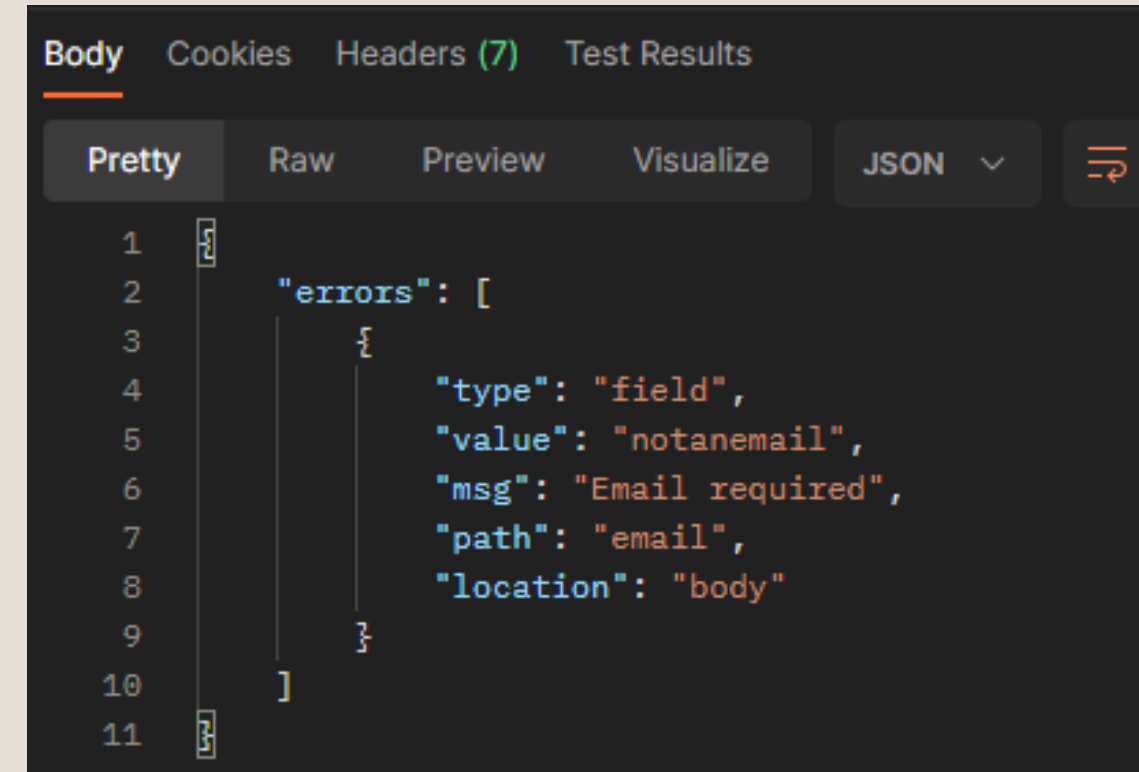


POST localhost:3000/users

Params Authorization Headers (9) Body

none form-data x-www-form-urlencoded

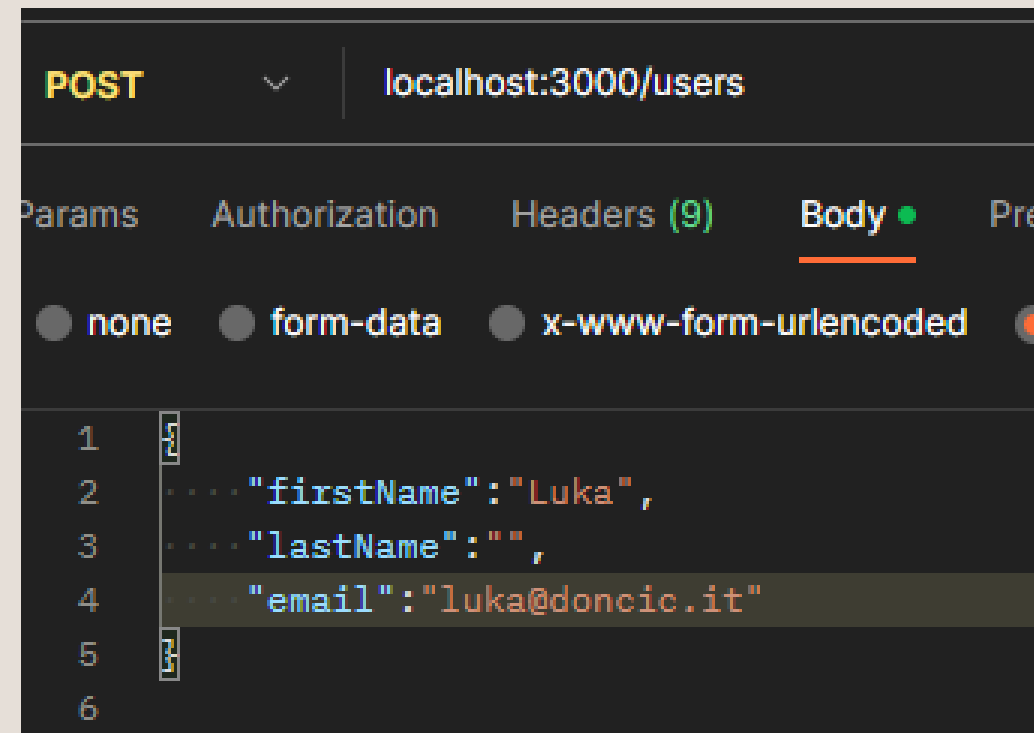
```
1 {
2   ... "firstName": "Luka",
3   ... "lastName": "Doncic",
4   ... "email": "notanemail"
5 }
6
```



Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   "errors": [
3     {
4       "type": "field",
5       "value": "notanemail",
6       "msg": "Email required",
7       "path": "email",
8       "location": "body"
9     }
10  ]
11 ]
```

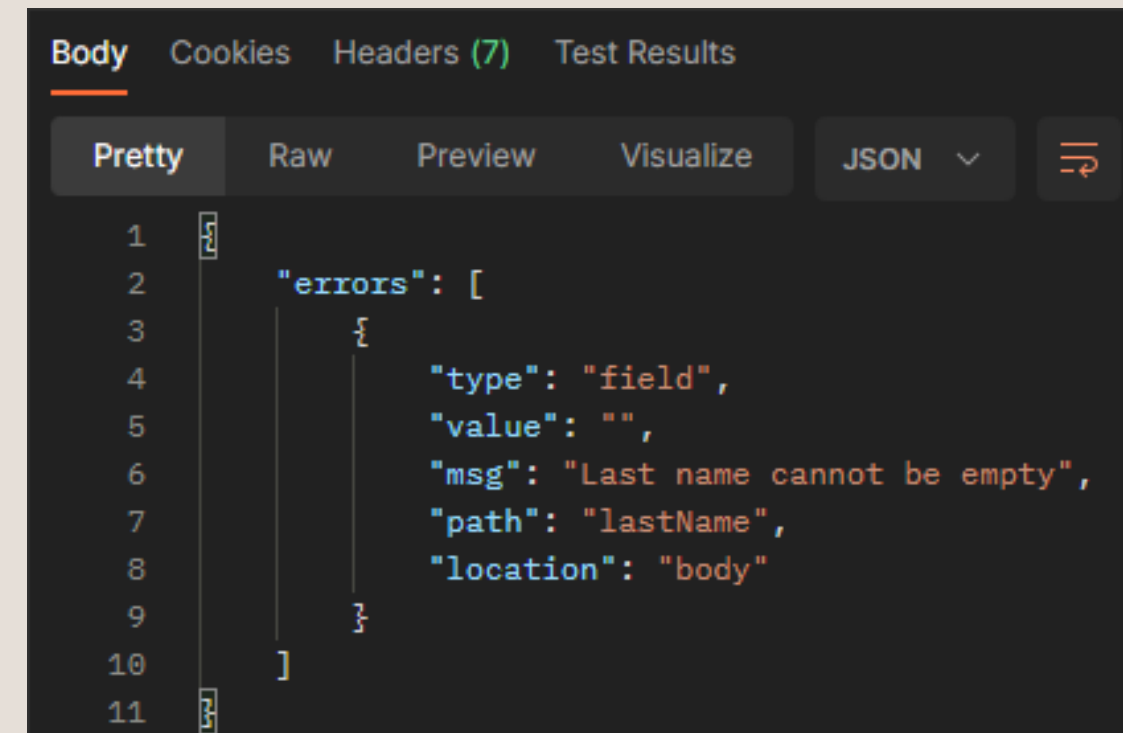


POST localhost:3000/users

Params Authorization Headers (9) Body

none form-data x-www-form-urlencoded

```
1 {
2   ... "firstName": "Luka",
3   ... "lastName": "",
4   ... "email": "luka@doncic.it"
5 }
6
```



Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   "errors": [
3     {
4       "type": "field",
5       "value": "",
6       "msg": "Last name cannot be empty",
7       "path": "lastName",
8       "location": "body"
9     }
10  ]
11 ]
```

06 - TESTING

This project includes 3 comprehensive and documented test suites with a total of 43 tests, created using **Jest**, **Supertest** and a **dedicated MongoDB in-memory server**. You can run these tests using the following command in a new terminal: ``npm test``

```
PASS  __tests__/users.test.js (5.102 s)
```

```
user
```

```
POST /users
```

- ✓ should create a new user (224 ms)
- ✓ should return 400 Bad Request for an invalid user (19 ms)
- ✓ should return 400 Bad Request for an invalid email input (29 ms)
- ✓ should return 500 Bad Request for an email that already exists (52 ms)

```
GET /users
```

- ✓ should return a list of users when users are present (48 ms)
- ✓ should return a specific user by ID (53 ms)
- ✓ should return an error when an ID does not exists (24 ms)

```
PUT /users/:id
```

- ✓ should update an existing user (69 ms)
- ✓ should return error 404 if the user to update does not exist (25 ms)
- ✓ should return an error if the user wants to update with an invalid email (49 ms)
- ✓ should return an error if the user wants to update with an email that already exists (58 ms)

```
DELETE /users/:id
```

- ✓ should delete an existing user by ID (74 ms)
- ✓ should return 404 Not Found if the user to delete does not exist (29 ms)
- ✓ should delete all users (56 ms)



06 - TESTING

PASS `__tests__/products.test.js`

`products`

`POST /products`

- ✓ `should create a new product` (143 ms)
- ✓ `should return 400 Bad Request for an invalid product` (13 ms)
- ✓ `should return 400 Bad Request for an empty image array` (12 ms)
- ✓ `should return 400 Bad Request for an image array that contains not valid image URLs` (19 ms)

`GET /products`

- ✓ `should return a list of products` (26 ms)
- ✓ `should return a specific product by ID` (42 ms)
- ✓ `should return an error when an ID does not exists` (17 ms)

`PUT /products/:id`

- ✓ `should update an existing product` (47 ms)
- ✓ `should return error 404 if the product to update does not exist` (20 ms)
- ✓ `should return 400 Bad Request for an invalid update` (40 ms)

`DELETE /products/:id`

- ✓ `should delete an existing product by ID` (58 ms)
- ✓ `should return 404 Not Found if the product ID does not exist` (24 ms)
- ✓ `should delete all products` (51 ms)

06 - TESTING

```
PASS __tests__/swaporders.test.js
swap orders
  POST /swap-orders
    ✓ should create a new swap order (45 ms)
    ✓ should return an error if the request is invalid (18 ms)
  GET /swap-orders
    ✓ should return swap orders (53 ms)
    ✓ should return a specific swap order by ID (81 ms)
    ✓ should return all swap orders involving a specific user by user ID (28 ms)
    ✓ should return all swap orders involving a specific product by product ID (28 ms)
    ✓ should return an error when a swap order ID does not exist (27 ms)
  PUT /swap-orders/:id
    ✓ should update an existing swap order (48 ms)
    ✓ should return an error if the request is invalid (33 ms)
    ✓ should return an error if the swap order ID does not exist (22 ms)
  DELETE /swap-orders/:id
    ✓ should delete an existing swap order by ID (64 ms)
    ✓ should return an error if the swap order ID does not exist (29 ms)
    ✓ should delete all swap orders involving a specific user upon user deletion (105 ms)
    ✓ should delete all swap orders involving a specific product upon product deletion (87 ms)
    ✓ should delete all swap orders (27 ms)
```

```
Test Suites: 3 passed, 3 total
Tests:       42 passed, 42 total
Snapshots:   0 total
Time:        9.801 s
Ran all test suites.
```





07 - CONTACTS

E - MAIL diego.boost@gmail.com

WEBSITE <https://diego-lauricella.netlify.app>

G I T H U B <https://github.com/diegoddie>