**Ecommerce Application Documentation**

**Documented on Jan 2024 By ---- Ramya Gopala**

**Table of Contents**

1. **Overview ------------------------------------------------------------------- Page 3**
2. **Requirements -------------------------------------------------------------------- Page 3**
3. **Building the Application ------------------------------------------------------------------ Page 3**
4. **Running the Application ------------------------------------------------------------------ Page 4**
5. **Integrate OpenAPI with Spring Boot -Test API Using Swagger UI -------------- Page 6  
   5.1. Testing Product API Calls ---------------------------------------------------------- Page 7  
   5.2. Testing Order API Calls ----------------------------------------------------------- Page 13  
   5.3. Testing Order Item API Calls -------------------------------------------------------- Page 19**
6. **Testing via JUnit and Mockito ----------------------------------------------------------- Page 23**

**6.1 Testing Overview ------------------------------------------------------------------------Page 25  
 6.2 Steps to Testing for Ecommerce Application ------------------------------------ Page 25**

1. **Overview**

“Ecommerce” is a Spring Boot web application that integrates with a PostgreSQL database and uses Docker to run in a container. The application has REST API that allows users to perform CRUD (create, read, update, delete) operations on a data model representing a simple e-commerce platform. The REST API has been defined using OpenAI.

1. **Requirements**
2. **Technology Stack:**🡺Spring Boot  
   🡺Java 17

<https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>  
🡺PostgreSQL  
 https://www.postgresql.org/download/  
🡺Docker

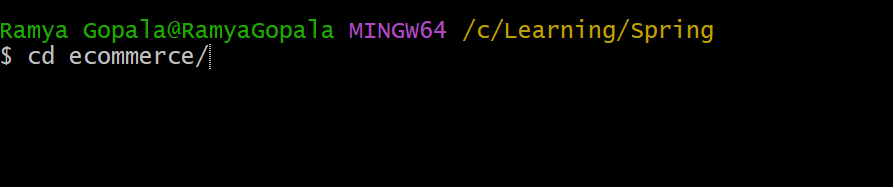
https://docs.docker.com/desktop/install/mac-install/

1. **Database Integration:  
   🡺**Integrated with PostgreSQL using Spring Data JPA**.**
2. **REST API:**🡺Allows users to perform CRUD operations.
3. **Steps to Build the Application**

**Step1:** Unzip the Ecommerce Application folder

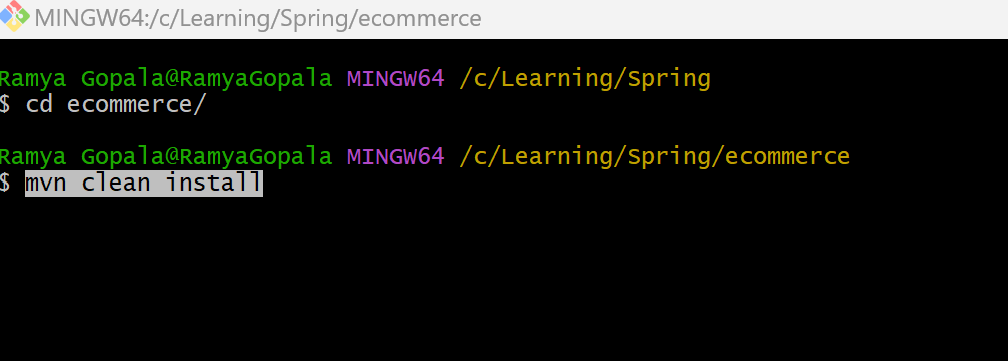
**Step2:** Navigate to the project directory

**Command**: cd ecommerce

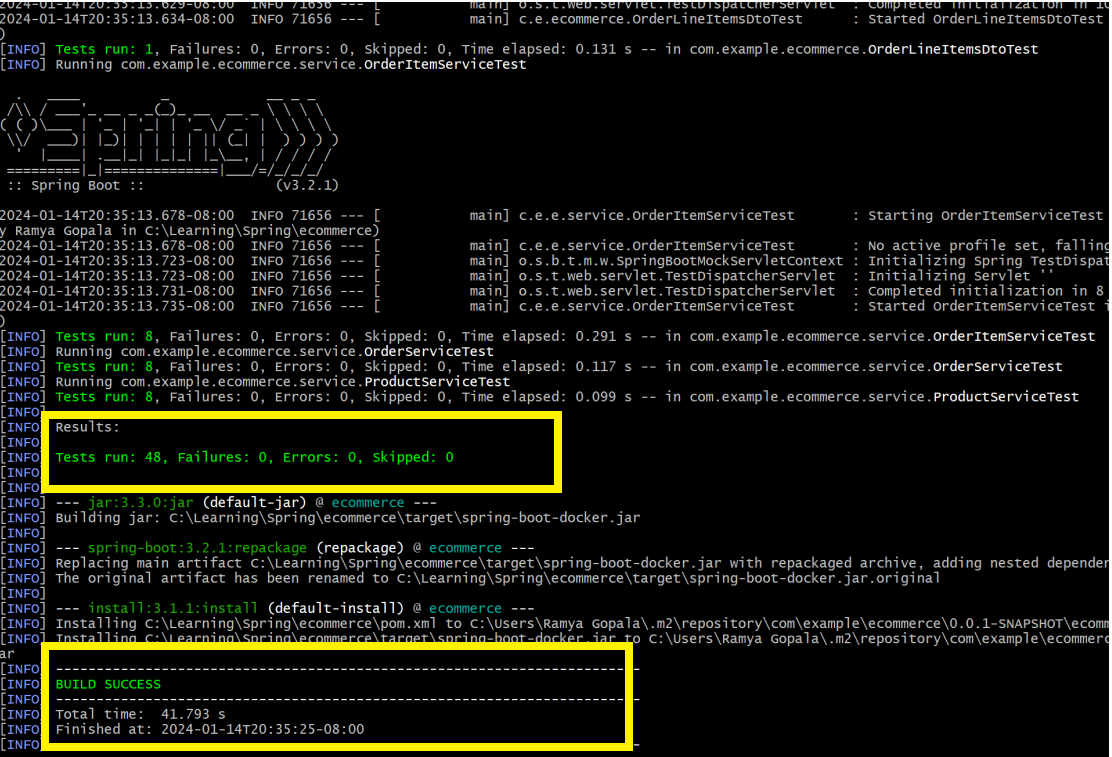


**Step3:** Build the application using Maven:

**Command**: mvn clean install

****

We can also see all the test cases for out application is running, and all the test cases have passed successfully.

****

1. **Steps to Run the Application**

We are using Docker Compose is a tool for defining and running multi-container Docker applications. **Dockerfile and docker-compose.yml file has already been defined** in the root of the project to define the multi-container setup.

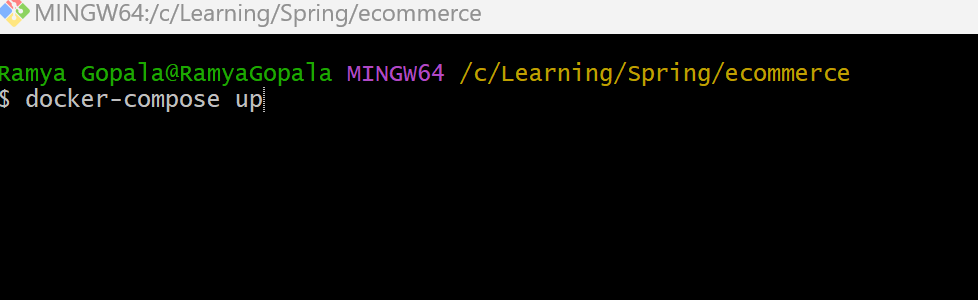
**Here are the steps to set up and start your Spring Boot e-commerce application using Docker Compose:**

**Note:** Please ensure Docker engine is up and running fine using command ‘docker -v’ else install Docker.



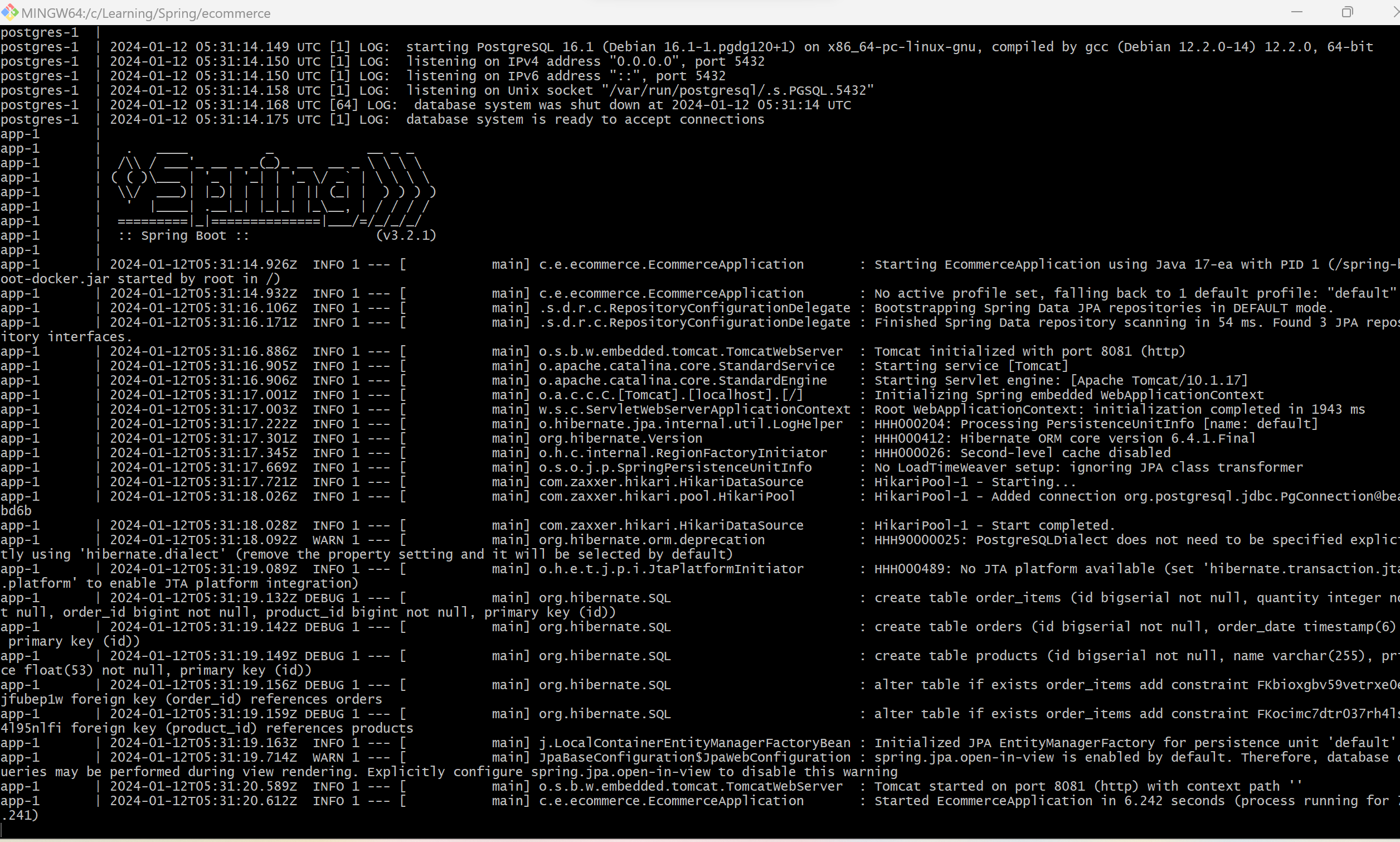
**Step1:** Open a terminal in the root of your project and run the following command to build and start the Docker containers:

**Command:** docker-compose up

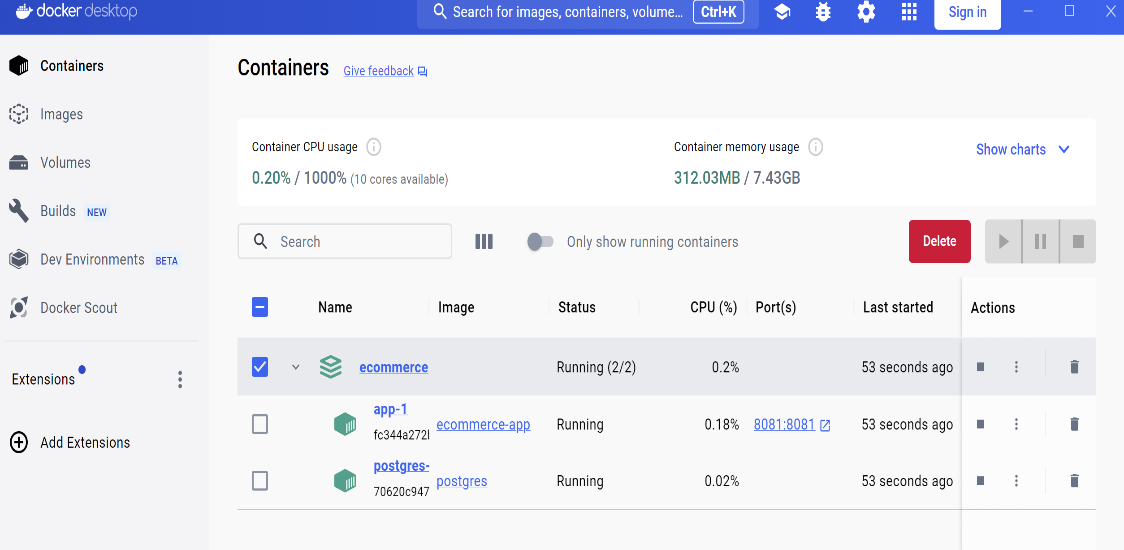
****

Once the containers are up and running, you can access your Spring Boot application at

<http://localhost:8081/swagger-ui/index.html>

****

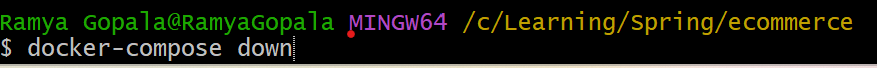
We can also view the containers of our application in Docker-UI as shown below

****

**Step2: Stop the application**

To stop the containers, open a terminal in the project's root directory and run command

Command: docker-compose down



1. **Integrate OpenAPI with Spring Boot -Test API Using Swagger UI**
2. **Run the Application:**

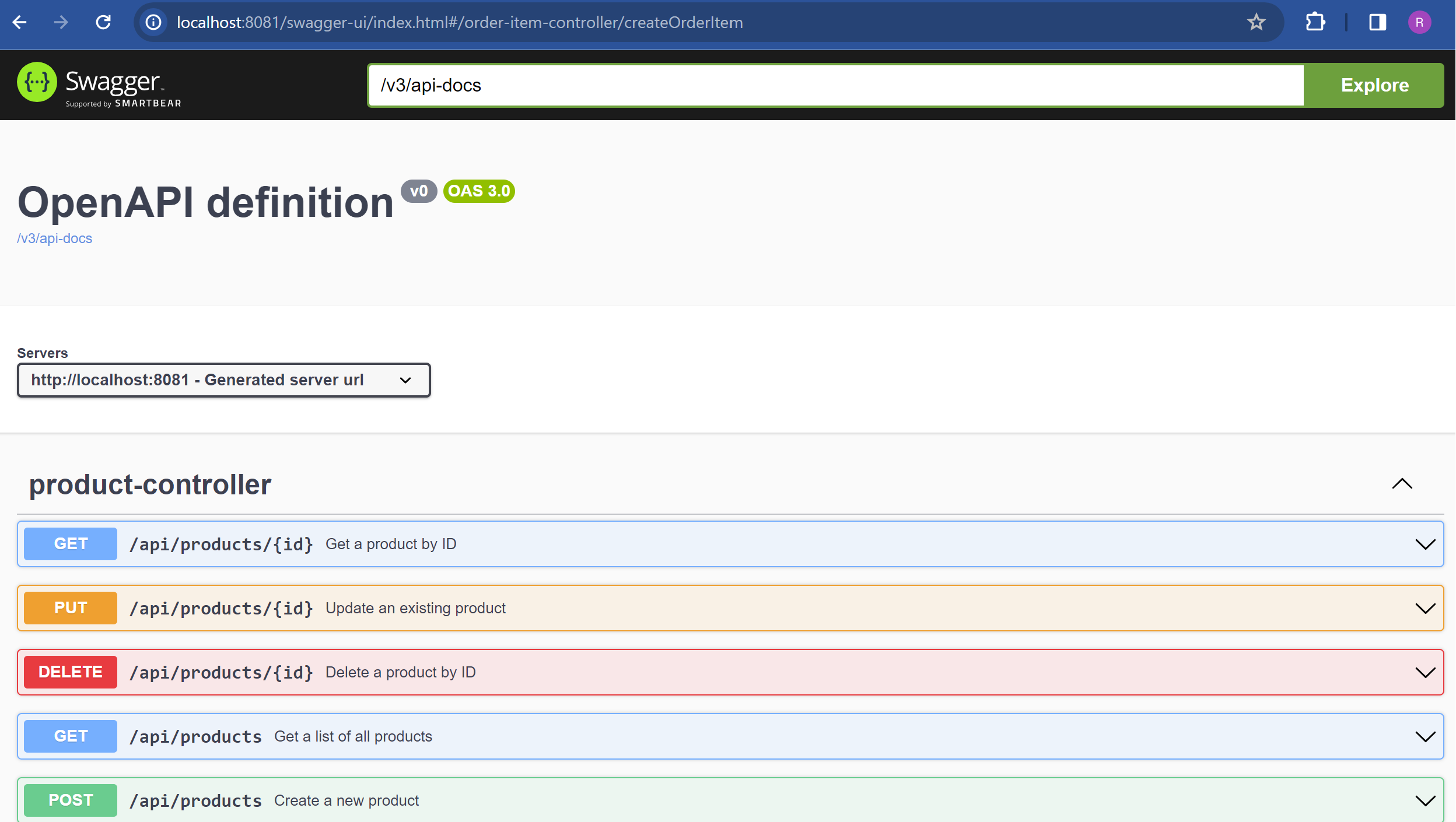
Ensure your Spring Boot application has started.

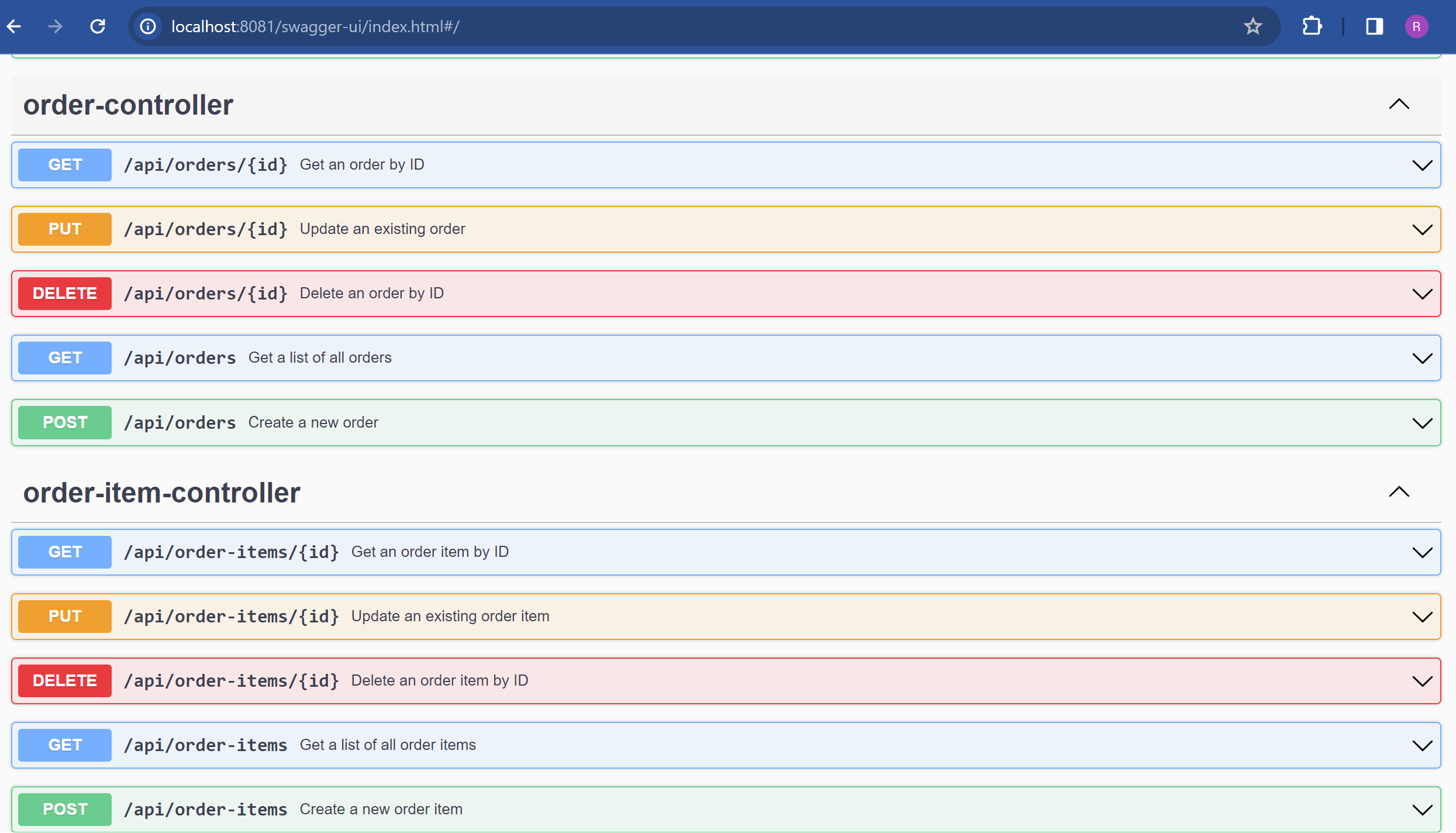
1. **Access Swagger UI:**

Open your web browser and navigate to

[**http://localhost:8080/swagger-ui.html**](http://localhost:8080/swagger-ui.html)**.**

Swagger UI provides an interactive interface to explore and test your API. You can send requests directly from the Swagger UI interface and see response. Below screenshot shows the page you see when you click the Swagger UI.





* 1. **Testing Product API Calls:**

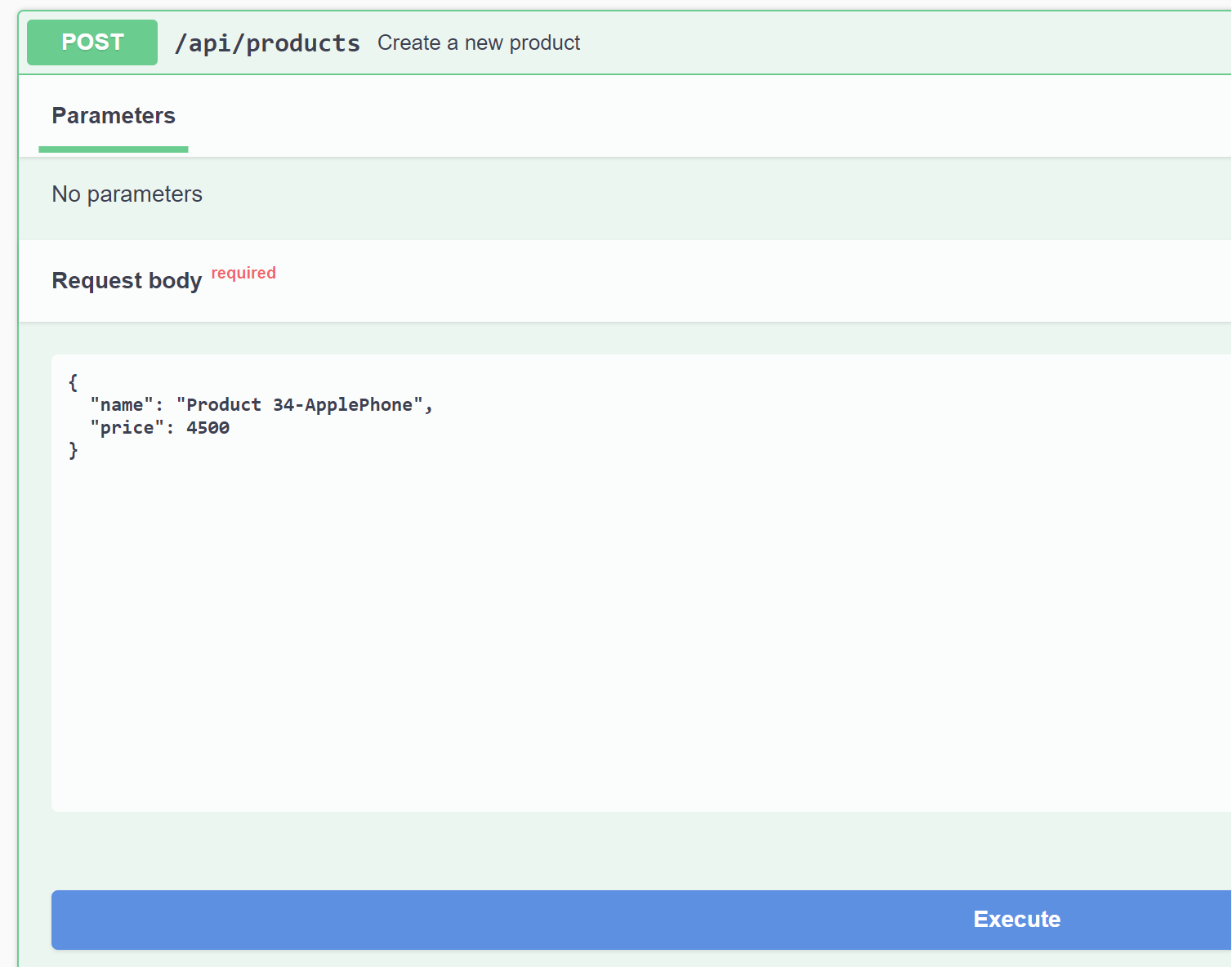
**1.a> POST /api/products - Create a new product**

**Enter the details in Request Body as below and click execute:**

{

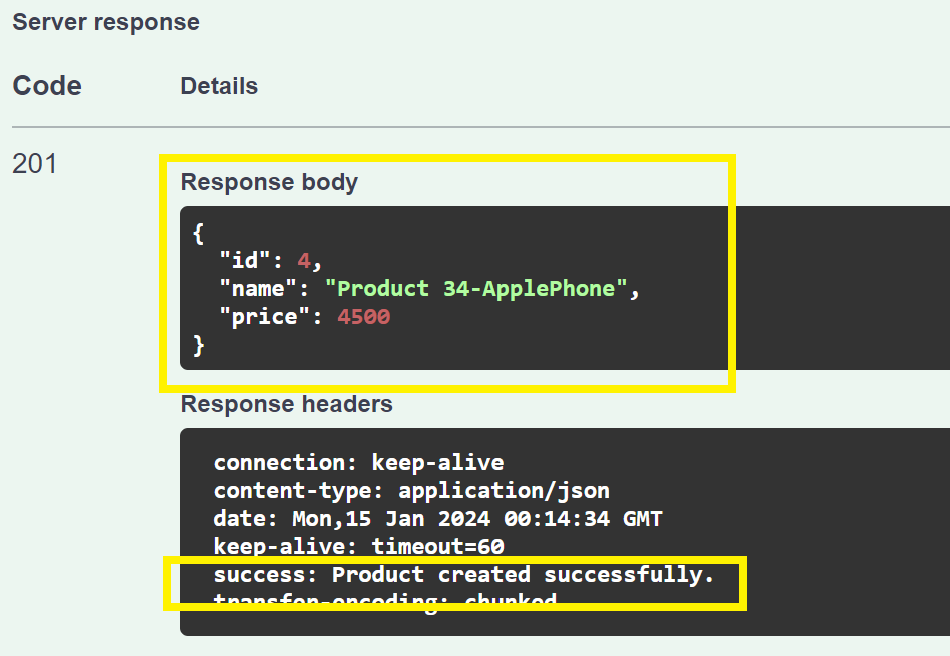
"name": "Product 34-ApplePhone",

"price": 4500

}

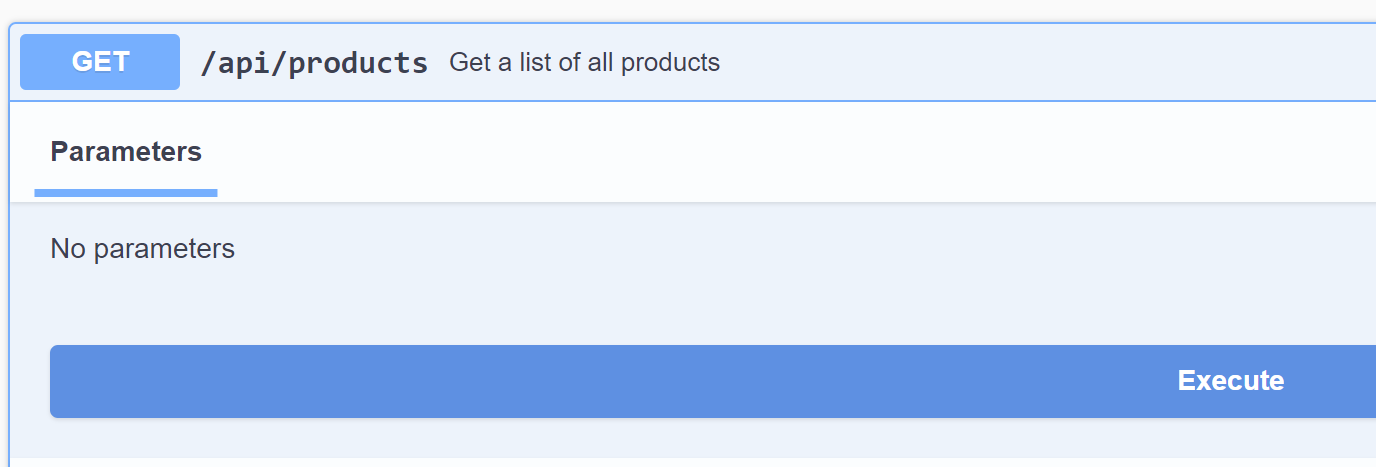
**Response of Create a new product:**

Once the product has been created, we can view the Success Message: **“Product created Successfully”**

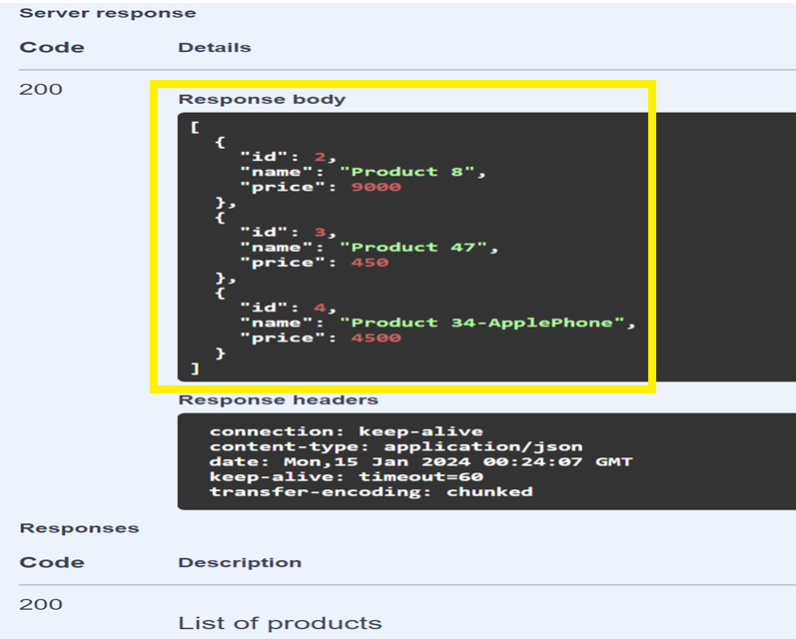


**1.b> GET /api/products - Get a list of all products**

Navigate to GET /api/products and click on execute to get a list of all products

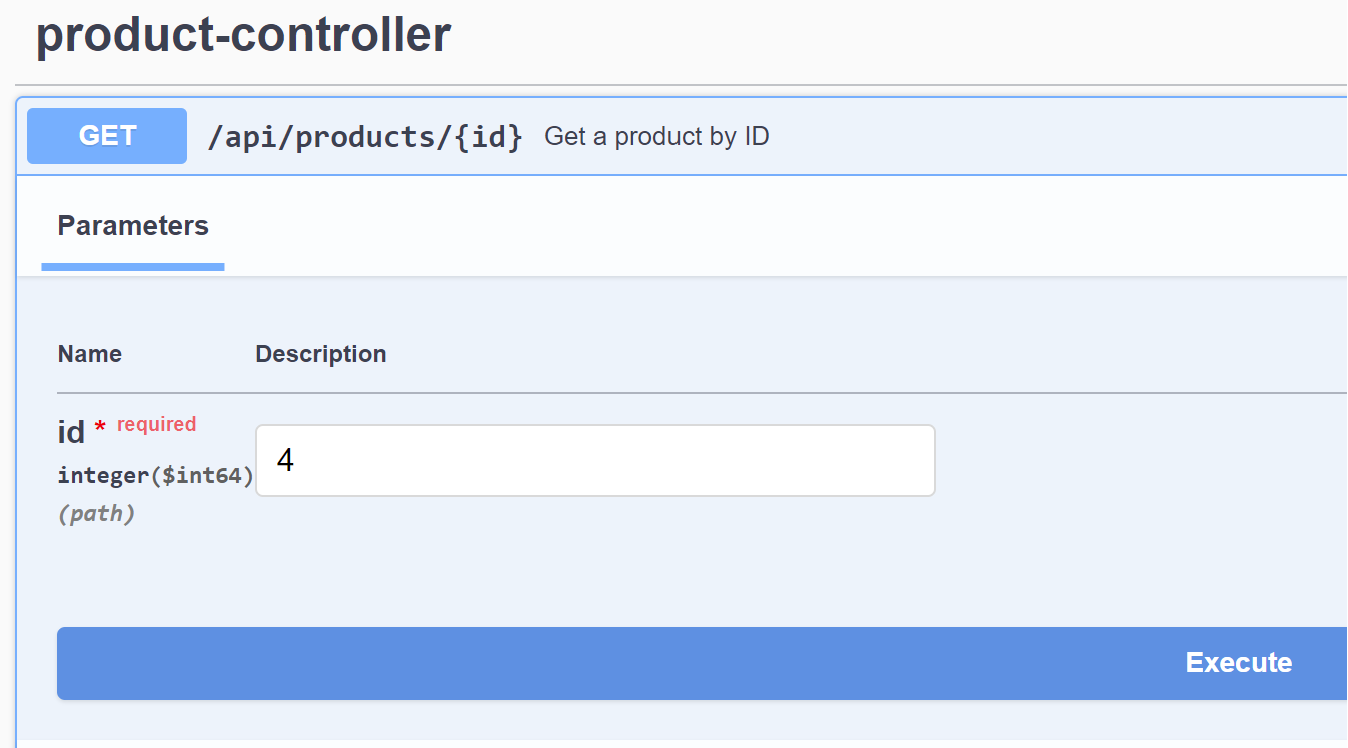


**Response:** We will get a list of all products which has been successfully created till now.

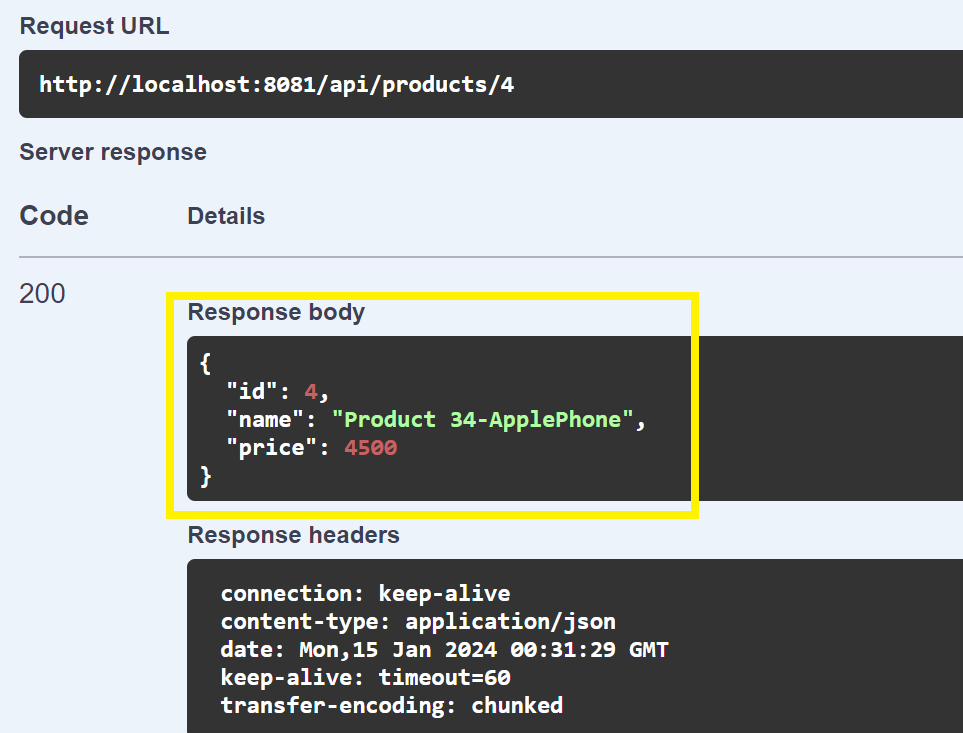


**1.c> GET /api/products/{id} - Get a product by ID**

Click on Execute by providing the Product ID for which you want to fetch details

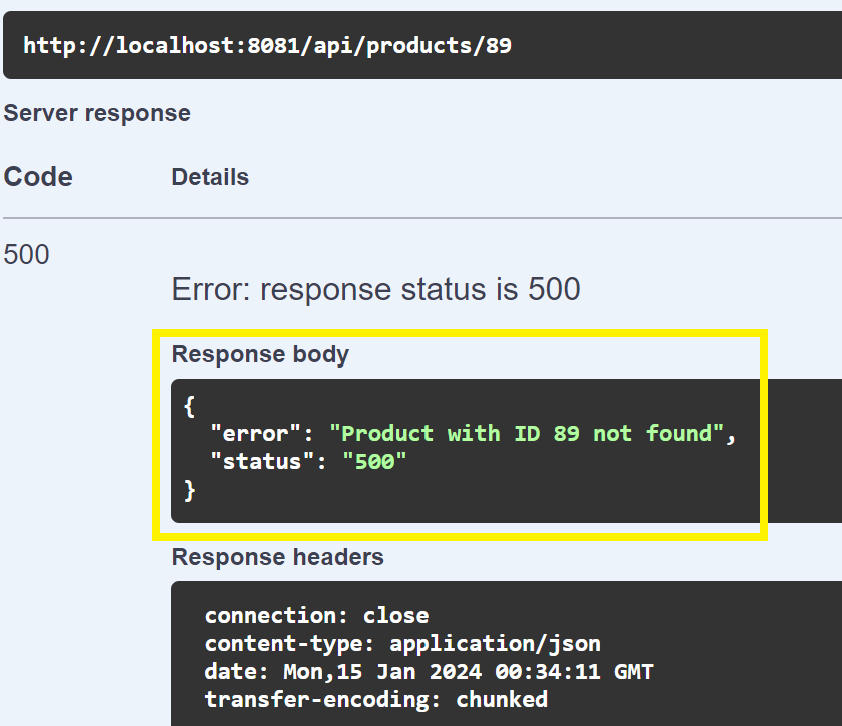


**Response**We will get the details of that particular product



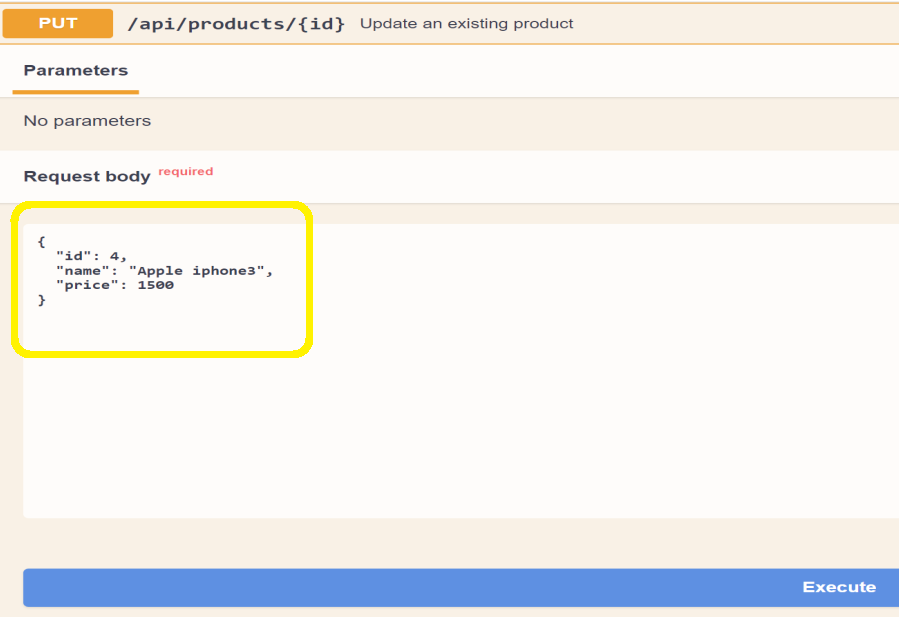
**Error Handling Scenario: Query the Product ID which has not been created.**

In case the Product is not found, we get an error “Product with ID not found"  
Let say if I want to Query Product ID: 89 which has not been created/present in database,  
We get an error as shown below

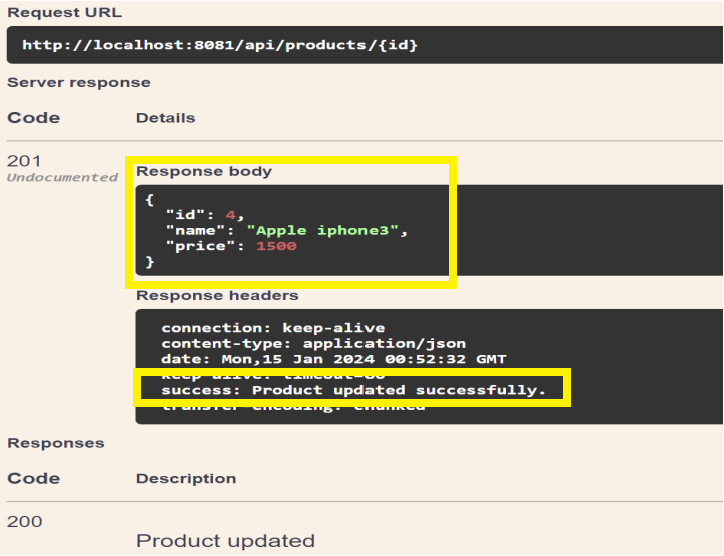


**1.d> PUT /api/products/{id} - Update an existing product**

In Oder to Update an existing product, enter the Productid, and the parameters you wish to edit and click Execute  
Ex: I will update the Product with ID: 4 as below



**Response**: Once the Product has been updated we can view Success message   
“Product Updated Successfully”



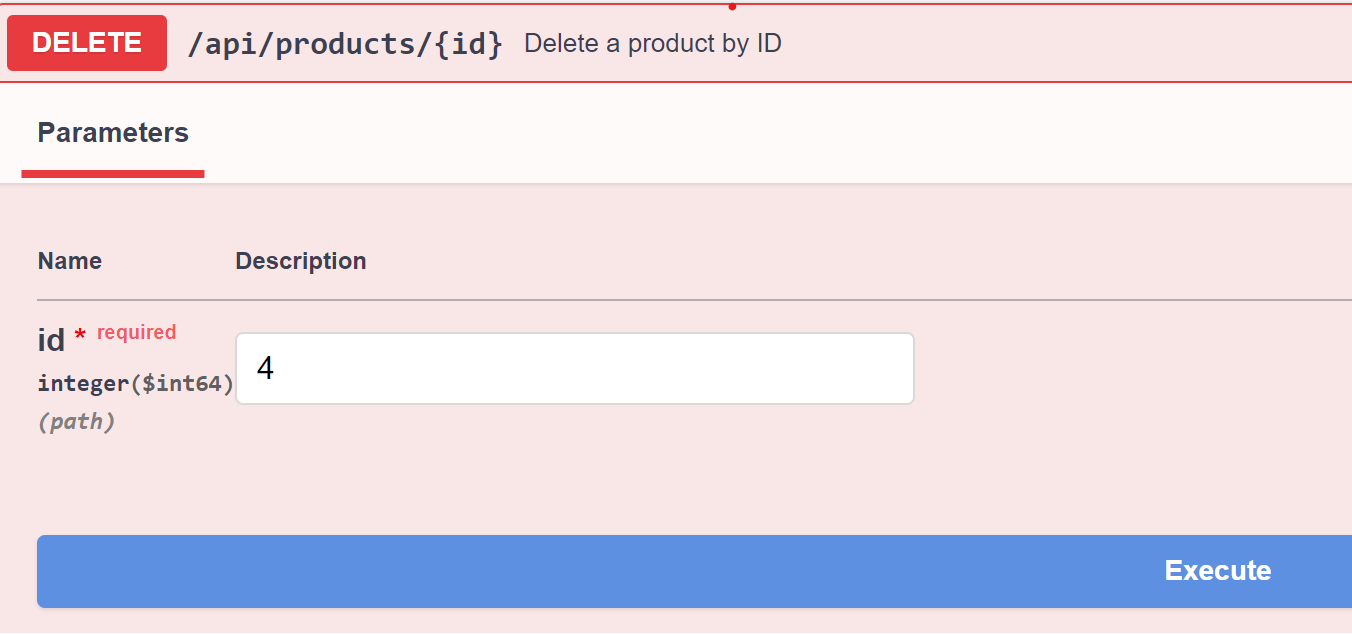
**Error Handling Scenario: Updating the Product ID which has not been created.**

In case the Product is not found, we get an error “Product with ID not found"  
Let say if I want to Update Product ID: 89 which has not been created/present in database  
We Get an error as shown below

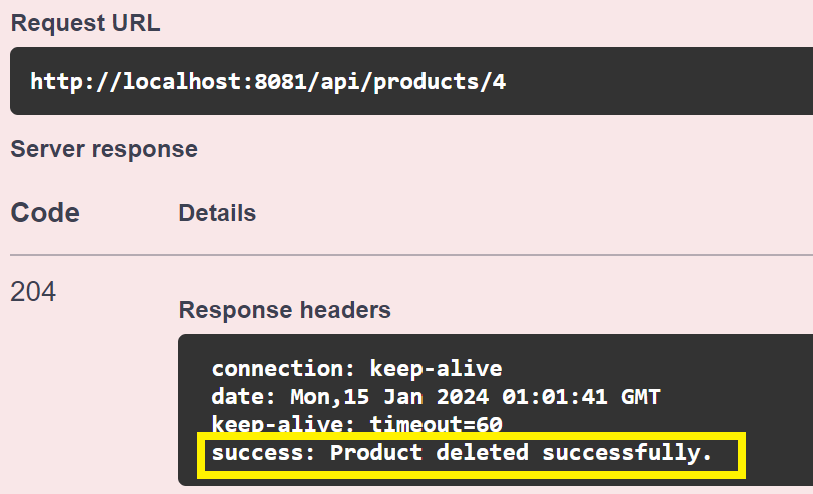
****

**1.e> DELETE /api/products/{id} - Delete a product by ID**

Enter the Product ID which you want to delete and click Execute.



**Response**: Once the Product has been deleted, we can view Success message   
“Product has been deleted Successfully”

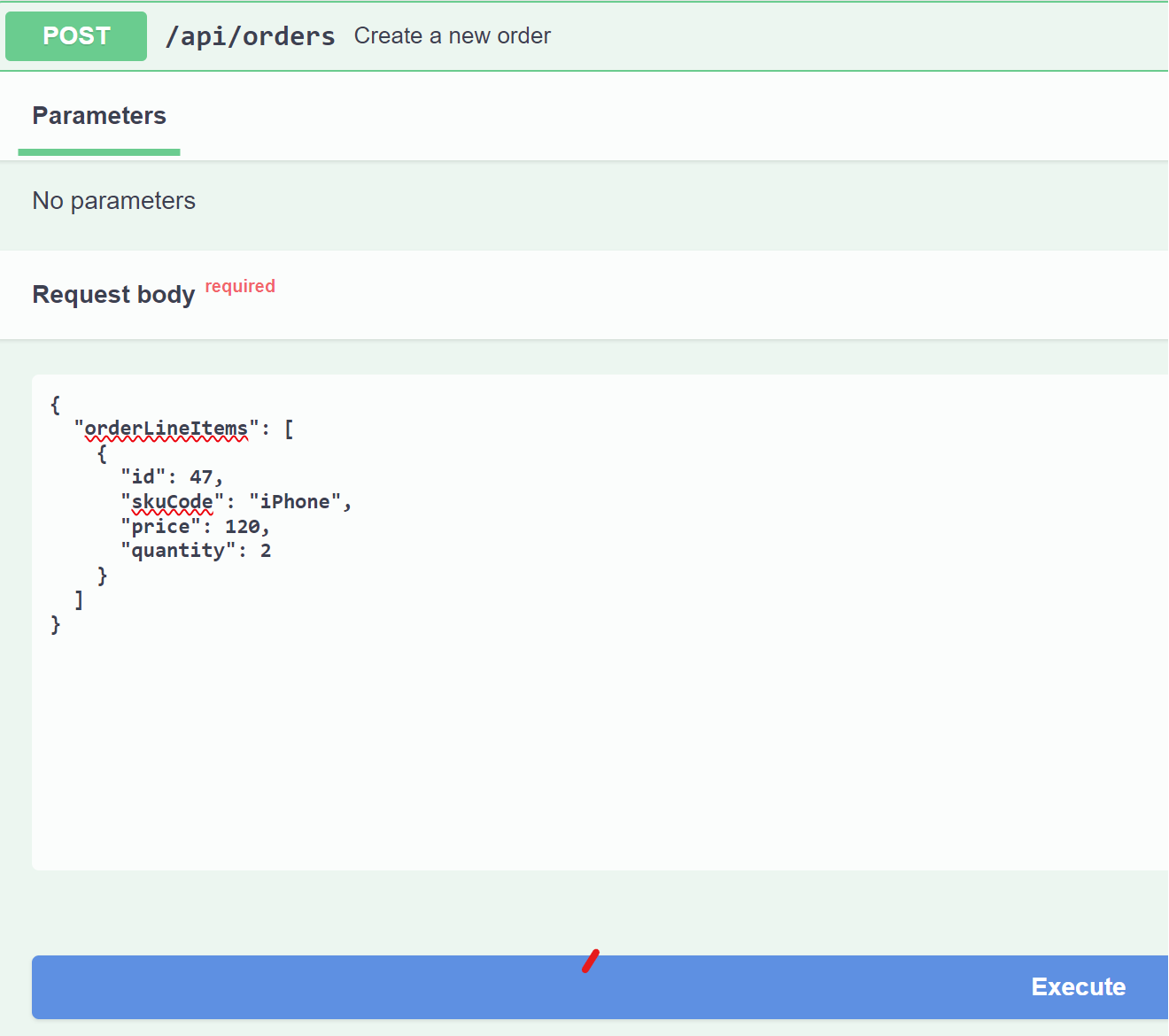


**5.2. Testing Order API Calls:**

**2.a> POST /api/orders - Create a new order**

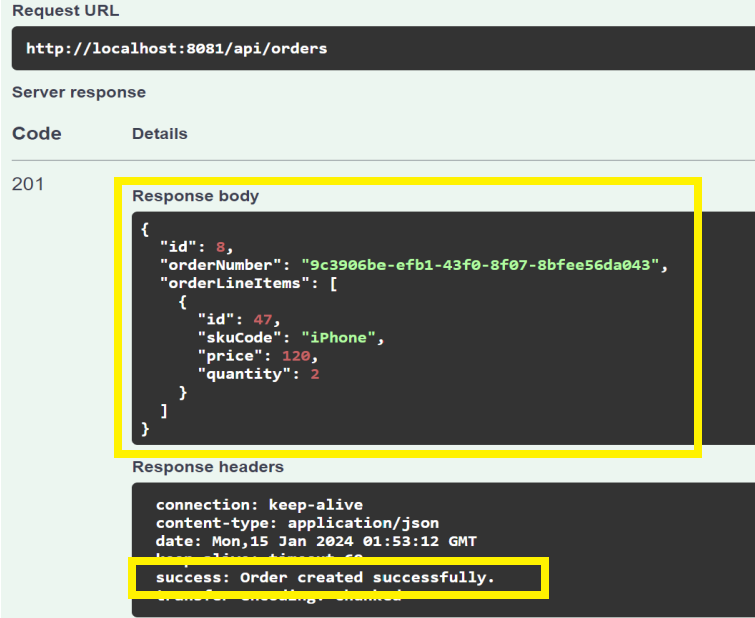
To create a new order, enter the fields in the request body and click execute,

**Request Body**: { "orderLineItems": [  
{ "id": 47,  
 "skuCode": "iPhone",   
 "price": 120,  
 "quantity": 2  
 }   
]}



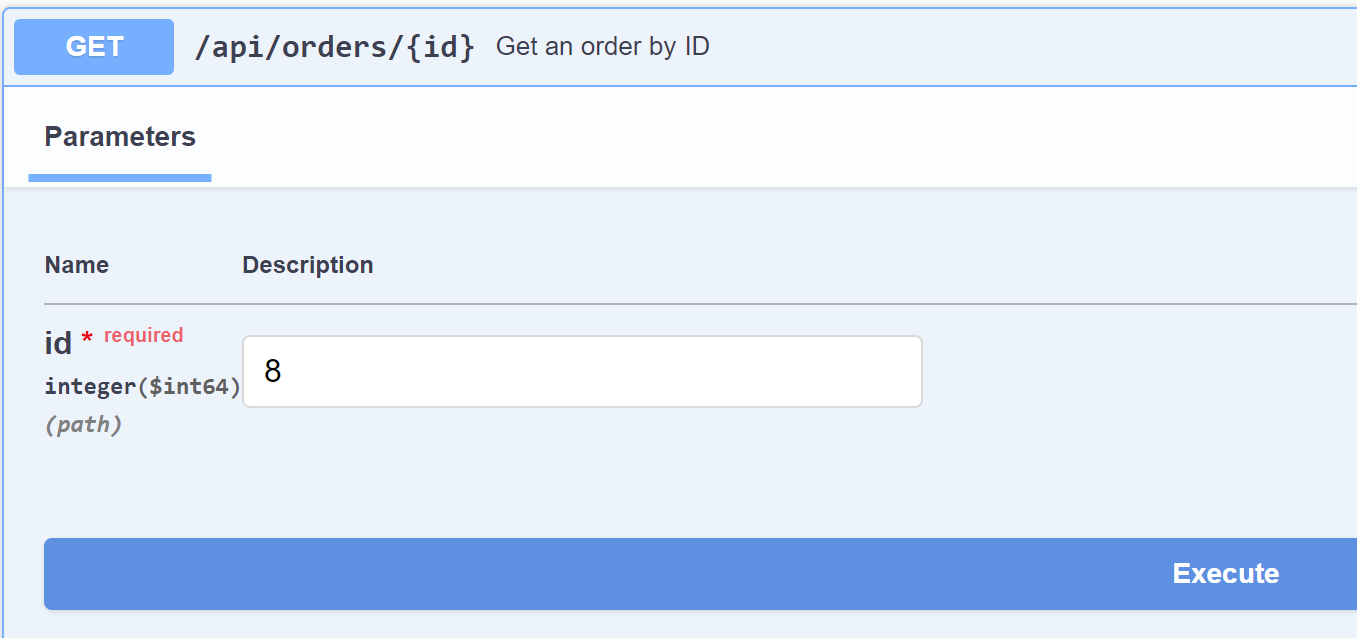
**Response of Create a new Order:**

Once the order has been created, we can view the Success Message: **“Order created Successfully”**

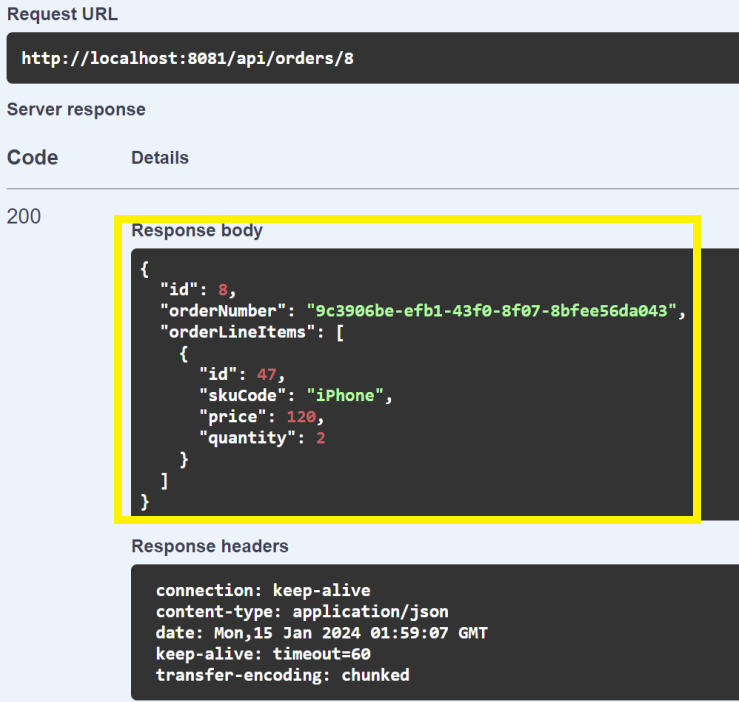


**2.b> GET /api/orders/{id} - Get an order by ID**

In order to fetch details based on ID, please provide the OrderID and click execute.

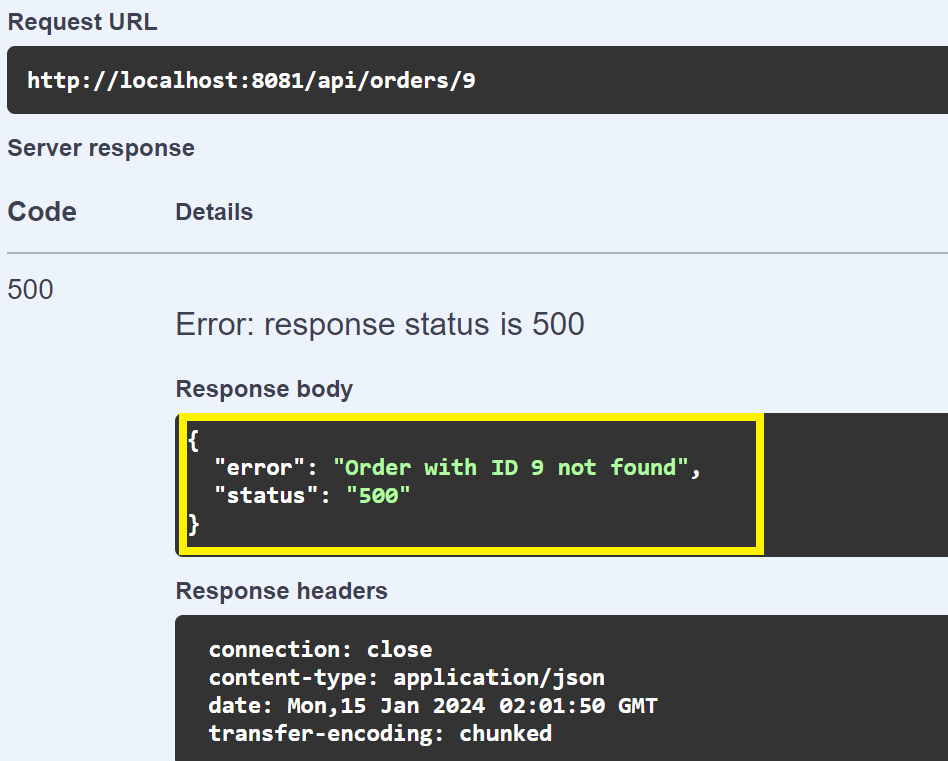


**Response**We will get the details of that particular Order.



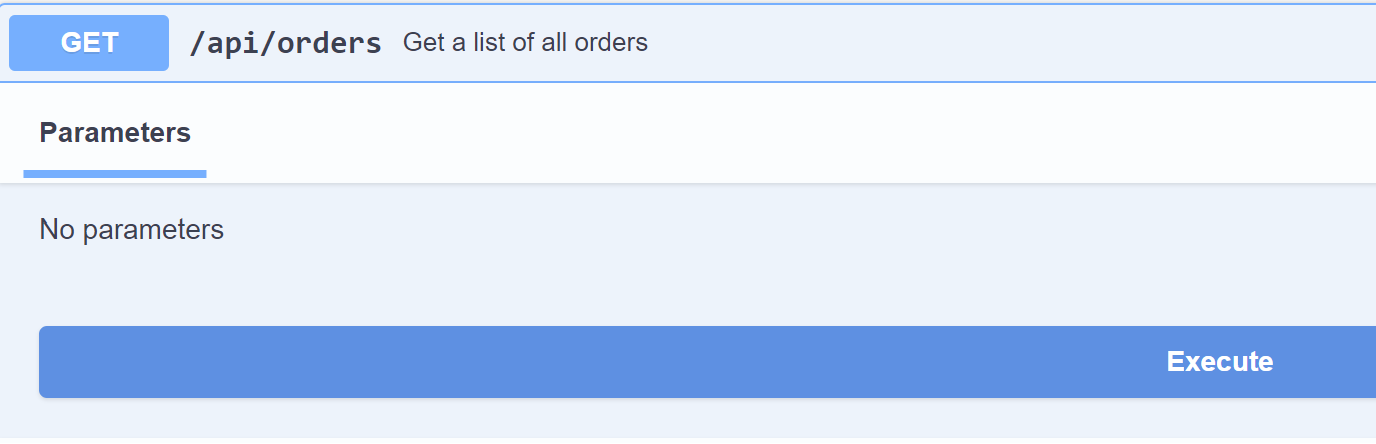
**Error Handling Scenario: Query the Order ID which has not been created.**

In case the Order is not found, we get an error “Order with ID not found"  
Let say if I want to Query Order ID: 9 which has not been created/present in database,  
We get an error as shown below

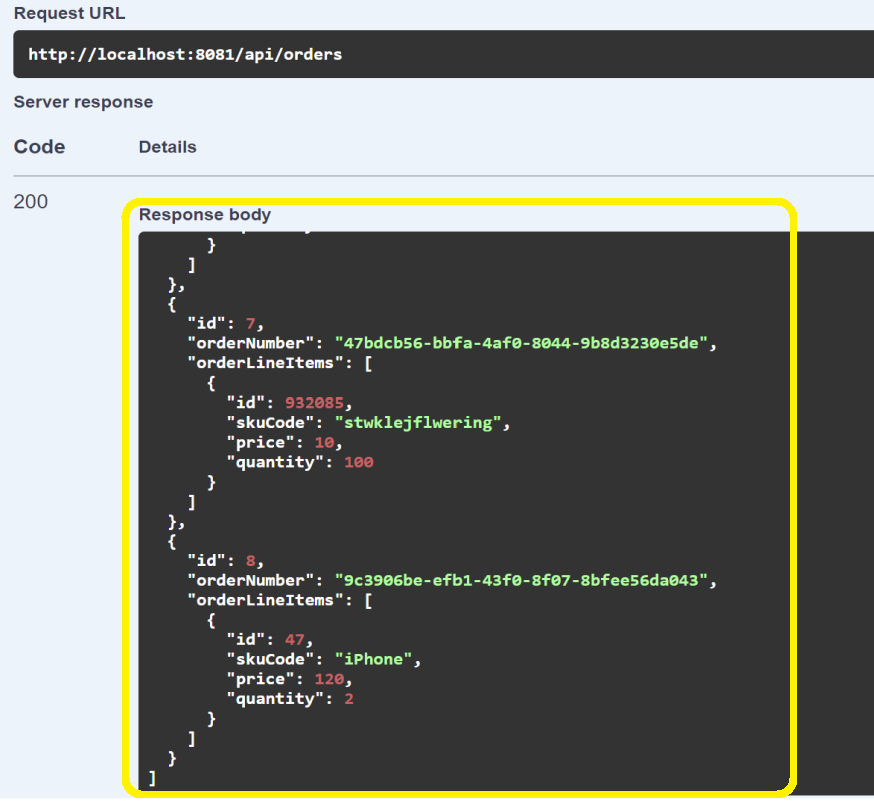
****

**2.c> GET /api/orders - Get a list of all orders**

Click execute to get list of all orders.



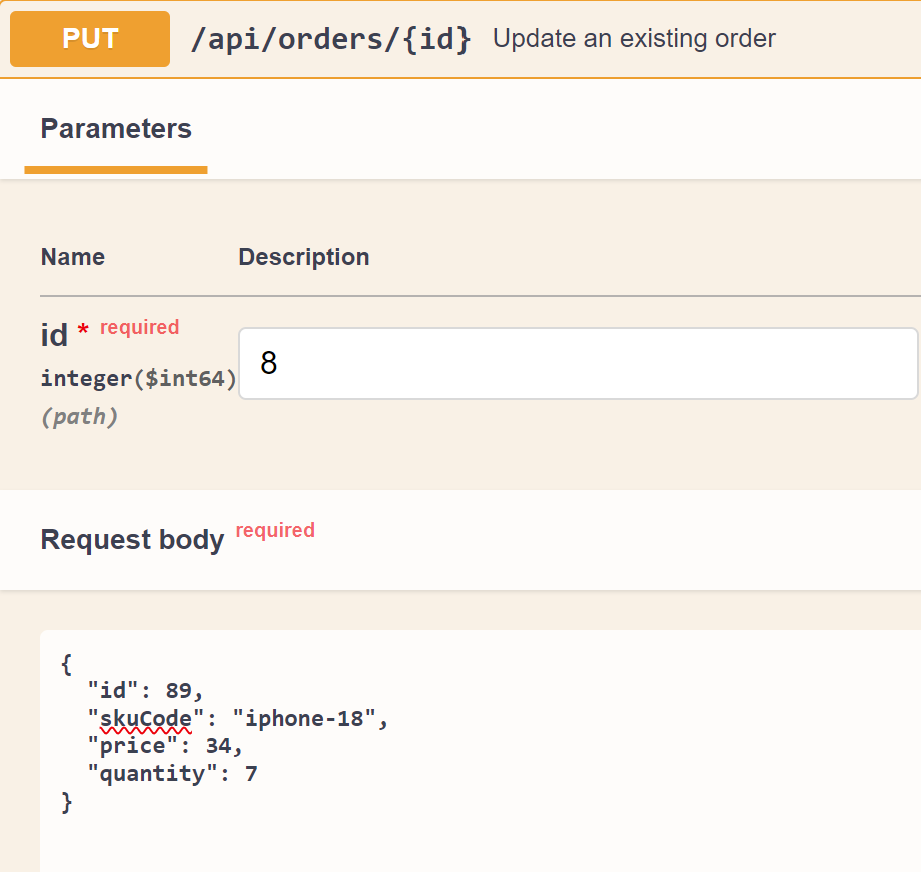
**Response:** We will get a list of all orders which has been successfully created till now.

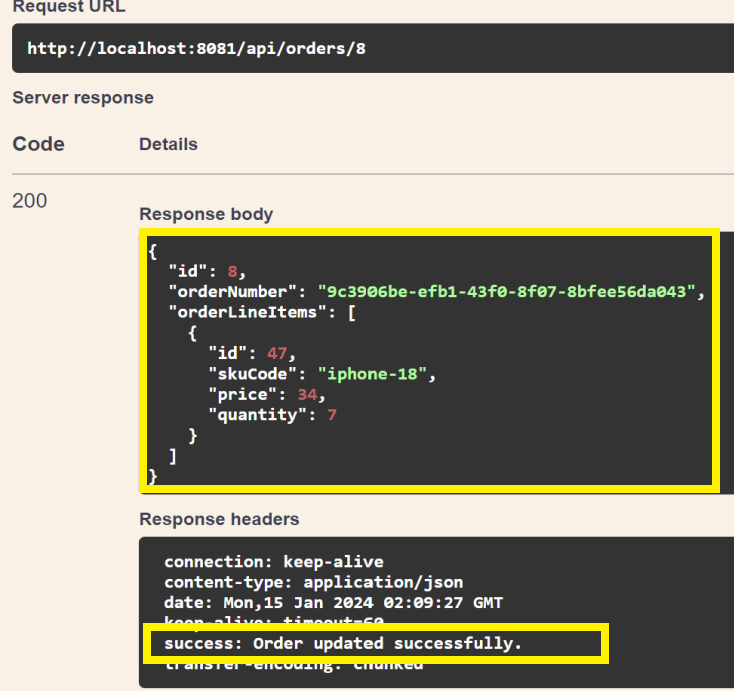


**2.d> PUT /api/orders/{id} - Update an existing order**

Update an existing order based on Order-ID

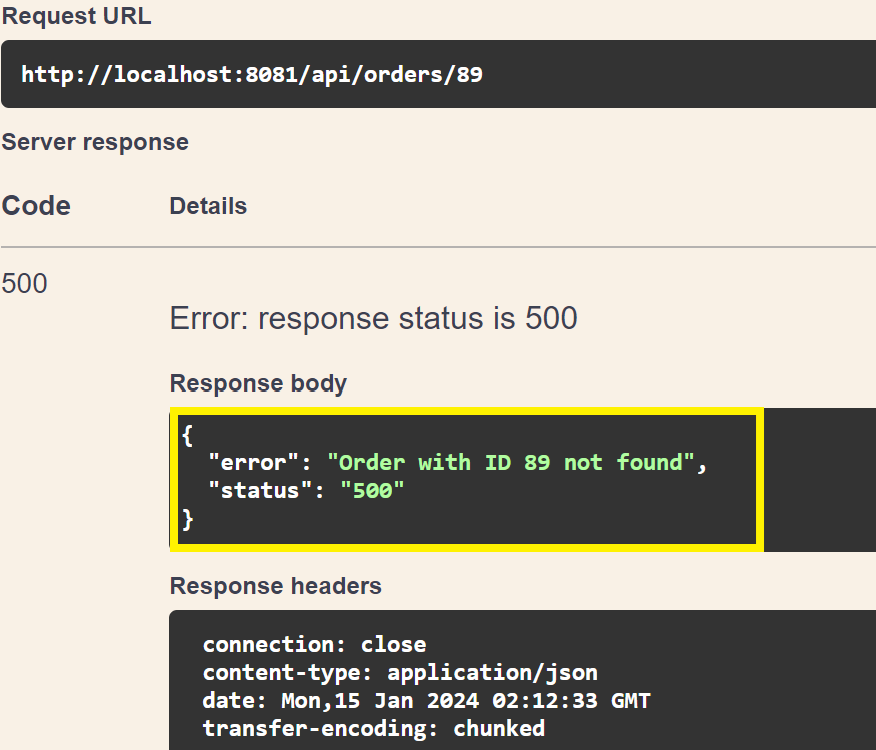
Ex: I will update the Order with ID: 8 as below

**  
Response**: Once the Order has been updated, we can view Success message   
“Order Updated Successfully”

****

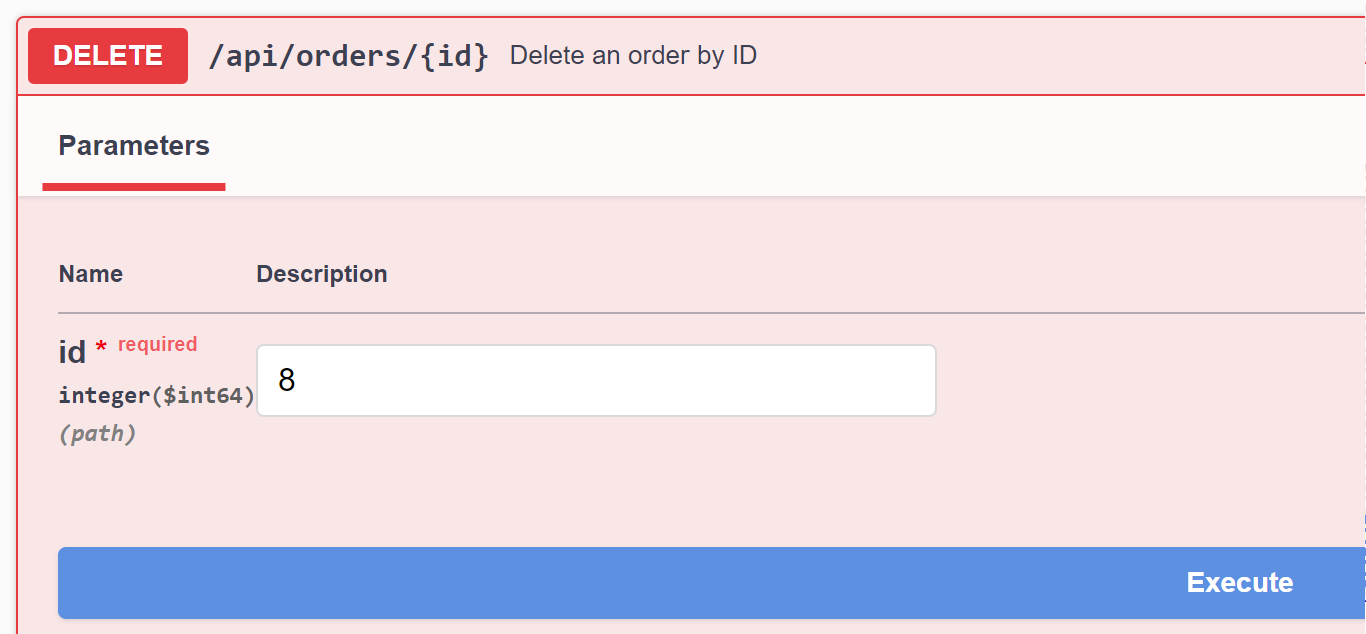
**Error Handling Scenario: Updating the Order ID which has not been created.**

In case the Order is not found, we get an error “Order with ID not found"  
Let say if I want to Update Order ID: 89 which has not been created/present in database  
We Get an error as shown below

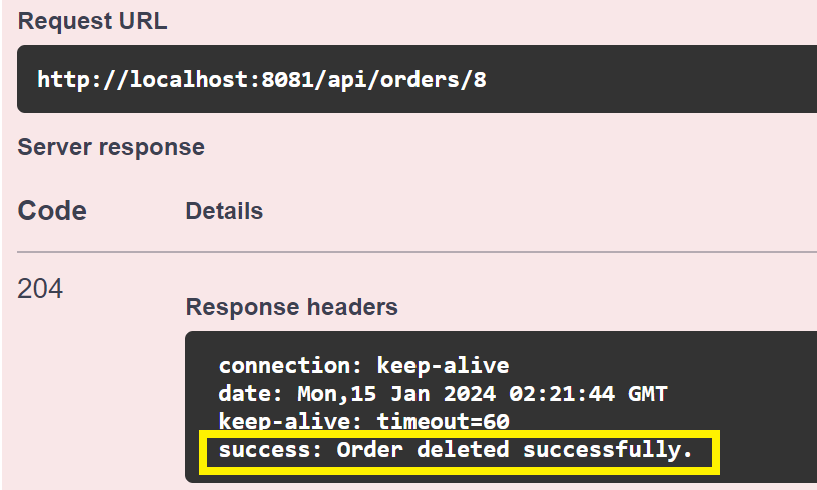


**2.f> DELETE /api/orders/{id} - Delete an order by ID**

Delete the Order by providing Order-Id and click execute



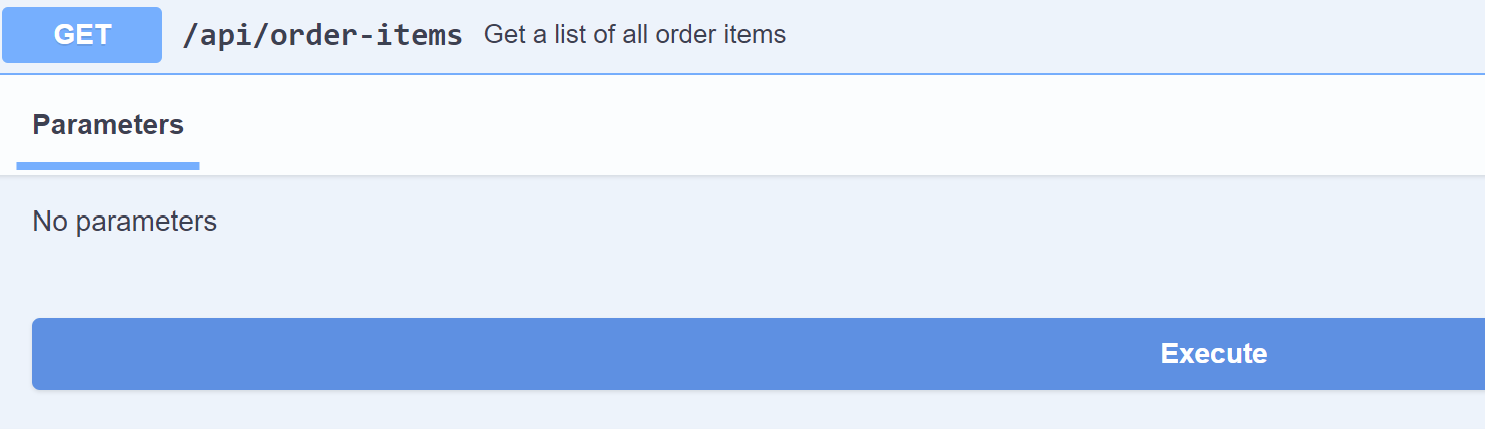
**Response**: Once the Order has been deleted, we can view Success message   
“Order has been deleted Successfully”



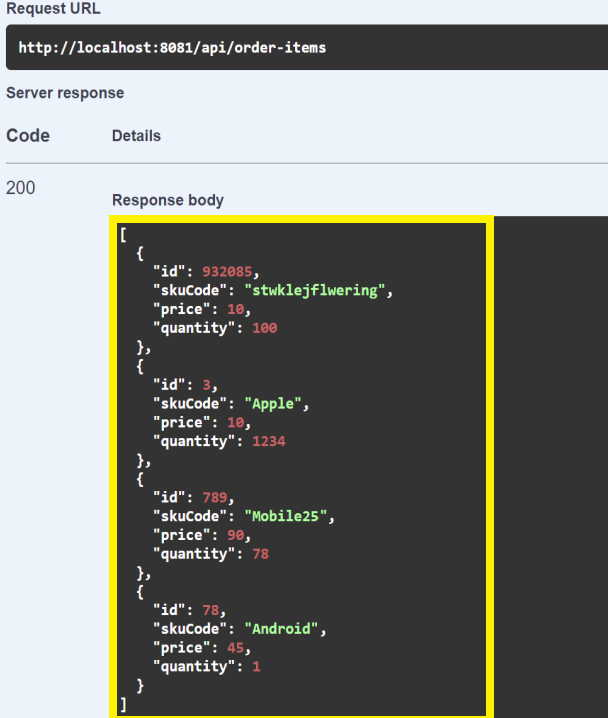
Now the Order has been Successfully Deleted.

**5.3. Testing OrderItem API Calls:**

1.a> GET /api/order-items - Get a list of all order items

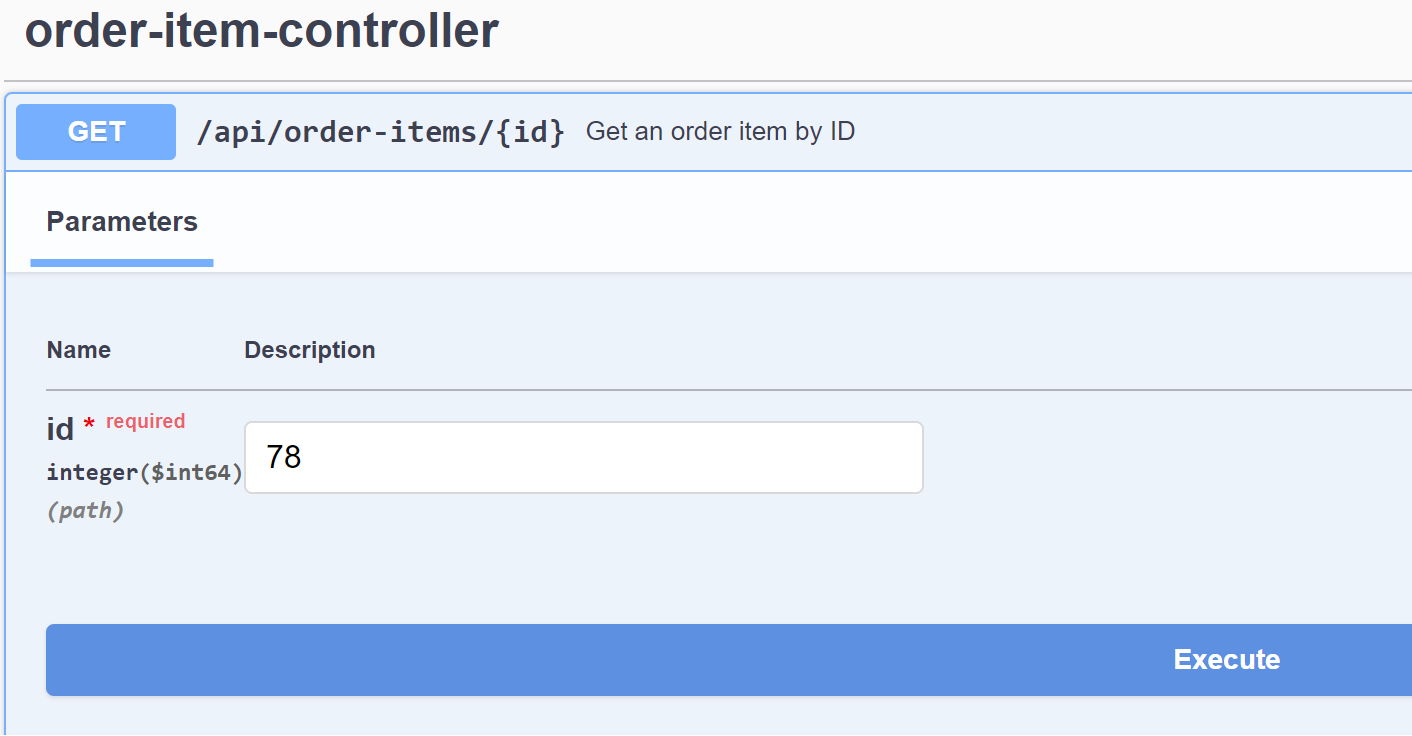


Now we will get all the list of Order-Items

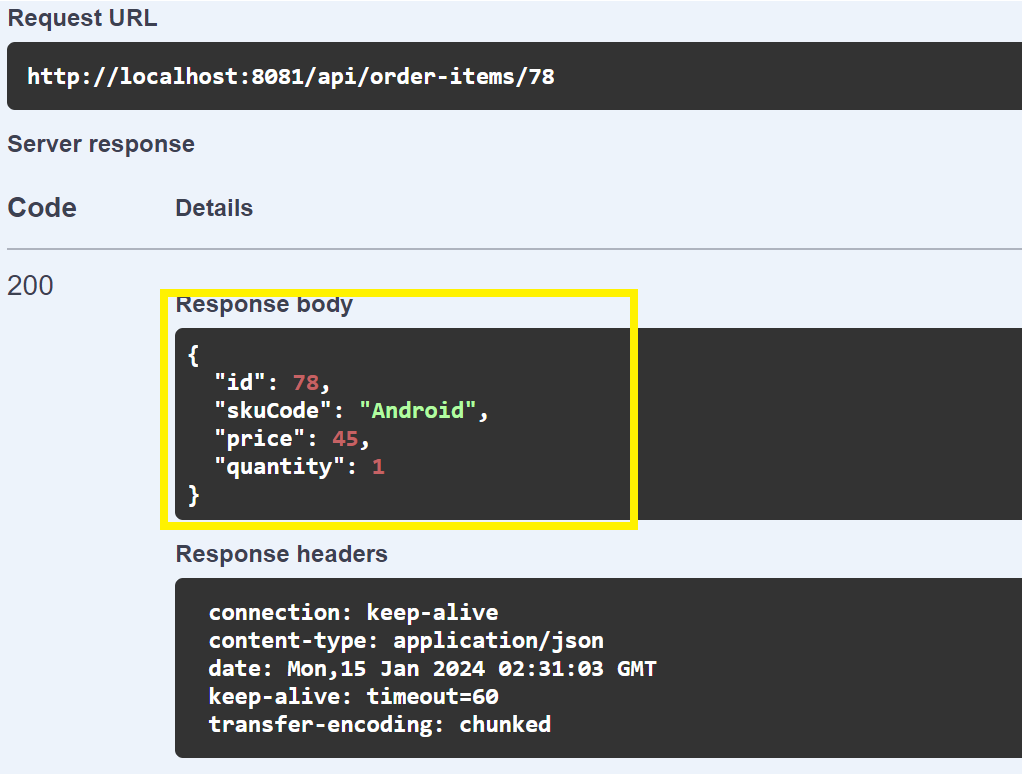


3.2> GET /api/order-items/{id} - Get an order item by ID

Enter the Order item ID to fetch details of that particular query

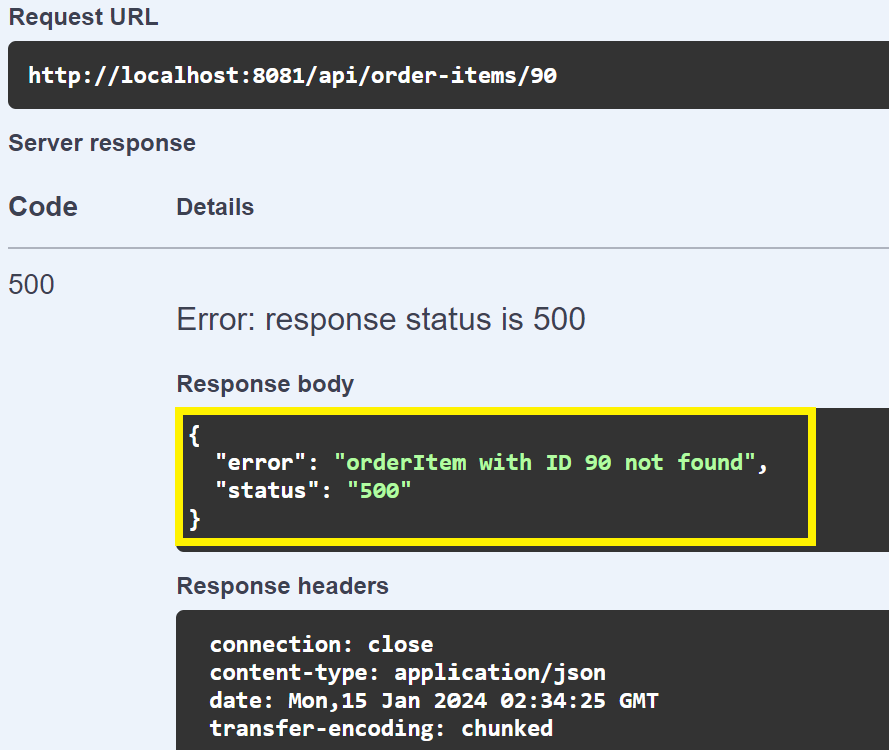


**Response**We will get the details of that particular OrderItem.



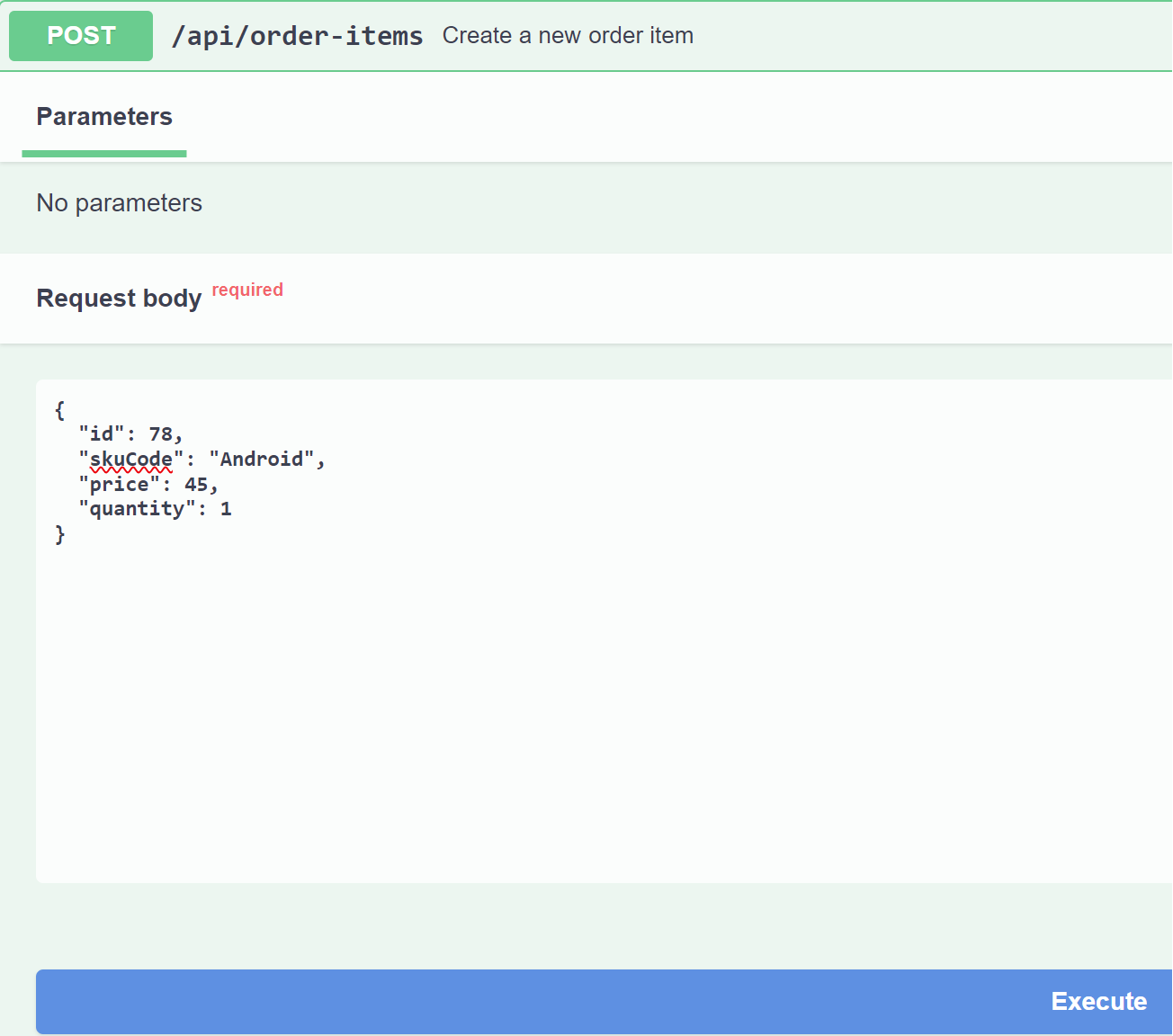
**Error Handling Scenario: Query the OrderItem ID which has not been created.**

In case the OrderItem is not found, we get an error “OrderItem with ID not found"  
Let say if I want to Query OrderItem ID: 90 which has not been created/present in database,  
We get an error as shown below

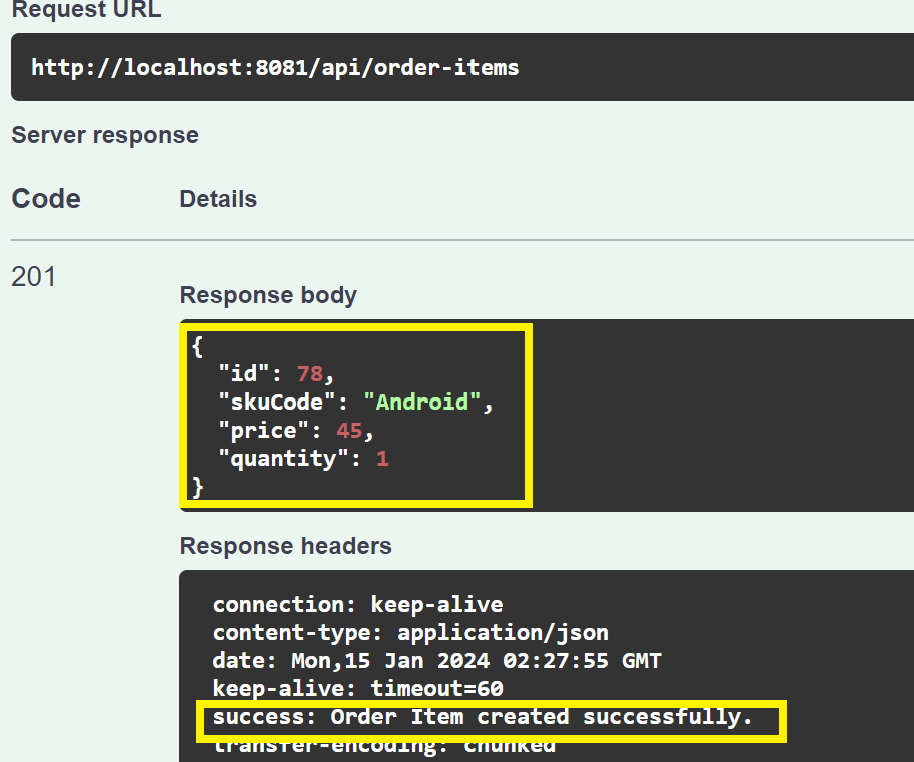


3.3> POST /api/order-items - Create a new order item   
To create a new order Item, enter the fields in the request body and click execute,

**Request Body**: { "id": 78,  
 "skuCode": "Android",  
 "price": 45,  
 "quantity": 1 }

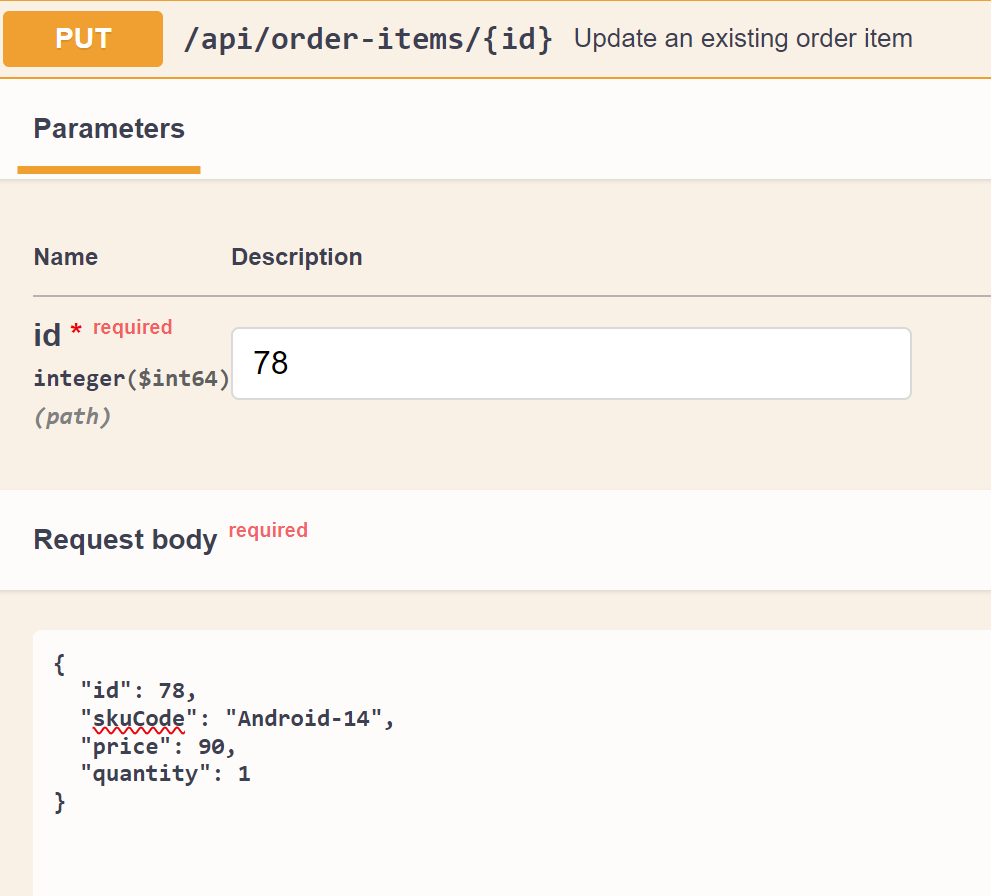


**Response of Create a new Order Item:**Once the OrderItem has been created, we can view the Success Message: **“Order Item created Successfully”**

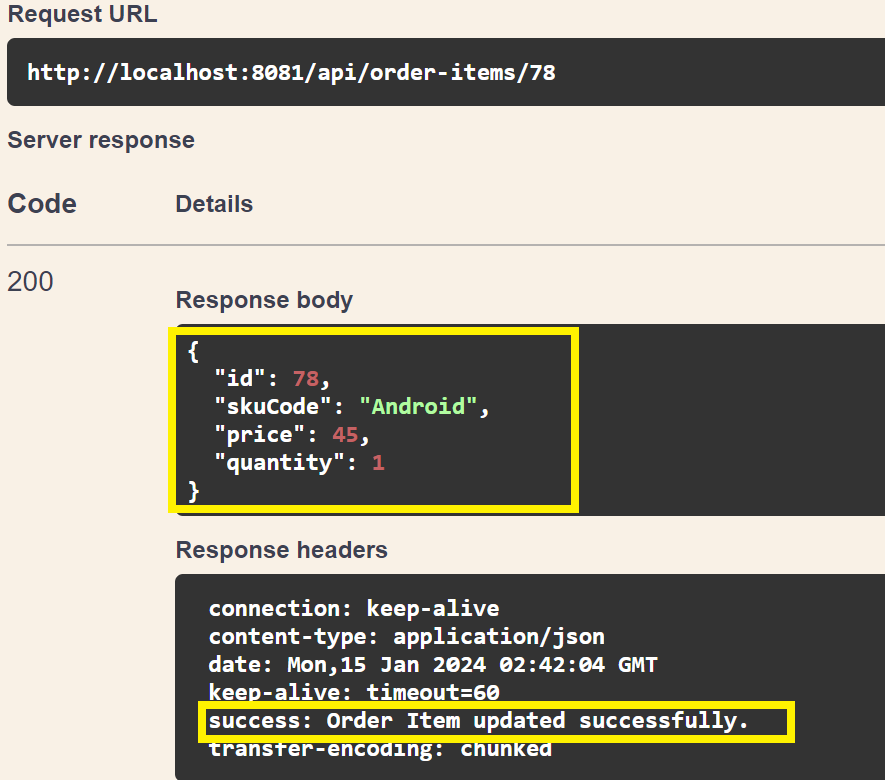
****

* 1. **PUT /api/order-items/{id} - Update an existing order item**

Update in Oder-Item based on OrderItem Id

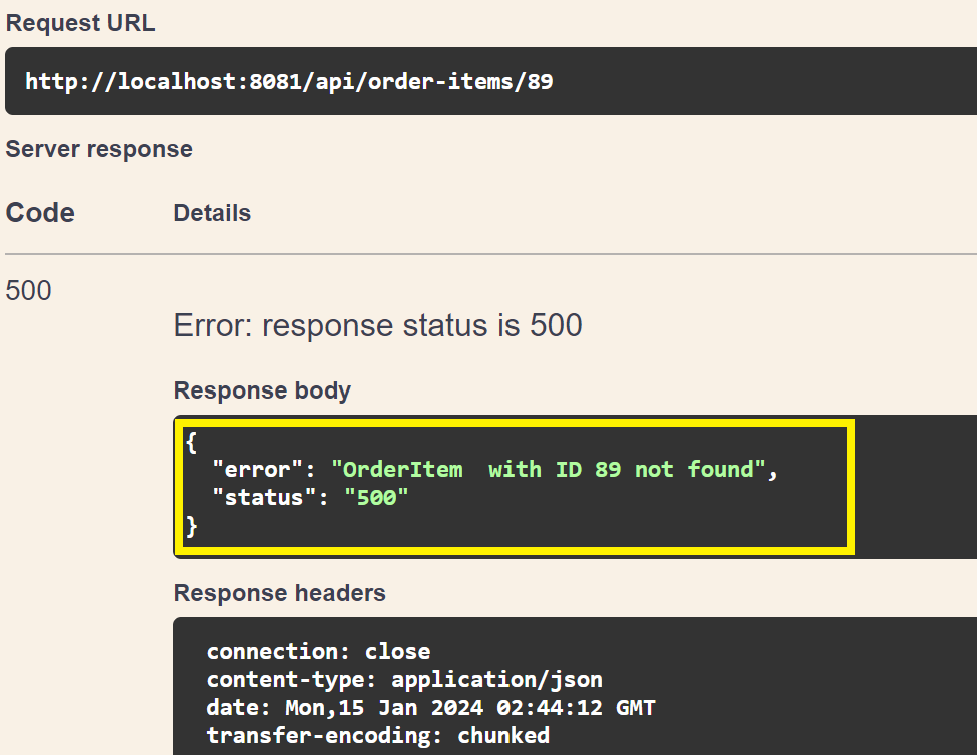


**Response**: Once the OrderItem has been updated, we can view Success message   
“OrderItem Updated Successfully”



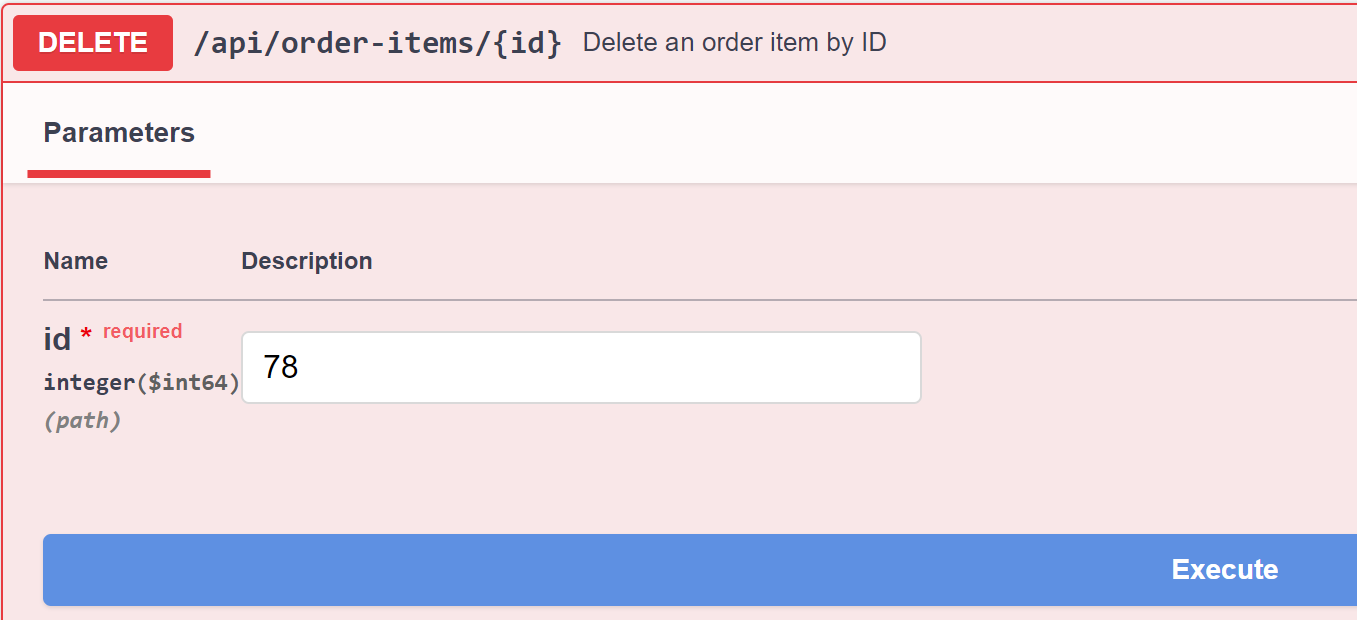
**Error Handling Scenario: Updating the OrderItem ID which has not been created.**

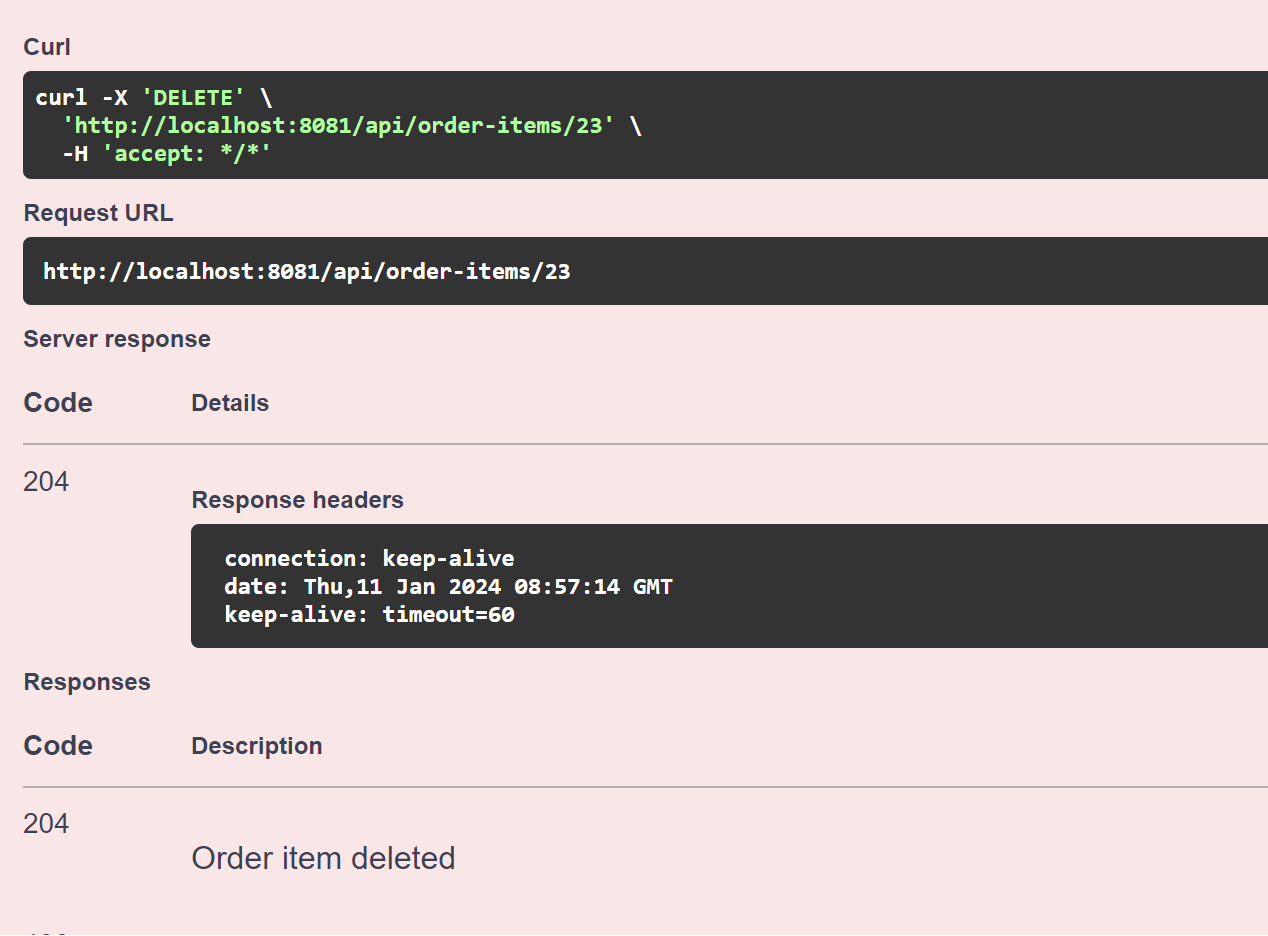
In case the OrderItem is not found, we get an error “OrderItem with ID not found"  
Let say if I want to Update OrderItem ID: 89 which has not been created/present in database  
We Get an error as shown below



* 1. DELETE /api/order-items/{id} - Delete an order item by ID

Delete the Order by providing OrderItem-Id and click execute

  
**Response:** Order-Item has been deleted successfully.



1. **Testing via JUnit and Mockito**

**6.1 Testing Overview:**

Test cases for this Spring Boot application have been written using a combination of JUnit 5 and Mockito, widely used testing frameworks in the Java ecosystem**.**

**Key Components:**

**JUnit 5:**

* JUnit 5 is the latest version of the JUnit testing framework.
* It provides annotations to define test methods, test lifecycle methods, and assertions for validating expected outcomes.

**Mockito:**

* Mockito is a mocking framework for Java that allows the creation of mock objects.
* Mock objects are used to isolate and test specific components of the application independently.

**Testing Scenarios:**

**Unit Testing:**

JUnit 5 is employed for unit testing individual components (classes, methods) in isolation.

Test cases verify the correctness of business logic and individual functionalities.

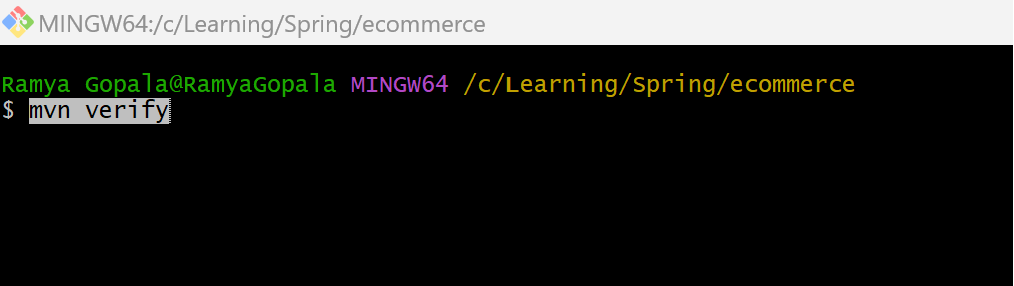
**Integration Testing:**

Mockito is utilized for creating mock objects to simulate the behaviour of dependencies.

Integration tests focus on interactions between components, ensuring they work together as expected.

**6.2. Steps to Testing Scenarios for Ecommerce Application:**

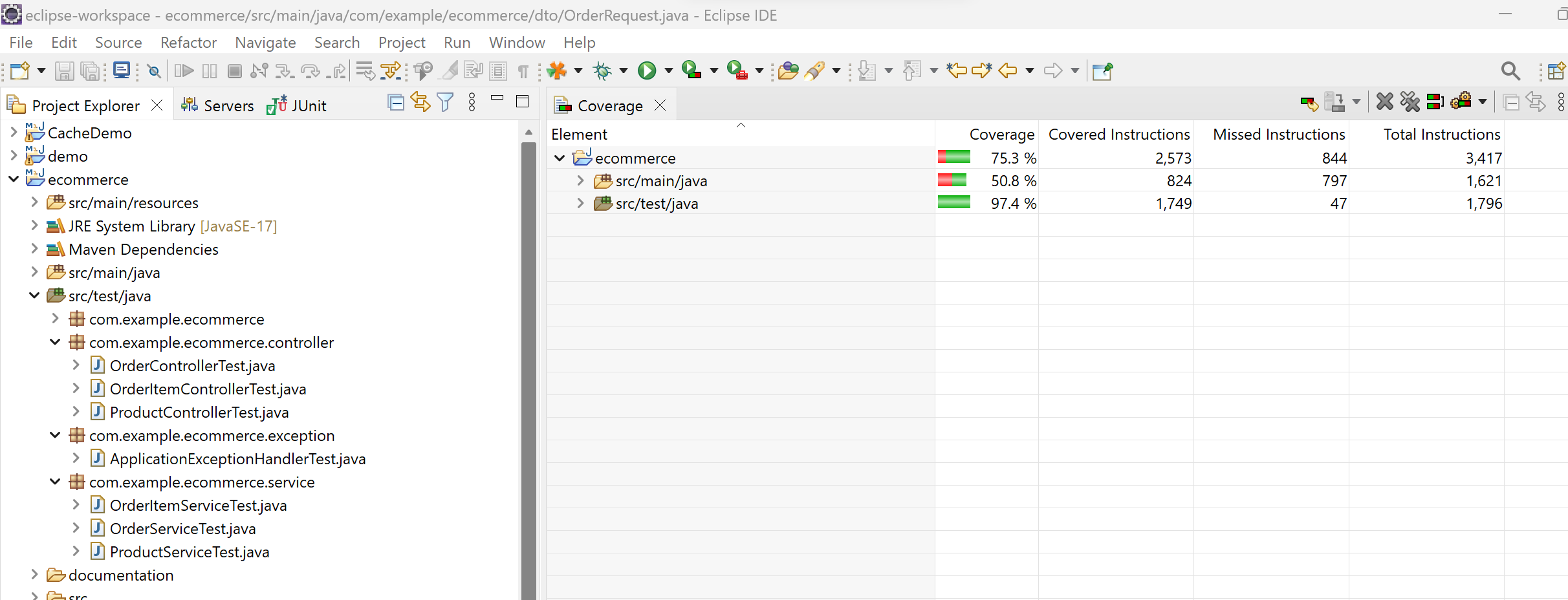
**Step1> Navigate to your Project Folder and run below Command**

****

We can view the test coverage

****

**I have covered 76% of test Coverage**

****

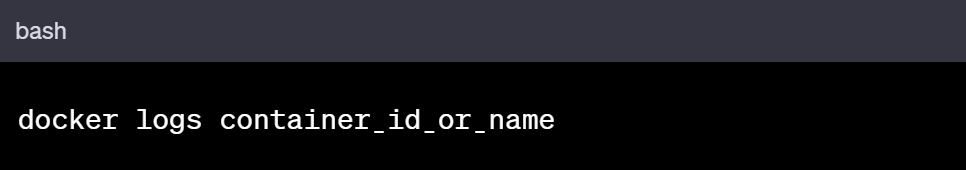
**Alternatively, we can inspect Junit test case logs via docker compose**🡺 Inspect Specific Containers

I have multiple services defined in my docker-compose.yml file, and you want to inspect the logs of a specific service, you can use the container name or ID. First, list the running containers:

Identify the container ID or name associated with your application, and then use the following command to view its logs:



Replace container\_id\_or\_name with the actual ID or name of your container.



By following these steps, you can view the Spring Boot JUnit test case output when running your application and tests via Docker Compose. Adjust the commands based on your specific setup and configurations.