E-commerce Backend Documentation

Developed by Mohamad Lakkis

November 30, 2024

Contents

1 Overview

The E-commerce Backend project consists of four independent services running on Docker containers, each assigned a specific port. These services communicate with a shared PostgreSQL database to provide functionalities such as customer management, inventory management, sales processing, and review handling. *Note: The GitHub Repository for this project is available at: https://github.com/mohamadlakkis/Ecommerce-BackEnd*

2 Services and APIs (This will cover points 1 to 6 from the requirements)

1. Note: Here you can find general details on each API of each servive, to see the examples and comments as requested in point 5, check the file "/api_endpoints.md" which is available in each sub-diretory of each service
2. Note: I chose to use as my database "PostgreSQL" as an image from docker hub which is running ona dedicated container port: 5432, and for each service I have created a seperate table in the database. The details of the tables are also available in the following sections
3. Note: Each service is running on a different port, and all of the necessary details are available in the following sections, BUT if you want to see the composed and seperate docker files, they are avilable in "Dockerfile" in each sub-directory for each service. As for the docker-compose.yml you can find it in the parent directory of the project's folder: "docker-compose.yml"

2.1 Customers Service

2.1.1 Description

Manages customer information, including registration, account updates, and wallet management.

2.1.2 APIs

API	Method	Description
Register Customer	POST	Registers a new customer.
Delete Customer	DELETE	Deletes a customer by username.
Update Customer Info	PATCH	Updates customer information.
Get All Customers	GET	Retrieves all customer records.
Get Customer by Username	GET	Retrieves details of a specific customer.
Charge Wallet	PUT	Adds funds to the customer's wallet.
Deduct Wallet Funds	PUT	Deducts funds from the customer's wallet.

2.1.3 Running Port

Port: **5001**

2.1.4 Database Structure

Field	Type	Description
id	SERIAL (PK)	Unique identifier for the customer.
full_name	VARCHAR(100)	Full name of the customer.
username	VARCHAR(50)	Unique username.
password	VARCHAR(100)	Encrypted password.
age	INTEGER	Age of the customer.
address	VARCHAR(200)	Address of the customer.
gender	VARCHAR(10)	Gender of the customer.
marital_status	VARCHAR(20)	Marital status of the customer.
wallet_balance	NUMERIC	Available funds in the customer's wallet.

2.1.5 Postman Testing

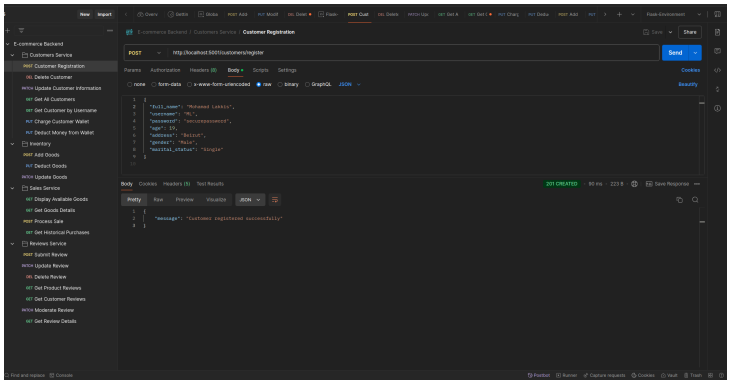

Figure 1: Customer registration

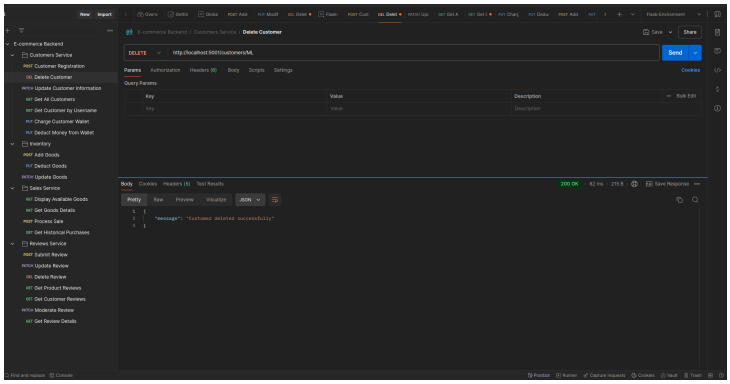

Figure 2: Delete customer

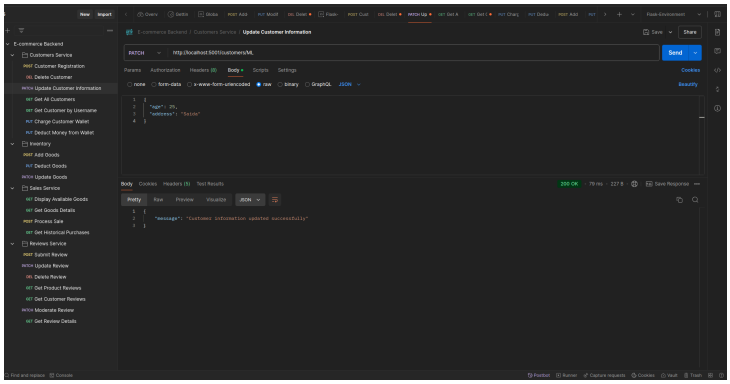

Figure 3: Update customer information

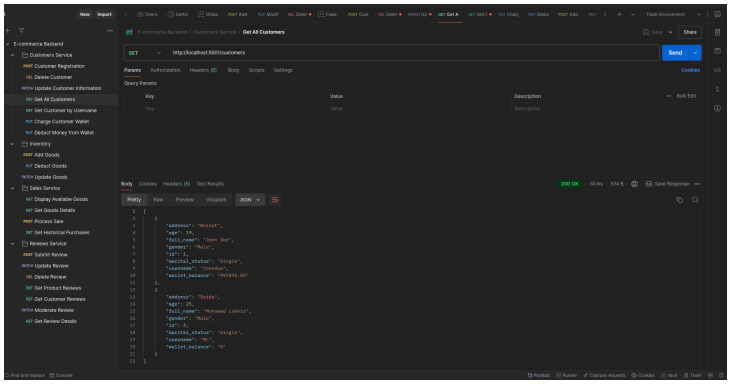

Figure 4: Get all customers

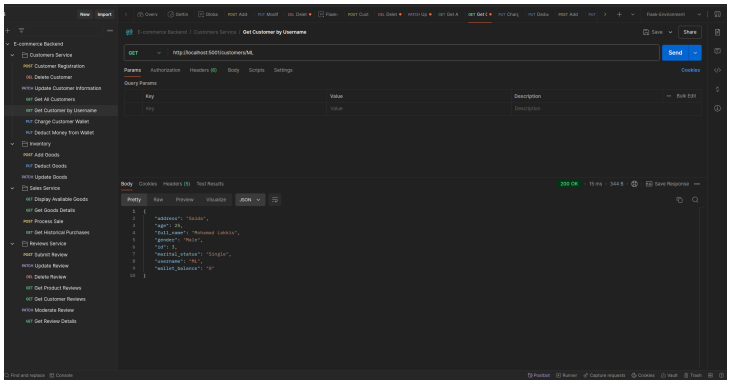

Figure 5: Get customer per username

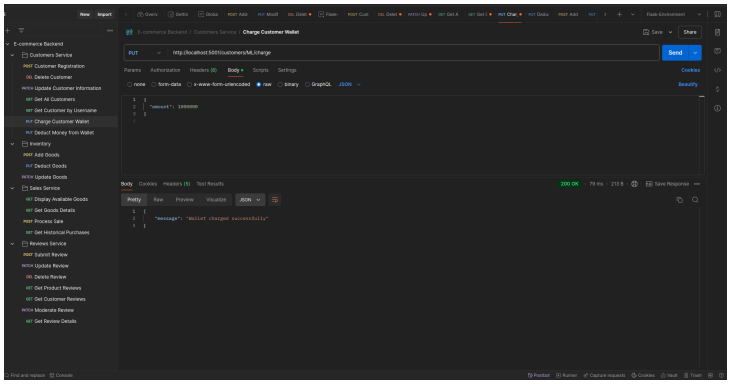

Figure 6: Charge customer wallet in dollars

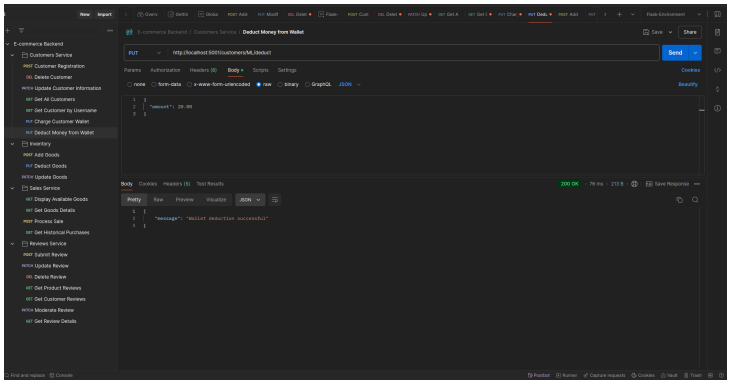

Figure 7: Deduct wallet funds

2.2 Inventory Service

2.2.1 Description

Handles inventory management, including adding, updating, and removing goods.

2.2.2 APIs

API	Method	Description
Add Goods	POST	Adds a new product to the inventory.
Deduct Goods	PUT	Removes items from stock.
Update Goods	PATCH	Updates product details.

2.2.3 Running Port

Port: **5002**

2.2.4 Database Structure

Field	Type	Description
id	SERIAL (PK)	Unique identifier for the product.

name	VARCHAR(100)	Name of the product.
category	VARCHAR(50)	Category of the product (e.g., food).
price	NUMERIC	Price per item.
description	TEXT	Detailed description of the product.
count	INTEGER	Available stock count.

2.2.5 Postman Testing

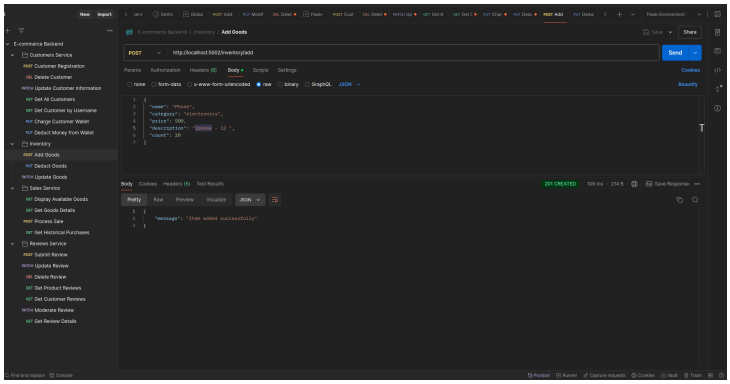

Figure 8: Add goods

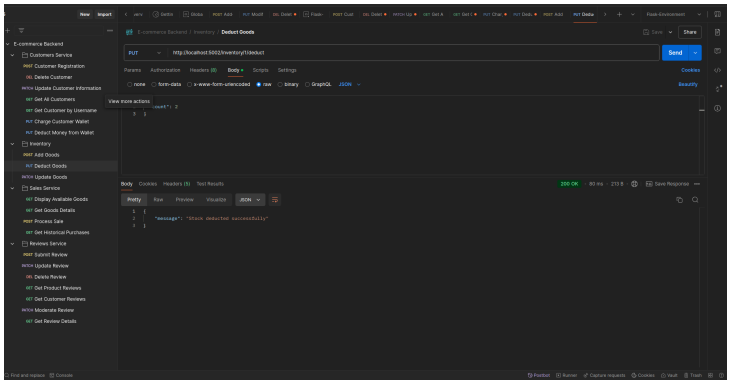

Figure 9: Deduct goods

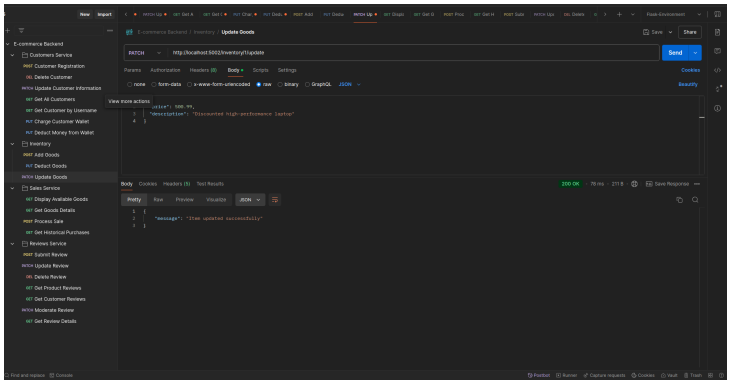

Figure 10: Update goods

2.3 Sales Service

2.3.1 Description

Manages the sale of goods, including processing purchases and tracking historical sales.

2.3.2 APIs

API	Method	Description
Display Goods	GET	Lists all available goods with prices.
Get Goods Details	GET	Retrieves detailed information about a product.
Process Sale	POST	Processes a sale by deducting stock and funds.
Get Purchase History	GET	Retrieves historical purchases for a customer.

2.3.3 Running Port

Port: **5003**

2.3.4 Database Structure

Field	Type	Description
id	SERIAL (PK)	Unique identifier for the sale.
customer_id	INTEGER (FK)	References the customer making the purchase.
good_id	INTEGER (FK)	References the product being purchased.
quantity	INTEGER	Number of items purchased.
total_price	NUMERIC	Total price of the purchase.
sale_date	TIMESTAMP	Date and time of the purchase.

2.3.5 Postman Testing

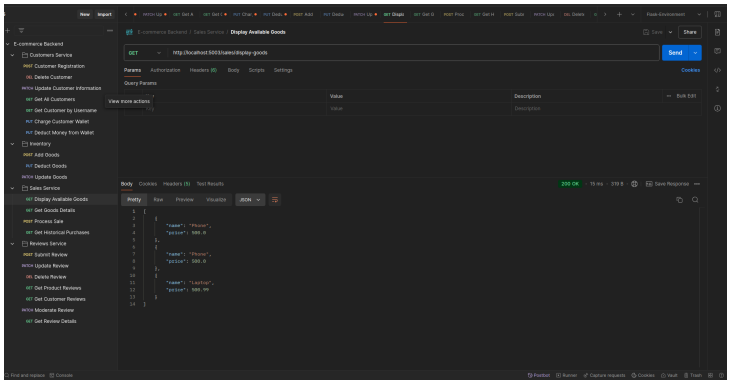

Figure 11: Display goods

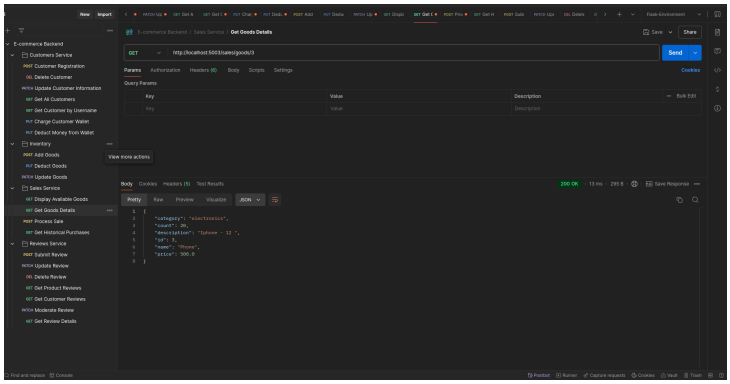

Figure 12: Get goods details

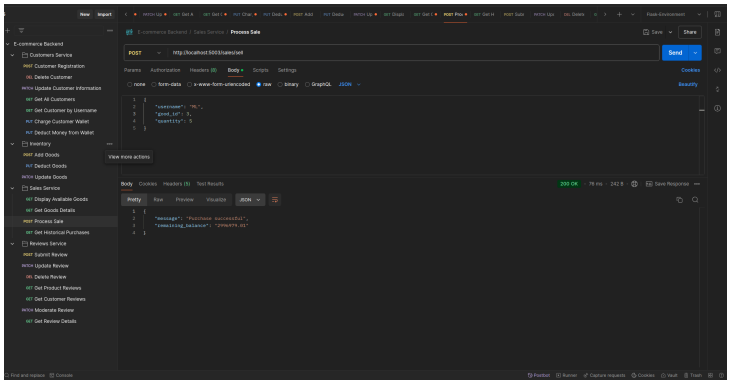

Figure 13: Process sale

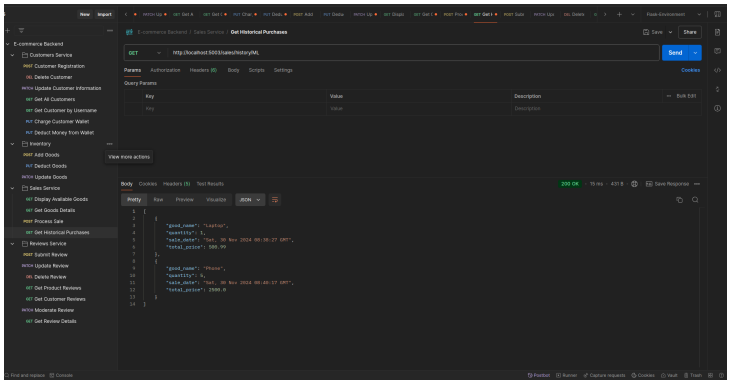

Figure 14: Get purchase history

2.4 Reviews Service

2.4.1 Description

Manages product reviews and ratings submitted by customers.

2.4.2 APIs

API	Method	Description
Submit Review	POST	Submits a review for a product.
Update Review	PATCH	Updates a specific review.
Delete Review	DELETE	Deletes a specific review.
Get Product Reviews	GET	Retrieves reviews for a specific product.
Get Customer Reviews	GET	Retrieves all reviews submitted by a customer.
Moderate Review	PATCH	Flags or approves a review.
Get Review Details	GET	Retrieves details of a specific review.

2.4.3 Running Port

Port: **5004**

2.4.4 Database Structure

Field	Type	Description
id	SERIAL (PK)	Unique identifier for the review.
customer_id	INTEGER (FK)	References the customer submitting the review.
product_id	INTEGER (FK)	References the reviewed product.
rating	INTEGER	Rating given to the product (1-5).
comment	TEXT	Customer's feedback.
status	VARCHAR(20)	Status of the review (e.g., pending).
review_date	TIMESTAMP	Date and time the review was submitted.

2.4.5 Postman Testing

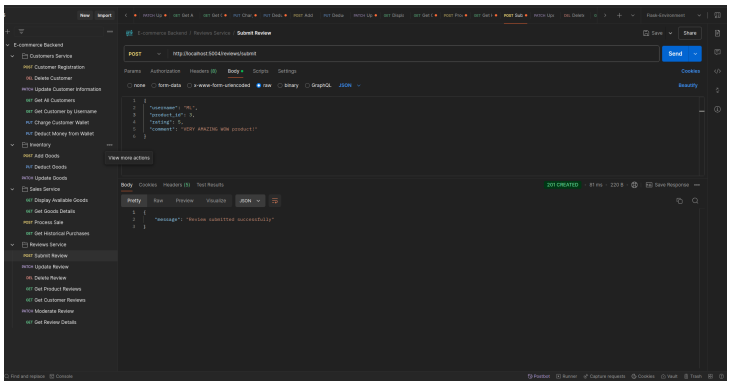

Figure 15: Submit review

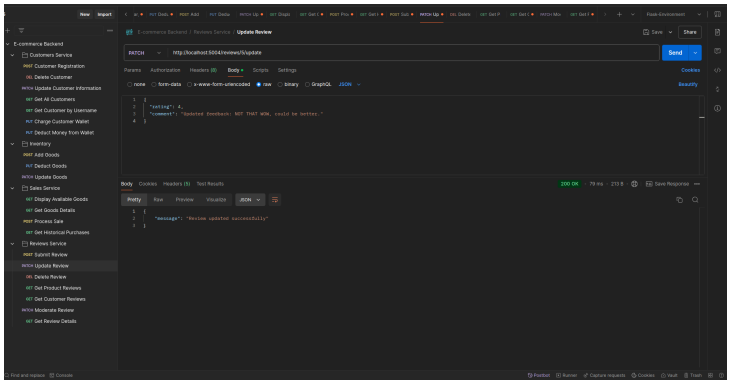

Figure 16: Update review

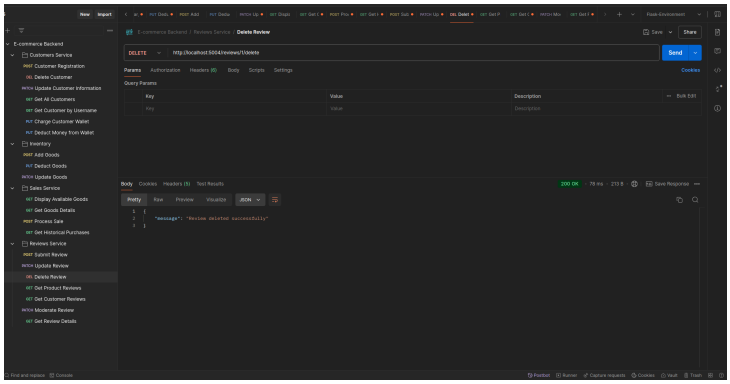

Figure 17: Delete review

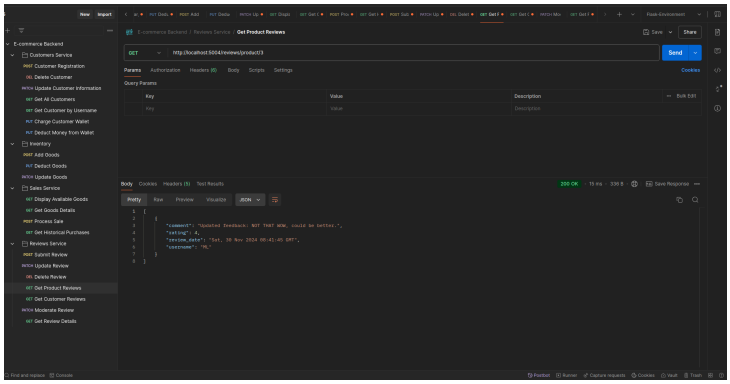

Figure 18: Get product reviews

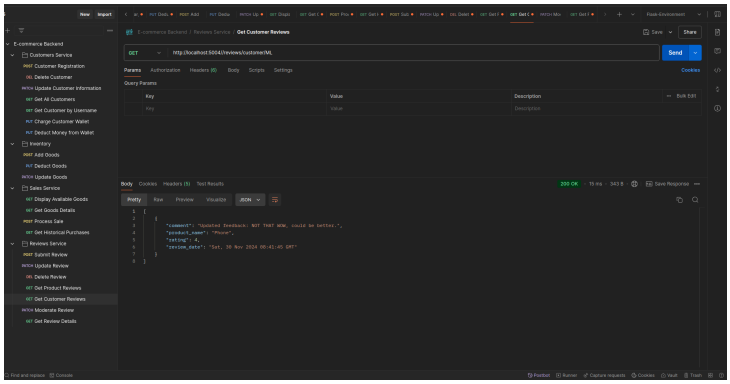

Figure 19: Get customer reviews

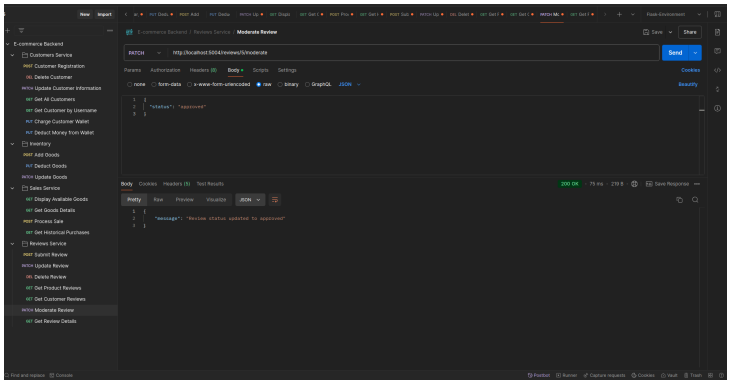

Figure 20: Moderate review

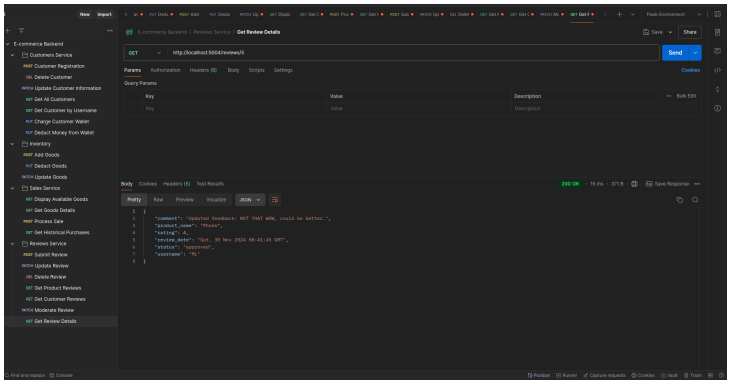

Figure 21: Get review details

3 Error Handling and Validation

3.1 Error Management

Errors in the system are managed to ensure smooth operations and meaningful responses to clients. The following strategies are implemented:

1. **HTTP Status Codes**

- **2xx Success Codes:** Indicate successful operations (e.g., `200 OK`, `201 Created`).
- **4xx Client Errors:** Inform the client of issues with their request:
  - `400 Bad Request:` Returned for missing or invalid fields in the request payload.
  - `404 Not Found:` Returned when the requested resource does not exist.
- **5xx Server Errors:** Catch unexpected errors during processing (e.g., database connection issues).

2. **Exception Handling**
   - **Database Exceptions:** Managed using `try-except` blocks to handle SQL errors and ensure proper rollbacks.
   - **Flask-Specific Exceptions:** Use Flask's built-in error handling to provide user-friendly error messages.
3. **Fallback Mechanism:** Unexpected errors return a generic `500 Internal Server Error` during production, with debug information enabled during development.

3.2 Validation

The system includes comprehensive validation to ensure data integrity and appropriate operations. Below are the implemented validations:

1. **Request Payload Validation:**
   - Each API validates incoming payloads to ensure required fields are present and properly formatted.
   - Examples include:
     - Customers Service: Ensures username uniqueness and verifies mandatory fields like `username`, `password`, and `age`.
     - Sales Service: Validates wallet balance and product availability before processing a sale.
     - Reviews Service: Checks that the `rating` is between 1 and 5.
2. **Database Constraints:**
   - Enforced at the schema level:
     - Primary Keys prevent duplicate entries.
     - Foreign Keys ensure relationships between tables are respected (e.g., `customer_id` in `sales_history` must exist in `customers` table).
     - Check Constraints validate numeric ranges (e.g., `rating` between 1 and 5).
3. **Error Response for Missing Resources:**
   - APIs return `404 Not Found` when:
     - A customer, product, or review cannot be found.

4. **Status Validation in Reviews:**
    - Only allows valid statuses (`approved`, `flagged`) in the moderation process.
5. **Flask-Level Validation:**
    - Enforces clean input, such as ensuring numeric IDs (e.g., `<int:review_id>`).

4 Testing (This will cover point 7 from the requirements)