**Assignment 2: SUSTech Merch Store**

**CS328 - Distributed and Cloud Computing**

**Marietta – 12112450**

**I. Environment Setup**

The environment setup is essential for developing, testing, and running the various services that make up the SUSTech Merch Store. These services include the API Service, DB Service, and Logging Service, which interact with each other using gRPC for communication and Kafka for log aggregation. Below are the procedures and tools used to implement each component:

- Tools Used: Python (Flask, gRPC), PostgreSQL, Kafka (log aggregator for scalable logging systems), Docker Compose (for orchestrating the services)

**-** Initial Configuration:

1. PostgreSQL & Kafka: Both services were pre-configured and deployed using Docker Compose. This setup makes it easier to manage service dependencies and network configurations.
2. .env Files: Sensitive credentials, such as database passwords and secret keys for JWT, are stored in .env files to manage environment variables securely.
3. Integration of gRPC, Kafka, and Flask:
   * gRPC is used for communication between services, allowing for efficient message serialization and remote procedure calls (RPC).
   * Kafka is used as a logging aggregator to collect logs from different services, ensuring scalability and non-blocking behavior.
   * Flask serves as the main API framework for handling incoming HTTP requests.

**-** Code Generation and Business Logic:

* **API Service (ClientStub.py)**:
  + Flask routes for handling requests related to products, users, and orders.
  + Interacts with the DB Service via gRPC for CRUD operations on the database.
  + The middleware is used to log incoming requests (@app.before\_request) and outgoing responses (@app.after\_request). These logs are sent to the Logging Service using Kafka for asynchronous processing.
* **DB Service (db\_service.py)**:
  + A gRPC server that handles CRUD operations on the PostgreSQL database.
  + Provides services for managing products, users, and orders.
* **Logging Service (local\_publisher.py)**:
  + Implements server-side streaming to handle log messages sent via gRPC.
  + Logs are published to Kafka, which serves as a log aggregator.
  + Logs are handled in a separate thread to ensure non-blocking operations.

**\*** Error Handling: Improved error logging with detailed traceback information. Consistent use of HTTP status codes.

**II. APIs Requiring Authentication**

For most user-related operations, authentication is required. JWT tokens are used for securing API calls after the user has logged in. Authentication Implementation:

1. Login & Registration: JWT tokens are generated after successful login or user registration.
2. Secure API Routes:
   * APIs like get-user, update-user, place-order, etc., require a valid JWT token in the Authorization header.
   * The token is validated for each request, ensuring that only authenticated users can access the resources.

Example API Calls Requiring Authentication:

* get-user
* deactivate-user
* update-user
* place-order
* get-order
* update-order
* cancel-order
* products (get & list

**JWT** is used for secure communication, and sensitive data like passwords are hashed using **bcrypt** before being stored in the database.

**III. Field Data Type Selection**

The field data types are selected to ensure consistency and security across the application. The selection process adheres to OpenAPI, Proto definitions, and the PostgreSQL schema.

Data Type Selection:

* User ID: Stored as INT in PostgreSQL, int32 in Proto, and integer in OpenAPI.
* Sensitive Data (e.g., passwords): Stored as VARCHAR with a defined length limit in PostgreSQL.
* Handling Decimal Types: PostgreSQL may return Decimal types, which are converted to float before being sent over gRPC or returned in the API response.

**IV. gRPC Message Encoding Analysis**

(For gRPC-based services, select an arbitrary Proto message from your definition and analyze how it is encoded into binary format. Use Protobuf to programmatically verify the encoding result.)

Proto Definition: used to send order details between the client and the server.

message OrderRequest {

int32 user\_id = 1;

int32 product\_id = 2;

int32 quantity = 3;

}

Encoding:

* Serialization: The fields are serialized with a tag (field number) and value, and then the whole message is encoded into a binary format.
* Verification: Using Protobuf's Python library to serialize an OrderRequest message:

“order\_request = OrderRequest(user\_id=1, product\_id=101, quantity=2)”

“serialized = order\_request.SerializeToString()”

This serialized binary format can then be transmitted over gRPC and deserialized at the receiver's end.

**V. Logging Service: Server-Side Streaming**

The Logging Service uses server-side streaming to handle continuous log messages sent from various services. Mechanism:

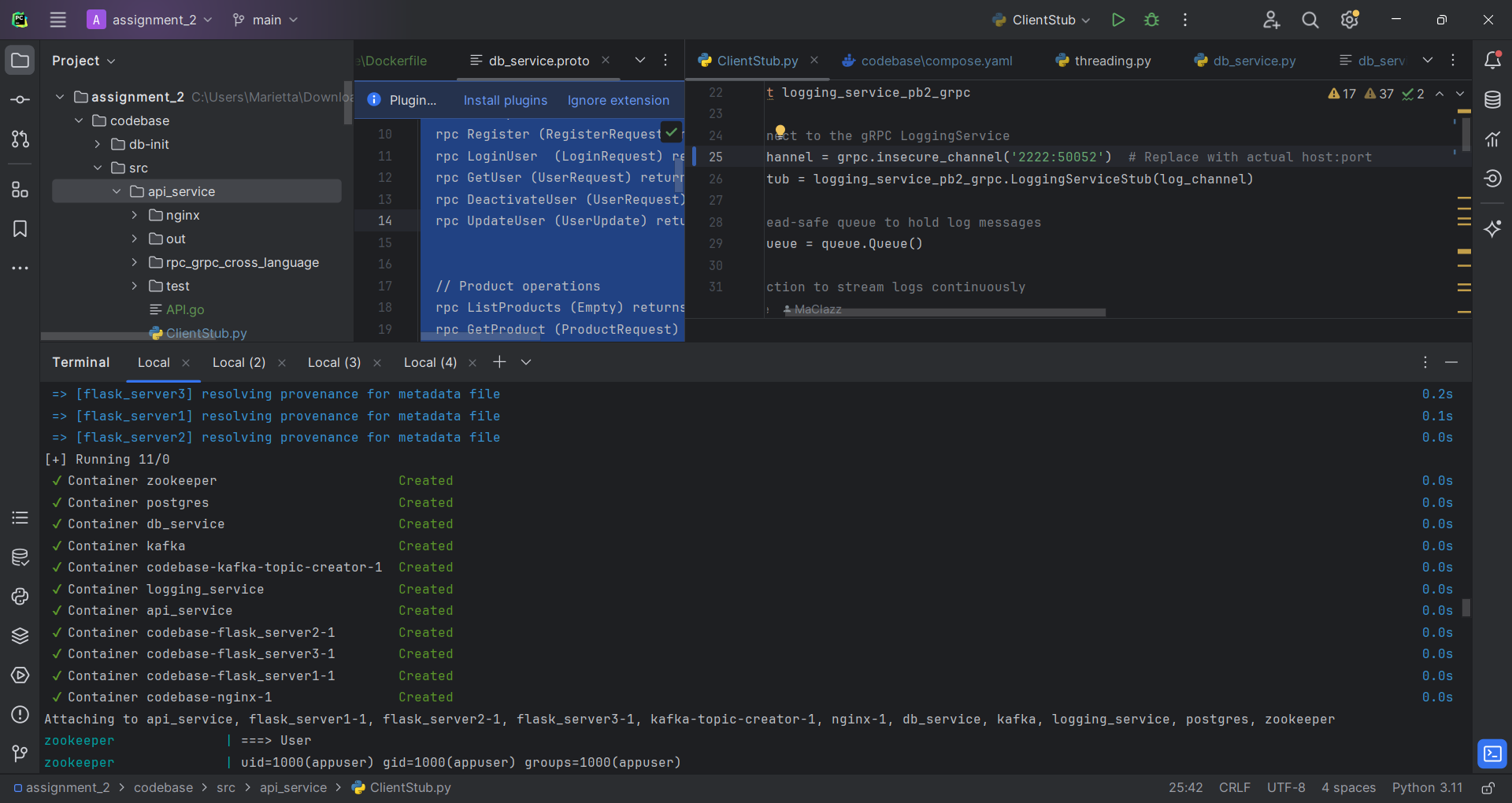
1. Server-side Streaming: The LogOperation RPC maintains a long-lived connection between the client (API/DB Service) and the server (Logging Service).
2. Log Messages: Logs are streamed in real-time as they are generated by the client services.
3. Kafka: The log messages are forwarded to Kafka, where they are aggregated and stored for monitoring or further processing.

**VI. Docker and Docker Compose Configuration**

The services are managed using Docker Compose to ensure smooth inter-service communication.

Configuration Details:

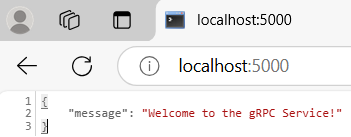
1. Networks: All services are part of the same network (app\_network), allowing them to communicate via service names (e.g., db\_service, logging\_service).
2. Dependencies: The depends\_on directive ensures that services like the API service start only after dependencies like the DB and Logging services are up.
3. Ports: Ports are mapped to allow external access to the services, e.g., 5000 for the Flask API, 50052 for gRPC.

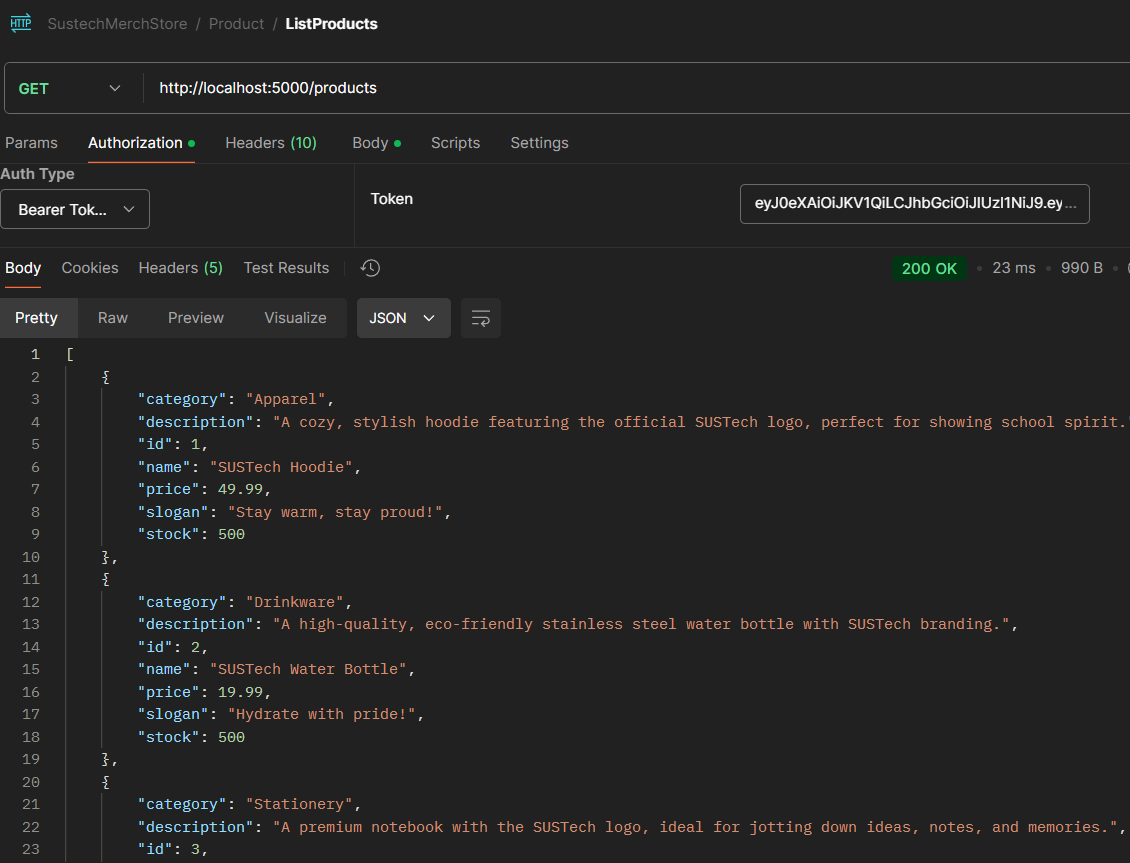
****

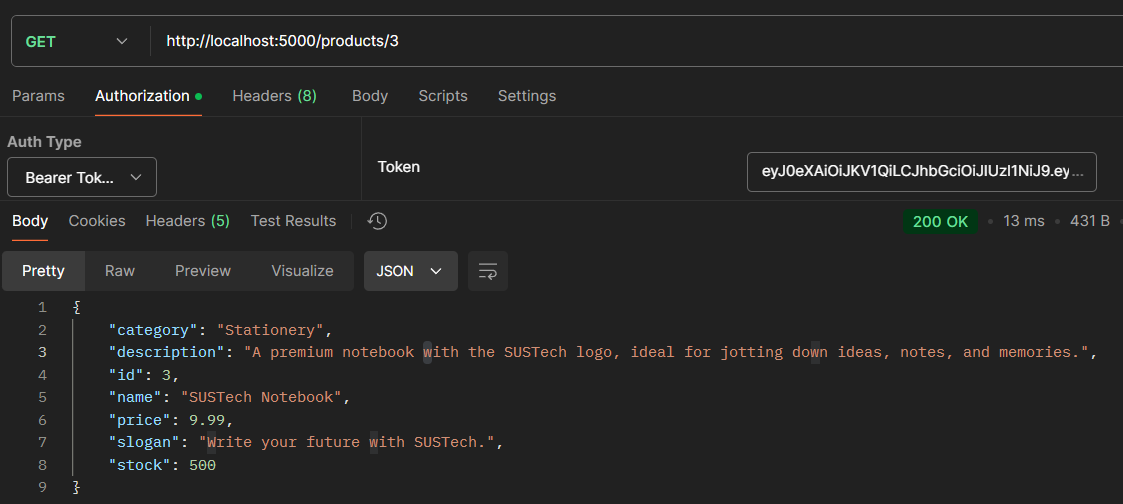
**VII. Experiment and Testing**

(How do you run the experiment? Which tool (i.e., cURL, Postman, Swagger UI) do you use to test your API Service? How do you monitor the log messages from the Kafka topic?)

Tools used for API testing are cURL for Command-line testing for basic operations &Postman for visual testing of RESTful APIs for simulating different user scenarios

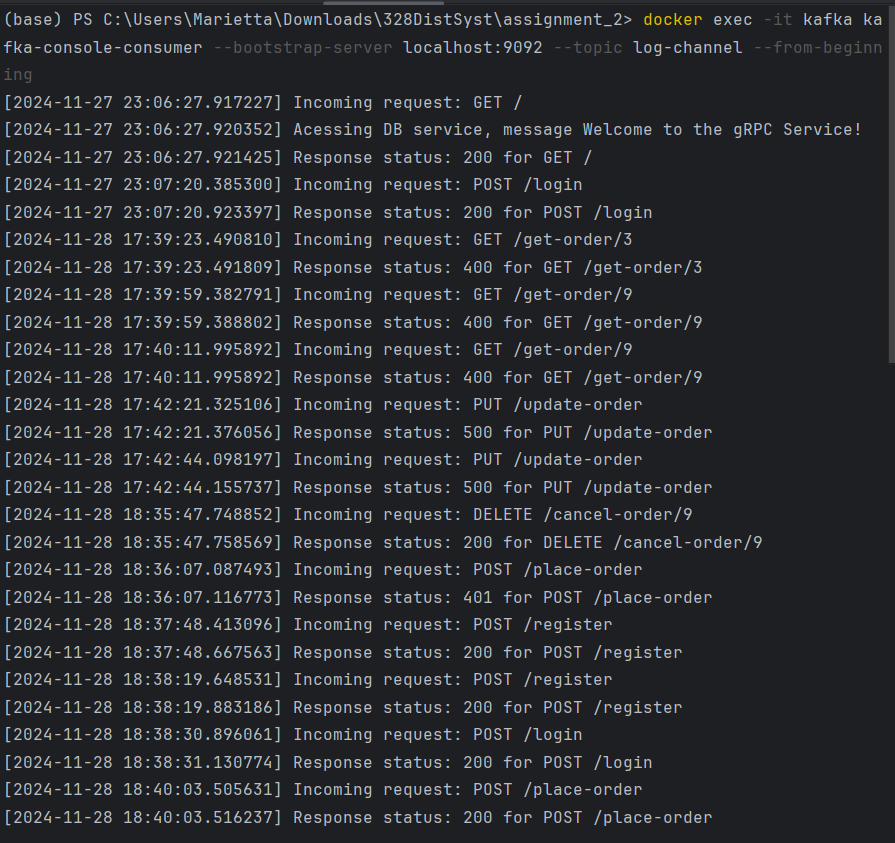
the other API calls will need JWT Authorization Bearer Token

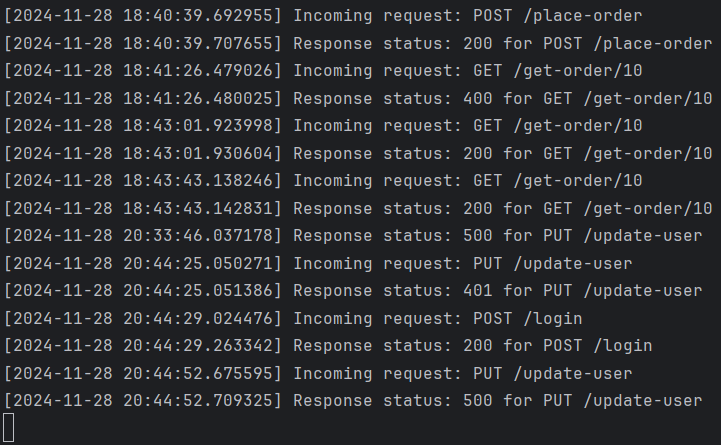




\*here only provide the result of product API call test (… the rest of Postman API call test screenshots are in the appendix)

Log Monitoring: Kafka CLI will be used monitor the log messages from the Kafka topic as it subscribed to topics to view real-time logs.

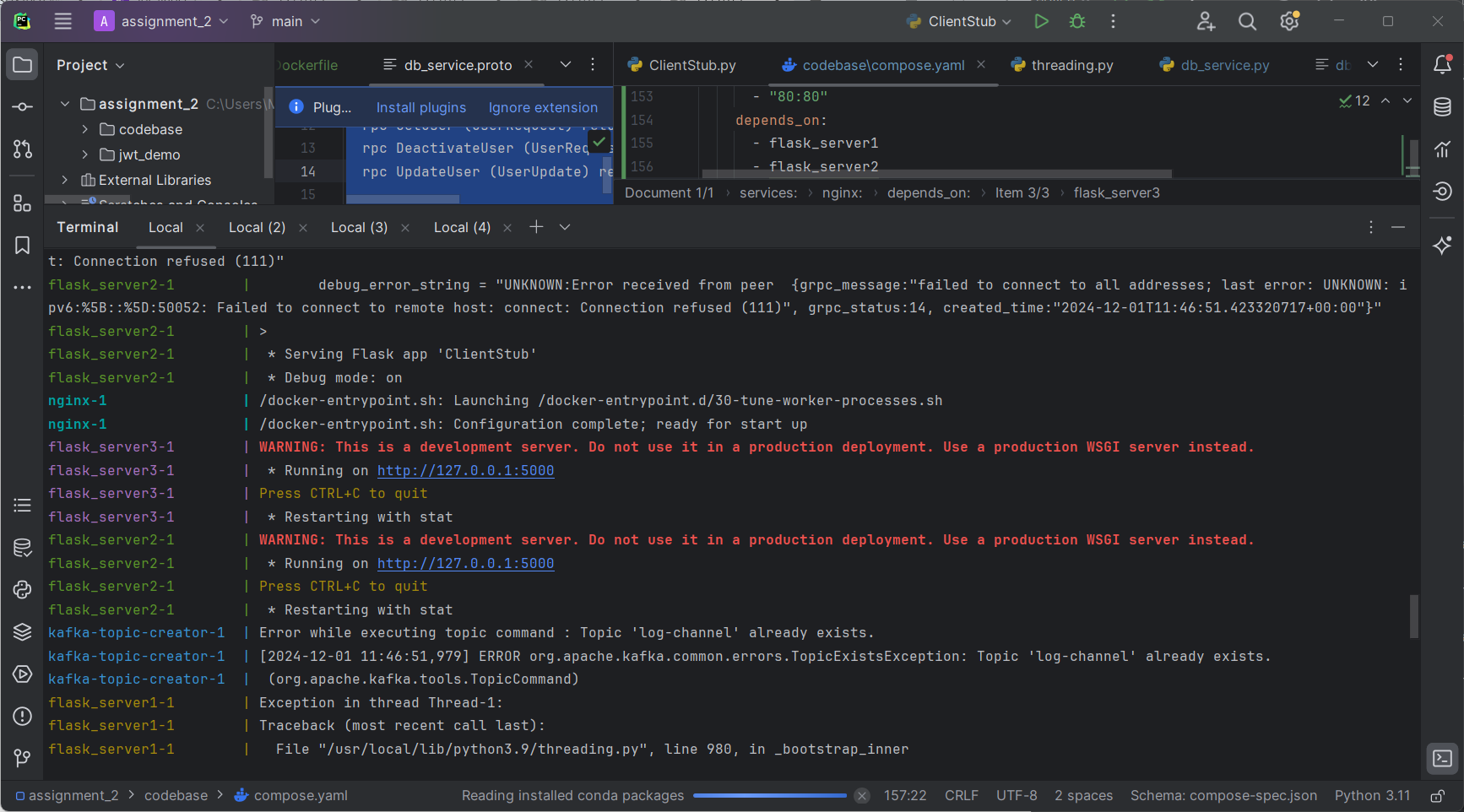
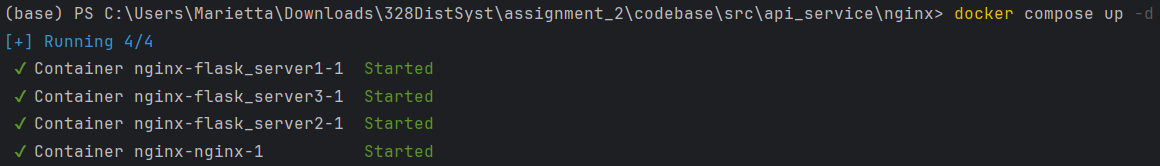
****

****

**VIII. Bonus Implementation**

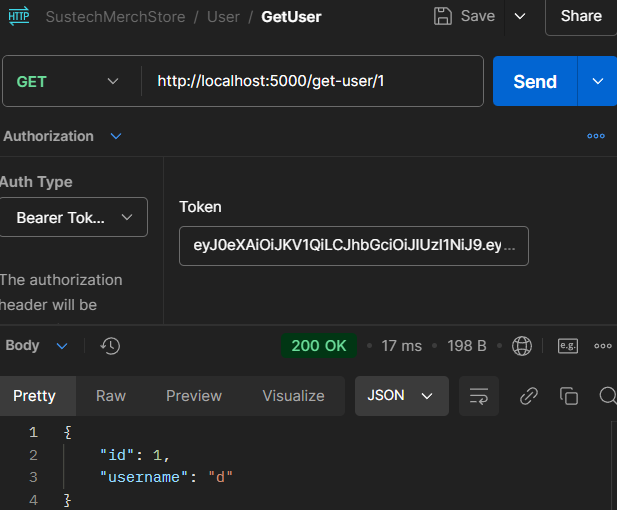
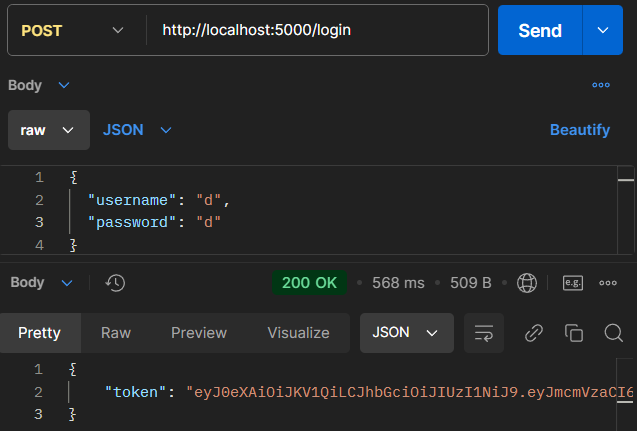
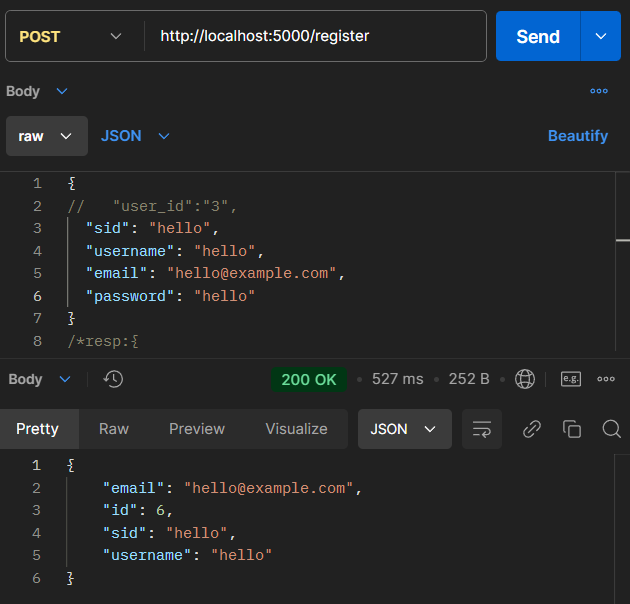
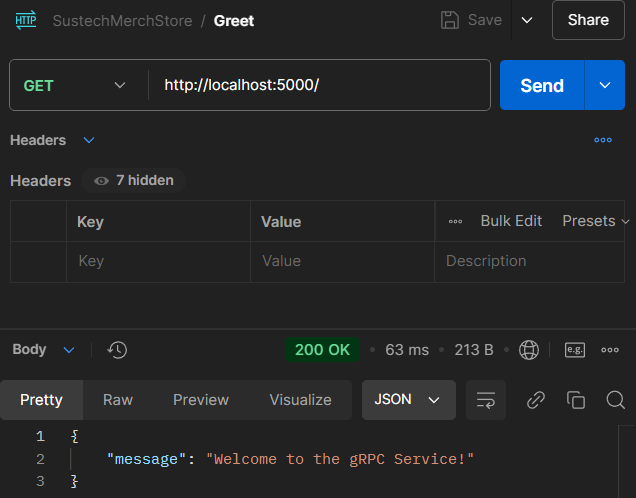
**Cross-Language Example:** Use different languages (i.e., Go andPython) for the gRPC client and gRPC server. gRPC Client in Go: A Go-based gRPC client was implemented to interact with the Python-based DB Service.

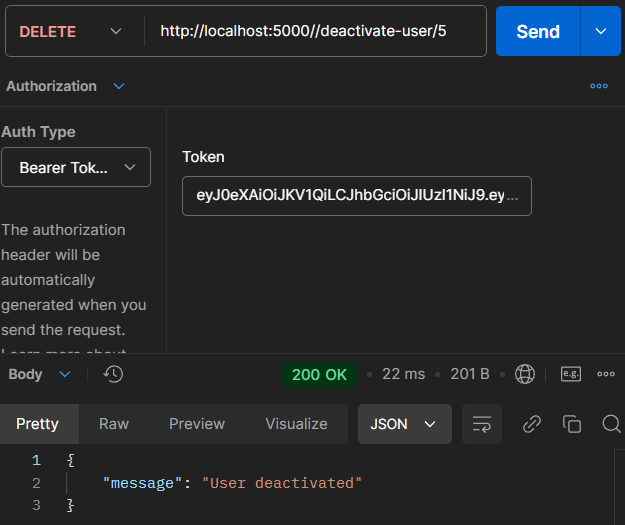
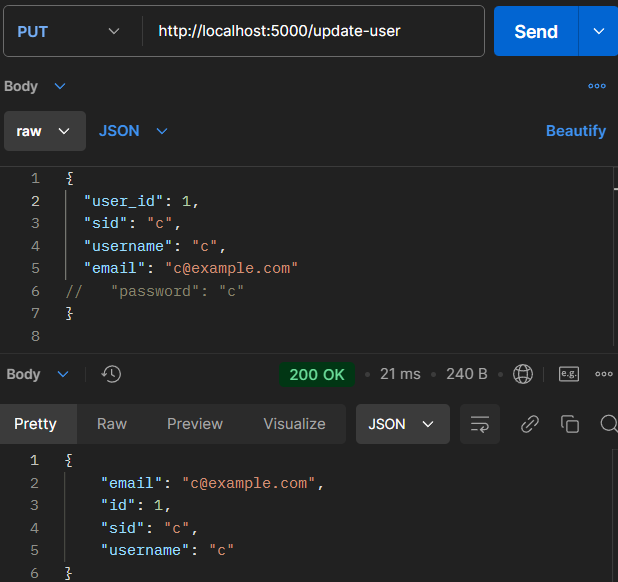
**Load Balancing:** Use NGINX to load balance requests to multiple duplicates of RESTful API Servers. It is configured to distribute API requests across multiple API service instances.



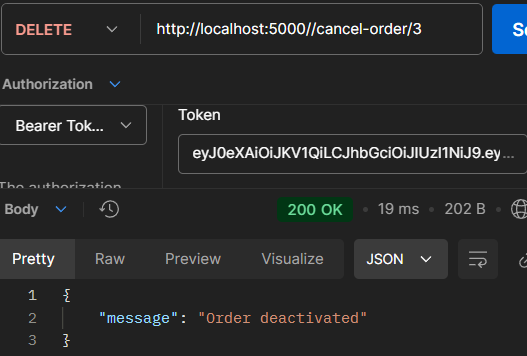
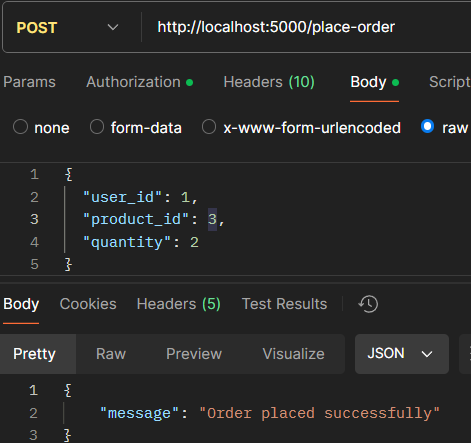
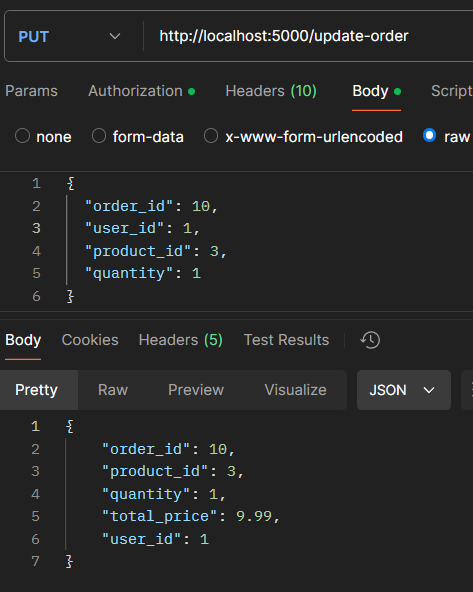
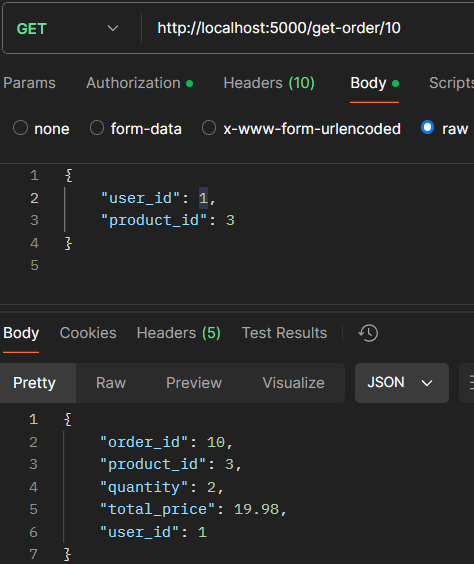
**IX. APPENDIX**

**USER API CALLS:** Details for each user-related API call are provided, including endpoints and expected behavior.

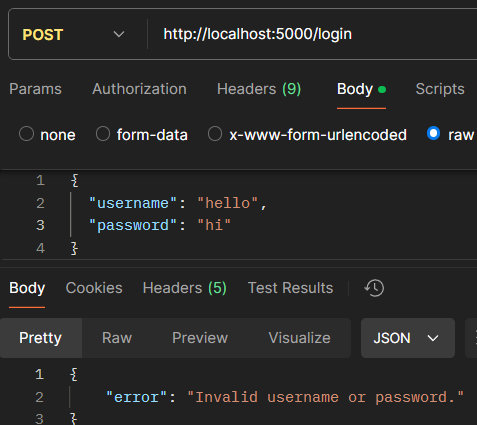




**ORDER API CALLS:** Details for order-related API calls, including tests for placing, updating, and canceling orders.

**API CALLS Validation Error Constraints:** Explanation of validation constraints and error handling when API inputs are incorrect, such as invalid passwords.

 \*provided that hello(username) password is not ‘hi’ but ‘hello’

