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.

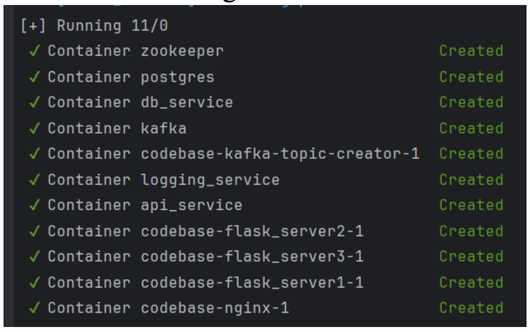
VI. Docker and Docker Compose Configuration

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.

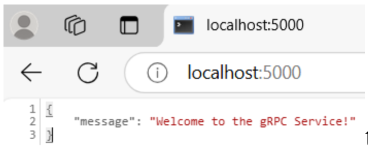


```
[+] Running 11/0
 ✓ Container zookeeper                          Created
 ✓ Container postgres                           Created
 ✓ Container db_service                         Created
 ✓ Container kafka                              Created
 ✓ Container codebase-kafka-topic-creator-1     Created
 ✓ Container logging_service                    Created
 ✓ Container api_service                        Created
 ✓ Container codebase-flask_server2-1           Created
 ✓ Container codebase-flask_server3-1           Created
 ✓ Container codebase-flask_server1-1           Created
 ✓ Container codebase-nginx-1                   Created
```

VII. Experiment and Testing

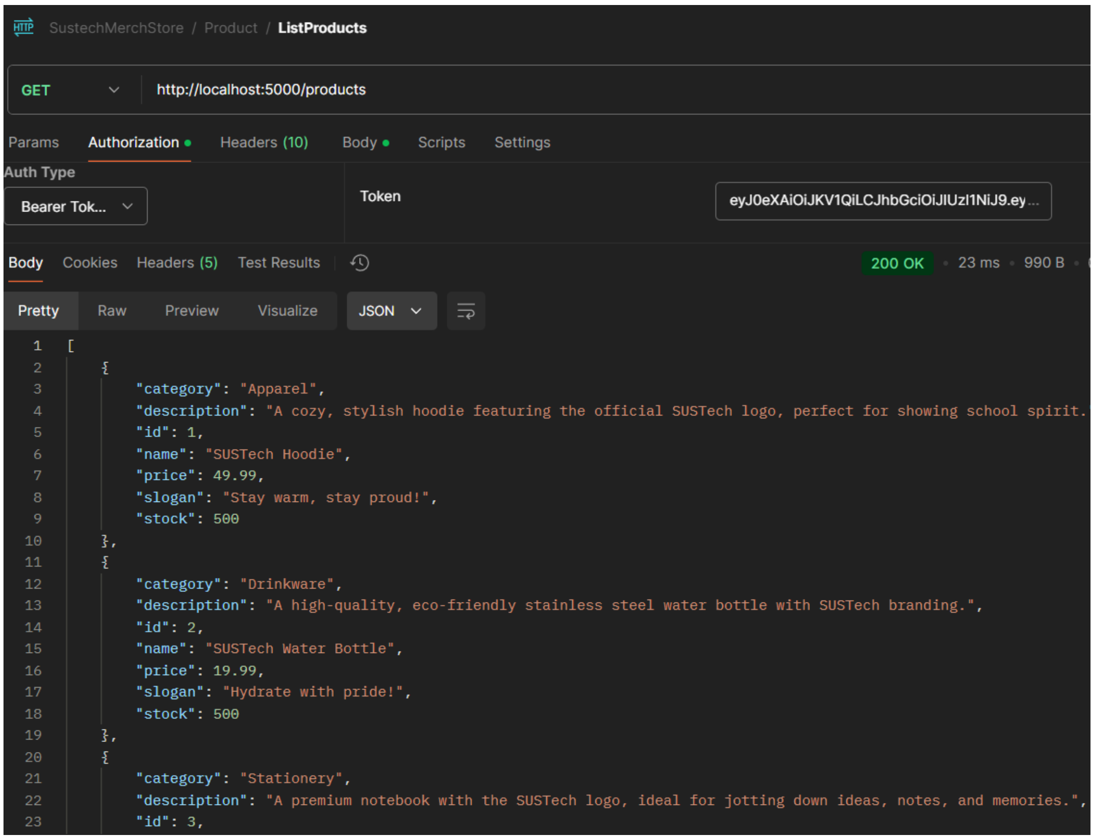
(How do you run the experiment? Which tool (i.e., cURL, Postman, Swagger UI) do you use to test your API Service? How do you monitor the log messages from the Kafka topic?)

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



```
{
  "message": "Welcome to the gRPC Service!"
}
```

the other API calls will need JWT Authorization Bearer Token



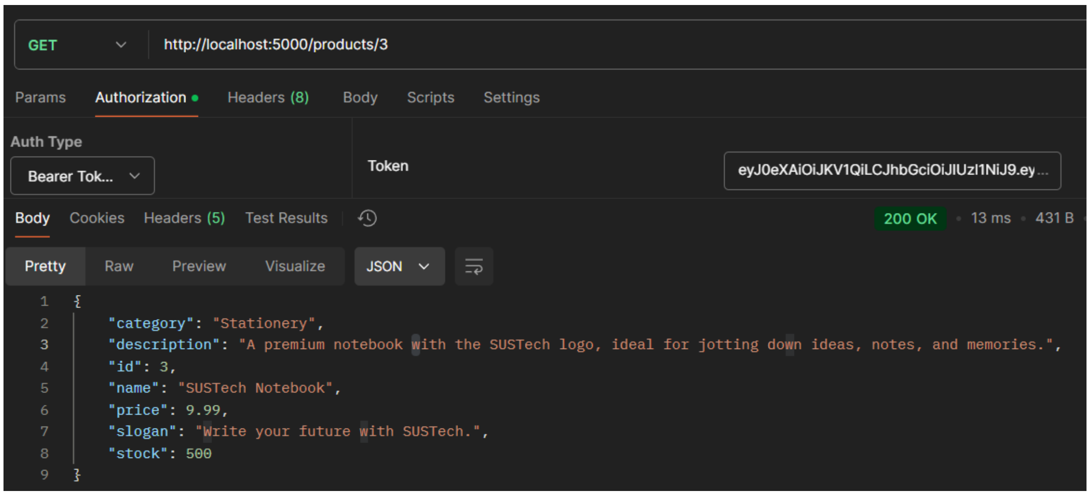
GET http://localhost:5000/products

Auth Type: Bearer Tok...

Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...

200 OK · 23 ms · 990 B

```
[
    {
        "category": "Apparel",
        "description": "A cozy, stylish hoodie featuring the official SUSTech logo, perfect for showing school spirit.
        "id": 1,
        "name": "SUSTech Hoodie",
        "price": 49.99,
        "slogan": "Stay warm, stay proud!",
        "stock": 500
    },
    {
        "category": "Drinkware",
        "description": "A high-quality, eco-friendly stainless steel water bottle with SUSTech branding.",
        "id": 2,
        "name": "SUSTech Water Bottle",
        "price": 19.99,
        "slogan": "Hydrate with pride!",
        "stock": 500
    },
    {
        "category": "Stationery",
        "description": "A premium notebook with the SUSTech logo, ideal for jotting down ideas, notes, and memories.",
        "id": 3,
```


GET http://localhost:5000/products/3

Params | Authorization ● | Headers (8) | Body | Scripts | Settings

Auth Type: Bearer Tok...   Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...

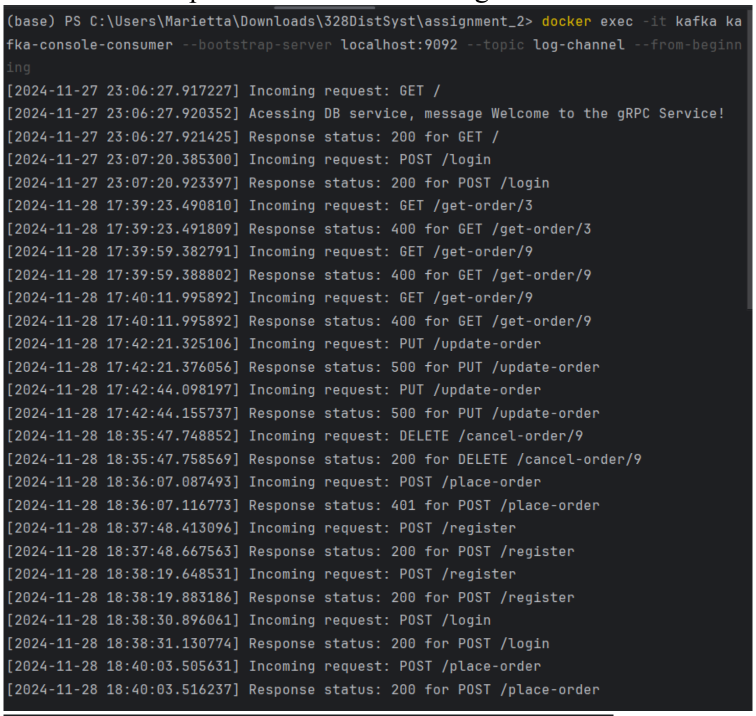
Body | Cookies | Headers (5) | Test Results   200 OK • 13 ms • 431 B

Pretty | Raw | Preview | Visualize | JSON

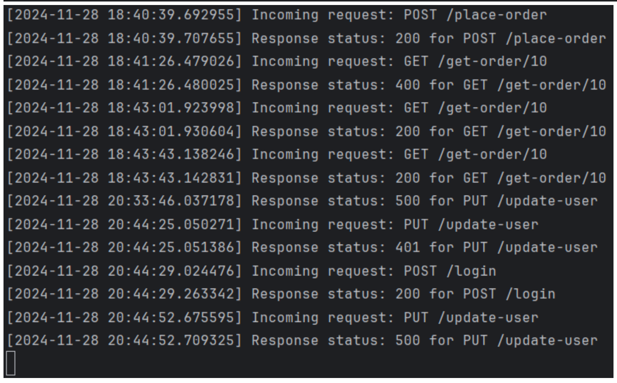
```
{
    "category": "Stationery",
    "description": "A premium notebook with the SUSTech logo, ideal for jotting down ideas, notes, and memories.",
    "id": 3,
    "name": "SUSTech Notebook",
    "price": 9.99,
    "slogan": "Write your future with SUSTech.",
    "stock": 500
}
```

*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.



```
(base) PS C:\Users\Marietta\Downloads\328DistSyst\assignment_2> docker exec -it kafka ka
fka-console-consumer --bootstrap-server localhost:9092 --topic log-channel --from-beginn
ing
[2024-11-27 23:06:27.917227] Incoming request: GET /
[2024-11-27 23:06:27.920352] Acessing DB service, message Welcome to the gRPC Service!
[2024-11-27 23:06:27.921425] Response status: 200 for GET /
[2024-11-27 23:07:20.385300] Incoming request: POST /login
[2024-11-27 23:07:20.923397] Response status: 200 for POST /login
[2024-11-28 17:39:23.490810] Incoming request: GET /get-order/3
[2024-11-28 17:39:23.491809] Response status: 400 for GET /get-order/3
[2024-11-28 17:39:59.382791] Incoming request: GET /get-order/9
[2024-11-28 17:39:59.388802] Response status: 400 for GET /get-order/9
[2024-11-28 17:40:11.995892] Incoming request: GET /get-order/9
[2024-11-28 17:40:11.995892] Response status: 400 for GET /get-order/9
[2024-11-28 17:42:21.325106] Incoming request: PUT /update-order
[2024-11-28 17:42:21.376056] Response status: 500 for PUT /update-order
[2024-11-28 17:42:44.098197] Incoming request: PUT /update-order
[2024-11-28 17:42:44.155737] Response status: 500 for PUT /update-order
[2024-11-28 18:35:47.748852] Incoming request: DELETE /cancel-order/9
[2024-11-28 18:35:47.758569] Response status: 200 for DELETE /cancel-order/9
[2024-11-28 18:36:07.087493] Incoming request: POST /place-order
[2024-11-28 18:36:07.116773] Response status: 401 for POST /place-order
[2024-11-28 18:37:48.413096] Incoming request: POST /register
[2024-11-28 18:37:48.667563] Response status: 200 for POST /register
[2024-11-28 18:38:19.648531] Incoming request: POST /register
[2024-11-28 18:38:19.883186] Response status: 200 for POST /register
[2024-11-28 18:38:30.896061] Incoming request: POST /login
[2024-11-28 18:38:31.130774] Response status: 200 for POST /login
[2024-11-28 18:40:03.505631] Incoming request: POST /place-order
[2024-11-28 18:40:03.516237] Response status: 200 for POST /place-order
```

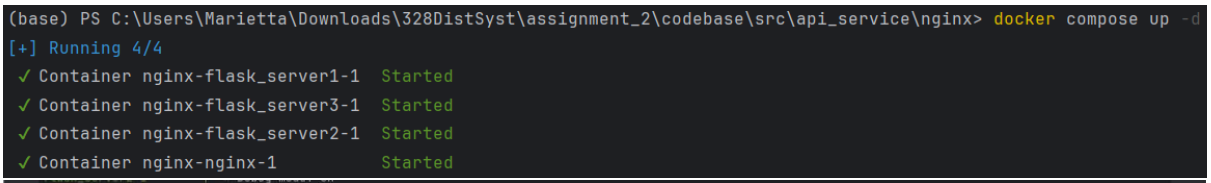


```
[2024-11-28 18:40:39.692955] Incoming request: POST /place-order
[2024-11-28 18:40:39.707655] Response status: 200 for POST /place-order
[2024-11-28 18:41:26.479026] Incoming request: GET /get-order/10
[2024-11-28 18:41:26.480025] Response status: 400 for GET /get-order/10
[2024-11-28 18:43:01.923998] Incoming request: GET /get-order/10
[2024-11-28 18:43:01.930604] Response status: 200 for GET /get-order/10
[2024-11-28 18:43:43.138246] Incoming request: GET /get-order/10
[2024-11-28 18:43:43.142831] Response status: 200 for GET /get-order/10
[2024-11-28 20:33:46.037178] Response status: 500 for PUT /update-user
[2024-11-28 20:44:25.050271] Incoming request: PUT /update-user
[2024-11-28 20:44:25.051386] Response status: 401 for PUT /update-user
[2024-11-28 20:44:29.024476] Incoming request: POST /login
[2024-11-28 20:44:29.263342] Response status: 200 for POST /login
[2024-11-28 20:44:52.675595] Incoming request: PUT /update-user
[2024-11-28 20:44:52.709325] Response status: 500 for PUT /update-user
```

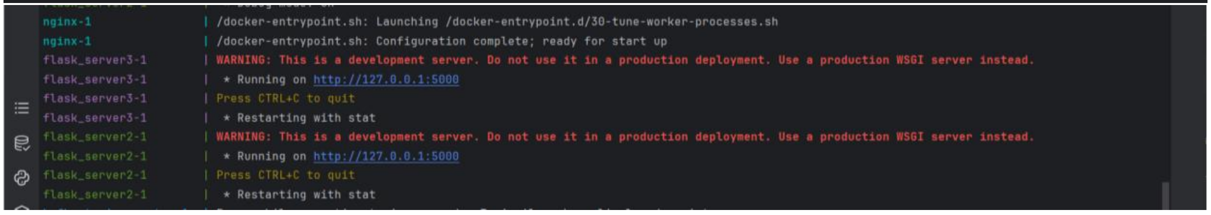
VIII. Bonus Implementation

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.



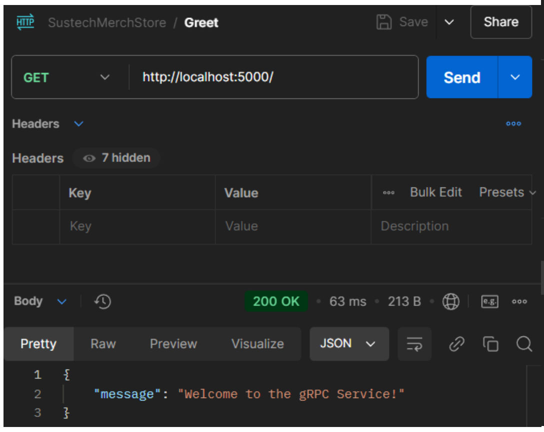
```
(base) PS C:\Users\Marietta\Downloads\328DistSyst\assignment_2\codebase\src\api_service\nginx> docker compose up -d
[+] Running 4/4
 ✓ Container nginx-flask_server1-1  Started
 ✓ Container nginx-flask_server3-1  Started
 ✓ Container nginx-flask_server2-1  Started
 ✓ Container nginx-nginx-1          Started
```



```
nginx-1              | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx-1              | /docker-entrypoint.sh: Configuration complete; ready for start up
flask_server3-1      | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
flask_server3-1      |  * Running on http://127.0.0.1:5000
flask_server3-1      | Press CTRL+C to quit
flask_server3-1      |  * Restarting with stat
flask_server2-1      | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
flask_server2-1      |  * Running on http://127.0.0.1:5000
flask_server2-1      | Press CTRL+C to quit
flask_server2-1      |  * Restarting with stat
```

IX. APPENDIX

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


SustechMerchStore / Greet

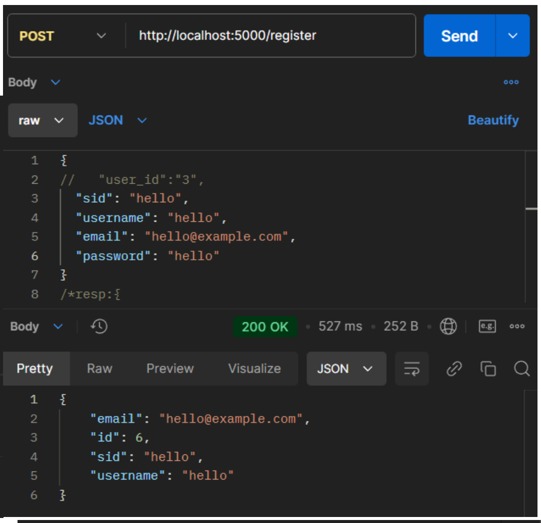
GET http://localhost:5000/

Headers: 7 hidden

Key	Value	Description
Key	Value	Description

Body — 200 OK · 63 ms · 213 B

```
{
  "message": "Welcome to the gRPC Service!"
}
```

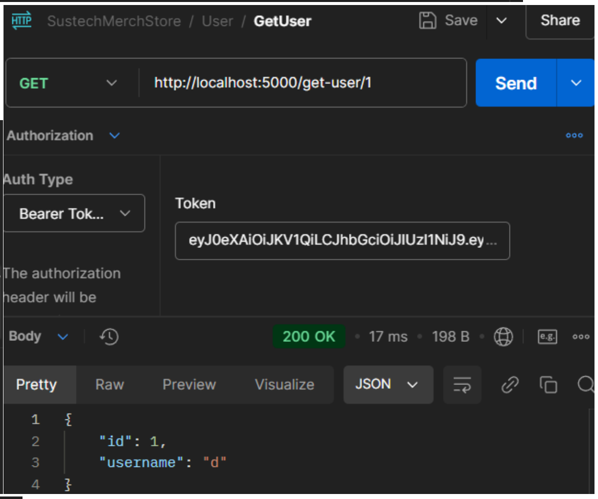

POST http://localhost:5000/register

Body — raw JSON

```
{
//   "user_id":"3",
  "sid": "hello",
  "username": "hello",
  "email": "hello@example.com",
  "password": "hello"
}
/*resp:{
```

Body — 200 OK · 527 ms · 252 B

```
{
    "email": "hello@example.com",
    "id": 6,
    "sid": "hello",
    "username": "hello"
}
```


SustechMerchStore / User / GetUser

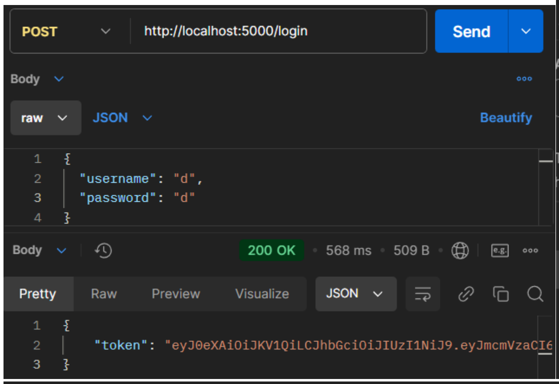
GET http://localhost:5000/get-user/1

Authorization — Auth Type: Bearer Tok... — Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...

The authorization header will be

Body — 200 OK · 17 ms · 198 B

```
{
    "id": 1,
    "username": "d"
}
```

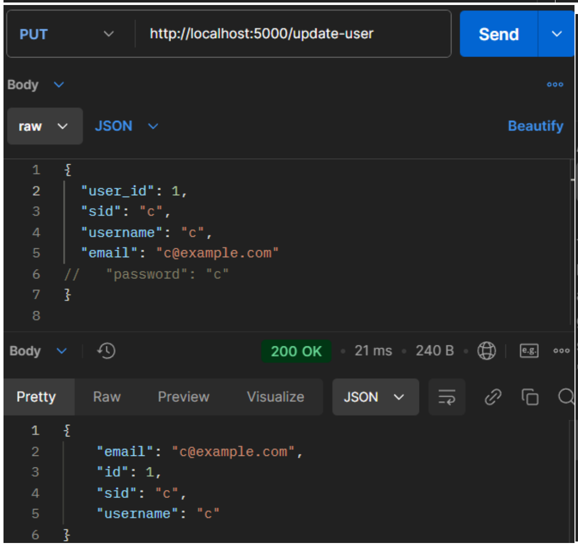

POST http://localhost:5000/login

Body — raw JSON

```
{
  "username": "d",
  "password": "d"
}
```

Body — 200 OK · 568 ms · 509 B

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6
}
```

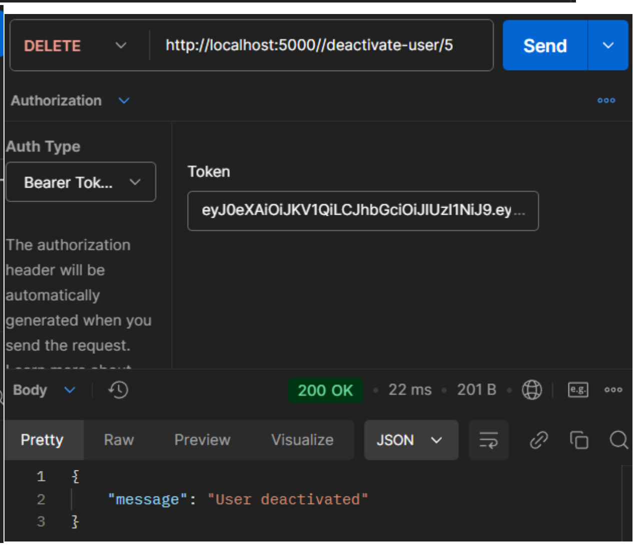

PUT http://localhost:5000/update-user

Body — raw JSON

```
{
  "user_id": 1,
  "sid": "c",
  "username": "c",
  "email": "c@example.com"
//   "password": "c"
}
```

Body — 200 OK · 21 ms · 240 B

```
{
    "email": "c@example.com",
    "id": 1,
    "sid": "c",
    "username": "c"
}
```


DELETE http://localhost:5000//deactivate-user/5

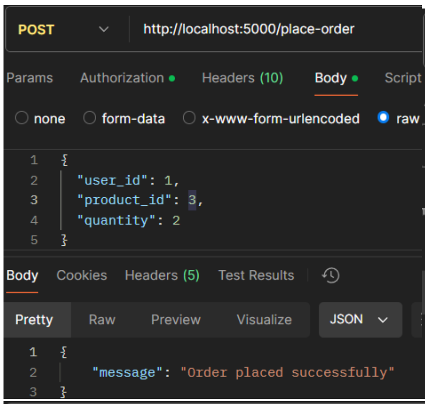
Authorization — Auth Type: Bearer Tok... — Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...

The authorization header will be automatically generated when you send the request.

Body — 200 OK · 22 ms · 201 B

```
{
    "message": "User deactivated"
}
```

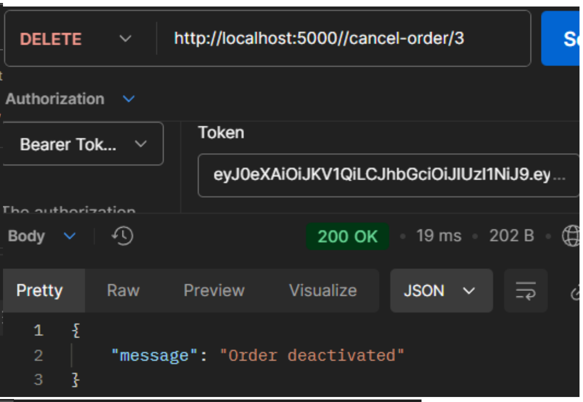
ORDER API CALLS: Details for order-related API calls, including tests for placing, updating, and canceling orders.


POST http://localhost:5000/place-order

Body: raw

```
{
  "user_id": 1,
  "product_id": 3,
  "quantity": 2
}
```

```
{
    "message": "Order placed successfully"
}
```

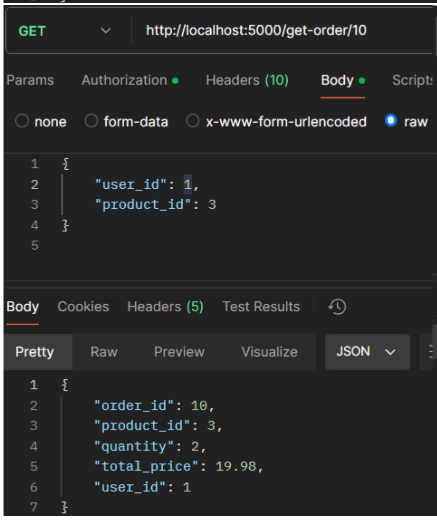

DELETE http://localhost:5000//cancel-order/3

Authorization: Bearer Tok...

Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...

200 OK • 19 ms • 202 B

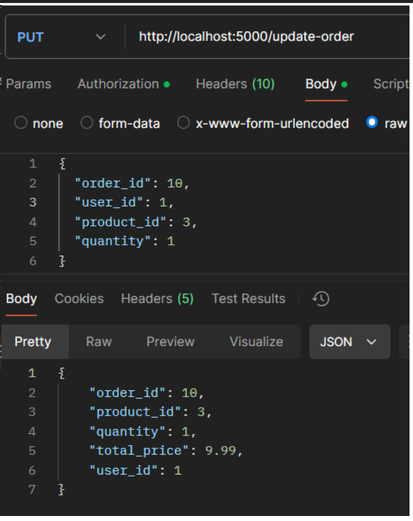
```
{
    "message": "Order deactivated"
}
```


GET http://localhost:5000/get-order/10

Body: raw

```
{
    "user_id": 1,
    "product_id": 3
}
```

```
{
    "order_id": 10,
    "product_id": 3,
    "quantity": 2,
    "total_price": 19.98,
    "user_id": 1
}
```

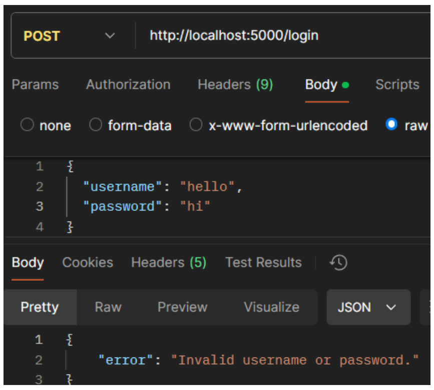

PUT http://localhost:5000/update-order

Body: raw

```
{
  "order_id": 10,
  "user_id": 1,
  "product_id": 3,
  "quantity": 1
}
```

```
{
    "order_id": 10,
    "product_id": 3,
    "quantity": 1,
    "total_price": 9.99,
    "user_id": 1
}
```

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

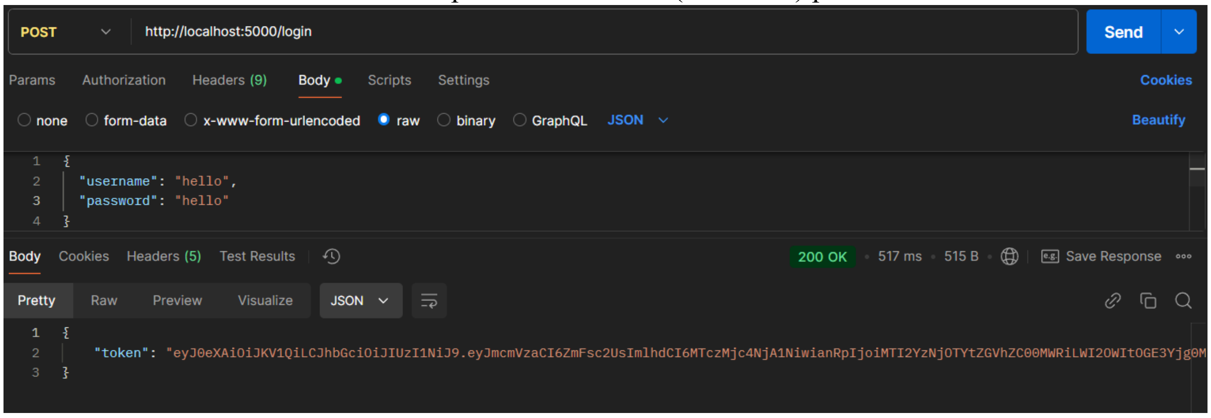

POST http://localhost:5000/login

Body: raw

```
{
  "username": "hello",
  "password": "hi"
}
```

```
{
    "error": "Invalid username or password."
}
```

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

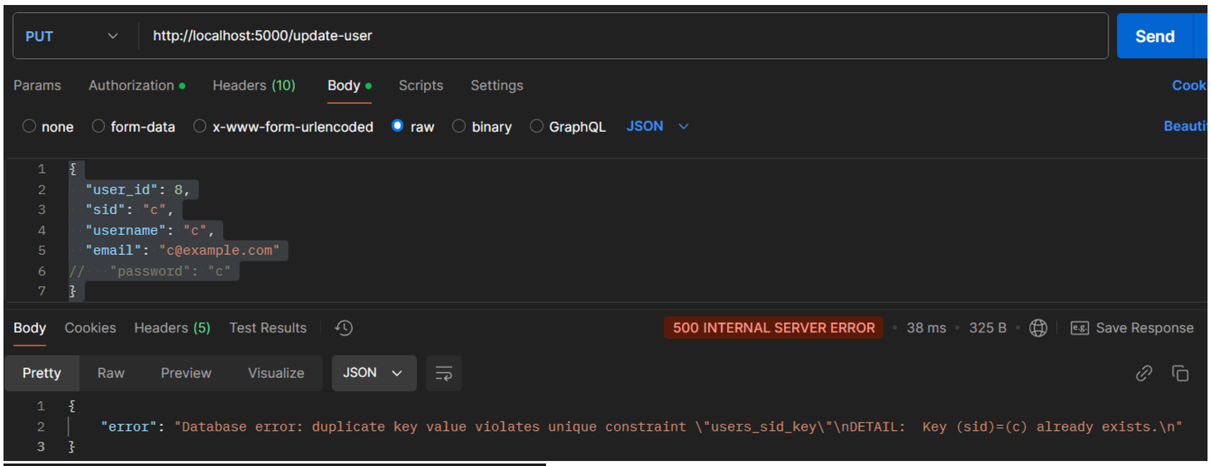

POST http://localhost:5000/login

Body: raw JSON

```
{
  "username": "hello",
  "password": "hello"
}
```

200 OK • 517 ms • 515 B

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczMjc4NjA1NiwianRpIjoiMTI2YzNjOTYtZGVhZC00MWRiLWI2OWItOGE3Yjg0M
}
```

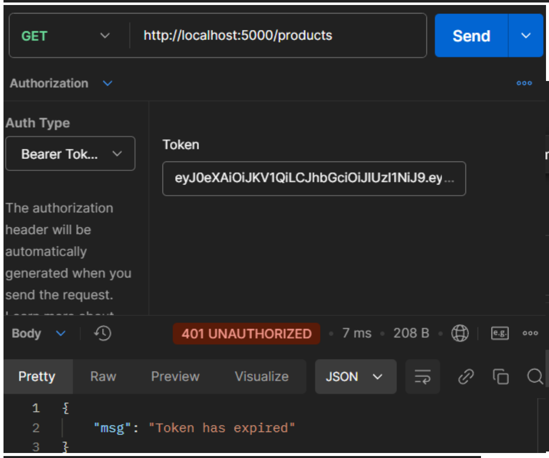


PUT http://localhost:5000/update-user

```
{
  "user_id": 8,
  "sid": "c",
  "username": "c",
  "email": "c@example.com"
//   "password": "c"
}
```

500 INTERNAL SERVER ERROR · 38 ms · 325 B

```
{
    "error": "Database error: duplicate key value violates unique constraint \"users_sid_key\"\nDETAIL:  Key (sid)=(c) already exists.\n"
}
```



GET http://localhost:5000/products

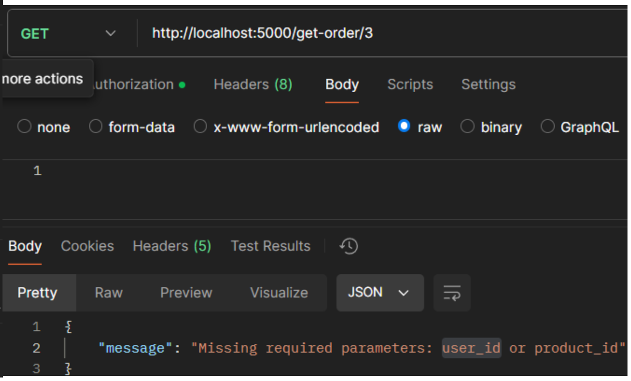
Auth Type: Bearer Tok...

Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...

The authorization header will be automatically generated when you send the request.

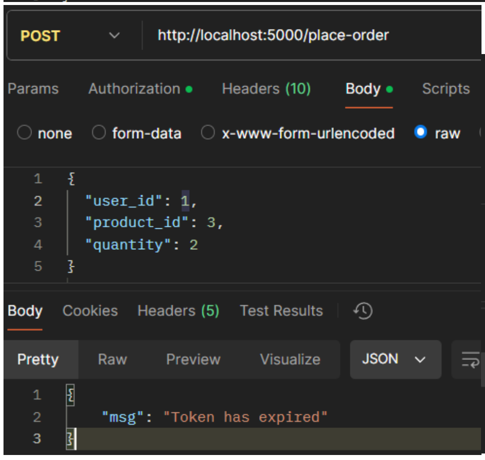
401 UNAUTHORIZED · 7 ms · 208 B

```
{
    "msg": "Token has expired"
}
```



GET http://localhost:5000/get-order/3

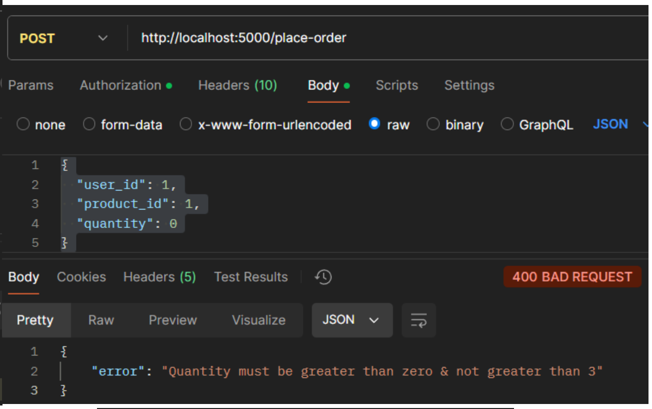
```
{
    "message": "Missing required parameters: user_id or product_id"
}
```



POST http://localhost:5000/place-order

```
{
  "user_id": 1,
  "product_id": 3,
  "quantity": 2
}
```

```
{
    "msg": "Token has expired"
}
```

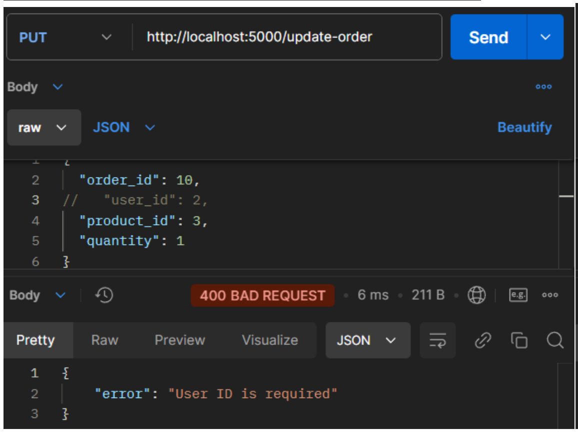


POST http://localhost:5000/place-order

```
{
  "user_id": 1,
  "product_id": 1,
  "quantity": 0
}
```

400 BAD REQUEST

```
{
    "error": "Quantity must be greater than zero & not greater than 3"
}
```

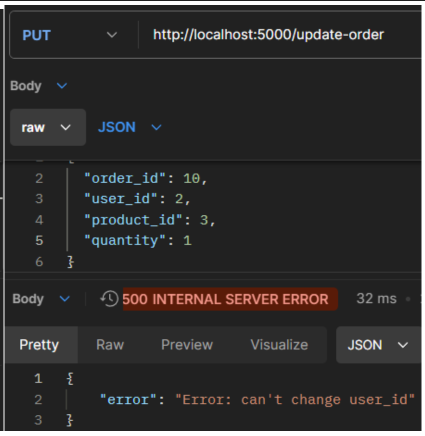


PUT http://localhost:5000/update-order

```
  "order_id": 10,
//   "user_id": 2,
  "product_id": 3,
  "quantity": 1
}
```

400 BAD REQUEST · 6 ms · 211 B

```
{
    "error": "User ID is required"
}
```

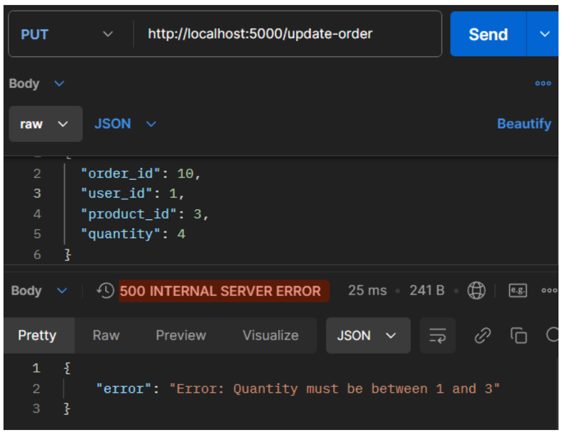


PUT http://localhost:5000/update-order

```
  "order_id": 10,
  "user_id": 2,
  "product_id": 3,
  "quantity": 1
}
```

500 INTERNAL SERVER ERROR · 32 ms

```
{
    "error": "Error: can't change user_id"
}
```



```
http://localhost:5000/update-order
```

```
  "order_id": 10,
  "user_id": 1,
  "product_id": 3,
  "quantity": 4
}
```

```
500 INTERNAL SERVER ERROR   25 ms • 241 B
```

```
{
    "error": "Error: Quantity must be between 1 and 3"
}
```