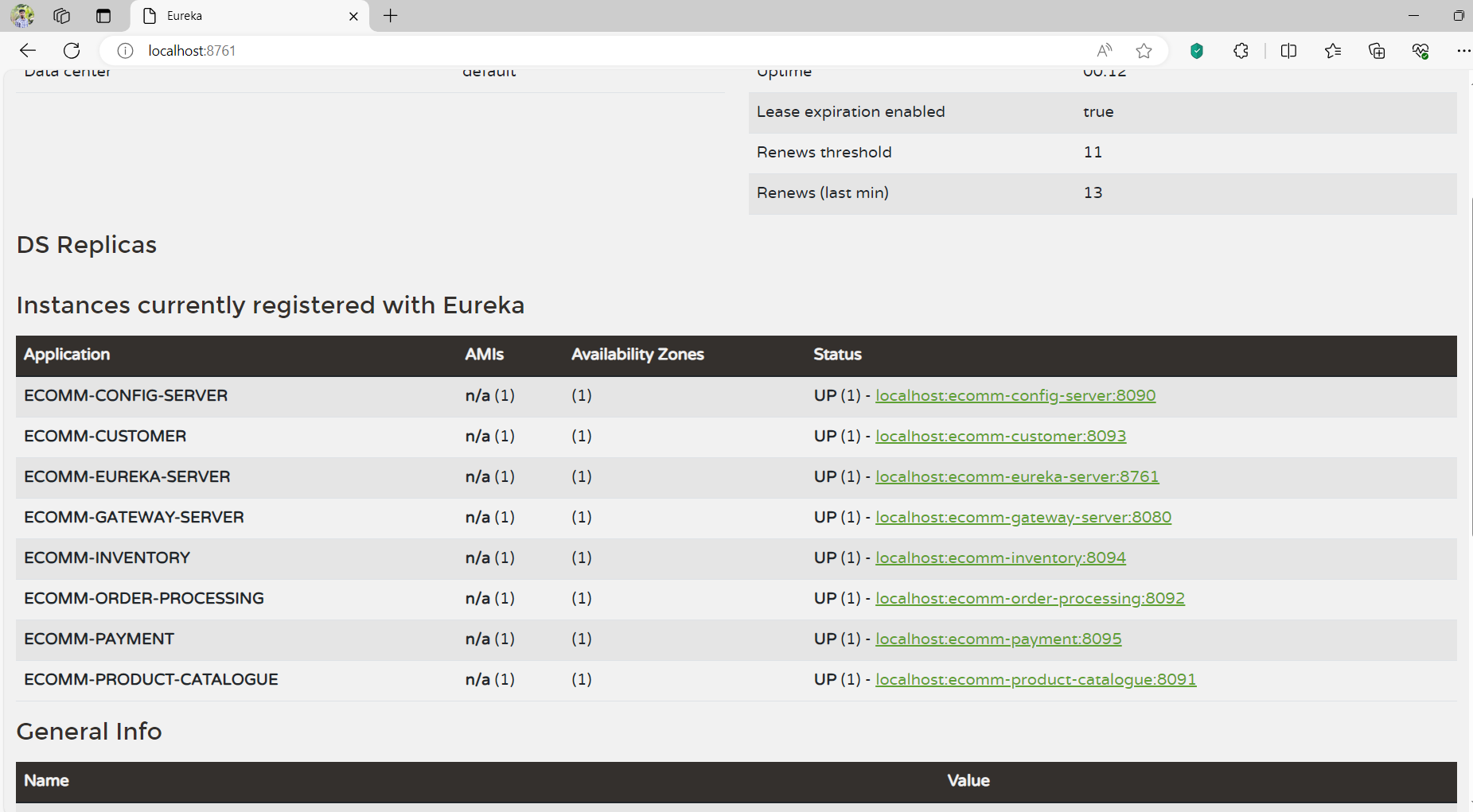
# Ecommerce Application Repository

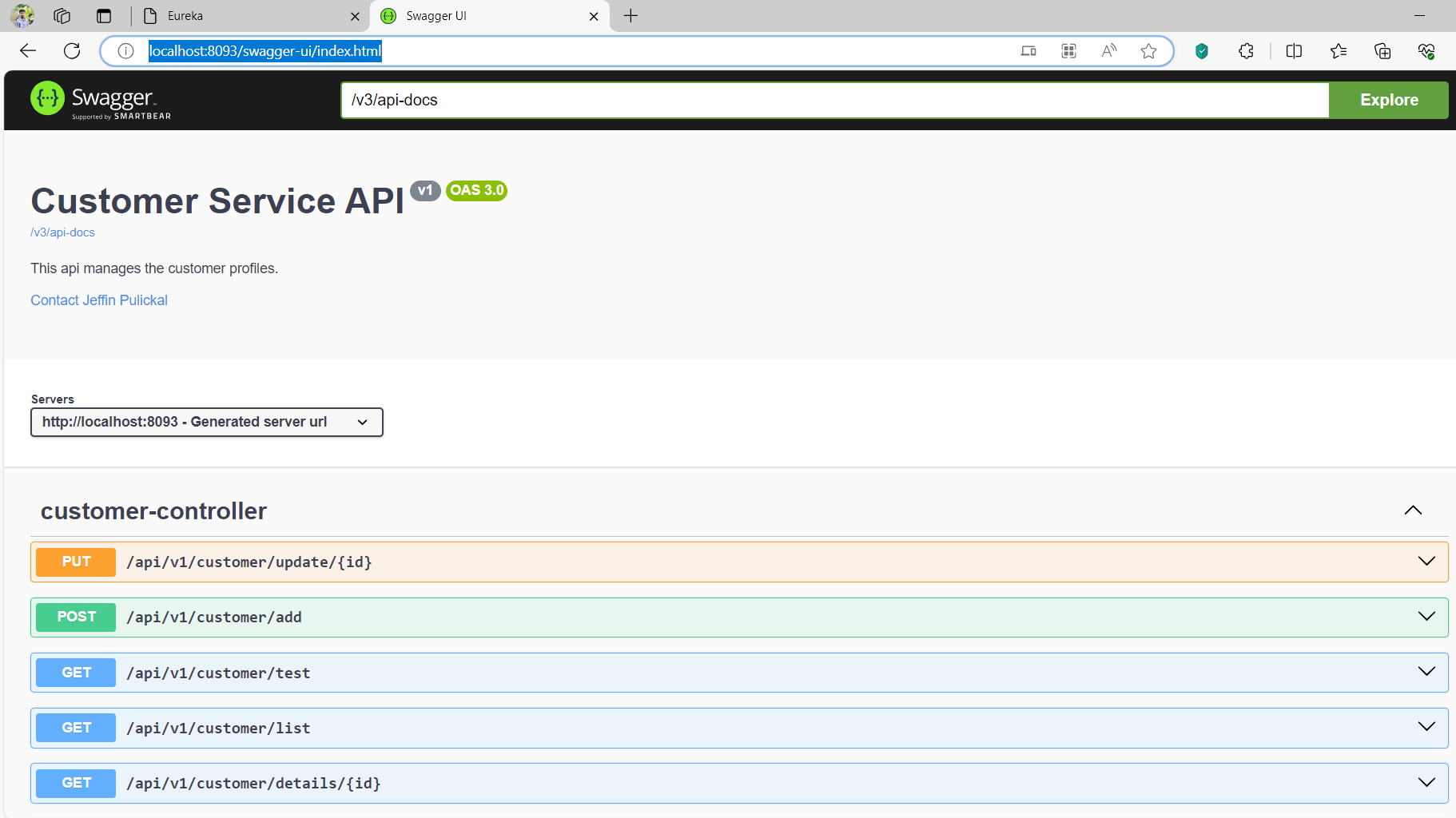
1. Config server - <https://github.com/jeffinjude/ecomm-config-server>
2. Config server props - <https://github.com/jeffinjude/ecomm-config-server-props>
3. Eureka server - <https://github.com/jeffinjude/ecomm-eureka-server>
4. Gateway server - <https://github.com/jeffinjude/ecomm-gateway-server>
5. Customer service - <https://github.com/jeffinjude/ecomm-customer>
6. Product catalogue service - <https://github.com/jeffinjude/ecomm-product-catalogue>
7. Inventory service - <https://github.com/jeffinjude/ecomm-inventory>
8. Order processing service - <https://github.com/jeffinjude/ecomm-order-processing>
9. Payment service - <https://github.com/jeffinjude/ecomm-payment>
10. Kubernetes config files - <https://github.com/jeffinjude/ecomm-k8s-deployment>
11. Ecomm resources - <https://github.com/jeffinjude/ecomm-resources>

# Ecommerce Application Setup

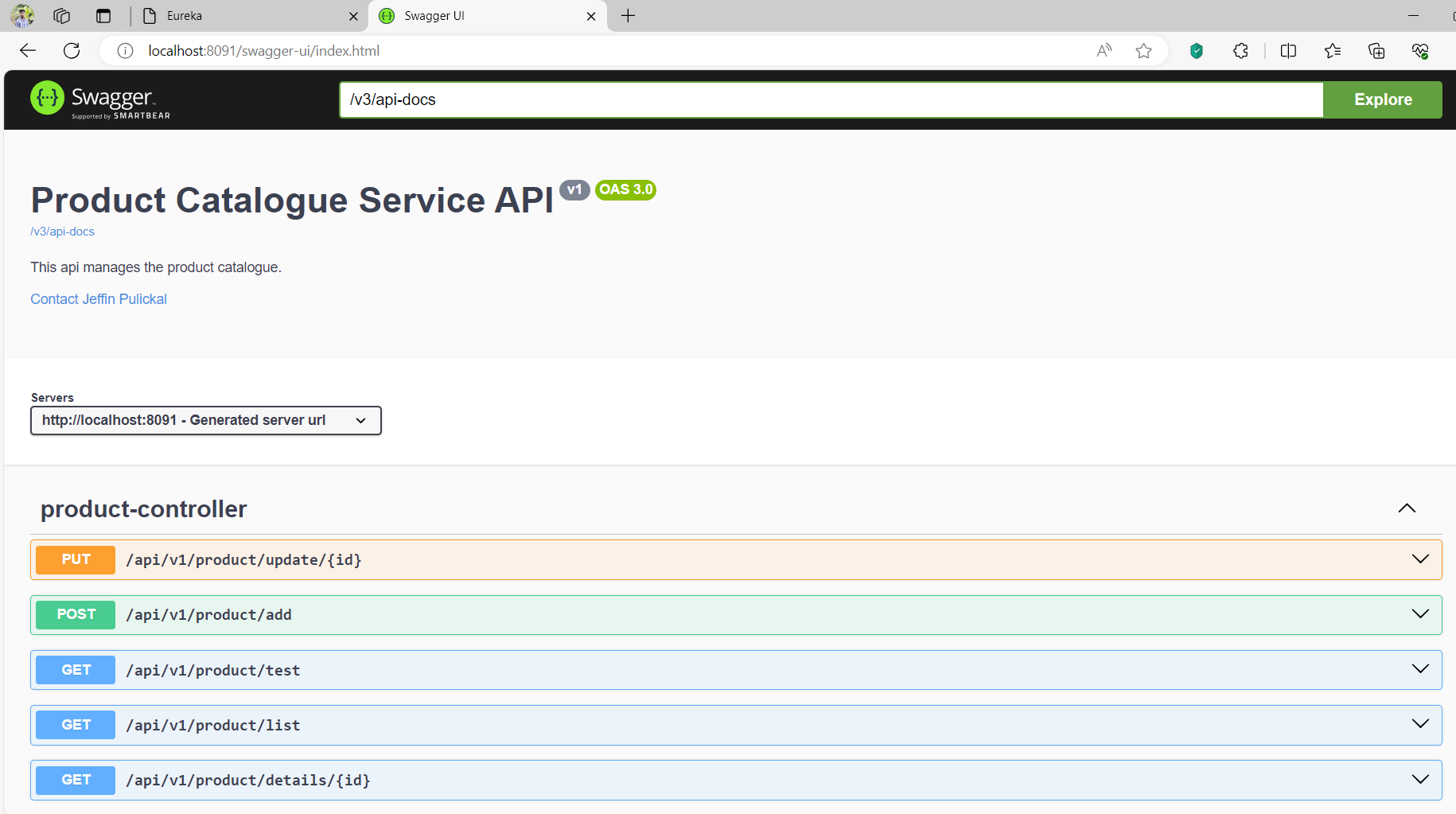
1. Start the apps in the following order. Make sure to ensure config, eureka and gateway server are completely started to prevent errors in the business microservices.
   * ecomm-config-server
   * ecomm-eureka-server
   * ecomm-gateway-server
   * ecomm-customer
   * ecomm-product-catalogue
   * ecomm-inventory
   * ecomm-order-processing
   * ecomm-payment
2. Access eureka server on localhost:8761 to list out all the registered services.



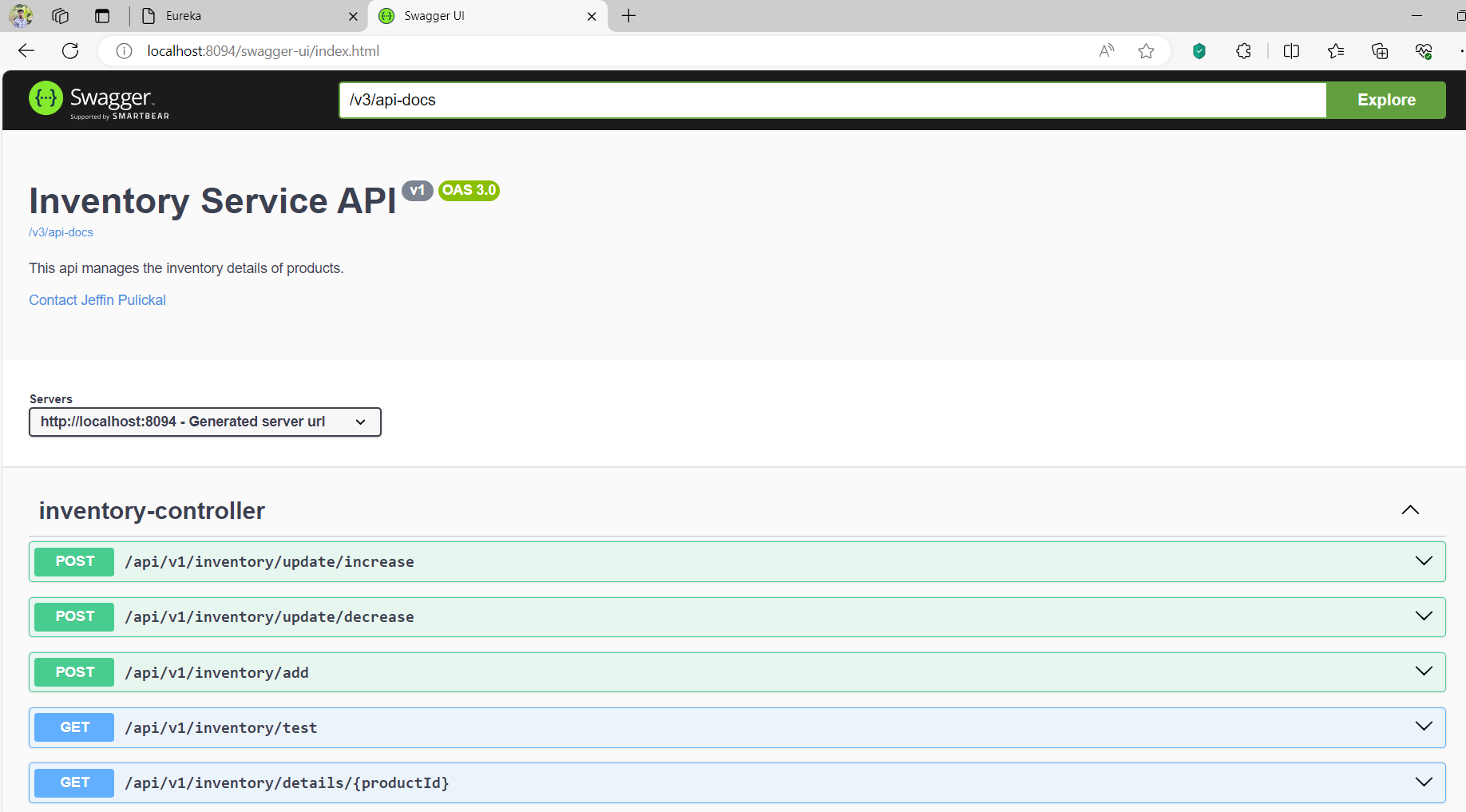
1. Access swagger endpoints of the services.
   * Customer Service - <http://localhost:8093/swagger-ui/index.html>



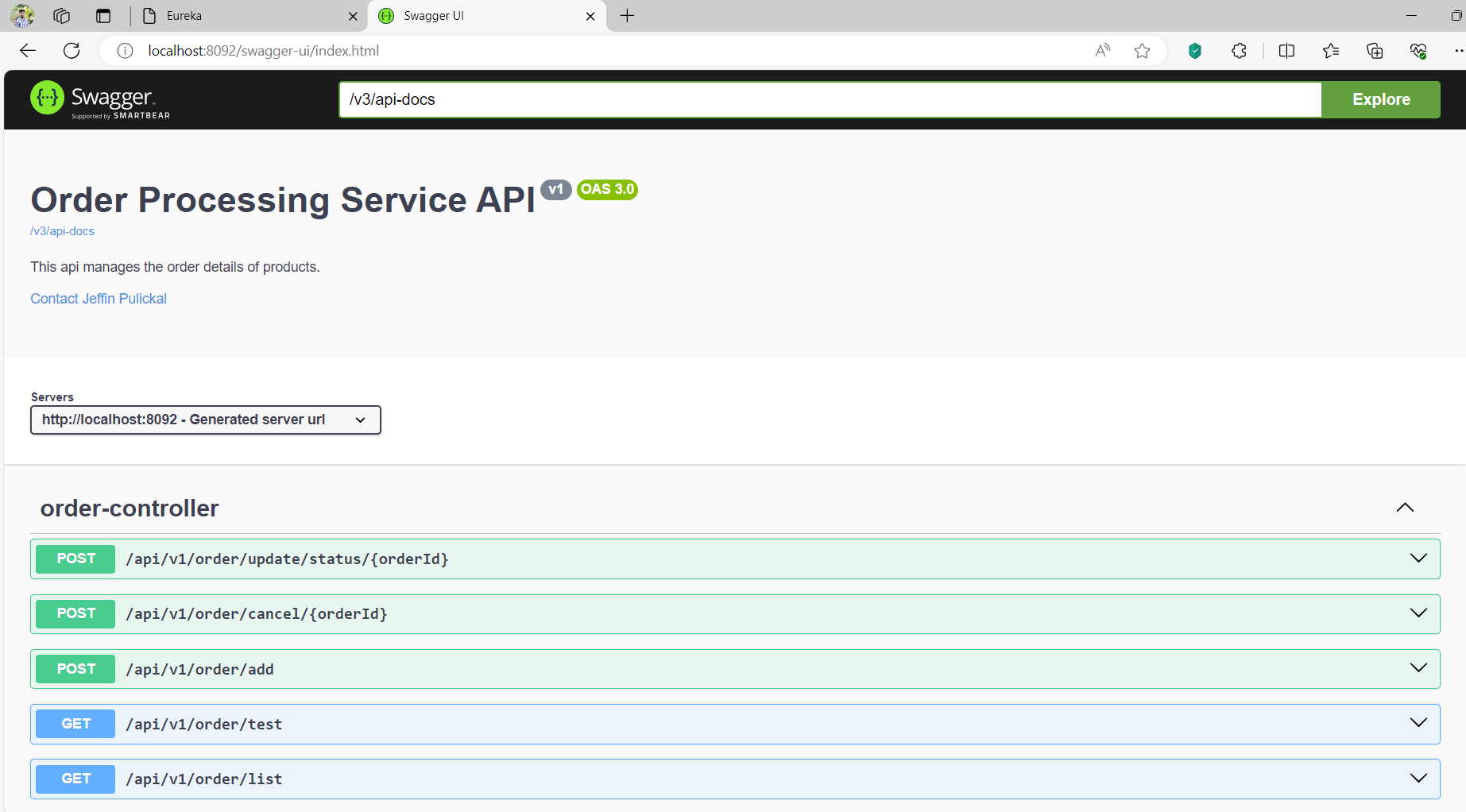
* Product catalogue service - <http://localhost:8091/swagger-ui/index.html>



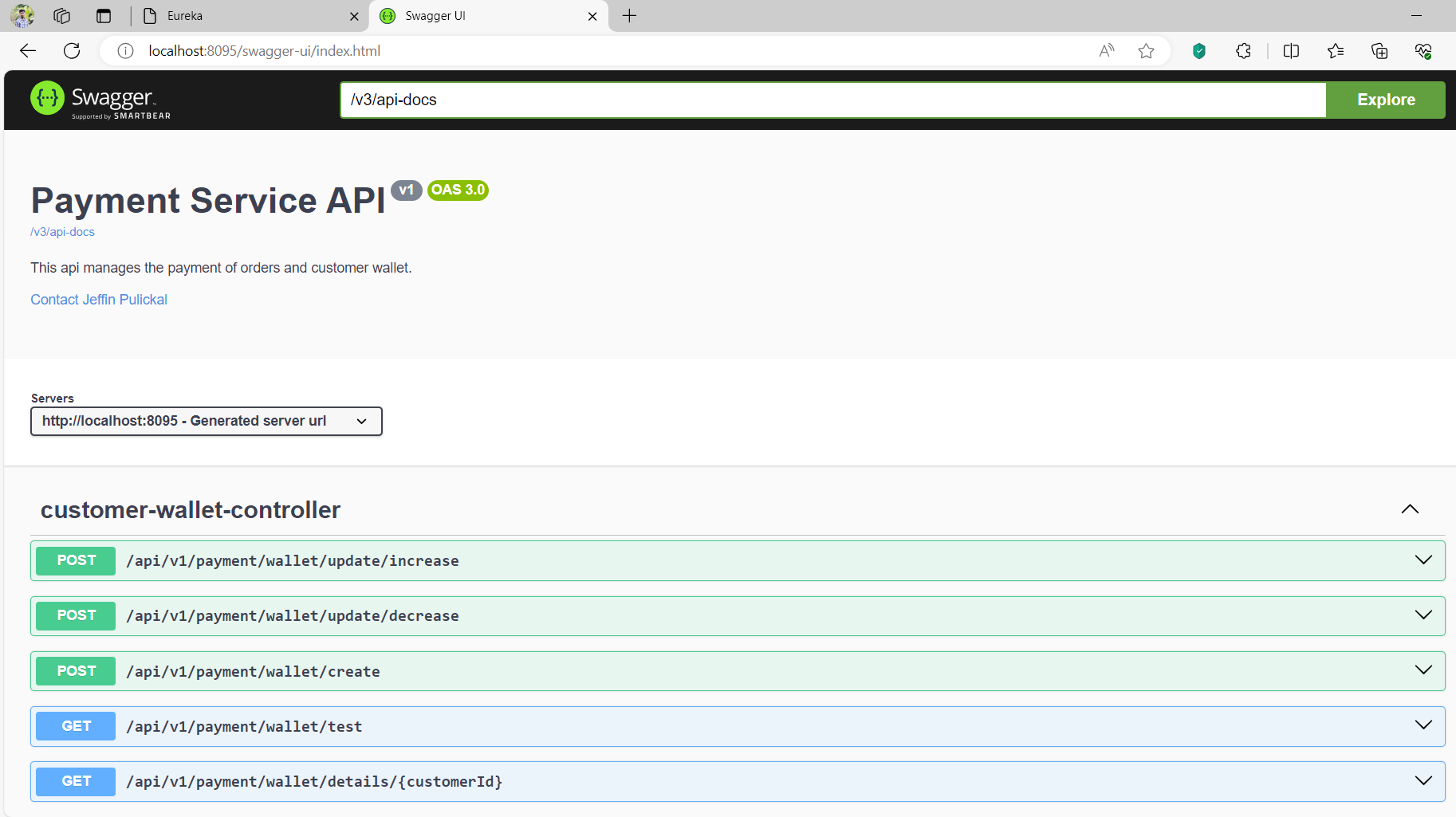
* Inventory service - <http://localhost:8094/swagger-ui/index.html>



* Order processing service - http://localhost:8092/swagger-ui/index.html



* Payment service - <http://localhost:8095/swagger-ui/index.html>



1. Run the zipkin server. Give command ‘java -jar <zipkinjar>’. The zipkin server will be accessible at <http://localhost:9411/zipkin/>.
2. Run kafka.
   * Run zookeeper

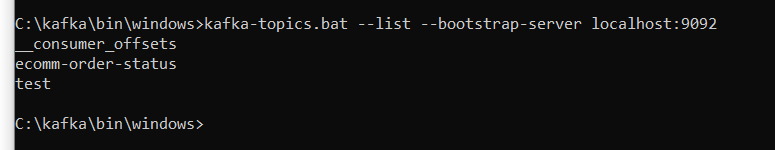
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

* + Start kafka server

.\bin\windows\kafka-server-start.bat .\config\server.properties

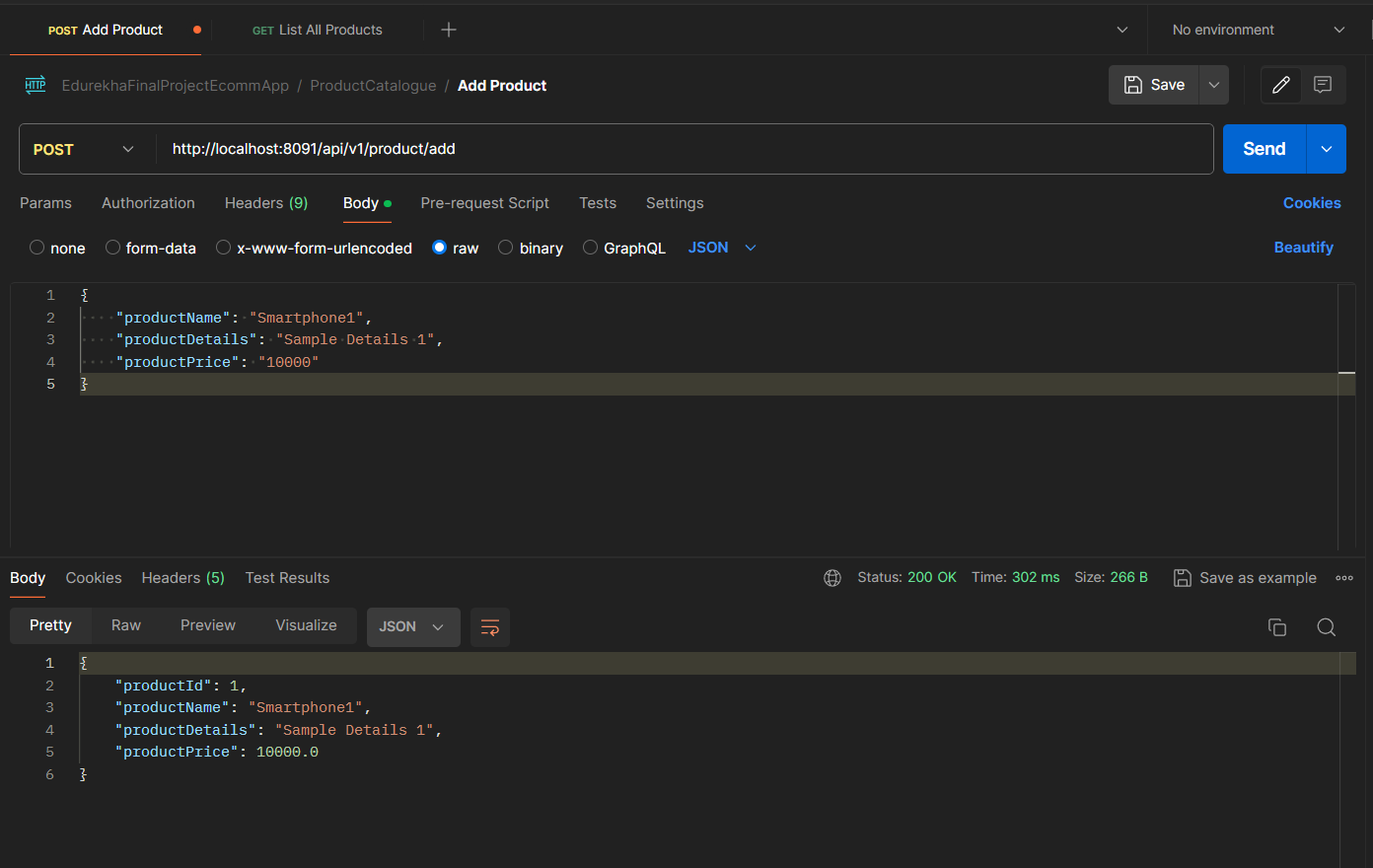
* + Creata a kafka topic named ‘ecomm-order-status’.

kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 1 --partition 1 --topic ecomm-order-status

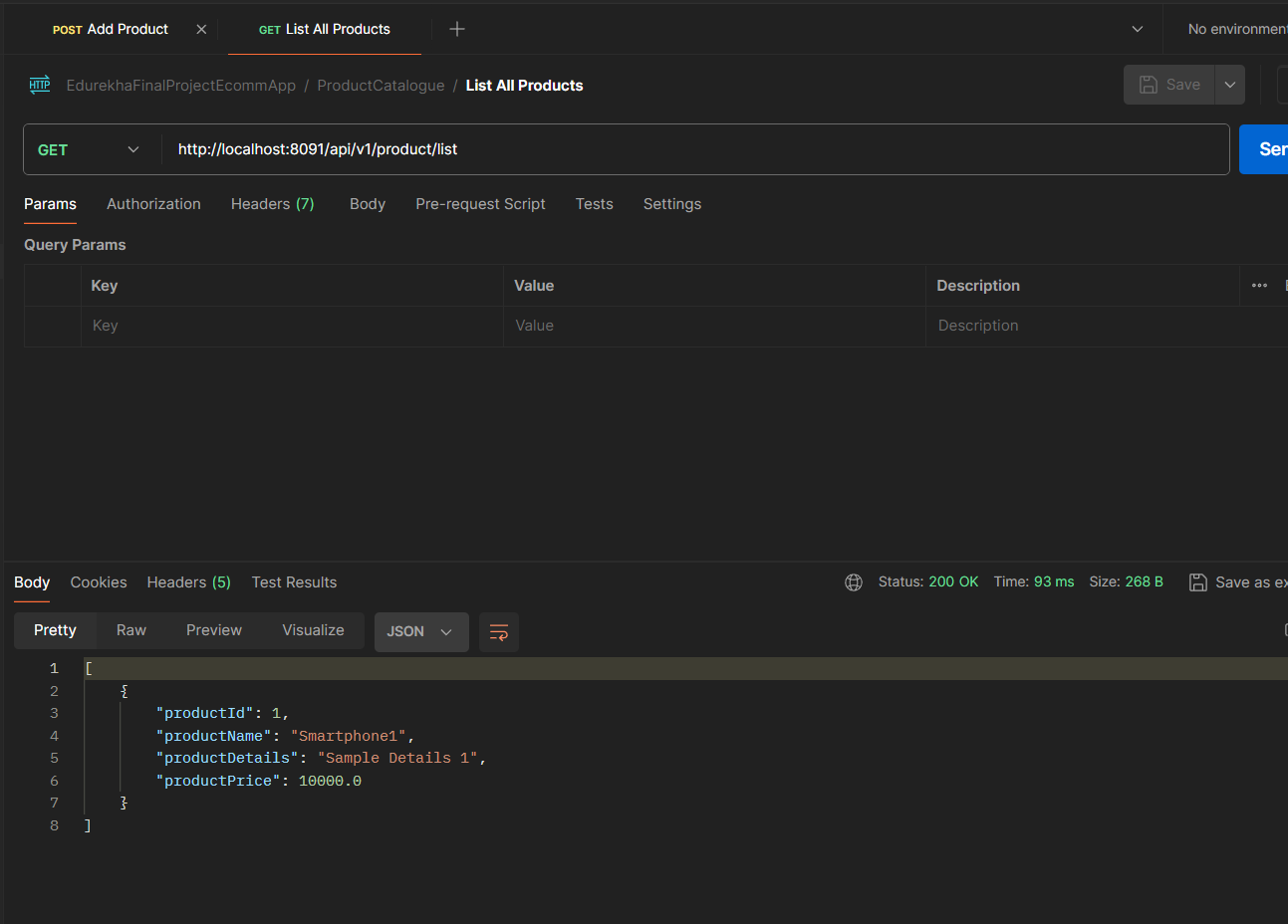
* + Verify kafka topic 

# Sample Business Flow

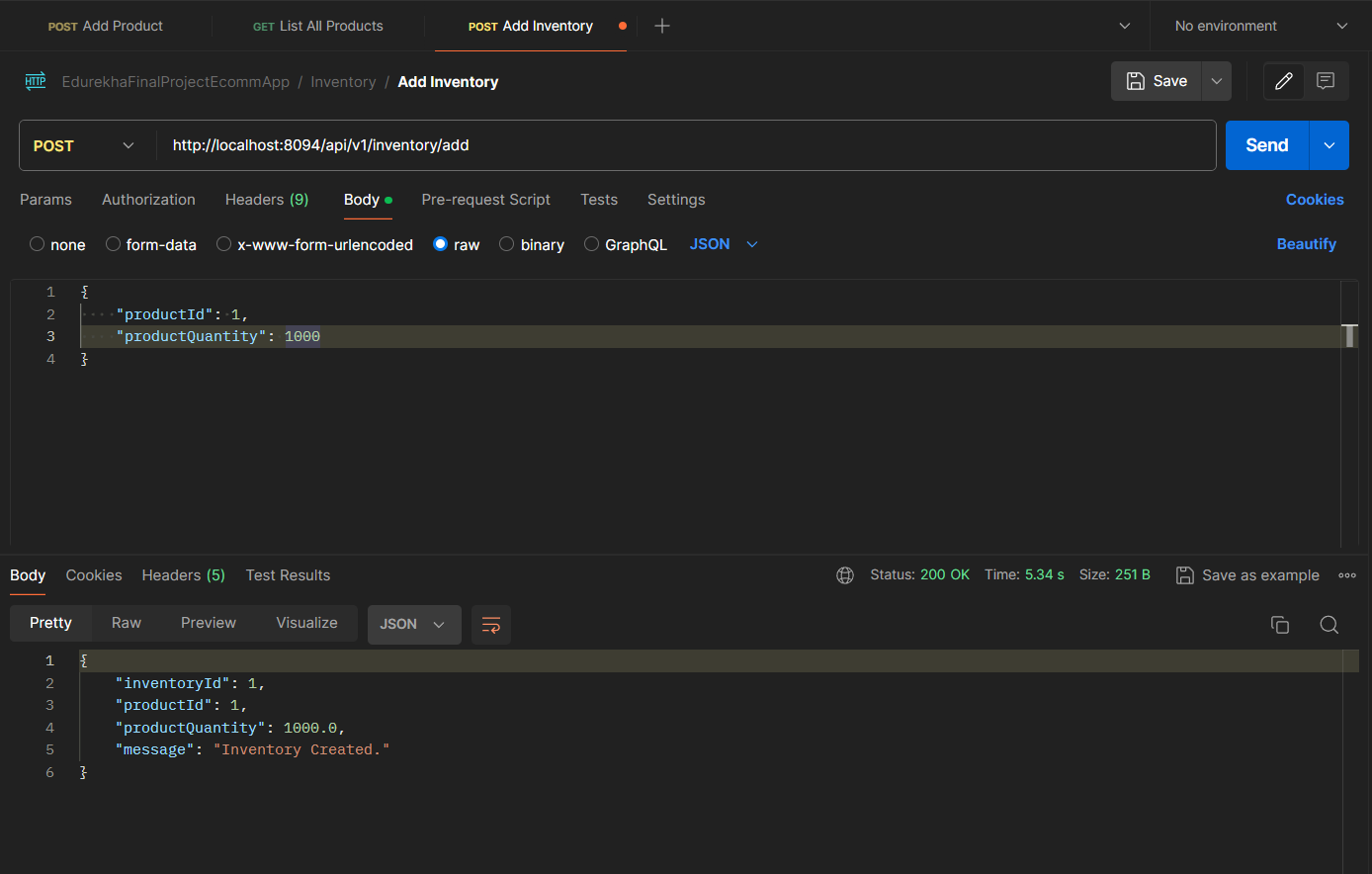
1. Create a product.



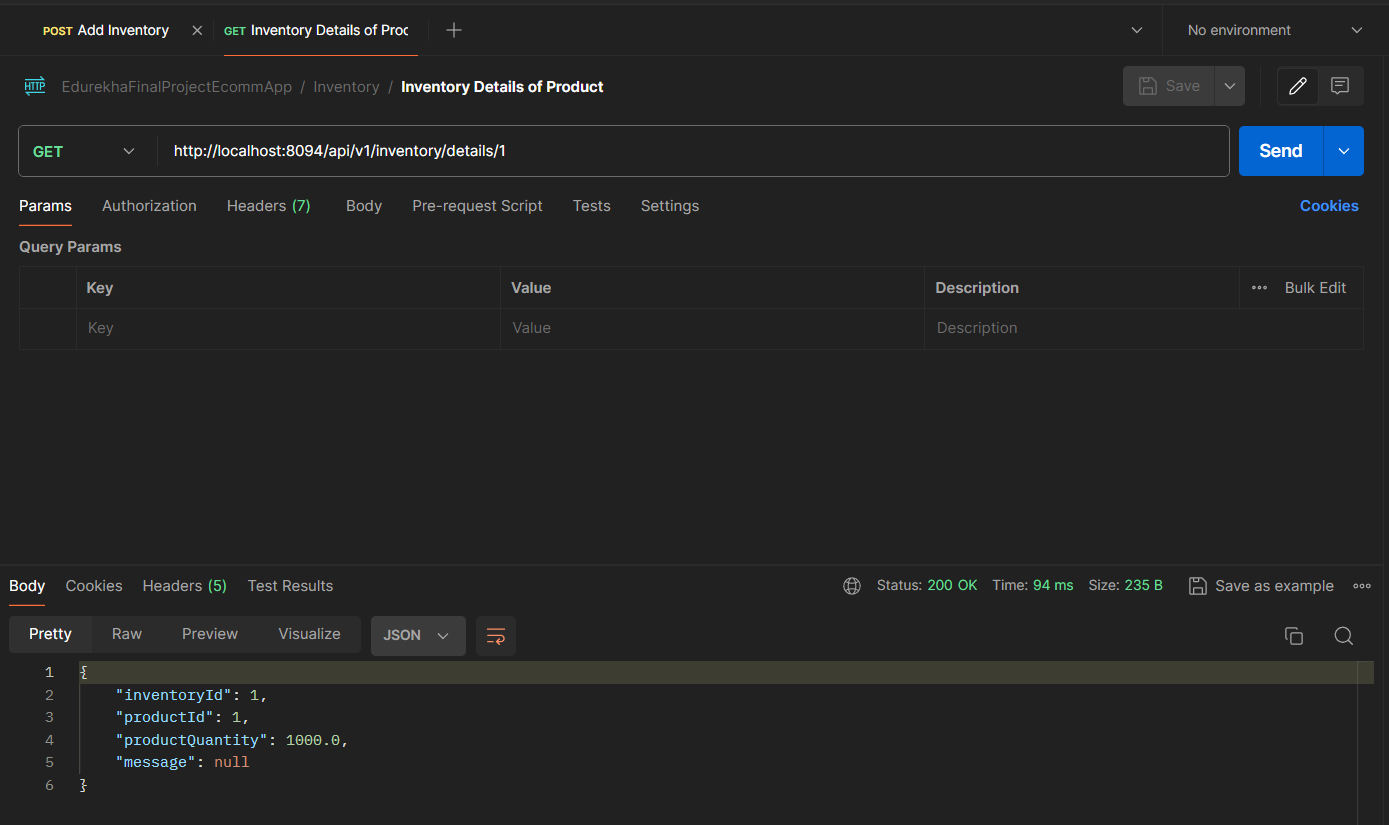
1. Verify the product is created by listing out the products in the DB.



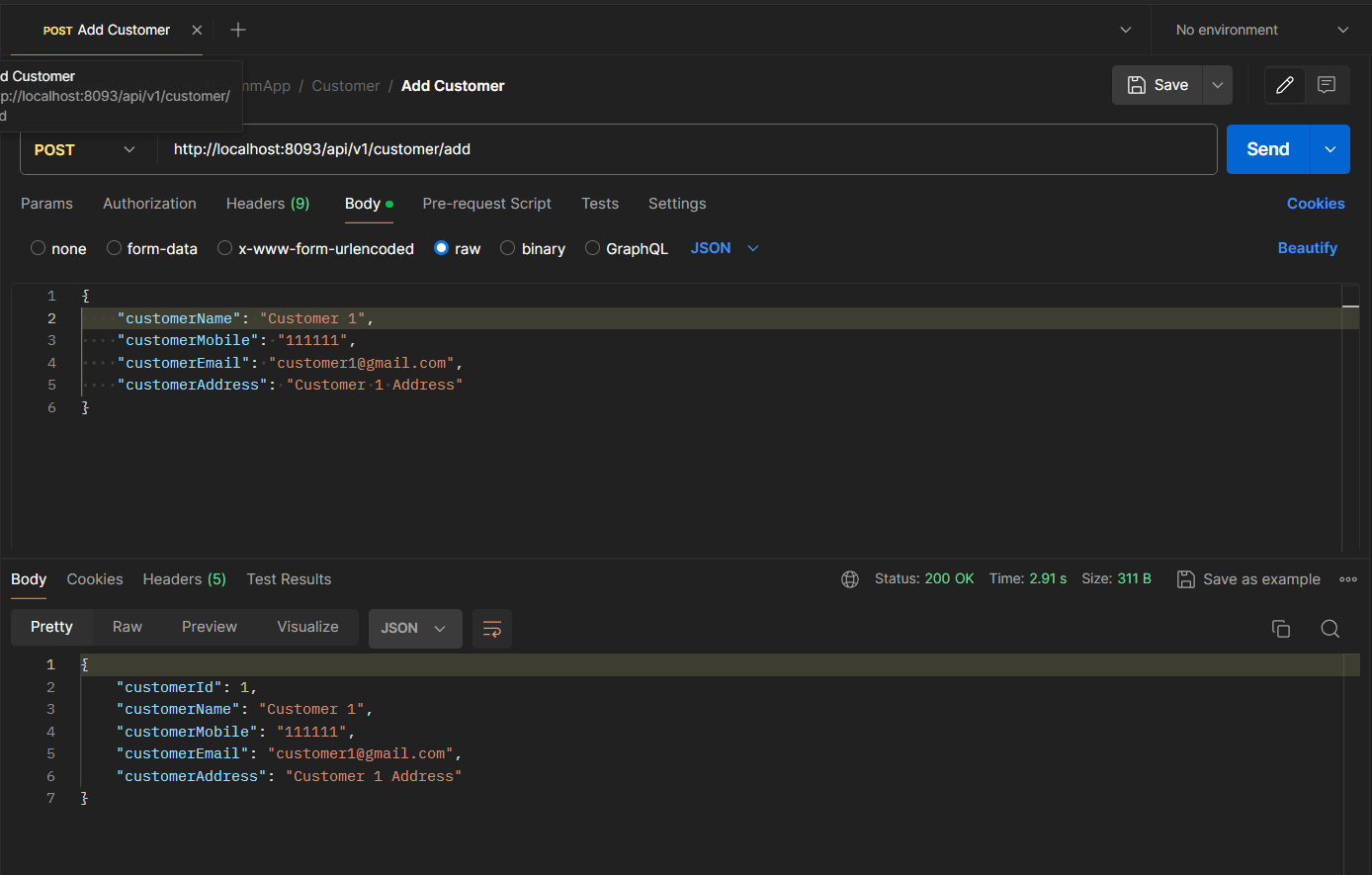
1. Create an inventory for the product.



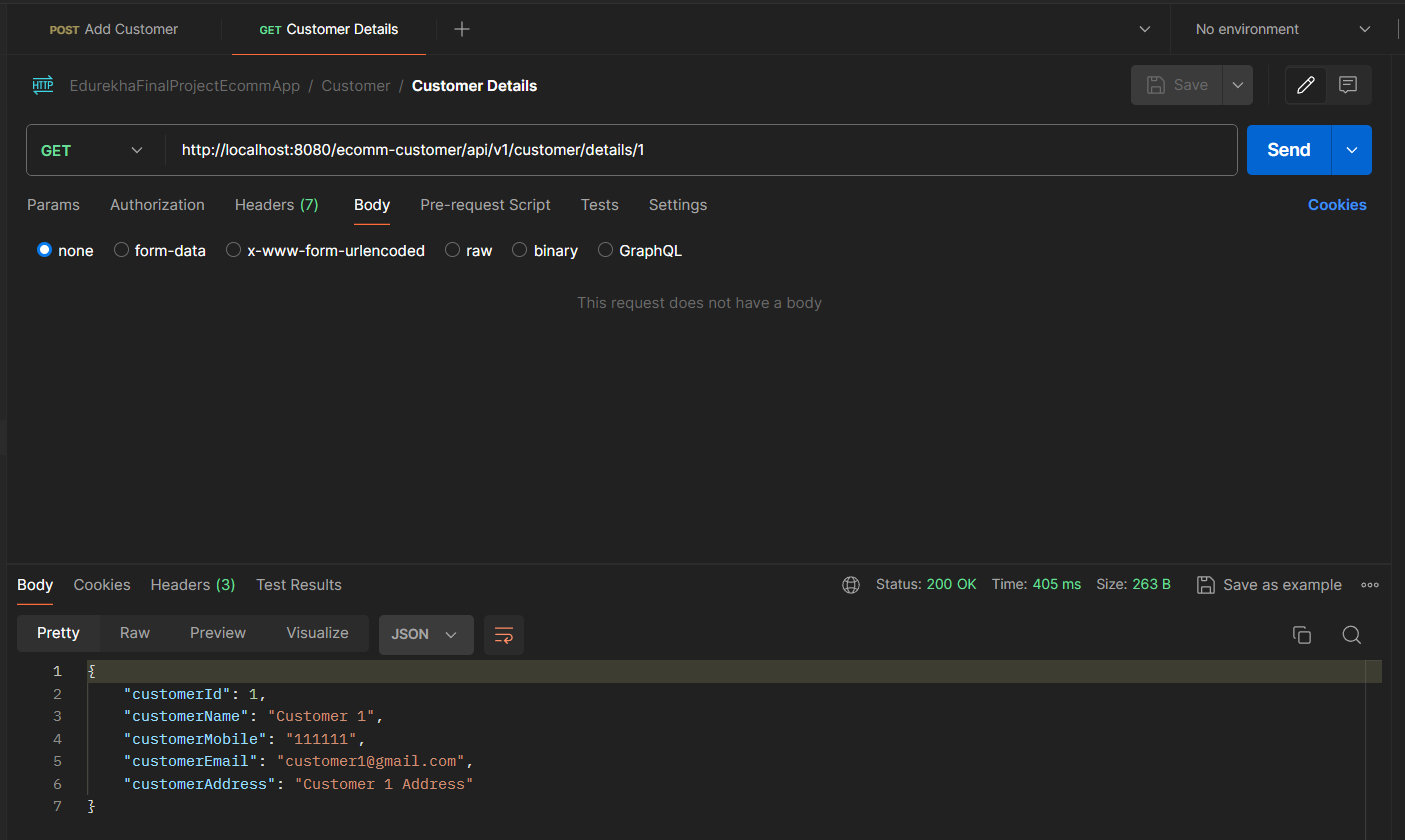
1. Verify the inventory details.



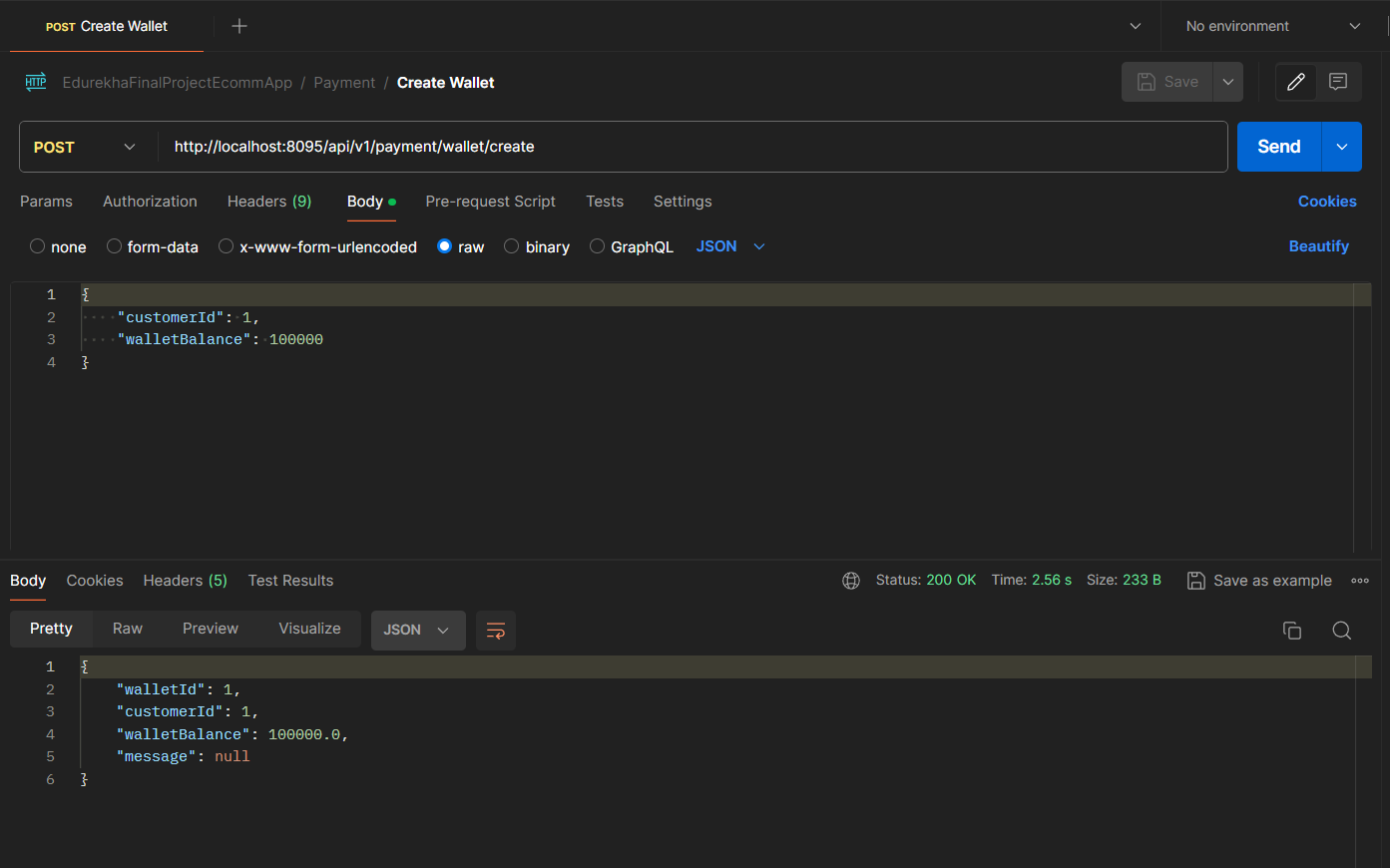
1. Create a customer profile.



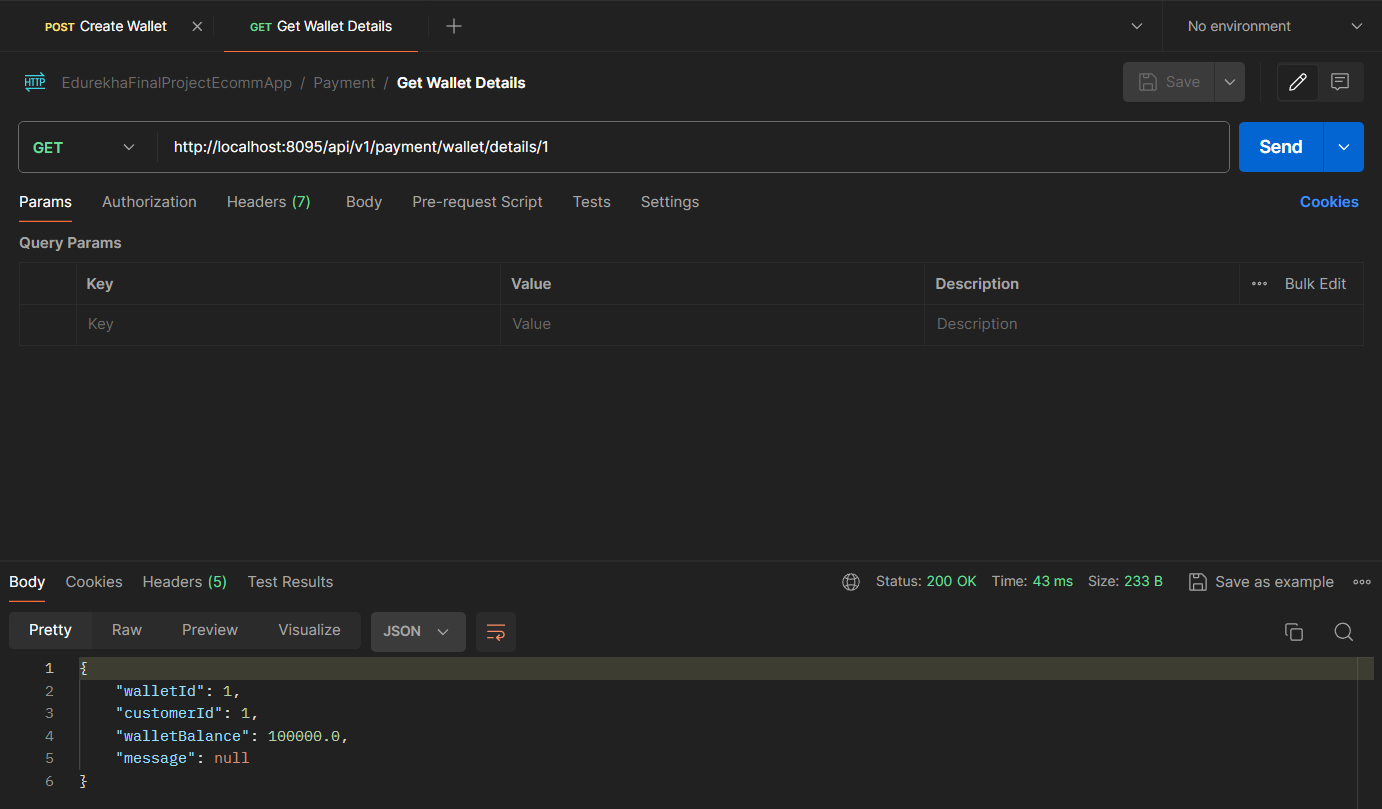
1. Verify the customer details.



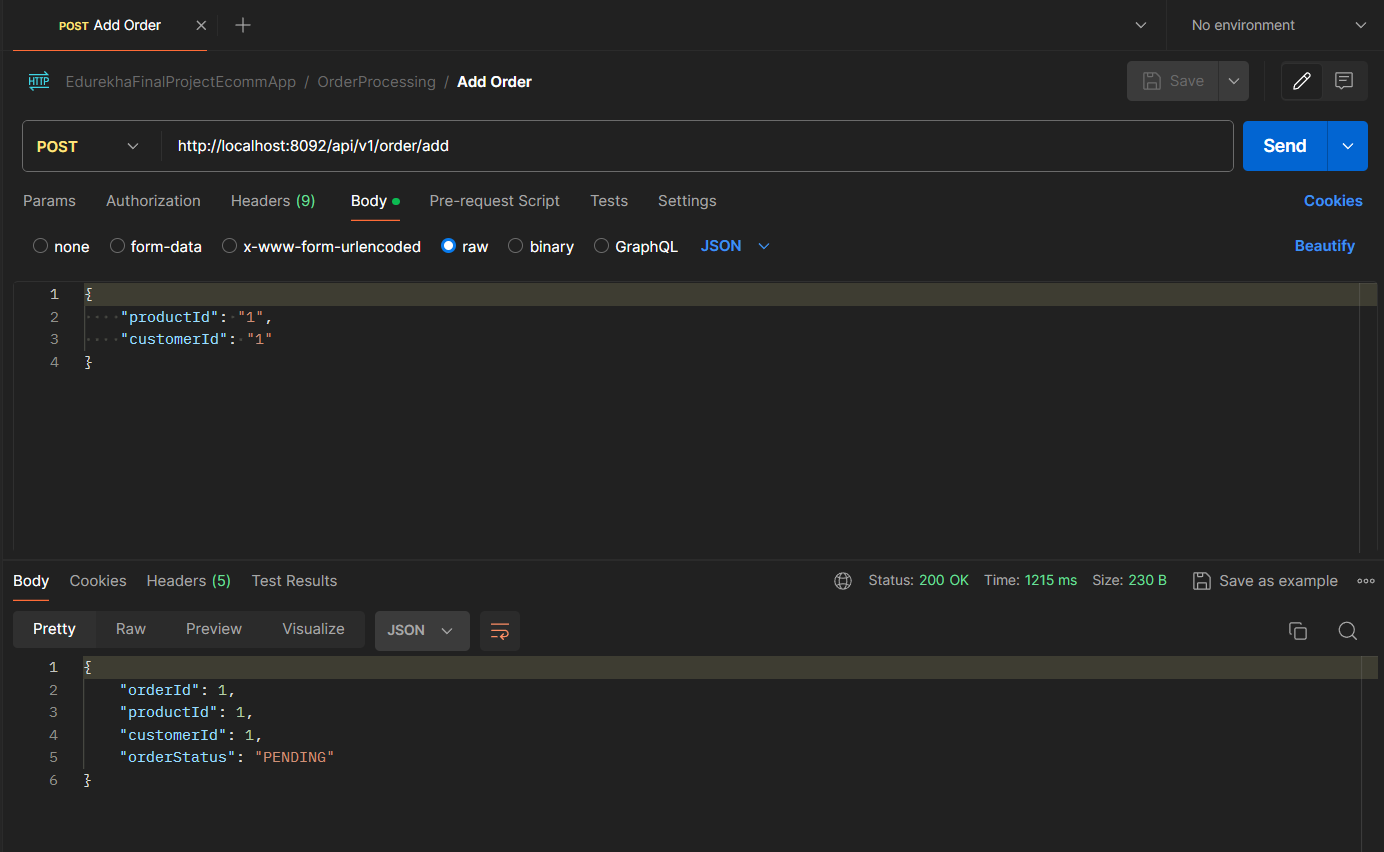
1. Create a customer payment wallet.



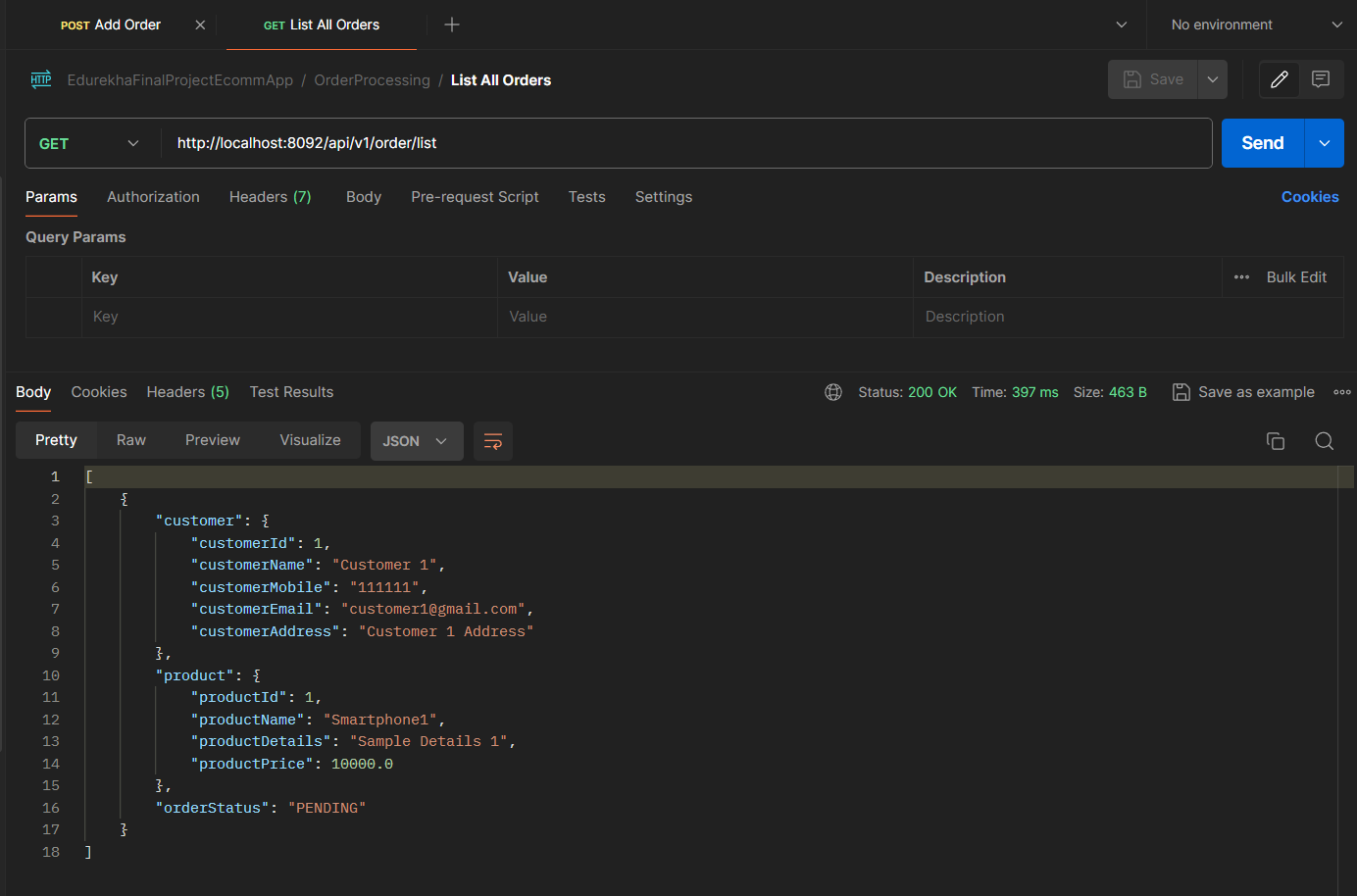
1. Verify the available wallet balance of the customer.



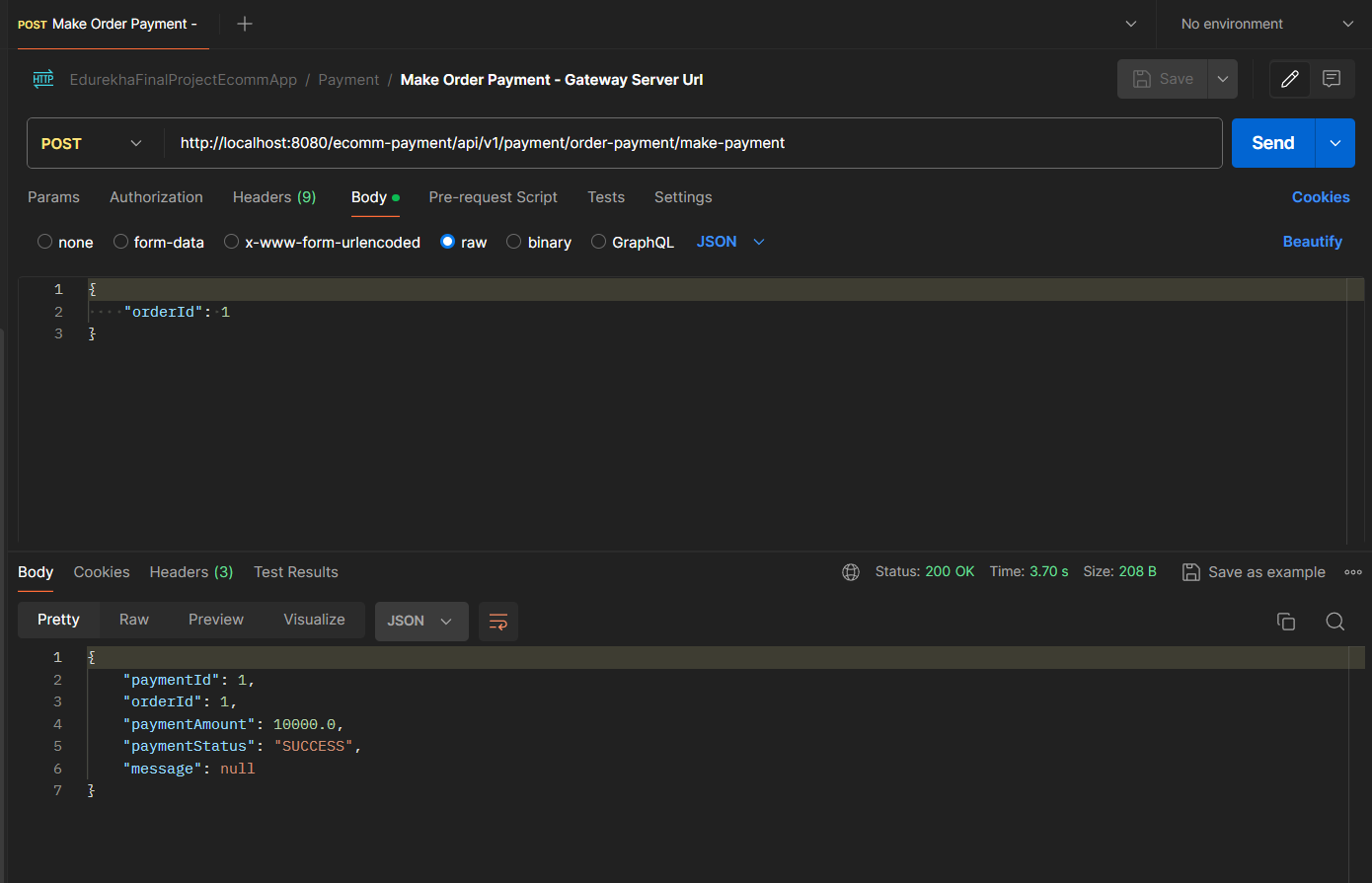
1. Create an order for customer – Customer1 places order for Product1.



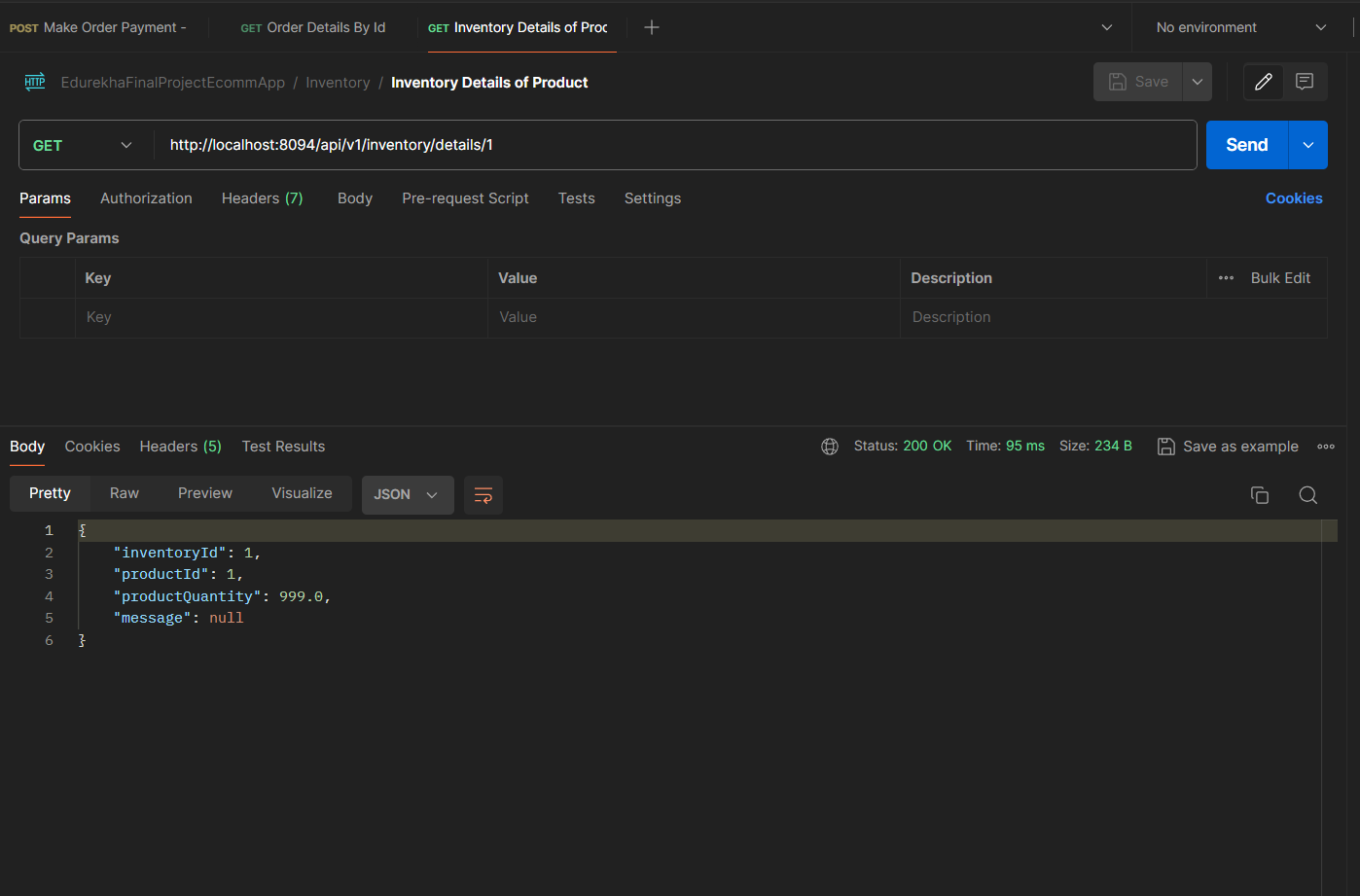
1. Verify the order details. The order will be in PENDING status until a payment is done.



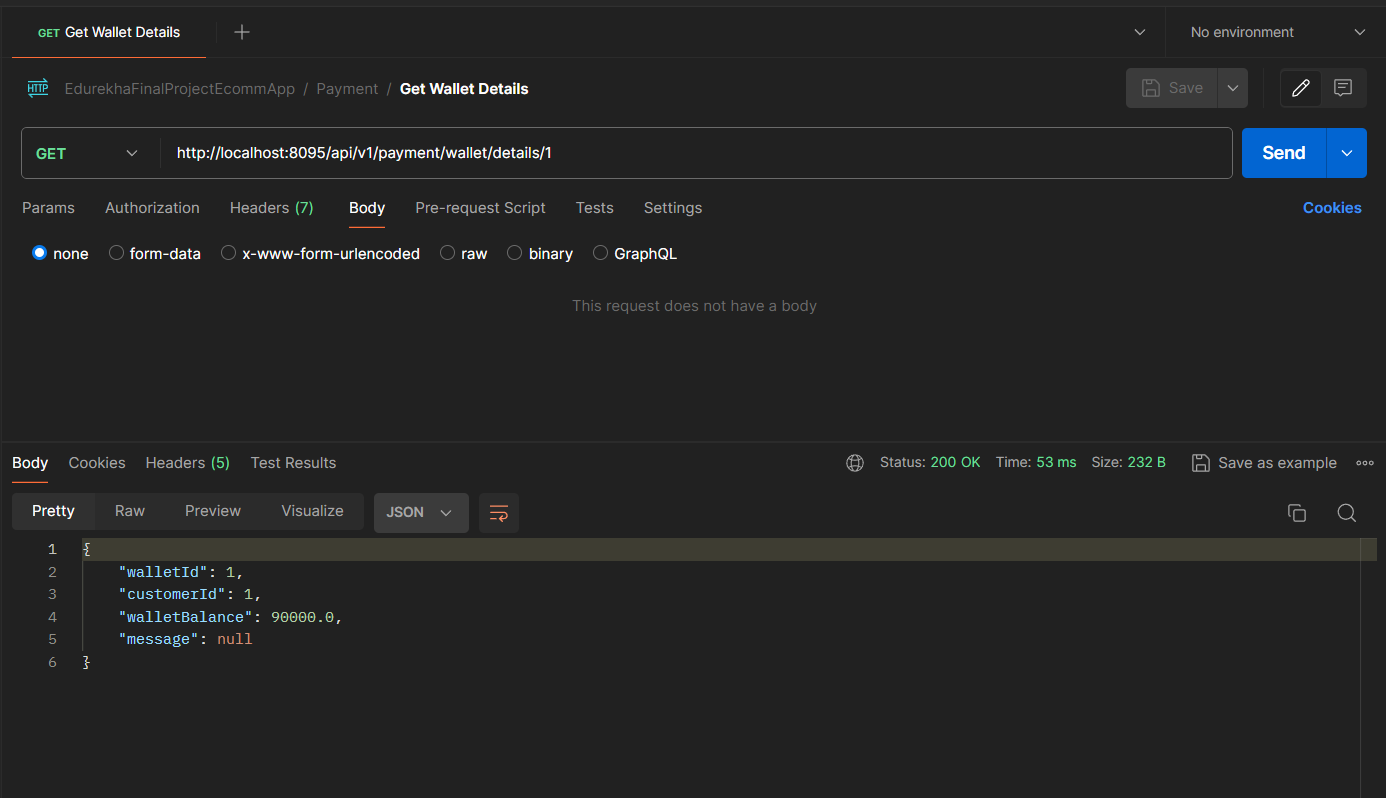
1. Make payment for the Order1. Note that all endpoints can also be accessed from the spring cloud gateway server url by appending appropriate service name.



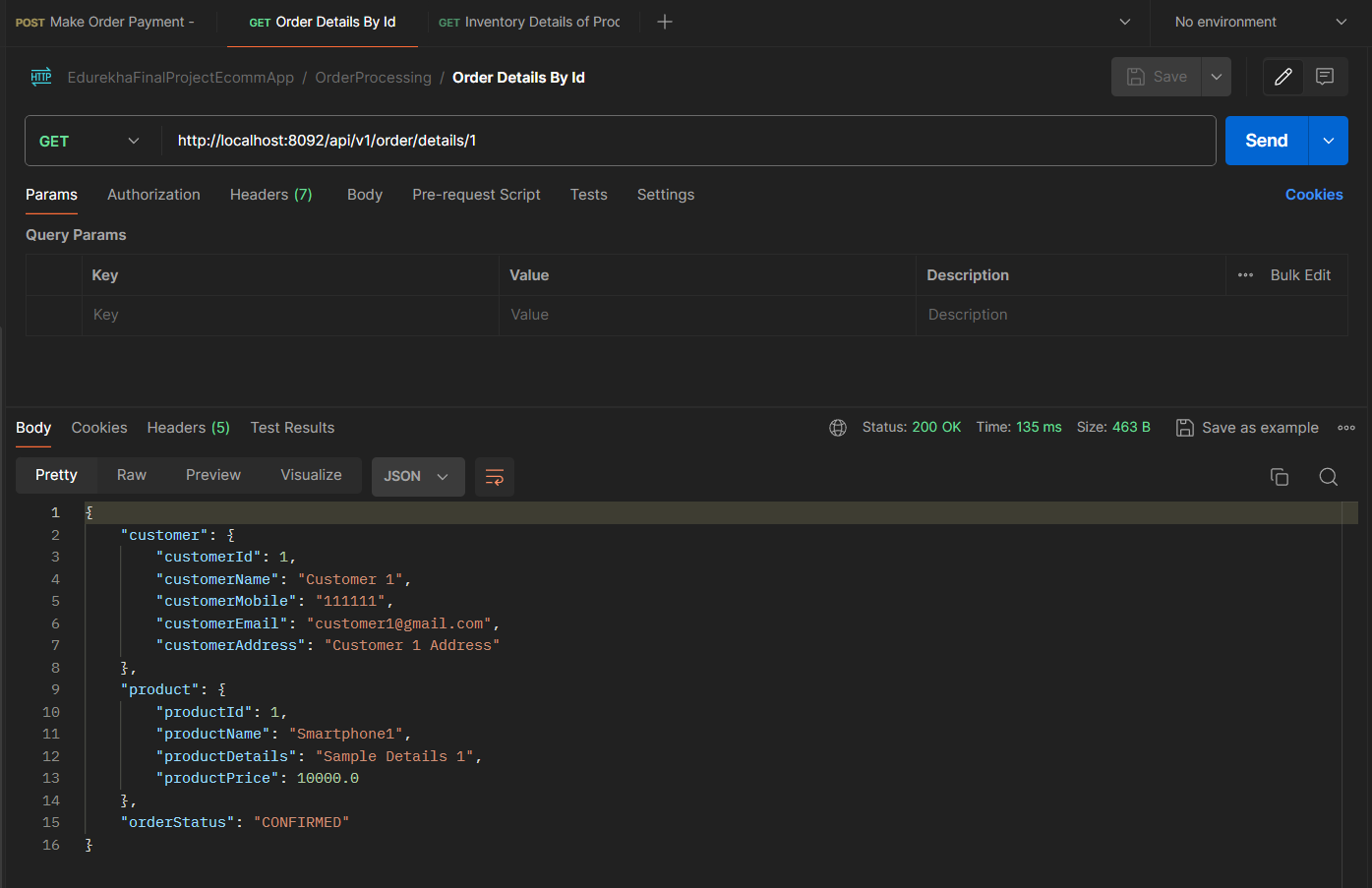
1. Verify the inventory is reduced by 1.



1. Verify that product price (Rs 10000) is debited from the customer wallet. So the available balance should be Rs 90000.

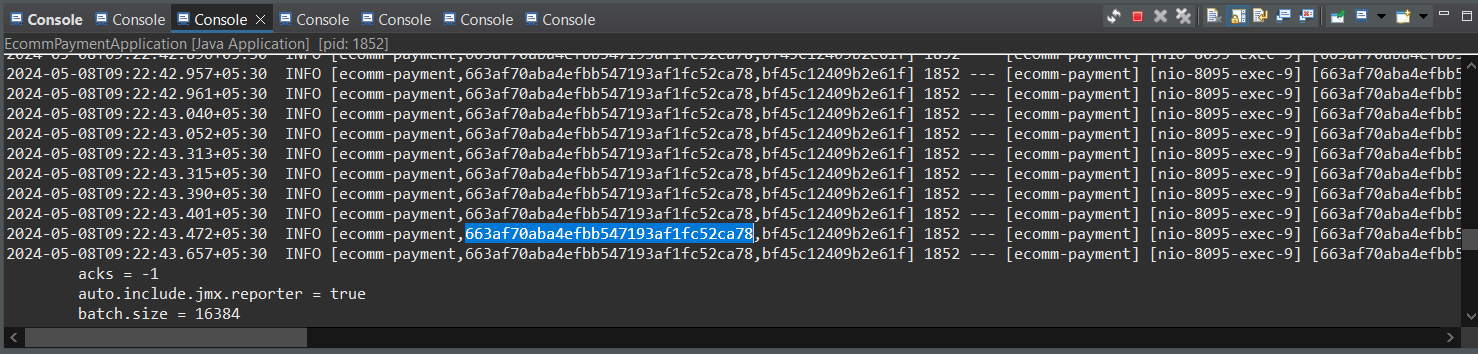


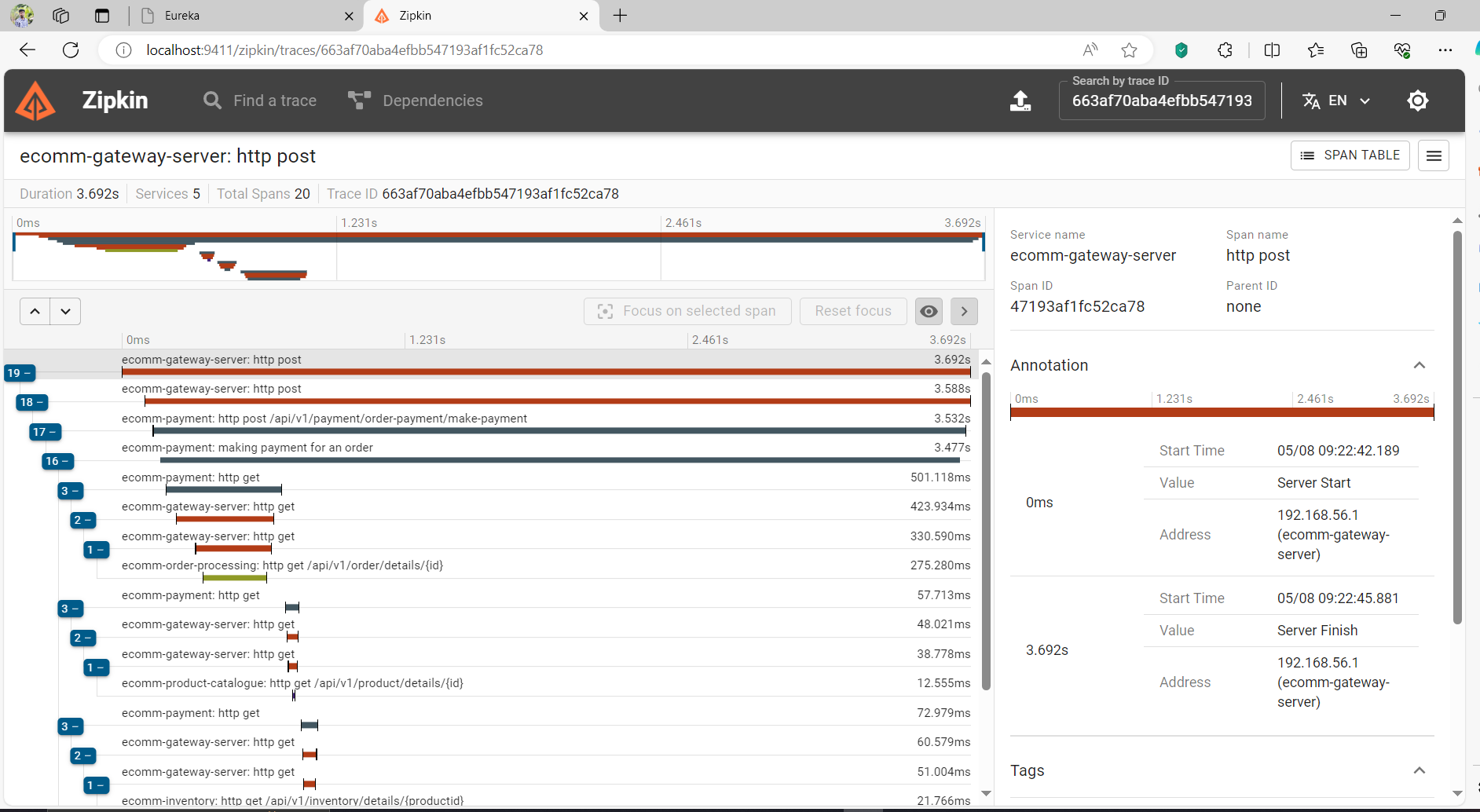
1. Once payment is success payment microservice will send message to kafka that order is confirmed. This message will be consumed by order processing service which updates the product status as ‘CONFIRMED’.

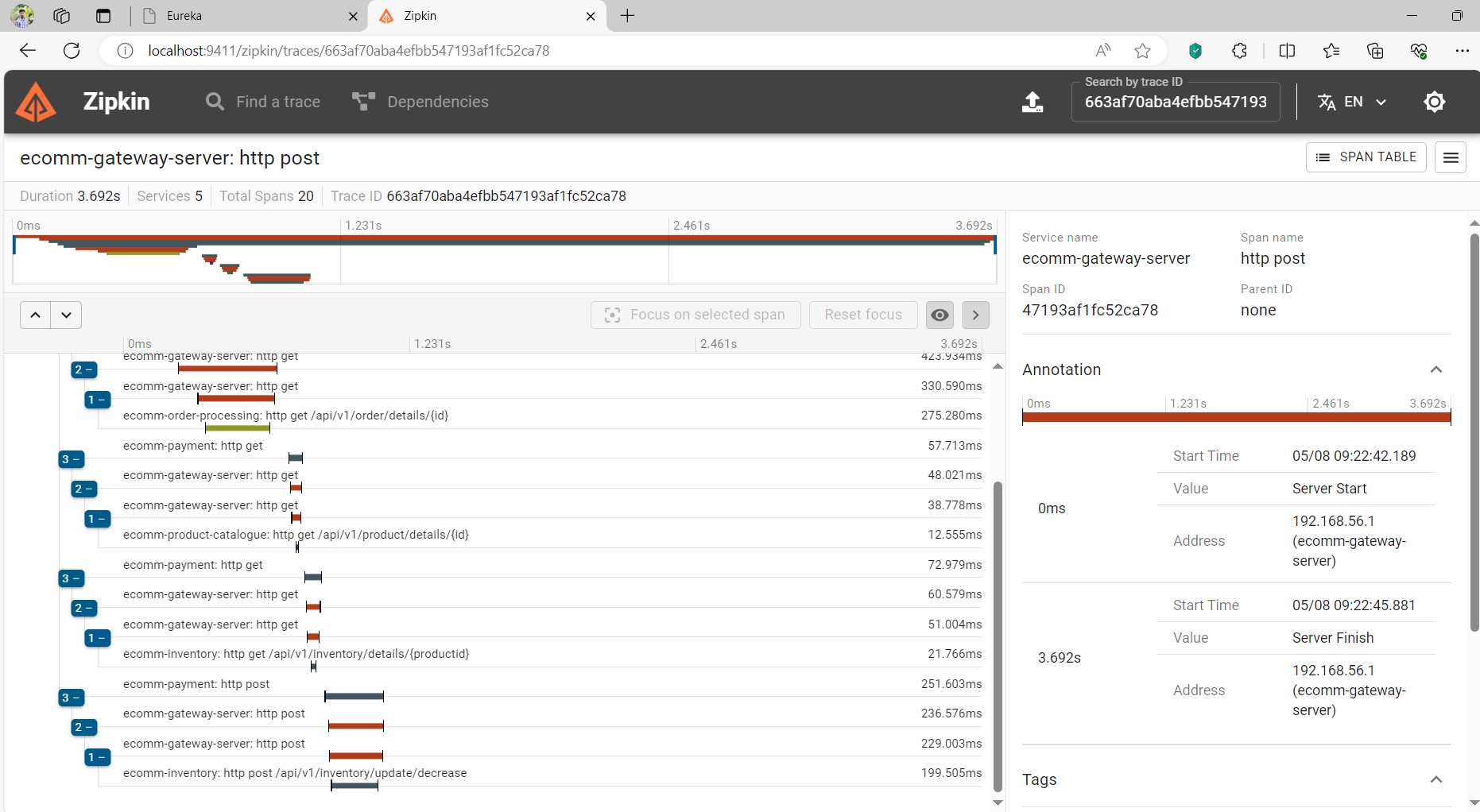


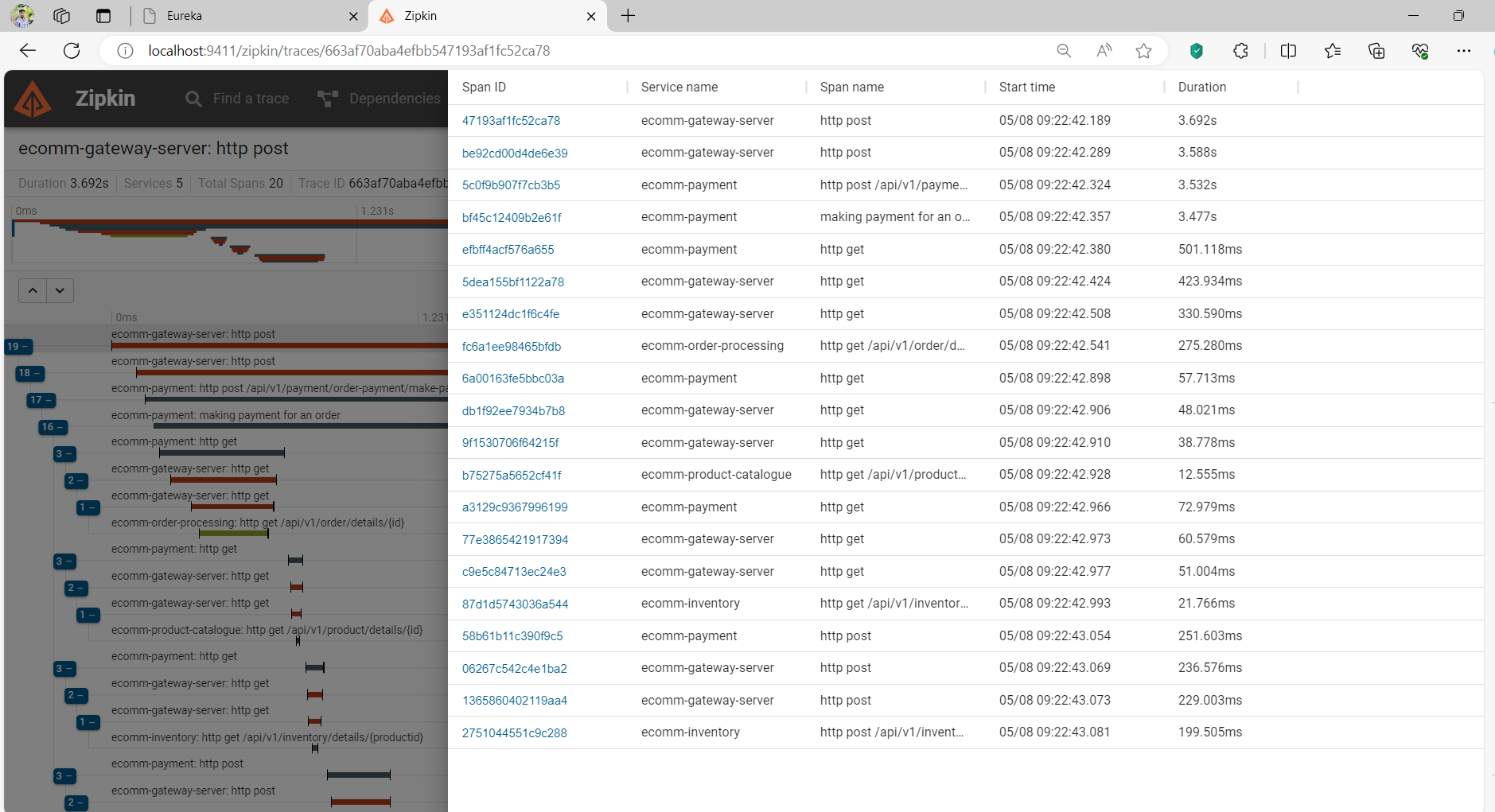
# Observability

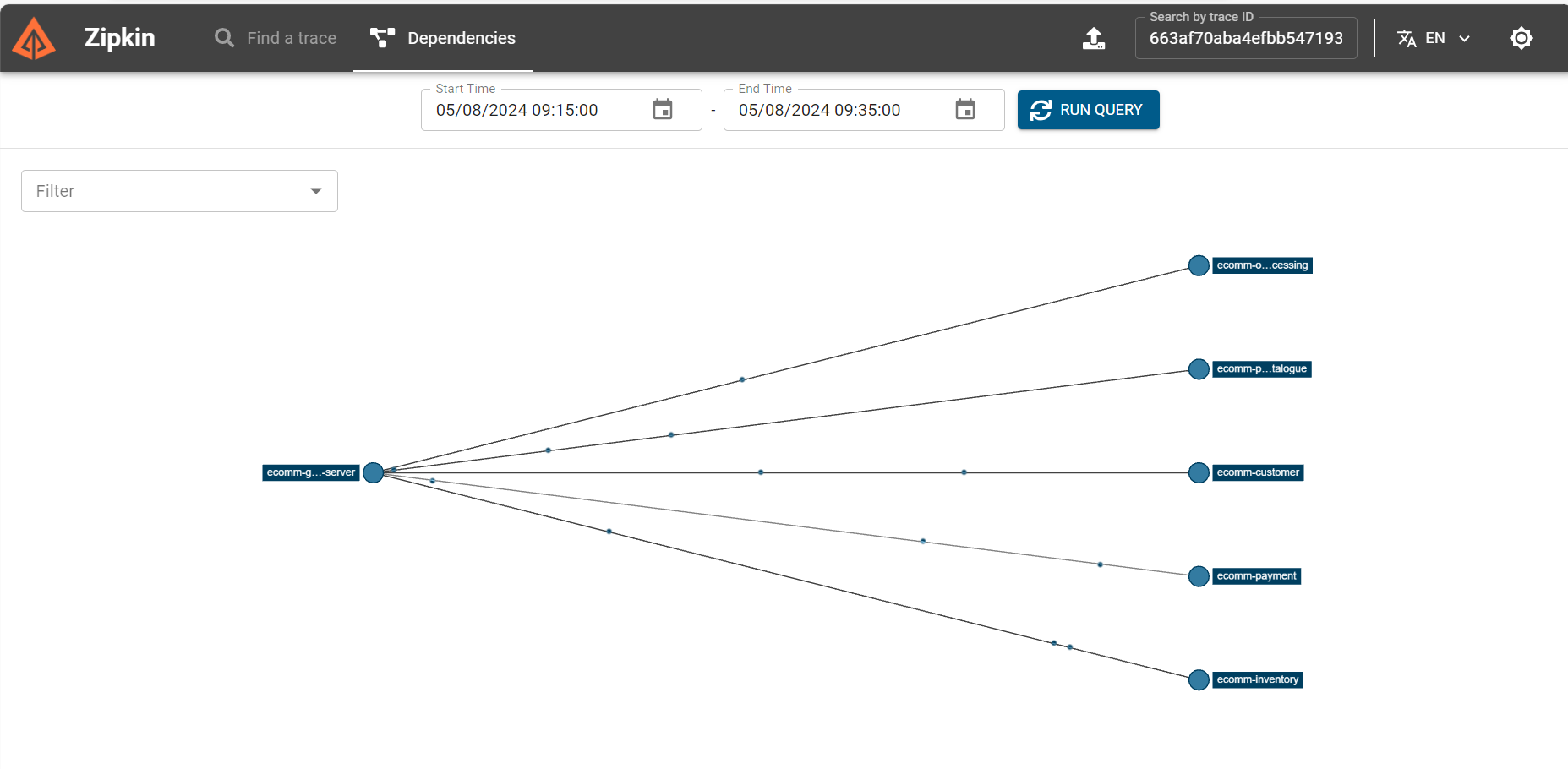
1. The trace id and span id for the above business flow is generated and can be verified in the zipkin server.



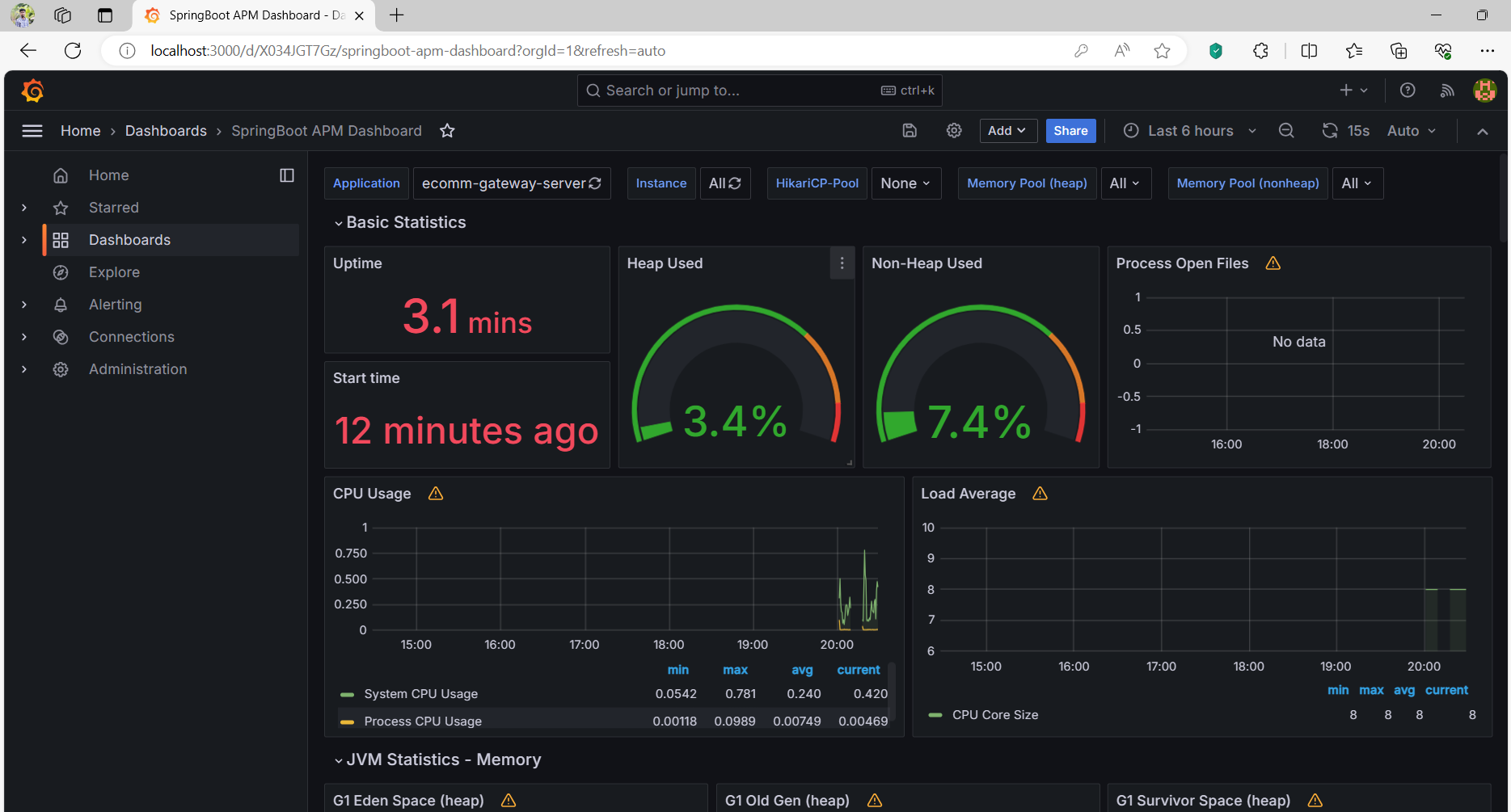


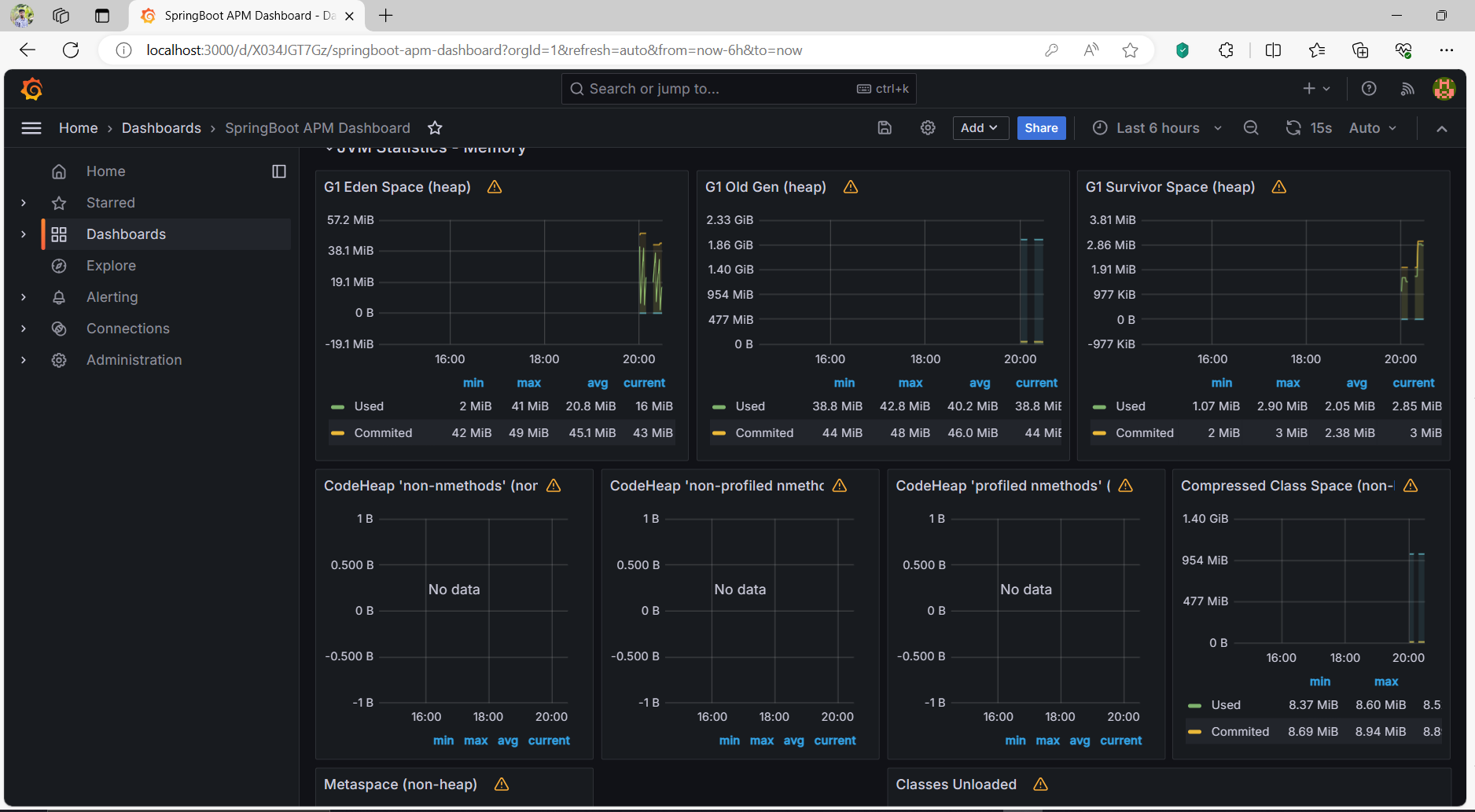


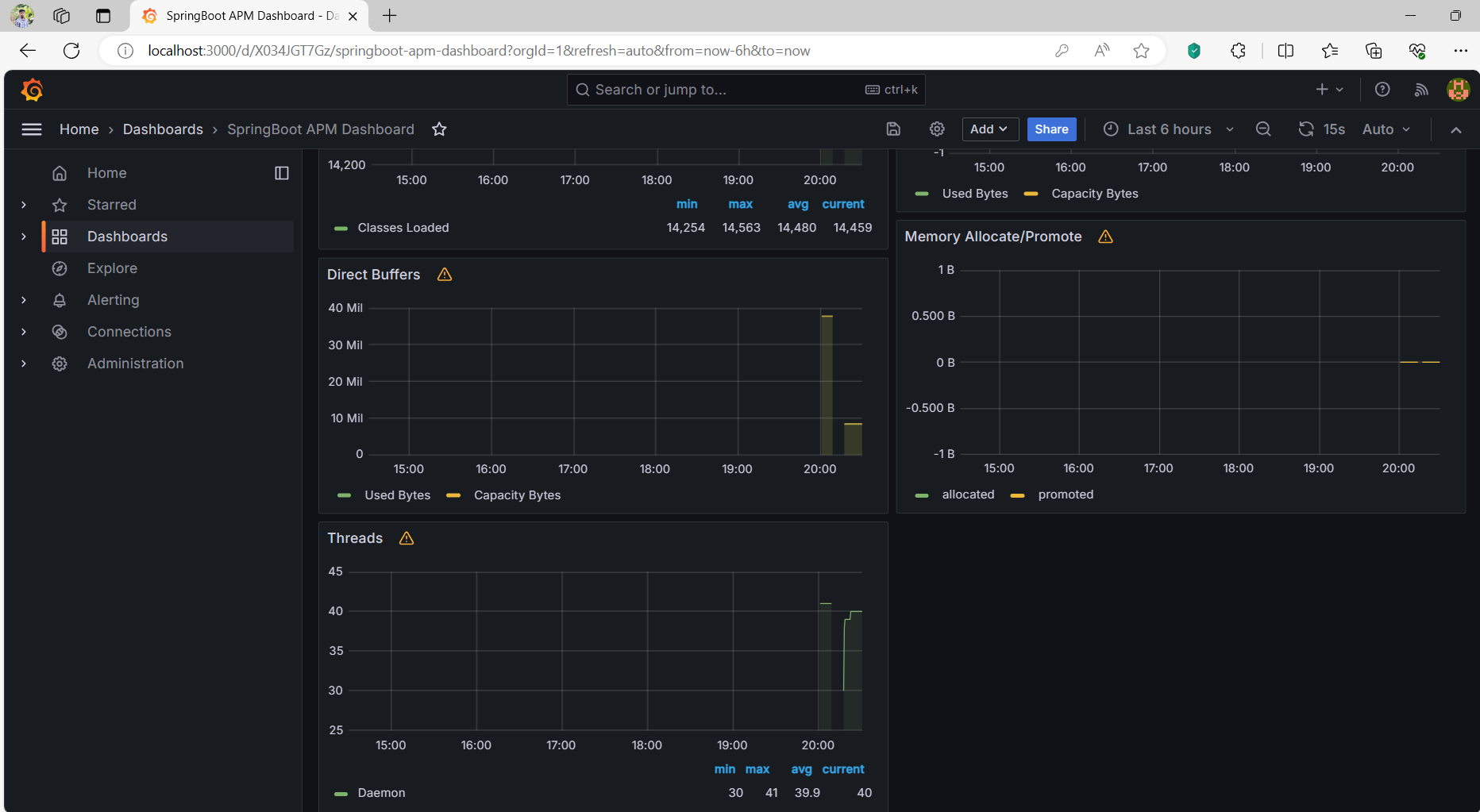




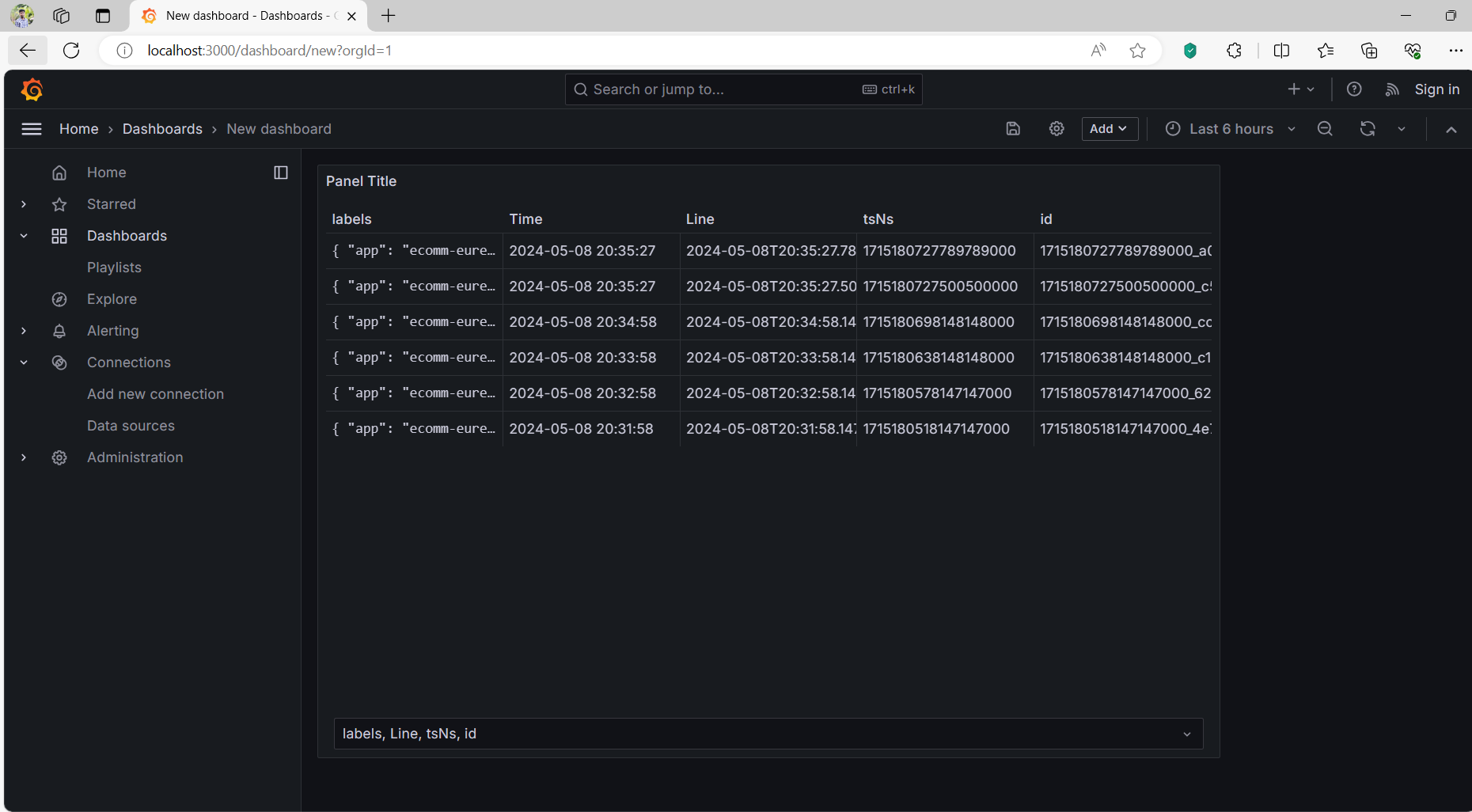
1. Graphana dashboard is configured to show the metrics from Prometheus:







1. Log aggregation is done using loki. The loki datastore is configured to display app logs in dashboard.



# Deployment

1. The app is built using dockerfile and pushed to docker hub.
2. Minikube is used to deploy the application to a Kubernetes cluster in local:

