# ****Roteiro de Testes Manuais****

### ****Introdução****

Este roteiro documenta os passos necessários para testar manualmente a aplicação, garantindo que os serviços essenciais como MongoDB, Redis e RabbitMQ estejam funcionando corretamente.

## ****Verificar se Todos os Containers Estão Rodando****

scripts/set-env-vars.sh  
scripts/recreate-containers.sh  
docker ps

✅ **Verifique se os seguintes serviços estão rodando:**  
✔ order-api (porta 8080)  
✔ rabbitmq (porta 5672)  
✔ mongodb (porta 27017)  
✔ redis (porta 6379)

❌ **Se algum serviço não estiver rodando, reinicie todos os containers:**

docker-compose down && docker-compose up -d –build

## ****Testar Conexão com o MongoDB****

docker ps | grep mongo  
docker-compose up -d mongo  
  
docker exec -it mongodb mongosh --eval "use orderdb; db.orders.find().pretty()"  
  
mongo --host localhost --port 27017  
  
docker logs $(docker ps -q --filter ancestor=mongo:7.0)

docker exec -it mongodb mongosh  
show dbs  
use orderdb  
show collections  
db.orders.find().pretty()

## ****Testar Conexão com o Redis****

docker rmi redis:6.2 -f

docker pull redis:6.2  
  
docker ps | grep redis  
docker-compose up -d redis

docker exec -it redis redis-cli keys "\*"  
docker exec -it redis redis-cli GET orderId:999

## ****Testar Conexão com o RabbitMQ****

docker ps | grep rabbitmq  
docker-compose up -d rabbitmq  
  
docker exec -it rabbitmq rabbitmqctl list\_queues

http://localhost:15672

**criar fila**  
docker exec -it rabbitmq rabbitmqctl add\_queue orders.generated.queue  
docker exec -it rabbitmq rabbitmqctl add\_queue orders.processed.queue

## ****Testar Publicação de Mensagem no RabbitMQ (Simulando Produto-Externo-A)****

docker exec -it rabbitmq rabbitmqadmin publish exchange=orders.generated.exchange routing\_key=orders.generated.key payload='{"orderId":"1234", "products": [{"name":"Produto Externo A","price":15.0,"quantity":2}] }'

## ****Testar o Consumo da Mensagem pela order-api****

docker logs order-api --tail 50

## ****Testar o Consumo da Mensagem na Fila****

docker exec -it rabbitmq rabbitmqadmin get queue=orders.generated.queue  
docker exec -it rabbitmq rabbitmqadmin get queue=orders.processed.queue  
  
  
**Testar a Publicação na Fila (Produto-A)**

docker exec -it rabbitmq rabbitmqadmin publish exchange=orders.generated.queue routing\_key=orders.generated.key payload='{"orderId":"1234","products":[{"name":"Produto X","price":50.0}]}' content\_type="application/json"  
  
  
  
**Verificar o Processamento do Pedido**  
  
docker exec -it rabbitmq rabbitmqadmin get queue=orders.processed.queue

### **Criar um Pedido via API**

curl -X POST http://localhost:8080/orders \

-H "Content-Type: application/json" \

-d '{"orderId":"999", "products": [{"name":"Produto Teste","price":50.0,"quantity":2}] }'

### **Consultar um Pedido via API**

curl -X GET http://localhost:8080/orders/999

### **Testar a Verificação de Duplicação**

- Executar POST

- Verificar log: docker logs order-api --tail 50

**SWAGGER**

http://localhost:8080/swagger-ui.html

{

"orderId": "123",

"items": [

{ "product": "Mouse", "quantity": 2, "price": 50.0 }

]

}

**TESTES**

mvn verify

mvn test -Dtest=InfrastructureTests  
mvn test -Dtest=OrderControllerIT  
  
**Infraestrutura com Testcontainers**  
mvn failsafe:integration-test -Dtest=\*IT  
  
**Testes Unitários**  
mvn test

**ACTUATOR**  
<http://localhost:8080/actuator>  
http://localhost:8080/actuator/metrics

**ESCALABILIDADE**  
docker-compose up --scale order-api=3