Node.js Developer Technical Assessment Documentation

FABA INTERNATIONAL
فابا انترناشيونال

Murat TUNÇ

Senior Back-End & Cloud Software Development Engineer

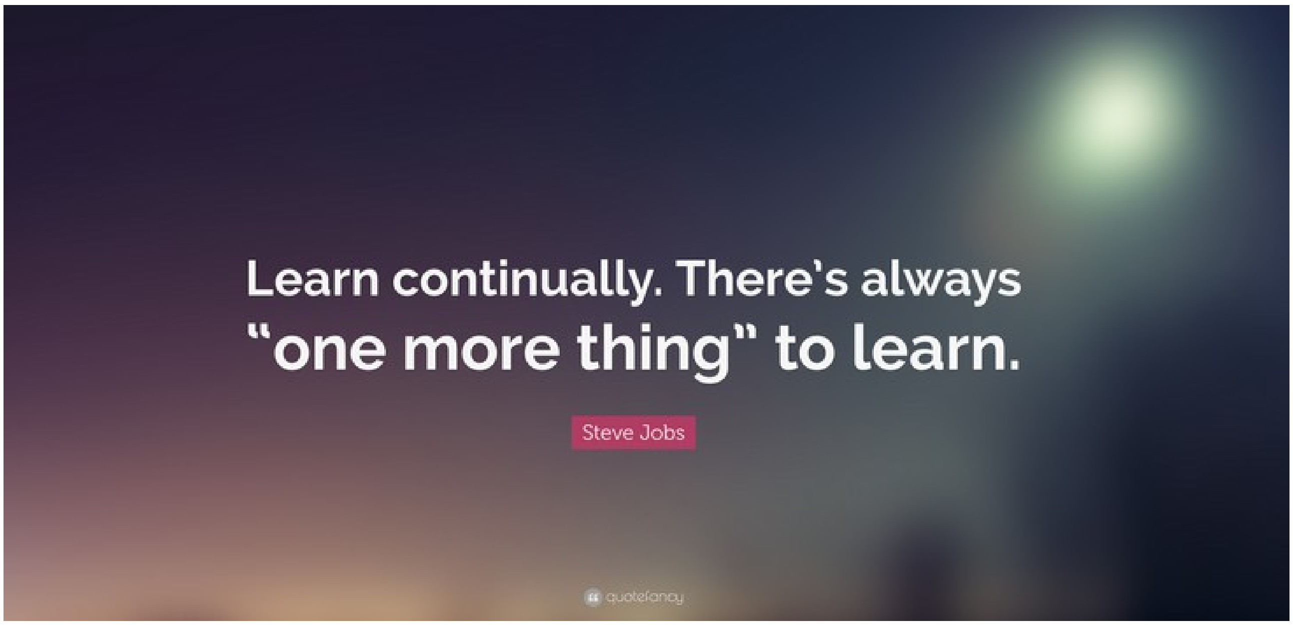
Golang -NodeJS-Java & React-NextJS with Linux Based Microservices

📍 Istanbul, Turkey – Kadıkoy | 📧 murat.tunc8558@gmail.com | 📞 +90 (531) 731-58-54
🔗 GitHub | LinkedIn | 🌐 Dev.to | Medium

📘 Documentation Index

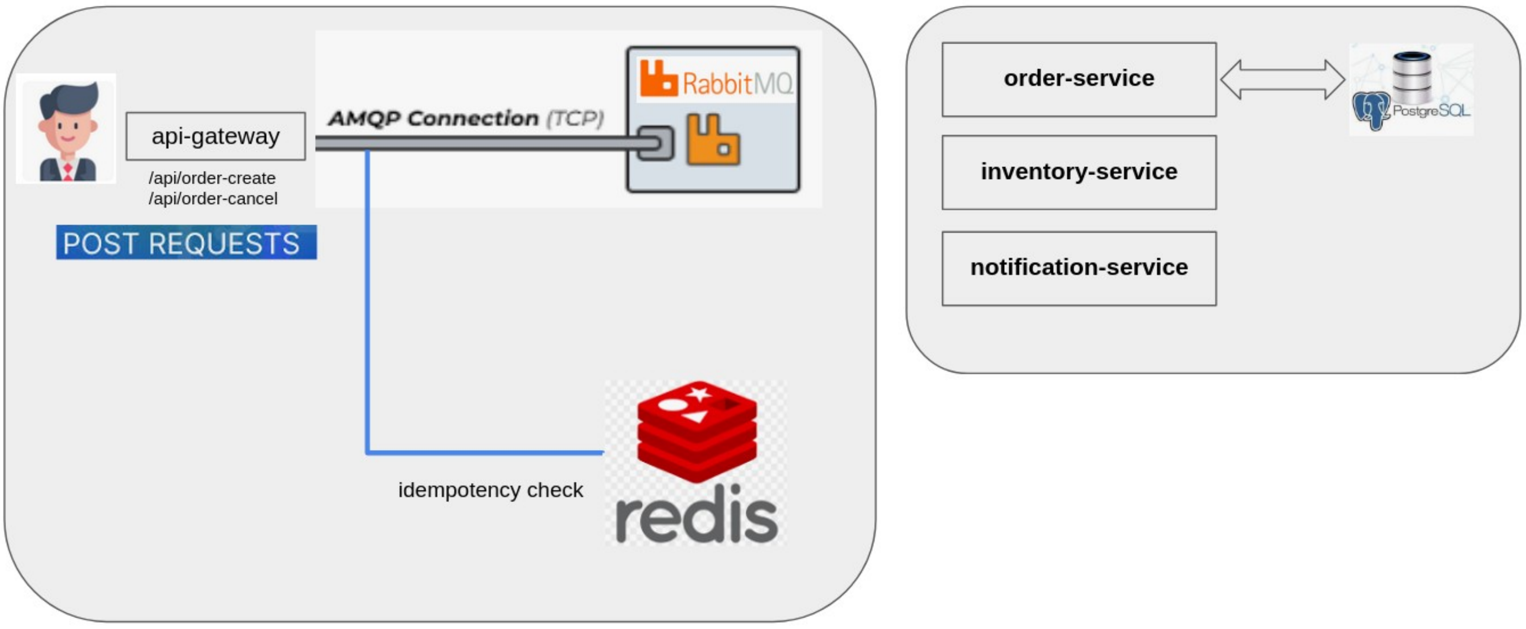
INTRODUCTION

In this study, I will try to briefly explain the micro service architecture that I designed in the Node.js environment using RabbitMQ and Redis storage methods. I will definitely have some shortcomings, please send your feedback to my contact information, I would be very happy. Good readers in advance.



SYSTEM OVERVIEW

I divided the system into 2 parts to make it easier to understand, you can think of it as the door and the inside of the room.The part on the left side performs the necessary checks on the http requests coming from the user and forwards them to the RabbitMQ queue structures. On the right side, there are micro services. Each micro service can work independently. This is also **event-driven microservices architecture** powered by **RabbitMQ** for asynchronous communication between decoupled services.

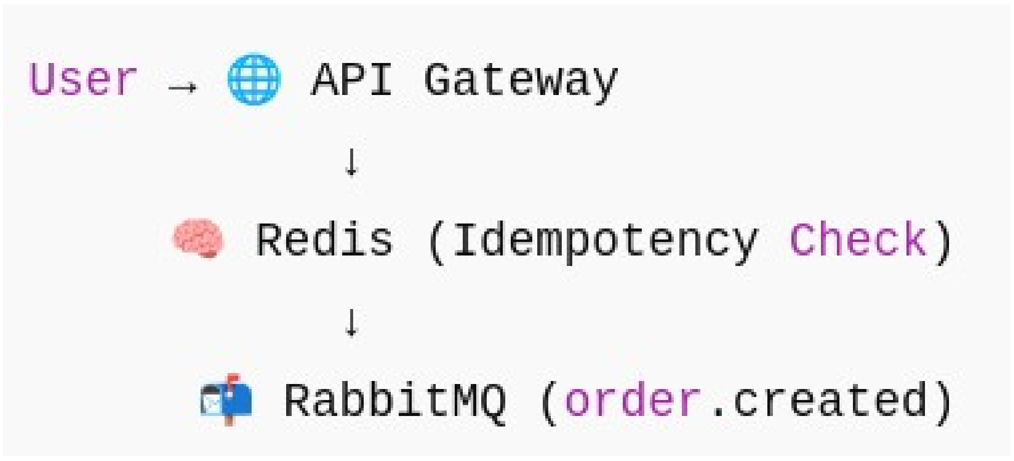


From the API Gateway, the order information is published into **four distinct RabbitMQ channels**: one for new orders, one for cancellations, and their respective **Dead Letter Queues (DLQs)**. These channels are consumed by the **Order Service**, which handles storing the order in its database and emits an order.created event to the order-events exchange. It also listens for cancellations via a separate queue and emits an order.cancelled event when needed.

ARCHITECTURE DIAGRAM

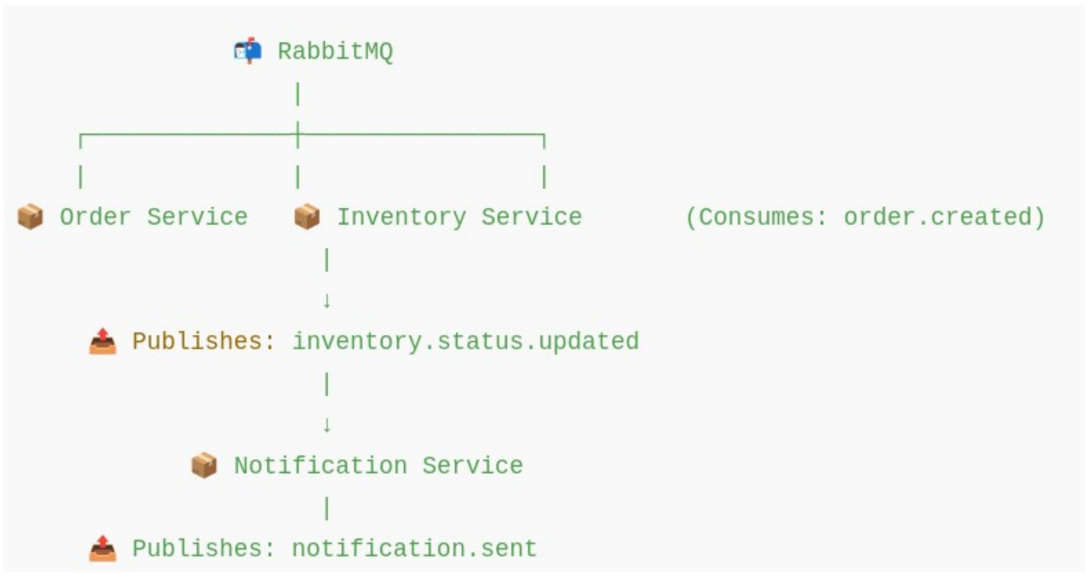
When we look at the system architecture, a flow occurs in the following structure.

First Part



```
User → 🌐 API Gateway
              ↓
      🧠 Redis (Idempotency Check)
              ↓
       📫 RabbitMQ (order.created)
```

Second Part



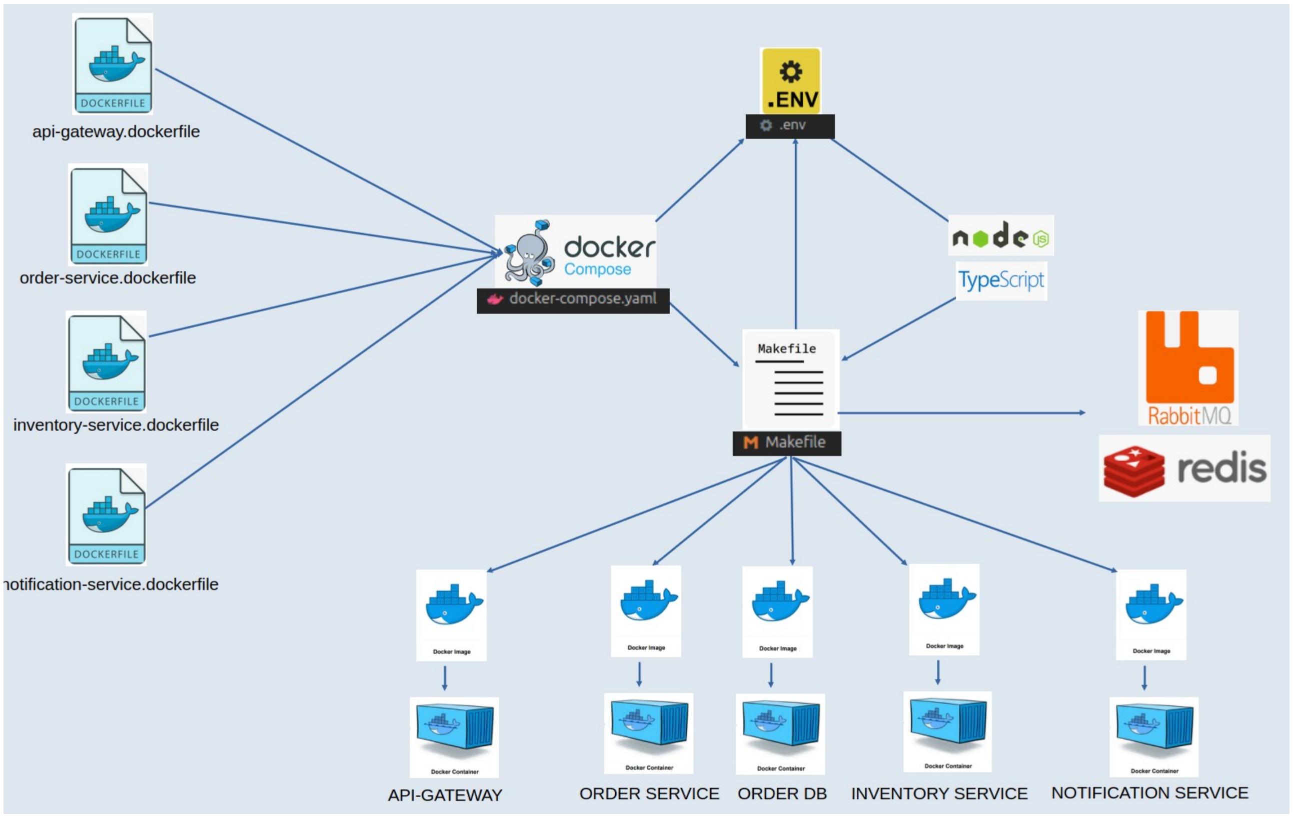
```
              📫 RabbitMQ
                  |
     ┌────────────┼────────────┐
     |            |            |
📦 Order Service  📦 Inventory Service      (Consumes: order.created)
                    |
                    ↓
    📤 Publishes: inventory.status.updated
                    |
                    ↓
          📦 Notification Service
                    |
    📤 Publishes: notification.sent
```

BUILDING TOOLS

All services rabbtimq and redis services, all settings of the postgres database are in the .env file. docker-compose.yaml, Makefile and Project Typescript files also read variable values in the .env file.In order to prepare the runtime environment (for Makefile and compose.yaml file), the prepare_development_pc.sh script must be run.

```
📁 build-tools
├── 📄 .env
├── 📄 docker-compose.yaml
├── 📄 init-order-db.sql
├── 📄 integration-test.sh
├── 📄 Makefile
├── 📄 prepare_development_pc.sh
```

Project compilation diagram



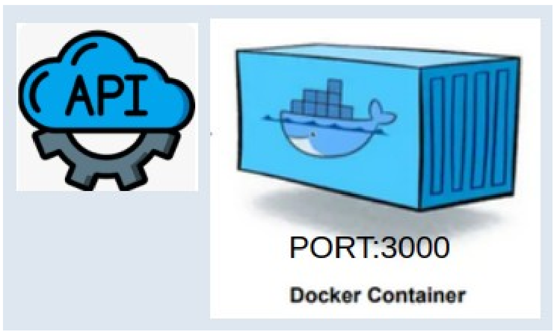
MICROSERVICE DESCRIPTION

There are 3 micro services operating on the backend side. These are order-service, inventory-service and notification-service.

API-GATEWAY

The api-gateway service receives http requests coming from outside the application, uses the redis service for idempotency compliance, stores the "idempotency_key" value coming with the post request in redis, and prevents the same request from being processed repeatedly.

```
📁 api-gateway
├── 📄 api-gateway.dockerfile
├── 📁 node_modules
├── 📄 package.json
├── 📄 package-lock.json
├── 📁 src
│   ├── 📁 config
│   ├── 📁 handlers
│   │   ├── 📄 order-cancel.ts
│   │   └── 📄 order-create.ts
│   ├── 📄 index.ts
│   ├── 📁 middleware
│   │   ├── 📄 errorHandler.ts
│   │   ├── 📄 idempotencyMiddleware.ts
│   │   ├── 📄 loggingMiddleware.ts
│   │   └── 📄 requestLogger.ts
│   ├── 📁 models
│   ├── 📁 queues
│   ├── 📁 routes
│   │   └── 📄 routes.ts
│   ├── 📁 services
│   │   └── 📄 rabbitmqService.ts
│   └── 📁 utils
│      ├── 📄 custom-error.ts
│      ├── 📄 gracefulShutdown.ts
│      └── 📄 logger.ts
├── 📄 tsconfig.json
```



API Gateway architecture

```
+--------------------------------------------------+
|                  API Gateway                      |
|   (HTTP Requests to API Gateway -> RabbitMQ)      |
+--------------------------------------------------+
         |
         v
+-------------------------+      +---------------------------+
|     Redis (Idempotency)|      |  Redis (Idempotency)      |
|   (Check if message has |      | (Check if cancel message)|
|   already been processed)|      |   has already been        |
+-------------------------+      |     processed             |
         |                        +---------------------------+
         v
+-------------------------+      +---------------------------+
|   ORDER_QUEUE           |      |  ORDER_CANCEL_QUEUE       |
|   (Main Queue)          |      |  (Main Cancel Queue)      |
|   Messages from Users   |      |  Cancel Messages          |
|   inserted by API       |      |  inserted by API          |
+-------------------------+      +---------------------------+
         |                                  |
         v                                  v
+-------------------------+      +---------------------------+
| ORDER_DLQ (Dead Letter)|      | ORDER_CANCEL_DLQ (DLQ)    |
| (For Failed Orders)     |      | (For Failed Cancels)      |
+-------------------------+      +---------------------------+
         ^                                  ^
         |                                  |
         +----------------------------------+
                      RabbitMQ Server
                      (Handles Queue Management)
```

Gracefulshutdown handler:

App listens for OS signals (`SIGINT`, `SIGTERM`)

- When received, it:

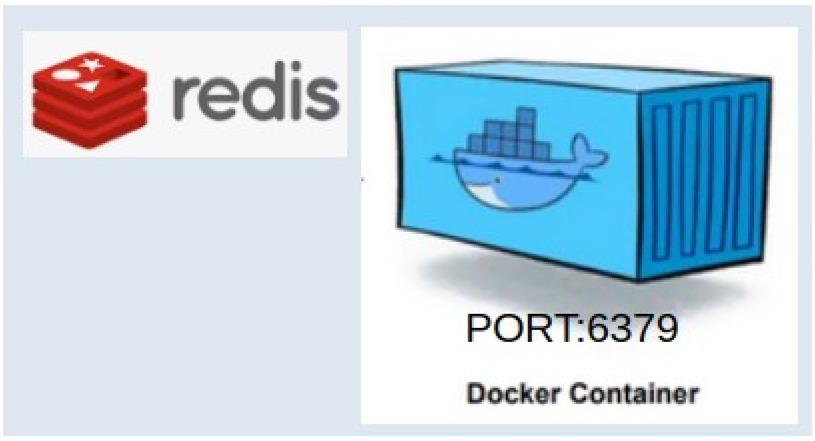
1. Closes the RabbitMQ channel
2. Closes the HTTP server
3. Logs cleanup results
4. Exits the process

- If cleanup takes more than 10 seconds → force shutdown

```
                 [ Application Running ]
                         |
     -----------------------------------------------
     |                                              |
[Handling HTTP Requests]                  [RabbitMQ Channel Active]
     |                                              |
     |<----------------- Signals ----------------->|
     |          (SIGINT or SIGTERM received)        |
     |                                              |
     --------------------+-----------------------
                          |
                [Start Graceful Shutdown]
                          |
          ---------------------------------
          |                                |
[Close RabbitMQ Channel]        [Close HTTP Server]
     |                                   |
[Log success or error]           [Log "connections closed"]
     |                                   |
     ------------> [Exit process (0)] <-------------
                          |
                 (if takes too long...)
                          |
              [Force shutdown after 10s]
                          |
                  [Exit process (1)]
```

REDIS

The idempotency check happens here. If the request has been processed already (i.e., exists in Redis with a specific key), it prevents further processing. If not, it allows the request to proceed and stores the ID for future checks.



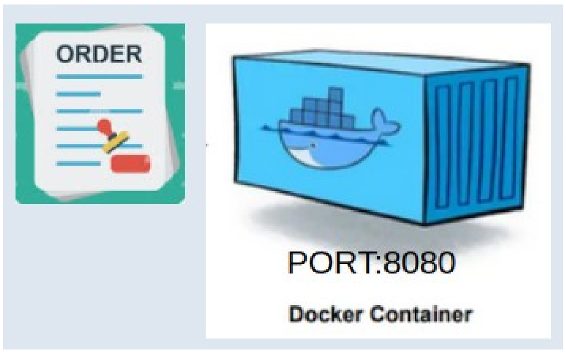
ORDER-SERVICE

Order-service receives data from 4 queues in rabbitmq. Requests coming from the user with api-gateway service are stored in 4 queues by api gateway service, and this stored information is absorbed by order-service.

Responsible for handling all aspects of order processing within the system. It listens to RabbitMQ queues for incoming order messages, including newly created orders and order cancellations. When a new order message is received from the orderQueue, it parses the data, saves the order to the database, and then emits an order.created event to notify other services—such as the Inventory or Notification services—about the new order. Similarly, it handles cancellation requests by consuming messages from the orderCancelQueue, processing the cancel data, and publishing an order.cancelled event.

This service also incorporates Dead Letter Queue (DLQ) consumers (orderQueue_DLQ and orderCancelQueue_DLQ) to ensure failed messages are not lost; it retries or logs them for further investigation. The event messages it emits are versioned (starting with v1) to support future evolution without breaking consumers. Overall, the Order Service acts as the entry point for order-related events in the system's event-driven architecture, providing a clean and reliable interface for order lifecycle management.

```
📁 order-service
├── 📄 order-service.dockerfile
├── 📁 node_modules
├── 📄 package.json
├── 📄 package-lock.json
├── 📄 tsconfig.json
└── 📁 src
    ├── 📁 config
    │   ├── 📄 appconfig.ts
    │   └── 📄 database.ts
    ├── 📁 handlers
    ├── 📄 index.ts
    ├── 📁 middleware
    │   └── 📄 errorHandler.ts
    ├── 📁 models
    │   └── 📄 order.ts
    ├── 📁 queues
    │   └── 📄 orderConsumer.ts
    ├── 📁 routes
    ├── 📁 services
    │   └── 📄 rabbitmqService.ts
    └── 📁 utils
        ├── 📄 custom-error.ts
        └── 📄 logger.ts
```



Event versioning:

```
├── 📁 queues
│   └── 📄 orderConsumer.ts
```

In this file, event versioning is handled by attaching a version field to every outgoing event message. The constant EVENT_VERSION is set to 'v1', and this value is used whenever an event is published to RabbitMQ. For example, when an order is successfully saved to the database, an order.created event is published with both the event name and version ({ event: ORDER_CREATED_EVENT, version: EVENT_VERSION, data: savedOrder.toJSON() }).

order-service architecture

```
+--------------------+         +----------------------------+
|   API Gateway      |         |   RabbitMQ (Main Queue)  |
| (HTTP Requests)    |         |   - ORDER_QUEUE          |
|                    |         |   - ORDER_CANCEL_QUEUE   |
+--------------------+         +----------------------------+
        |                              |
        v                              v
+------------------------+     +------------------------------+
| Order Service          |     | Consume from ORDER_QUEUE    |
| - Consume Orders       |     | (Process Orders)           |
| - Consume DLQ          |     |                             |
| - Consume Cancel       |     +------------------------------+
|   Orders               |
+------------------------+
        |
        v
+-----------------------------+        +-----------------------------+
|    Publish Order Created    |        |    Publish Order Cancelled    |
|    Event to EXCHANGE        |        |    Event to EXCHANGE          |
|    - ORDER_CREATED_EVENT    |        |    - ORDER_CANCELLED_EVENT    |
+-----------------------------+        +-----------------------------+
        |                                   |
        v                                   v
+------------------------+      +------------------------------+
|  RabbitMQ (DLQ)        |      |  RabbitMQ (Cancel DLQ)      |
|  - ORDER_DLQ           |      |  - CANCEL_DLQ               |
|  (For Failed Orders)   |      |  (For Failed Cancels)       |
+------------------------+      +------------------------------+
```

error handling:

└── 📁 utils
    ├── 📄 custom-error.ts

The CustomError class extends the built-in JavaScript Error class to create a more structured and informative error object tailored for our application's needs.

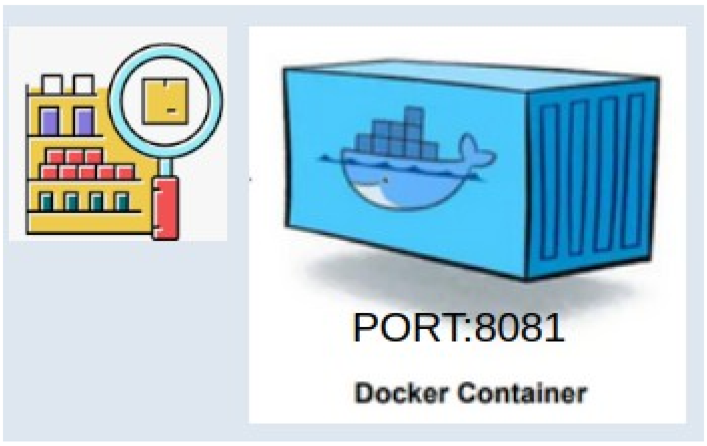
```
┌────────────────────────────┐
|        CustomError         |
|     (extends Error)        |
└────────────┬───────────────┘
             |
             ▼
┌─────────────────────────────────────────┐
| constructor(                            |
|   code: string,                         |
|   message: string,                      |
|   statusCode: number = 500,             |
|   details: any = {}                     |
| )                                       |
└─────────────────────────────────────────┘
             |
             ▼
     ┌───────────────────────────┐
     | super(message)            |  ◄── Calls parent Error constructor
     └───────────────────────────┘
             |
             ▼
     ┌───────────────────────────┐
     | this.code = code          |
     | this.message = message    |
     | this.statusCode = statusCode |
     | this.details = details    |
     └───────────────────────────┘
             |
             ▼
┌───────────────────────────────────────────────┐
| Error.captureStackTrace(this, this.constructor)|
| ► Captures a clean stack trace for debugging  |
└───────────────────────────────────────────────┘
```

INVENTORY-SERVICE

The inventory-service is responsible for reacting to new orders placed in the system. When the order-service publishes an order.created event, the Inventory Service consumes this message through RabbitMQ. It processes the order data—typically checking whether the required items are in stock or simply simulating that action—and then generates a new event, inventory.status.updated, to communicate the outcome.

This event is published back to the message broker, specifically to the inventory-events exchange, where other services like the Notification Service can consume it.

```
📁 inventory-service
├── 📄 inventory-service.dockerfile
├── 📁 node_modules
├── 📄 package.json
├── 📄 package-lock.json
├── 📄 tsconfig.json
└── 📁 src
    ├── 📁 config
    │   └── 📄 appconfig.ts
    ├── 📁 handlers
    ├── 📄 index.ts
    ├── 📁 middleware
    │   └── 📄 errorHandler.ts
    ├── 📁 models
    ├── 📁 queues
    │   └── 📄 inventoryConsumer.ts
    ├── 📁 routes
    ├── 📁 services
    │   └── 📄 rabbitmqService.ts
    └── 📁 utils
        ├── 📄 custom-error.ts
        └── 📄 logger.ts
```



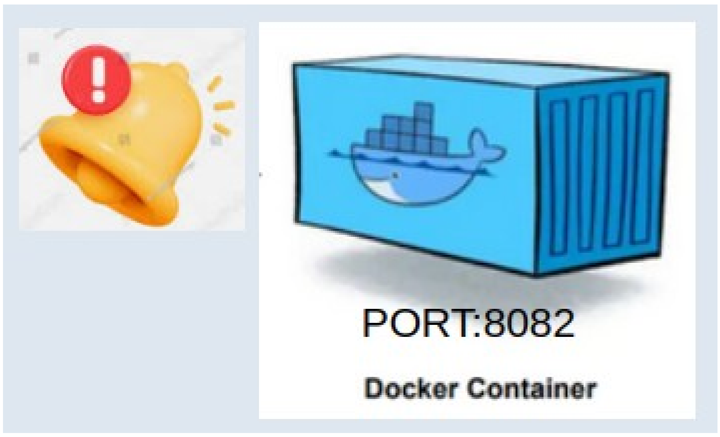
inventory-service architecture

```
+--------------------------------------------------------+
|                   RabbitMQ (order-events Exchange)     |
|                     Routing Key: 'order.created'       |
+-------------------------------+------------------------+
                                |
                                v
                   +-----------------------------+
                   |      Inventory Service      |
                   |-----------------------------|
                   |  • consumeOrderCreated()    |
                   |    - Receives 'order.created' event
                   |    - Validates & processes order
                   |    - Publishes 'inventory.status.updated' event
                   +-----------------------------+
                                |
                                v
+--------------------------------------------------------+
|         RabbitMQ (inventory-events Exchange)           |
|         Routing Key: 'inventory.status.updated'        |
+--------------------------------------------------------+
```

NOTIFICATION-SERVICE

The notification-service is designed to handle communication with users based on important system events. It listens for the inventory.status.updated event, which is published by the Inventory Service once it processes an order. When this event is received, the Notification Service interprets it—typically confirming that the inventory has been updated for a specific order—and then generates a follow-up event, notification.sent. This event indicates that a notification has been triggered, which could be used for logging, alerting users via email/SMS, or updating UI dashboards.

```
📁 notification-service
├── 📄 notification-service.dockerfile
├── 📁 node_modules
├── 📄 package.json
├── 📄 package-lock.json
├── 📄 tsconfig.json
└── 📁 src
    ├── 📁 config
    │   └── 📄 appconfig.ts
    ├── 📁 handlers
    ├── 📄 index.ts
    ├── 📁 middleware
    │   └── 📄 errorHandler.ts
    ├── 📁 models
    ├── 📁 queues
    │   └── 📄 notificationConsumer.ts
    ├── 📁 routes
    ├── 📁 services
    │   └── 📄 rabbitmqService.ts
    └── 📁 utils
        ├── 📄 custom-error.ts
        └── 📄 logger.ts
```



notification-service architecture

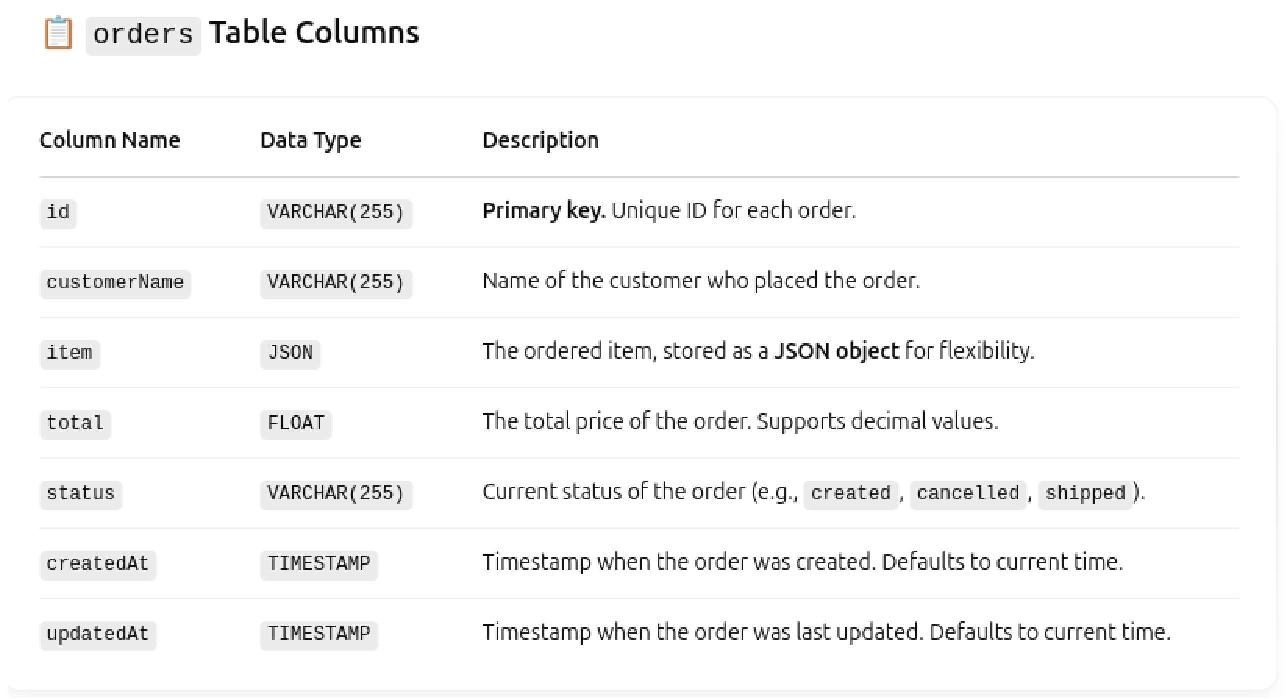
```
+---------------------------------------------------------+
|      RabbitMQ (inventory-events Exchange)              |
|        Routing Key: 'inventory.status.updated'         |
+-------------------------------+-------------------------+
                                |
                                v
                  +-----------------------------------+
                  |      Notification Service        |
                  |-----------------------------------|
                  |  • consumeInventoryStatusUpdated()|
                  |    - Listens on NOTIFICATION_QUEUE
                  |    - Consumes 'inventory.status.updated'
                  |    - Logs and prepares notification
                  |    - Publishes 'notification.sent'
                  +-----------------------------------+
                                |
                                v
+---------------------------------------------------------+
|      RabbitMQ (inventory-events Exchange)              |
|        Routing Key: 'notification.sent'                |
+---------------------------------------------------------+
```

ORDER-SERVICE DATABASE

Schema:

```
📁 build-tools
├── 📄 init-order-db.sql
```

This SQL file is used to initialize the `orders` table in database. By using this script in the docker-compose.yaml file, a database volume is created while starting the order-db service.



📋 `orders` Table Columns

Column Name	Data Type	Description
`id`	`VARCHAR(255)`	**Primary key.** Unique ID for each order.
`customerName`	`VARCHAR(255)`	Name of the customer who placed the order.
`item`	`JSON`	The ordered item, stored as a **JSON object** for flexibility.
`total`	`FLOAT`	The total price of the order. Supports decimal values.
`status`	`VARCHAR(255)`	Current status of the order (e.g., `created`, `cancelled`, `shipped`).
`createdAt`	`TIMESTAMP`	Timestamp when the order was created. Defaults to current time.
`updatedAt`	`TIMESTAMP`	Timestamp when the order was last updated. Defaults to current time.

TypeScript:

```
📁 order-service
└── 📁 src
    ├── 📁 config
    │   ├── 📄 appconfig.ts
    │   └── 📄 database.ts

    ├── 📁 models
    │   └── 📄 order.ts
```

order.ts:

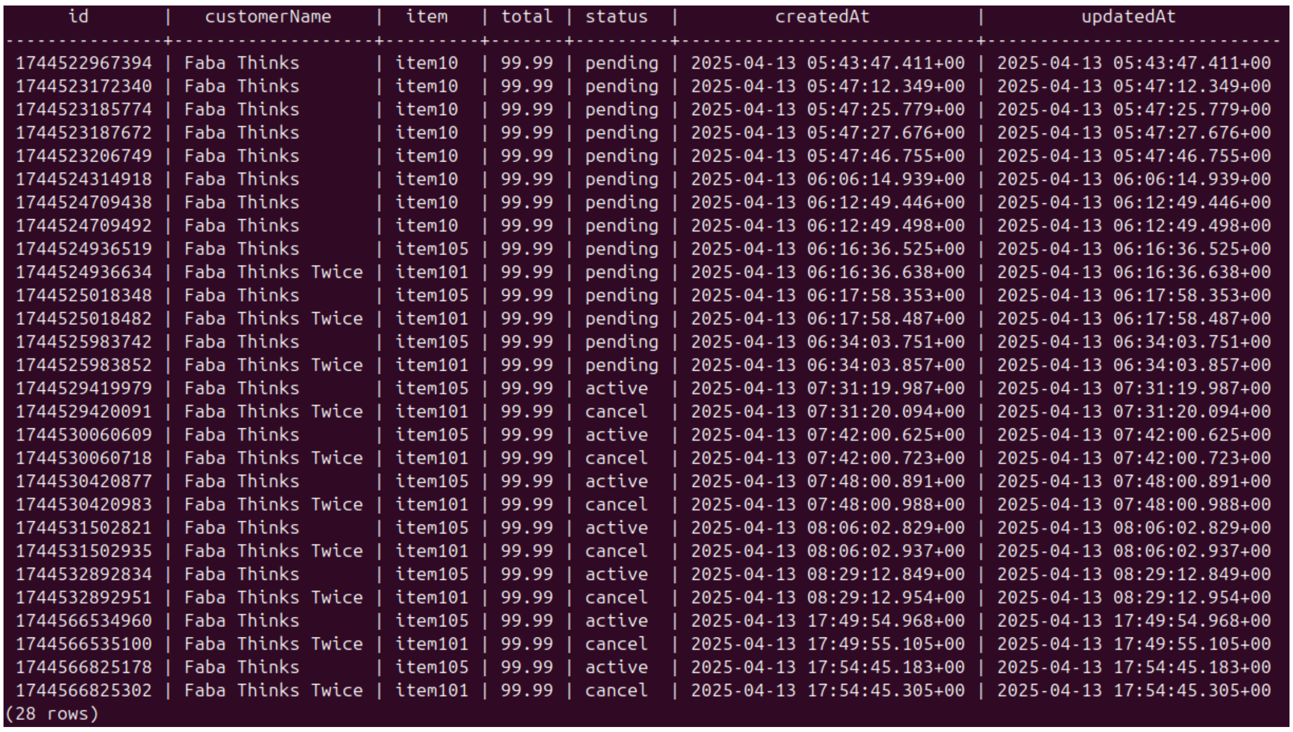
This file defines the **Order model** using **Sequelize**, which is an ORM (Object-Relational Mapping) for Node.js and TypeScript.

Fields:

- **id**: Primary key, required
- **customerName**: String, required
- **item**: Changed from an array (items) to a single string field (item)
- **total**: A float (e.g., 99.99), required
- **status**: E.g., "created", "cancelled", etc., required

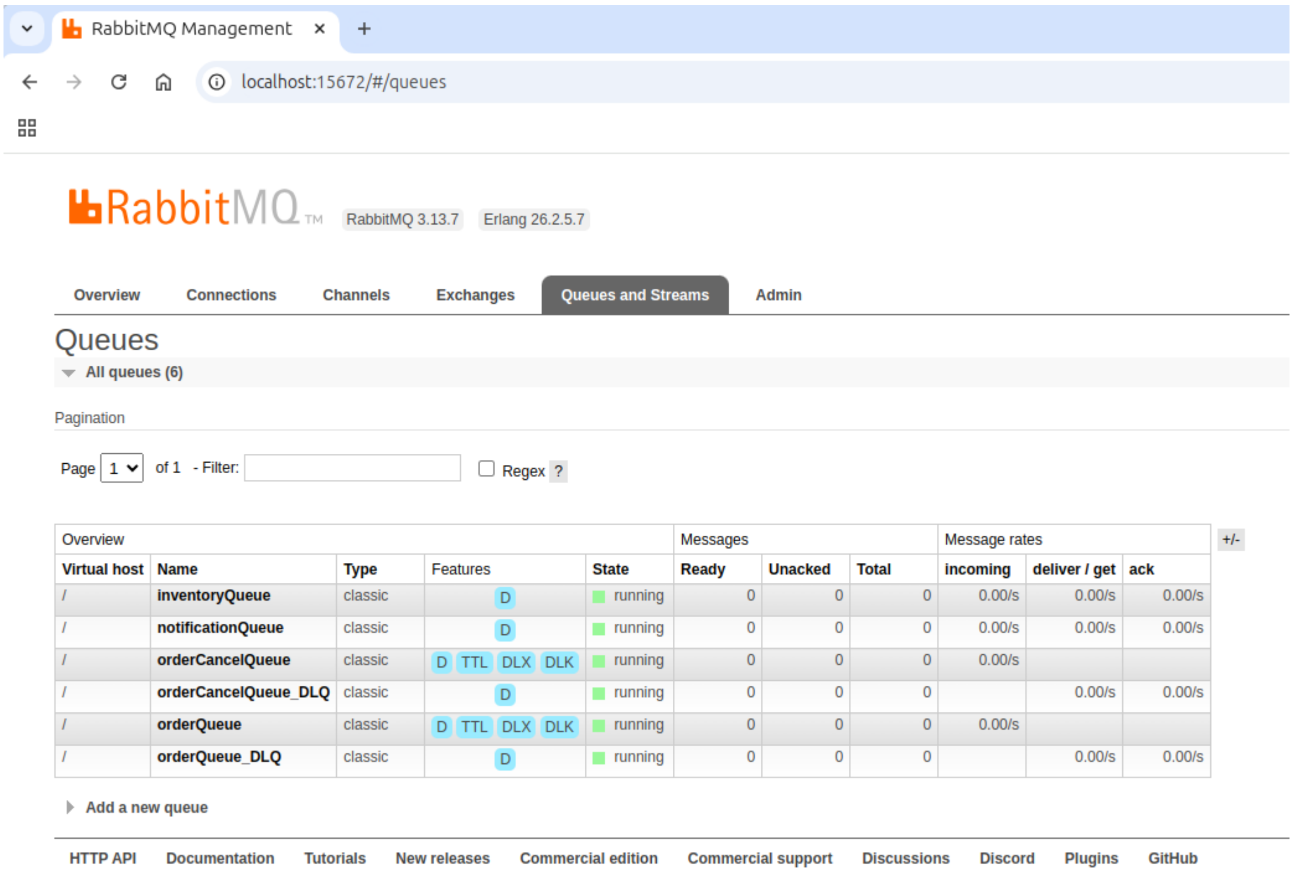
Options:

- sequelize: links to the DB connection
- modelName: internal name Sequelize uses
- tableName: actual name in DB
- timestamps: true: adds createdAt and updatedAt fields automatically



```
      id       |   customerName    |  item   | total | status  |         createdAt          |         updatedAt
---------------+-------------------+---------+-------+---------+----------------------------+----------------------------
 1744522967394 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:43:47.411+00 | 2025-04-13 05:43:47.411+00
 1744523172340 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:12.349+00 | 2025-04-13 05:47:12.349+00
 1744523185774 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:25.779+00 | 2025-04-13 05:47:25.779+00
 1744523187672 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:27.676+00 | 2025-04-13 05:47:27.676+00
 1744523206749 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:46.755+00 | 2025-04-13 05:47:46.755+00
 1744524314918 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 06:06:14.939+00 | 2025-04-13 06:06:14.939+00
 1744524709438 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 06:12:49.446+00 | 2025-04-13 06:12:49.446+00
 1744524709492 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 06:12:49.498+00 | 2025-04-13 06:12:49.498+00
 1744524936519 | Faba Thinks       | item105 | 99.99 | pending | 2025-04-13 06:16:36.525+00 | 2025-04-13 06:16:36.525+00
 1744524936634 | Faba Thinks Twice | item101 | 99.99 | pending | 2025-04-13 06:16:36.638+00 | 2025-04-13 06:16:36.638+00
 1744525018348 | Faba Thinks       | item105 | 99.99 | pending | 2025-04-13 06:17:58.353+00 | 2025-04-13 06:17:58.353+00
 1744525018482 | Faba Thinks Twice | item101 | 99.99 | pending | 2025-04-13 06:17:58.487+00 | 2025-04-13 06:17:58.487+00
 1744525983742 | Faba Thinks       | item105 | 99.99 | pending | 2025-04-13 06:34:03.751+00 | 2025-04-13 06:34:03.751+00
 1744525983852 | Faba Thinks Twice | item101 | 99.99 | pending | 2025-04-13 06:34:03.857+00 | 2025-04-13 06:34:03.857+00
 1744529419979 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 07:31:19.987+00 | 2025-04-13 07:31:19.987+00
 1744529420091 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 07:31:20.094+00 | 2025-04-13 07:31:20.094+00
 1744530060609 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 07:42:00.625+00 | 2025-04-13 07:42:00.625+00
 1744530060718 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 07:42:00.723+00 | 2025-04-13 07:42:00.723+00
 1744530420877 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 07:48:00.891+00 | 2025-04-13 07:48:00.891+00
 1744530420983 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 07:48:00.988+00 | 2025-04-13 07:48:00.988+00
 1744531502821 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 08:06:02.829+00 | 2025-04-13 08:06:02.829+00
 1744531502935 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 08:06:02.937+00 | 2025-04-13 08:06:02.937+00
 1744532892834 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 08:29:12.849+00 | 2025-04-13 08:29:12.849+00
 1744532892951 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 08:29:12.954+00 | 2025-04-13 08:29:12.954+00
 1744566534960 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 17:49:54.968+00 | 2025-04-13 17:49:54.968+00
 1744566535100 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 17:49:55.105+00 | 2025-04-13 17:49:55.105+00
 1744566825178 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 17:54:45.183+00 | 2025-04-13 17:54:45.183+00
 1744566825302 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 17:54:45.305+00 | 2025-04-13 17:54:45.305+00
(28 rows)
```

RABBITMQ UI



RabbitMQ Management

localhost:15672/#/queues

RabbitMQ 3.13.7 Erlang 26.2.5.7

Overview Connections Channels Exchanges Queues and Streams Admin

Queues

All queues (6)

Pagination

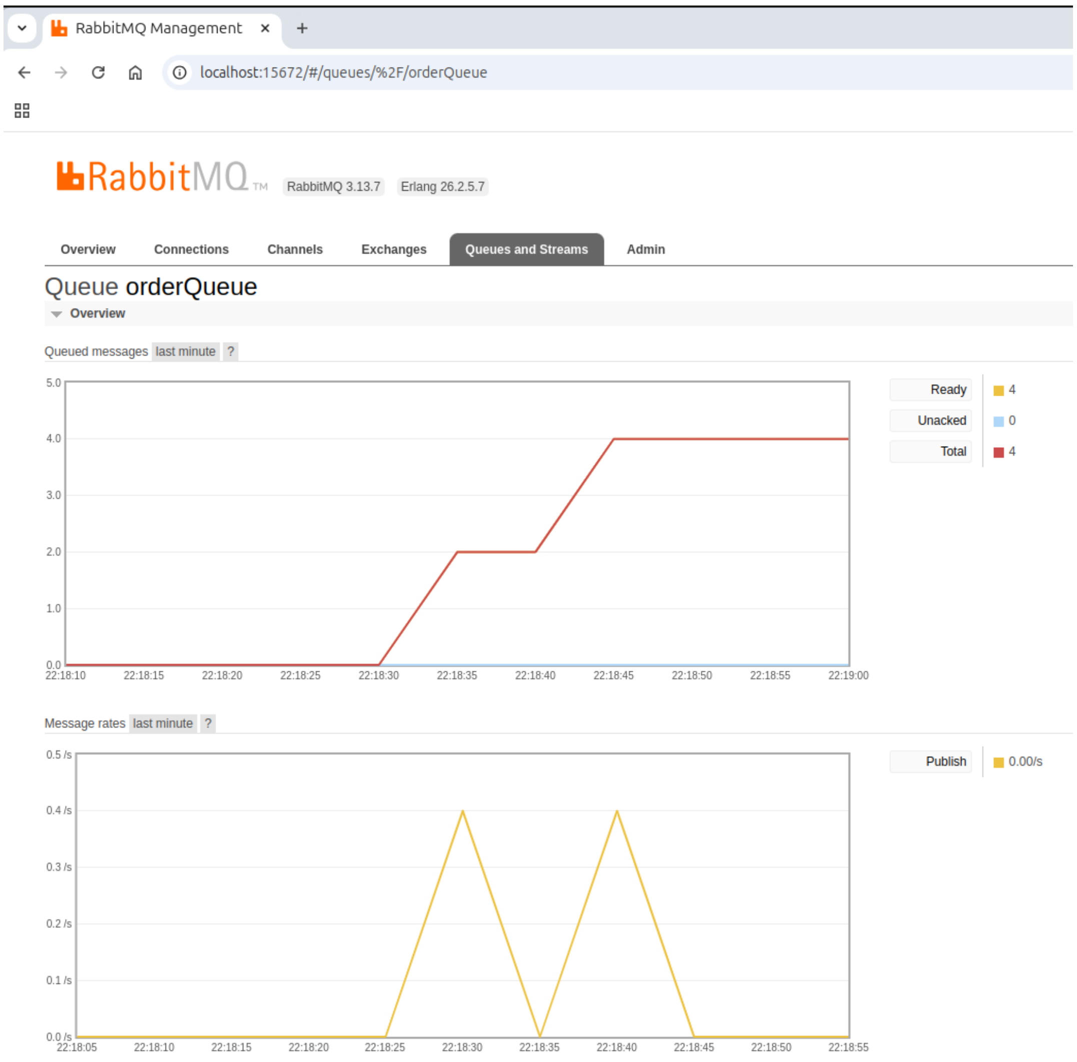
Page 1 of 1 - Filter: Regex ?

Overview					Messages			Message rates		
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	inventoryQueue	classic	D	🟩 running	0	0	0	0.00/s	0.00/s	0.00/s
/	notificationQueue	classic	D	🟩 running	0	0	0	0.00/s	0.00/s	0.00/s
/	orderCancelQueue	classic	D TTL DLX DLK	🟩 running	0	0	0	0.00/s		
/	orderCancelQueue_DLQ	classic	D	🟩 running	0	0	0		0.00/s	0.00/s
/	orderQueue	classic	D TTL DLX DLK	🟩 running	0	0	0	0.00/s		
/	orderQueue_DLQ	classic	D	🟩 running	0	0	0		0.00/s	0.00/s

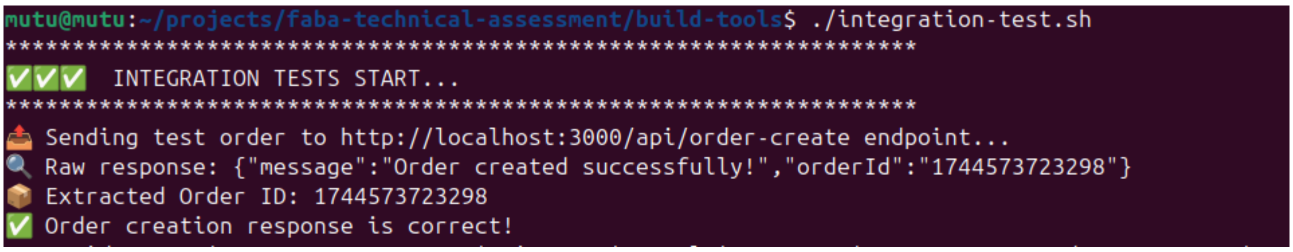
+/-

Add a new queue

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Discussions Discord Plugins GitHub

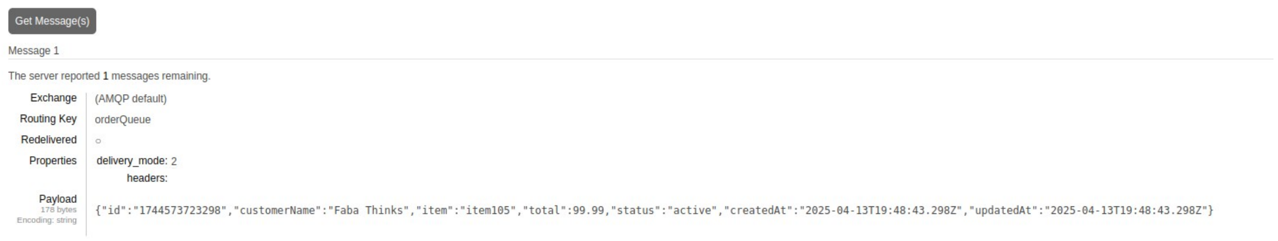


Run integration-test.sh:



```
mutu@mutu:~/projects/faba-technical-assessment/build-tools$ ./integration-test.sh
*********************************************************************
✅✅✅  INTEGRATION TESTS START...
*********************************************************************
📤 Sending test order to http://localhost:3000/api/order-create endpoint...
🔍 Raw response: {"message":"Order created successfully!","orderId":"1744573723298"}
📦 Extracted Order ID: 1744573723298
✅ Order creation response is correct!
```

Get Messege(s):



Get Message(s)

Message 1

The server reported **1** messages remaining.

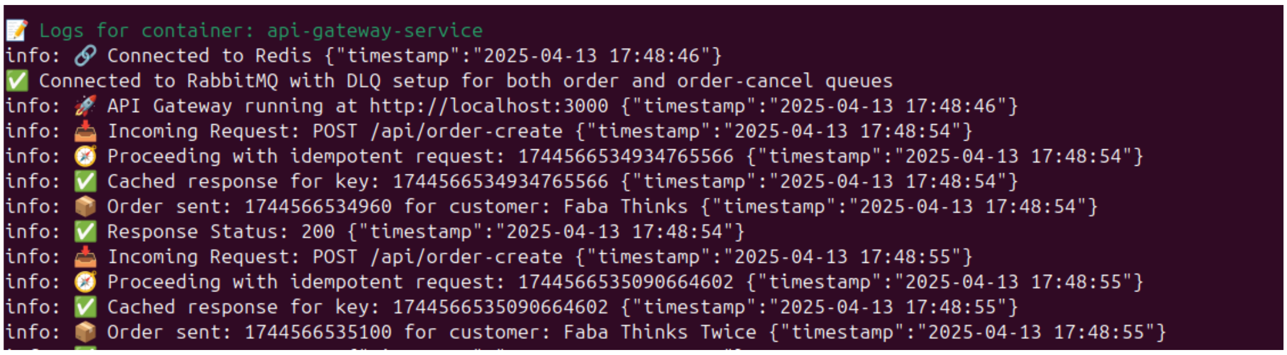
Exchange	(AMQP default)
Routing Key	orderQueue
Redelivered	○
Properties	delivery_mode: 2 headers:
Payload 178 bytes Encoding: string	`{"id":"1744573723298","customerName":"Faba Thinks","item":"item105","total":99.99,"status":"active","createdAt":"2025-04-13T19:48:43.298Z","updatedAt":"2025-04-13T19:48:43.298Z"}`

LOGGING

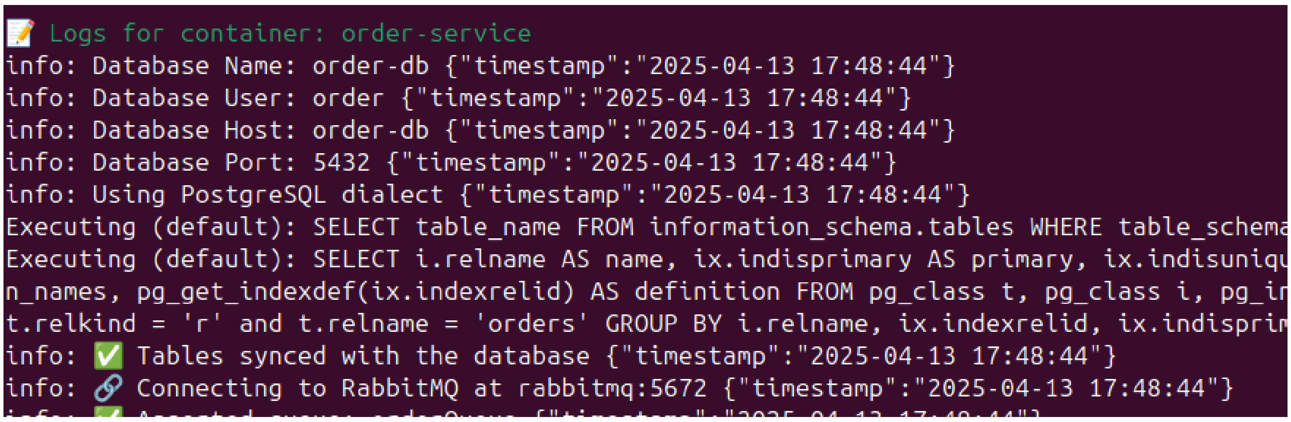
logger:

```
│   └── 📁 utils
│       └── 📄 logger.ts
```

The logger.ts file sets up a centralized logging system using the Winston library, which is a popular and flexible logging tool for Node.js applications. It begins by importing Winston, then creates a logger instance using winston.createLogger. This logger is configured to capture all log messages at the info level or higher (such as warn and error). For formatting, it combines a timestamp (formatted as YYYY-MM-DD HH:mm:ss) with a custom printf function that structures the log output to include the timestamp, log level, and the actual message.



```
📝 Logs for container: api-gateway-service
info: 🔗 Connected to Redis {"timestamp":"2025-04-13 17:48:46"}
✅ Connected to RabbitMQ with DLQ setup for both order and order-cancel queues
info: 🚀 API Gateway running at http://localhost:3000 {"timestamp":"2025-04-13 17:48:46"}
info: 📥 Incoming Request: POST /api/order-create {"timestamp":"2025-04-13 17:48:54"}
info: 🧭 Proceeding with idempotent request: 1744566534934765566 {"timestamp":"2025-04-13 17:48:54"}
info: ✅ Cached response for key: 1744566534934765566 {"timestamp":"2025-04-13 17:48:54"}
info: 📦 Order sent: 1744566534960 for customer: Faba Thinks {"timestamp":"2025-04-13 17:48:54"}
info: ✅ Response Status: 200 {"timestamp":"2025-04-13 17:48:54"}
info: 📥 Incoming Request: POST /api/order-create {"timestamp":"2025-04-13 17:48:55"}
info: 🧭 Proceeding with idempotent request: 1744566535090664602 {"timestamp":"2025-04-13 17:48:55"}
info: ✅ Cached response for key: 1744566535090664602 {"timestamp":"2025-04-13 17:48:55"}
info: 📦 Order sent: 1744566535100 for customer: Faba Thinks Twice {"timestamp":"2025-04-13 17:48:55"}
```

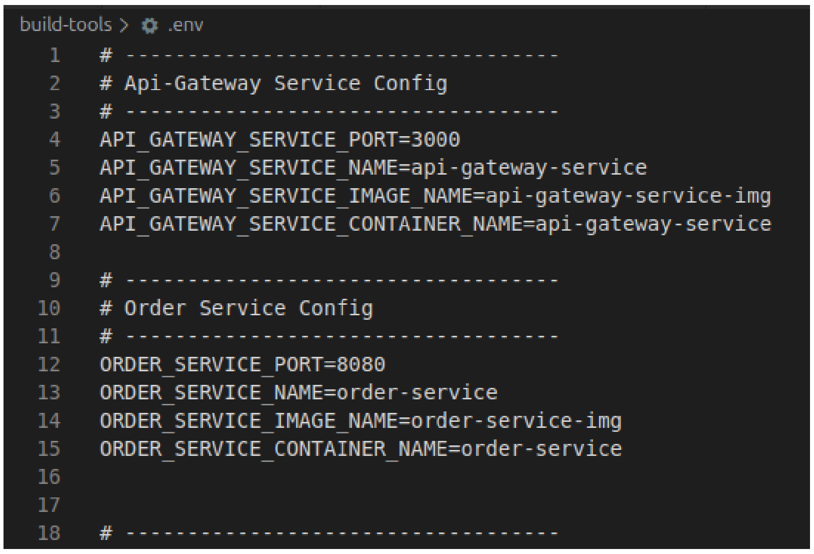


```
📝 Logs for container: order-service
info: Database Name: order-db {"timestamp":"2025-04-13 17:48:44"}
info: Database User: order {"timestamp":"2025-04-13 17:48:44"}
info: Database Host: order-db {"timestamp":"2025-04-13 17:48:44"}
info: Database Port: 5432 {"timestamp":"2025-04-13 17:48:44"}
info: Using PostgreSQL dialect {"timestamp":"2025-04-13 17:48:44"}
Executing (default): SELECT table_name FROM information_schema.tables WHERE table_schema
Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisuniqu
n_names, pg_get_indexdef(ix.indexrelid) AS definition FROM pg_class t, pg_class i, pg_in
t.relkind = 'r' and t.relname = 'orders' GROUP BY i.relname, ix.indexrelid, ix.indisprim
info: ✅ Tables synced with the database {"timestamp":"2025-04-13 17:48:44"}
info: 🔗 Connecting to RabbitMQ at rabbitmq:5672 {"timestamp":"2025-04-13 17:48:44"}
```

ENVIRONMENT CONFIGURATION

All service information is set permanently in this file.

```
📁 build-tools
├── 📄 .env
```


build-tools > .env

```
# ----------------------------------
# Api-Gateway Service Config
# ----------------------------------
API_GATEWAY_SERVICE_PORT=3000
API_GATEWAY_SERVICE_NAME=api-gateway-service
API_GATEWAY_SERVICE_IMAGE_NAME=api-gateway-service-img
API_GATEWAY_SERVICE_CONTAINER_NAME=api-gateway-service

# ----------------------------------
# Order Service Config
# ----------------------------------
ORDER_SERVICE_PORT=8080
ORDER_SERVICE_NAME=order-service
ORDER_SERVICE_IMAGE_NAME=order-service-img
ORDER_SERVICE_CONTAINER_NAME=order-service


# ----------------------------------
```

GIT HUB REPOSITORY

Follow the steps below to clone and run the project locally.

Clone the Repository:

You can use **HTTPS** or **SSH** to clone:

🔷 **Using HTTPS:**

git clone https://github.com/MuratTunc/faba-technical-assessment.git

🔷 **Using SSH:**

git clone git@github.com:MuratTunc/faba-technical-assessment.git

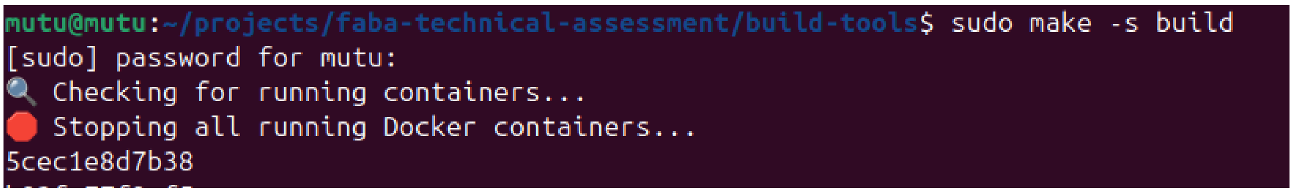
Install Dependencies:

Navigate to the project directory and install dependencies:

cd faba-technical-assessment

cd build-tools

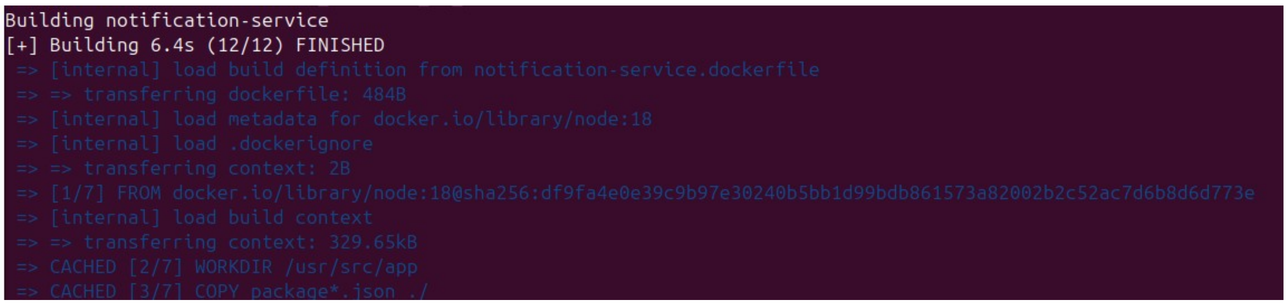
sudo make -s build



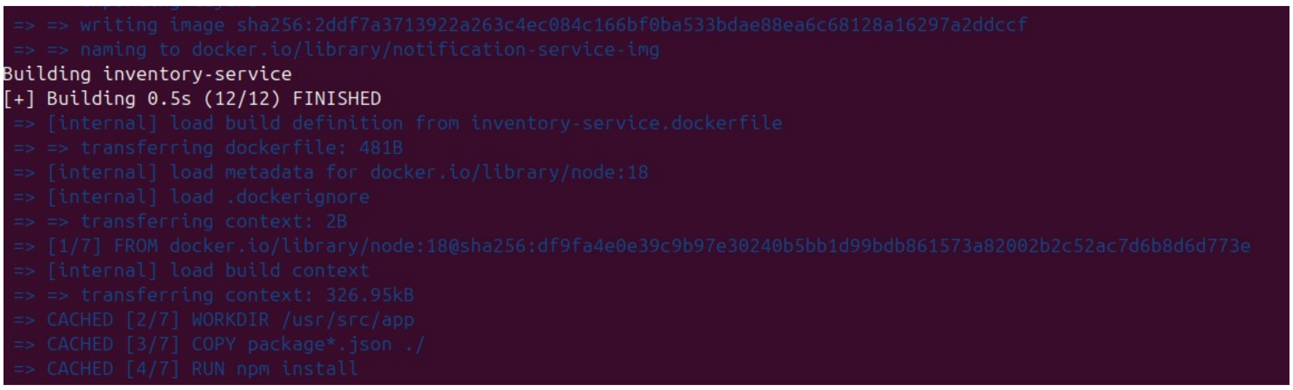
```
mutu@mutu:~/projects/faba-technical-assessment/build-tools$ sudo make -s build
[sudo] password for mutu:
🔍 Checking for running containers...
🛑 Stopping all running Docker containers...
5cec1e8d7b38
```

All services started to be compiled in order.

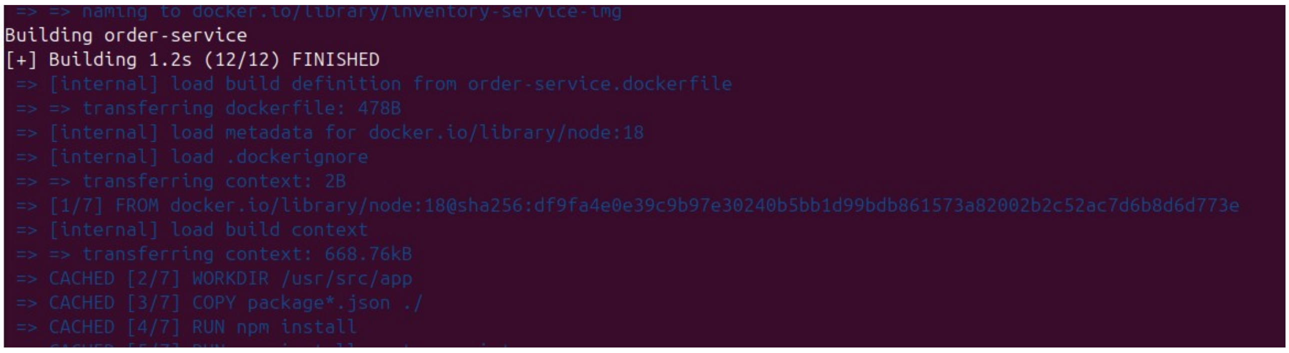
Like these:



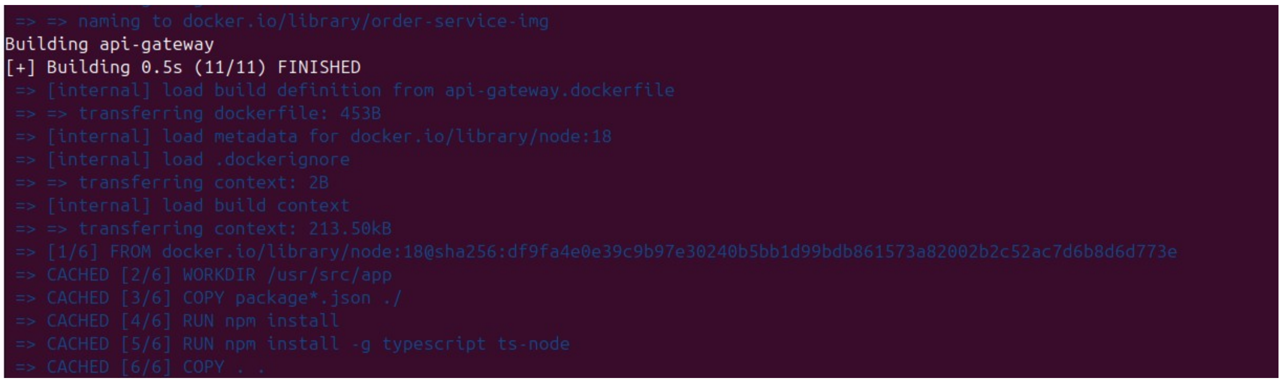
```
Building notification-service
[+] Building 6.4s (12/12) FINISHED
 => [internal] load build definition from notification-service.dockerfile
 => => transferring dockerfile: 484B
 => [internal] load metadata for docker.io/library/node:18
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/7] FROM docker.io/library/node:18@sha256:df9fa4e0e39c9b97e30240b5bb1d99bdb861573a82002b2c52ac7d6b8d6d773e
 => [internal] load build context
 => => transferring context: 329.65kB
 => CACHED [2/7] WORKDIR /usr/src/app
 => CACHED [3/7] COPY package*.json ./
```



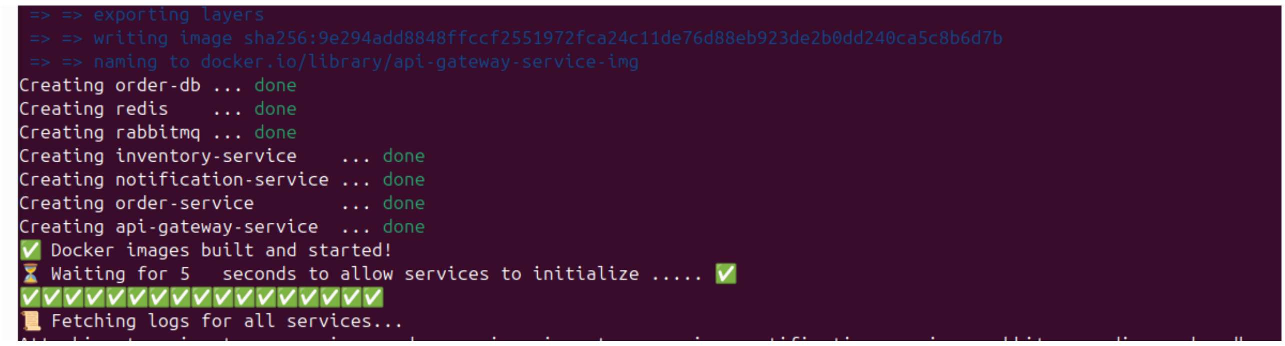
```
 => => writing image sha256:2ddf7a3713922a263c4ec084c166bf0ba533bdae88ea6c68128a16297a2ddccf
 => => naming to docker.io/library/notification-service-img
Building inventory-service
[+] Building 0.5s (12/12) FINISHED
 => [internal] load build definition from inventory-service.dockerfile
 => => transferring dockerfile: 481B
 => [internal] load metadata for docker.io/library/node:18
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/7] FROM docker.io/library/node:18@sha256:df9fa4e0e39c9b97e30240b5bb1d99bdb861573a82002b2c52ac7d6b8d6d773e
 => [internal] load build context
 => => transferring context: 326.95kB
 => CACHED [2/7] WORKDIR /usr/src/app
 => CACHED [3/7] COPY package*.json ./
 => CACHED [4/7] RUN npm install
```



```
 => => naming to docker.io/library/inventory-service-img
Building order-service
[+] Building 1.2s (12/12) FINISHED
 => [internal] load build definition from order-service.dockerfile
 => => transferring dockerfile: 478B
 => [internal] load metadata for docker.io/library/node:18
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/7] FROM docker.io/library/node:18@sha256:df9fa4e0e39c9b97e30240b5bb1d99bdb861573a82002b2c52ac7d6b8d6d773e
 => [internal] load build context
 => => transferring context: 668.76kB
 => CACHED [2/7] WORKDIR /usr/src/app
 => CACHED [3/7] COPY package*.json ./
 => CACHED [4/7] RUN npm install
```

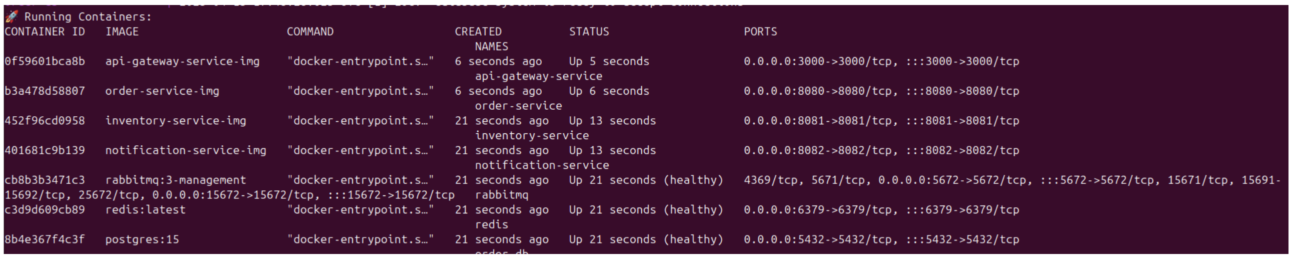


```
 => => naming to docker.io/library/order-service-img
Building api-gateway
[+] Building 0.5s (11/11) FINISHED
 => [internal] load build definition from api-gateway.dockerfile
 => => transferring dockerfile: 453B
 => [internal] load metadata for docker.io/library/node:18
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 213.50kB
 => [1/6] FROM docker.io/library/node:18@sha256:df9fa4e0e39c9b97e30240b5bb1d99bdb861573a82002b2c52ac7d6b8d6d773e
 => CACHED [2/6] WORKDIR /usr/src/app
 => CACHED [3/6] COPY package*.json ./
 => CACHED [4/6] RUN npm install
 => CACHED [5/6] RUN npm install -g typescript ts-node
 => CACHED [6/6] COPY . .
```



```
 => => exporting layers
 => => writing image sha256:9e294add8848ffccf2551972fca24c11de76d88eb923de2b0dd240ca5c8b6d7b
 => => naming to docker.io/library/api-gateway-service-img
Creating order-db ... done
Creating redis    ... done
Creating rabbitmq ... done
Creating inventory-service    ... done
Creating notification-service ... done
Creating order-service        ... done
Creating api-gateway-service  ... done
✅ Docker images built and started!
⏳ Waiting for 5   seconds to allow services to initialize ..... ✅
✅✅✅✅✅✅✅✅✅✅✅✅✅✅✅✅✅
📜 Fetching logs for all services...
```

All services are ready.

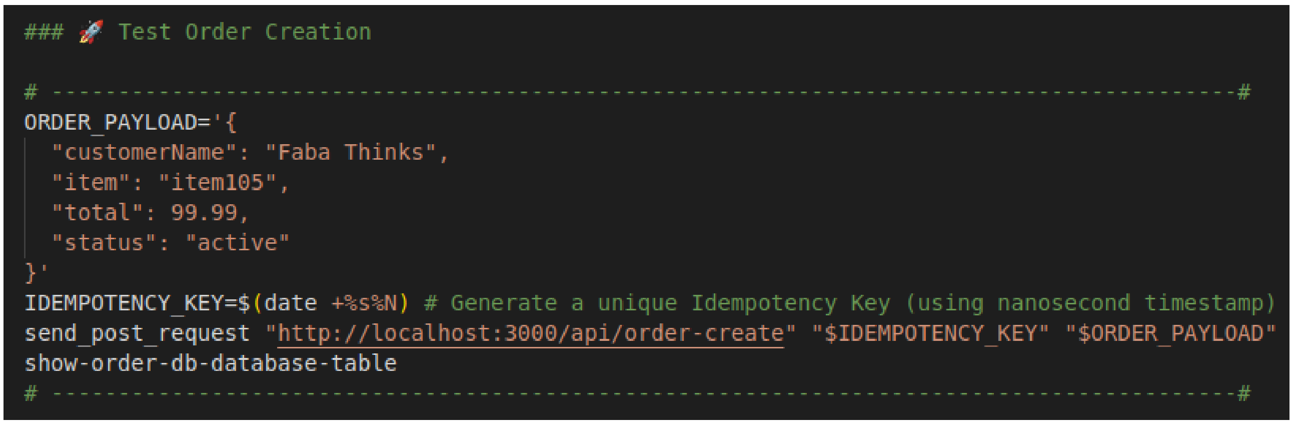


```
🚀 Running Containers:
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS                     PORTS
                                                                     NAMES
0f59601bca8b   api-gateway-service-img    "docker-entrypoint.s…"   6 seconds ago    Up 5 seconds               0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
                                                                     api-gateway-service
b3a478d58807   order-service-img          "docker-entrypoint.s…"   6 seconds ago    Up 6 seconds               0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
                                                                     order-service
452f96cd0958   inventory-service-img      "docker-entrypoint.s…"   21 seconds ago   Up 13 seconds              0.0.0.0:8081->8081/tcp, :::8081->8081/tcp
                                                                     inventory-service
401681c9b139   notification-service-img   "docker-entrypoint.s…"   21 seconds ago   Up 13 seconds              0.0.0.0:8082->8082/tcp, :::8082->8082/tcp
                                                                     notification-service
cb8b3b3471c3   rabbitmq:3-management      "docker-entrypoint.s…"   21 seconds ago   Up 21 seconds (healthy)   4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 15671/tcp, 15691-
15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp   rabbitmq
c3d9d609cb89   redis:latest               "docker-entrypoint.s…"   21 seconds ago   Up 21 seconds (healthy)   0.0.0.0:6379->6379/tcp, :::6379->6379/tcp
                                                                     redis
8b4e367f4c3f   postgres:15                "docker-entrypoint.s…"   21 seconds ago   Up 21 seconds (healthy)   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp
                                                                     order-db
```

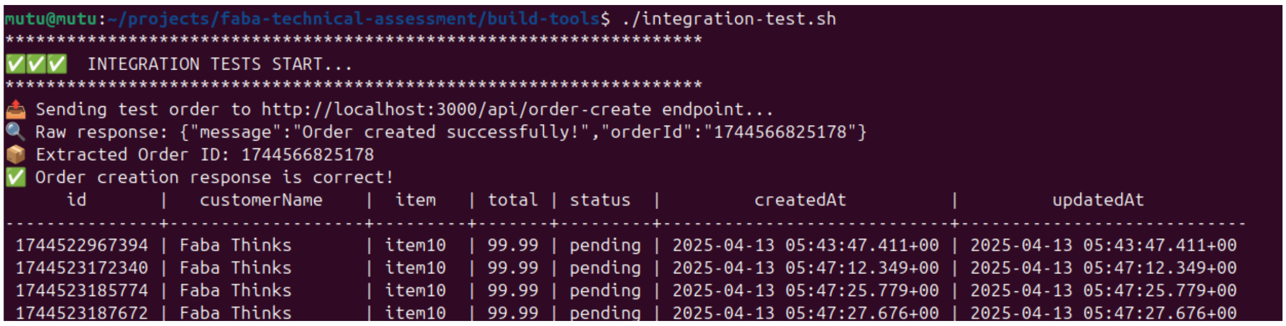
INTEGRATION TESTS

```
📁 build-tools
├── 📄 integration-test.sh
```

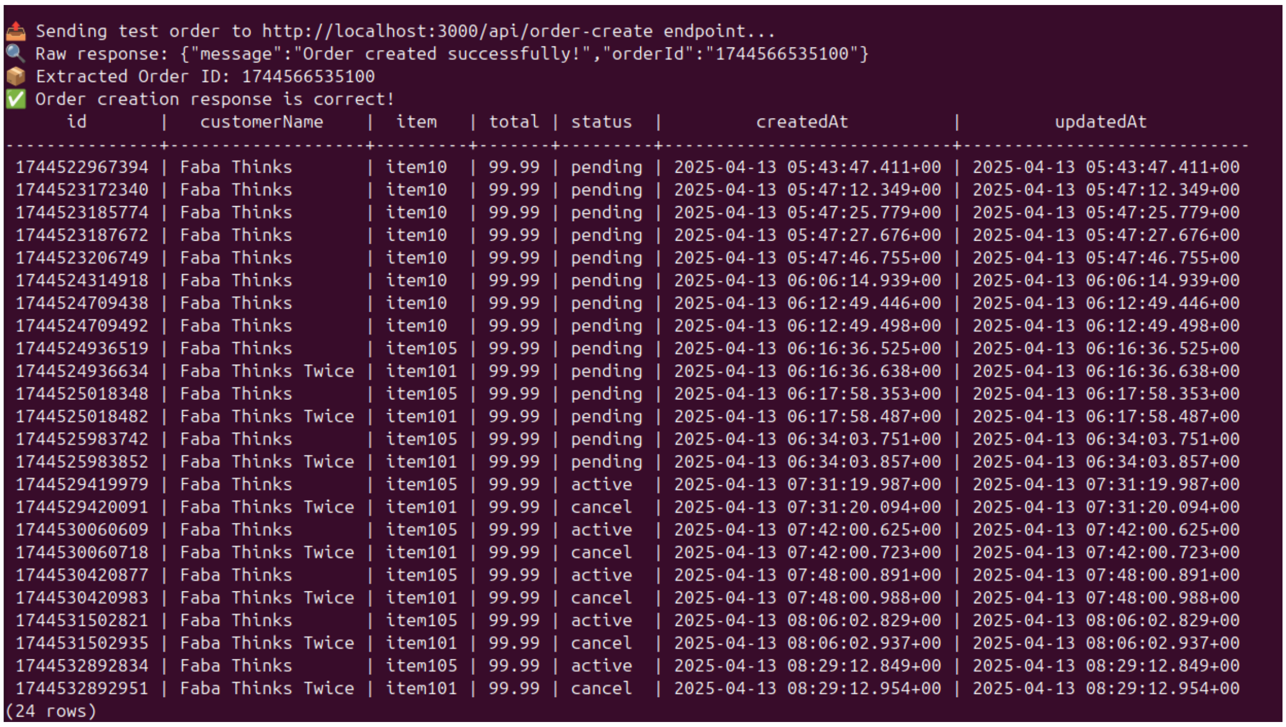
Sending Post Request to api-gateway service:



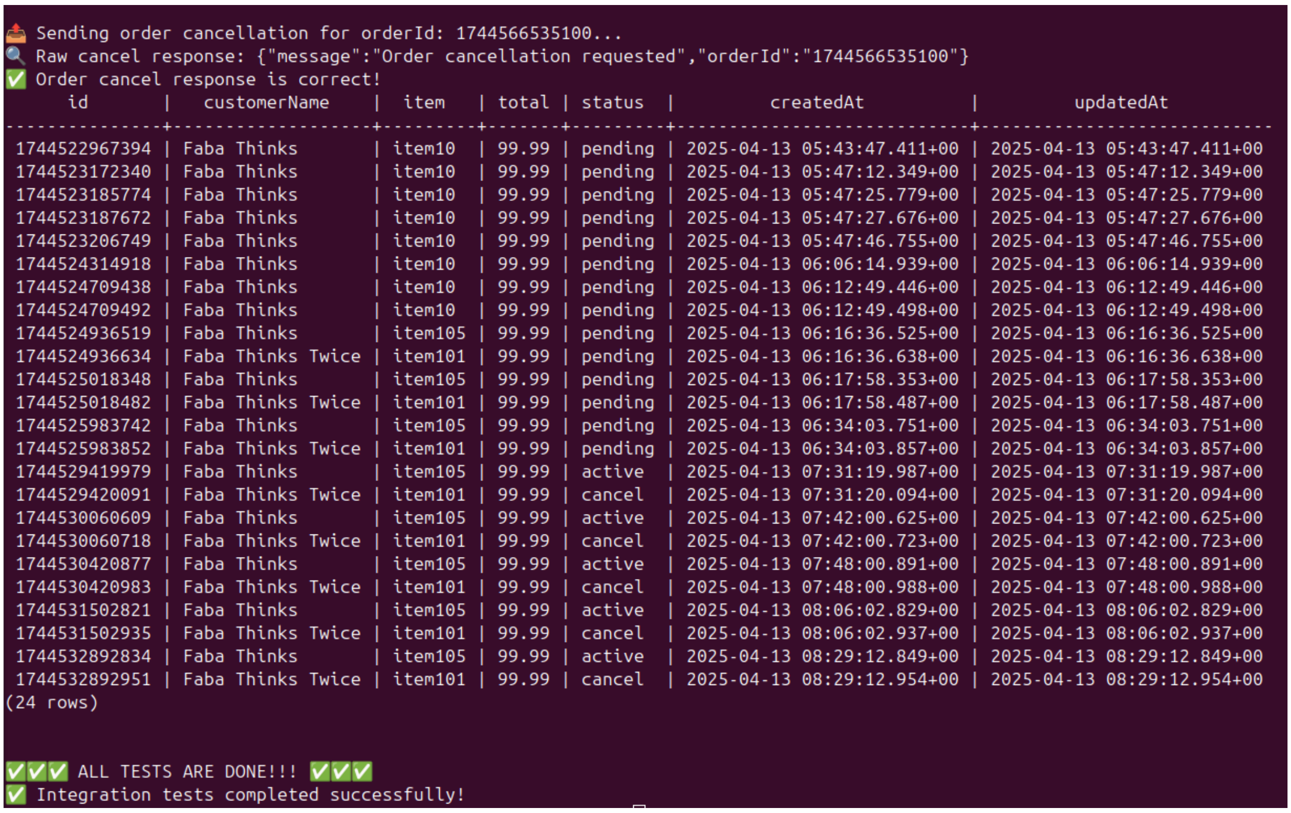
```
### 🚀 Test Order Creation

# ---------------------------------------------------------------------------------------#
ORDER_PAYLOAD='{
  "customerName": "Faba Thinks",
  "item": "item105",
  "total": 99.99,
  "status": "active"
}'
IDEMPOTENCY_KEY=$(date +%s%N) # Generate a unique Idempotency Key (using nanosecond timestamp)
send_post_request "http://localhost:3000/api/order-create" "$IDEMPOTENCY_KEY" "$ORDER_PAYLOAD"
show-order-db-database-table
# ---------------------------------------------------------------------------------------#
```



```
mutu@mutu:~/projects/faba-technical-assessment/build-tools$ ./integration-test.sh
***********************************************************************
✅✅✅  INTEGRATION TESTS START...
***********************************************************************
📤 Sending test order to http://localhost:3000/api/order-create endpoint...
🔍 Raw response: {"message":"Order created successfully!","orderId":"1744566825178"}
📦 Extracted Order ID: 1744566825178
✅ Order creation response is correct!
      id       |    customerName     |  item   | total | status  |         createdAt          |         updatedAt
---------------+--------------------+---------+-------+---------+----------------------------+----------------------------
 1744522967394 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:43:47.411+00 | 2025-04-13 05:43:47.411+00
 1744523172340 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:47:12.349+00 | 2025-04-13 05:47:12.349+00
 1744523185774 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:47:25.779+00 | 2025-04-13 05:47:25.779+00
 1744523187672 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:47:27.676+00 | 2025-04-13 05:47:27.676+00
```



```
📤 Sending test order to http://localhost:3000/api/order-create endpoint...
🔍 Raw response: {"message":"Order created successfully!","orderId":"1744566535100"}
📦 Extracted Order ID: 1744566535100
✅ Order creation response is correct!
      id       |    customerName     |  item   | total | status  |         createdAt          |         updatedAt
---------------+--------------------+---------+-------+---------+----------------------------+----------------------------
 1744522967394 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:43:47.411+00 | 2025-04-13 05:43:47.411+00
 1744523172340 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:47:12.349+00 | 2025-04-13 05:47:12.349+00
 1744523185774 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:47:25.779+00 | 2025-04-13 05:47:25.779+00
 1744523187672 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:47:27.676+00 | 2025-04-13 05:47:27.676+00
 1744523206749 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 05:47:46.755+00 | 2025-04-13 05:47:46.755+00
 1744524314918 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 06:06:14.939+00 | 2025-04-13 06:06:14.939+00
 1744524709438 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 06:12:49.446+00 | 2025-04-13 06:12:49.446+00
 1744524709492 | Faba Thinks        | item10  | 99.99 | pending | 2025-04-13 06:12:49.498+00 | 2025-04-13 06:12:49.498+00
 1744524936519 | Faba Thinks        | item105 | 99.99 | pending | 2025-04-13 06:16:36.525+00 | 2025-04-13 06:16:36.525+00
 1744524936634 | Faba Thinks Twice  | item101 | 99.99 | pending | 2025-04-13 06:16:36.638+00 | 2025-04-13 06:16:36.638+00
 1744525018348 | Faba Thinks        | item105 | 99.99 | pending | 2025-04-13 06:17:58.353+00 | 2025-04-13 06:17:58.353+00
 1744525018482 | Faba Thinks Twice  | item101 | 99.99 | pending | 2025-04-13 06:17:58.487+00 | 2025-04-13 06:17:58.487+00
 1744525983742 | Faba Thinks        | item105 | 99.99 | pending | 2025-04-13 06:34:03.751+00 | 2025-04-13 06:34:03.751+00
 1744525983852 | Faba Thinks Twice  | item101 | 99.99 | pending | 2025-04-13 06:34:03.857+00 | 2025-04-13 06:34:03.857+00
 1744529419979 | Faba Thinks        | item105 | 99.99 | active  | 2025-04-13 07:31:19.987+00 | 2025-04-13 07:31:19.987+00
 1744529420091 | Faba Thinks Twice  | item101 | 99.99 | cancel  | 2025-04-13 07:31:20.094+00 | 2025-04-13 07:31:20.094+00
 1744530060609 | Faba Thinks        | item105 | 99.99 | active  | 2025-04-13 07:42:00.625+00 | 2025-04-13 07:42:00.625+00
 1744530060718 | Faba Thinks Twice  | item101 | 99.99 | cancel  | 2025-04-13 07:42:00.723+00 | 2025-04-13 07:42:00.723+00
 1744530420877 | Faba Thinks        | item105 | 99.99 | active  | 2025-04-13 07:48:00.891+00 | 2025-04-13 07:48:00.891+00
 1744530420983 | Faba Thinks Twice  | item101 | 99.99 | cancel  | 2025-04-13 07:48:00.988+00 | 2025-04-13 07:48:00.988+00
 1744531502821 | Faba Thinks        | item105 | 99.99 | active  | 2025-04-13 08:06:02.829+00 | 2025-04-13 08:06:02.829+00
 1744531502935 | Faba Thinks Twice  | item101 | 99.99 | cancel  | 2025-04-13 08:06:02.937+00 | 2025-04-13 08:06:02.937+00
 1744532892834 | Faba Thinks        | item105 | 99.99 | active  | 2025-04-13 08:29:12.849+00 | 2025-04-13 08:29:12.849+00
 1744532892951 | Faba Thinks Twice  | item101 | 99.99 | cancel  | 2025-04-13 08:29:12.954+00 | 2025-04-13 08:29:12.954+00
(24 rows)
```



```
📤 Sending order cancellation for orderId: 1744566535100...
🔍 Raw cancel response: {"message":"Order cancellation requested","orderId":"1744566535100"}
✅ Order cancel response is correct!
      id       |   customerName    |  item   | total | status  |         createdAt          |         updatedAt
---------------+-------------------+---------+-------+---------+----------------------------+----------------------------
 1744522967394 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:43:47.411+00 | 2025-04-13 05:43:47.411+00
 1744523172340 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:12.349+00 | 2025-04-13 05:47:12.349+00
 1744523185774 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:25.779+00 | 2025-04-13 05:47:25.779+00
 1744523187672 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:27.676+00 | 2025-04-13 05:47:27.676+00
 1744523206749 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 05:47:46.755+00 | 2025-04-13 05:47:46.755+00
 1744524314918 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 06:06:14.939+00 | 2025-04-13 06:06:14.939+00
 1744524709438 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 06:12:49.446+00 | 2025-04-13 06:12:49.446+00
 1744524709492 | Faba Thinks       | item10  | 99.99 | pending | 2025-04-13 06:12:49.498+00 | 2025-04-13 06:12:49.498+00
 1744524936519 | Faba Thinks       | item105 | 99.99 | pending | 2025-04-13 06:16:36.525+00 | 2025-04-13 06:16:36.525+00
 1744524936634 | Faba Thinks Twice | item101 | 99.99 | pending | 2025-04-13 06:16:36.638+00 | 2025-04-13 06:16:36.638+00
 1744525018348 | Faba Thinks       | item105 | 99.99 | pending | 2025-04-13 06:17:58.353+00 | 2025-04-13 06:17:58.353+00
 1744525018482 | Faba Thinks Twice | item101 | 99.99 | pending | 2025-04-13 06:17:58.487+00 | 2025-04-13 06:17:58.487+00
 1744525983742 | Faba Thinks       | item105 | 99.99 | pending | 2025-04-13 06:34:03.751+00 | 2025-04-13 06:34:03.751+00
 1744525983852 | Faba Thinks Twice | item101 | 99.99 | pending | 2025-04-13 06:34:03.857+00 | 2025-04-13 06:34:03.857+00
 1744529419979 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 07:31:19.987+00 | 2025-04-13 07:31:19.987+00
 1744529420091 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 07:31:20.094+00 | 2025-04-13 07:31:20.094+00
 1744530060609 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 07:42:00.625+00 | 2025-04-13 07:42:00.625+00
 1744530060718 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 07:42:00.723+00 | 2025-04-13 07:42:00.723+00
 1744530420877 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 07:48:00.891+00 | 2025-04-13 07:48:00.891+00
 1744530420983 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 07:48:00.988+00 | 2025-04-13 07:48:00.988+00
 1744531502821 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 08:06:02.829+00 | 2025-04-13 08:06:02.829+00
 1744531502935 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 08:06:02.937+00 | 2025-04-13 08:06:02.937+00
 1744532892834 | Faba Thinks       | item105 | 99.99 | active  | 2025-04-13 08:29:12.849+00 | 2025-04-13 08:29:12.849+00
 1744532892951 | Faba Thinks Twice | item101 | 99.99 | cancel  | 2025-04-13 08:29:12.954+00 | 2025-04-13 08:29:12.954+00
(24 rows)


✅✅✅ ALL TESTS ARE DONE!!! ✅✅✅
✅ Integration tests completed successfully!
```