

**Container Automation & Orchestration**

**Semester – VI**

**Under the Guidance of : Srinivasan**

**Ramachandran**

**Submitted by : Anushri Sharma**

**SAP ID : 500108300**

**Enrollment No: R2142220956**

**Table of Contents**

|  |
| --- |
| 1. Project Introduction |
| 1. Technologies Used |
| 1. System Design |
| 1. Functional Flow |
| 1. API Specifications |
| 1. Database Design |
| 1. RabbitMQ In |
| 1. Frontend Features |
| 1. Access URLs |
| 10.Design Decisions Report |

## 1. Project Summary

This project builds a simple and scalable order processing system. It connects a React frontend with a Node.js backend, using PostgreSQL to store data, Redis to cache frequently needed data, and RabbitMQ to handle background tasks like processing orders.

## 2. Technologies Used

**Frontend:** React

**Backend:** Node.js, Express

**Database:** PostgreSQL

**Caching:** Redis

**Message Broker:** RabbitMQ

**Containerization:** Docker

**3. System Design Overview**

The architecture promotes separation of concerns and easy scaling:

* Frontend Layer: Captures user actions and communicates with the backend.
* Backend Layer: Processes business logic, database operations, caching, and message queuing.
* Database Layer: Manages persistent data such as products and order records.
* Caching Layer: Boosts read performance for frequently queried data.
* Messaging Layer: Asynchronously handles order processing tasks.

## 4. How the System Works

**Order Placement:**

* A user picks a product and places an order through the frontend.
* The backend saves it in PostgreSQL with a 'pending' status.
* The order is sent to RabbitMQ for processing.
* The order is also cached in Redis using the product ID.

**Order Processing:**

* A worker reads the order from RabbitMQ.
* It processes the order and updates the status to 'completed' in PostgreSQL.
* Redis cache is also updated.

**Order Retrieval:**

* When a user looks for orders by product ID, the backend checks Redis first.
* If data is found, it’s sent back. Otherwise, the database is checked, and Redis is updated.

## 5. API Endpoints

* **POST /orders:** Create a new order.
* **GET /orders/:product\_id:** Get all orders for a product ID.
* **GET /products:** List all products.

## 6. Database Design

**Products Table:**

* id (Primary Key)
* name (Product Name)
* price (Product Price)

**Orders Table:**

* id (Primary Key)
* product\_id (Link to Products table)
* status (Order status - 'pending' or 'completed')

## 7. RabbitMQ Usage

* **Queue Name:** order\_queue
* **Producer:** Backend sends new orders to this queue.
* **Consumer:** Worker processes the orders and updates their status.

## 8. Frontend Features

* Show products.
* Place orders.
* Search orders by product ID.

**9. Start the Application with Docker Compose:**

docker-compose up --build

**Access URLs:**

* Frontend: <http://localhost:3000>
* Backend API: <http://localhost:5000>
* RabbitMQ Dashboard: <http://localhost:15672>

## 10. Reason Behind Design Choices

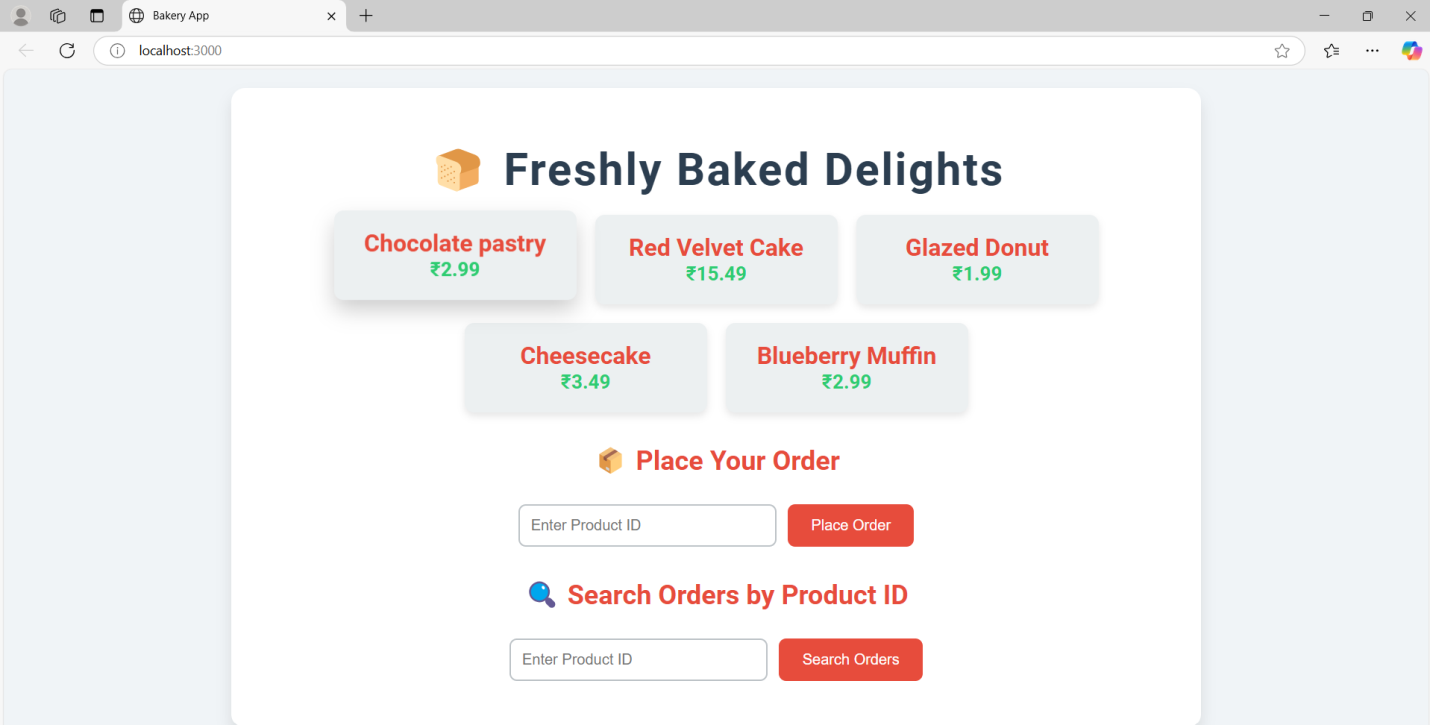
**10.1. Technology Selection:**

* **React:** Easy to build fast and responsive web pages.
* **Node.js + Express:** Perfect for creating APIs quickly.
* **PostgreSQL:** Reliable database that handles complex queries easily.
* **Redis:** Fast temporary storage to make reads quicker.
* **RabbitMQ:** Helps to handle background tasks separately, keeping the app fast.

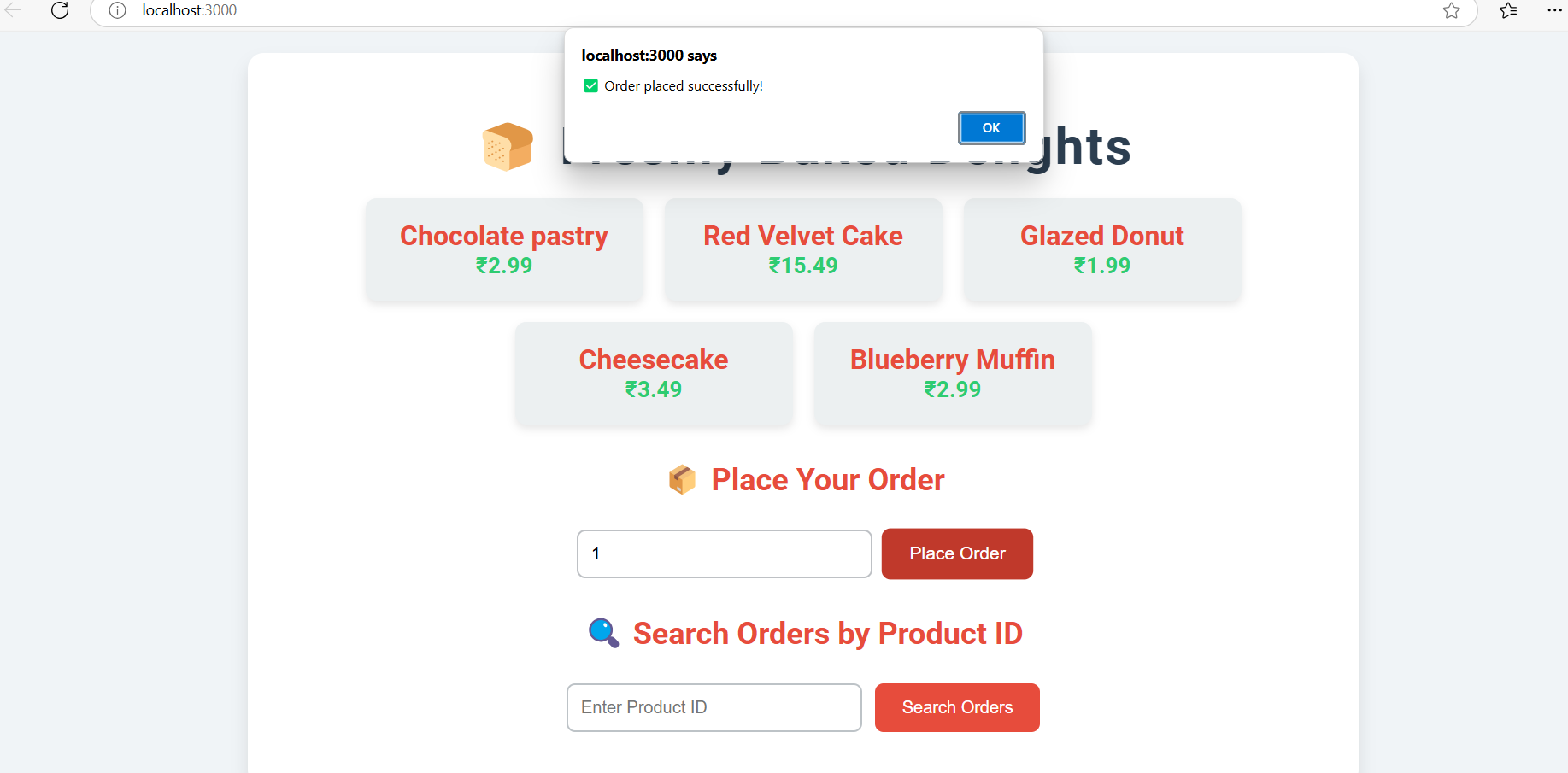
**10.2. Architecture Planning:**

* **Scalable:** RabbitMQ helps in handling more orders without slowing down.
* **Fast Performance:** Redis caching reduces waiting time for data.
* **Reliable:** PostgreSQL safely stores important data like orders.
* **Easy Maintenance:** Each part (frontend, backend, cache, message queue) is kept separate for easier changes in the future.
* **Expandable:** Can easily add new features like authentication, real-time updates, and payment integration later.

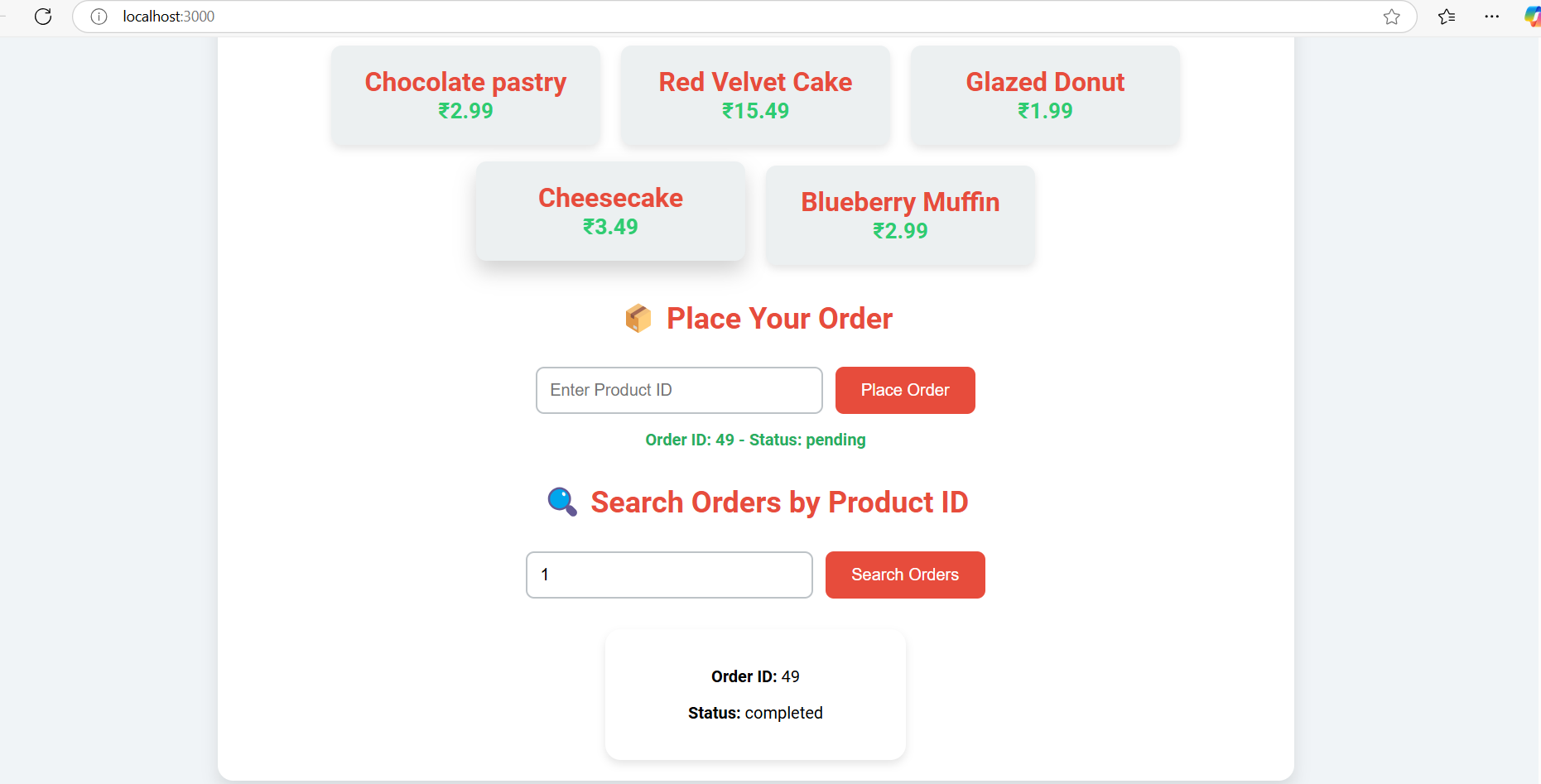
Frontend-



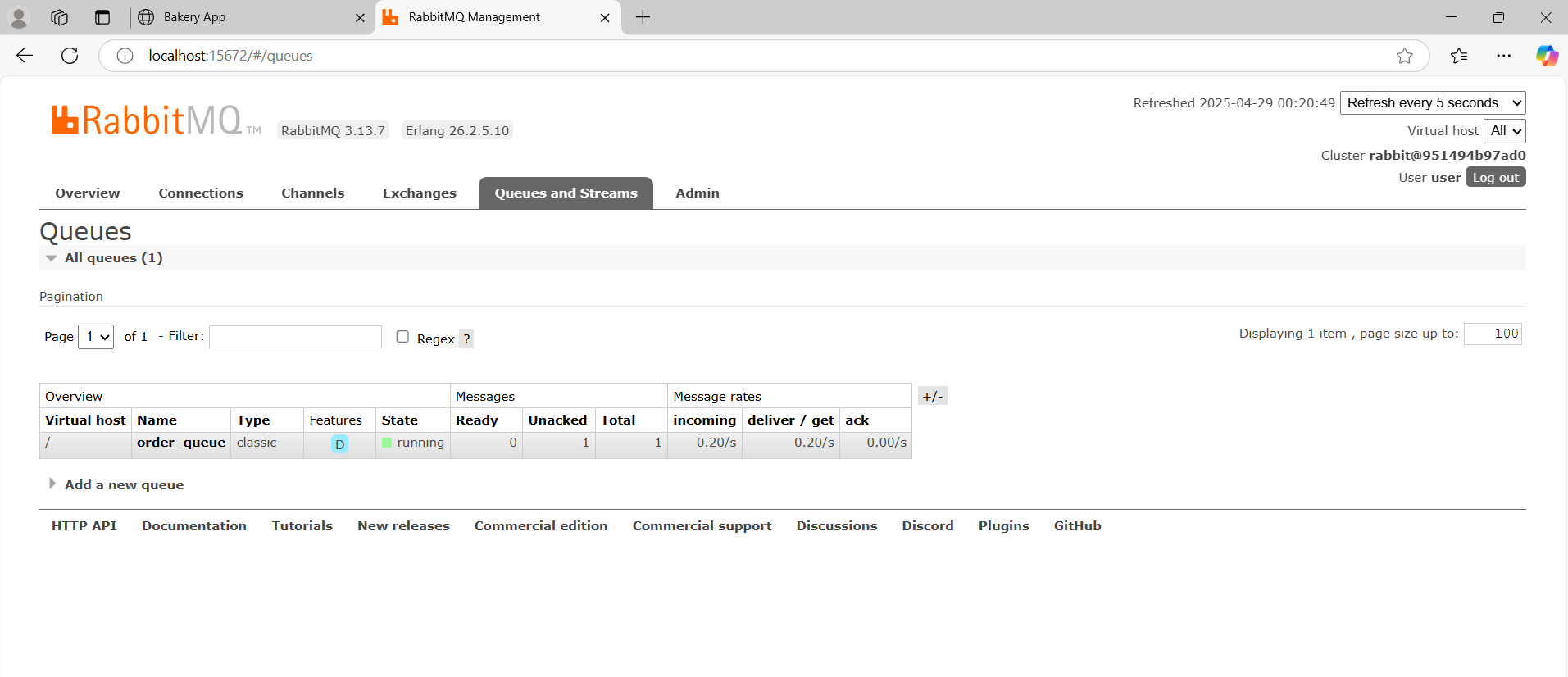
Here I am placing an order-



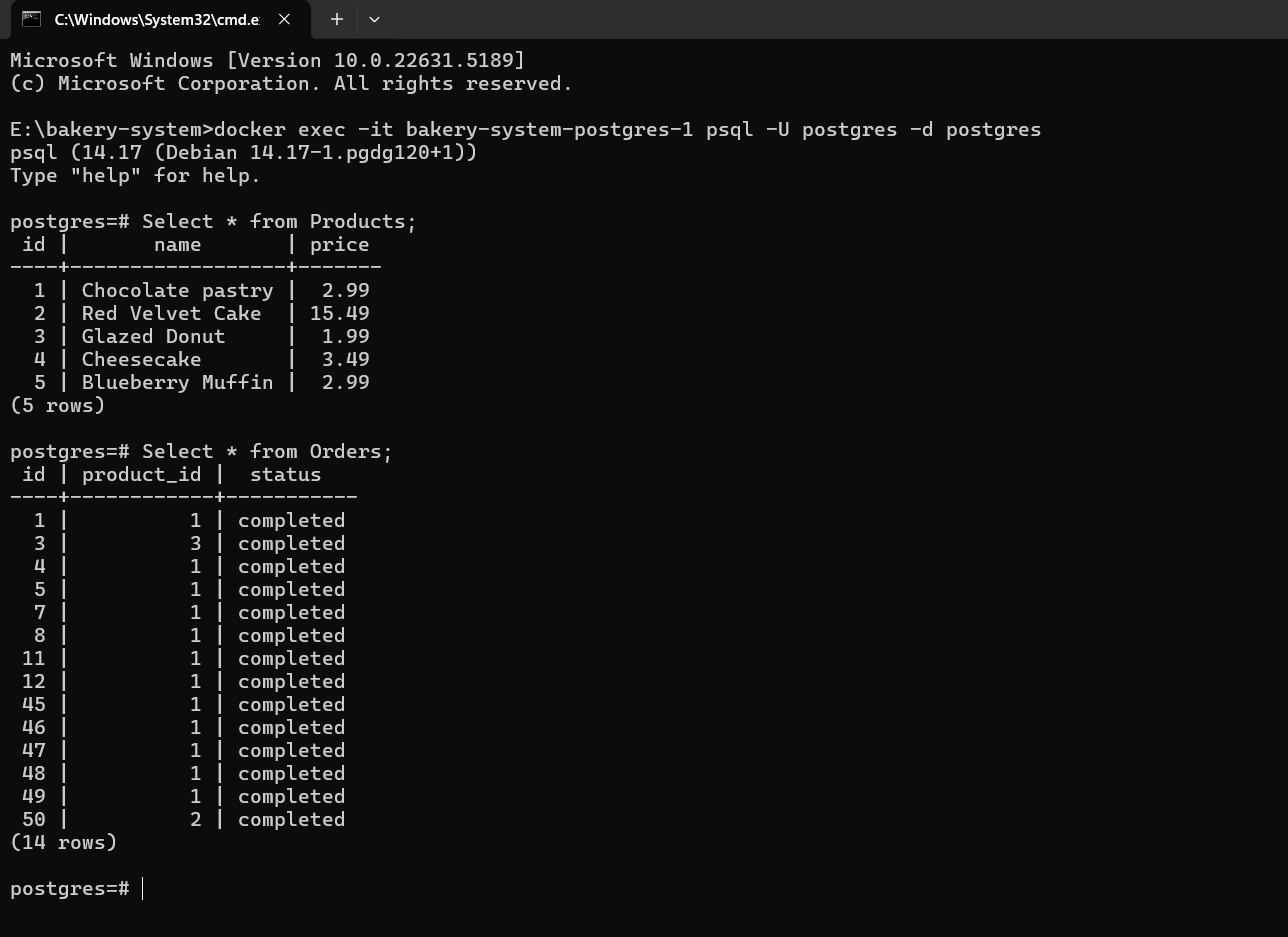
Searching the Orders by Product\_id before it was pending now the status is completed



Rabbit Mq is processing the orders



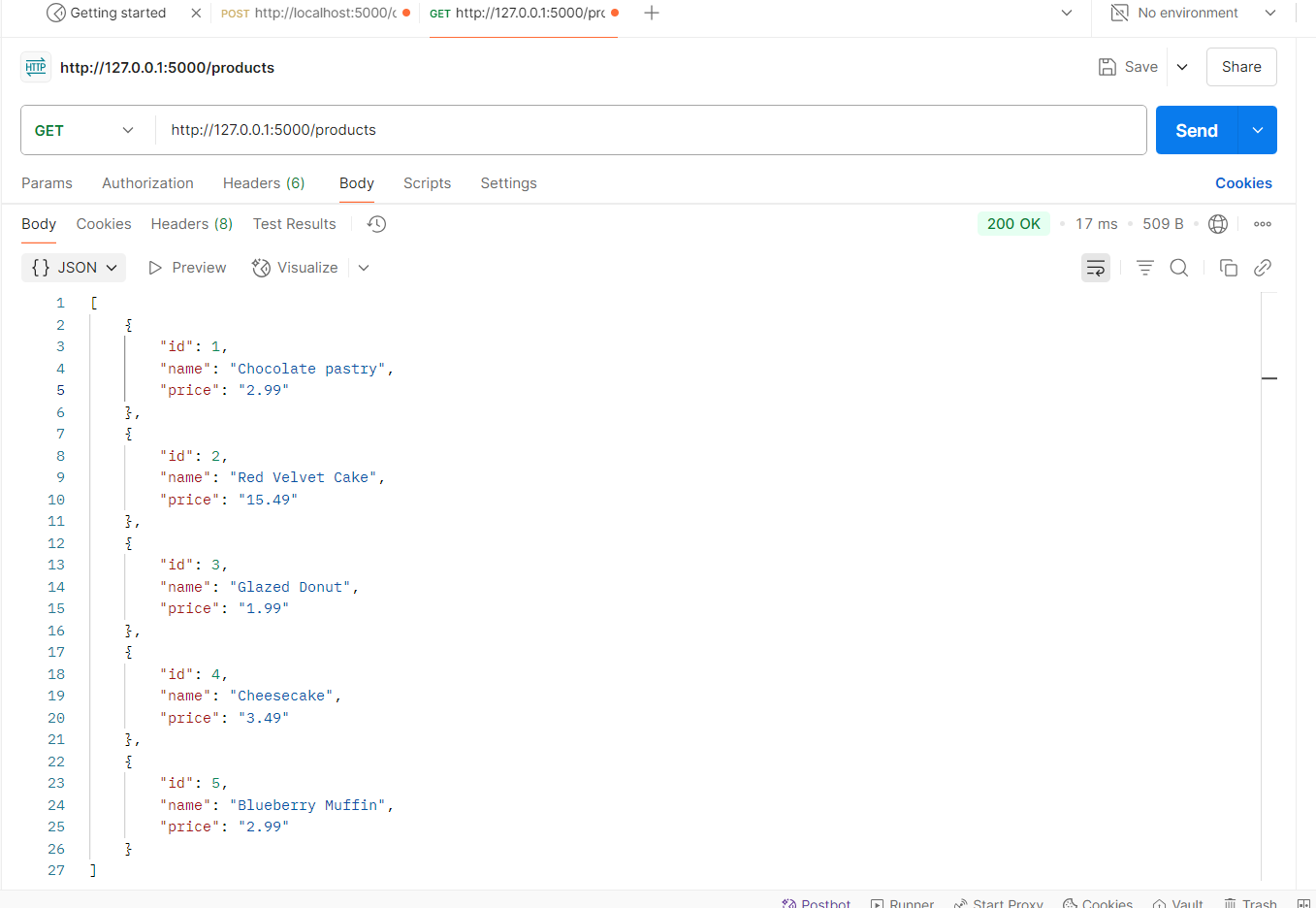
Database-



API endpoints working

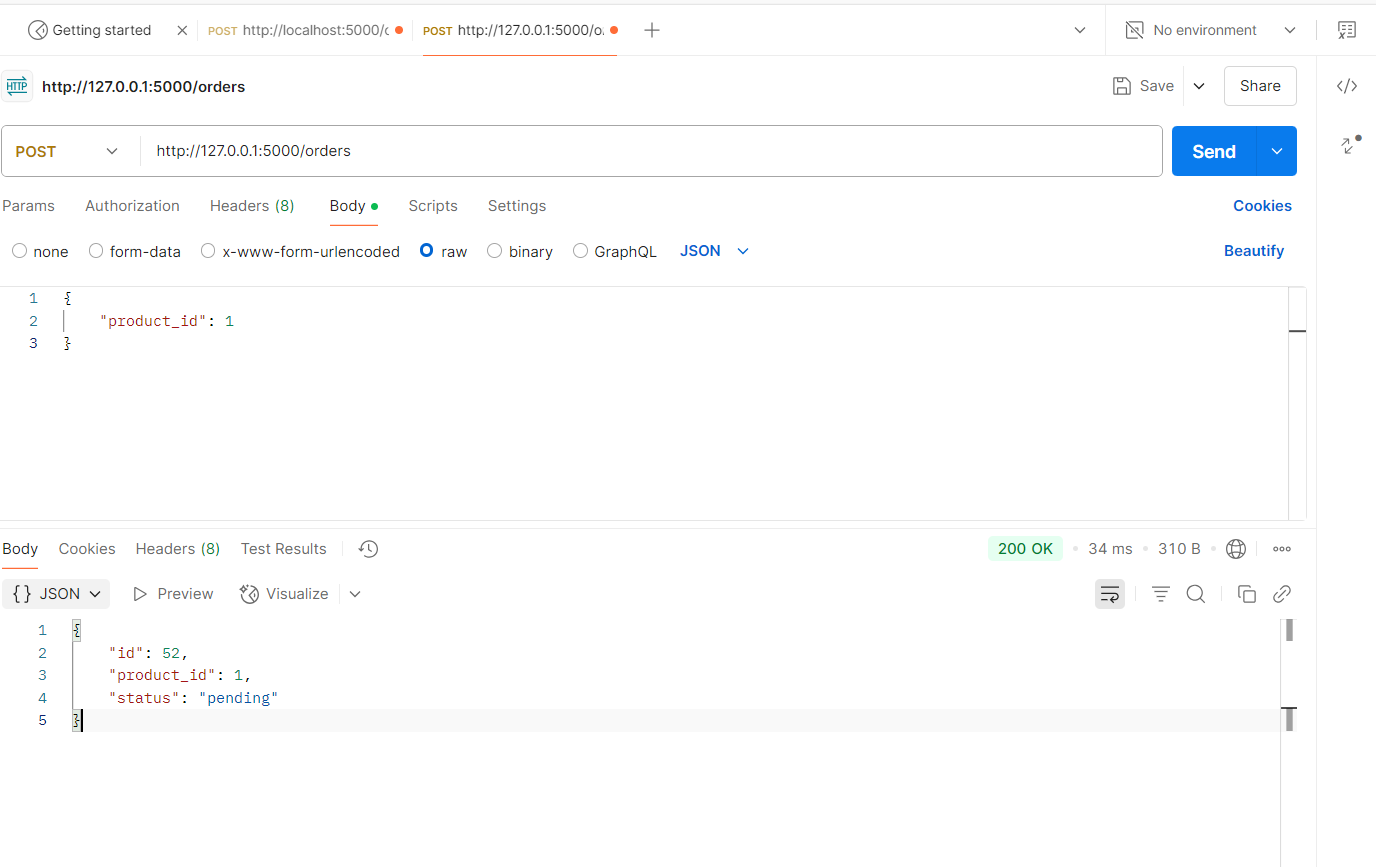
1 Get all products

<http://127.0.0.1:5000/products>



2.Create an Order

<http://127.0.0.1:5000/orders>



3.Check Orders by product\_id

<http://127.0.0.1:5000/orders/1>

