

[**Container Orchestration & Automation**](https://lms.upes.ac.in/course/view.php?id=17759)

**Assignment**

**Semester – VI**

**Submitted by : Anushka Sharma**

**SAP ID : 500108378**

**Enrollment No:R2142220955**

**Batch no : B7**

**Bakery System**

**1. Introduction**

This project builds a simple and expandable system for managing orders. It connects a React-based frontend to a Node.js backend. PostgreSQL stores all important data, Redis speeds up frequently needed queries, and RabbitMQ handles tasks like processing orders in the background.

**2. Technologies Used**

* **Frontend**: React.js
* **Backend**: Node.js, Express.js
* **Database**: PostgreSQL
* **Caching**: Redis
* **Message Queue**: RabbitMQ
* **Containerization**: Docker

**3. System Architecture Overview**

The system is neatly divided into layers to make it easier to maintain and grow:

* **Frontend Layer:** Captures user interactions and sends requests to the backend.
* **Backend Layer:** Handles all business rules, database activities, caching, and messaging tasks.
* **Database Layer:** Stores permanent data like product information and order records.
* **Caching Layer:** Improves speed for frequently asked data.
* **Messaging Layer:** Manages order processing tasks in the background, away from the main app flow.

**4. Functional Flow**

**A. Order Creation**

* The user selects a product and places an order through the frontend.
* The backend stores the order in PostgreSQL with the status set to 'pending'.
* The order details are pushed to RabbitMQ for background processing.
* The order is also stored in Redis using the product ID for faster retrieval.

**B. Order Processing**

* A background worker picks up the order from RabbitMQ.
* It processes the order and updates its status to 'completed' in PostgreSQL.
* Redis is updated accordingly to reflect the new status.

**C. Order Retrieval**

* When a user asks for orders based on product ID, the backend first checks Redis.
* If Redis has the data, it returns it instantly.
* If not, the backend fetches it from PostgreSQL and then updates Redis.

**5. API Endpoints**

* **POST /orders** → Create a new order.
* **GET /orders/:product\_id** → Fetch all orders for a specific product ID.
* **GET /products** → Retrieve the list of all available products.

**6. Database Structure**

**Products Table:**

* id (Primary key)
* name (Product name)
* price (Product price)

**Orders Table:**

* id (Primary key)
* product\_id (Foreign key linking to Products)
* status (Current status of the order: 'pending' or 'completed')

**7. RabbitMQ Usage**

* **Queue Name**: order\_queue
* **Producers**: Backend APIs publishing new orders.
* **Consumers**: Worker services processing orders and updating statuses.

**8. Frontend Overview**

* Display all available products.
* Allow users to place new orders.
* Allow users to search orders by product ID easily.

**9. Launch Services using Docker Compose**

docker-compose up --build

**10. Access Points**

* Frontend: [http://localhost:3000](http://localhost:3000/)
* Backend API: [http://localhost:5000](http://localhost:5000/)
* RabbitMQ Dashboard: [http://localhost:15672](http://localhost:15672/)

**11. Design Choices Explained**

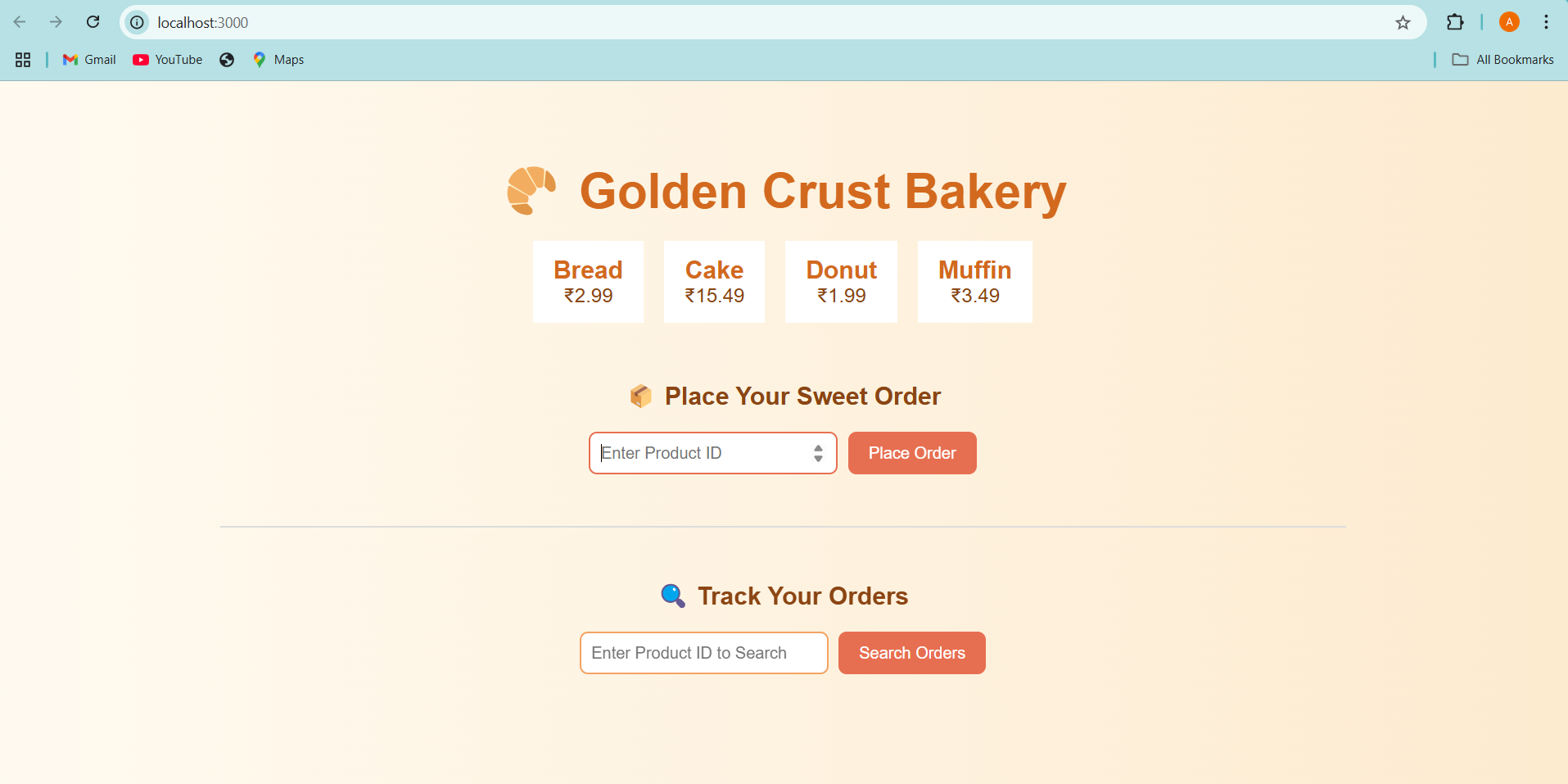
**11.1. Choice of Technologies:**

* React: Ideal for creating dynamic and quick user interfaces.
* Node.js + Express: Makes backend development efficient and fast.
* PostgreSQL: Robust and reliable database for complex operations.
* Redis: Provides fast access to cached data.
* RabbitMQ: Perfect for handling background tasks without slowing down user experience.

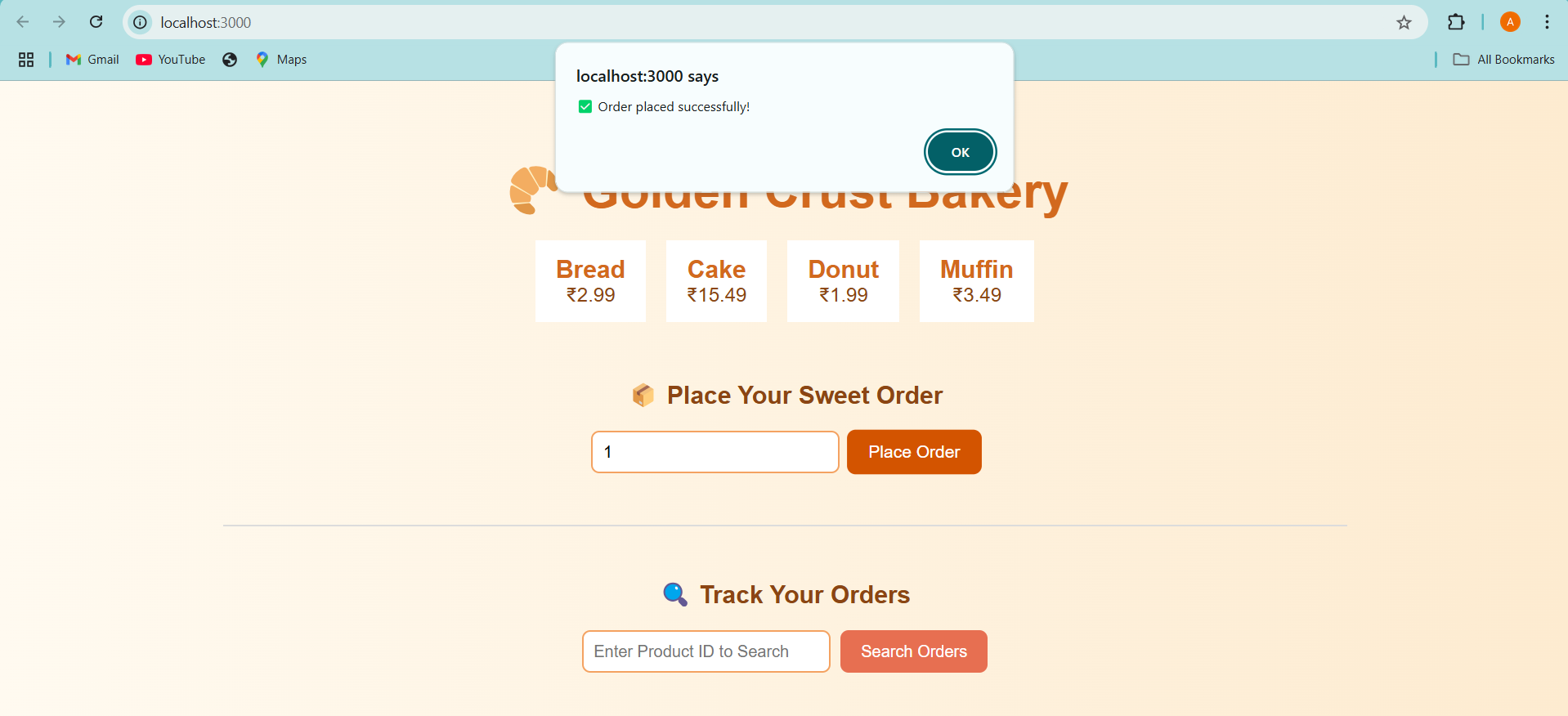
**11.2. System Planning:**

* **Scalability:** RabbitMQ ensures that many orders can be processed without slowing the system.
* **Performance:** Redis boosts the speed of reading data.
* **Reliability:** PostgreSQL safely handles and stores important data.
* **Ease of Maintenance:** Separation of frontend, backend, cache, and queue makes upgrades and changes simple.
* **Flexibility:** New features like authentication, real-time updates, and online payments can be easily added later.

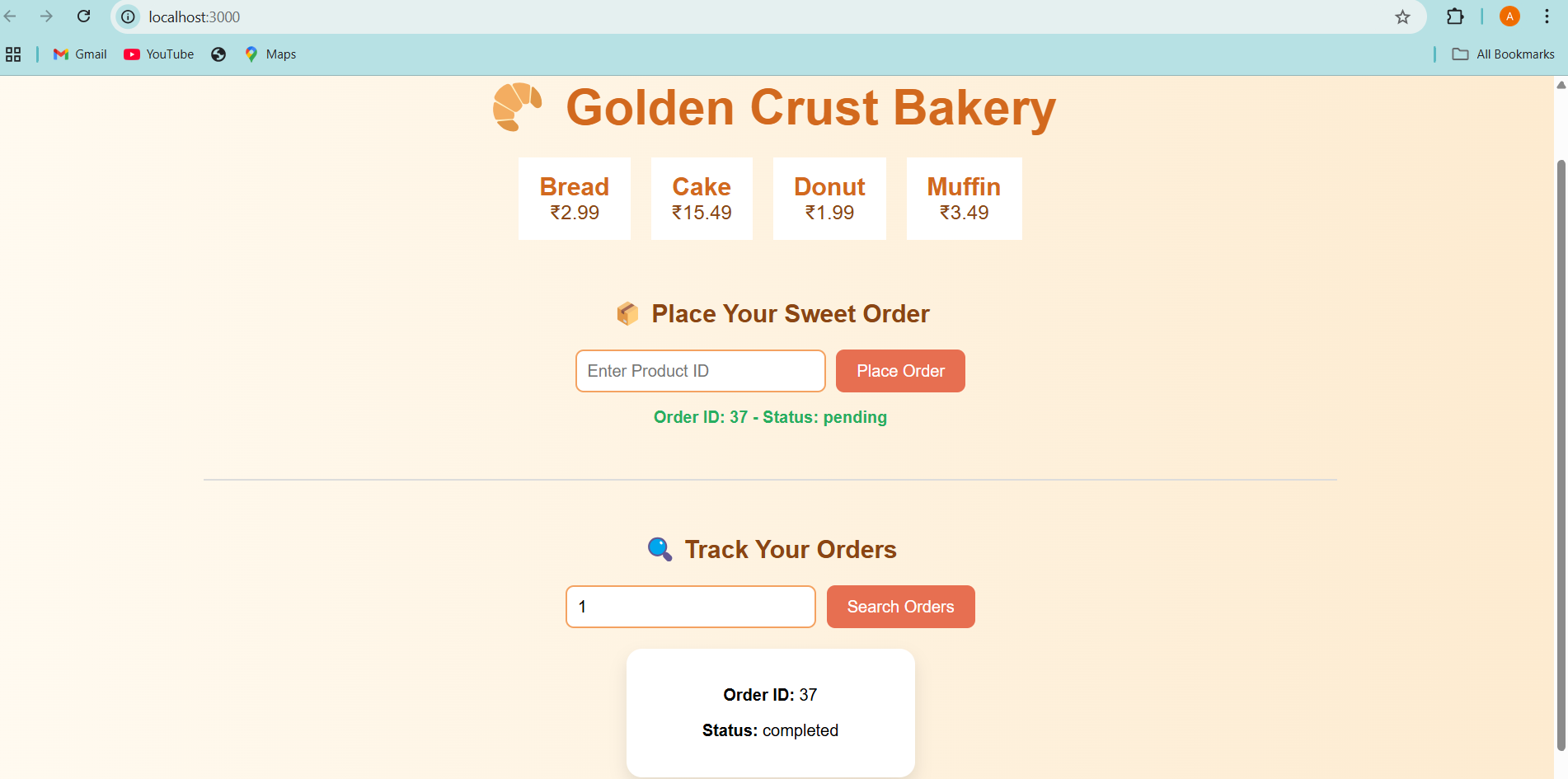
Frontend-



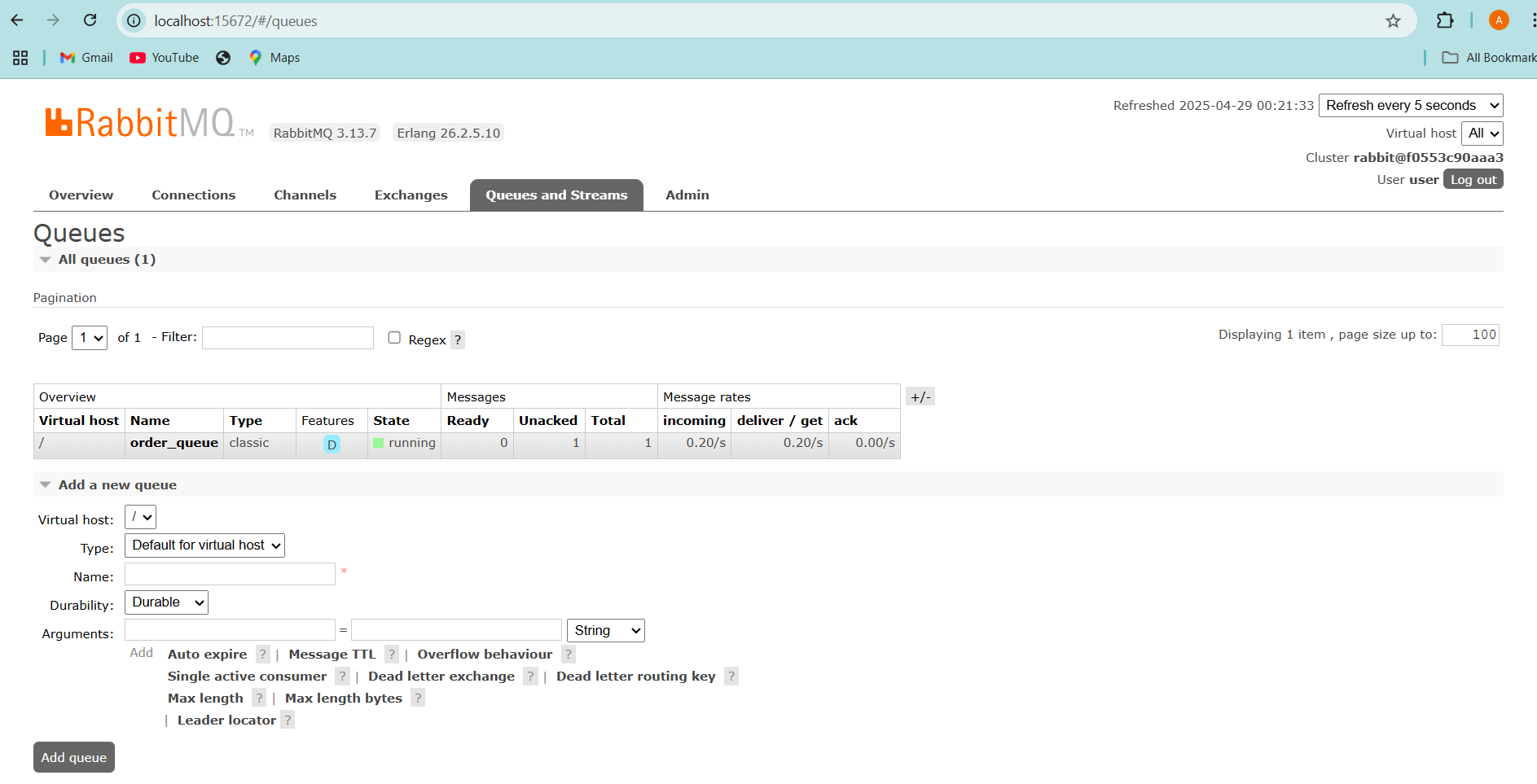
Order placed Successfully



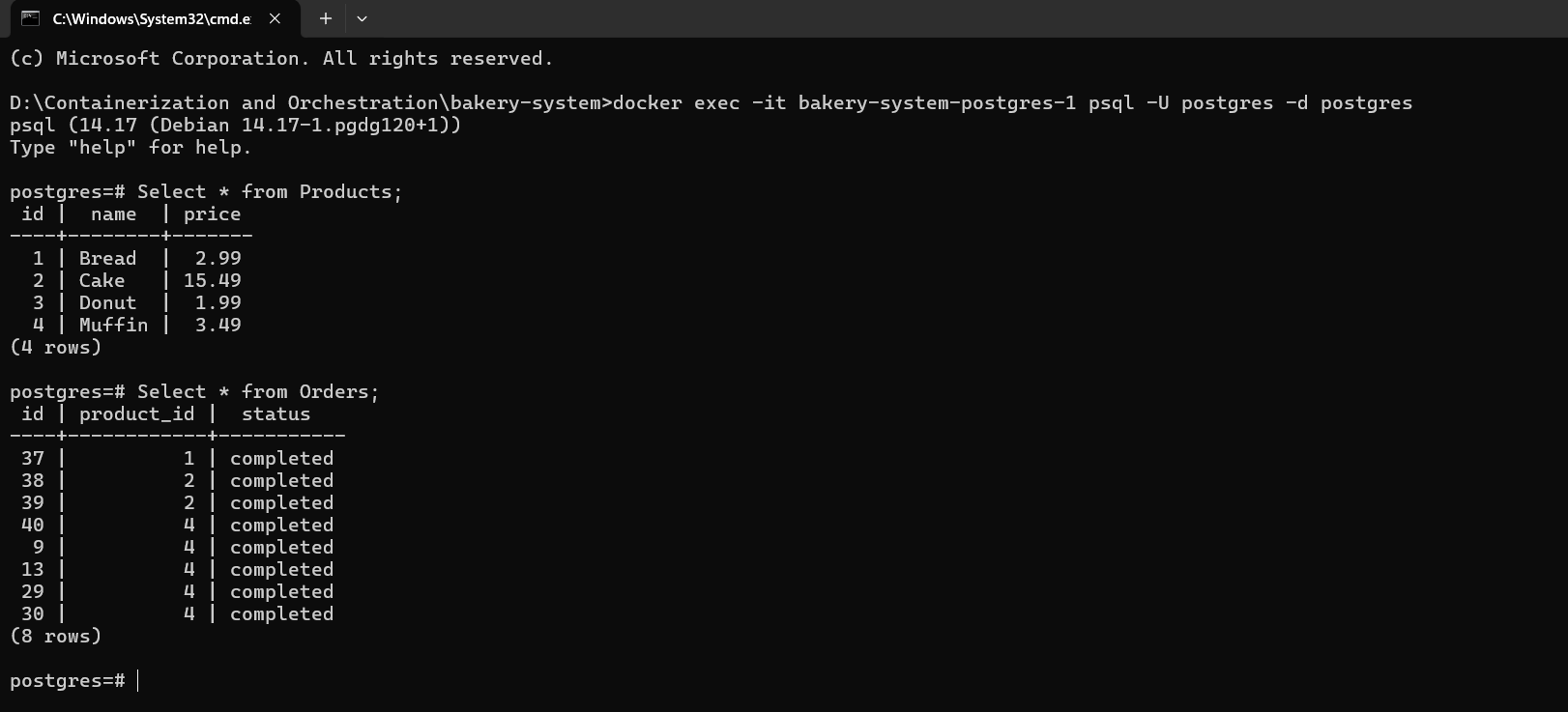
Track your order by product\_id



The orders are being processed by RabbitMQ



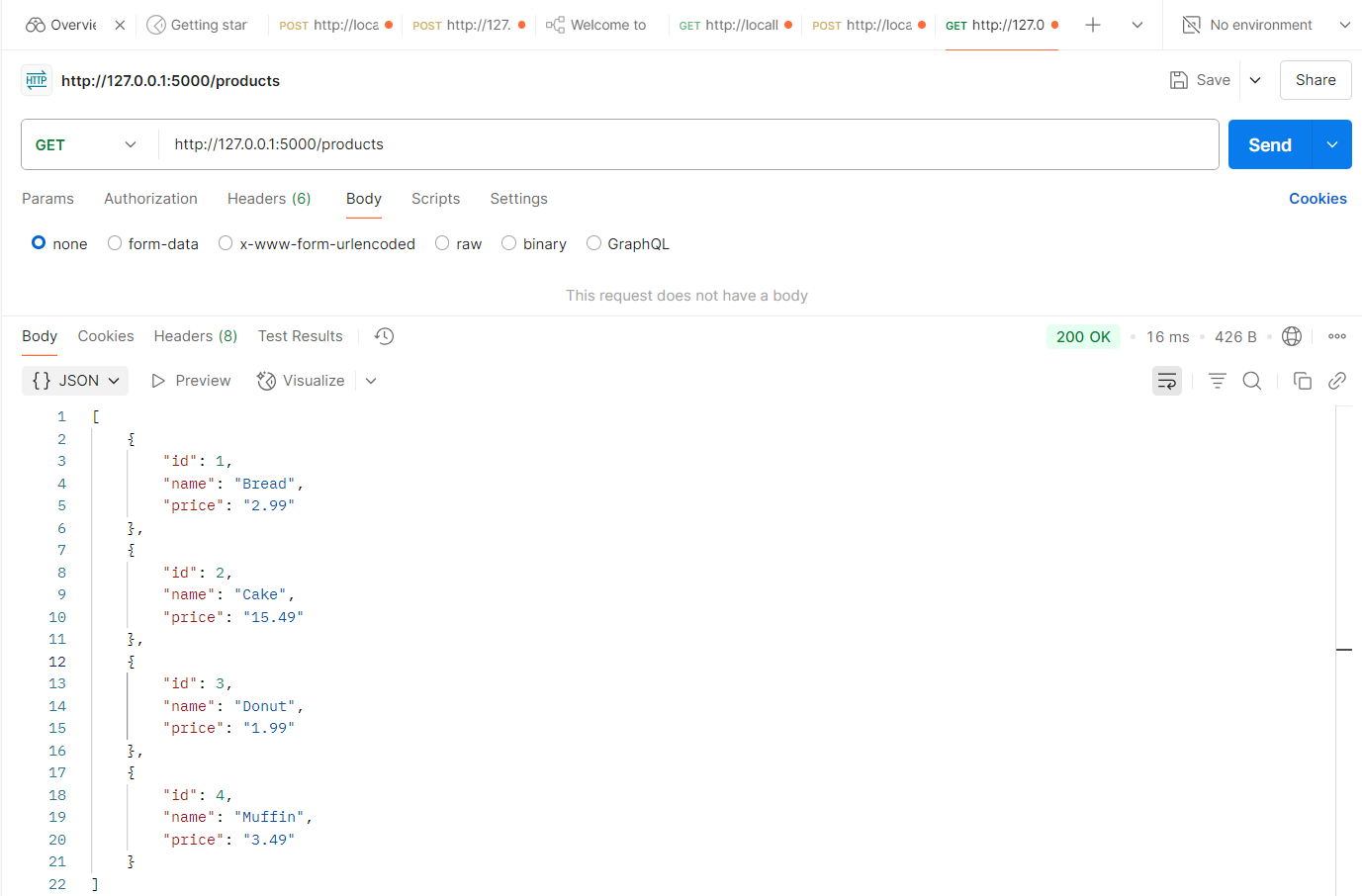
**Database**



**Working of API Endpoints**

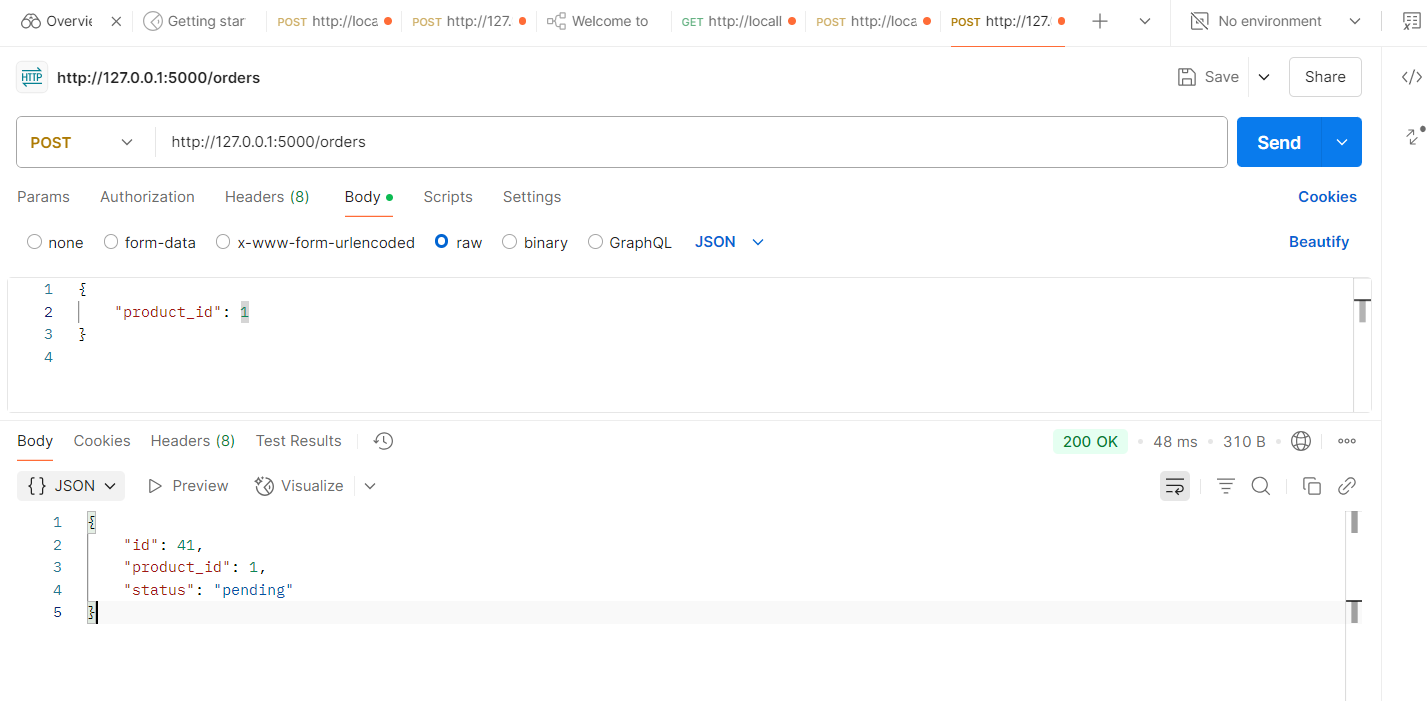
1.Get all Products

http://127.0.0.1:5000/products



2. Post an order

http://127.0.0.1:5000/orders



3. Get orders by product id.

http://127.0.0.1:5000/orders/1

