**Design Decision Report**

**Submitted by:**

**Name:** Prachita Srivastava

**Batch:** B7

**Roll number:** R2142220593

**SAP:** 500105221

**Date:** 19.04.2025

1. **Project Overview:** This full-stack bakery web application allows users to view products, place orders, and track them. It uses a containerized microservice architecture built with React, Flask, PostgreSQL, and RabbitMQ, all orchestrated with Docker Compose.
2. **Tech Stack Choice**

* **React** was chosen for its component-based structure and ease of API integration.
* **Flask** was selected for its simplicity and efficiency in building lightweight REST APIs.
* **PostgreSQL** was used as a reliable relational database for structured product/order data.
* **RabbitMQ** allowed us to offload time-consuming order processing asynchronously.
* **Docker Compose** enabled seamless container orchestration for a consistent environment.

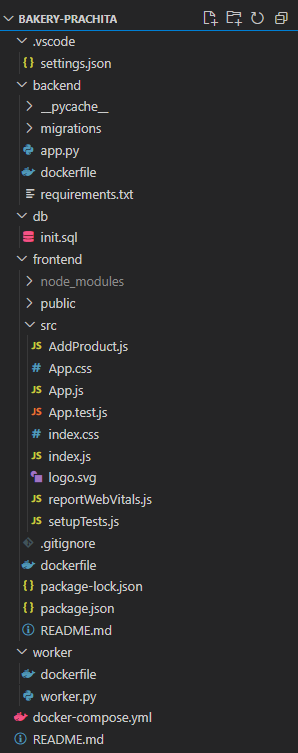
1. **Backend Flow**
2. Frontend sends a product/order request to Flask API.
3. Flask inserts data into PostgreSQL (using SQLAlchemy ORM).
4. For orders, Flask also publishes a message to RabbitMQ.
5. A background **Python Worker** (also a container) listens to the queue.
6. The worker processes the order and updates its status in the database.
7. **Advantages**

* **Scalability**: Order processing is decoupled from the main app using RabbitMQ.
* **Maintainability**: Clean separation between services (frontend, backend, DB, queue).

1. **Challenges & How We Solved Them**

* **Cross-Origin Issues**: Resolved using Flask-CORS middleware to allow frontend-backend communication.
* **Docker Networking**: Used Docker Compose networks so all services can communicate by container name.
* **Async Communication**: Used pika for robust connection handling between Flask and RabbitMQ.

1. **Folder Structure**



1. **Future Improvements**

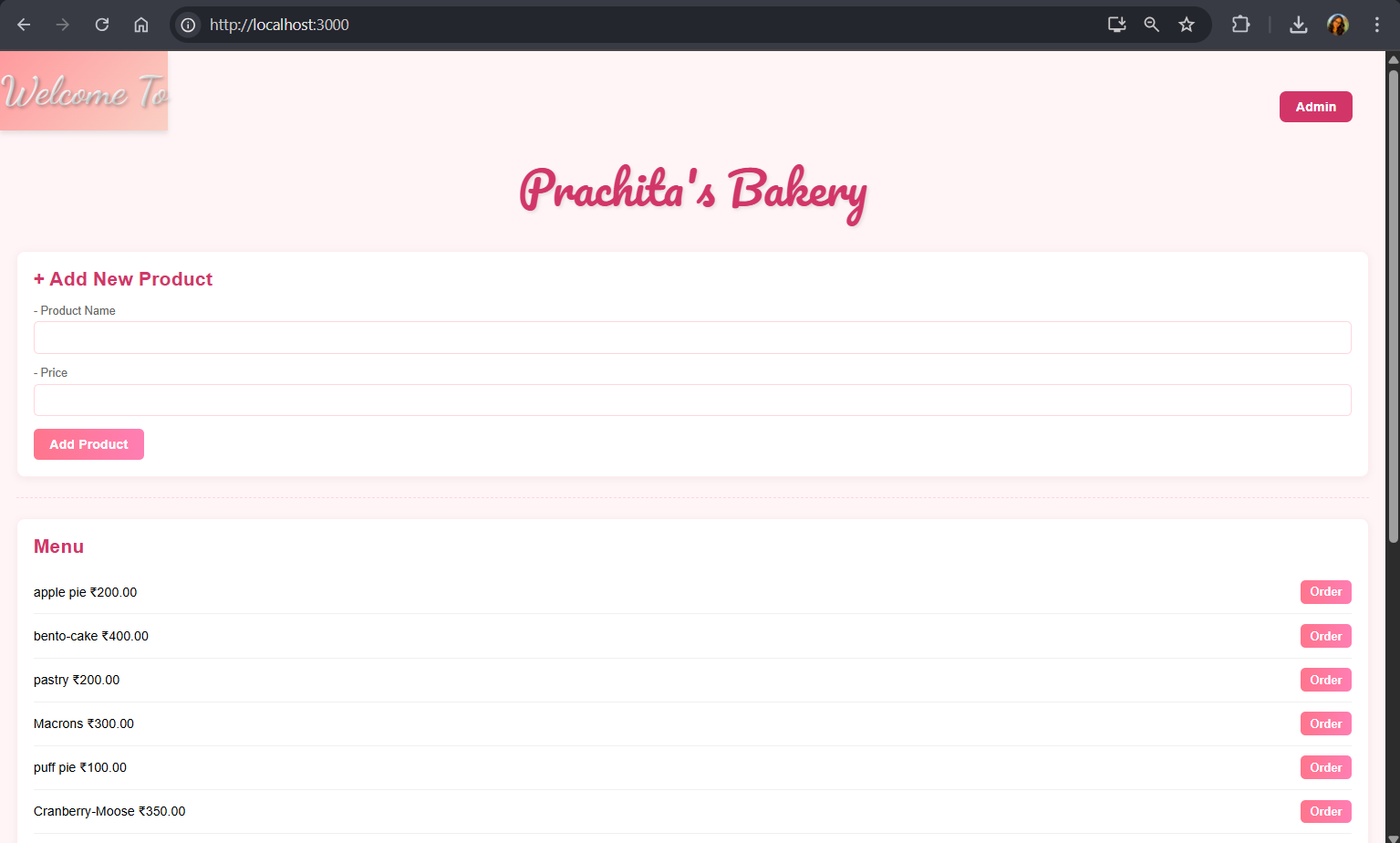
* Add user authentication and admin dashboard
* Implement category filtering for products
* Add email/SMS notification integration
* CI/CD pipeline for deployment
* Dashboard analytics for bakery operations

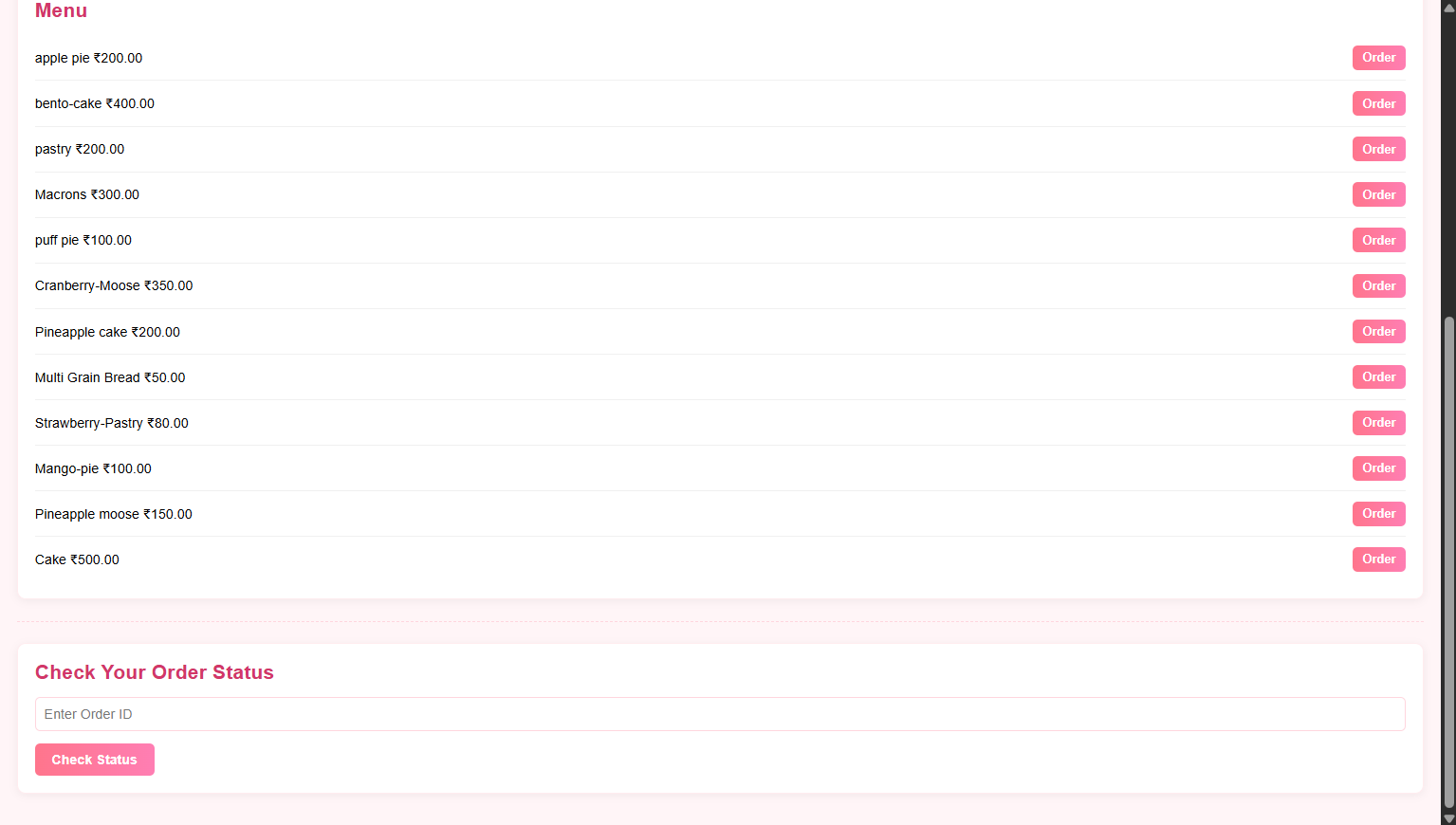
1. **API Endpoints:**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| GET | /products | Get all products |
| POST | /products | Add a new product |
| POST | /order | Place a new order |
| GET | /order/<id> | Get order status by ID |

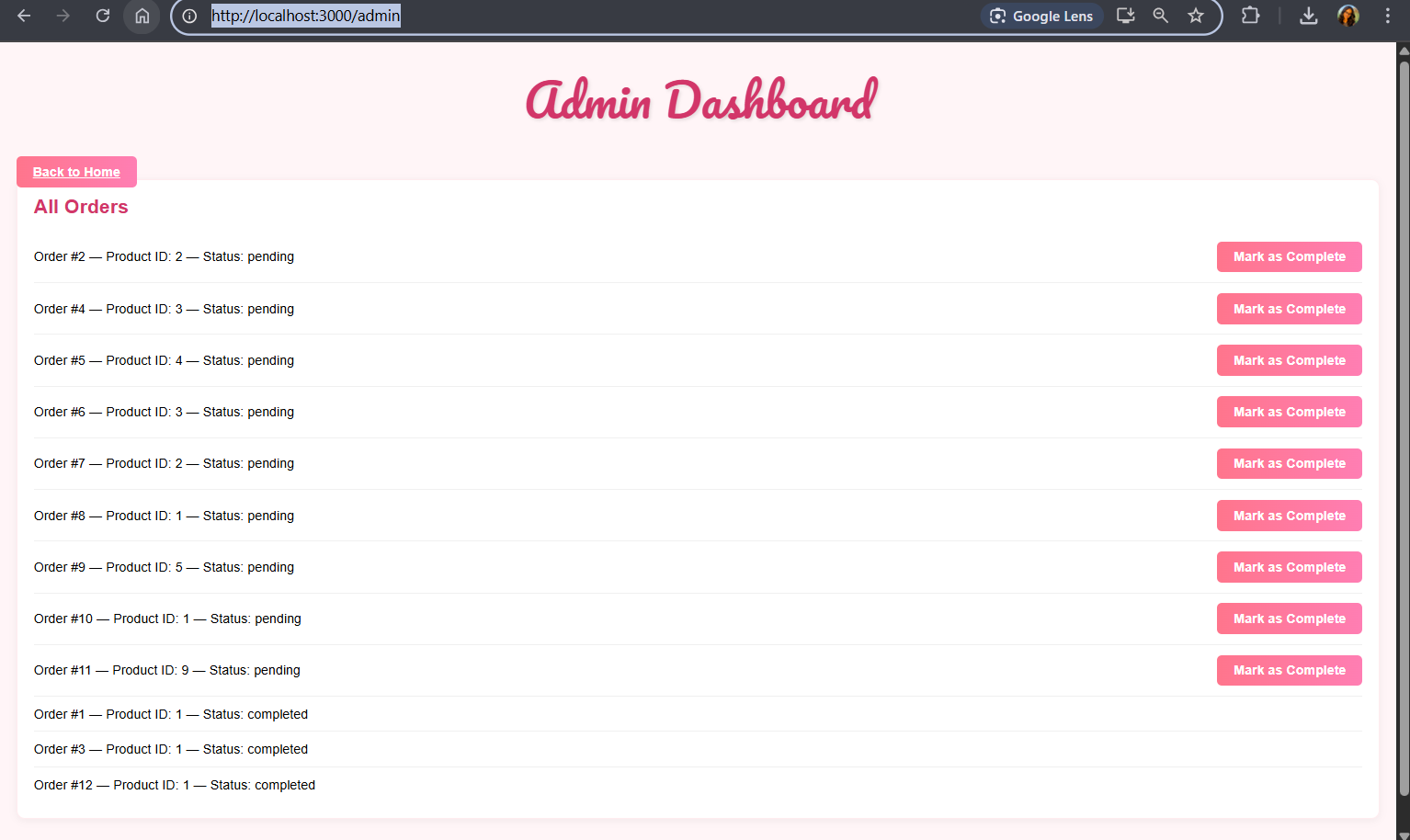
1. **Screenshots:**
2. **FRONTEND:**

* ENDPOINT: <http://localhost:3000>



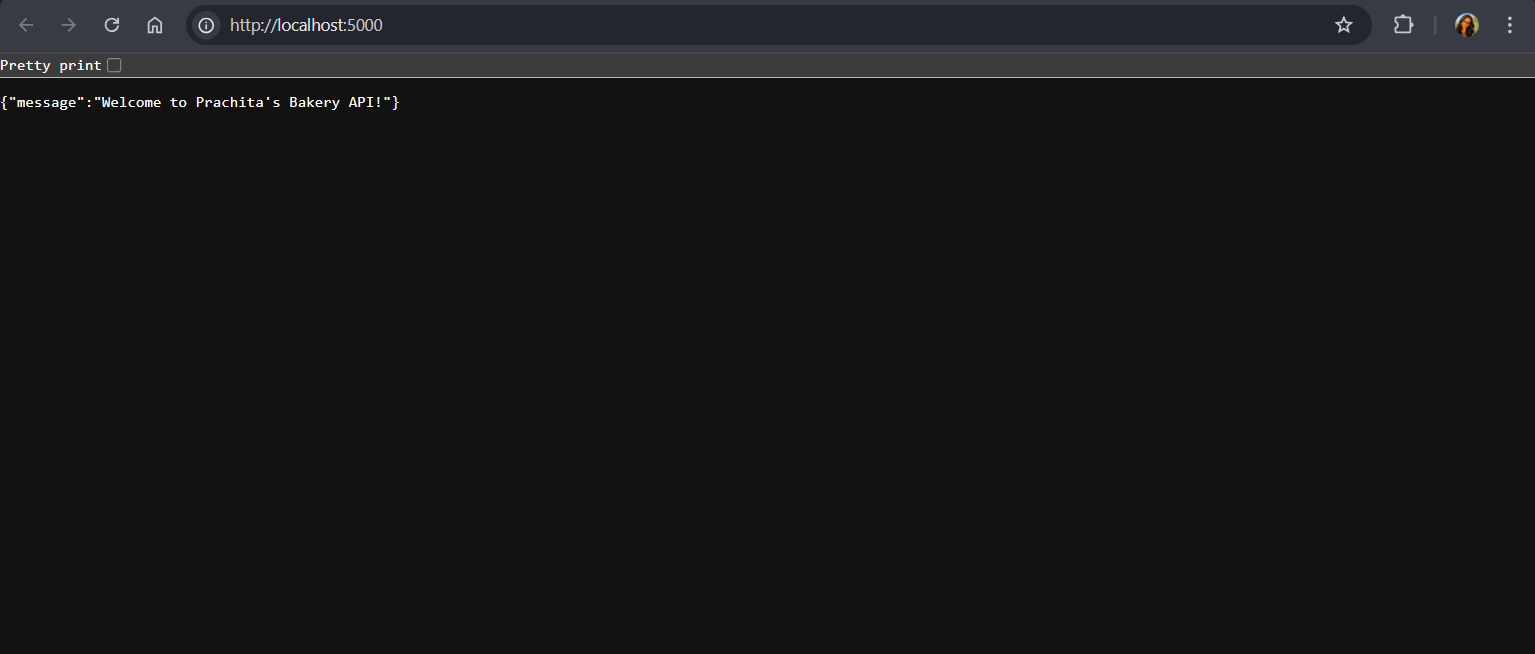


* ENDPOINT: <http://localhost:3000/admin>

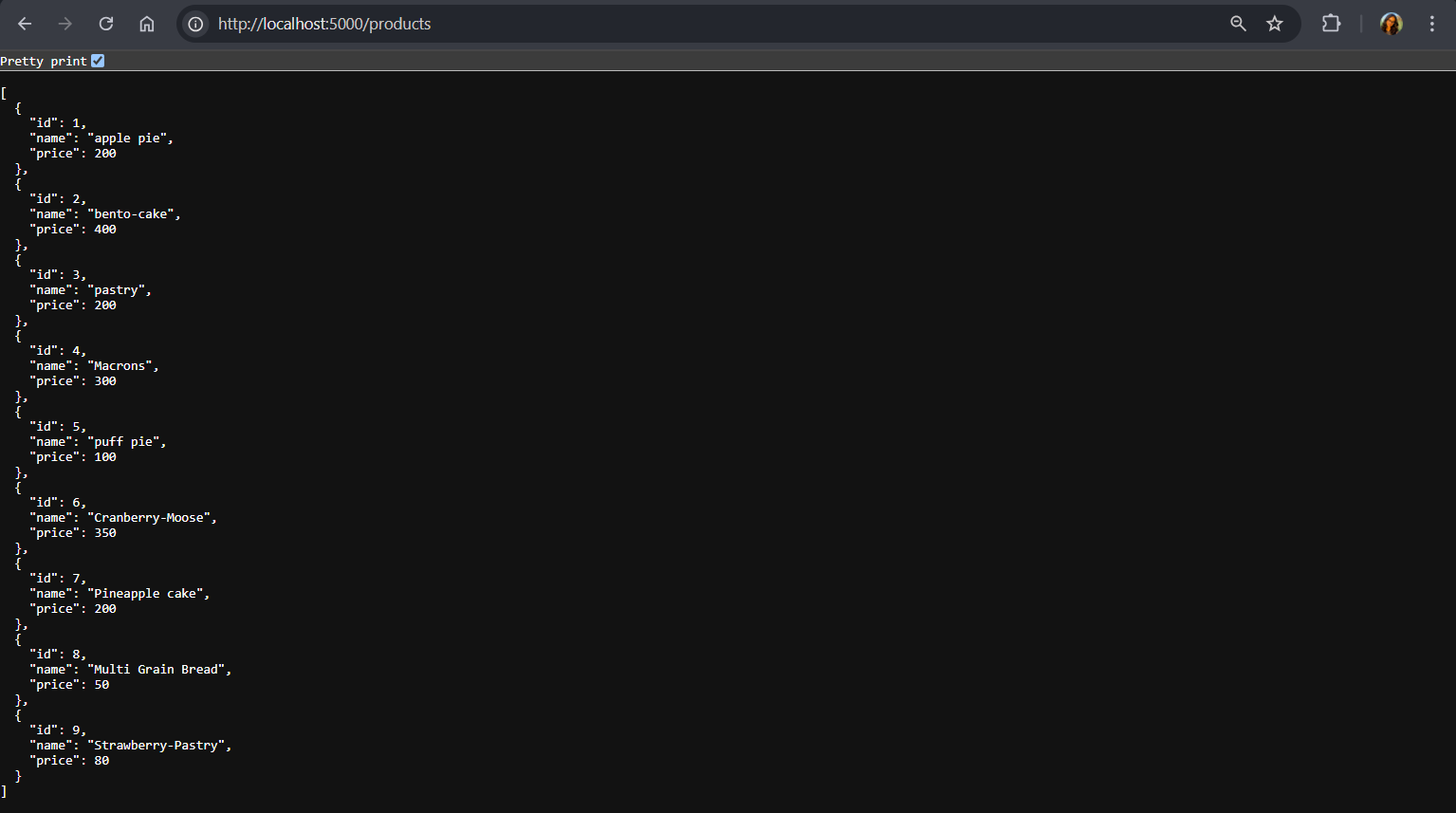


1. **BACKEND:**

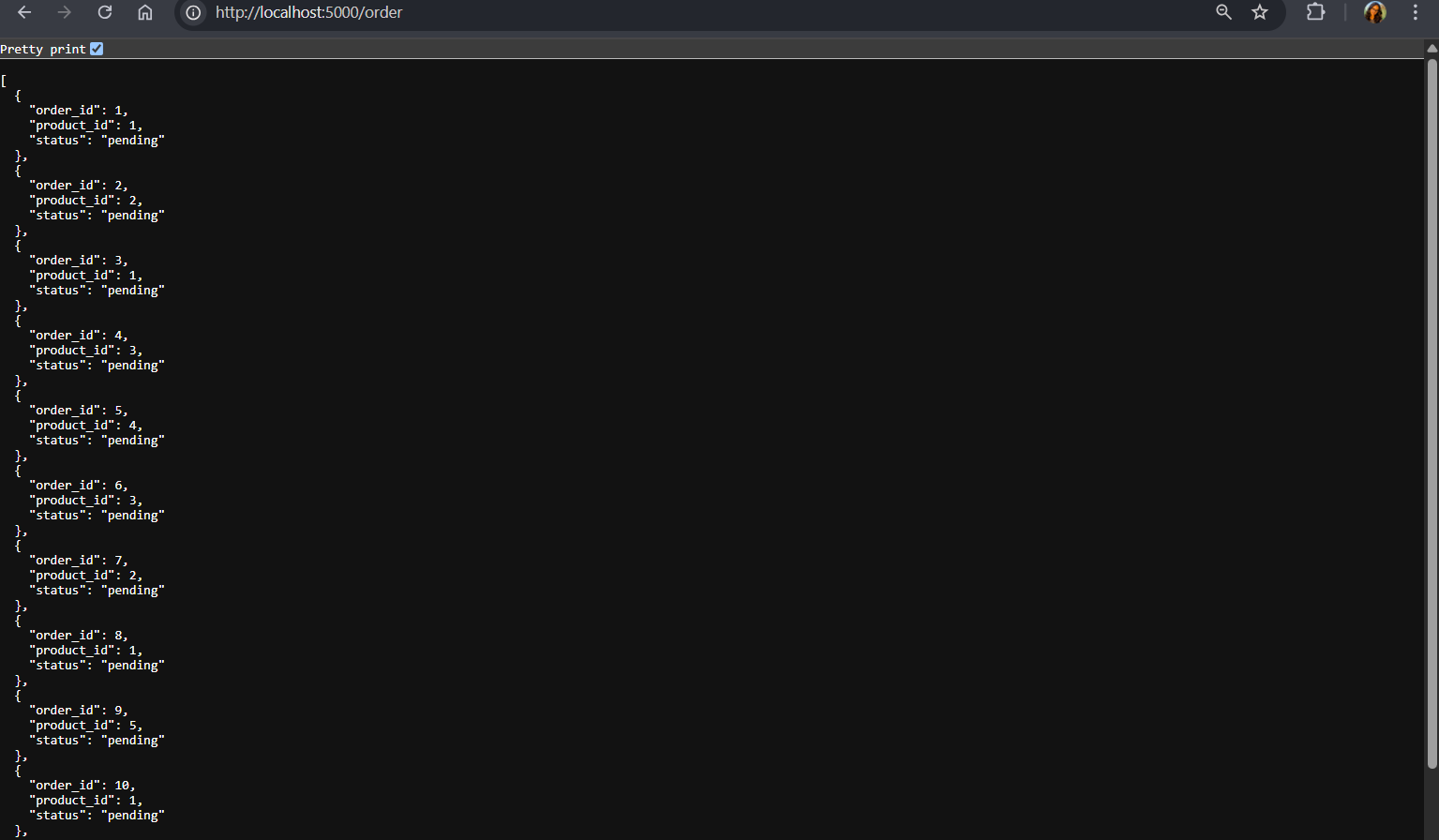
* ENDPOINT: <http://localhost:5000>



* ENDPOINT: <http://localhost:5000/products>

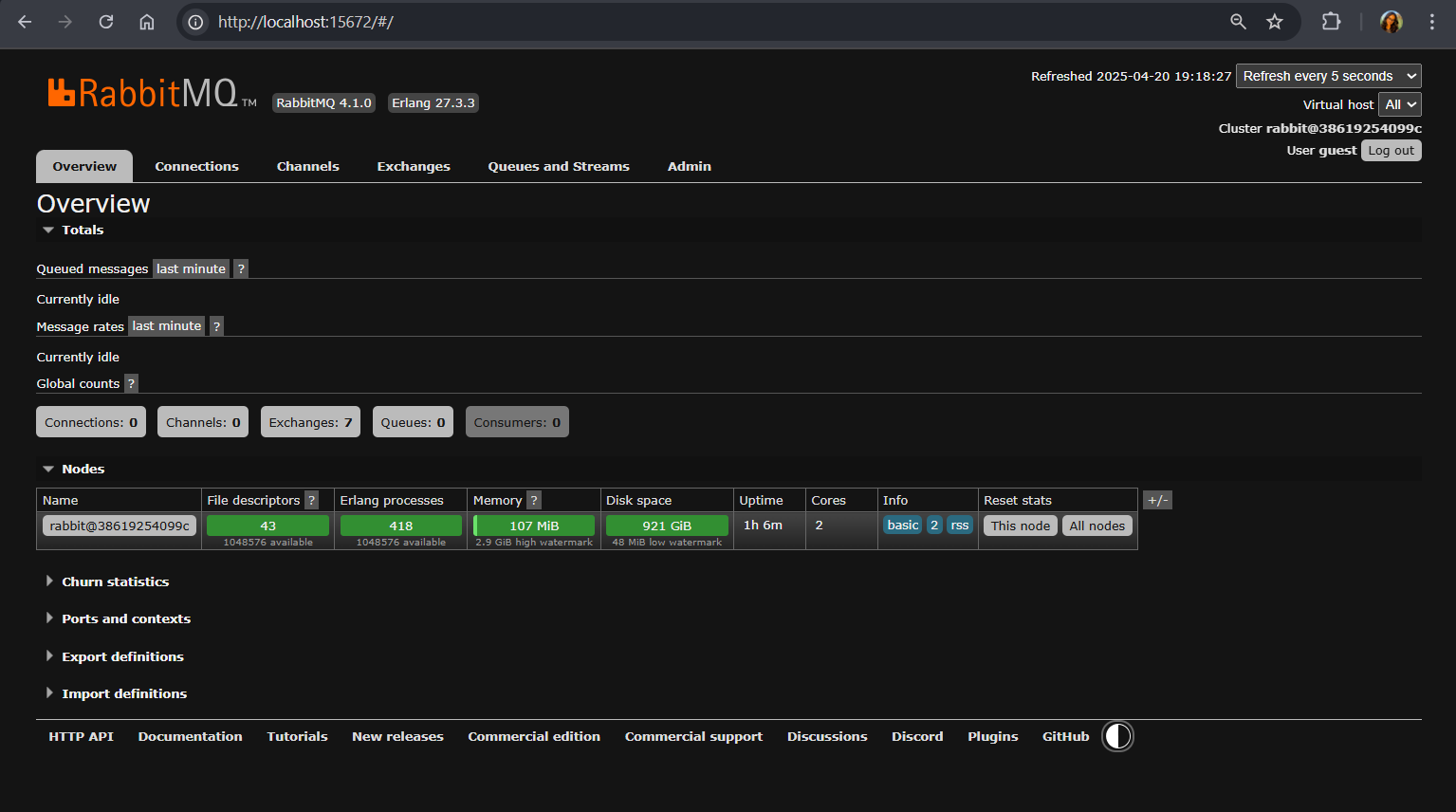


* **ENDPOINT**: <http://localhost:5000/order>



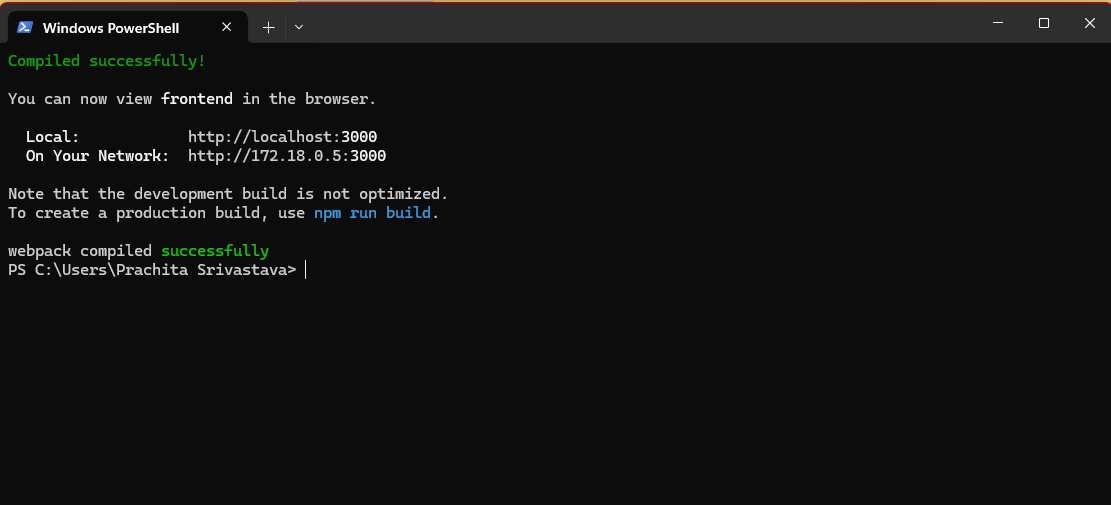
1. **RabbitMQ:**

* ENDPOINT: <http://localhost:15672/#/>

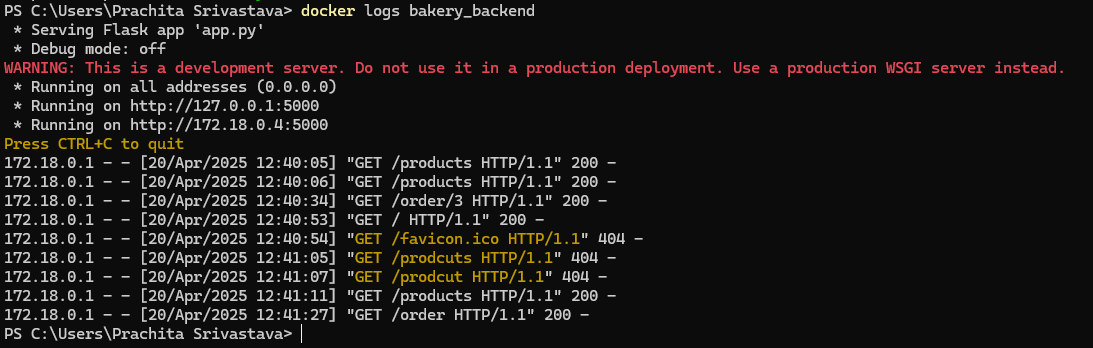


1. **Configuring logs:**

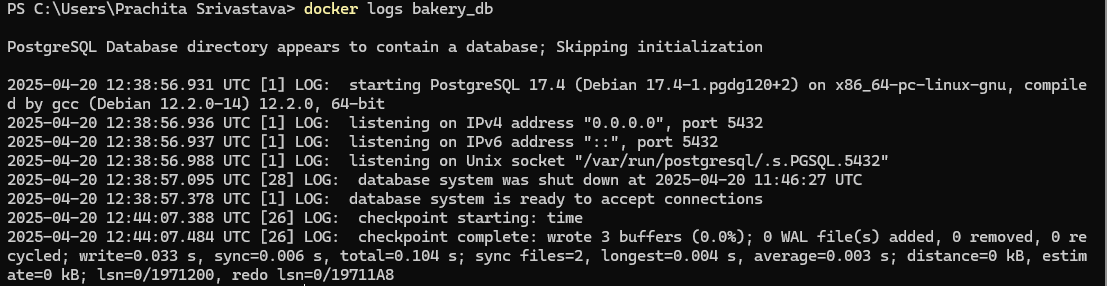
* **Frontend:** docker logs bakery\_frontend



* **Backend:** docker logs bakery\_backend



* **Database:** docker logs bakery\_db



* **RabbitMQ:** docker logs bakery\_rabbitmq

