Program: B.Tech(CSE-CCVT)

Course: Container Orchestration & automation

ASSIGNMNET 1 – REPORT

Submitted by

VIBHAV.

500101994

B-7

7TH HEAVEN- BAKERY MANAGEMENT SYSTEM USING DOCKER COMPOSE

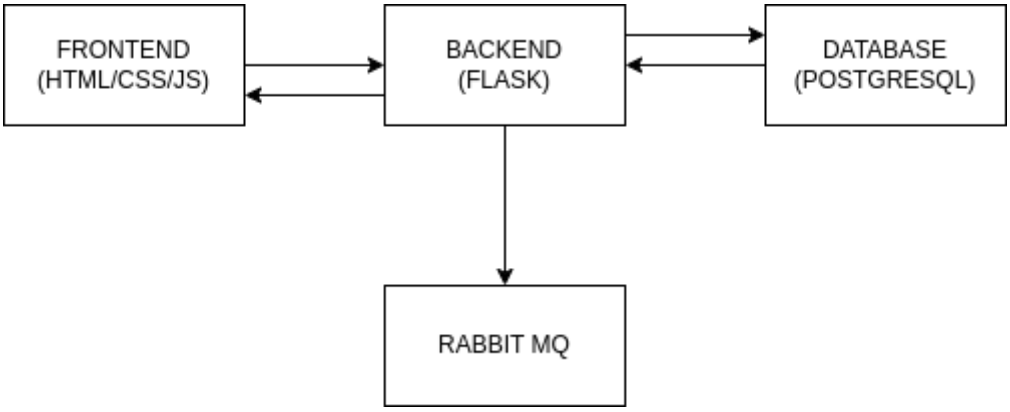
This report outlines the design and implementation decisions made for the 7th Heaven Bakery System, a containerized application that includes a database, backend API, frontend web application, and message queue. The system allows customers to browse bakery products, place orders, and check order status.

System Architecture

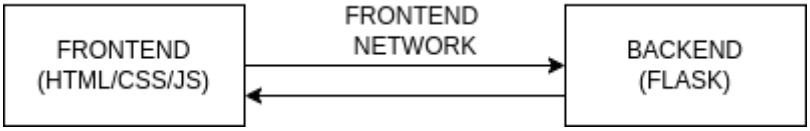
The system consists of four main components:

1. **PostgreSQL Database**: Stores information about bakery products and customer orders along with ordered items.
2. **Backend API (Flask)**: Provides endpoints for listing products, placing orders, and checking order status.
3. **Frontend (HTML/CSS/JS)**: A user-friendly web interface to access the api endpoints and place orders.
4. **RabbitMQ**: Handles asynchronous processing of orders.
5. **Docker compose file**: To manage all the containers along with the health checks, resource limits, environment variables and networking between components as frontend-network and backend-network where bakery-frontend-network is used by frontend and backend to connects the frontend container (bakery-frontend) to the backend container (bakery-backend). The backend-network is used by backend, db(PostgreSQL) and rabbitmq. This ensures isolation and security between frontend, backend and internal components. All the voumes, environment variuables, etc. Are configured inside the .yaml file as well

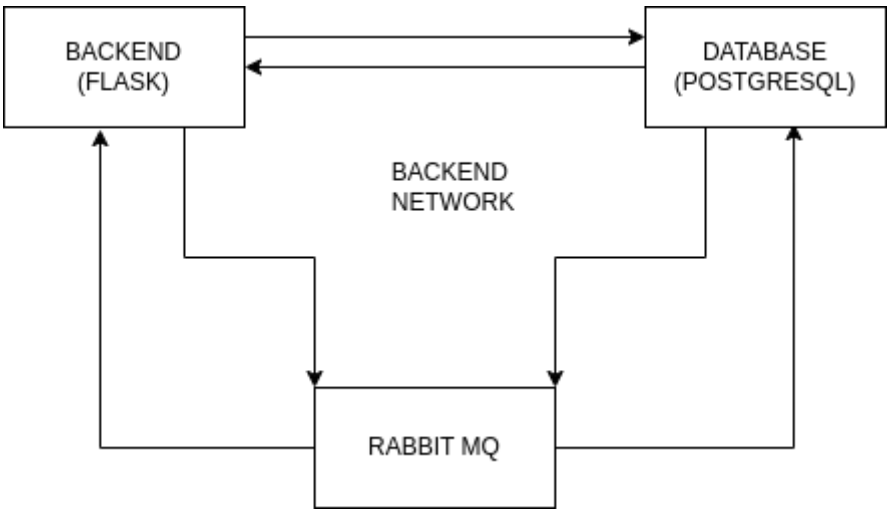
All components are containerized using Docker and orchestrated with Docker Compose.



Connection between components



Frontend Network



Backend network

Project directory structure

Each component runs in its own Docker container, orchestrated using Docker Compose. This approach provides several benefits:

- **Isolation**: Each component operates independently
- **Scalability**: Individual components can be scaled as needed
- **Maintainability**: Services can be updated or replaced without affecting others
- **Portability**: The entire system can run consistently across different environments
- **Small Blast Radius:** Even if one service fails others keep working as they are containerized separately and have different networks.
- **Security:** Separate networks for frontend and backend components provide security and isolation.

Application Flow

1. The user interacts with the frontend web interface
2. The frontend makes API calls to the backend service
3. The backend service queries the database for data
4. When orders are placed, the backend sends a message to RabbitMQ
5. RabbitMQ queues the message for asynchronous processing

Implementation Details:

- **Database (PostgreSQL):**
  The database schema includes three tables:
  - `products`: Stores bakery products information.
  - `orders`: Stores customer order details.
  - `order_items`: Stores the relationship between orders and products.

- **Backend API (Flask):**
  The API includes three endpoints:
  - 1. `GET /api/products`: Lists all available bakery products.
  - 2. `POST /api/orders`: Places a new order.
  - 3. `GET /api/orders/<id>`: Checks the status of an existing order.

- **Frontend (HTML/CSS/JS):**
  The frontend was implemented using HTML, CSS, and JavaScript.

- **Message Queue (RabbitMQ):**

  RabbitMQ is used to handle order processing asynchronously, which helps decouple the order placement from the processing workflow.

Advanced Features Implementation

1. Health Checks

Health checks were implemented for all containers to ensure robustness and reliability:

- **PostgreSQL**: Uses pg_isready command to verify database availability.
- **RabbitMQ**: Uses rabbitmqctl status to check the message broker status.
- **Backend**: Provides a /health endpoint that returns HTTP 200 when healthy.
- **Frontend**: Uses a simple CURL check to verify the service is responding.

These health checks help Docker Compose monitor the state of each service and restart them if needed.

Implementation example:

```python
@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({"status": "healthy"}), 200
```

```
restart: unless-stopped
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U vibhav -d bakery"]
  interval: 10s
  timeout: 5s
  retries: 5
```

```
vibhav@VIBHAVsUBUNTU:~/Desktop/7th-heaven-bakery/7th-heaven-bakery$ docker inspect bakery-frontend
```


```
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 5979,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-04-20T10:13:36.569910362Z",
            "FinishedAt": "0001-01-01T00:00:00Z",
            "Health": {
                "Status": "healthy",
                "FailingStreak": 0,
                "Log": [
                    {
                        "Start": "2025-04-20T15:52:07.561149528+05:30",
                        "End": "2025-04-20T15:52:07.612377005+05:30",
                        "ExitCode": 0,
```

2. Resource Limits

Resource limits were defined for all containers to prevent resource exhaustion and ensure fair resource allocation:

- **PostgreSQL**: 0.5 CPU, 512MB memory
- **RabbitMQ**: 0.3 CPU, 256MB memory
- **Backend**: 0.5 CPU, 256MB memory
- **Frontend**: 0.3 CPU, 128MB memory

These limits were chosen based on typical resource usage patterns for each service type, with higher allocations for more intensive services like the database.

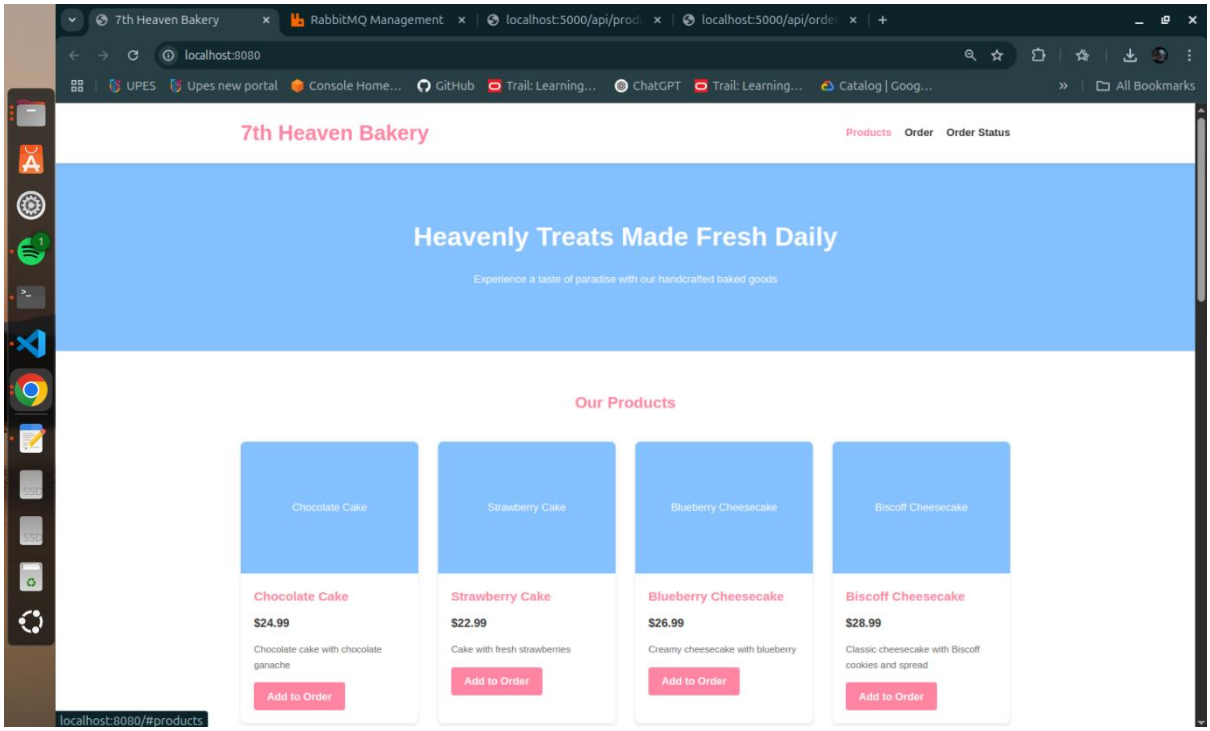
Implementation example:

```
deploy:
  resources:
    limits:
      cpus: '0.3'
      memory: 128M
```

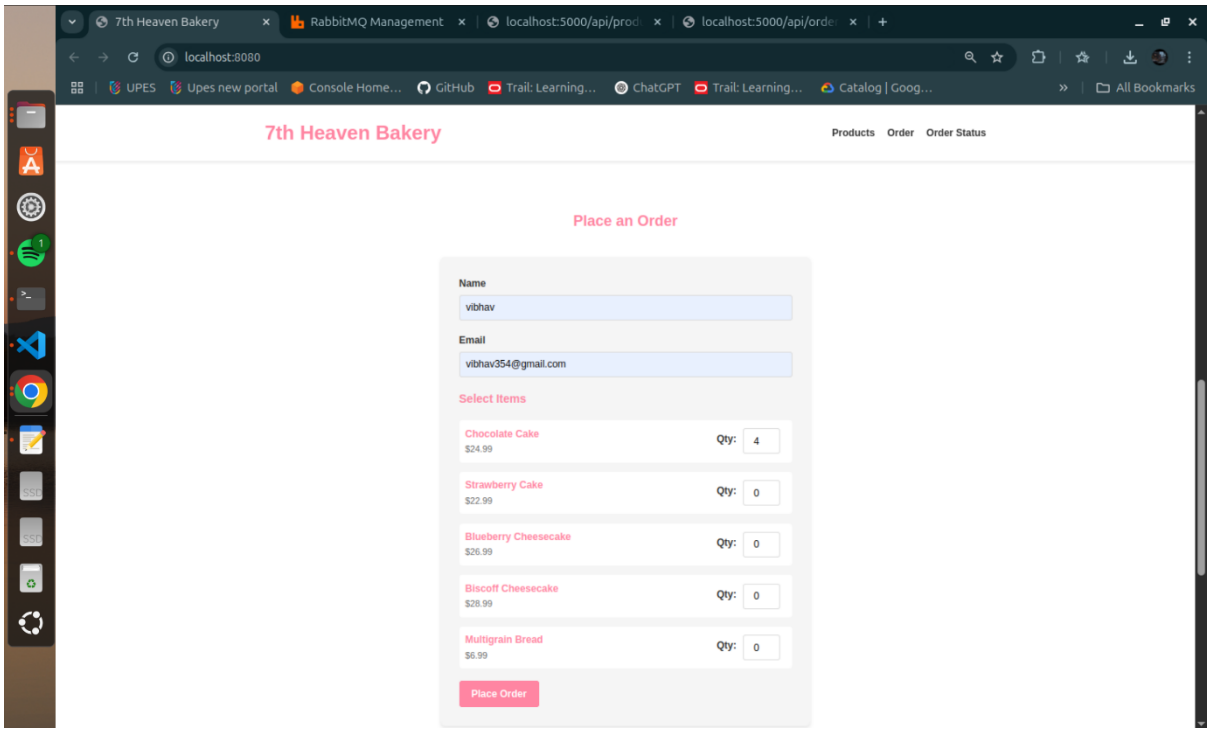
```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT   MEM %     NET I/O           BLOCK I/O         PIDS
279884e85f24   bakery-frontend   0.00%     22.15MiB / 128MiB   17.30%    171kB / 3.06kB    10.3MB / 12.3kB   17
62c00381a44d   bakery-backend    0.02%     39.37MiB / 256MiB   15.38%    354kB / 18.9kB    22.4MB / 102kB    1
5ce4b91e437e   bakery-db         0.00%     37MiB / 512MiB      7.23%     173kB / 8.76kB    21.6MB / 11.3MB   7
3b81a8a3857d   bakery-rabbitmq   8.71%     170.6MiB / 256MiB   66.63%    853kB / 1.38MB    32.8kB / 684kB    107
```

Results:

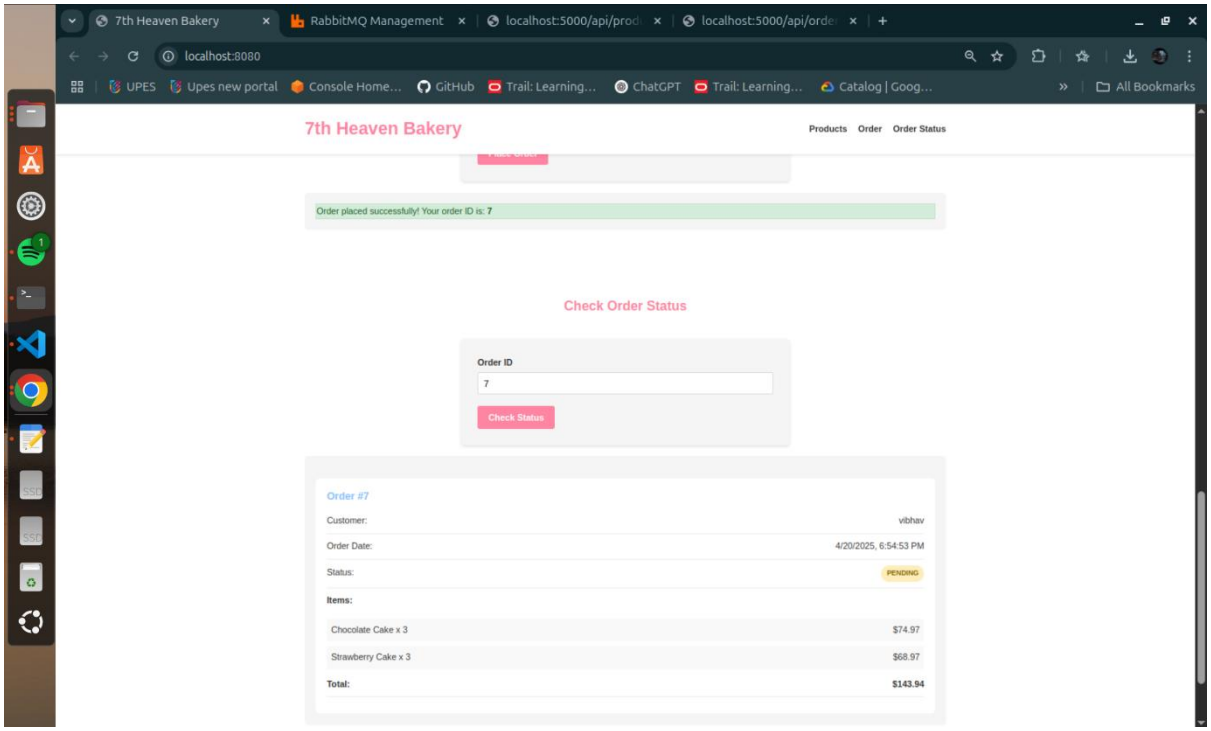
Frontend: With 3 API Endpoints.



List all bakery products

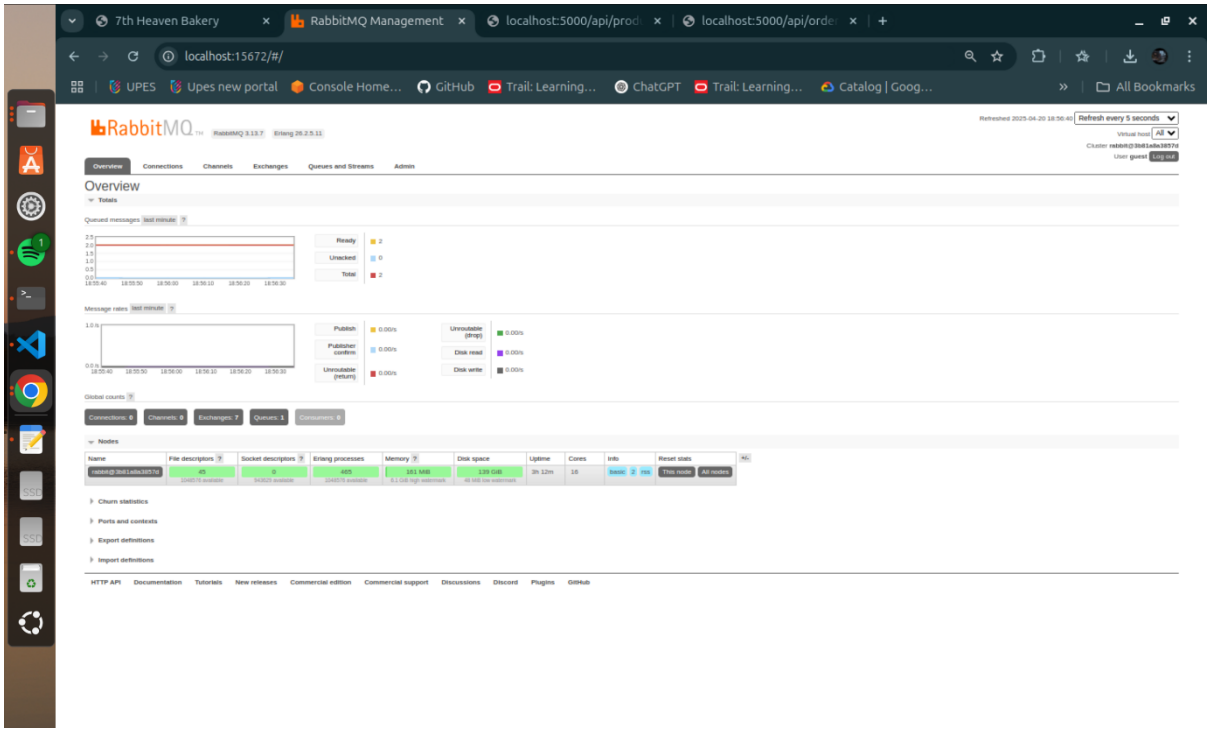


Place an order

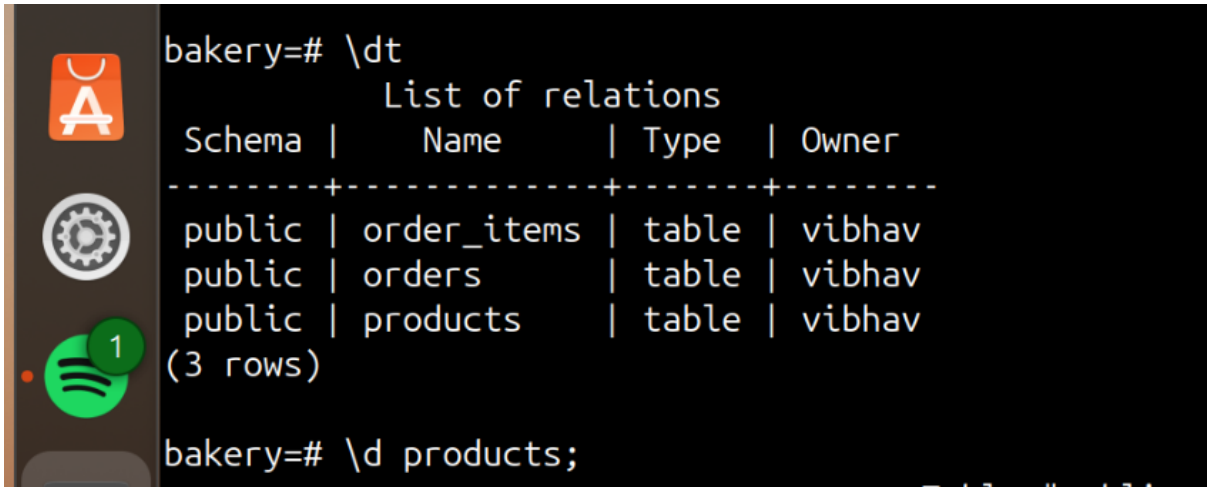


Check order status

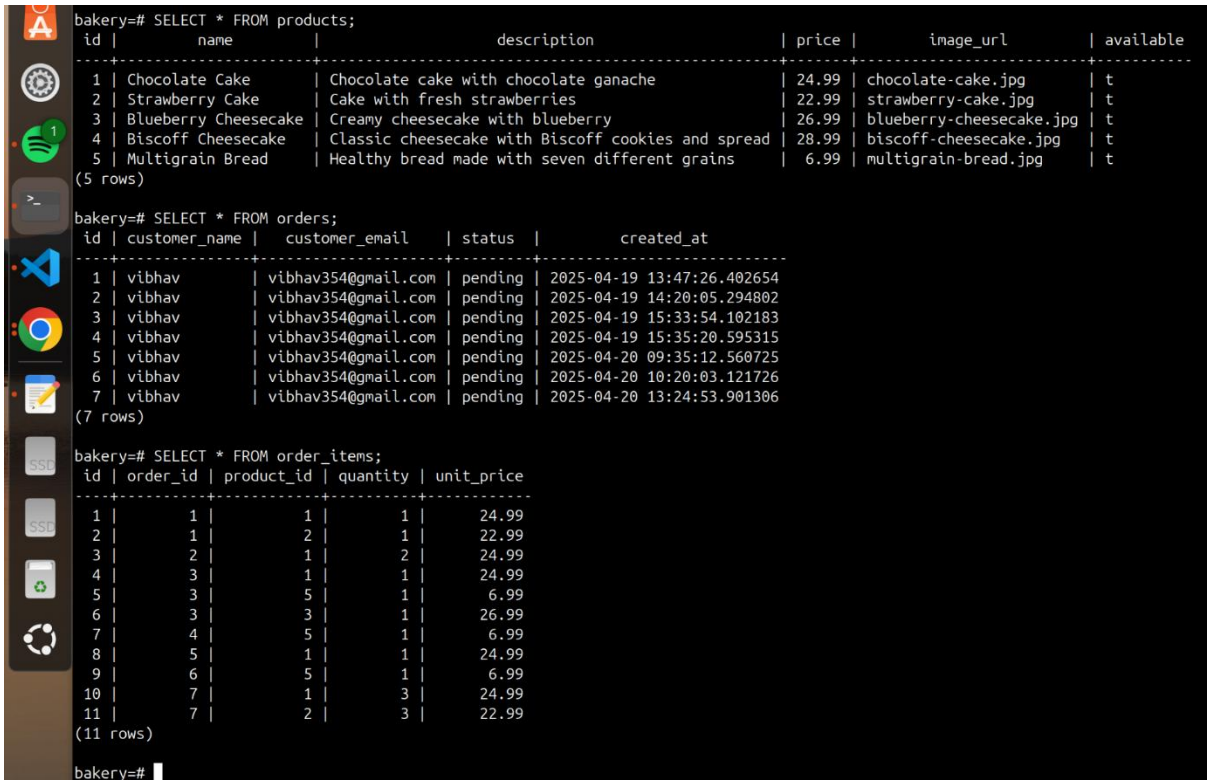
RabbitMQ:



Database(PostgreSQL):



Tables in the bakery database



Data in the 3 tables