**Docker Assignment 1**

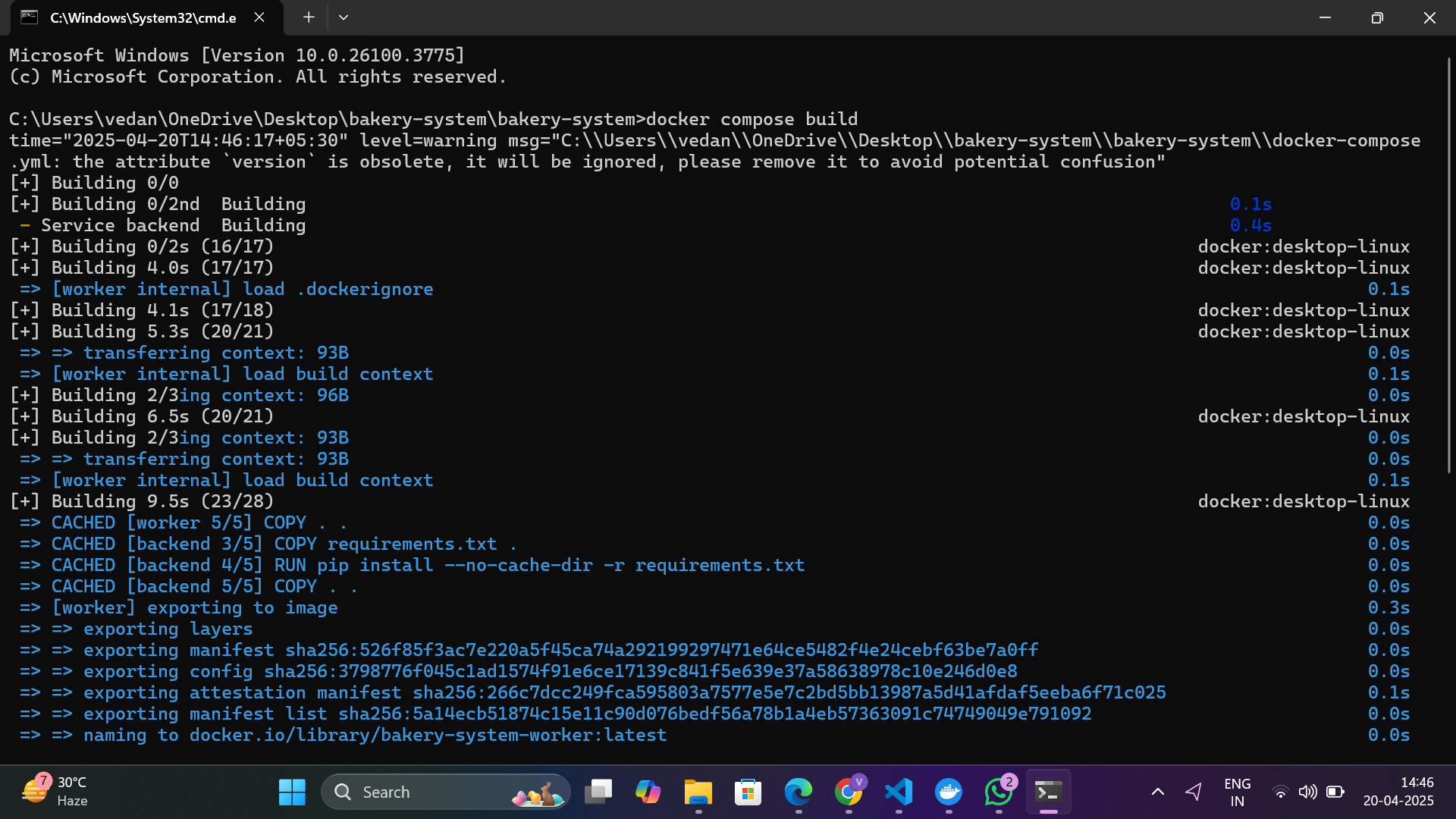
**Name: Vedanti Verma**

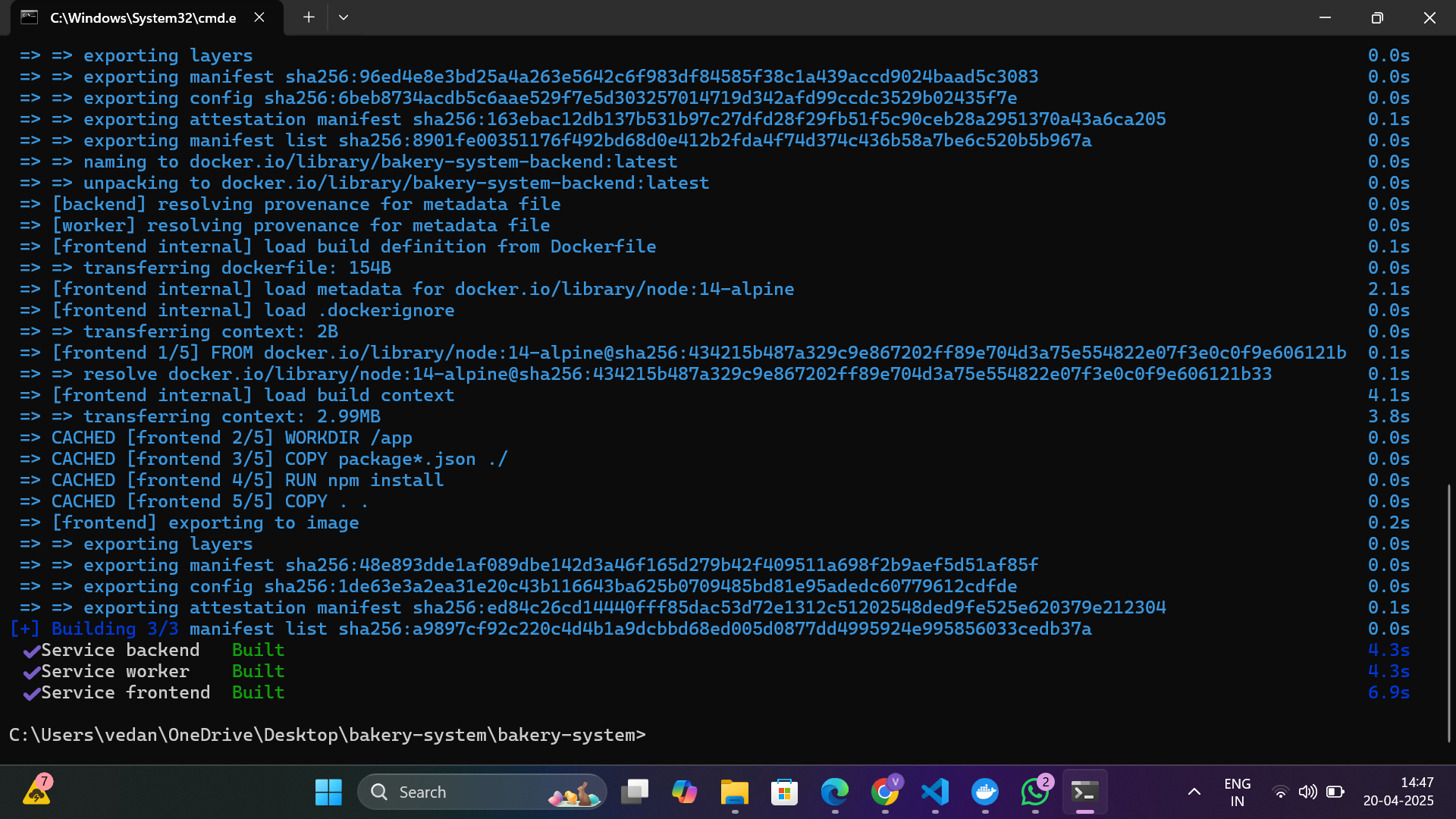
**Sap: 500107991**

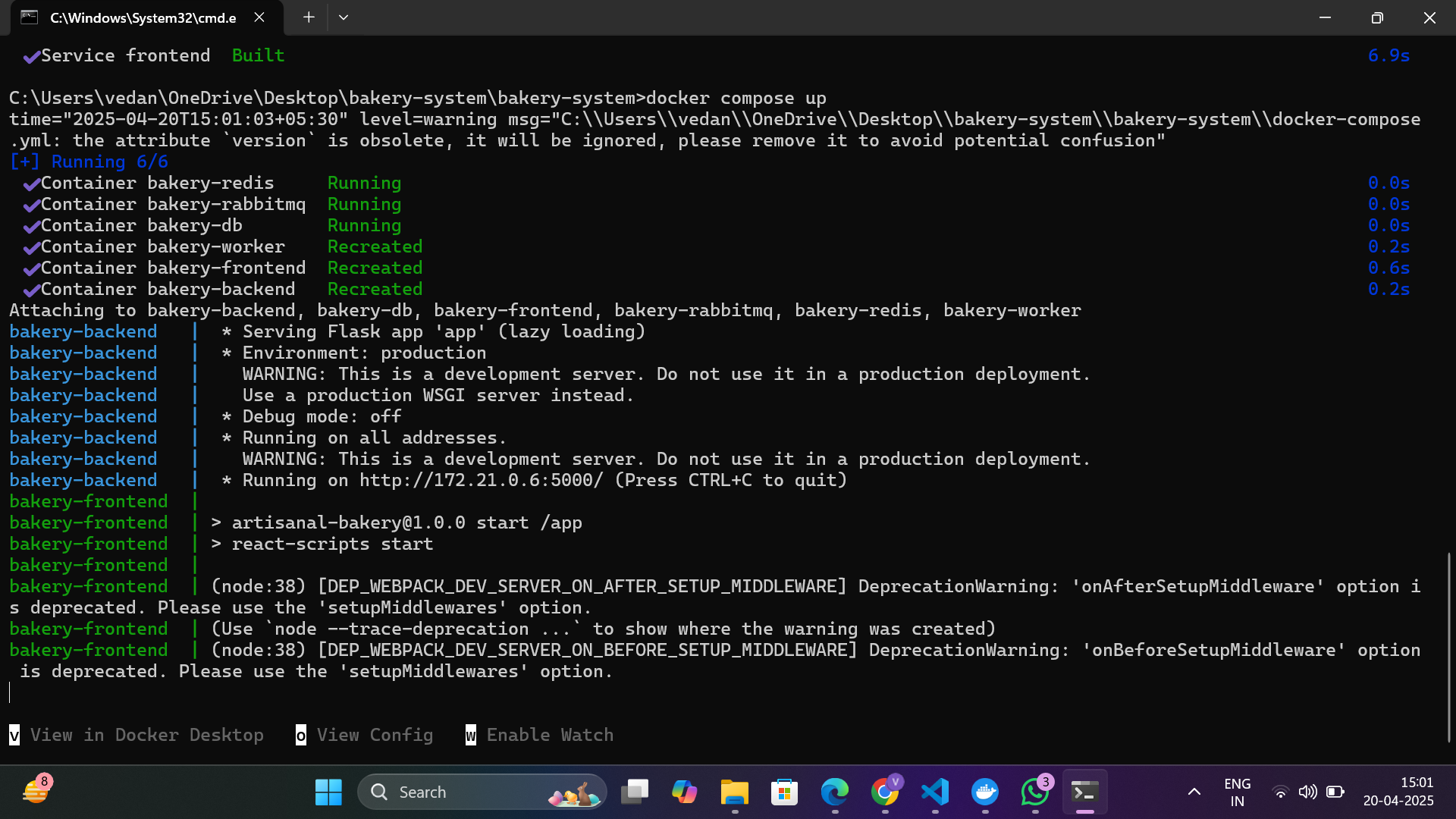
**R2142221248**

**DOCUMENTATION**

This project has been made on Windows. All the screenshots for commands used throughout are given below.







**Database Container:**

CREATE TABLE products (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

description TEXT,

price DECIMAL(10, 2) NOT NULL,

image\_url VARCHAR(255),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE orders (

id SERIAL PRIMARY KEY,

customer\_name VARCHAR(100) NOT NULL,

customer\_email VARCHAR(100) NOT NULL,

status VARCHAR(50) DEFAULT 'pending',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE order\_items (

id SERIAL PRIMARY KEY,

order\_id INTEGER REFERENCES orders(id),

product\_id INTEGER REFERENCES products(id),

quantity INTEGER NOT NULL,

unit\_price DECIMAL(10, 2) NOT NULL

);

-- Insert some sample products

INSERT INTO products (name, description, price, image\_url) VALUES

('Chocolate Cake', 'Rich chocolate layer cake with ganache', 29.99, 'https://www.bbc.co.uk/food/recipes/easy\_chocolate\_cake\_31070'),

('Sourdough Bread', 'Artisanal sourdough bread', 8.99, 'https://cookidoo.co.uk/recipes/recipe/en-GB/r643607'),

('Blueberry Muffin', 'Moist muffin loaded with blueberries', 3.99, 'https://www.wildernesswife.com/blog/2019/08/04/jordan-marsh-blueberry-muffin-recipe/'),

('Croissant', 'Buttery, flaky French pastry', 4.50, 'https://www.istockphoto.com/photos/butter-croissant');

volumes:

postgres\_data:

redis\_data:

rabbitmq\_data:

**Database in compose.yml file:**

postgres:

image: postgres:13

container\_name: bakery-db

environment:

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: postgres

POSTGRES\_DB: bakery

volumes:

- postgres\_data:/var/lib/postgresql/data

- ./database/init:/docker-entrypoint-initdb.d

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U postgres"]

interval: 10s

timeout: 5s

retries: 5

restart: unless-stopped

**docker-compose.yml file:**

version: '3.8'

services:

postgres:

image: postgres:13

container\_name: bakery-db

environment:

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: postgres

POSTGRES\_DB: bakery

volumes:

- postgres\_data:/var/lib/postgresql/data

- ./database/init:/docker-entrypoint-initdb.d

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U postgres"]

interval: 10s

timeout: 5s

retries: 5

restart: unless-stopped

redis:

image: redis:6

container\_name: bakery-redis

ports:

- "6379:6379"

volumes:

- redis\_data:/data

healthcheck:

test: ["CMD", "redis-cli", "ping"]

interval: 10s

timeout: 5s

retries: 5

restart: unless-stopped

rabbitmq:

image: rabbitmq:3-management

container\_name: bakery-rabbitmq

ports:

- "5672:5672"

- "15672:15672"

volumes:

- rabbitmq\_data:/var/lib/rabbitmq

healthcheck:

test: ["CMD", "rabbitmqctl", "status"]

interval: 30s

timeout: 10s

retries: 5

restart: unless-stopped

backend:

build: ./backend

container\_name: bakery-backend

depends\_on:

- postgres

- rabbitmq

- redis

environment:

DB\_HOST: postgres

DB\_NAME: bakery

DB\_USER: postgres

DB\_PASSWORD: postgres

RABBITMQ\_HOST: rabbitmq

REDIS\_HOST: redis

ports:

- "5000:5000"

restart: unless-stopped

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost:5000/health"]

interval: 30s

timeout: 10s

retries: 3

worker:

build: ./worker

container\_name: bakery-worker

depends\_on:

- postgres

- rabbitmq

environment:

DB\_HOST: postgres

DB\_NAME: bakery

DB\_USER: postgres

DB\_PASSWORD: postgres

RABBITMQ\_HOST: rabbitmq

restart: unless-stopped

frontend:

build: ./frontend

container\_name: bakery-frontend

depends\_on:

- backend

environment:

REACT\_APP\_API\_URL: http://localhost:5000

ports:

- "3000:3000"

restart: unless-stopped

volumes:

postgres\_data:

redis\_data:

rabbitmq\_data:

Dockerfile for frontend:

FROM node:14-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]

Dockerfile for Backend:

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]

requirements.txt for backend:

Flask==2.0.1

Werkzeug==2.0.3 # Add this line

Flask-CORS==3.0.10

psycopg2-binary==2.9.1

pika==1.2.0

redis==3.5.3

Dockerfile for Worker.py:

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "worker.py"]

requirements.txt for worker

psycopg2-binary==2.9.1

pika==1.2.0

## **Advanced Features:**

### **1. Redis-Based Caching Layer**

To enhance system responsiveness and reduce database query overhead, a Redis caching mechanism has been integrated into the backend service. This layer acts as a high-speed data store for frequently accessed content, particularly product listings and order summaries.

#### **✅ Key Benefits:**

* **Reduced Load on PostgreSQL**: Caching product data significantly decreases the number of direct database hits.
* **Accelerated Response Times**: Repeated product or order fetch requests are served faster from memory.
* **Smart Invalidation Strategy**: Whenever a new order is placed, corresponding cached data is invalidated or refreshed to maintain consistency and accuracy across views.

#### **🔧 Docker Compose Configuration for Redis:**

The Redis service is orchestrated using Docker Compose, with the following configuration embedded in the docker-compose.yml:

redis:

image: redis:6

container\_name: bakery-redis

ports:

- "6379:6379"

volumes:

- redis\_data:/data

healthcheck:

test: ["CMD", "redis-cli", "ping"]

interval: 10s

timeout: 5s

retries: 5

restart: unless-stopped

* **Volume**: Ensures data persistence in the event of container restarts.
* **Healthcheck**: Regularly pings Redis to verify that it's up and responsive.
* **Auto-Restart**: Keeps the service resilient in case of failures.

Redis caching is initialized and connected via the backend Flask application using redis-py or similar clients.

### **2. Asynchronous Order Handling via Worker Service and RabbitMQ**

To support asynchronous, non-blocking order processing, a **dedicated worker service** has been built. It listens for incoming tasks pushed into RabbitMQ queues and updates order status based on the processing state.

#### **Features of the Worker Service:**

* **Asynchronous Processing**: Orders are handled outside the main request/response cycle, improving throughput and responsiveness.
* **Status Update Pipeline**: The worker communicates with the PostgreSQL database to modify order records based on processing stages (e.g., pending → in progress → completed).
* **Error Handling and Retry Logic**: If an order fails due to any backend exception, the system retries a limited number of times, ensuring robustness.

#### **RabbitMQ Configuration in Docker Compose:**

rabbitmq:

image: rabbitmq:3-management

container\_name: bakery-rabbitmq

ports:

- "5672:5672" # AMQP protocol

- "15672:15672" # Web management interface

volumes:

- rabbitmq\_data:/var/lib/rabbitmq

healthcheck:

test: ["CMD", "rabbitmqctl", "status"]

interval: 30s

timeout: 10s

retries: 5

restart: unless-stopped

* **Web UI Access**: Accessible at [http://localhost:15672](http://localhost:15672/) with default credentials (guest / guest) for monitoring queues.
* **Persistent Queue Data**: Ensures message durability via mounted volumes.
* **Health Monitoring**: Confirms RabbitMQ status before enabling service interactions.

The worker service connects to RabbitMQ via pika or similar AMQP clients, consuming messages and executing business logic accordingly.

SCREENSHOTS:

