Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИМЕНИ ПЕРВОГО ПРЕЗИДЕНТА РОССИИ Б.Н. ЕЛЬЦИНА
(УрФУ имени первого Президента России Б.Н. Ельцина)
Институт радиоэлектроники и информационных технологий — РТФ
Школа профессионального и академического образования

ОСНОВЫ РАБОТЫ С RABBITMQ

Отчет по лабораторной работе №6

по дисциплине «Разработка приложений»

	Дата	Подпись	
Преподаватель:	_____	_____	Кузьмин Д.И.
Студенты:	_____	_____	Пантелеев Е.А.

Группа: РИМ-150950

Екатеринбург

2025

ЦЕЛЬ РАБОТЫ

Освоить основные принципы работы с брокером сообщений RabbitMQ в Python, изучить различные паттерны обмена сообщениями и научиться создавать распределенные приложения.

ХОД РАБОТЫ

Создаем файл docker compose с rabbitmq.

docker-compose.yml

```yaml
version: '3.8'
Run All Services
services:
  Run Service
  app:
    build: .
    ports:
      - "8000:8000"
    depends_on:
      - rabbitmq
    environment:
      - RABBITMQ_URL=amqp://guest:guest@rabbitmq:5672

  Run Service
  consumer:
    build: .
    depends_on:
      - rabbitmq
      - app
    command: uv run python -m app.messaging.consumer
    restart: "on-failure:5"

  Run Service
  rabbitmq:
    image: rabbitmq:management
    ports:
      - "5672:5672"
      - "15672:15672"
    healthcheck:
      test: ["CMD", "rabbitmq-diagnostics", "ping"]
      interval: 5s
      timeout: 10s
      retries: 10
      start_period: 30s
```

Создаем consumer.py, на скрине представлены уже очереди для товаров и заказов.

app > messaging > consumer.py > ...

```python
# app/messaging/consumer.py
import asyncio
import logging

from faststream import FastStream
from faststream.rabbit import RabbitBroker

from app.schemas import OrderMessage, ProductMessage

# Настройка логгирования
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Подключение к RabbitMQ
broker = RabbitBroker("amqp://guest:guest@rabbitmq:5672/")
app = FastStream(broker)


GigaCode: explain | explain step by step | doc | test
@broker.subscriber("order")
async def handle_order(msg: OrderMessage):
    """Обработчик событий создания заказа"""
    try:
        logger.info(
            f"📦 Order #{msg.order_id} received | "
            f"User: {msg.user_id} | "
            f"Status: {msg.status} | "
            f"Total: ${msg.total_amount} | "
            f"Created: {msg.created_at}"
        )

    except Exception as e:
        logger.error(f"❌ Error processing order #{msg.order_id}: {e}")
        # Можно настроить retry или dead letter queue
        raise


GigaCode: explain | explain step by step | doc | test
@broker.subscriber("products")
async def handle_product(msg: ProductMessage):
    logger.info(f"🛒 Product created: {msg.name} (${msg.price})")
```

```python
GigaCode: explain | explain step by step | doc | test
async def main():
    """Точка входа для consumer"""
    logger.info("Order consumer starting...")
    await app.run()


if __name__ == "__main__":
    asyncio.run(main())
```

Создаем producer.py

app > messaging > producer.py > publish_order_created > status

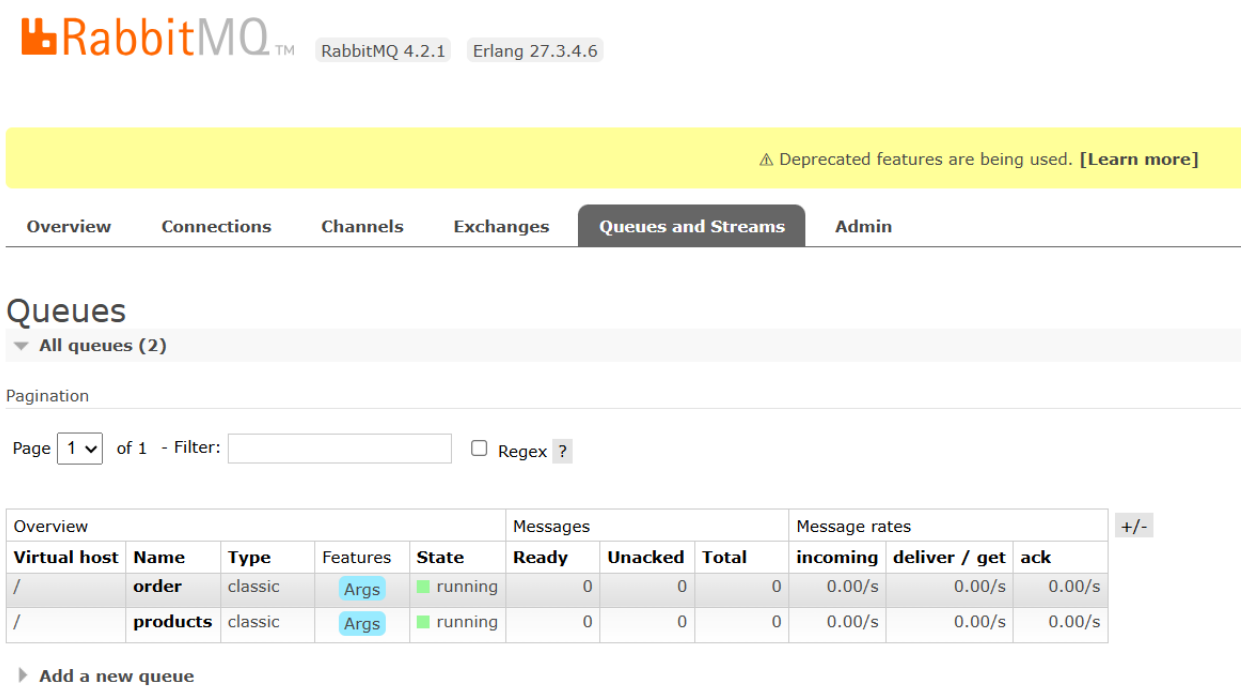
```python
import asyncio
from datetime import datetime
from decimal import Decimal

from faststream.rabbit import RabbitBroker

# Создаём брокер
broker = RabbitBroker("amqp://guest:guest@rabbitmq:5672/")


GigaCode: explain | explain step by step | doc | test
class _BrokerConnection:
    """Менеджер подключения к брокеру"""

    _initialized = False

    @classmethod
    async def ensure_connected(cls):
        """Обеспечивает подключение к брокеру"""
        if not cls._initialized:
            await broker.connect()
            cls._initialized = True


GigaCode: explain | explain step by step | doc | test
async def publish_order_created(
    order_id: int, user_id: int, total_amount: Decimal, status: str = "created"
) -> None:
    """Отправляет событие создания заказа в RabbitMQ"""
    # Подключаемся если ещё не подключены
    await _BrokerConnection.ensure_connected()
    message = {
        "order_id": order_id,
        "user_id": user_id,
        "status": status,
        "total_amount": str(total_amount),
        "created_at": datetime.utcnow().isoformat(),
    }

    await broker.publish(message, queue="order")
```

```python
GigaCode: explain | explain step by step | doc | test
async def publish_product_created(product_id: int, name: str, price: Decimal) -> None:
    message = {
        "product_id": product_id,
        "name": name,
        "price": str(price),
        "created_at": datetime.utcnow().isoformat(),
    }
    await broker.publish(message, queue="products")
```

Проверим работу RabbitMQ и его UI. Попробуем создать заказ и посмотрим на то что происходит с очередью. Пока что реализовано логирование.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>curl -X POST "http://localhost:8000/orders" -H "Content-Type: applicati
on/json" -d "{\"user_id\": 1, \"address_id\": 1, \"items\": [{\"product_id\": 1, \"quantity\": 2}]}"
{"id":4,"user_id":1,"address_id":1,"status":"pending","total_amount":"290000.00","created_at":"2025-12-12T10:42:50.903326","u
pdated_at":"2025-12-12T10:42:50.911067","items":[{"id":4,"product_id":1,"quantity":2,"price_at_purchase":"145000.00","total_p
rice":"290000.00"}]}
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>docker-compose logs app --tail=50
```

```
app-1  | INFO - 2025-12-12 10:42:50,926 - sqlalchemy.engine.Engine - base - [generated in 0.00055s] (4,)
app-1  | INFO - 2025-12-12 10:42:50,927 - app.services.order_service - order_service - 📤 Sending RabbitMQ event for order 4
app-1  | 2025-12-12 10:42:50,949 INFO sqlalchemy.engine.Engine SELECT order_items.id AS order_items_id, order_items.order_id
AS order_items_order_id, order_items.product_id AS order_items_product_id, order_items.quantity AS order_items_quantity, orde
r_items.price_at_purchase AS order_items_price_at_purchase, order_items.total_price AS order_items_total_price
app-1  | FROM order_items
app-1  | WHERE ? = order_items.order_id
app-1  | INFO - 2025-12-12 10:42:50,945 - app.services.order_service - order_service - ✅ RabbitMQ event sent for order 4
 pp-1  | 2025-12-12 10:42:50,950 INFO sqlalchemy.engine.Engine [generated in 0.00079s] (4,)
app-1  | 2025-12-12 10:42:50,950 INFO sqlalchemy.engine.Engine [generated in 0.00079s] (4,)
app-1  | INFO - 2025-12-12 10:42:50,949 - sqlalchemy.engine.Engine - base - SELECT order_items.id AS order_items_id, order_it
ems.order_id AS order_items_order_id, order_items.product_id AS order_items_product_id, order_items.quantity AS order_items_q
uantity, order_items.price_at_purchase AS order_items_price_at_purchase, order_items.total_price AS order_items_total_price
app-1  | FROM order_items
app-1  | WHERE ? = order_items.order_id
app-1  | INFO - 2025-12-12 10:42:50,950 - sqlalchemy.engine.Engine - base - [generated in 0.00079s] (4,)
app-1  | 2025-12-12 10:42:50,953 INFO sqlalchemy.engine.Engine ROLLBACK
app-1  | INFO - 2025-12-12 10:42:50,953 - sqlalchemy.engine.Engine - base - ROLLBACK
app-1  | INFO:     172.18.0.1:58270 - "POST /orders HTTP/1.1" 201 Created
```



Реализуем тест приложения с помощью скрипта seed_rabbit.py.

app > messaging > seed_rabbit.py > send_to_queue

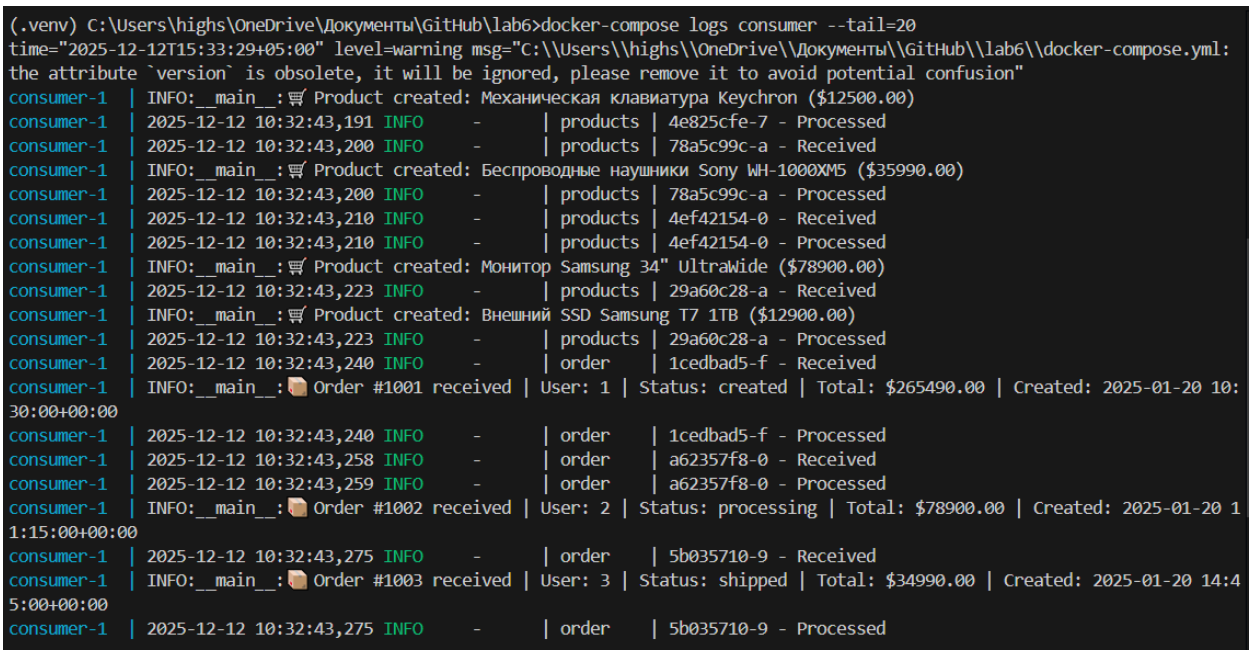
```python
import json
import pika


GigaCode: explain | explain step by step | doc | test
def send_to_queue(queue: str, data: dict):
    connection = pika.BlockingConnection(
        pika.ConnectionParameters(
            host="rabbitmq",  # или 'localhost' не в Docker
            port=5672,
            credentials=pika.PlainCredentials("guest", "guest"),
        )
    )
    channel = connection.channel()
    channel.queue_declare(queue=queue)
    channel.basic_publish(
        exchange="",
        routing_key=queue,
        body=json.dumps(data),
        properties=pika.BasicProperties(delivery_mode=2),
    )
    connection.close()
    print(f"📤 Sent to {queue}: {data['name'] if 'name' in data else 'order'}")
```

```python
products = [
    {
        "product_id": 101,
        "name": "Игровой ноутбук ASUS ROG",
        "price": "199999.00",
        "created_at": "2025-01-15T14:30:00Z",
    },
    {
        "product_id": 102,
        "name": "Механическая клавиатура Keychron",
        "price": "12500.00",
        "created_at": "2025-01-16T10:15:00Z",
    },
    {
        "product_id": 103,
        "name": "Беспроводные наушники Sony WH-1000XM5",
        "price": "35990.00",
        "created_at": "2025-01-17T16:45:00Z",
    },
    {
        "product_id": 104,
        "name": 'Монитор Samsung 34" UltraWide',
        "price": "78900.00",
        "created_at": "2025-01-18T12:20:00Z",
    },
    {
        "product_id": 105,
        "name": "Внешний SSD Samsung T7 1TB",
        "price": "12900.00",
        "created_at": "2025-01-19T09:30:00Z",
    },
]
```

```python
orders = [
    {
        "order_id": 1001,
        "user_id": 1,
        "status": "created",
        "total_amount": "265490.00",
        "created_at": "2025-01-20T10:30:00Z",
    },
    {
        "order_id": 1002,
        "user_id": 2,
        "status": "processing",
        "total_amount": "78900.00",  # Монитор
        "created_at": "2025-01-20T11:15:00Z",
    },
    {
        "order_id": 1003,
        "user_id": 3,
        "status": "shipped",
        "total_amount": "34990.00",  # Наушники
        "created_at": "2025-01-20T14:45:00Z",
    },
]

if __name__ == "__main__":
    print("🚀 Sending test data to RabbitMQ...")

    for product in products:
        send_to_queue(
            "products", product
        )  # очередь 'products' (во множественном числе)

    for order in orders:
        send_to_queue("order", order)  # очередь 'order'

    print("✅ Test data sent successfully")

```

Проверим его работу, запустим его в докере.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>docker-compose exec app uv run python -m app.messaging.seed_rabbit
time="2025-12-12T15:32:42+05:00" level=warning msg="C:\\Users\\highs\\OneDrive\\Документы\\GitHub\\lab6\\docker-compose.yml:
the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
🚀 Sending test data to RabbitMQ...
📤 Sent to products: Игровой ноутбук ASUS ROG
📤 Sent to products: Механическая клавиатура Keychron
📤 Sent to products: Беспроводные наушники Sony WH-1000XM5
📤 Sent to products: Монитор Samsung 34" UltraWide
📤 Sent to products: Внешний SSD Samsung T7 1TB
📤 Sent to order: order
📤 Sent to order: order
📤 Sent to order: order
✅ Test data sent successfully
```



Ответы на вопросы

1. Что такое AMQP? Каковы его основные преимущества?

AMQP (Advanced Message Queuing Protocol) – это открытый стандартный протокол для обмена сообщениями между приложениями. Его преимущества: плюсы: гарантированная доставка, кроссплатформенность, гибкая маршрутизация и встроенная поддержка транзакций.

2. В чем разница между очередями сообщений и шинами сообщений?

Очереди – точка-точка, одно сообщение – один потребитель.

Шины – publish-subscribe, одно сообщение – много подписчиков через exchange.

3. Как обеспечить надежную доставку сообщений в RabbitMQ?

Подтверждение от потребителя (ack) – сообщение удаляется из очереди только после успешной обработки.

Persistent messages – сообщения сохраняются на диск.

Подтверждение публикации – продюсер знает, что брокер принял сообщение. Без этого сообщения могут теряться при сбоях.

4. Что произойдет с сообщением, если consumer упадет во время обработки?

RabbitMQ вернет сообщение в очередь (если не было ack) и отправит другому consumer'у или тому же после перезапуска.

5. Как обеспечить сохранность сообщений при перезапуске RabbitMQ?

Durable queues и persistent messages. Сообщения пишутся на диск и переживают перезапуск брокера.

6. Что такое TTL (Time To Live) для сообщений и как его настроить?

TTL – время жизни сообщения. Настраивается на уровне очереди (x-message-ttl) или сообщения (expiration).

ЗАКЛЮЧЕНИЕ

В ходе работы были освоены основные принципы работы с брокером сообщений RabbitMQ в Python и изучены различные паттерны обмена сообщениями.