

Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Уральский федеральный университет имени первого Президента России Б.Н.  
Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

**Лабораторная работа №6  
Основы работы с RabbitMQ**

Студент группы РИМ – 150950: \_\_\_\_\_ Вальнева А.Д.

## Цель работы

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

## Задачи

### 1) Настройка инфраструктуры RabbitMQ

- Добавить сервис RabbitMQ в Docker Compose с образом rabbitmq:3-management
- Настроить порты для AMQP протокола (5672) и веб-интерфейса управления (15672)
- Обеспечить персистентность данных через volume и настроить healthcheck
- Интегрировать RabbitMQ с существующим приложением через переменные окружения

### 2) Реализация обработчиков очередей

- Создать подписчики для очередей order и product с использованием FastStream
- Реализовать обработку операций создания, обновления продукции и пометки товаров как недоступных
- Реализовать обработку создания заказов с валидацией остатков, обновления статусов и возврата товаров при отмене
- Обеспечить транзакционную обработку сообщений с логированием и обработкой ошибок

### 3) Интеграция брокера с жизненным циклом приложения

- Встроить запуск и остановку RabbitBroker в lifespan контекст Litestar
- Реализовать retry логику с автоматическими повторными попытками подключения к брокеру
- Обеспечить graceful shutdown с корректным закрытием соединений при остановке приложения

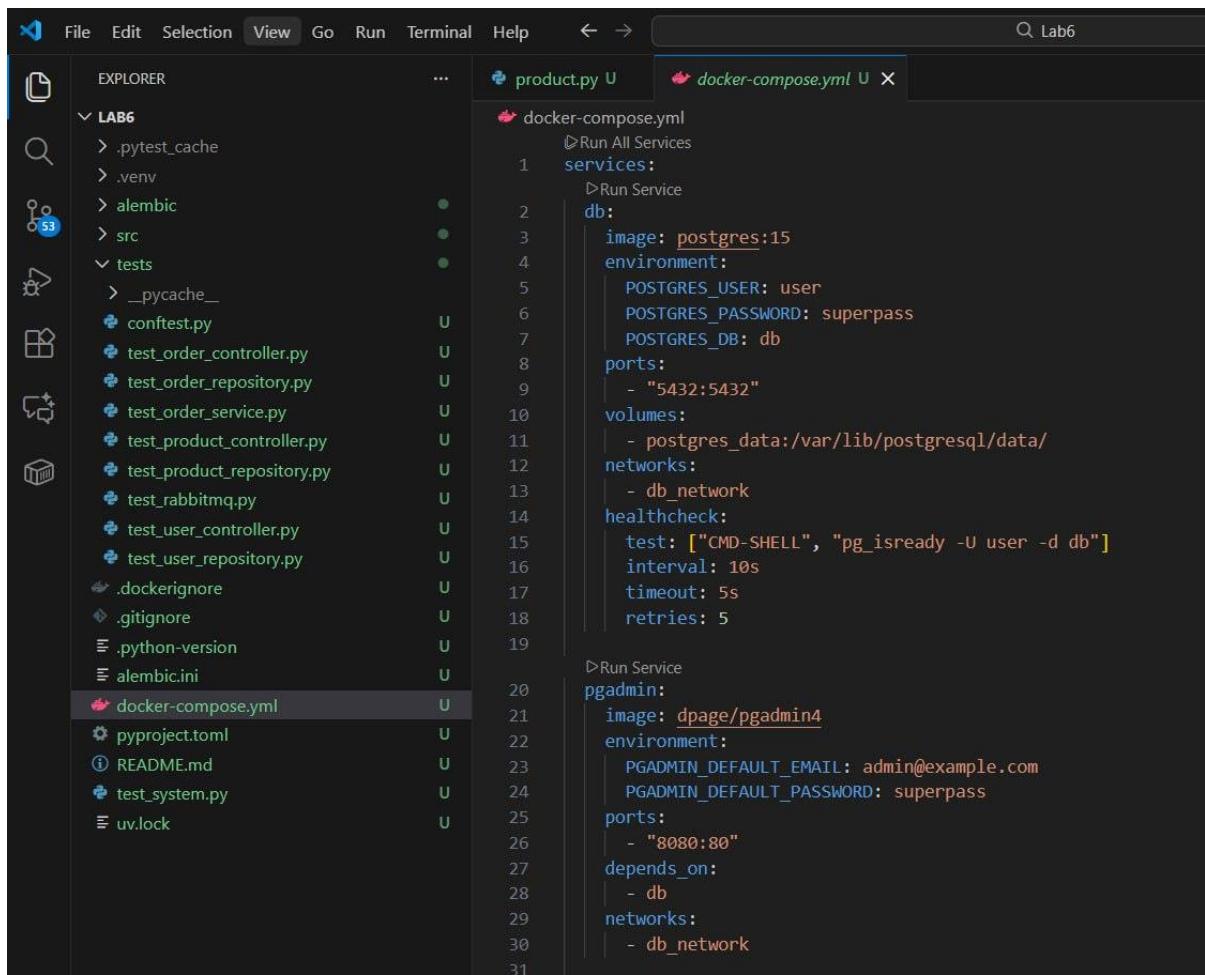
- Разделить ответственность: команды через RabbitMQ, запросы через REST API

#### 4) Разработка продюсера и тестирование системы

- Создать скрипт-продюсер с использованием библиотеки aio-pika для отправки сообщений
- Реализовать создание 5 тестовых продуктов и 3 заказов с различными параметрами
- Протестировать все операции системы: создание, обновление, изменение статусов, пометку товаров
- Проверить обработку граничных случаев (недостаточное количество товара, отмена заказа)

## Часть 1: Настройка контейнера с RabbitMQ

Файл: *docker-compose.yml*



```

File Edit Selection View Go Run Terminal Help < > Q Lab6

EXPLORER
LAB6
> .pytest_cache
> .venv
> alembic
> src
tests
> __pycache__
conf.py
test_order_controller.py
test_order_repository.py
test_order_service.py
test_product_controller.py
test_product_repository.py
test_rabbitmq.py
test_user_controller.py
test_user_repository.py
.dockerignore
.gitignore
.python-version
alembic.ini
docker-compose.yml
pyproject.toml
README.md
test_system.py
uv.lock

product.py U docker-compose.yml U

docker-compose.yml
Run All Services
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: superpass
      POSTGRES_DB: db
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    networks:
      - db_network
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U user -d db"]
      interval: 10s
      timeout: 5s
      retries: 5
  pgadmin:
    image: dpage/pgadmin4
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@example.com
      PGADMIN_DEFAULT_PASSWORD: superpass
    ports:
      - "8080:80"
    depends_on:
      - db
    networks:
      - db_network

```

The screenshot shows the VS Code interface with the 'docker-compose.yml' file open in the editor tab. The file defines two services: 'rabbitmq' and 'app'. The 'rabbitmq' service uses the 'rabbitmq:3-management' image, maps ports 5672 and 15672, and has environment variables for RABBITMQ\_DEFAULT\_VHOST, RABBITMQ\_DEFAULT\_USER, and RABBITMQ\_DEFAULT\_PASS. It also specifies volumes for rabbitmq\_data and networks for db\_network. The 'app' service is built from the current directory, uses a Dockerfile, and has environment variables for DATABASE\_URL, RABBITMQ\_URL, and PYTHONPATH. It maps port 8000 and depends on the 'db' service.

```
version: '3.1'
services:
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      - RABBITMQ_DEFAULT_VHOST=/
      - RABBITMQ_DEFAULT_USER=guest
      - RABBITMQ_DEFAULT_PASS=guest
    volumes:
      - rabbitmq_data:/var/lib/rabbitmq
    networks:
      - db_network
    healthcheck:
      test: ["CMD", "rabbitmq-diagnostics", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 40s
  app:
    build:
      context: .
      dockerfile: src/Dockerfile
    environment:
      DATABASE_URL: postgresql+asyncpg://user:superpass@db:5432/db
      RABBITMQ_URL: amqp://guest:guest@rabbitmq:5672/
      PYTHONPATH: /app
    ports:
      - "8000:8000"
    depends_on:
      db:
```

This screenshot shows the same VS Code interface, but the 'healthcheck' section of the 'rabbitmq' service is currently selected or expanded. The configuration includes a timeout of 5 seconds, 5 retries, and a start period of 40 seconds.

```
version: '3.1'
services:
  rabbitmq:
    healthcheck:
      timeout: 5s
      retries: 5
      start_period: 40s
```

**Ход выполнения:**

Файл содержит конфигурацию сервисов

- PostgreSQL
- RabbitMQ с Management UI
- app

Был добавлен сервис rabbitmq с образом rabbitmq:3-management:

- Порты: 5672 (AMQP протокол), 15672 (веб-интерфейс управления)
- Переменные окружения для дефолтного vhost и учетных данных (guest/guest)
- Volume rabbitmq\_data для персистентности данных
- Healthcheck с использованием rabbitmq-diagnostics ping для проверки готовности сервиса

Сервис app обновлен:

- Добавлена переменная RABBITMQ\_URL для подключения к брокеру
- Зависимость от rabbitmq с условием service\_healthy — приложение стартует только после готовности RabbitMQ

По адресу <http://localhost:15672> проверяем работу RabbitMQ

The screenshot shows the RabbitMQ Management UI at <http://localhost:15672/queues>. The top navigation bar includes links for Overview, Connections, Channels, Exchanges, Queues and Streams (which is selected), and Admin. The top right corner shows the refresh interval as "Refresh every 5 seconds", the virtual host as "All", the cluster as "rabbit@d9963ff22931", and the user as "guest". The main content area is titled "Queues" and shows a table with two rows. The columns are: Virtual host, Name, Type, Features, State, Ready, Unacked, Total, incoming, deliver / get, and ack. The "order" queue is under the "/" virtual host and the "product" queue is under the "/" virtual host. Both queues are listed as "running". The "incoming" and "deliver / get" columns show 0 for both. The "ack" column shows 0 for both. At the bottom of the table, there is a link to "Add a new queue". The footer of the page includes links for HTTP API, Documentation, Tutorials, New releases, Commercial edition, Commercial support, Discussions, Discord, Plugins, and GitHub.

Файл: *pyproject.toml*

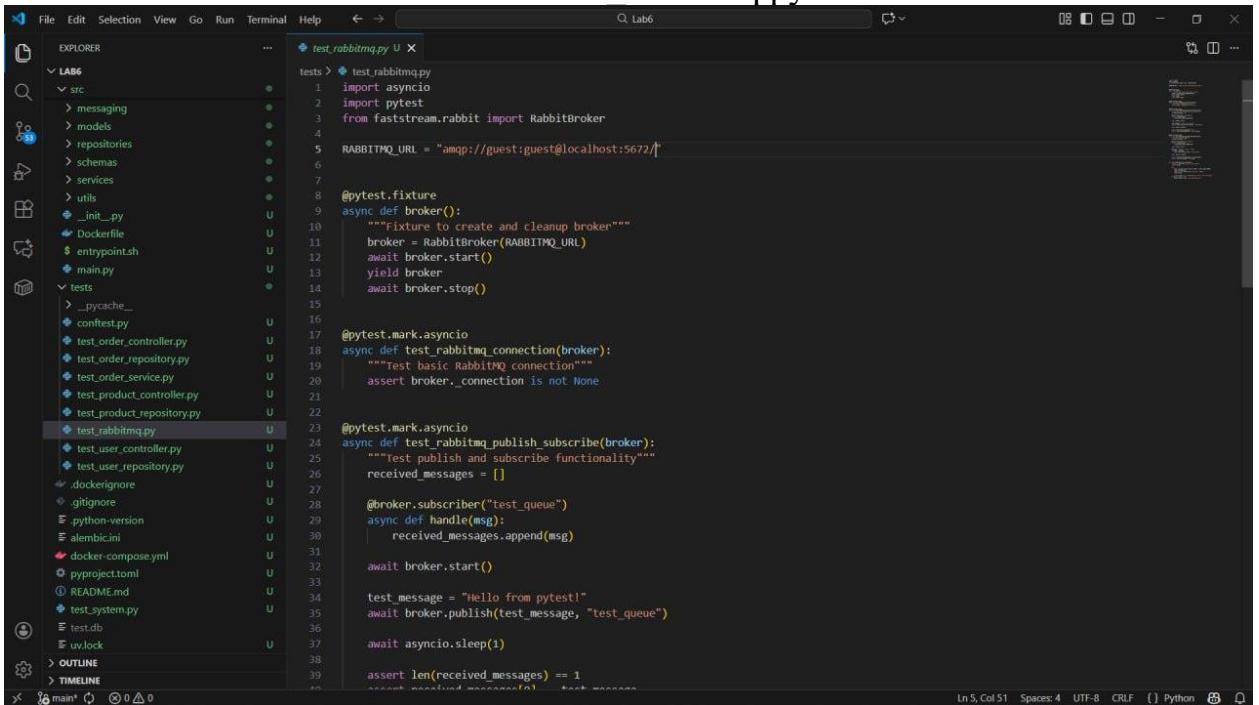
```
pyproject.toml
1 [project]
2 name = "lab6"
3 version = "0.1.0"
4 description = "Add your description here"
5 readme = "README.md"
6 requires-python = ">=3.13"
7 dependencies = [
8     "aio-pika>=9.4.3",
9     "aiosqlite>0.21.0",
10    "alembic>=1.17.2",
11    "asyncpg>0.30.0",
12    "black>=25.11.0",
13    "fast-depends>=3.0.5",
14    "faststream[rabbit]>=0.1.4",
15    "isort>=7.0.0",
16    "litestar[standard]>=2.18.0",
17    "polyfactory>=3.0.0",
18    "pre-commit>=4.5.0",
19    "pydantic[email]>=2.12.4",
20    "pylint>=4.0.3",
21    "pytest>=9.0.1",
22    "pytest-asyncio>=1.3.0",
23    "sniffio>1.3.1",
24    "sqlalchemy[asyncio]>=2.0.44",
25    "uvicorn[standard]>=0.38.0",
26 ]
27
```

Установлены библиотеки для работы с RabbitMQ:

- faststream[rabbit] — фреймворк для асинхронной обработки сообщений
- aio-pika — асинхронный клиент AMQP для отправки сообщений из продюсера

## Часть 2: Подключение к очередям

Файл: test\_rabbitmq.py



```
test_rabbitmq.py U
tests > test_rabbitmq.py
1 import asyncio
2 import pytest
3 from faststream.rabbit import RabbitBroker
4
5 RABBITMQ_URL = "amqp://guest:guest@localhost:5672/"
6
7 @pytest.fixture
8 async def broker():
9     """Fixture to create and cleanup broker"""
10    broker = RabbitBroker(RABBITMQ_URL)
11    await broker.start()
12    yield broker
13    await broker.stop()
14
15
16
17 @pytest.mark.asyncio
18 async def test_rabbitmq_connection(broker):
19     """Test basic RabbitMQ connection"""
20     assert broker._connection is not None
21
22
23 @pytest.mark.asyncio
24 async def test_rabbitmq_publish_subscribe(broker):
25     """Test publish and subscribe functionality"""
26     received_messages = []
27
28     @broker.subscriber("test_queue")
29     async def handle(msg):
30         received_messages.append(msg)
31
32     await broker.start()
33
34     test_message = "Hello from pytest!"
35     await broker.publish(test_message, "test_queue")
36
37     await asyncio.sleep(1)
38
39     assert len(received_messages) == 1
```

```

File Edit Selection View Go Run Terminal Help ⏪ ⏴ Q Lab6
EXPLORER LAB6 src
messaging models repositories schemas services utils __init__.py Dockerfile $ entrypoint.sh main.py tests
__pycache__ contest.py test_order_controller.py test_order_repository.py test_order_service.py test_product_controller.py test_product_repository.py test_rabbitmq.py test_user_controller.py test_user_repository.py .dockerrcignore .gitignore .python-version alembic.ini docker-compose.yml pyproject.toml README.md test_system.py test_db uv.lock
OUTLINE TIMELINE
File Edit Selection View Go Run Terminal Help ⏪ ⏴ Q Lab6
EXPLORER LAB6 src
messaging models repositories schemas services utils __init__.py Dockerfile $ entrypoint.sh main.py tests
__pycache__ contest.py test_order_controller.py test_order_repository.py test_order_service.py test_product_controller.py test_product_repository.py test_rabbitmq.py test_user_controller.py test_user_repository.py .dockerrcignore .gitignore .python-version alembic.ini docker-compose.yml pyproject.toml README.md test_system.py test_db uv.lock
OUTLINE TIMELINE
Ln 5, Col 51 Spaces:4 UTF-8 CRLF Python
Ln 5, Col 51 Spaces:4 UTF-8 CRLF Python

```

```

# test_rabbitmq.py

tests > test_rabbitmq.publish_subscribe(broker):
    24     async def test_rabbitmq_publish_subscribe(broker):
    25         assert len(received_messages) == 1
    26         assert received_messages[0] == test_message
    27
    28     @pytest.mark.asyncio
    29     async def test_rabbitmq_multiple_messages(broker):
    30         """Test multiple messages"""
    31         received_messages = []
    32
    33         @broker.subscriber("multi_queue")
    34         async def handle(msg):
    35             received_messages.append(msg)
    36
    37         await broker.start()
    38
    39         messages = ["msg1", "msg2", "msg3"]
    40         for msg in messages:
    41             await broker.publish(msg, "multi_queue")
    42
    43         await asyncio.sleep(2)
    44
    45         assert len(received_messages) == len(messages)
    46         assert received_messages == messages
    47
    48     def test_rabbitmq_service_running():
    49         """Test if RabbitMQ service is accessible"""
    50         import socket
    51
    52         try:
    53             sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    54             sock.settimeout(2)
    55             result = sock.connect_ex(('localhost', 5672))
    56             sock.close()
    57
    58             assert result == 0, "RabbitMQ port 5672 is not accessible"
    59         except Exception as e:
    60             pytest.fail(f"Cannot reach RabbitMQ: {e}")
    61
    62
    63     def test_rabbitmq_multiple_messages(broker):
    64         """Test multiple messages"""
    65         received_messages = []
    66
    67         @broker.subscriber("multi_queue")
    68         async def handle(msg):
    69             received_messages.append(msg)
    70
    71         await broker.start()
    72
    73         messages = ["msg1", "msg2", "msg3"]
    74         for msg in messages:
    75             await broker.publish(msg, "multi_queue")
    76
    77         await asyncio.sleep(2)
    78
    79         assert len(received_messages) == len(messages)
    80         assert received_messages == messages
    81
    82     def test_rabbitmq_service_running():
    83         """Test if RabbitMQ service is accessible"""
    84         import socket
    85
    86         try:
    87             sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    88             sock.settimeout(2)
    89             result = sock.connect_ex(('localhost', 5672))
    90             sock.close()
    91
    92             assert result == 0, "RabbitMQ port 5672 is not accessible"
    93         except Exception as e:
    94             pytest.fail(f"Cannot reach RabbitMQ: {e}")
    95
    96
    97     def test_rabbitmq_connection(broker):
    98         """Test connection to RabbitMQ broker"""
    99         assert broker.is_connected() == True
    100
    101
    102     def test_rabbitmq_publish(broker):
    103         """Test publishing message to RabbitMQ broker"""
    104         broker.publish("test_message", "queue")
    105
    106         assert len(received_messages) == 1
    107         assert received_messages[0] == "test_message"
    108
    109
    110     def test_rabbitmq_subscribe(broker):
    111         """Test subscribing to RabbitMQ broker"""
    112         received_messages = []
    113
    114         @broker.subscriber("queue")
    115         async def handle(msg):
    116             received_messages.append(msg)
    117
    118         await broker.start()
    119
    120         broker.publish("test_message", "queue")
    121
    122         await asyncio.sleep(2)
    123
    124         assert len(received_messages) == 1
    125         assert received_messages[0] == "test_message"
    126
    127
    128     def test_rabbitmq_multiple_subscribers(broker):
    129         """Test multiple subscribers to RabbitMQ broker"""
    130         received_messages = []
    131
    132         @broker.subscriber("queue")
    133         async def handle1(msg):
    134             received_messages.append(msg)
    135
    136         @broker.subscriber("queue")
    137         async def handle2(msg):
    138             received_messages.append(msg)
    139
    140         await broker.start()
    141
    142         broker.publish("test_message", "queue")
    143
    144         await asyncio.sleep(2)
    145
    146         assert len(received_messages) == 2
    147         assert received_messages[0] == "test_message"
    148         assert received_messages[1] == "test_message"
    149
    150
    151     def test_rabbitmq_error_handling(broker):
    152         """Test error handling for RabbitMQ broker"""
    153         received_messages = []
    154
    155         @broker.subscriber("queue")
    156         async def handle_error(msg):
    157             raise ValueError("Test error")
    158
    159         await broker.start()
    160
    161         broker.publish("test_message", "queue")
    162
    163         await asyncio.sleep(2)
    164
    165         assert len(received_messages) == 0
    166
    167         with pytest.raises(ValueError):
    168             received_messages[0]
    169
    170
    171     def test_rabbitmq_connection_error(broker):
    172         """Test connection error for RabbitMQ broker"""
    173         received_messages = []
    174
    175         @broker.subscriber("queue")
    176         async def handle_error(msg):
    177             raise ConnectionError("Test connection error")
    178
    179         await broker.start()
    180
    181         broker.publish("test_message", "queue")
    182
    183         await asyncio.sleep(2)
    184
    185         assert len(received_messages) == 0
    186
    187         with pytest.raises(ConnectionError):
    188             received_messages[0]
    189
    190
    191     def test_rabbitmq_multiple_queues(broker):
    192         """Test multiple queues for RabbitMQ broker"""
    193         received_messages = []
    194
    195         @broker.subscriber("queue1")
    196         async def handle1(msg):
    197             received_messages.append(msg)
    198
    199         @broker.subscriber("queue2")
    200         async def handle2(msg):
    201             received_messages.append(msg)
    202
    203         await broker.start()
    204
    205         broker.publish("test_message", "queue1")
    206
    207         await asyncio.sleep(2)
    208
    209         assert len(received_messages) == 1
    210         assert received_messages[0] == "test_message"
    211
    212         broker.publish("test_message", "queue2")
    213
    214         await asyncio.sleep(2)
    215
    216         assert len(received_messages) == 2
    217         assert received_messages[0] == "test_message"
    218         assert received_messages[1] == "test_message"
    219
    220
    221     def test_rabbitmq_qos(broker):
    222         """Test QoS for RabbitMQ broker"""
    223         received_messages = []
    224
    225         @broker.subscriber("queue")
    226         async def handle(msg):
    227             received_messages.append(msg)
    228
    229             if len(received_messages) == 2:
    230                 broker.stop()
    231
    232         await broker.start()
    233
    234         broker.publish("test_message", "queue")
    235
    236         await asyncio.sleep(2)
    237
    238         assert len(received_messages) == 2
    239         assert received_messages[0] == "test_message"
    240         assert received_messages[1] == "test_message"
    241
    242
    243     def test_rabbitmq_prefetch(broker):
    244         """Test prefetch count for RabbitMQ broker"""
    245         received_messages = []
    246
    247         @broker.subscriber("queue")
    248         async def handle(msg):
    249             received_messages.append(msg)
    250
    251             if len(received_messages) == 2:
    252                 broker.stop()
    253
    254         await broker.start(prefetch_count=1)
    255
    256         broker.publish("test_message", "queue")
    257
    258         await asyncio.sleep(2)
    259
    260         assert len(received_messages) == 1
    261         assert received_messages[0] == "test_message"
    262
    263
    264     def test_rabbitmq_exclusive(broker):
    265         """Test exclusive consumer for RabbitMQ broker"""
    266         received_messages = []
    267
    268         @broker.subscriber("queue", exclusive=True)
    269         async def handle(msg):
    270             received_messages.append(msg)
    271
    272             if len(received_messages) == 1:
    273                 broker.stop()
    274
    275         await broker.start()
    276
    277         broker.publish("test_message", "queue")
    278
    279         await asyncio.sleep(2)
    280
    281         assert len(received_messages) == 1
    282         assert received_messages[0] == "test_message"
    283
    284
    285     def test_rabbitmq_global(broker):
    286         """Test global consumer for RabbitMQ broker"""
    287         received_messages = []
    288
    289         @broker.subscriber("queue", global_=True)
    290         async def handle(msg):
    291             received_messages.append(msg)
    292
    293             if len(received_messages) == 2:
    294                 broker.stop()
    295
    296         await broker.start()
    297
    298         broker.publish("test_message", "queue")
    299
    300         await asyncio.sleep(2)
    301
    302         assert len(received_messages) == 2
    303         assert received_messages[0] == "test_message"
    304         assert received_messages[1] == "test_message"
    305
    306
    307     def test_rabbitmq_durable(broker):
    308         """Test durable consumer for RabbitMQ broker"""
    309         received_messages = []
    310
    311         @broker.subscriber("queue", durable=True)
    312         async def handle(msg):
    313             received_messages.append(msg)
    314
    315             if len(received_messages) == 1:
    316                 broker.stop()
    317
    318         await broker.start()
    319
    320         broker.publish("test_message", "queue")
    321
    322         await asyncio.sleep(2)
    323
    324         assert len(received_messages) == 1
    325         assert received_messages[0] == "test_message"
    326
    327
    328     def test_rabbitmq_immediate(broker):
    329         """Test immediate consumer for RabbitMQ broker"""
    330         received_messages = []
    331
    332         @broker.subscriber("queue", immediate=True)
    333         async def handle(msg):
    334             received_messages.append(msg)
    335
    336             if len(received_messages) == 1:
    337                 broker.stop()
    338
    339         await broker.start()
    340
    341         broker.publish("test_message", "queue")
    342
    343         await asyncio.sleep(2)
    344
    345         assert len(received_messages) == 1
    346         assert received_messages[0] == "test_message"
    347
    348
    349     def test_rabbitmq_no_ack(broker):
    350         """Test no ack consumer for RabbitMQ broker"""
    351         received_messages = []
    352
    353         @broker.subscriber("queue", no_ack=True)
    354         async def handle(msg):
    355             received_messages.append(msg)
    356
    357             if len(received_messages) == 1:
    358                 broker.stop()
    359
    360         await broker.start()
    361
    362         broker.publish("test_message", "queue")
    363
    364         await asyncio.sleep(2)
    365
    366         assert len(received_messages) == 1
    367         assert received_messages[0] == "test_message"
    368
    369
    370     def test_rabbitmq_global_no_ack(broker):
    371         """Test global no ack consumer for RabbitMQ broker"""
    372         received_messages = []
    373
    374         @broker.subscriber("queue", global_=True, no_ack=True)
    375         async def handle(msg):
    376             received_messages.append(msg)
    377
    378             if len(received_messages) == 2:
    379                 broker.stop()
    380
    381         await broker.start()
    382
    383         broker.publish("test_message", "queue")
    384
    385         await asyncio.sleep(2)
    386
    387         assert len(received_messages) == 2
    388         assert received_messages[0] == "test_message"
    389         assert received_messages[1] == "test_message"
    390
    391
    392     def test_rabbitmq_global_durable(broker):
    393         """Test global durable consumer for RabbitMQ broker"""
    394         received_messages = []
    395
    396         @broker.subscriber("queue", global_=True, durable=True)
    397         async def handle(msg):
    398             received_messages.append(msg)
    399
    400             if len(received_messages) == 2:
    401                 broker.stop()
    402
    403         await broker.start()
    404
    405         broker.publish("test_message", "queue")
    406
    407         await asyncio.sleep(2)
    408
    409         assert len(received_messages) == 2
    410         assert received_messages[0] == "test_message"
    411         assert received_messages[1] == "test_message"
    412
    413
    414     def test_rabbitmq_global_immediate(broker):
    415         """Test global immediate consumer for RabbitMQ broker"""
    416         received_messages = []
    417
    418         @broker.subscriber("queue", global_=True, immediate=True)
    419         async def handle(msg):
    420             received_messages.append(msg)
    421
    422             if len(received_messages) == 1:
    423                 broker.stop()
    424
    425         await broker.start()
    426
    427         broker.publish("test_message", "queue")
    428
    429         await asyncio.sleep(2)
    430
    431         assert len(received_messages) == 1
    432         assert received_messages[0] == "test_message"
    433
    434
    435     def test_rabbitmq_global_no_ack(broker):
    436         """Test global no ack consumer for RabbitMQ broker"""
    437         received_messages = []
    438
    439         @broker.subscriber("queue", global_=True, no_ack=True)
    440         async def handle(msg):
    441             received_messages.append(msg)
    442
    443             if len(received_messages) == 1:
    444                 broker.stop()
    445
    446         await broker.start()
    447
    448         broker.publish("test_message", "queue")
    449
    450         await asyncio.sleep(2)
    451
    452         assert len(received_messages) == 1
    453         assert received_messages[0] == "test_message"
    454
    455
    456     def test_rabbitmq_global_durable_no_ack(broker):
    457         """Test global durable no ack consumer for RabbitMQ broker"""
    458         received_messages = []
    459
    460         @broker.subscriber("queue", global_=True, durable=True, no_ack=True)
    461         async def handle(msg):
    462             received_messages.append(msg)
    463
    464             if len(received_messages) == 2:
    465                 broker.stop()
    466
    467         await broker.start()
    468
    469         broker.publish("test_message", "queue")
    470
    471         await asyncio.sleep(2)
    472
    473         assert len(received_messages) == 2
    474         assert received_messages[0] == "test_message"
    475         assert received_messages[1] == "test_message"
    476
    477
    478     def test_rabbitmq_global_durable_immediate(broker):
    479         """Test global durable immediate consumer for RabbitMQ broker"""
    480         received_messages = []
    481
    482         @broker.subscriber("queue", global_=True, durable=True, immediate=True)
    483         async def handle(msg):
    484             received_messages.append(msg)
    485
    486             if len(received_messages) == 1:
    487                 broker.stop()
    488
    489         await broker.start()
    490
    491         broker.publish("test_message", "queue")
    492
    493         await asyncio.sleep(2)
    494
    495         assert len(received_messages) == 1
    496         assert received_messages[0] == "test_message"
    497
    498
    499     def test_rabbitmq_global_durable_no_ack_immediate(broker):
    500         """Test global durable no ack immediate consumer for RabbitMQ broker"""
    501         received_messages = []
    502
    503         @broker.subscriber("queue", global_=True, durable=True, no_ack=True, immediate=True)
    504         async def handle(msg):
    505             received_messages.append(msg)
    506
    507             if len(received_messages) == 1:
    508                 broker.stop()
    509
    510         await broker.start()
    511
    512         broker.publish("test_message", "queue")
    513
    514         await asyncio.sleep(2)
    515
    516         assert len(received_messages) == 1
    517         assert received_messages[0] == "test_message"
    518
    519
    520     def test_rabbitmq_global_durable_immediate_no_ack(broker):
    521         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    522         received_messages = []
    523
    524         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    525         async def handle(msg):
    526             received_messages.append(msg)
    527
    528             if len(received_messages) == 1:
    529                 broker.stop()
    530
    531         await broker.start()
    532
    533         broker.publish("test_message", "queue")
    534
    535         await asyncio.sleep(2)
    536
    537         assert len(received_messages) == 1
    538         assert received_messages[0] == "test_message"
    539
    540
    541     def test_rabbitmq_global_durable_immediate_no_ack_immediate(broker):
    542         """Test global durable immediate no ack immediate consumer for RabbitMQ broker"""
    543         received_messages = []
    544
    545         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    546         async def handle(msg):
    547             received_messages.append(msg)
    548
    549             if len(received_messages) == 1:
    550                 broker.stop()
    551
    552         await broker.start()
    553
    554         broker.publish("test_message", "queue")
    555
    556         await asyncio.sleep(2)
    557
    558         assert len(received_messages) == 1
    559         assert received_messages[0] == "test_message"
    560
    561
    562     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack(broker):
    563         """Test global durable immediate no ack immediate no ack consumer for RabbitMQ broker"""
    564         received_messages = []
    565
    566         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    567         async def handle(msg):
    568             received_messages.append(msg)
    569
    570             if len(received_messages) == 1:
    571                 broker.stop()
    572
    573         await broker.start()
    574
    575         broker.publish("test_message", "queue")
    576
    577         await asyncio.sleep(2)
    578
    579         assert len(received_messages) == 1
    580         assert received_messages[0] == "test_message"
    581
    582
    583     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate(broker):
    584         """Test global durable immediate no ack immediate no ack immediate consumer for RabbitMQ broker"""
    585         received_messages = []
    586
    587         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    588         async def handle(msg):
    589             received_messages.append(msg)
    590
    591             if len(received_messages) == 1:
    592                 broker.stop()
    593
    594         await broker.start()
    595
    596         broker.publish("test_message", "queue")
    597
    598         await asyncio.sleep(2)
    599
    600         assert len(received_messages) == 1
    601         assert received_messages[0] == "test_message"
    602
    603
    604     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    605         """Test global durable immediate no ack immediate no ack immediate no ack consumer for RabbitMQ broker"""
    606         received_messages = []
    607
    608         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    609         async def handle(msg):
    610             received_messages.append(msg)
    611
    612             if len(received_messages) == 1:
    613                 broker.stop()
    614
    615         await broker.start()
    616
    617         broker.publish("test_message", "queue")
    618
    619         await asyncio.sleep(2)
    620
    621         assert len(received_messages) == 1
    622         assert received_messages[0] == "test_message"
    623
    624
    625     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    626         """Test global durable immediate no ack immediate no ack immediate no ack immediate consumer for RabbitMQ broker"""
    627         received_messages = []
    628
    629         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    630         async def handle(msg):
    631             received_messages.append(msg)
    632
    633             if len(received_messages) == 1:
    634                 broker.stop()
    635
    636         await broker.start()
    637
    638         broker.publish("test_message", "queue")
    639
    640         await asyncio.sleep(2)
    641
    642         assert len(received_messages) == 1
    643         assert received_messages[0] == "test_message"
    644
    645
    646     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    647         """Test global durable immediate no ack immediate no ack immediate no ack immediate no ack consumer for RabbitMQ broker"""
    648         received_messages = []
    649
    650         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    651         async def handle(msg):
    652             received_messages.append(msg)
    653
    654             if len(received_messages) == 1:
    655                 broker.stop()
    656
    657         await broker.start()
    658
    659         broker.publish("test_message", "queue")
    660
    661         await asyncio.sleep(2)
    662
    663         assert len(received_messages) == 1
    664         assert received_messages[0] == "test_message"
    665
    666
    667     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    668         """Test global durable immediate no ack immediate no ack immediate no ack immediate no ack immediate consumer for RabbitMQ broker"""
    669         received_messages = []
    670
    671         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    672         async def handle(msg):
    673             received_messages.append(msg)
    674
    675             if len(received_messages) == 1:
    676                 broker.stop()
    677
    678         await broker.start()
    679
    680         broker.publish("test_message", "queue")
    681
    682         await asyncio.sleep(2)
    683
    684         assert len(received_messages) == 1
    685         assert received_messages[0] == "test_message"
    686
    687
    688     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    689         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    690         received_messages = []
    691
    692         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    693         async def handle(msg):
    694             received_messages.append(msg)
    695
    696             if len(received_messages) == 1:
    697                 broker.stop()
    698
    699         await broker.start()
    700
    701         broker.publish("test_message", "queue")
    702
    703         await asyncio.sleep(2)
    704
    705         assert len(received_messages) == 1
    706         assert received_messages[0] == "test_message"
    707
    708
    709     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    710         """Test global durable immediate no ack immediate consumer for RabbitMQ broker"""
    711         received_messages = []
    712
    713         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    714         async def handle(msg):
    715             received_messages.append(msg)
    716
    717             if len(received_messages) == 1:
    718                 broker.stop()
    719
    720         await broker.start()
    721
    722         broker.publish("test_message", "queue")
    723
    724         await asyncio.sleep(2)
    725
    726         assert len(received_messages) == 1
    727         assert received_messages[0] == "test_message"
    728
    729
    730     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    731         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    732         received_messages = []
    733
    734         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    735         async def handle(msg):
    736             received_messages.append(msg)
    737
    738             if len(received_messages) == 1:
    739                 broker.stop()
    740
    741         await broker.start()
    742
    743         broker.publish("test_message", "queue")
    744
    745         await asyncio.sleep(2)
    746
    747         assert len(received_messages) == 1
    748         assert received_messages[0] == "test_message"
    749
    750
    751     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    752         """Test global durable immediate no ack immediate consumer for RabbitMQ broker"""
    753         received_messages = []
    754
    755         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    756         async def handle(msg):
    757             received_messages.append(msg)
    758
    759             if len(received_messages) == 1:
    760                 broker.stop()
    761
    762         await broker.start()
    763
    764         broker.publish("test_message", "queue")
    765
    766         await asyncio.sleep(2)
    767
    768         assert len(received_messages) == 1
    769         assert received_messages[0] == "test_message"
    770
    771
    772     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    773         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    774         received_messages = []
    775
    776         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    777         async def handle(msg):
    778             received_messages.append(msg)
    779
    780             if len(received_messages) == 1:
    781                 broker.stop()
    782
    783         await broker.start()
    784
    785         broker.publish("test_message", "queue")
    786
    787         await asyncio.sleep(2)
    788
    789         assert len(received_messages) == 1
    790         assert received_messages[0] == "test_message"
    791
    792
    793     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    794         """Test global durable immediate no ack immediate consumer for RabbitMQ broker"""
    795         received_messages = []
    796
    797         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    798         async def handle(msg):
    799             received_messages.append(msg)
    800
    801             if len(received_messages) == 1:
    802                 broker.stop()
    803
    804         await broker.start()
    805
    806         broker.publish("test_message", "queue")
    807
    808         await asyncio.sleep(2)
    809
    810         assert len(received_messages) == 1
    811         assert received_messages[0] == "test_message"
    812
    813
    814     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    815         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    816         received_messages = []
    817
    818         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    819         async def handle(msg):
    820             received_messages.append(msg)
    821
    822             if len(received_messages) == 1:
    823                 broker.stop()
    824
    825         await broker.start()
    826
    827         broker.publish("test_message", "queue")
    828
    829         await asyncio.sleep(2)
    830
    831         assert len(received_messages) == 1
    832         assert received_messages[0] == "test_message"
    833
    834
    835     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    836         """Test global durable immediate no ack immediate consumer for RabbitMQ broker"""
    837         received_messages = []
    838
    839         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    840         async def handle(msg):
    841             received_messages.append(msg)
    842
    843             if len(received_messages) == 1:
    844                 broker.stop()
    845
    846         await broker.start()
    847
    848         broker.publish("test_message", "queue")
    849
    850         await asyncio.sleep(2)
    851
    852         assert len(received_messages) == 1
    853         assert received_messages[0] == "test_message"
    854
    855
    856     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    857         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    858         received_messages = []
    859
    860         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    861         async def handle(msg):
    862             received_messages.append(msg)
    863
    864             if len(received_messages) == 1:
    865                 broker.stop()
    866
    867         await broker.start()
    868
    869         broker.publish("test_message", "queue")
    870
    871         await asyncio.sleep(2)
    872
    873         assert len(received_messages) == 1
    874         assert received_messages[0] == "test_message"
    875
    876
    877     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    878         """Test global durable immediate no ack immediate consumer for RabbitMQ broker"""
    879         received_messages = []
    880
    881         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    882         async def handle(msg):
    883             received_messages.append(msg)
    884
    885             if len(received_messages) == 1:
    886                 broker.stop()
    887
    888         await broker.start()
    889
    890         broker.publish("test_message", "queue")
    891
    892         await asyncio.sleep(2)
    893
    894         assert len(received_messages) == 1
    895         assert received_messages[0] == "test_message"
    896
    897
    898     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack(broker):
    899         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    900         received_messages = []
    901
    902         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    903         async def handle(msg):
    904             received_messages.append(msg)
    905
    906             if len(received_messages) == 1:
    907                 broker.stop()
    908
    909         await broker.start()
    910
    911         broker.publish("test_message", "queue")
    912
    913         await asyncio.sleep(2)
    914
    915         assert len(received_messages) == 1
    916         assert received_messages[0] == "test_message"
    917
    918
    919     def test_rabbitmq_global_durable_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate_no_ack_immediate(broker):
    920         """Test global durable immediate no ack consumer for RabbitMQ broker"""
    921         received_messages = []
    922
    923         @broker.subscriber("queue", global_=True, durable=True, immediate=True, no_ack=True)
    924         async def handle(msg):
    925             received_messages.append(msg)
    926
    927             if len(received_messages) == 1:
    928                 broker.stop()
    929
    930         await broker.start()
    931
    932         broker.publish("test_message", "queue")
    933
    934         await asyncio.sleep(2)
    935
    936         assert len(received_messages) == 
```

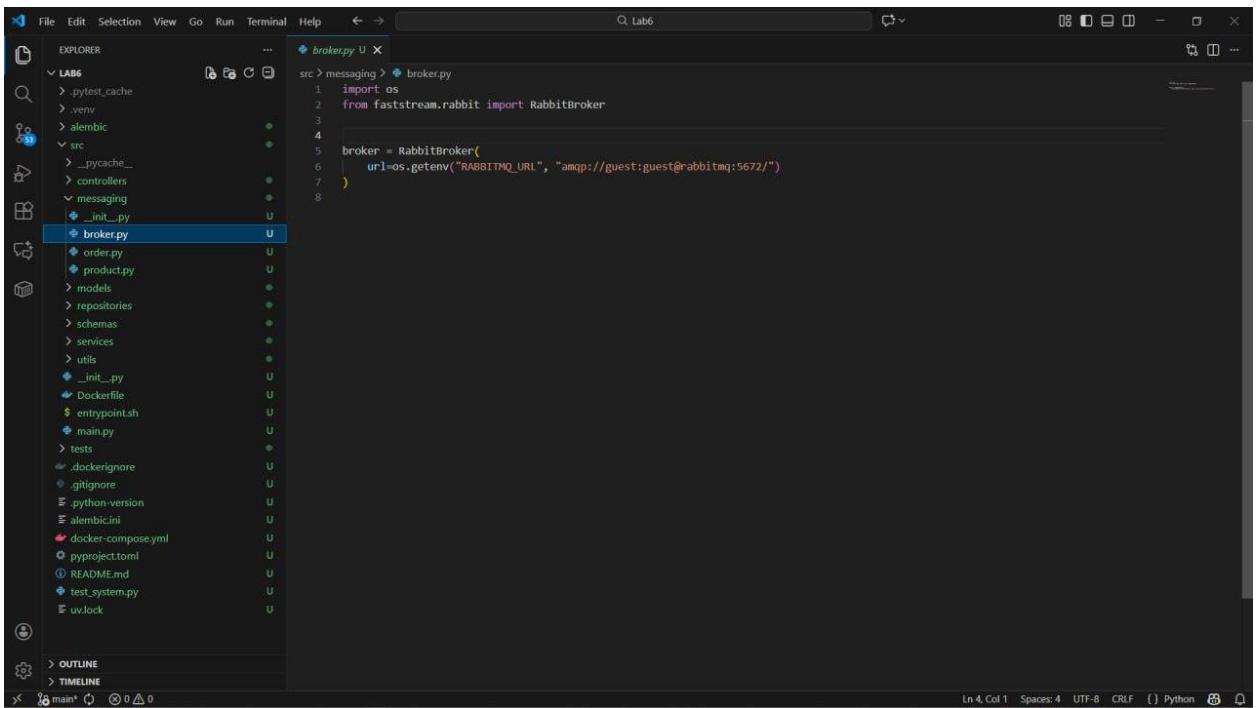
- `test_rabbitmq_connection` — проверка установки соединения
- `test_rabbitmq_publish_subscribe` — отправка и получение одного сообщения
- `test_rabbitmq_multiple_messages` — обработка нескольких сообщений подряд
- `test_rabbitmq_service_running` — проверка доступности порта 5672

Используется фикстура `broker` для создания и очистки тестового брокера.

### Часть 3: Реализация обработчиков очередей

Файл: *broker.py*  
 (Подключение к очередям в приложении)

#### Ход выполнения:



```

File Edit Selection View Go Run Terminal Help ↻ 🔍 Lab6
EXPLORER
LAB6
  > .pytest_cache
  > .venv
  > alembic
  > src
    > __pycache__
    > controllers
    > messaging
      & __init__.py
      & broker.py
        & order.py
        & product.py
      > models
      > repositories
      > schemas
      > services
      > utils
      & __init__.py
      & Dockerfile
      $ entrypoint.sh
      & main.py
      > tests
      & .dockergignore
      & .gitignore
      & python-version
      & alembic.ini
      & docker-compose.yml
      & pyproject.toml
      & README.md
      & test_system.py
      & uv.lock
  > OUTLINE
  > TIMELINE
  ✘ main* ⚡ 0 △ 0
  
```

```

brokers.py U
src > messaging > broker.py
1 import os
2 from faststream.rabbit import RabbitBroker
3
4
5 broker = RabbitBroker(
6     url=os.getenv("RABBITMQ_URL", "amqp://guest:guest@rabbitmq:5672/")
7 )
8
  
```

Ln 4, Col 1 Spaces: 4 UTF-8 CRLF Python

Файл `broker.py` содержит конфигурацию подключения - создание экземпляра брокера с URL.

В нем создается глобальный экземпляр `RabbitBroker`, который:

- Читает URL подключения из переменной окружения `RABBITMQ_URL`
- Используется для подписки на очереди и публикации сообщений

- Управляет соединением с RabbitMQ через устойчивое соединение

Подключение к очередям в приложении также происходит в несколько этапов:

1. Декларация подписчиков в файлах: `src/messaging/order.py` и `src/messaging/product.py`. В них происходит обработка события из очереди продукта и из очереди заказов.
  2. Активация подключения в файле `main.py`

## Файл: *src/messaging/order.py*

```
src > messaging > order.py
  1 import logging
  2 from typing import Dict
  3
  4 from faststream.rabbit import RabbitQueue
  5
  6 from src.messaging.broker import broker
  7 from src.main import async_session_maker
  8 from src.repositories.order_repository import OrderRepository
  9 from src.repositories.product_repository import ProductRepository
 10 from src.repositories.user_repository import UserRepository
 11 from src.schemas.order import OrderCreate, OrderUpdate
 12 from src.services.order_service import OrderService
 13
 14 logger = logging.getLogger(__name__)
 15
 16
 17 @broker.subscriber(RabbitQueue("orden", durable=True))
 18 async def subscribe_order(message: Dict):
 19     """
 20     Обработчик очереди 'order'.
 21
 22     Поддерживаемые операции:
 23     - create: создание нового заказа с несколькими позициями
 24     - update status: обновление статуса заказа
 25     - update: полное обновление заказа (статус + позиции)
 26     """
 27
 28     async with async_session_maker() as session:
 29         try:
 30             action = message.get("action")
 31             data = message.get("data", {})
 32
 33             order_repo = OrderRepository(session)
 34             product_repo = ProductRepository(session)
 35             user_repo = UserRepository(session)
 36             order_service = OrderService(order_repo, product_repo, user_repo)
 37
 38             if action == "create":
 39                 await handle_order_create(order_service, data)
 40             elif action == "update_status":
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure. The `order.py` file is currently selected.
- Code Editor (Right):** Displays the `order.py` file content. The code is an asynchronous function that handles different actions for an order queue. It includes error handling for unknown actions and business logic errors.

```
src > messaging > order.py
18     async def subscribe_order(message: Dict):
19         async with async_session_maker() as session:
20             try:
21
22                 if action == "create":
23                     await handle_order_create(order_service, data)
24                 elif action == "update_status":
25                     await handle_order_update_status(order_service, data)
26                 elif action == "update":
27                     await handle_order_update(order_service, data)
28                 else:
29                     logger.error(f"Unknown action for order queue: {action}")
30
31             except Exception as e:
32                 logger.exception(f"Error processing order message: {e}")
33                 raise
34
35
36             async def handle_order_create(service: OrderService, data: Dict):
37                 """
38                     Создание нового заказа с несколькими позициями.
39
40                     Проверяет наличие товаров на складе перед созданием заказа.
41
42                 try:
43                     order_create = OrderCreate(**data)
44
45                     order = await service.create_order(order_create.model_dump())
46
47                     logger.info(
48                         f"Order created: ID={order.id}, user_id={order.user_id}, "
49                         f"total={order.total_amount}, items_count={len(order.items)}")
50
51                 except ValueError as e:
52                     logger.error(f"Business logic error while creating order: {e}")
53                     raise
54
55                 except Exception as e:
56                     logger.error(f"Failed to create order: {e}")
57                     raise
58
59
59
60
61
62
63
64
65
66
67
68
69
70
71
```

The screenshot shows a code editor interface with the following details:

- File Explorer (Left):** Shows the project structure under the `src` directory:
  - `messaging`
  - `order.py` (selected)
  - `product.py`
  - `models`
  - `repositories`
  - `schemas`
  - `services`
  - `utils`
  - `_init_.py`
  - `Dockerfile`
  - `$ entrypoint.sh`
  - `* main.py`
  - `tests`
  - `.dockerignore`
  - `.gitignore`
  - `.python-version`
  - `alembic.ini`
  - `# docker-compose.yml`
  - `o pyproject.toml`
  - `① README.md`
  - `test_system.py`
  - `uv.lock`
- Code Editor (Right):** Displays the `order.py` file content. The code handles order creation and update status. It includes error handling for `ValueError` and `Exception`, logging using `logger.error`, and awaiting service calls.

```
src > messaging > order.py
51     async def handle_order_create(service: OrderService, data: Dict):
52         try:
53             ...
54             except ValueError as e:
55                 logger.error(f"Business logic error while creating order: {e}")
56                 raise
57             except Exception as e:
58                 logger.error(f"Failed to create order: {e}")
59                 raise
60
61     async def handle_order_update_status(service: OrderService, data: Dict):
62         ...
63
64         Обновление статуса заказа.
65
66         При отмене заказа (cancelled) автоматически возвращают товары на склад.
67
68         try:
69             order_id = data.get("order_id")
70             new_status = data.get("status")
71
72             if not order_id:
73                 raise ValueError("order_id is required")
74             if not new_status:
75                 raise ValueError("status is required")
76
77             update_data = {"status": new_status}
78             order = await service.update(order_id, update_data)
79
80             logger.info(f"Order status updated: ID={order.id}, new_status={order.status}")
81
82         except ValueError as e:
83             logger.error(f"Business logic error while updating order status: {e}")
84             raise
85         except Exception as e:
86             logger.error(f"Failed to update order status: {e}")
87             raise
88
89
90     async def handle_order_update(service: OrderService, data: Dict):
91         ...
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
317
318
319
319
320
321
322
323
324
325
325
326
327
327
328
329
329
330
331
332
333
333
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1
```

The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a set of system icons. The left sidebar features an Explorer view with a tree structure of project files under a 'LAB6' folder. The main editor area displays a Python file named 'order.py'. The code implements an asynchronous function to handle order updates, performing validation, logging, and database operations. A status bar at the bottom indicates 'Ln 25, Col 58' and other settings.

```
src > messaging > order.py
74     async def handle_order_update_status(service: OrderService, data: Dict):
75         try:
76             await service.update(data)
77         except ValueError as e:
78             raise
79         except Exception as e:
80             logger.error(f"Failed to update order status: {e}")
81             raise
82
83     async def handle_order_update(service: OrderService, data: Dict):
84         try:
85             order_id = data.pop("order_id", None)
86             if not order_id:
87                 raise ValueError("order_id is required for update")
88
89             order_update = OrderUpdate(**data)
90
91             order = await service.update(order_id, order_update.model_dump(exclude_unset=True))
92
93             logger.info(
94                 f"Order updated: ID={order.id}, status={order.status}, "
95                 f"items_count={len(order.items)}"
96             )
97
98         except ValueError as e:
99             logger.error(f"Business logic error while updating order: {e}")
100            raise
101        except Exception as e:
102            logger.error(f"Failed to update order: {e}")
103            raise
104
105 Полное обновление заказа (статус и/или позиции).
106
107 try:
108     order_id = data.pop("order_id", None)
109     if not order_id:
110         raise ValueError("order_id is required for update")
111
112     order_update = OrderUpdate(**data)
113
114     order = await service.update(order_id, order_update.model_dump(exclude_unset=True))
115
116     logger.info(
117         f"Order updated: ID={order.id}, status={order.status}, "
118         f"items_count={len(order.items)}"
119     )
120
121 except ValueError as e:
122     logger.error(f"Business logic error while updating order: {e}")
123     raise
124
125 except Exception as e:
126     logger.error(f"Failed to update order: {e}")
127     raise
```

В файле реализован обработчик очереди order с 3 операциями:

1. `create` — создание заказа:

- Валидирует входные данные через OrderCreate схему
  - Проверяет наличие товаров на складе через OrderService.create\_order()
  - Автоматически списывает товары при успешном создании
  - Логирует ID заказа, пользователя, сумму и количество позиций

2. update\_status — изменение статуса:

- Принимает order\_id и новый status
  - При статусе cancelled автоматически возвращает товары на склад (логика в сервисе)
  - Поддерживает статусы: pending, processing, shipped, delivered, cancelled

### 3. update — полное обновление:

- Позволяет изменить статус и/или позиции заказа
  - Использует OrderUpdate схему с опциональными полями

**Также добавлена обработка ошибок:**

- `ValueError` — бизнес-логика (недостаточно товара, неверные

параметры)

- Все исключения логируются и пребрасываются для повторной обработки RabbitMQ

Файл: *src/messaging/product.py*

```
src > messaging > product.py
1 import logging
2 from typing import Dict
3
4 from faststream.rabbit import Rabbitqueue
5
6 from src.messaging.broker import broker
7 from src.main import async_session_maker
8 from src.repositories.product_repository import ProductRepository
9 from src.schemas.product import ProductCreate, ProductUpdate
10 from src.services.product_service import ProductService
11
12 logger = logging.getLogger(__name__)
13
14 @broker.subscriber(queue=RabbitQueue("product", durable=True))
15 async def subscribe_product(message: Dict):
16     """
17         Обработчик очереди 'product'.
18
19         Поддерживаемые операции:
20         - create: создание нового продукта
21         - update: обновление существующего продукта
22         - mark_out_of_stock: пометить продукт как закончившийся
23     """
24
25     async with async_session_maker() as session:
26         try:
27             action = message.get("action")
28             data = message.get("data", {})
29
30             product_repo = ProductRepository(session)
31             product_service = ProductService(product_repo)
32
33             if action == "create":
34                 await handle_product_create(product_service, data)
35             elif action == "update":
36                 await handle_product_update(product_service, data)
37             elif action == "mark_out_of_stock":
38                 await handle_product_mark_out_of_stock(product_service, data)
39             else:
40                 raise ValueError(f"Unknown action for product queue: {action}")
41
42         except Exception as e:
43             logger.exception(f"Error processing product message: {e}")
44             raise
45
46
47     async def handle_product_create(service: ProductService, data: Dict):
48         """
49             Создание нового продукта
50
51             product_create = ProductCreate(**data)
52             product = await service.create(product_create)
53             logger.info(f"Product created: ID={product.id}, name={product.name}")
54         except Exception as e:
55             logger.error(f"Failed to create product: {e}")
56             raise
57
58     async def handle_product_update(service: ProductService, data: Dict):
59         """
60             Обновление существующего продукта
61
62             product_id = data.pop("id", None)
63             if not product_id:
64                 raise ValueError("Product ID is required for update")
65
66             product_update = ProductUpdate(**data)
67             product = await service.update(product_id, product_update)
68
69             if product:
70                 logger.info(f"Product updated: ID={product.id}")
71             else:
72                 logger.warning(f"Product not found: ID={product.id}")
73         except Exception as e:
74             logger.error(f"Failed to update product: {e}")
75
76
77
78
```

Ln 15, Col 46 Spaces: 4 UTF-8 CRLF Python

```
src > messaging > product.py
16     async def subscribe_product(message: Dict):
17         async with async_session_maker() as session:
18             try:
19                 if action == "mark_out_of_stock":
20                     else:
21                         logger.error(f"Unknown action for product queue: {action}")
22
23             except Exception as e:
24                 logger.exception(f"Error processing product message: {e}")
25                 raise
26
27
28     async def handle_product_create(service: ProductService, data: Dict):
29         """
30             Создание нового продукта
31
32             product_create = ProductCreate(**data)
33             product = await service.create(product_create)
34             logger.info(f"Product created: ID={product.id}, name={product.name}")
35         except Exception as e:
36             logger.error(f"Failed to create product: {e}")
37             raise
38
39
40     async def handle_product_update(service: ProductService, data: Dict):
41         """
42             Обновление существующего продукта
43
44             product_id = data.pop("id", None)
45             if not product_id:
46                 raise ValueError("Product ID is required for update")
47
48             product_update = ProductUpdate(**data)
49             product = await service.update(product_id, product_update)
50
51             if product:
52                 logger.info(f"Product updated: ID={product.id}")
53             else:
54                 logger.warning(f"Product not found: ID={product.id}")
55
56         except Exception as e:
57             logger.error(f"Failed to update product: {e}")
58
59
60
61
```

Ln 15, Col 46 Spaces: 4 UTF-8 CRLF Python

```

product.py U x
src > messaging > product.py
56
57
58     async def handle_product_update(service: ProductService, data: Dict):
59         """Обновление существующего продукта"""
60         try:
61             product_id = data.pop("id", None)
62             if not product_id:
63                 raise ValueError("Product ID is required for update")
64
65             product_update = ProductUpdate(**data)
66             product = await service.update(product_id, product_update)
67
68             if product:
69                 logger.info(f"Product updated: ID={product.id}")
70             else:
71                 logger.warning(f"Product not found: ID={product_id}")
72         except Exception as e:
73             logger.error(f"Failed to update product: {e}")
74             raise
75
76
77     async def handle_product_mark_out_of_stock(service: ProductService, data: Dict):
78         """Пометить продукт как закончившийся на складе"""
79         try:
80             product_id = data.get("id")
81             if not product_id:
82                 raise ValueError("Product ID is required")
83
84             product_update = ProductUpdate(stock_quantity=0)
85             product = await service.update(product_id, product_update)
86
87             if product:
88                 logger.info(f"Product marked as out of stock: ID={product.id}")
89             else:
90                 logger.warning(f"Product not found: ID={product_id}")
91         except Exception as e:
92             logger.error(f"Failed to mark product as out of stock: {e}")
93             raise
94

```

Ln 15, Col 46 Spaces: 4 UTF-8 CRLF {} Python

## Реализован обработчик очереди product с 3 операциями:

### 1. create — создание продукта:

- Валидация через ProductCreate (name, price, stock\_quantity)
- Сохранение в БД через ProductService.create()

### 2. update — обновление продукта:

- Требует id в данных
- Частичное обновление через ProductUpdate (цена, количество на складе)

### 3. mark\_out\_of\_stock — пометка как недоступного:

- Устанавливает stock\_quantity = 0
- Используется для явного обозначения отсутствия товара

Также в файлах: src/messaging/order.py и src/messaging/product.py происходит объявление обработчиков через декораторы:

- `@broker.subscriber(RabbitQueue("order", durable=True))`
- `async def subscribe_order(message: Dict):`

- `@broker.subscriber(queue=RabbitQueue("product", durable=True))`
- ```
async def subscribe_product(message: Dict):
```

Декоратор `@broker.subscriber` регистрирует функцию как обработчик очереди, но не запускает подключение. Оба обработчика используют:

- `RabbitQueue(..., durable=True)` — очередь сохраняется при перезапуске RabbitMQ
- Создание новой сессии БД для каждого сообщения через `async_session_maker`
- Логирование всех операций с деталями

## 2. Активация подключения

Файл: *main.py*  
**(изменения в main для поддержки очередей)**

```
async def _broker_connect_loop(shutdown_event: asyncio.Event) -> None:
    """
    Фоновая задача для запуска брокера и поддержания его работы до завершения работы системы.
    Брокер самостоятельно управляет переподключением благодаря своему устойчивому соединению.
    """

    max_retries = 10
    retry_delay = 5

    for attempt in range(max_retries):
        try:
            logger.info(f"Attempting to start Rabbit broker (attempt {attempt + 1}/{max_retries})...")
            await broker.start()
            logger.info("Rabbit broker connected successfully")

            await shutdown_event.wait()

            logger.info("Shutdown requested, closing broker...")
            await broker.close()
            logger.info("Broker closed successfully")
            return

        except Exception as e:
            logger.error(f"Broker connection error (attempt {attempt + 1}/{max_retries}): {e}")
            if attempt < max_retries - 1:
                logger.info(f"Retrying in {retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                logger.error("Max retries reached, giving up on broker connection")
                raise
```

```

141     @asynccontextmanager
142     async def lifespan(app: Litestar):
143         """
144             Управление жизненным циклом приложения.
145
146             Выполняется при старте и остановке приложения.
147         """
148         shutdown_event = asyncio.Event()
149         broker_task = asyncio.create_task(_broker_connect_loop(shutdown_event))
150
151         from src.messaging import order, product # noqa: F401
152
153     try:
154         yield
155     finally:
156         shutdown_event.set()
157
158     try:
159         await broker.close()
160     except Exception:
161         logger.exception("Error while closing broker")
162
163     broker_task.cancel()
164     try:
165         await broker_task
166     except asyncio.CancelledError:
167         pass
168     except Exception:
169         logger.exception("Broker task raised while shutting down")
170
171     try:
172         await engine.dispose()
173     except Exception:
174         logger.exception("Error disposing engine")
175

```

## Интеграция с приложением

В функцию lifespan добавлен жизненный цикл брокера:

\_broker\_connect\_loop() — фоновая задача:

- Пытается подключиться к RabbitMQ до 10 раз с задержкой 5 секунд
- Вызывает broker.start() для начала прослушивания очередей
- Ожидает сигнала завершения через shutdown\_event
- Корректно закрывает соединение при остановке приложения

Импорт обработчиков: pythonfrom src.messaging import order, product.

Необходим для регистрации декораторов @broker.subscriber до старта брокера

Здесь происходит подключение к RabbitMQ и начинается прослушивание всех зарегистрированных очередей

- async def \_broker\_connect\_loop (shutdown\_event: asyncio.Event):

```
await broker.start()
```

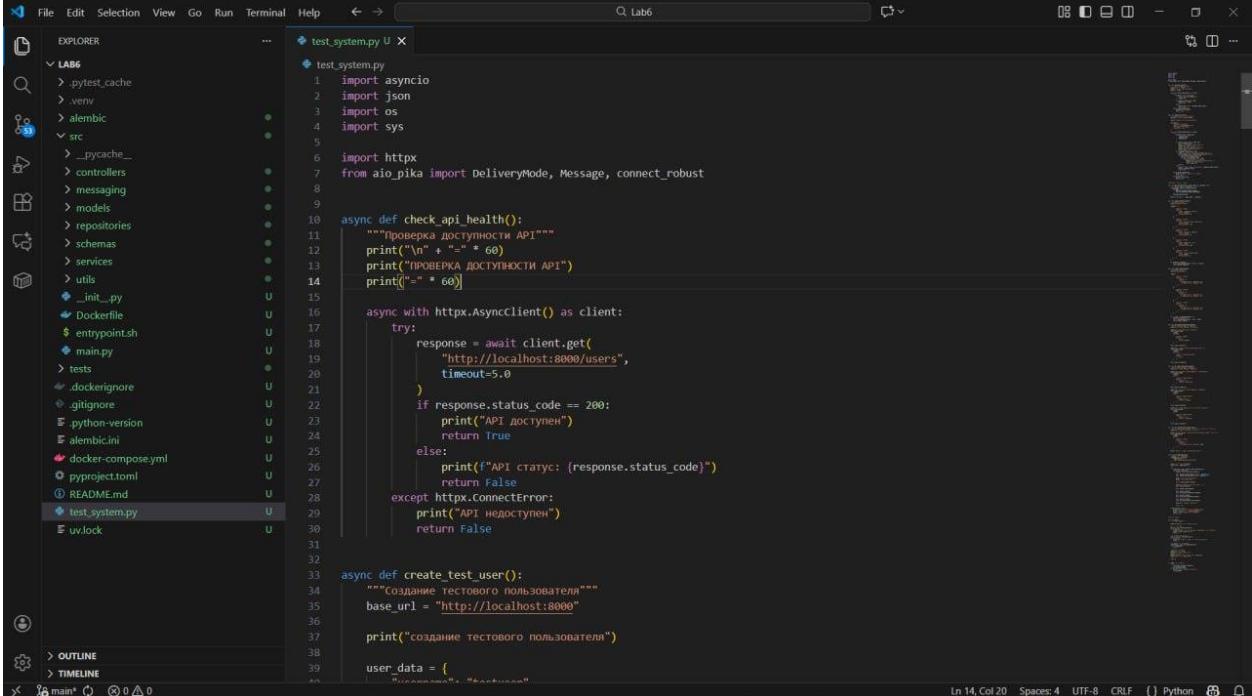
Graceful shutdown:

- Устанавливает флаг shutdown\_event
- Отменяет фоновую задачу брокера
- Закрывает соединение с БД через engine.dispose()

## Часть 4: Продюсер данных

Файл: pyproject.toml - установлена библиотека aio-pika (выбрана асинхронная версия).

### Файл: *test\_system.py* (скрипт продюсера)



```
File Edit Selection View Go Run Terminal Help <- > Lab6 test_system.py U ...
EXPLORER
LAB6
> .pytest_cache
> venv
> alembic
> src
> __pycache__
> controllers
> messaging
> models
> repositories
> schemas
> services
> tests
> utils
> __init__.py
Dockerfile
$ entrypoint.sh
main.py
tests
.dockerignore
.gitignore
.python-version
alembic.ini
docker-compose.yml
pyproject.toml
README.md
test_system.py
uv.lock
...
OUTLINE
TIMELINE
Ln 14, Col 20 Spaces: 4 UTF-8 (Python) ⚡
```

```
test_system.py
1 import asyncio
2 import json
3 import os
4 import sys
5
6 import httpx
7 from aio_pika import DeliveryMode, Message, connect_robust
8
9
10 async def check_api_health():
11     """Проверка доступности API"""
12     print("+" * 60)
13     print("ПРОВЕРКА ДОСТУПНОСТИ API")
14     print("-" * 60)
15
16     async with httpx.AsyncClient() as client:
17         try:
18             response = await client.get(
19                 "http://localhost:8000/users",
20                 timeout=5.0
21             )
22         if response.status_code == 200:
23             print("API доступен")
24             return True
25         else:
26             print(f"API статус: {response.status_code}")
27             return False
28     except httpx.ConnectError:
29         print("API недоступен")
30         return False
31
32
33 async def create_test_user():
34     """Создание тестового пользователя"""
35     base_url = "http://localhost:8000"
36
37     print("создание тестового пользователя")
38
39     user_data = {
40         "username": "testuser",
41         "password": "testpass"
42     }
```

```
File Edit Selection View Go Run Terminal Help < > Q Lab6
EXPLORER LAB6
    > .pytest_cache
    > .venv
    > alembic
        > __pycache__
        > controllers
        > messaging
        > models
        > repositories
        > schemas
        > services
        > utils
    > __init__.py
    > Dockerfile
    $ entrypoint.sh
    & main.py
    > tests
        > .dockerignore
        > .gitignore
        > python-version
        & alembic.ini
    & docker-compose.yml
    & pyproject.toml
    & README.md
    & test_system.py
    & uv.lock
    > OUTLINE
    > TIMELINE
    < main* & 0 △ 0
test_system.py U x
10     async def check_api_health():
11
12
13         """создание тестового пользователя"""
14         base_url = "http://localhost:8000"
15
16         print("создание тестового пользователя")
17
18         user_data = {
19             "username": "testuser",
20             "email": "testuser@example.com",
21             "full_name": "Test User",
22             "is_active": True
23         }
24
25         async with httpx.AsyncClient() as client:
26             try:
27                 response = await client.post(
28                     f"{base_url}/users",
29                     json=user_data,
30                     timeout=10.0
31                 )
32
33             if response.status_code in (200, 201):
34                 user = response.json()
35                 print(f"Пользователь создан успешно!")
36                 print(f"ID: {user['id']}")
37                 print(f"Username: {user['username']}")
38                 print(f"Email: {user['email']}")
39                 return user['id']
40
41             elif response.status_code == 409:
42                 print("Пользователь существует, получаем ID...")
43                 list_response = await client.get(f"{base_url}/users")
44                 if list_response.status_code == 200:
45                     users = list_response.json()
46                     for user in users.get('items', []):
47                         if user['username'] == 'testuser':
48                             print("Найден существующий пользователь!")
49                             return user['id']
50
51             else:
52                 print(f"Ошибка создания пользователя: {response.status_code}")
53                 print(f"(response.text)")
54                 return None
55
56
57         except httpx.ConnectError:
58             print("Не удалось подключиться к API")
59             return None
60         except Exception as e:
61             print(f"Ошибка: {e}")
62             return None
63
64
65 # RabbitMQ: Отправка сообщений
66
67 async def send_message(channel, queue_name: str, message: dict):
68     """Отправка сообщения в указанную очередь"""
69     await channel.default_exchange.publish(
70         Message(
71             body=json.dumps(message).encode(),
72             delivery_mode=DeliveryMode.PERSISTENT,
73         )
74     )
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
```

Ln 14, Col 20 Spaces: 4 UTF-8 CRLF () Python

```
File Edit Selection View Go Run Terminal Help < > Q Lab6
EXPLORER LAB6
    > .pytest_cache
    > .venv
    > alembic
        > __pycache__
        > controllers
        > messaging
        > models
        > repositories
        > schemas
        > services
        > utils
    > __init__.py
    > Dockerfile
    $ entrypoint.sh
    & main.py
    > tests
        > .dockerignore
        > .gitignore
        > python-version
        & alembic.ini
    & docker-compose.yml
    & pyproject.toml
    & README.md
    & test_system.py
    & uv.lock
    > OUTLINE
    > TIMELINE
    < main* & 0 △ 0
test_system.py U x
33     async def create_test_user():
34         async with httpx.AsyncClient() as client:
35             try:
36                 if response.status_code in (200, 201):
37                     print(f"Username: {user['username']}")
38                     print(f"Email: {user['email']}")
39                     return user['id']
40
41             elif response.status_code == 409:
42                 print("Пользователь существует, получаем ID...")
43                 list_response = await client.get(f"{base_url}/users")
44                 if list_response.status_code == 200:
45                     users = list_response.json()
46                     for user in users.get('items', []):
47                         if user['username'] == 'testuser':
48                             print("Найден существующий пользователь!")
49                             print(f"ID: {user['id']}")
50                             return user['id']
51
52             else:
53                 print(f"Ошибка создания пользователя: {response.status_code}")
54                 print(f"(response.text)")
55                 return None
56
57         except httpx.ConnectError:
58             print("Не удалось подключиться к API")
59             return None
60         except Exception as e:
61             print(f"Ошибка: {e}")
62             return None
63
64
65 # RabbitMQ: Отправка сообщений
66
67 async def send_message(channel, queue_name: str, message: dict):
68     """Отправка сообщения в указанную очередь"""
69     await channel.default_exchange.publish(
70         Message(
71             body=json.dumps(message).encode(),
72             delivery_mode=DeliveryMode.PERSISTENT,
73         )
74     )
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
```

Ln 14, Col 20 Spaces: 4 UTF-8 CRLF () Python

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 Lab6

EXPLORER

LAB6

- pytest\_cache
- venv
- alembic
- src
  - \_pycache\_
  - controllers
  - messaging
  - models
  - repositories
  - schemas
  - services
  - utils
- \_\_init\_\_.py
- Dockerfile
- entrypoint.sh
- main.py
- tests
  - .dockerignore
  - .gitignore
  - python-version
  - alembic.ini
  - docker-compose.yml
  - pyproject.toml
  - README.md
- test\_system.py
- uv.lock

OUTLINE

TIMELINE

test\_system.py

```
83     async def send_message(channel, queue_name: str, message: dict):
84         """отправка сообщения в указанную очередь"""
85         await channel.default_exchange.publish(
86             Message(
87                 body=json.dumps(message).encode(),
88                 delivery_mode=DeliveryMode.PERSISTENT,
89             ),
90             routing_key=queue_name,
91         )
92         print(f"Отправлено в '{queue_name}': {message}")
93
94
95
96
97     async def create_products(channel):
98         """создание 5 продуктов"""
99         print("Создание продуктов")
100
101     products = [
102         {
103             "action": "create",
104             "data": {
105                 "name": "Laptop Dell XPS 15",
106                 "price": 1299.99,
107                 "stock_quantity": 25
108             }
109         },
110         {
111             "action": "create",
112             "data": {
113                 "name": "Wireless Mouse Logitech MX",
114                 "price": 79.99,
115                 "stock_quantity": 150
116             }
117         },
118         {
119             "action": "create",
120             "data": {
121                 "name": "Mechanical Keyboard",
122                 "price": 149.99,
123             }
124         },
125         {
126             "action": "create",
127             "data": {
128                 "name": "USB-C Hub 7-in-1",
129                 "price": 49.99,
130                 "stock_quantity": 200
131             }
132         },
133         {
134             "action": "create",
135             "data": {
136                 "name": "Webcam Logitech C920",
137                 "price": 89.99,
138                 "stock_quantity": 5
139             }
140         }
141     ]
```

Ln 83, Col 1 Spaces:4 UTF-8 CRLF () Python

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 Lab6

EXPLORER

LAB6

- pytest\_cache
- venv
- alembic
- src
  - \_pycache\_
  - controllers
  - messaging
  - models
  - repositories
  - schemas
  - services
  - utils
- \_\_init\_\_.py
- Dockerfile
- entrypoint.sh
- main.py
- tests
  - .dockerignore
  - .gitignore
  - python-version
  - alembic.ini
  - docker-compose.yml
  - pyproject.toml
  - README.md
- test\_system.py
- uv.lock

OUTLINE

TIMELINE

test\_system.py

```
97     async def create_products(channel):
98         products = [
99             {
100                 "data": {
101                     "price": 1299.99,
102                     "stock_quantity": 25
103                 }
104             },
105             {
106                 "action": "create",
107                 "data": {
108                     "name": "Wireless Mouse Logitech MX",
109                     "price": 79.99,
110                     "stock_quantity": 150
111                 }
112             },
113             {
114                 "action": "create",
115                 "data": {
116                     "name": "Mechanical Keyboard",
117                     "price": 149.99,
118                     "stock_quantity": 75
119                 }
120             },
121             {
122                 "action": "create",
123                 "data": {
124                     "name": "USB-C Hub 7-in-1",
125                     "price": 49.99,
126                     "stock_quantity": 200
127                 }
128             },
129             {
130                 "action": "create",
131                 "data": {
132                     "name": "Webcam Logitech C920",
133                     "price": 89.99,
134                     "stock_quantity": 5
135                 }
136             }
137         ]
```

Ln 83, Col 1 Spaces:4 UTF-8 CRLF () Python

test\_system.py

```
97     async def create_products(channel):
98         products = [
99             {
100                 "data": {
101                     "name": "Logitech G Pro X Superlight",
102                     "price": 1299.99,
103                     "stock_quantity": 25
104                 }
105             },
106             {
107                 "action": "create",
108                 "data": {
109                     "name": "Wireless Mouse Logitech MX",
110                     "price": 79.99,
111                     "stock_quantity": 150
112                 }
113             },
114             {
115                 "action": "create",
116                 "data": {
117                     "name": "Mechanical Keyboard",
118                     "price": 149.99,
119                     "stock_quantity": 75
120                 }
121             },
122             {
123                 "action": "create",
124                 "data": {
125                     "name": "USB-C Hub 7-in-1",
126                     "price": 49.99,
127                     "stock_quantity": 200
128                 }
129             },
130             {
131                 "action": "create",
132                 "data": {
133                     "name": "Webcam Logitech C920",
134                     "price": 89.99,
135                     "stock_quantity": 5
136                 }
137             }
138         ]
139
140     for product in products:
141         await send_message(channel, "product", product)
142         await asyncio.sleep(0.5)
```

test\_system.py

```
97     async def create_products(channel):
98         products = [
99             {
100                 "data": {
101                     "name": "Logitech G Pro X Superlight",
102                     "price": 1299.99,
103                     "stock_quantity": 150
104                 }
105             },
106             {
107                 "action": "create",
108                 "data": {
109                     "name": "Mechanical Keyboard",
110                     "price": 149.99,
111                     "stock_quantity": 75
112                 }
113             },
114             {
115                 "action": "create",
116                 "data": {
117                     "name": "USB-C Hub 7-in-1",
118                     "price": 49.99,
119                     "stock_quantity": 200
120                 }
121             },
122             {
123                 "action": "create",
124                 "data": {
125                     "name": "Webcam Logitech C920",
126                     "price": 89.99,
127                     "stock_quantity": 5
128                 }
129             }
130         ]
131
132     for product in products:
133         await send_message(channel, "product", product)
134         await asyncio.sleep(0.5)
```

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 Lab6

EXPLORER

LAB6

test\_system.py

```
97     async def create_products(channel):
98         products = [
99             {
100                 "data": {
101                     "products": [
102                         {
103                             "product": "apple",
104                             "quantity": 10
105                         }
106                     ]
107                 }
108             }
109         ]
110
111         for product in products:
112             await send_message(channel, "product", product)
113             await asyncio.sleep(0.5)
114
115     async def create_orders(channel):
116         """Создание 3 заказов"""
117         print("Создание заказов")
118
119         orders = [
120             {
121                 "action": "create",
122                 "data": {
123                     "user_id": 1,
124                     "items": [
125                         {"product_id": 1, "quantity": 1},
126                         {"product_id": 2, "quantity": 2},
127                     ]
128                 }
129             },
130             {
131                 "action": "create",
132                 "data": {
133                     "user_id": 1,
134                     "items": [
135                         {"product_id": 3, "quantity": 1},
136                         {"product_id": 4, "quantity": 3},
137                     ]
138                 }
139             },
140             {
141                 "action": "create",
142                 "data": {
143                     "user_id": 1,
144                     "items": [
145                         {"product_id": 3, "quantity": 1},
146                         {"product_id": 5, "quantity": 2},
147                     ]
148                 }
149             }
150         ]
151
152         for i, order in enumerate(orders, 1):
153             print(f"\nЗаказ {i}:")
154             await send_message(channel, "order", order)
155             await asyncio.sleep(1)
156
157     async def test_product_operations(channel):
158         """Тестирование операций с продуктами"""
159         print("Тестирование операций с продуктами")
```

uv.lock

OUTLINE

TIMELINE

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 Lab6

EXPLORER

LAB6

test\_system.py

```
149     async def create_orders(channel):
150         orders = [
151             {
152                 "action": "create",
153                 "data": {
154                     "user_id": 1,
155                     "items": [
156                         {"product_id": 1, "quantity": 1},
157                         {"product_id": 2, "quantity": 2},
158                     ]
159                 }
160             },
161             {
162                 "action": "create",
163                 "data": {
164                     "user_id": 1,
165                     "items": [
166                         {"product_id": 3, "quantity": 1},
167                         {"product_id": 4, "quantity": 3},
168                     ]
169                 }
170             },
171             {
172                 "action": "create",
173                 "data": {
174                     "user_id": 1,
175                     "items": [
176                         {"product_id": 3, "quantity": 1},
177                         {"product_id": 5, "quantity": 2},
178                     ]
179                 }
180             },
181             {
182                 "action": "create",
183                 "data": {
184                     "user_id": 1,
185                     "items": [
186                         {"product_id": 2, "quantity": 1},
187                         {"product_id": 5, "quantity": 2},
188                     ]
189                 }
190             }
191         ]
192
193         for i, order in enumerate(orders, 1):
194             print(f"\nЗаказ {i}:")
195             await send_message(channel, "order", order)
196             await asyncio.sleep(1)
197
198     async def test_product_operations(channel):
199         """Тестирование операций с продуктами"""
200         print("Тестирование операций с продуктами")
```

uv.lock

OUTLINE

TIMELINE

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 Lab6

EXPLORER

LAB6

- pytest\_cache
- .venv
- alembic
- src
  - \_pycache\_
  - controllers
  - messaging
  - models
  - repositories
  - schemas
  - services
  - utils
- \_\_init\_\_.py
- Dockerfile
- entrypoint.sh
- main.py
- tests
- .dockerignore
- .gitignore
- python-version
- alembic.ini
- docker-compose.yml
- pyproject.toml
- README.md
- test\_system.py
- uv.lock

OUTLINE

TIMELINE

test\_system.py

```
149     async def create_orders(channel):  
150         """Тестирование операций с продуктами"""  
151         print("Тестирование операций с продуктами")  
152         await send_message(  
153             channel,  
154             "product",  
155             {  
156                 "action": "update",  
157                 "data": {  
158                     "id": 1,  
159                     "price": 1199.99  
160                 }  
161             }  
162         )  
163         await asyncio.sleep(0.5)  
164  
165         print("\n2. Пометка продукта #5 как закончившегося:")  
166         await send_message(  
167             channel,  
168             "product",  
169             {  
170                 "action": "mark_out_of_stock",  
171                 "data": {  
172                     "id": 5  
173                 }  
174             }  
175         )  
176         await asyncio.sleep(0.5)  
177  
178     async def test_order_operations(channel):  
179         """Тестирование операций с заказами"""  
180         print("Тестирование операций с заказами")  
181  
182         print("\n1. Обновление цены продукта #1: ")  
183         await send_message(  
184             channel,  
185             "product",  
186             {  
187                 "action": "update",  
188                 "data": {  
189                     "id": 1,  
190                     "price": 1199.99  
191                 }  
192             }  
193         )  
194         await asyncio.sleep(0.5)  
195  
196         print("\n2. Пометка продукта #5 как закончившегося:")  
197         await send_message(  
198             channel,  
199             "product",  
200             {  
201                 "action": "mark_out_of_stock",  
202                 "data": {  
203                     "id": 5  
204                 }  
205             }  
206         )  
207         await asyncio.sleep(0.5)  
208  
209         print("\n3. Обновление статуса заказа #1 -> processing:")  
210         await send_message(  
211             channel,  
212             "order",  
213             {  
214                 "action": "update_status",  
215                 "data": {  
216                     "order_id": 1,  
217                     "status": "processing"  
218                 }  
219             }  
220         )  
221         await asyncio.sleep(0.5)  
222  
223         print("\n4. Обновление статуса заказа #2 -> shipped:")  
224         await send_message(  
225             channel,  
226             "order",  
227             {  
228                 "action": "update_status",  
229                 "data": {  
230                     "order_id": 2,  
231                     "status": "shipped"  
232                 }  
233             }  
234         )  
235         await asyncio.sleep(0.5)  
236  
237         print("\n5. Обновление статуса заказа #3 -> delivered:")  
238         await send_message(  
239             channel,  
240             "order",  
241             {  
242                 "action": "update_status",  
243                 "data": {  
244                     "order_id": 3,  
245                     "status": "delivered"  
246                 }  
247             }  
248         )  
249         await asyncio.sleep(0.5)  
250  
251         print("\n6. Обновление статуса заказа #4 -> delivered:")  
252         await send_message(  
253             channel,  
254             "order",  
255             {  
256                 "action": "update_status",  
257                 "data": {  
258                     "order_id": 4,  
259                     "status": "delivered"  
260                 }  
261             }  
262         )  
263         await asyncio.sleep(0.5)  
264  
265         print("\n7. Обновление статуса заказа #5 -> delivered:")  
266         await send_message(  
267             channel,  
268             "order",  
269             {  
270                 "action": "update_status",  
271                 "data": {  
272                     "order_id": 5,  
273                     "status": "delivered"  
274                 }  
275             }  
276         )  
277         await asyncio.sleep(0.5)  
278  
279         print("\n8. Обновление статуса заказа #6 -> delivered:")  
280         await send_message(  
281             channel,  
282             "order",  
283             {  
284                 "action": "update_status",  
285                 "data": {  
286                     "order_id": 6,  
287                     "status": "delivered"  
288                 }  
289             }  
290         )  
291         await asyncio.sleep(0.5)  
292  
293         print("\n9. Обновление статуса заказа #7 -> delivered:")  
294         await send_message(  
295             channel,  
296             "order",  
297             {  
298                 "action": "update_status",  
299                 "data": {  
300                     "order_id": 7,  
301                     "status": "delivered"  
302                 }  
303             }  
304         )  
305         await asyncio.sleep(0.5)  
306  
307         print("\n10. Обновление статуса заказа #8 -> delivered:")  
308         await send_message(  
309             channel,  
310             "order",  
311             {  
312                 "action": "update_status",  
313                 "data": {  
314                     "order_id": 8,  
315                     "status": "delivered"  
316                 }  
317             }  
318         )  
319         await asyncio.sleep(0.5)  
320  
321         print("\n11. Обновление статуса заказа #9 -> delivered:")  
322         await send_message(  
323             channel,  
324             "order",  
325             {  
326                 "action": "update_status",  
327                 "data": {  
328                     "order_id": 9,  
329                     "status": "delivered"  
330                 }  
331             }  
332         )  
333         await asyncio.sleep(0.5)  
334  
335         print("\n12. Обновление статуса заказа #10 -> delivered:")  
336         await send_message(  
337             channel,  
338             "order",  
339             {  
340                 "action": "update_status",  
341                 "data": {  
342                     "order_id": 10,  
343                     "status": "delivered"  
344                 }  
345             }  
346         )  
347         await asyncio.sleep(0.5)  
348  
349         print("\n13. Обновление статуса заказа #11 -> delivered:")  
350         await send_message(  
351             channel,  
352             "order",  
353             {  
354                 "action": "update_status",  
355                 "data": {  
356                     "order_id": 11,  
357                     "status": "delivered"  
358                 }  
359             }  
360         )  
361         await asyncio.sleep(0.5)  
362  
363         print("\n14. Обновление статуса заказа #12 -> delivered:")  
364         await send_message(  
365             channel,  
366             "order",  
367             {  
368                 "action": "update_status",  
369                 "data": {  
370                     "order_id": 12,  
371                     "status": "delivered"  
372                 }  
373             }  
374         )  
375         await asyncio.sleep(0.5)  
376  
377         print("\n15. Обновление статуса заказа #13 -> delivered:")  
378         await send_message(  
379             channel,  
380             "order",  
381             {  
382                 "action": "update_status",  
383                 "data": {  
384                     "order_id": 13,  
385                     "status": "delivered"  
386                 }  
387             }  
388         )  
389         await asyncio.sleep(0.5)  
390  
391         print("\n16. Обновление статуса заказа #14 -> delivered:")  
392         await send_message(  
393             channel,  
394             "order",  
395             {  
396                 "action": "update_status",  
397                 "data": {  
398                     "order_id": 14,  
399                     "status": "delivered"  
400                 }  
401             }  
402         )  
403         await asyncio.sleep(0.5)  
404  
405         print("\n17. Обновление статуса заказа #15 -> delivered:")  
406         await send_message(  
407             channel,  
408             "order",  
409             {  
410                 "action": "update_status",  
411                 "data": {  
412                     "order_id": 15,  
413                     "status": "delivered"  
414                 }  
415             }  
416         )  
417         await asyncio.sleep(0.5)  
418  
419         print("\n18. Обновление статуса заказа #16 -> delivered:")  
420         await send_message(  
421             channel,  
422             "order",  
423             {  
424                 "action": "update_status",  
425                 "data": {  
426                     "order_id": 16,  
427                     "status": "delivered"  
428                 }  
429             }  
430         )  
431         await asyncio.sleep(0.5)  
432  
433         print("\n19. Обновление статуса заказа #17 -> delivered:")  
434         await send_message(  
435             channel,  
436             "order",  
437             {  
438                 "action": "update_status",  
439                 "data": {  
440                     "order_id": 17,  
441                     "status": "delivered"  
442                 }  
443             }  
444         )  
445         await asyncio.sleep(0.5)  
446  
447         print("\n20. Обновление статуса заказа #18 -> delivered:")  
448         await send_message(  
449             channel,  
450             "order",  
451             {  
452                 "action": "update_status",  
453                 "data": {  
454                     "order_id": 18,  
455                     "status": "delivered"  
456                 }  
457             }  
458         )  
459         await asyncio.sleep(0.5)  
460  
461         print("\n21. Обновление статуса заказа #19 -> delivered:")  
462         await send_message(  
463             channel,  
464             "order",  
465             {  
466                 "action": "update_status",  
467                 "data": {  
468                     "order_id": 19,  
469                     "status": "delivered"  
470                 }  
471             }  
472         )  
473         await asyncio.sleep(0.5)  
474  
475         print("\n22. Обновление статуса заказа #20 -> delivered:")  
476         await send_message(  
477             channel,  
478             "order",  
479             {  
480                 "action": "update_status",  
481                 "data": {  
482                     "order_id": 20,  
483                     "status": "delivered"  
484                 }  
485             }  
486         )  
487         await asyncio.sleep(0.5)  
488  
489         print("\n23. Обновление статуса заказа #21 -> delivered:")  
490         await send_message(  
491             channel,  
492             "order",  
493             {  
494                 "action": "update_status",  
495                 "data": {  
496                     "order_id": 21,  
497                     "status": "delivered"  
498                 }  
499             }  
500         )  
501         await asyncio.sleep(0.5)  
502  
503         print("\n24. Обновление статуса заказа #22 -> delivered:")  
504         await send_message(  
505             channel,  
506             "order",  
507             {  
508                 "action": "update_status",  
509                 "data": {  
510                     "order_id": 22,  
511                     "status": "delivered"  
512                 }  
513             }  
514         )  
515         await asyncio.sleep(0.5)  
516  
517         print("\n25. Обновление статуса заказа #23 -> delivered:")  
518         await send_message(  
519             channel,  
520             "order",  
521             {  
522                 "action": "update_status",  
523                 "data": {  
524                     "order_id": 23,  
525                     "status": "delivered"  
526                 }  
527             }  
528         )  
529         await asyncio.sleep(0.5)  
530  
531         print("\n26. Обновление статуса заказа #24 -> delivered:")  
532         await send_message(  
533             channel,  
534             "order",  
535             {  
536                 "action": "update_status",  
537                 "data": {  
538                     "order_id": 24,  
539                     "status": "delivered"  
540                 }  
541             }  
542         )  
543         await asyncio.sleep(0.5)  
544  
545         print("\n27. Обновление статуса заказа #25 -> delivered:")  
546         await send_message(  
547             channel,  
548             "order",  
549             {  
550                 "action": "update_status",  
551                 "data": {  
552                     "order_id": 25,  
553                     "status": "delivered"  
554                 }  
555             }  
556         )  
557         await asyncio.sleep(0.5)  
558  
559         print("\n28. Обновление статуса заказа #26 -> delivered:")  
560         await send_message(  
561             channel,  
562             "order",  
563             {  
564                 "action": "update_status",  
565                 "data": {  
566                     "order_id": 26,  
567                     "status": "delivered"  
568                 }  
569             }  
570         )  
571         await asyncio.sleep(0.5)  
572  
573         print("\n29. Обновление статуса заказа #27 -> delivered:")  
574         await send_message(  
575             channel,  
576             "order",  
577             {  
578                 "action": "update_status",  
579                 "data": {  
580                     "order_id": 27,  
581                     "status": "delivered"  
582                 }  
583             }  
584         )  
585         await asyncio.sleep(0.5)  
586  
587         print("\n30. Обновление статуса заказа #28 -> delivered:")  
588         await send_message(  
589             channel,  
590             "order",  
591             {  
592                 "action": "update_status",  
593                 "data": {  
594                     "order_id": 28,  
595                     "status": "delivered"  
596                 }  
597             }  
598         )  
599         await asyncio.sleep(0.5)  
600  
601         print("\n31. Обновление статуса заказа #29 -> delivered:")  
602         await send_message(  
603             channel,  
604             "order",  
605             {  
606                 "action": "update_status",  
607                 "data": {  
608                     "order_id": 29,  
609                     "status": "delivered"  
610                 }  
611             }  
612         )  
613         await asyncio.sleep(0.5)  
614  
615         print("\n32. Обновление статуса заказа #30 -> delivered:")  
616         await send_message(  
617             channel,  
618             "order",  
619             {  
620                 "action": "update_status",  
621                 "data": {  
622                     "order_id": 30,  
623                     "status": "delivered"  
624                 }  
625             }  
626         )  
627         await asyncio.sleep(0.5)  
628  
629         print("\n33. Обновление статуса заказа #31 -> delivered:")  
630         await send_message(  
631             channel,  
632             "order",  
633             {  
634                 "action": "update_status",  
635                 "data": {  
636                     "order_id": 31,  
637                     "status": "delivered"  
638                 }  
639             }  
640         )  
641         await asyncio.sleep(0.5)  
642  
643         print("\n34. Обновление статуса заказа #32 -> delivered:")  
644         await send_message(  
645             channel,  
646             "order",  
647             {  
648                 "action": "update_status",  
649                 "data": {  
650                     "order_id": 32,  
651                     "status": "delivered"  
652                 }  
653             }  
654         )  
655         await asyncio.sleep(0.5)  
656  
657         print("\n35. Обновление статуса заказа #33 -> delivered:")  
658         await send_message(  
659             channel,  
660             "order",  
661             {  
662                 "action": "update_status",  
663                 "data": {  
664                     "order_id": 33,  
665                     "status": "delivered"  
666                 }  
667             }  
668         )  
669         await asyncio.sleep(0.5)  
670  
671         print("\n36. Обновление статуса заказа #34 -> delivered:")  
672         await send_message(  
673             channel,  
674             "order",  
675             {  
676                 "action": "update_status",  
677                 "data": {  
678                     "order_id": 34,  
679                     "status": "delivered"  
680                 }  
681             }  
682         )  
683         await asyncio.sleep(0.5)  
684  
685         print("\n37. Обновление статуса заказа #35 -> delivered:")  
686         await send_message(  
687             channel,  
688             "order",  
689             {  
690                 "action": "update_status",  
691                 "data": {  
692                     "order_id": 35,  
693                     "status": "delivered"  
694                 }  
695             }  
696         )  
697         await asyncio.sleep(0.5)  
698  
699         print("\n38. Обновление статуса заказа #36 -> delivered:")  
700         await send_message(  
701             channel,  
702             "order",  
703             {  
704                 "action": "update_status",  
705                 "data": {  
706                     "order_id": 36,  
707                     "status": "delivered"  
708                 }  
709             }  
710         )  
711         await asyncio.sleep(0.5)  
712  
713         print("\n39. Обновление статуса заказа #37 -> delivered:")  
714         await send_message(  
715             channel,  
716             "order",  
717             {  
718                 "action": "update_status",  
719                 "data": {  
720                     "order_id": 37,  
721                     "status": "delivered"  
722                 }  
723             }  
724         )  
725         await asyncio.sleep(0.5)  
726  
727         print("\n40. Обновление статуса заказа #38 -> delivered:")  
728         await send_message(  
729             channel,  
730             "order",  
731             {  
732                 "action": "update_status",  
733                 "data": {  
734                     "order_id": 38,  
735                     "status": "delivered"  
736                 }  
737             }  
738         )  
739         await asyncio.sleep(0.5)  
740  
741         print("\n41. Обновление статуса заказа #39 -> delivered:")  
742         await send_message(  
743             channel,  
744             "order",  
745             {  
746                 "action": "update_status",  
747                 "data": {  
748                     "order_id": 39,  
749                     "status": "delivered"  
750                 }  
751             }  
752         )  
753         await asyncio.sleep(0.5)  
754  
755         print("\n42. Обновление статуса заказа #40 -> delivered:")  
756         await send_message(  
757             channel,  
758             "order",  
759             {  
760                 "action": "update_status",  
761                 "data": {  
762                     "order_id": 40,  
763                     "status": "delivered"  
764                 }  
765             }  
766         )  
767         await asyncio.sleep(0.5)  
768  
769         print("\n43. Обновление статуса заказа #41 -> delivered:")  
770         await send_message(  
771             channel,  
772             "order",  
773             {  
774                 "action": "update_status",  
775                 "data": {  
776                     "order_id": 41,  
777                     "status": "delivered"  
778                 }  
779             }  
780         )  
781         await asyncio.sleep(0.5)  
782  
783         print("\n44. Обновление статуса заказа #42 -> delivered:")  
784         await send_message(  
785             channel,  
786             "order",  
787             {  
788                 "action": "update_status",  
789                 "data": {  
790                     "order_id": 42,  
791                     "status": "delivered"  
792                 }  
793             }  
794         )  
795         await asyncio.sleep(0.5)  
796  
797         print("\n45. Обновление статуса заказа #43 -> delivered:")  
798         await send_message(  
799             channel,  
800             "order",  
801             {  
802                 "action": "update_status",  
803                 "data": {  
804                     "order_id": 43,  
805                     "status": "delivered"  
806                 }  
807             }  
808         )  
809         await asyncio.sleep(0.5)  
810  
811         print("\n46. Обновление статуса заказа #44 -> delivered:")  
812         await send_message(  
813             channel,  
814             "order",  
815             {  
816                 "action": "update_status",  
817                 "data": {  
818                     "order_id": 44,  
819                     "status": "delivered"  
820                 }  
821             }  
822         )  
823         await asyncio.sleep(0.5)  
824  
825         print("\n47. Обновление статуса заказа #45 -> delivered:")  
826         await send_message(  
827             channel,  
828             "order",  
829             {  
830                 "action": "update_status",  
831                 "data": {  
832                     "order_id": 45,  
833                     "status": "delivered"  
834                 }  
835             }  
836         )  
837         await asyncio.sleep(0.5)  
838  
839         print("\n48. Обновление статуса заказа #46 -> delivered:")  
840         await send_message(  
841             channel,  
842             "order",  
843             {  
844                 "action": "update_status",  
845                 "data": {  
846                     "order_id": 46,  
847                     "status": "delivered"  
848                 }  
849             }  
850         )  
851         await asyncio.sleep(0.5)  
852  
853         print("\n49. Обновление статуса заказа #47 -> delivered:")  
854         await send_message(  
855             channel,  
856             "order",  
857             {  
858                 "action": "update_status",  
859                 "data": {  
860                     "order_id": 47,  
861                     "status": "delivered"  
862                 }  
863             }  
864         )  
865         await asyncio.sleep(0.5)  
866  
867         print("\n50. Обновление статуса заказа #48 -> delivered:")  
868         await send_message(  
869             channel,  
870             "order",  
871             {  
872                 "action": "update_status",  
873                 "data": {  
874                     "order_id": 48,  
875                     "status": "delivered"  
876                 }  
877             }  
878         )  
879         await asyncio.sleep(0.5)  
880  
881         print("\n51. Обновление статуса заказа #49 -> delivered:")  
882         await send_message(  
883             channel,  
884             "order",  
885             {  
886                 "action": "update_status",  
887                 "data": {  
888                     "order_id": 49,  
889                     "status": "delivered"  
890                 }  
891             }  
892         )  
893         await asyncio.sleep(0.5)  
894  
895         print("\n52. Обновление статуса заказа #50 -> delivered:")  
896         await send_message(  
897             channel,  
898             "order",  
899             {  
900                 "action": "update_status",  
901                 "data": {  
902                     "order_id": 50,  
903                     "status": "delivered"  
904                 }  
905             }  
906         )  
907         await asyncio.sleep(0.5)  
908  
909         print("\n53. Обновление статуса заказа #51 -> delivered:")  
910         await send_message(  
911             channel,  
912             "order",  
913             {  
914                 "action": "update_status",  
915                 "data": {  
916                     "order_id": 51,  
917                     "status": "delivered"  
918                 }  
919             }  
920         )  
921         await asyncio.sleep(0.5)  
922  
923         print("\n54. Обновление статуса заказа #52 -> delivered:")  
924         await send_message(  
925             channel,  
926             "order",  
927             {  
928                 "action": "update_status",  
929                 "data": {  
930                     "order_id": 52,  
931                     "status": "delivered"  
932                 }  
933             }  
934         )  
935         await asyncio.sleep(0.5)  
936  
937         print("\n55. Обновление статуса заказа #53 -> delivered:")  
938         await send_message(  
939             channel,  
940             "order",  
941             {  
942                 "action": "update_status",  
943                 "data": {  
944                     "order_id": 53,  
945                     "status": "delivered"  
946                 }  
947             }  
948         )  
949         await asyncio.sleep(0.5)  
950  
951         print("\n56. Обновление статуса заказа #54 -> delivered:")  
952         await send_message(  
953             channel,  
954             "order",  
955             {  
956                 "action": "update_status",  
957                 "data": {  
958                     "order_id": 54,  
959                     "status": "delivered"  
960                 }  
961             }  
962         )  
963         await asyncio.sleep(0.5)  
964  
965         print("\n57. Обновление статуса заказа #55 -> delivered:")  
966         await send_message(  
967             channel,  
968             "order",  
969             {  
970                 "action": "update_status",  
971                 "data": {  
972                     "order_id": 55,  
973                     "status": "delivered"  
974                 }  
975             }  
976         )  
977         await asyncio.sleep(0.5)  
978  
979         print("\n58. Обновление статуса заказа #56 -> delivered:")  
980         await send_message(  
981             channel,  
982             "order",  
983             {  
984                 "action": "update_status",  
985                 "data": {  
986                     "order_id": 56,  
987                     "status": "delivered"  
988                 }  
989             }  
990         )  
991         await asyncio.sleep(0.5)  
992  
993         print("\n59. Обновление статуса заказа #57 -> delivered:")  
994         await send_message(  
995             channel,  
996             "order",  
997             {  
998                 "action": "update_status",  
999                 "data": {  
1000                     "order_id": 57,  
1001                     "status": "delivered"  
1002                 }  
1003             }  
1004         )  
1005         await asyncio.sleep(0.5)  
1006  
1007         print("\n60. Обновление статуса заказа #58 -> delivered:")  
1008         await send_message(  
1009             channel,  
1010             "order",  
1011             {  
1012                 "action": "update_status",  
1013                 "data": {  
1014                     "order_id": 58,  
1015                     "status": "delivered"  
1016                 }  
1017             }  
1018         )  
1019         await asyncio.sleep(0.5)  
1020  
1021         print("\n61. Обновление статуса заказа #59 -> delivered:")  
1022         await send_message(  
1023             channel,  
1024             "order",  
1025             {  
1026                 "action": "update_status",  
1027                 "data": {  
1028                     "order_id": 59,  
1029                     "status": "delivered"  
1030                 }  
1031             }  
1032         )  
1033         await asyncio.sleep(0.5)  
1034  
1035         print("\n62. Обновление статуса заказа #60 -> delivered:")  
1036         await send_message(  
1037             channel,  
1038             "order",  
1039             {  
1040                 "action": "update_status",  
1041                 "data": {  
1042                     "order_id": 60,  
1043                     "status": "delivered"  
1044                 }  
1045             }  
1046         )  
1047         await asyncio.sleep(0.5)  
1048  
1049         print("\n63. Обновление статуса заказа #61 -> delivered:")  
1050         await send_message(  
1051             channel,  
1052             "order",  
1053             {  
1054                 "action": "update_status",  
1055                 "data": {  
1056                     "order_id": 61,  
1057                     "status": "delivered"  
1058                 }  
1059             }  
1060         )  
1061         await asyncio.sleep(0.5)  
1062  
1063         print("\n64. Обновление статуса заказа #62 -> delivered:")  
1064         await send_message(  
1065             channel,  
1066             "order",  
1067             {  
1068                 "action": "update_status",  
1069                 "data": {  
1070                     "order_id": 62,  
1071                     "status": "delivered"  
1072                 }  
1073             }  
1074         )  
1075         await asyncio.sleep(0.5)  
1076  
1077         print("\n65. Обновление статуса заказа #63 -> delivered:")  
1078         await send_message(  
1079             channel,  
1080             "order",  
1081             {  
1082                 "action": "update_status",  
1083                 "data": {  
1084                     "order
```

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 Lab6

EXPLORER

- LAB6
  - \_\_pycache\_\_
  - .venv
  - alembic
  - src
    - \_\_pycache\_\_
    - controllers
    - messaging
    - models
    - repositories
    - schemas
    - services
    - utils
  - \_\_init\_\_.py
  - Dockerfile
  - entrypoint.sh
  - main.py
  - tests
  - .dockerignore
  - .gitignore
  - python-version
  - alembic.ini
  - docker-compose.yml
  - pyproject.toml
  - README.md
  - test\_system.py
  - uv.lock
- OUTLINE
- TIMELINE

test\_system.py

```
192     async def test_product_operations(channel):
193         await send_message(
194             {
195                 "action": "mark_out_of_stock",
196                 "data": {
197                     "id": 5
198                 }
199             }
200         )
201         await asyncio.sleep(0.5)
202
203     async def test_order_operations(channel):
204         """тестирование операций с заказами"""
205         print("тестирование операций с заказами")
206
207         print("\n1. Обновление статуса заказа #1 -> processing:")
208         await send_message(
209             channel,
210             {
211                 "order": {
212                     "action": "update_status",
213                     "data": {
214                         "order_id": 1,
215                         "status": "processing"
216                     }
217                 }
218             }
219         )
220         await asyncio.sleep(0.5)
221
222         print("\n2. Обновление статуса заказа #2 -> shipped:")
223         await send_message(
224             channel,
225             {
226                 "order": {
227                     "action": "update_status",
228                     "data": {
229                         "order_id": 2,
230                         "status": "shipped"
231                     }
232                 }
233             }
234         )
235         await asyncio.sleep(0.5)
236
237     print("\n3. Обновление статуса заказа #3 -> canceled:")
238     await send_message(
239         channel,
240         {
241             "order": {
242                 "action": "update_status",
243                 "data": {
244                     "order_id": 3,
245                     "status": "canceled"
246                 }
247             }
248         }
249     )
250
251     await asyncio.sleep(0.5)
```

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 Lab6

EXPLORER

- LAB6
  - \_\_pycache\_\_
  - .venv
  - alembic
  - src
    - \_\_pycache\_\_
    - controllers
    - messaging
    - models
    - repositories
    - schemas
    - services
    - utils
  - \_\_init\_\_.py
  - Dockerfile
  - entrypoint.sh
  - main.py
  - tests
  - .dockerignore
  - .gitignore
  - python-version
  - alembic.ini
  - docker-compose.yml
  - pyproject.toml
  - README.md
  - test\_system.py
  - uv.lock
- OUTLINE
- TIMELINE

test\_system.py

```
192     async def test_product_operations(channel):
193         await send_message(
194             {
195                 "action": "mark_out_of_stock",
196                 "data": {
197                     "id": 5
198                 }
199             }
200         )
201         await asyncio.sleep(0.5)
202
203     async def test_order_operations(channel):
204         """тестирование операций с заказами"""
205         print("тестирование операций с заказами")
206
207         print("\n1. Обновление статуса заказа #1 -> processing:")
208         await send_message(
209             channel,
210             {
211                 "order": {
212                     "action": "update_status",
213                     "data": {
214                         "order_id": 1,
215                         "status": "processing"
216                     }
217                 }
218             }
219         )
220         await asyncio.sleep(0.5)
221
222         print("\n2. Обновление статуса заказа #2 -> shipped:")
223         await send_message(
224             channel,
225             {
226                 "order": {
227                     "action": "update_status",
228                     "data": {
229                         "order_id": 2,
230                         "status": "shipped"
231                     }
232                 }
233             }
234         )
235         await asyncio.sleep(0.5)
236
237     print("\n3. Обновление статуса заказа #3 -> canceled:")
238     await send_message(
239         channel,
240         {
241             "order": {
242                 "action": "update_status",
243                 "data": {
244                     "order_id": 3,
245                     "status": "canceled"
246                 }
247             }
248         }
249     )
250
251     await asyncio.sleep(0.5)
```

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python

```
File Edit Selection View Go Run Terminal Help < - > Q Lab6
EXPLORER LAB6
    > .pytest_cache
    > venv
    > alembic
        > __pycache__
        > controllers
        > messaging
        > models
        > repositories
        > schemas
        > services
        > utils
    > __init__.py
    > Dockerfile
    $ entrypoint.sh
    & main.py
    > tests
        > dockerignore
        > .gitignore
        & python-version
        & alembic.ini
        & docker-compose.yml
        & pyproject.toml
        & README.md
    & test_system.py
    & uv.lock
    > OUTLINE
    > TIMELINE
    < main* ⚡ 0 △ 0
test_system.py U X
224 & async def test_order_operations(channel):
241
242     await asyncio.sleep(0.5)
243
244     print("\n2. Обновление статуса заказа #2 -> shipped:")
245     await send_message(
246         channel,
247         "order",
248         {
249             "action": "update_status",
250             "data": {
251                 "order_id": 2,
252                 "status": "shipped"
253             }
254         }
255     )
256     await asyncio.sleep(0.5)
257
258     print("\n3. Отмена заказа #3 (товары вернутся на склад):")
259     await send_message(
260         channel,
261         "order",
262         {
263             "action": "update_status",
264             "data": {
265                 "order_id": 3,
266                 "status": "cancelled"
267             }
268         }
269     )
270     await asyncio.sleep(0.5)
271
272     async def test_insufficient_stock(channel):
273         """Тестирование попытки заказа с недостаточным количеством товара"""
274         print("Тестируем: недостаточно товара")
275
276         print("\nПопытка заказать 100 единиц продукта #5 (на складе только 5):")
277         await send_message(
278             channel,
279             "order",
280             {
281                 "action": "create",
282                 "data": {
283                     "user_id": 1,
284                     "items": [
285                         {"product_id": 5, "quantity": 100}
286                     ]
287                 }
288             }
289         )
290         print("Ожидается ошибка: 'Insufficient stock'")

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python
```

```
File Edit Selection View Go Run Terminal Help < - > Q Lab6
EXPLORER LAB6
    > .pytest_cache
    > venv
    > alembic
        > __pycache__
        > controllers
        > messaging
        > models
        > repositories
        > schemas
        > services
        > utils
    > __init__.py
    > Dockerfile
    $ entrypoint.sh
    & main.py
    > tests
        > dockerignore
        > .gitignore
        & python-version
        & alembic.ini
        & docker-compose.yml
        & pyproject.toml
        & README.md
    & test_system.py
    & uv.lock
    > OUTLINE
    > TIMELINE
    < main* ⚡ 0 △ 0
test_system.py U X
224 & async def test_order_operations(channel):
241
242     await send_message(
243         channel,
244         "order",
245         {
246             "action": "update_status",
247             "data": {
248                 "order_id": 2,
249                 "status": "shipped"
250             }
251         }
252     )
253     await asyncio.sleep(0.5)
254
255     async def test_insufficient_stock(channel):
256         """Тестирование попытки заказа с недостаточным количеством товара"""
257         print("Тестируем: недостаточно товара")
258
259         print("\nПопытка заказать 100 единиц продукта #5 (на складе только 5):")
260         await send_message(
261             channel,
262             "order",
263             {
264                 "action": "create",
265                 "data": {
266                     "user_id": 1,
267                     "items": [
268                         {"product_id": 5, "quantity": 100}
269                     ]
270                 }
271             }
272         )
273         print("Ожидается ошибка: 'Insufficient stock'")

292     async def run_rabbitmq_tests():
293         """Запуск тестов RabbitMQ"""
294         rabbitmq_url = os.getenv(
295             "RABBITMQ_URL",
296             "amqp://guest:guest@localhost:5672/"
297         )
298
299         print("Подключение к RABBITMQ")
300         print(f"URL: {rabbitmq_url}")

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python
```

```
File Edit Selection View Go Run Terminal Help < > Q Lab6
EXPLORER LAB6
    > .pytest_cache
    > .venv
    > alembic
        > __pycache__
        > controllers
        > messaging
        > models
        > repositories
        > schemas
        > services
        > utils
            & __init__.py
            & Dockerfile
            $ entrypoint.sh
            & main.py
        > tests
            & .dockerignore
            & .gitignore
            & python-version
            & alembic.ini
            & docker-compose.yml
            & pyproject.toml
            & README.md
            & test_system.py
            & uv.lock
    > OUTLINE
    > TIMELINE
test_system.py x
    test_system.py
    271 & async def test_insufficient_stock(channel):
    290
    291
    292     """Запуск тестов RabbitMQ"""
    293     rabbitmq_url = os.getenv(
    294         "RABBITMQ_URL",
    295         "amqp://guest:guest@localhost:5672/"
    296     )
    297
    298     print("Подключение к RABBITMQ")
    299     print(f"URL: {rabbitmq_url}")
    300
    301     try:
    302         connection = await connect_robust(rabbitmq_url)
    303         async with connection:
    304             channel = await connection.channel()
    305
    306             await channel.declare_queue("product", durable=True)
    307             await channel.declare_queue("order", durable=True)
    308
    309             print("Подключение установлено")
    310             print("Очереди обновлены")
    311
    312             await create_products(channel)
    313
    314             print("\n Ожидание обработки продуктов...")
    315             await asyncio.sleep(3)
    316
    317             await create_orders(channel)
    318
    319             await asyncio.sleep(2)
    320             await test_product_operations(channel)
    321
    322             await asyncio.sleep(2)
    323             await test_order_operations(channel)
    324
    325             await asyncio.sleep(2)
    326             await test_insufficient_stock(channel)
    327
    328             print("Все сообщения отправлены")
    329
    330     return True
    331
    332
    333 except Exception as e:
    334     print(f"\nОшибка подключения к RabbitMQ: {e}")
    335     print("Убедитесь, что RabbitMQ запущен:")
    336     print("docker-compose up -d rabbitmq")
    337     return False
    338
    339 # Главная функция
    340
```

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python

```
File Edit Selection View Go Run Terminal Help < > Q Lab6
EXPLORER LAB6
    > .pytest_cache
    > .venv
    > alembic
        > __pycache__
        > controllers
        > messaging
        > models
        > repositories
        > schemas
        > services
        > utils
            & __init__.py
            & Dockerfile
            $ entrypoint.sh
            & main.py
        > tests
            & .dockerignore
            & .gitignore
            & python-version
            & alembic.ini
            & docker-compose.yml
            & pyproject.toml
            & README.md
            & test_system.py
            & uv.lock
    > OUTLINE
    > TIMELINE
test_system.py x
    test_system.py
    292 & async def run_rabbitmq_tests():
    302     try:
    303         async with connection:
    304             channel = await connection.channel()
    305
    306             await channel.declare_queue("product", durable=True)
    307             await channel.declare_queue("order", durable=True)
    308
    309             print("Подключение установлено")
    310             print("Очереди обновлены")
    311
    312             await create_products(channel)
    313
    314             print("\n Ожидание обработки продуктов...")
    315             await asyncio.sleep(3)
    316
    317             await create_orders(channel)
    318
    319             await asyncio.sleep(2)
    320             await test_product_operations(channel)
    321
    322             await asyncio.sleep(2)
    323             await test_order_operations(channel)
    324
    325             await asyncio.sleep(2)
    326             await test_insufficient_stock(channel)
    327
    328             print("Все сообщения отправлены")
    329
    330     return True
    331
    332
    333 except Exception as e:
    334     print(f"\nОшибка подключения к RabbitMQ: {e}")
    335     print("Убедитесь, что RabbitMQ запущен:")
    336     print("docker-compose up -d rabbitmq")
    337     return False
    338
    339 # Главная функция
    340
```

Ln 83, Col 1 Spaces: 4 UTF-8 CRLF () Python

```
File Edit Selection View Go Run Terminal Help < > Q: Lab6
EXPLORER
LAB6
> .pytest_cache
> venv
> alembic
> src
>   > __pycache__
>   > controllers
>   > messaging
>   > models
>   > repositories
>   > schemas
>   > services
>   > utils
>   > __init__.py
> Dockerfile
$ entrypoint.sh
$ main.py
> tests
> .dockerignore
> .gitignore
> python-version
> alembic.ini
$ docker-compose.yml
$ pyproject.toml
$ README.md
$ test_system.py
$ uv.lock
332
333     except Exception as e:
334         print(f"\"{e}\")")
335         print("Убедитесь, что RabbitMQ запущен:")
336         print(" docker-compose up -d rabbitmq")
337         return False
338
339     async def main():
340         """Основная функция"""
341
342         print("Комплексное тестирование системы")
343
344         # 1. Проверка API
345         api_ok = await check_api_health()
346         if not api_ok:
347             print("\nЗапустите приложение перед выполнением этого скрипта!")
348             print(" docker-compose up -d")
349             return 1
350
351         # 2. Создание пользователя
352         user_id = await create_test_user()
353         if not user_id:
354             print("\nНе удалось создать тестового пользователя")
355             return 1
356
357         # 3. Запуск тестов RabbitMQ
358         rabbitmq_ok = await run_rabbitmq_tests()
359         if not rabbitmq_ok:
360             return 1
361
362
363         # Финальное сообщение
364         print("\n" + "=" * 70)
365         print("тестирование завершено")
366         print("=" * 70)
367         print(f"\nТестовый пользователь: ID = {user_id}")
368         print("\nПроверьте результаты:")
369
Ln 338, Col 1  Spaces: 4  UTF-8  CRLF  ()  Python  ⚙  🔍
```

Данный скрипт представляет собой комплексный тестовый клиент для проверки работы системы, состоящей из REST API (FastAPI) и асинхронного брокера сообщений RabbitMQ.

## Краткое описание функциональности:

Проверка доступности (check\_api\_health):

- Отправляет HTTP-запрос к эндпоинту /users для проверки работоспособности основного API.

Подготовка тестовых данных (create\_test\_user):

- Создает в системе тестового пользователя через API. Если пользователь уже существует — получает его ID.

Интеграционное тестирование через RabbitMQ (run\_rabbitmq\_tests):

- Устанавливает соединение с RabbitMQ и объявляет две очереди: product и order.

Сценарий тестирования:

- Создание товаров: отправляет в очередь product 5 сообщений для создания товаров.
- Создание заказов: отправляет в очередь order 3 сообщения для создания заказов от имени тестового пользователя.

- Тест операций с товарами: имитирует обновление цены товара и пометку его как отсутствующего на складе.
- Тест операций с заказами: имитирует изменение статусов заказов (в обработке, отправлен, отменен).
- Тест обработки ошибок: пытается создать заказ с количеством товара, превышающим остаток на складе, для проверки корректной валидации

```
(Lab6) PS C:\Users\Anastasia\Desktop\application-development-homework\Lab6> python .\test_system.py
Комплексное тестирование системы

=====
ПРОВЕРКА ДОСТУПНОСТИ API
=====

API доступен
создание тестового пользователя
Пользователь существует, получаем ID...
Найден существующий пользователь!
ID: 1
Подключение к RABBITMQ
URL: amqp://guest:guest@localhost:5672/
Подключение установлено
Очереди обявлены
Создание продуктов
Отправлено в 'product': {'action': 'create', 'data': {'name': 'Laptop Dell XPS 15', 'price': 1299.99, 'stock_quantity': 25}}
Отправлено в 'product': {'action': 'create', 'data': {'name': 'Wireless Mouse Logitech MX', 'price': 79.99, 'stock_quantity': 150}}
Отправлено в 'product': {'action': 'create', 'data': {'name': 'Mechanical Keyboard', 'price': 149.99, 'stock_quantity': 75}}
Отправлено в 'product': {'action': 'create', 'data': {'name': 'USB-C Hub 7-in-1', 'price': 49.99, 'stock_quantity': 200}}
Отправлено в 'product': {'action': 'create', 'data': {'name': 'Webcam Logitech C920', 'price': 89.99, 'stock_quantity': 5}}
Ожидание обработки продуктов...

Создание заказов

Заказ #1:
Отправлено в 'order': {'action': 'create', 'data': {'user_id': 1, 'items': [{'product_id': 1, 'quantity': 1}, {'product_id': 2, 'quantity': 2]}]}

Заказ #2:
Отправлено в 'order': {'action': 'create', 'data': {'user_id': 1, 'items': [{'product_id': 3, 'quantity': 1}, {'product_id': 4, 'quantity': 3]}]}

Заказ #3:
Отправлено в 'order': {'action': 'create', 'data': {'user_id': 1, 'items': [{'product_id': 2, 'quantity': 1}, {'product_id': 5, 'quantity': 2]}]}
Тестирование операций с продуктами

1. Обновление цены продукта #1:
Отправлено в 'product': {'action': 'update', 'data': {'id': 1, 'price': 1199.99}}

2. Пометка продукта #5 как закончившегося:
Отправлено в 'product': {'action': 'mark_out_of_stock', 'data': {'id': 5}}
Тестирование операций с заказами

1. Обновление статуса заказа #1 -> processing:
Отправлено в 'order': {'action': 'update_status', 'data': {'order_id': 1, 'status': 'processing'}}

2. Обновление статуса заказа #2 -> shipped:
Отправлено в 'order': {'action': 'update_status', 'data': {'order_id': 2, 'status': 'shipped'}}

3. Отмена заказа #3 (товары вернутся на склад):
Отправлено в 'order': {'action': 'update_status', 'data': {'order_id': 3, 'status': 'cancelled'}}
Тестирование: недостаточно товара
```

```
3. Отмена заказа #3 (товары вернутся на склад):
Отправлено в 'order': {'action': 'update_status', 'data': {'order_id': 3, 'status': 'cancelled'}}
Тестирование: недостаточно товара

Попытка заказать 100 единиц продукта #5 (на складе только 5):
Отправлено в 'order': {'action': 'create', 'data': {'user_id': 1, 'items': [{'product_id': 5, 'quantity': 100}]}}
Ожидается ошибка: 'Insufficient stock'
Все сообщения отправлены

=====
Тестирование завершено
=====

Тестовый пользователь: ID = 1
```

**Результат: 5 продуктов + 3 заказа отправлены в очередь RabbitMQ**

### Общий поток данных

#### Асинхронный поток (через RabbitMQ)

RabbitMQ

- FastStream
- Queue Handler
- Service
- Repository
- PostgreSQL

#### Синхронный поток (через HTTP)

HTTP Client

- Litestar Controller
- Service
- Repository
- PostgreSQL

### Бизнес-логика обработки заказов

#### Сценарий 1: Успешное создание заказа

- Приходит сообщение в очередь order с action create
- Проверяется существование пользователя
- Проверяется существование всех продуктов
- Проверяется наличие товара на складе для каждой позиции

Если всё хорошо:

- Создается заказ со статусом pending
- Товары списываются со склада
- Рассчитывается общая сумма заказа

### **Сценарий 2: Недостаточно товара на складе**

- Приходит заказ с quantity: 10 для продукта, у которого stock\_quantity: 5
- Выбрасывается ошибка "Insufficient stock"
- Заказ НЕ создается
- Товары НЕ списываются

### **Сценарий 3: Отмена заказа**

- Приходит сообщение с action update\_status и status: "cancelled"
- Находится заказ
- Для каждой позиции заказа товары возвращаются на склад
- Статус заказа меняется на cancelled

### **Сценарий 4: Продукт закончился на складе**

- Администратор отправляет сообщение с action mark\_out\_of\_stock
- У продукта устанавливается stock\_quantity: 0
- Все последующие заказы с этим продуктом будут отклонены с ошибкой "Insufficient stock"

### **Пример логов приложения**

```
WHERE orders.id = $1::INTEGER
2025-12-07 20:11:37,470 INFO sqlalchemy.engine.Engine [cached since 1.109s ago] (3,)
INFO - 2025-12-07 20:11:37,469 - sqlalchemy.engine.Engine - base - SELECT orders.id
FROM orders
WHERE orders.id = $1::INTEGER
INFO - 2025-12-07 20:11:37,470 - sqlalchemy.engine.Engine - base - [cached since 1.109s ago] (3,)
2025-12-07 20:11:37,473 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, order_items.unit_price AS order_items_unit_price
FROM order_items
WHERE $1::INTEGER = order_items.order_id
2025-12-07 20:11:37,473 INFO sqlalchemy.engine.Engine [cached since 1.105s ago] (3,)
INFO - 2025-12-07 20:11:37,473 - sqlalchemy.engine.Engine - base - 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, order_items.unit_price AS order_items_unit_price
FROM order_items
WHERE $1::INTEGER = order_items.order_id
INFO - 2025-12-07 20:11:37,473 - sqlalchemy.engine.Engine - base - [cached since 1.105s ago] (3,)
INFO - 2025-12-07 20:11:37,477 - src.messaging.order - order - Order status updated: ID=3, new_status=cancelled
2025-12-07 20:11:37,478 INFO sqlalchemy.engine.Engine ROLLBACK
INFO - 2025-12-07 20:11:37,478 - sqlalchemy.engine.Engine - base - ROLLBACK
2025-12-07 20:11:37,480 INFO -          | order    | 19c6d9a0bf - Processed
2025-12-07 20:11:39,981 INFO -          | order    | cd11f83c95 - Received
Looking for user with ID: 1 (type: <class 'int'>)
```

```
INFO - users
2025-12-07 20:11:39,984 INFO sqlalchemy.engine.Engine [cached since 11.92s ago] (1,)
WHERE users.id = $1::INTEGER
INFO - 2025-12-07 20:11:39,984 - sqlalchemy.engine.Engine - base - [cached since 11.92s ago] (1,)
Found user: User(id=1, username=testuser, email=testuser@example.com)
2025-12-07 20:11:39,987 INFO sqlalchemy.engine.Engine SELECT products.id AS products_id, products.name AS products_name, products.price AS products_price, products.stock_quantity AS products_stock_quantity
FROM products
WHERE products.id = $1::INTEGER
2025-12-07 20:11:39,988 INFO sqlalchemy.engine.Engine [cached since 11.89s ago] (5,)
INFO - 2025-12-07 20:11:39,987 - sqlalchemy.engine.Engine - base - SELECT products.id AS products_id, products.name AS products_name, products.price AS products_price, products.stock_quantity AS products_stock_quantity
FROM products
WHERE products.id = $1::INTEGER
INFO - 2025-12-07 20:11:39,988 - sqlalchemy.engine.Engine - base - [cached since 11.89s ago] (5,)
ERROR - 2025-12-07 20:11:39,990 - src.messaging.order - order - Business logic error while creating order: Insufficient stock
2025-12-07 20:11:40,012 INFO sqlalchemy.engine.Engine ROLLBACK
ERROR - 2025-12-07 20:11:39,990 - src.messaging.order - order - Error processing order message: Insufficient stock
Traceback (most recent call last):
  File "/app/src/messaging/order.py", line 44, in subscribe_order
    await handle_order_create(order_service, data)
  File "/app/src/messaging/order.py", line 66, in handle_order_create
    order = await service.create_order(order_create.model_dump())
              ^^^^^^^^^^^^^^^^^^
  File "/app/src/services/order_service.py", line 46, in create_order
    raise ValueError("Insufficient stock")
ValueError: Insufficient stock
INFO - 2025-12-07 20:11:40,012 - sqlalchemy.engine.Engine - base - ROLLBACK
2025-12-07 20:11:40,014 ERROR -          | order    | cd11f83c95 - ValueError: Insufficient stock
Traceback (most recent call last):
  File "/app/.venv/lib/python3.13/site-packages/faststream/_internal/endpoint/subscriber/usecase.py", line 364, in process_message
    await h.call()
```

## Состояние базы после тестирования

## Server response

| Code | Details |
|------|---------|
|------|---------|

200

### Response body

```
{  
    "total": 1,  
    "items": [  
        {  
            "username": "testuser",  
            "email": "testuser@example.com",  
            "full_name": "Test User",  
            "is_active": true,  
            "id": 1,  
            "created_at": "2025-12-07T20:19:51.141Z",  
            "updated_at": "2025-12-07T20:19:51.141Z"  
        }  
    ]  
}
```

## Response headers

| Code | Details |
|------|---------|
|------|---------|

200

### Response body

```
{  
    "total": 5,  
    "items": [  
        {  
            "id": 3,  
            "name": "Mechanical Keyboard",  
            "price": 149.99,  
            "stock_quantity": 74  
        },  
        {  
            "id": 4,  
            "name": "USB-C Hub 7-in-1",  
            "price": 49.99,  
            "stock_quantity": 197  
        },  
        {  
            "id": 1,  
            "name": "Laptop Dell XPS 15",  
            "price": 1199.99,  
            "stock_quantity": 24  
        },  
        {  
            "id": 2,  
            "name": "Wireless Mouse Logitech MX",  
            "price": 79.99,  
            "stock_quantity": 148  
        },  
        {  
            "id": 5,  
            "name": "Smartphone Samsung Galaxy S24",  
            "price": 999.99,  
            "stock_quantity": 100  
        }  
    ]  
}
```

## Response headers

**Code**

**Details**

200

**Response body**

```
        "stock_quantity": 197
    },
    {
        "id": 4,
        "name": "USB-C Hub 7-in-1",
        "price": 49.99,
        "stock_quantity": 197
    },
    {
        "id": 1,
        "name": "Laptop Dell XPS 15",
        "price": 1199.99,
        "stock_quantity": 24
    },
    {
        "id": 2,
        "name": "Wireless Mouse Logitech MX",
        "price": 79.99,
        "stock_quantity": 148
    },
    {
        "id": 5,
        "name": "Webcam Logitech C920",
        "price": 89.99,
        "stock_quantity": 2
    }
]
```

**Code**

**Details**

200

**Response body**

```
{
    "total": 3,
    "items": [
        {
            "id": 1,
            "user_id": 1,
            "total_amount": 1459.97,
            "status": "processing",
            "items": [
                {
                    "id": 1,
                    "product_id": 1,
                    "quantity": 1,
                    "unit_price": 1299.99
                },
                {
                    "id": 2,
                    "product_id": 2,
                    "quantity": 2,
                    "unit_price": 79.99
                }
            ]
        },
        {
            "id": 2,
            "user_id": 1,
            "total_amount": 299.96000000000004,
            "status": "processing"
        }
    ]
}
```

#### Response body

```
        }
    ],
},
{
  "id": 2,
  "user_id": 1,
  "total_amount": 299.96000000000004,
  "status": "shipped",
  "items": [
    {
      "id": 3,
      "product_id": 3,
      "quantity": 1,
      "unit_price": 149.99
    },
    {
      "id": 4,
      "product_id": 4,
      "quantity": 3,
      "unit_price": 49.99
    }
  ]
},
{
  "id": 3,
  "user_id": 1,
  "total_amount": 259.96999999999997,
```

#### Response headers

200

#### Response body

```
        "quantity": 3,
        "unit_price": 49.99
      }
    ],
},
{
  "id": 3,
  "user_id": 1,
  "total_amount": 259.96999999999997,
  "status": "cancelled",
  "items": [
    {
      "id": 5,
      "product_id": 2,
      "quantity": 1,
      "unit_price": 79.99
    },
    {
      "id": 6,
      "product_id": 5,
      "quantity": 2,
      "unit_price": 89.99
    }
  ]
}
```

## **Ответы на вопросы**

### **1. Что такое AMQP? Каковы его основные преимущества?**

AMQP (Advanced Message Queuing Protocol) - открытый протокол для обмена сообщениями между приложениями через брокер сообщений (например RabbitMQ). Он определяет формат сообщений, семантику подтверждений, маршрутизацию и управление очередями.

#### **Основные преимущества:**

- Стандартизованный протокол — совместимость между разными реализациями/клиентами.
- Надежность: гарантированная доставка сообщений с подтверждением (acknowledgments)
- Маршрутизация: гибкая система exchange и binding для направления сообщений
- Безопасность: встроенная аутентификация и шифрование (SASL, TLS)
- Независимость от платформы: единый стандарт для разных языков и систем
- Транзакционность: поддержка транзакций для атомарных операций

В работе используется через URL: <amqp://guest:guest@rabbitmq:5672/>

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

Очередь сообщений - механизм асинхронного взаимодействия между приложениями, при котором сообщения, отправленные производителем (producer), временно сохраняются в упорядоченной структуре и доставляются ровно одному получателю (consumer)

- Имеет point-to-point модель: одно сообщение - один получатель
- Сообщения хранятся до обработки
- Гарантия порядка обработки (FIFO) - сообщение, которое первым попало в очередь, будет первым обработано.

Пример: очереди order и product с одним consumer на сообщение (только один обработчик получит это сообщение):

- `@broker.subscriber(RabbitQueue("order", durable=True))`  
`async def subscribe_order(message: Dict):`

Шина сообщений - архитектурный подход к обмену сообщениями между системами, обеспечивающая обмен сообщениями между множеством сервисов, маршрутизацию, трансформации, публикацию/подписку и согласованные контракты.

- Имеет Pub/Sub модель: одно сообщение - множество подписчиков
- Broadcast доставка всем подписчикам
- Используются topic/fanout exchanges

Пример: уведомления о событиях (order.created - email service, analytics, warehouse)

### 3. Как обеспечить надежную доставку сообщений в RabbitMQ?

**Ключевые практики** (комбинируются для надёжности):

- 1) **Durable очереди** (они переживают перезапуск RabbitMQ). Нужно создавать очереди с `durable=true`. Пример из лабораторной:  
`@broker.subscriber(RabbitQueue("order", durable=True))`
- 2) **Persistent сообщения** (сохраняются на диск). Это требуется, чтобы сообщения пережили перезапуск брокера. Пример:  
`delivery_mode=DeliveryMode.PERSISTENT`.
- 3) **Acknowledgments** (автоматические в FastStream): при успешной обработке сообщение удаляется из очереди, а при исключении сообщение возвращается в очередь для повторной обработки.
- 4) **Publisher confirms** – позволяет включать подтверждения (publisher confirms) как способ узнать, что брокер принял сообщение на запись.

**5) Обработка отказов: reject / nack + DLQ.** При ошибках сообщение либо возвращается в очередь (requeue=true), либо перенаправляется в Dead Letter Queue для дальнейшего анализа.

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

Конкретное поведение в этом случае зависит от настроек.

Если consumer использует manual acknowledgements (auto\_ack=false), есть несколько вариантов:

- при условии, что потребитель не успел отправить ack и его соединение/канал закрывается, RabbitMQ пометит сообщение как unacknowledged и переложит его обратно в очередь — оно будет повторно доставлено другому (или тому же при восстановлении) consumer. При повторной доставке будет установлен флаг redelivered=true. **Этот вариант используется в коде лабораторной**
- при условии, если потребитель успел отправить ack до падения - то сообщение считается доставленным и удаляется из очереди.

Если consumer использует auto-ack = true (no-ack):

- в этом случае сообщение считается доставленным сразу после отправки брокером; если consumer падает во время обработки, брокер не будет повторно отправлять сообщение - обработка потеряна.

Если consumer сделал nack/reject с requeue=false: сообщение либо отброшено, либо попадёт в DLX (если настроен dead-letter-exchange).

#### **5. Как обеспечить сохранность сообщений при перезапуске RabbitMQ?**

Чтобы сообщения пережили перезапуск брокера в работе были реализованы:

1. Durable очереди (объявлены при старте): очереди пересоздаются после перезапуска с сохранением настроек.
2. Persistent сообщения: python delivery\_mode=DeliveryMode.PERSISTENT. Сообщения записываются на диск перед подтверждением.
3. Volume для данных RabbitMQ (docker-compose.yml): все данные

(очереди, сообщения, настройки) сохраняются между перезапусками контейнера.

Что произойдет при перезапуске:

- Очереди восстановятся из volume
- Необработанные persistent сообщения останутся в очередях
- Non-persistent сообщения будут потеряны

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

TTL (Time To Live) — максимальное время, в течение которого сообщение считается «жизнеспособным» в очереди. По истечении TTL сообщение либо удаляется, либо отправляется в Dead-Letter Exchange (если настроен DLX).

Применение:

- Предотвращение накопления устаревших сообщений
- Автоматическая очистка временных данных
- Защита от переполнения очередей

Способы настройки:

- На уровне сообщения через свойство expiration. Значение задаётся в миллисекундах строкой. Если у очереди тоже задан x-message-ttl, то используется минимальное значение.
- На уровне очереди через аргумент x-message-ttl. Это TTL для всех сообщений в очереди и классический вариант для бизнес-очередей.
- Для Dead Letter Queue (DLQ) часто используют для: временного хранения ошибок или реализации повторных попыток (retry).

**Вывод:**

В ходе выполнения лабораторной работы была реализована полноценная система обработки заказов и продуктов с использованием RabbitMQ как брокера сообщений, с разделением команд (через MQ) и запросов (через REST

API), обеспечивающая надежность, масштабируемость и согласованность данных.

Практическая реализация:

- Настроена инфраструктура в Docker Compose с RabbitMQ, PostgreSQL и приложением
- Созданы две очереди (order и product) с поддержкой множественных операций через единый обработчик
- Реализована бизнес-логика с проверкой остатков товаров, автоматическим списанием при создании заказа и возвратом при отмене
- Разделена ответственность: команды (create/update) через RabbitMQ, запросы (get/list) через REST API

Разработан комплексный тестовый скрипт, создающий 5 продуктов и 3 заказа, демонстрирующий все основные операции системы, включая негативные сценарии с недостаточным количеством товара.

Полученный опыт применим для построения микросервисных архитектур, систем обработки событий и асинхронных workflow, где требуется надежная доставка сообщений и отказоустойчивость.