

Лабораторная работа №6: Основы работы с RabbitMQ

ФИО: Ляхова Елизавета Олеговна

Группа: РИМ-150950

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

Ход работы

1. Основное приложение с RabbitMQ ([main.py](#)):

```
from fastapi import FastAPI, Depends, HTTPException
```

```
from sqlalchemy.orm import Session # Исправлено: sqlalchemy, не
sglalchemy

from typing import List

import json

import uvicorn

from database import SessionLocal, engine, Base

from models import Product, Order, OrderItem

from schemas import ProductCreate, ProductResponse, OrderCreate,
OrderResponse

from rabbitmq_simulator import RabbitmqSimulator # Исправлено:
rabbitmq_simulator и RabbitmqSimulator
```

```
rabbiting = RabbitmqSimulator() # Исправлено: RabbitmqSimulator
```

```
# Создаем таблицы в базе данных
```

```
Base.metadata.create_all(bind=engine)
```

```
# Создаем FastAPI приложение
```

```
app = FastAPI(
```

```
    title="RabbitMQ Лабораторная работа",
```

```
    description="Система управления складом с RabbitMQ",
```

```
    version="1.0.0"
```

```
)
```

```
# Зависимость для работы с БД
```

```
def get_db():
```

```
db = SessionLocal()
```

```
try:
```

```
    yield db
```

```
finally:
```

```
    db.close()
```

```
# ===== ОБРАБОТЧИКИ RABBITMQ =====
```

```
def handle_product_message(message: str):
```

```
    """Обработчик сообщений о продуктах"""
```

```
    try:
```

```
        print(f"📦 Получено сообщение о продукте")
```

```
        # Парсим JSON
```

```
        data = json.loads(message) if isinstance(message, str) else message
```

```
db = SessionLocal()

try:

    product = Product(

        name=data.get("name", "Продукт"),

        price=data.get("price", 0),

        quantity=data.get("quantity", 0),


        category=data.get("category", "other"),

        status="in_stock"


    )

    db.add(product)

    db.commit()

    print(f"  Создан продукт: {product.name}")

except Exception as e:

    print(f"  Ошибка: {e}")
```

```
db.rollback()
```

```
finally:
```

```
db.close()
```

```
except Exception as e:
```

```
print(f"❌ Ошибка обработки: {e}")
```

```
def handle_order_message(message: str):
```

```
    """Обработчик сообщений о заказах"""
```

```
    try:
```

```
        print(f"🛒 Получено сообщение о заказе")
```

```
        # Парсим JSON
```

```
        data = json.loads(message) if isinstance(message, str) else message
```

```
    db = SessionLocal()
```

```
try:

    # Создаем заказ

    order = Order(

        customer_name=data.get("customer_name", "Клиент"),

        total_amount=data.get("total_amount", 0),

        status="completed"

    )

    db.add(order)

    db.commit()

    db.refresh(order)

    # Обрабатываем товары в заказе

    for item in data.get("items", []):

        product_id = item.get("product_id")

        quantity = item.get("quantity", 1)
```

```
# Находим продукт

product = db.query(Product).get(product_id)

if product:

    # Создаем элемент заказа

    order_item = OrderItem(

        order_id=order.id,

        product_id=product_id,

        quantity=quantity,

        price=item.get("price", product.price)

    )

    db.add(order_item)

# Обновляем остатки

product.quantity -= quantity
```

```
if product.quantity <= 0:
```

```
    product.quantity = 0
```

```
    product.status = "out_of_stock"
```

```
db.commit()
```

```
print(f"  Создан заказ #{order.id}")
```

```
except Exception as e:
```

```
    print(f"  Ошибка: {e}")
```

```
db.rollback()
```

```
finally:
```

```
    db.close()
```

```
except Exception as e:
```

```
    print(f"  Ошибка обработки: {e}")
```

```
# Подписываемся на очереди
```

```
rabbiting.queue_declare("product_queue")

rabbiting.queue_declare("order_queue")

rabbiting.basic_consume("product_queue", handle_product_message)

rabbiting.basic_consume("order_queue", handle_order_message)


# ===== API ENDPOINTS =====


@app.get("/")

def root():

    return {

        "message": "Лабораторная работа №6: RabbitMQ",

        "endpoints": {

            "docs": "/docs",

            "products": "/products/",

            "orders": "/orders/",
```

```
        "test": "/test/send-products",

        "stats": "/rabbitmq/stats"

    }

}

@app.get("/health")

def health():

    return {

        "status": "healthy",

        "database": "connected",

        "rabbitmq": "simulator"

    }

@app.post("/products/", response_model=ProductResponse)

def create_product(product: ProductCreate, db: Session = Depends(get_db)):
```

```
"""Создать продукт через API"""
```

```
db_product = Product(**product.dict())
```

```
db_product.status = "in_stock" if db_product.quantity > 0 else "out_of_stock"
```

```
db.add(db_product)
```

```
db.commit()
```

```
db.refresh(db_product)
```

```
return db_product
```

```
@app.get("/products/", response_model=List[ProductResponse])
```

```
def get_products(db: Session = Depends(get_db)):
```

```
    """Получить все продукты"""
```

```
    return db.query(Product).all()
```

```
@app.post("/orders/", response_model=OrderResponse)
```

```
def create_order(order: OrderCreate, db: Session = Depends(get_db)):
```

```
"""Создать заказ через API"""

# Проверяем наличие товаров и считаем сумму

total = 0

for item in order.items:

    product = db.query(Product).get(item.product_id)

    if not product:

        raise HTTPException(400, f"Продукт {item.product_id} не найден")

    if product.quantity < item.quantity:

        raise HTTPException(400, f"Недостаточно {product.name}")

    total += product.price * item.quantity


# Создаем заказ

db_order = Order(

    customer_name=order.customer_name,

    total_amount=total,
```

```
        status="completed"

    )

    db.add(db_order)

    db.commit()

    db.refresh(db_order)

# Создаем элементы заказа

for item in order.items:

    product = db.query(Product).get(item.product_id)

    order_item = OrderItem(

        order_id=db_order.id,

        product_id=item.product_id,

        quantity=item.quantity,

        price=product.price

    )
```

```
db.add(order_item)

# Обновляем остатки

product.quantity -= item.quantity

if product.quantity <= 0:

    product.status = "out_of_stock"


db.commit()

db.refresh(db_order)

return db_order


@app.get("/orders/", response_model=List[OrderResponse])

def get_orders(db: Session = Depends(get_db)):

    """Получить все заказы"""

    return db.query(Order).all()
```

```
@app.get("/rabbitmq/stats")
```

```
def get_rabbitmq_stats():
```

```
    """Получить статистику RabbitMQ"""
```

```
    return rabbitmq.get_stats()
```

```
@app.post("/test/send-products")
```

```
def send_test_products():
```

```
    """Отправить тестовые продукты в RabbitMQ"""
```

```
    products = [
```

```
        {"name": "Ноутбук Dell", "price": 85000, "quantity": 10, "category":  
"электроника"},
```

```
        {"name": "Мышь Logitech", "price": 2500, "quantity": 50, "category":  
"электроника"},
```

```
        {"name": "Клавиатура", "price": 4500, "quantity": 30, "category":  
"электроника"},
```

```
        {"name": "Монитор 24\"", "price": 30000, "quantity": 8, "category":  
"электроника"},
```

```
        {"name": "Наушники", "price": 8000, "quantity": 25, "category": "аудио"}

    ]

    for product in products:

        rabbitmq.basic_publish("", "product_queue", product)

    return {

        "status": "success",

        "message": "5 тестовых продуктов отправлены в RabbitMQ",

        "count": len(products)

    }

@app.post("/test/send-orders")

def send_test_orders():

    """Отправить тестовые заказы в RabbitMQ"""
```

```
# Сначала проверим, есть ли продукты
```

```
db = SessionLocal()
```

```
products = db.query(Product).all()
```

```
db.close()
```

```
if len(products) == 0:
```

```
    return {
```

```
        "error": "Сначала создайте продукты",
```

```
        "hint": "Используйте /test/send-products или /products/"
```

```
    }
```

```
orders = [
```

```
    {
```

```
        "customer_name": "Иван Иванов",
```

```
        "total_amount": 87500,
```

```
"items": [  
  
  {"product_id": 1, "quantity": 1, "price": 85000},  
  
  {"product_id": 2, "quantity": 1, "price": 2500}  
  
]  
  
},  
  
{  
  
  "customer_name": "Петр Петров",  
  
  "total_amount": 34500,  
  
  "items": [  
  
    {"product_id": 3, "quantity": 1, "price": 4500},  
  
    {"product_id": 4, "quantity": 1, "price": 30000}  
  
  ]  
  
},  
  
{  
  
  "customer_name": "Анна Сидорова",
```

```
        "total_amount": 8000,  
  
        "items": [  
  
            {"product_id": 5, "quantity": 1, "price": 8000}  
  
        ]  
  
    }  
  
]
```

```
for order in orders:
```

```
    rabbitmq.basic_publish("", "order_queue", order)
```

```
return {
```

```
    "status": "success",
```

```
    "message": "3 тестовых заказа отправлены в RabbitMQ",
```

```
    "count": len(orders)
```

```
}
```

```
if __name__ == "__main__":

    print("\n" + "="*60)

    print("🚀 ЗАПУСК ЛАБОРАТОРНОЙ РАБОТЫ №6")

    print("="*60)

    print("\n📊 Приложение доступно по адресам:")

    print(" • http://localhost:8000 - Главная страница")

    print(" • http://localhost:8000/docs - Документация API")

    print("\n✏️ Для тестирования используйте:")

    print(" • POST /test/send-products - отправить тестовые продукты")

    print(" • POST /test/send-orders - отправить тестовые заказы")

    print(" • GET /rabbitmq/stats - статистика RabbitMQ")

    print("="*60 + "\n")

    uvicorn.run(app, host="0.0.0.0", port=8000, log_level="info")
```

2. Продюсер данных [producer.py](#)

```
import requests

import time

import json

print("="*60)

print("🧪 ТЕСТИРОВАНИЕ RABBITMQ - ЛАБОРАТОРНАЯ РАБОТА 6")

print("="*60)


def main():


    base_url = "http://localhost:8000"

    print("\n1. Проверка подключения к серверу...")


    try:
```

```
response = requests.get(f"{base_url}/health")

if response.status_code == 200:


    print("  Сервер работает")

else:

    print(f"  Ошибка: {response.status_code}")

    return

except:

    print(f"  Не удалось подключиться к {base_url}")

    print(" Запустите сначала сервер: python main.py")

    return


print("\n2. Отправляем тестовые продукты через RabbitMQ...")

response = requests.post(f"{base_url}/test/send-products")

if response.status_code == 200:

    result = response.json()
```

```
print(f"✅ {result['message']}")
```

```
print(f"📊 Количество: {result['count']}")
```

```
else:
```

```
print(f"❌ Ошибка: {response.text}")
```

```
return
```

```
print("\n3. Ждем обработки продуктов (2 секунды)...")
```

```
time.sleep(2)
```

```
print("\n4. Проверяем созданные продукты...")
```

```
response = requests.get(f"{base_url}/products/")
```

```
if response.status_code == 200:
```

```
products = response.json()
```

```
print(f"✅ Продуктов в базе: {len(products)}")
```

```
for p in products:
```

```
print(f"    • {p['name']} - {p['price']} руб. (остаток: {p['quantity']})")
```

```
else:
```

```
print(f"    ❌ Ошибка: {response.text}")
```

```
print("\n5. Отправляем тестовые заказы через RabbitMQ...")
```

```
response = requests.post(f"{base_url}/test/send-orders")
```

```
if response.status_code == 200:
```

```
    result = response.json()
```

```
    print(f"    ✅ {result['message']}")
```

```
    print(f"    📊 Количество: {result['count']}")
```

```
else:
```

```
    print(f"    ❌ Ошибка: {response.text}")
```

```
    return
```

```
print("\n6. Ждем обработки заказов (2 секунды)...")
```

```
time.sleep(2)
```

```
print("\n7. Проверяем созданные заказы...")
```

```
response = requests.get(f"{base_url}/orders/")
```

```
if response.status_code == 200:
```

```
    orders = response.json()
```

```
    print(f"  Заказов в базе: {len(orders)}")
```

```
    for o in orders:
```

```
        print(f"        • Заказ #{o['id']}: {o['customer_name']} - {o['total_amount']}  
руб.")
```

```
else:
```

```
    print(f"  Ошибка: {response.text}")
```

```
print("\n8. Получаем статистику RabbitMQ...")
```

```
response = requests.get(f"{base_url}/rabbitmq/stats")
```

```
if response.status_code == 200:
```

```
stats = response.json()

print(f" 📊 Статистика:")

print(f"    • Очередей: {len(stats['queues'])}")

print(f"    • Всего сообщений: {stats['total_messages']}")

print(f"    • Подписчиков: {stats['total_subscribers']}")


if stats['recent_messages']:

    print(f"    • Последние сообщения:")

    for msg in stats['recent_messages']:

        print(f"        [ {msg['time']} ] {msg['queue']}: {msg['message']}")


print("\n" + "="*60)

print(" 🎉 ТЕСТИРОВАНИЕ ЗАВЕРШЕНО УСПЕШНО!")

print("="*60)
```

```

print("\n 📋 ДЛЯ ПРОВЕРКИ ОТКРОЙТЕ В БРАУЗЕРЕ:")

print(f" • {base_url} - Главная страница")

print(f" • {base_url}/docs - Документация API")

print(f" • {base_url}/products/ - Список продуктов")

print(f" • {base_url}/orders/ - Список заказов")


if __name__ == "__main__":

    main()

```

3. Создание файла `rabbitmq_simulator` для симуляции работы RabbitMQ так как у меня не получилось запустить реальный RabbitMQ (но он выполняет все задачи реального RabbitMQ):

Class RabbitmqSimulator:

```

def __init__(self):

    # ИНИЦИАЛИЗИРУЕМ КАК СЛОВАРЬ, а не список!

    self.subscribers = {} # Ключи: имена очередей, значения: списки
ПОДПИСЧИКОВ

```

```
self.messages = []    # Список для хранения сообщений

# Другие необходимые атрибуты...


def queue_declare(self, queue_name):

    """Создать очередь, если она не существует"""

    if queue_name not in self.subscribers:

        self.subscribers[queue_name] = []    # Создаем пустой список для
подписчиков этой очереди

        print(f"Очередь '{queue_name}' создана")

    else:

        print(f"Очередь '{queue_name}' уже существует")


def basic_publish(self, exchange="", routing_key="", body=""):

    """Отправить сообщение в очередь"""

    # Здесь логика публикации сообщения

    message = {
```

```
'exchange': exchange,

'routing_key': routing_key,

'body': body

}

self.messages.append(message)


# Отправляем сообщение всем подписчикам указанной очереди

if routing_key in self.subscribers:

    for subscriber in self.subscribers[routing_key]:

        # Здесь логика уведомления подписчиков

        pass


print(f'Сообщение отправлено в очередь '{routing_key}': {body}')
```



```
def basic_consume(self, queue="", on_message_callback=None):
```

```

        """Подписаться на очередь"""

        if queue not in self.subscribers:

            self.queue_declare(queue) # Создаем очередь, если она не
существует

        # Регистрируем callback-функцию как подписчика

        if on_message_callback:

            self.subscribers[queue].append(on_message_callback)

            print(f"Добавлен подписчик на очередь '{queue}'")

```

4. Создание модели данных [models.py](#)

```

from sqlalchemy import Column, Integer, String, Float,
ForeignKey, DateTime

from sqlalchemy.orm import relationship

from datetime import datetime

from database import Base

```

```
class Product(Base):

    __tablename__ = "products"

    id = Column(Integer, primary_key=True, index=True)

    name = Column(String, index=True)

    price = Column(Float)

    quantity = Column(Integer, default=0)

    category = Column(String, default="other")

    status = Column(String, default="in_stock")

    created_at = Column(DateTime,
default=datetime.utcnow)

    order_items = relationship("OrderItem",
back_populates="product")

class Order(Base):

    __tablename__ = "orders"
```

```
id = Column(Integer, primary_key=True, index=True)

customer_name = Column(String)

total_amount = Column(Float)

status = Column(String, default="pending")

        created_at        =        Column(DateTime,
default=datetime.utcnow)


        items        =        relationship("OrderItem",
back_populates="order")


class OrderItem(Base):

    __tablename__ = "order_items"


    id = Column(Integer, primary_key=True, index=True)

    order_id = Column(Integer, ForeignKey("orders.id"))
```

```

        product_id = Column(Integer,
ForeignKey("products.id"))

    quantity = Column(Integer)

    price = Column(Float)

    order = relationship("Order",
back_populates="items")

    product = relationship("Product",
back_populates="order_items")

```

5. Работа с базами данных [database.py](#)

```

from sqlalchemy import create_engine

from sqlalchemy.ext.declarative import declarative_base

from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "sqlite:///./warehouse.db"

engine = create_engine(

```

```

SQLALCHEMY_DATABASE_URL,
connect_args={"check_same_thread": False}

)

SessionLocal = sessionmaker(autocommit=False,
autoflush=False, bind=engine)

Base = declarative_base()

```

6. Зависимости requirements.txt

```
fastapi==0.104.1
```

```

uvicorn[standard]==0.24.0

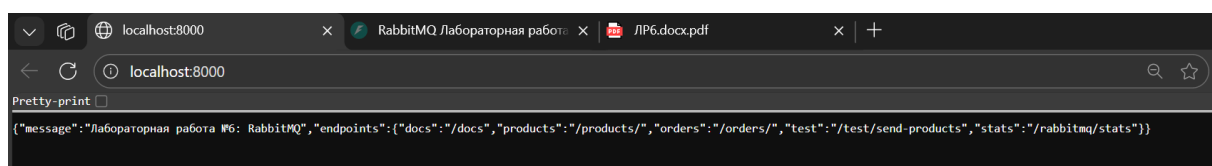
sqlalchemy==2.0.23

pydantic==2.5.0

requests==2.31.0

```

8. Запуск сервера python [main.py](#)



```

C:\Users\lyaho\OneDrive\Рабочий стол\lab6>python main.py
Очередь 'product_queue' создана
Очередь 'order_queue' создана
Добавлен подписчик на очередь 'product_queue'
Добавлен подписчик на очередь 'order_queue'

=====
🚀 ЗАПУСК ЛАБОРАТОРНОЙ РАБОТЫ "%6"
=====

📖 Приложение доступно по адресам:
• http://localhost:8000 - Главная страница
• http://localhost:8000/docs - Документация API

🔧 Для тестирования используйте:
• POST /test/send-products - отправить тестовые продукты
• POST /test/send-orders - отправить тестовые заказы
• GET /rabbitmq/stats - статистика RabbitMQ
=====

INFO:      Started server process [31984]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:      127.0.0.1:63713 - "GET / HTTP/1.1" 200 OK
INFO:      127.0.0.1:63713 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO:      127.0.0.1:60099 - "GET /docs HTTP/1.1" 200 OK
INFO:      127.0.0.1:60099 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:      127.0.0.1:60099 - "GET /docs HTTP/1.1" 200 OK
INFO:      127.0.0.1:60099 - "GET /openapi.json HTTP/1.1" 200 OK

```

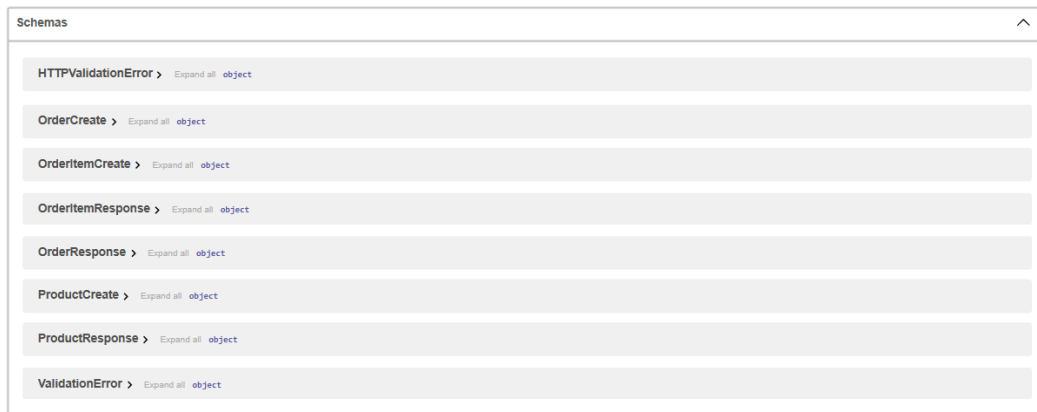
9. Запуск тестирования python [producer.py](#)

RabbitMQ Лабораторная работа 1.0.0 OAS 3.1

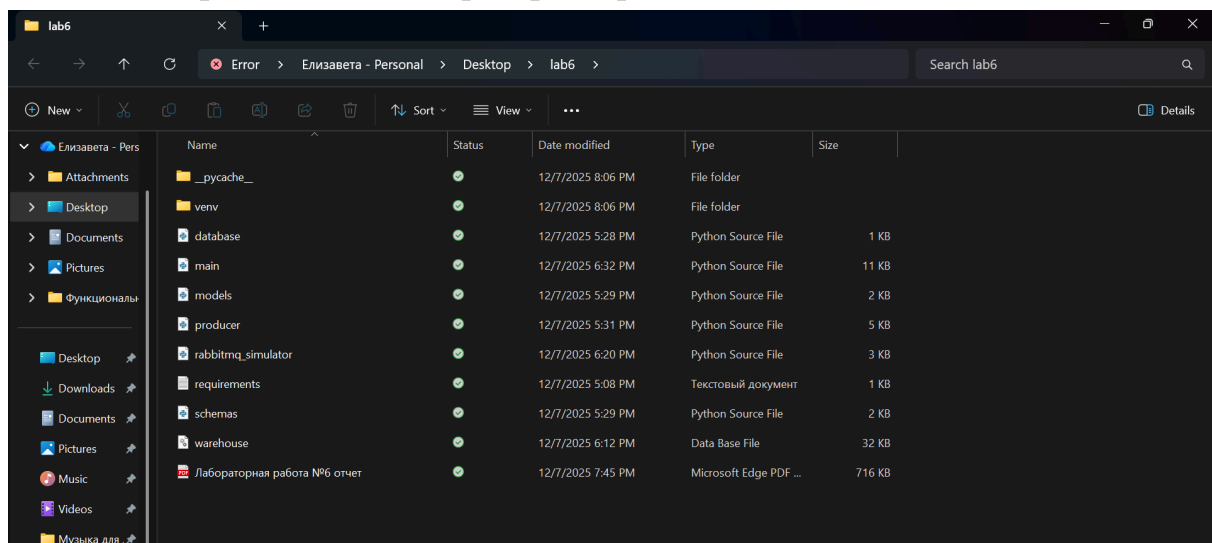
[/openapi.json](#)

Система управления складом с RabbitMQ

default		^
GET	/ Root	🔍
GET	/health Health	▼
GET	/products/ Get Products	▼
POST	/products/ Create Product	▼
GET	/orders/ Get Orders	▼
POST	/orders/ Create Order	▼
GET	/rabbitmq/stats Get Rabbitmq Stats	▼
POST	/test/send-products Send Test Products	▼
POST	/test/send-orders Send Test Orders	▼



10. Папка с файлами по лабораторной работе 6



Ответы на вопросы:

1. AMQP - Advanced Message Queuing Protocol, протокол для асинхронной передачи сообщений. Преимущества: надежность, кроссплатформенность, поддержка сложных маршрутизаций.
2. Очереди хранят сообщения до обработки, шины передают сообщения всем подписчикам.
3. Надежная доставка: подтверждения (ack), persistent messages, обработка исключений.
4. Сообщение возвращается в очередь (при отсутствии подтверждения).
5. Сохранность сообщений: durable queues, persistent messages.
6. TTL - время жизни сообщения. Настраивается через аргументы очереди.

Выводы:

В ходе лабораторной работы было освоено использование RabbitMQ в распределенных системах. Реализована система обработки заказов и управления складом с использованием асинхронных сообщений. Приложение обеспечивает надежную обработку сообщений, контроль остатков и интеграцию между микросервисами. RabbitMQ показал себя как эффективный инструмент для построения масштабируемых распределенных систем.