Задание 1

connectors/rabbit/task/rmq_worker.py — нужно реализовать WorkerClient и Worker.

В предыдущей главе мы реализовывали worker, который получает задачи из очереди asyncio.Queue. Теперь нужно написать worker, который будет получать задачи из брокера сообщений RabbitMQ.

Программный интерфейс

Клиент, который хочет добавить задачу в очередь, должен вызвать для worker метод put класса WorkerClient:

```
config = WorkerClientConfig(rabbit_url="rabbit", queue_name="queue")
async with WorkerClient(config) as worker:
    await worker.put({'type': 'event'})
```

После выполнения фрагмента кода задача *{'type': 'event'}* должна попасть в Rabbit в очередь queue.

Процесс, который хочет стать потребителем и непосредственно обрабатывать задачи, должен создать класс-наследник от Worker и реализовать метод *handler:*

```
class MetaclassWorker(Worker):
    async def handler(self, msg: IncomingMessage):
        await asyncio.sleep(5)
        print(msg.body.decode())


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    config = WorkerConfig(rabbit_url="rabbit", queue_name="queue", capacity=2)
    logger = MetaclassWorker(config)
    loop.create_task(logger.start())
    try:
        loop.run_forever()
    except KeyboardInterrupt:
        loop.run_until_complete(logger.stop())
```

В результате запуска фрагмента воркер должен начать получать задачи из очереди Rabbit, при этом максимальное количество одновременно работающих задач не должно быть больше 2 (capacity).

WorkerClient

Шаги выполнения

- **определить метод put,** который должен класть словарь data в очередь RabbitMQ с названием, которое нужно получить из конфига WorkerClientConfig.queue_name
- **определить метод stop,** который будет корректно завершать работы с WorkerClient

Как проверить руками

Запустить:

```
python3 run_worker_client.py
```

Далее открыть админку RabbitMQ. Найти очередь с название worker_metaclass — в ней должна лежать одна задача.

Как проверить тестами

```
pytest tests/connectors/rabbit/test_rmq_worker.py -vv -s
```

Worker

Шаги выполнения

1. определить метод _worker, который должен вызываться, когда пришло новое сообщение в очередь, вызывать внутри метод handler и в случае успеха помечать, что задача выполнена (ack)
2. определить метод start, который не должен блокировать исполнение программы, внутри этого метода нужно определить очередь и зарегистрировать _worker (queue.consume(...))
3. отпределить метод stop, который должен закрывать соединение с RabbitMQ
4. добавить ограничение, что в worker может одновременно обрабатываться не больше, чем WorkerConfig.capacity задач.
5. *[*] (Сложно и необязательно)* доработать метод stop так, чтобы он дожидался выполнения всех запущенных корутин. Для проверки корректности выполнения задания нужно убрать метку @pytest.mark.skip в тесте tests.connectors.rabbit.test_rmq_worker.TestWorker.test_cancel.

Как проверить руками

Запустить:

```
python3 run_worker_client.py
python3 run_worker.py
```

В терминале скрипта run_worker.py должны появиться сообщения.

Как проверить тестами

```
pytest tests/connectors/rabbit/test_rmq_worker.py -vv -s
```