Задание 2

connectors/rabbit/task/rmq_logger.py — нужно реализовать Logger.

Данный класс позволяет подключиться к общей шине событий и получать уведомления о произошедних событиях. В предыдущем предложении указан паттерн «общая шина событий», поэтому в этом задании с RabbitMQ нужно работать не как с очередью, а как со средством, которое позволяет организовать pub/sub взаимодействие.

Программный интерфейс

Программисту-пользователю нужно переопределить методы handle_info и handle_critical:

```
class MetaclassLogger(Logger):
    def __init__(self, url: str):
        # name указывается имя exchange, те шины куда должны отправляться сообщения
        super().__init__(LoggerConfig(rabbit_url=url, name="logger_metaclass"))

    async def handle_info(self, data: dict):
        print(data)

    async def handle_critical(self, data: dict):
        print(data)
```

Чтобы отправить событие в шину событий, нужно выполнить:

```
async with MetaclassLogger(rabbitmq_url) as logger:
    await logger.info({'info': True})
```

При этом нужно отметить две вещи:

- можно отправлять и логировать сообщения из разных процессов на разных серверах
- можно запускать несколько разных логгеров, подключенных к одной общей шине — все они будут получать уведомления, и их можно по-разному обрабатывать

Шаги выполнения

1. **Реализовать метод start.** Внутри метода нужно определить exchange, определить временную очередь, привязать временную очередь к exchange, для очереди определить consume (метод _handler). *ВАЖНО!* Уровень логирования будет определяться с помощью routing_key, обратите на это внимание при привязке очереди к exchange
2. **Реализовать метод stop.** Внутри метода нужно закрыть подключение к RabbitMQ
3. **Реализовать метод _handler.** Он является consumer очереди и будет вызываться всякий раз, когда кто-то будет отправлять событие в наш логгер. Внутри функция на основе routing_key должна определить, какому уровню логирования соответствует пришедшее сообщение, и вызвать один из двух хендлеров: handle_info или handle_critical
4. **Реализовать методы info и critical,** которые могут отправить события в шину событий

Как протестировать руками

Запустить в разных терминалах run_logger.py и run_logger_client.py:

```
python3 run_logger.py
python3 run_logger_client.py info
python3 run_logger_client.py critical
```

При запуске метода run_logger_client.py можно передать, какой уровень логирования нужно отправить.

После запуска скриптов в админке RabbitMQ в разделе exchanges должны появиться:

- exchange, который называется logger_metaclass
- временная очередь, которая привязана к этому exchange

При запуске run_logger_client.py в терминале run_logger.py должны появляться сообщения:

```
{'info': True}
{'critical': True}
```

Как протестировать тестами

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