САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №6

Выполнил:

Гнеушев Владислав

К3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

- подключить и настроить rabbitMQ/kafka;
- реализовать межсервисное взаимодействие посредством rabbitMQ/kafka.

Ход работы

Был добавлен сервис RabbitMQ в docker-compose файл.

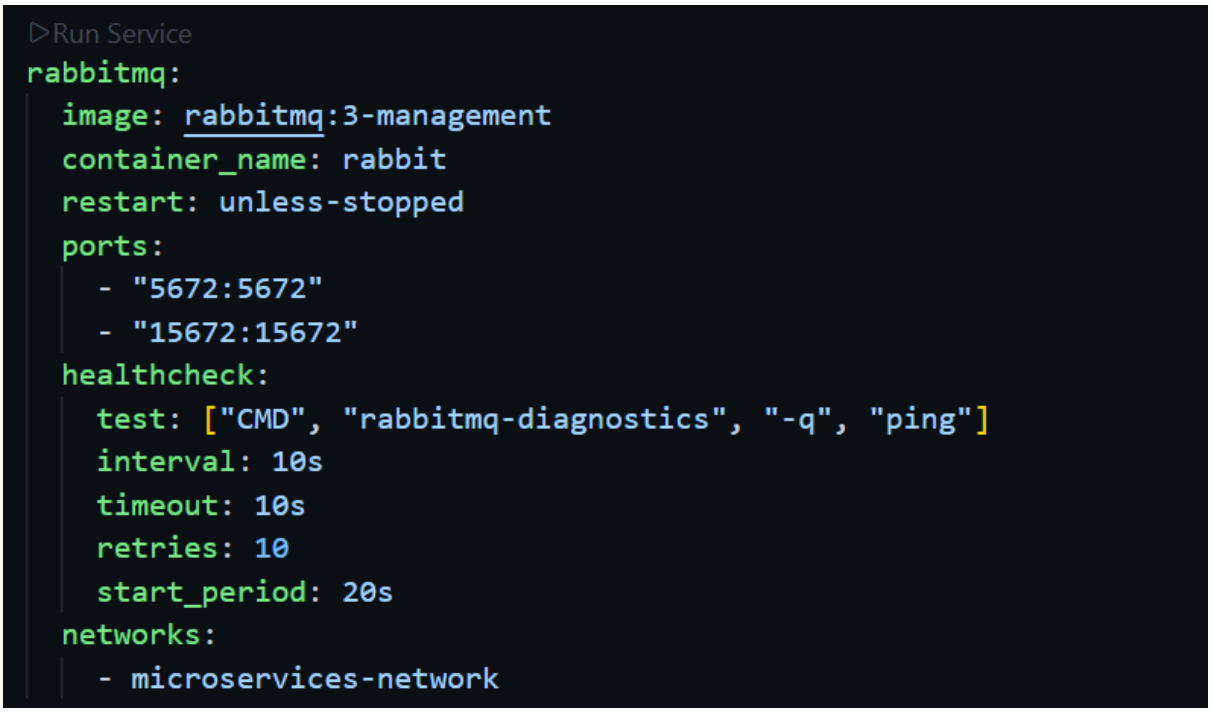
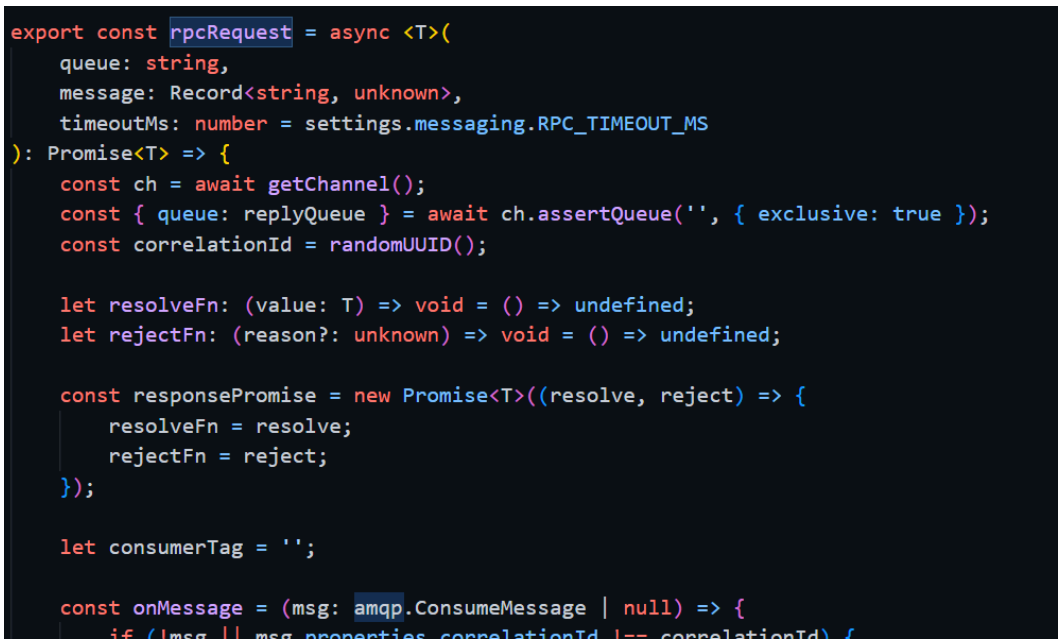


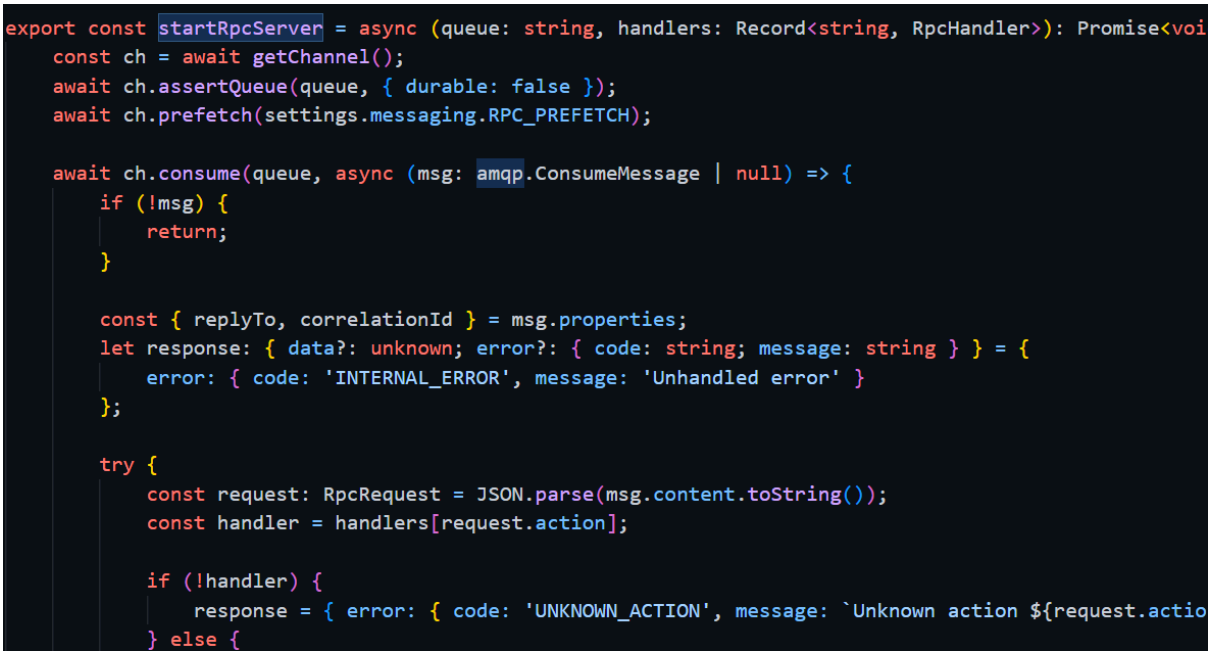
Рисунок 1 – Файл docker-compose.yml с сервисом rabbitmq

Далее произвели переход от HTTP синхронного взаимодействия сервисов к асинхронному. Взаимодействие сервисов при помощи rabbitMQ было решено построить с использованием RPC паттерна. Каждый микросервис получил RPC-client и RPC-server. RPC сервер получает запрос из очереди, отдает его в специальный обработчик сервисного слоя. Рассмотрим на примере.



```
export const rpcRequest = async <T>(
    queue: string,
    message: Record<string, unknown>,
    timeoutMs: number = settings.messaging.RPC_TIMEOUT_MS
): Promise<T> => {
    const ch = await getChannel();
    const { queue: replyQueue } = await ch.assertQueue('', { exclusive: true });
    const correlationId = randomUUID();

    let resolveFn: (value: T) => void = () => undefined;
    let rejectFn: (reason?: unknown) => void = () => undefined;

    const responsePromise = new Promise<T>((resolve, reject) => {
        resolveFn = resolve;
        rejectFn = reject;
    });

    let consumerTag = '';

    const onMessage = (msg: amqp.ConsumeMessage | null) => {
```

Рисунок 2 – Реализация RPC-клиента.



```
export const startRpcServer = async (queue: string, handlers: Record<string, RpcHandler>): Promise<voi
    const ch = await getChannel();
    await ch.assertQueue(queue, { durable: false });
    await ch.prefetch(settings.messaging.RPC_PREFETCH);

    await ch.consume(queue, async (msg: amqp.ConsumeMessage | null) => {
        if (!msg) {
            return;
        }

        const { replyTo, correlationId } = msg.properties;
        let response: { data?: unknown; error?: { code: string; message: string } } = {
            error: { code: 'INTERNAL_ERROR', message: 'Unhandled error' }
        };

        try {
            const request: RpcRequest = JSON.parse(msg.content.toString());
            const handler = handlers[request.action];

            if (!handler) {
                response = { error: { code: 'UNKNOWN_ACTION', message: `Unknown action ${request.actio
            } else {
```

Рисунок 3 – Реализация RPC-сервера.

Отклик на вакансию происходит пошагово таким образом:

1. Получение данных из сервиса Users
2. Получение данных из сервиса Jobs
3. Сохранение отклика, если все данные верны

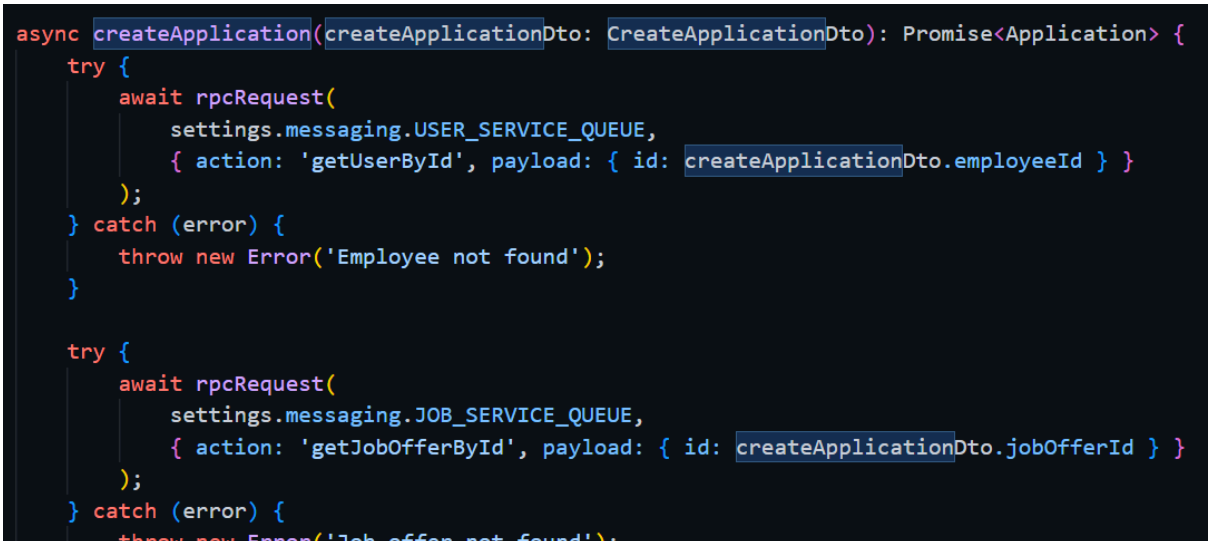


Рисунок 4 – Отклик на вакансию

Вывод

В ходе лабораторной работы был подключен и настроен RabbitMQ, а также реализовано межсервисное взаимодействие с его использованием. Синхронное HTTP-взаимодействие было заменено на асинхронное через RabbitMQ с применением RPC-паттерна, что позволило организовать обмен сообщениями между микросервисами и обеспечить обработку запросов через очереди.