САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

ЛР 6 - Реализация межсервисного взаимодействия посредством очередей сообщений

Выполнил:
Сергеев Виктор
K3441

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

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

Ход работы

Для реализации межсервисного взаимодействия через очередь задач сперва требуется подключить брокер задач. В данном проекте использовался rabbitMQ. Брокер поднимается в докер-контейнере и для этого был добавлен сервис в compose-файл.

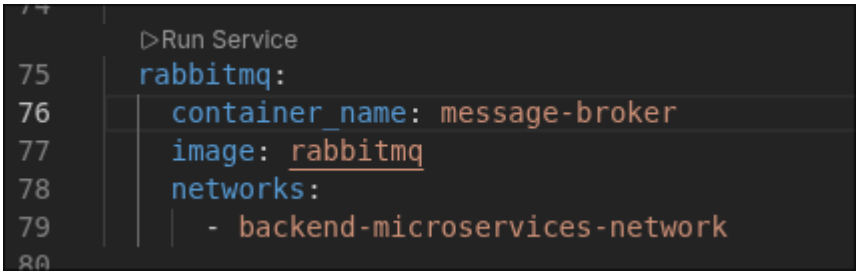

Рисунок 1 - Сервис rabbitMQ в docker-compose

Для использования rabbitMQ в приложении использовалась библиотека amqplib. Сперва микросервис подключается к брокеру - для этого реализована функция connectRabbitMQ.

Для взаимодействия были реализованы функция для send и consume.

gateway_service > src > TS gatewayMessageBroker.ts > consumeUserLoggedChannel > userLoggedChannel.consume("user.loggedIn") callback

```ts
import { Connection, Channel } from "amqplib";
import SETTINGS from "./config/settings";
const amqplib = require("amqplib")

let connection: Connection;
let userLoggedChannel: Channel

async function connectRabbitMQ() {
    try {
        connection = await amqplib.connect(SETTINGS.MESSAGE_BROKER_URL)
        userLoggedChannel = await connection.createChannel()
        await userLoggedChannel.assertQueue("user.loggedIn")
        console.log("[+] Gateway amqp connected successfully")

        await consumeUserLoggedChannel()
    } catch (err) {
        console.log("[-] Gateway amqp connection failed")
        console.log(err.message)
    }
}

async function consumeUserLoggedChannel() {
    await userLoggedChannel.consume("user.loggedIn", (msg) => {
        if (msg !== null) {
            try {
                const data = JSON.parse(msg.content.toString());
                console.log(`[+] User logged in with information: ${JSON.stringify(data)}`)
            } catch (err) {
            console.log("[-] failed to parse message content")
            }
        }
        userLoggedChannel.ack(msg);
    })
}

export default connectRabbitMQ
```

Рисунок 2 - Интерфейс для взаимодействия для gateway-service

user_service > src > TS userMessageBroker.ts > sendUserLoggedMessage

```ts
import { Connection, Channel } from "amqplib";
import SETTINGS from "./config/settings";
const amqplib = require("amqplib")

let connection: Connection;
let userLoggedChannel: Channel

export async function connectRabbitMQ() {
    try {
        connection = await amqplib.connect(SETTINGS.MESSAGE_BROKER_URL)
        userLoggedChannel = await connection.createChannel()
        await userLoggedChannel.assertQueue("user.loggedIn")
        console.log("[+] UserService amqp connected successfully")
    } catch (err) {
        console.log("[-] UserService amqp connection failed")
        console.log(err.message)
    }
}

export async function sendUserLoggedMessage(payload) {
    if (!userLoggedChannel) throw new Error("[-] Sending msg while channel not initialized");
    try {
        await userLoggedChannel.sendToQueue(
            "user.loggedIn",
            Buffer.from(JSON.stringify(payload))
        )
        console.log("[+] User login message sent")
    } catch (err) {
        console.log("[-] User login message sending failed")
        console.log(err)
    }
}
```

Рисунок 3 - Интерфейс для взаимодействия для user-service

Сервис пользователей уведомляет gateway о входе пользователя в аккаунт. Gateway получает сообщение о входе и выводит сообщение в поток вывода.

Рисунок 4 - Лог gateway-service

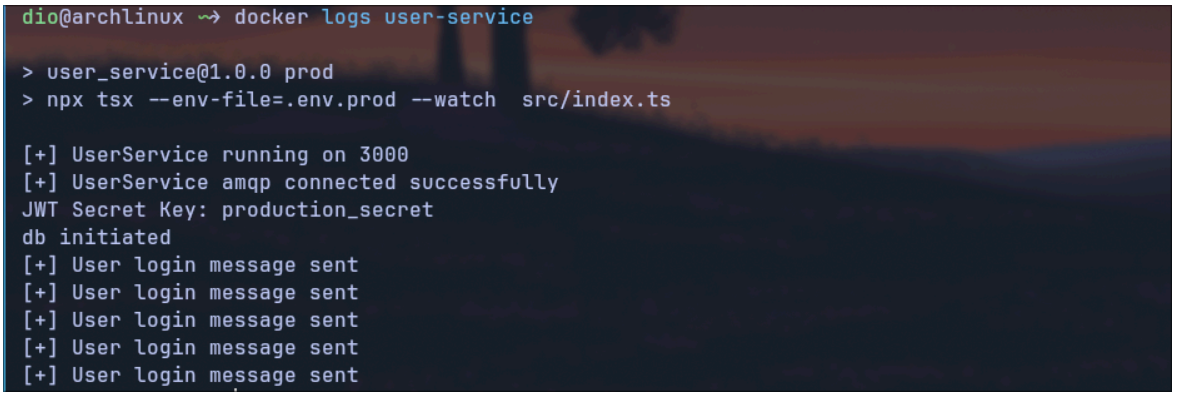

Рисунок 5 - Лог user-service

Вывод

В процессе работы был подключен rabbitMQ и реализовано межсервисное взаимодействие через асинхронную очередь заданий.