

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

**ЛР 6 - Реализация межсервисного взаимодействия
посредством очередей сообщений**

Выполнил:
Сергеев Виктор
К3441

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.

```
74 |  
75 |   >Run Service  
76 |   rabbitmq:  
77 |     container_name: message-broker  
78 |     image: rabbitmq  
79 |     networks:  
80 |       - backend-microservices-network
```

Рисунок 1 - Сервис rabbitMQ в docker-compose

Для использования rabbitMQ в приложении использовалась библиотека amqp-lib. 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 > consumeUserLoggedInChannel > userLoggedInChannel.consume("user.loggedIn") callback
1  import { Connection, Channel } from "amqplib";
2  import SETTINGS from "../config/settings";
3  const amqplib = require("amqplib")
4
5  let connection: Connection;
6  let userLoggedInChannel: Channel
7
8  async function connectRabbitMQ() {
9      try {
10         connection = await amqplib.connect(SETTINGS.MESSAGE_BROKER_URL)
11         userLoggedInChannel = await connection.createChannel()
12         await userLoggedInChannel.assertQueue("user.loggedIn")
13         console.log("[+] Gateway amqp connected successfully")
14
15         await consumeUserLoggedInChannel()
16     } catch (err) {
17         console.log("[-] Gateway amqp connection failed")
18         console.log(err.message)
19     }
20 }
21
22 async function consumeUserLoggedInChannel() {
23     await userLoggedInChannel.consume("user.loggedIn", (msg) => {
24         if (msg !== null) {
25             try {
26                 const data = JSON.parse(msg.content.toString());
27                 console.log(`[+] User logged in with information: ${JSON.stringify(data)}`)
28             } catch (err) {
29                 console.log("[-] failed to parse message content")
30             }
31         }
32         userLoggedInChannel.ack(msg);
33     })
34 }
35
36 export default connectRabbitMQ

```

Рисунок 2 - Интерфейс для взаимодействия для gateway-service

```

user_service > src > TS userMessageBroker.ts > sendUserLoggedInMessage
1  import { Connection, Channel } from "amqplib";
2  import SETTINGS from "../config/settings";
3  const amqplib = require("amqplib")
4
5  let connection: Connection;
6  let userLoggedInChannel: Channel
7
8  export async function connectRabbitMQ() {
9      try {
10         connection = await amqplib.connect(SETTINGS.MESSAGE_BROKER_URL)
11         userLoggedInChannel = await connection.createChannel()
12         await userLoggedInChannel.assertQueue("user.loggedIn")
13         console.log("[+] UserService amqp connected successfully")
14     } catch (err) {
15         console.log("[-] UserService amqp connection failed")
16         console.log(err.message)
17     }
18 }
19
20 export async function sendUserLoggedInMessage(payload) {
21     if (!userLoggedInChannel) throw new Error("[-] Sending msg while channel not initialized");
22     try {
23         await userLoggedInChannel.sendToQueue(
24             "user.loggedIn",
25             Buffer.from(JSON.stringify(payload))
26         )
27         console.log("[+] User login message sent")
28     } catch (err) {
29         console.log("[-] User login message sending failed")
30         console.log(err)
31     }
32 }
33

```

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.

```
dio@archlinux ~-> docker logs gateway-service

> user_service@1.0.0 prod
> npx tsx --env-file=.env.prod --watch src/index.ts

API Gateway running on port 3000
[+] Gateway amqp connected successfully
[+] User logged in with information: {"userId":1,"timestamp":"2026-01-17T15:32:26.486Z"}
[+] User logged in with information: {"userId":1,"timestamp":"2026-01-17T16:32:50.162Z"}
```

Рисунок 4 - Лог gateway-service

```
dio@archlinux ~-> docker logs user-service

> user_service@1.0.0 prod
> npx tsx --env-file=.env.prod --watch src/index.ts

[+] UserService running on 3000
[+] UserService amqp connected successfully
JWT Secret Key: production_secret
db initiated
[+] User login message sent
[+] User login message sent
[+] User login message sent
[+] User login message sent
[+] User login message sent
```

Рисунок 5 - Лог user-service

Вывод

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