**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Лабораторная работа №6

Выполнил:

Платонова Александра

Группа K3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

**Задача**

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

Проект: Прикладное программное обеспечение деятельности отдела заселения муниципальных общежитий администрации города.

**Ход работы**

Для обеспечения асинхронного взаимодействия между микросервисами был выбран брокер сообщений RabbitMQ. Выбор RabbitMQ обусловлен его простотой внедрения, хорошей поддержкой различных паттернов обмена сообщениями и достаточной производительностью для задач данного проекта.

Для начала была создана конфигурация подключения к RabbitMQ (рис. 1).

```
export const rabbitConfig = {
    url: process.env.RABBITMQ_URL || 'amqp://guest:guest@localhost:5672',
    exchange: 'hostel.events',
    exchangeType: 'topic',
    reconnectIntervalMs: 5000,
};
```

Рисунок 1 – Конфигурация RabbitMQ

Далее был реализован модуль для публикации событий.

```
import * as amqp from 'amqplib';

import { rabbitConfig } from '../../config/rabbitmq.config';

let channel: amqp.Channel | null = null;

export async function initPublisher(): Promise<void> {

    const connection = await amqp.connect(rabbitConfig.url);

    channel = await connection.createChannel();

            await    channel.assertExchange(rabbitConfig.exchange,
rabbitConfig.exchangeType, { durable: true });
```

```
}

export async function publishEvent(routingKey: string, payload:
any): Promise<void> {

   if (!channel) throw new Error('Publisher not initialized');

   const message = Buffer.from(JSON.stringify(payload));

    channel.publish(rabbitConfig.exchange, routingKey, message, {
persistent: true });

}
```

Пример публикации события:

```
await publishEvent('contract.created', {

   contractId,

   roomId,

   residentId,

   timestamp: new Date().toISOString(),

   action: 'check-in'

});
```

Пример консьюмера для события:

```
const queue = await channel.assertQueue('hostel.room-updates', {
durable: true });

await channel.bindQueue(queue.queue, rabbitConfig.exchange,
'room.status_changed');

channel.consume(queue.queue, async (msg) => {

  if (!msg) return;

  const event = JSON.parse(msg.content.toString());

  channel.ack(msg);

}, { noAck: false });
```

Асинхронная публикация событий позволяет выделить критические синхронные операции от уведомлений. Это повышает отказоустойчивость системы и упрощает добавление новых обработчиков событий в будущем.

**Вывод**

В ходе выполнения лабораторной работы была успешно реализована асинхронная система межсервисного взаимодействия на базе RabbitMQ в микросервисной архитектуре управления муниципальными общежитиями. Применение topic exchange и persistent сообщений обеспечило надежную доставку событий и устойчивость системы к временным сбоям сети. Разделение синхронных критических операций (HTTP) и асинхронных уведомлений позволило повысить производительность и отказоустойчивость приложения в целом. Полученный опыт подтверждает целесообразность использования RabbitMQ для задач с умеренной нагрузкой и сложной бизнес-логикой распределенного взаимодействия.