САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №6

Выполнил:

Бархатова Наталья

Группа К3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

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

Ход работы

Добавление RabbitMQ в DockerFile. Для организации обмена сообщениями между микросервисами был выбран RabbitMQ. В docker-compose.yml был добавлен сервис RabbitMQ с административным интерфейсом. Порт 5672 используется для передачи сообщений, а порт 15672 позволяет получить доступ к веб-интерфейсу брокера для мониторинга очередей и сообщений. После поднятия контейнера RabbitMQ стал доступен для подключения всех микросервисов, начиная с сервиса аутентификации.

Листинг 1 – docker-compose.yml (RabbitMQ)

```
services:
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest
```

RabbitMQ подключался в сервис аутентификации. На старте каждого сервиса выполняется подключение к RabbitMQ. Для этого используется helper-функция connectRabbit, которая создаёт соединение и канал для работы с брокером. После успешного подключения сервис готов к

публикации и потреблению событий, что обеспечивает асинхронное взаимодействие между компонентами системы.

Листинг 2 – Инициализация подключения

```
src/app.ts

import { connectRabbit } from './lib/rabbit';

const rabbitUrl = process.env.RABBITMQ_URL || 'amqp://localhost';

await connectRabbit(rabbitUrl);
```

Листинг 3 – Хелпер для работы с RabbitMQ

```
src/lib/rabbit.ts

import amqp from 'amqplib';

let channel: amqp.Channel;

export async function connectRabbit(url: string) {

  const connection = await amqp.connect(url);

  channel = await connection.createChannel();

   await channel.assertExchange('events', 'topic', { durable: false });

  return { connection, channel };

}

export async function publishEvent(routingKey: string, payload: any) {

                  channel.publish('events', routingKey, Buffer.from(JSON.stringify(payload)));

}

export async function subscribe(routingKey: string, handler: (msg: any) => void) {

   const q = await channel.assertQueue('', { exclusive: true });
```

```
  await channel.bindQueue(q.queue, 'events', routingKey);

            channel.consume(q.queue,       msg       =>       {
handler(JSON.parse(msg.content.toString()));  channel.ack(msg);
});

}
```

Модуль управляет подключением и каналом к RabbitMQ с использованием библиотеки amqplib. Он создаёт topic exchange с именем events, через который происходит маршрутизация сообщений между сервисами. Функции connectRabbit, publishEvent и subscribe позволяют микросервисам публиковать события и подписываться на них для асинхронного взаимодействия.

Листинг 3 – Пример consumer

src/consumers/WorkoutCompletedConsumer.ts

```
import { subscribe } from '../lib/rabbit';


export async function startWorkoutCompletedConsumer() {

  await subscribe('workout.completed', async (data) => {

        console.log('[Consumer]  workout.completed  received:',
data);

  });

}
```

Для события workout.completed создаётся временная эксклюзивная очередь, чтобы каждый потребитель получал свои копии сообщений. В обработчике выполняется бизнес-логика микросервиса, включая обновление статистики пользователя и другие действия, связанные с завершением тренировки. Микросервис, который фиксирует окончание тренировки, публикует соответствующее событие для оповещения других сервисов, таких как сервис уведомлений или начисления баллов.

Вывод

В ходе работы была реализована микросервисная архитектура фитнес-приложения с использованием Docker и RabbitMQ, обеспечивающая асинхронное взаимодействие между сервисами. Публикация и подписка на события через topic exchange позволила каждому компоненту системы работать независимо и получать необходимые данные без дублирования.