САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа #6

Выполнил:
Космач Мария Романовна

Группа:
K3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

● подключить и настроить rabbitMQ/kafka;

● реализовать межсервисное взаимодействие посредством rabbitMQ/kafka.

Ход работы

Для разворачивания rabbitMQ была добавлена следующая конфигурация в docker-compose

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

в shared-модуле был добавлен файл rabbit.ts, который предоставляет методы для получения канала, а также публикации и чтения очереди

```
import amqp, { Connection, Channel, ConsumeMessage } from "amqplib";
const RABBIT_URL = process.env.RABBIT_URL ??
"amqp://guest:guest@localhost:5672";

let conn: Connection | null = null;
let channel: Channel | null = null;

export async function getChannel(): Promise<Channel> {
  if (channel) return channel;
```

```
  conn = await amqp.connect(RABBIT_URL);
  channel = await conn.createChannel();
  channel.prefetch(10);

  return channel;
}

export async function publishToQueue(queue: string, payload: unknown) {
  const ch = await getChannel();
  await ch.assertQueue(queue, { durable: true });

  const body = Buffer.from(JSON.stringify(payload));
  ch.sendToQueue(queue, body, {
    persistent: true,
    contentType: "application/json",
  });
}

export async function consumeQueue(
  queue: string,
  handler: (data: any) => Promise<void>
) {
  const ch = await getChannel();
  await ch.assertQueue(queue, { durable: true });

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

    try {
      const raw = msg.content.toString("utf-8");
      const data = JSON.parse(raw);
```

```
      await handler(data);
      ch.ack(msg);
    } catch (e) {
      ch.nack(msg, false, false);
    }
  });
}
```

при создании объявления отправляется событие advertisement.created

```
import { publishToQueue } from "../messaging/rabbit";

export async function createAdvertisement(dto: Advertisement) {
  const created = { dto };

  await publishToQueue("advertisement.created", {
    eventId: crypto.randomUUID?.() ?? String(Date.now()),
    occurredAt: new Date().toISOString(),
    payload: {
      advertisementId: created.id,
      ownerId: created.ownerId,
      title: created.title,
    },
  });

  return created;
}
```

для считывания ивента из очереди создан consumer

```
import { consumeQueue } from "../messaging/rabbit";
```

```
export async function startConsumers() {
  await consumeQueue("advertisement.created", async (event) => {
    console.log("[notification-service] got event:", event);
  });
}
```

Вывод

В результате выполнения работы было настроено взаимодействие микросервисов с использованием брокера сообщений RabbitMQ и реализован механизм публикации событий, обеспечивающий асинхронный обмен данными между сервисами системы.