САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 6

Выполнил:

Беломытцев Андрей

К3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Реализация межсервисного взаимодействия посредством очередей сообщений

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

Ход работы

Сетевое взаимодействие реализовано с помощью RabbitMQ.

RabbitMQ подключён с помощью docker compose.

```
rabbitmq:
  image: rabbitmq:management
  ports:
    - '5672:5672'
    - '15672:15672'
  environment:
    RABBITMQ_DEFAULT_USER: ${RABBITMQ_DEFAULT_USER}
    RABBITMQ_DEFAULT_PASS: ${RABBITMQ_DEFAULT_PASS}
  networks:
    - backend
  healthcheck:
    test: rabbitmq-diagnostics -q ping
    interval: 30s
    timeout: 30s
    retries: 3

user-service:
  container_name: user-service
  build:
    context: ./user-service
    dockerfile: Dockerfile
  depends_on:
    user-db:
      condition: service_healthy
    rabbitmq:
      condition: service_healthy
  env_file:
    - .env
  ports:
    - '3000:3000'
  networks:
    - backend
```

В микросервисах создан файл rabbit.ts с функциями sendToQueue и listenToQueue используемыми для взаимодействия с RabbitMQ с использованием amqplib.

```
import amqplib from 'amqplib'
```

```
export const sendToQueue = async (queue: string, message: any) => {
  const conn = await amqplib.connect('amqp://rabbitmq')
  const channel = await conn.createChannel()
  await channel.assertQueue(queue, { durable: true })
  channel.sendToQueue(queue, Buffer.from(JSON.stringify(message)))
  console.log(`Message sent to ${queue}:`, message)
  await channel.close()
  await conn.close()
}

export const listenToQueue = async (queue: string, callback: (content: any)
=> any) => {
  const conn = await amqplib.connect('amqp://rabbitmq')
  const channel = await conn.createChannel()
  await channel.assertQueue(queue, { durable: true })
  channel.consume(queue, (msg) => {
    if (msg) {
      const content = JSON.parse(msg.content.toString())
      console.log(`Received from ${queue}:`, content)
      callback(content)
      channel.ack(msg)
    }
  })
}
```

Реализована передача сообщений от channel-service к video-service. В моменты добавления и удаления канала, соотвествующие видео должны добавляться и удаляться соотвественно. Реализуя нечто вроде cascade, который использовался, когда API ещё было монолитным.

Следующий код добавлен в channel-service

```
import { sendToQueue, listenToQueue } from '../rabbit'
```

Запускается при добавлении канала

```
await sendToQueue('add_videos', { channelId: channelId })
```

Запускается при удалении канала

```
await sendToQueue('delete_videos', { channelId: id })
```

Следующий код добавлен в video-service

```
import { sendToQueue, listenToQueue } from './rabbit'

const repository = AppDataSource.getRepository(Video)

const getVideos = async (channelId: string, maxResults: number = 50) => {
  const uploads = 'UULF' + channelId.slice(2)
  const videos: any = await (await
fetch(`https://www.googleapis.com/youtube/v3/playlistItems?part=snippet%2Ccon
tentDetails&maxResults=${maxResults}&playlistId=${uploads}&key=${config.YT_AP
I_KEY}`)).json()
  const videosList: Video[] = []
```

```
  for(let m of videos['items']){
    m = m['snippet']
    videosList.push({
      'id': m['resourceId']['videoId'],
      'channelId': m['channelId'],
      'title': m['title'],
      'publishedAt': m['publishedAt'],
      'thumbnail': m['thumbnails']['maxres' in m['thumbnails'] ? 'maxres' :
'medium']['url'],
      'description': m['description'],
    } as Video)
  }
  return repository.save(videosList)
}

const deleteVideos = async (channelId: string) => {
  await repository.delete({ channelId: channelId })
}

listenToQueue('add_videos', (content) => getVideos(content.channelId))
listenToQueue('delete_videos', (content) => deleteVideos(content.channelId))
```

Протестирована работа API с помощью расширения REST Client для VS Code. Для проверки взаимодействия всех микросервисов между собой проведены следующие тесты:

Регистрация

```
POST http://127.0.0.1:3000/user/register
Content-Type: application/json

{
  "username": "andrei",
  "email": "andrei@example.com",
  "password": "qwerty"
}
```

Получение JWT

```
POST http://127.0.0.1:3000/user/login
Content-Type: application/json

{
  "username": "andrei",
  "password": "qwerty"
}
```

Добавление канала

```
POST http://127.0.0.1:3001/channel
Authorization: Bearer ...
Content-Type: application/json

{
  "id": "UCHnyfMqiRRG1u-2MsSQLbXA",
  "lang": "en",
  "category": "popsci",
```

```
  "theme": "all"
}
```

Проверка появился ли канал (да)

```
GET http://127.0.0.1:3001/channel
```

Проверка появились ли видео (да)

```
GET http://127.0.0.1:3002/video
```

Удаление видео (проверено, что истёкший JWT или JWT без нужных прав не работает)

```
DELETE http://127.0.0.1:3001/channel/UCHnyfMqiRRG1u-2MsSQLbXA
Authorization: Bearer ...
Content-Type: application/json
```

Проверка пропал ли канал (да)

```
GET http://127.0.0.1:3001/channel
```

Проверка пропали ли видео (да)

```
GET http://127.0.0.1:3002/video
```

Вывод

В результате было реализовано межсервисное взаимодействия посредством очередей сообщений с помощью RabbitMQ. Подключён и настроен RabbitMQ с использованием Docker и amqplib. Протестирована работа API.