САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая/Лабораторная работа

Выполнил:

Габов Михаил

К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задание

Целью данного задания было подключение и настройка RabbitMQ/Kafka, а также реализация межсервисного взаимодействия с использованием выбранной системы очередей сообщений. В рамках данной работы был выбран RabbitMQ.

1. Подключение и настройка RabbitMQ

1.1. Настройка RabbitMQ через Docker Compose

RabbitMQ был интегрирован в микросервисную архитектуру с использованием docker-compose.yml.

В файле docker-compose.yml RabbitMQ определен как сервис rabbitmq:

```
rabbitmq:
  image: rabbitmq:3-management-alpine
  container_name: rabbitmq
  hostname: rabbitmq
  ports:
    - "5672:5672"
    - "15672:15672"
  environment:
    RABBITMQ_DEFAULT_USER: user
    RABBITMQ_DEFAULT_PASS: password
  volumes:
    - rabbitmq-data:/var/lib/rabbitmq/
    - rabbitmq-log:/var/log/rabbitmq
  networks:
    - my-network
  healthcheck:
    test: ["CMD", "rabbitmq-diagnostics", "ping"]
    interval: 30s
    timeout: 10s
    retries: 5
```

Дополнительно настроена проверка работоспособности (healthcheck) для правильной последовательности запуска зависимых микросервисов.

1.2. Подключение микросервисов к RabbitMQ

Каждый микросервис, использующий RabbitMQ (auth-users-service, recipes-service, interactions-service, api-gateway), настраивается для подключения к нему через переменные окружения, определенные в docker-compose.yml:

- RABBITMQ_HOST: rabbitmq (имя хоста контейнера RabbitMQ в Docker сети)
- RABBITMQ_PORT: 5672
- RABBITMQ_USER: user
- RABBITMQ_PASS: password

Модуль src/rabbitmq-config.ts в сервисах recipes и interactions содержит универсальную функцию connectRabbitMQ для установления соединения:

Пример из src/rabbitmq-config.ts
```
import amqp from 'amqplib';
import 'dotenv/config';

const RABBITMQ_HOST = process.env.RABBITMQ_HOST || 'localhost';
const RABBITMQ_PORT = +(process.env.RABBITMQ_PORT || 5672);
const RABBITMQ_USER = process.env.RABBITMQ_USER || 'user';
const RABBITMQ_PASS = process.env.RABBITMQ_PASS || 'password';

const RABBITMQ_URL =
`amqp://${RABBITMQ_USER}:${RABBITMQ_PASS}@${RABBITMQ_HOST
}:${RABBITMQ_PORT}`;

export const connectRabbitMQ = async (retries = 5, delay = 5000) => {
```

```
    ...
};
```

2. Реализация межсервисного взаимодействия

2.1. Сервис-производитель: recipes-service

Сервис recipes-service выступает в роли производителя сообщений. В частности, при создании нового рецепта он публикует событие в RabbitMQ.

Место реализации: services/recipes/src/controllers/recipe.controller.ts

В методе createRecipe после успешного сохранения нового рецепта в базе данных, сервис отправляет сообщение в очередь:

services/recipes/src/controllers/recipe.controller.ts

```
import { rabbitMQChannel } from '../index';

 внутри метода createRecipe

if (rabbitMQChannel) {
    const queue = 'new_recipe_events';
    const msg = JSON.stringify({
        type: 'RecipeCreated',
        recipeId: savedRecipe.id,
        title: savedRecipe.title,
        userId: savedRecipe.userId,
        createdAt: savedRecipe.createdAt,
    });
    await rabbitMQChannel.assertQueue(queue, { durable: true });
    rabbitMQChannel.sendToQueue(queue, Buffer.from(msg))
    console.log(`[Recipes Service] Sent message to ${queue}: ${msg}`);
} else {
    console.warn('[Recipes Service] RabbitMQ channel not available. Message for new recipe not sent.');
```

}

Сообщение содержит информацию о только что созданном рецепте. Очередь 'new_recipe_events' объявлена как durable (сохранится после перезапуска RabbitMQ)

2.2. Сервис-потребитель: interactions-service

Сервис interactions-service выступает в роли потребителя сообщений, подписываясь на события о создании новых рецептов.

Место реализации: services/interactions/src/index.ts

В файле index.ts этого сервиса, после успешного подключения к базе данных, устанавливается соединение с RabbitMQ, и затем настраивается потребитель для очереди new_recipe_events:

services/interactions/src/index.ts
```
import { connectRabbitMQ } from './rabbitmq-config';

внутри AppDataSource.initialize().then()

try {
   const { channel } = await connectRabbitMQ();
   (app as any).rabbitMQChannel = channel;

   const queue = 'new_recipe_events';
   await channel.assertQueue(queue, { durable: true });
   console.log(`[Interactions Service] Waiting for messages in queue: ${queue}. To exit press CTRL+C`);

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

```
            try {
                const event = JSON.parse(msg.content.toString());
                console.log(`[Interactions Service] Received message:`, event);
                channel.ack(msg);
                console.log(`[Interactions Service] Message processed and
acknowledged.`);
            } catch (parseError) {
                console.error('[Interactions Service] Error parsing message:', parseError);
                channel.reject(msg, false);
            }
        }
    });

} catch (error) {
    console.error("Failed to connect to RabbitMQ, interactions service might not be
fully functional for messaging.", error);
}
```

При получении сообщения из очереди, interactions-service парсит его содержимое и логирует его.

Заключение

Реализована асинхронная межсервисная коммуникация с использованием RabbitMQ. Сервис recipes-service выступает в роли производителя событий о создании новых рецептов, публикуя их в очередь new_recipe_events. Сервис interactions-service подписан на эту очередь и потребляет эти события