САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэкэнд разработка

Отчет

Лабораторная работа №6

Выполнил:

Габов Михаил

Группа K3440

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Цель работы

Внедрить асинхронное межсервисное взаимодействие через очереди сообщений RabbitMQ

Реализация RabbitMQ в Docker Compose

```yaml
services:
 rabbitmq:
  image: rabbitmq:3-management-alpine
  container_name: rabbitmq
  ports:
   - "5672:5672"
   - "15672:15672"
  environment:
   RABBITMQ_DEFAULT_USER: admin
   RABBITMQ_DEFAULT_PASS: admin
  healthcheck:
   test: ["CMD", "rabbitmq-diagnostics", "check_port_connectivity"]
   interval: 10s
   timeout: 5s
   retries: 5
  networks:
   - app-network

 recipes-service:
  depends_on:
   rabbitmq:
    condition: service_healthy
  environment:
   RABBITMQ_URL: amqp://admin:admin@rabbitmq:5672

 interactions-service:
  depends_on:
   rabbitmq:
    condition: service_healthy
  environment:
   RABBITMQ_URL: amqp://admin:admin@rabbitmq:5672
  notifications-service:
    build:
      context: ./services/notifications-service
    depends_on:
      rabbitmq:
        condition: service_healthy
    environment:
      RABBITMQ_URL: amqp://admin:admin@rabbitmq:5672
      AUTH_SERVICE_URL: http://auth-users-service:3001
      RECIPES_SERVICE_URL: http://recipes-service:3002
    networks:
      - app-network
```

Healthcheck: гарантирует, что RabbitMQ готов до запуска сервисов

Management UI: веб-интерфейс на порту 15672 (admin/admin)

Alpine образ: легкий образ с минимальным размером

Producer: Публикация событий

Пример с событием RecipeCreated

```
import amqp, { Channel, Connection } from 'amqplib';

let connection: Connection;
let channel: Channel;

export async function initRabbitMQ(url: string): Promise<void> {
  try {
    connection = await amqp.connect(url);
    channel = await connection.createChannel();

    await channel.assertQueue('new_recipe_events', { durable: true });

        console.log('[RabbitMQ] Connected and queue "new_recipe_events" created');
  } catch (error) {
        console.error('[RabbitMQ] Connection error:', error);
        throw error;
    }
}

export function getRabbitMQChannel(): Channel {
    if (!channel) {
        throw new Error('RabbitMQ channel not initialized');
    }
    return channel;
}
```

Публикация события при создании рецепта:

```
import { getRabbitMQChannel } from '../rabbitmq-config';

@Route('recipes')
@Tags('Recipes')
export class RecipeController extends Controller {

    @Post('/')
    public async createRecipe(
        @Body() body: RecipeCreateRequest,
        @Header('x-user-id') userId: number
    ): Promise<RecipeResponse> {
        const recipe = this.recipeRepository.create({
            title: body.title,
            description: body.description,
            difficulty: body.difficulty,
            userId: userId
        });
        const savedRecipe = await this.recipeRepository.save(recipe);

        const rabbitMQChannel = getRabbitMQChannel();
        const queue = 'new_recipe_events';

        const message = JSON.stringify({
            type: 'RecipeCreated',
            recipeId: savedRecipe.id,
            title: savedRecipe.title,
            userId: savedRecipe.userId,
            timestamp: new Date().toISOString()
        });

        rabbitMQChannel.sendToQueue(queue, Buffer.from(message), {
            persistent: true
        });

        console.log('[RabbitMQ] Published RecipeCreated event:', message);

        return {
            id: savedRecipe.id,
            title: savedRecipe.title,
            // ...
        };
    }
}
```

Consumer: Обработка событий

```
import amqp from 'amqplib';
import axios from 'axios';

const RABBITMQ_URL = process.env.RABBITMQ_URL ||
'amqp://admin:admin@rabbitmq:5672';
const AUTH_SERVICE_URL = process.env.AUTH_SERVICE_URL ||
'http://auth-users-service:3001';
const RECIPES_SERVICE_URL =
process.env.RECIPES_SERVICE_URL || 'http://recipes-service:3002';

async function start() {
    try {
        const connection = await amqp.connect(RABBITMQ_URL);
        const channel = await connection.createChannel();
    console.log('[Notifications Service] Connected to RabbitMQ');

    await channel.assertQueue('new_recipe_events', { durable: true });

    channel.consume('new_recipe_events', async (msg) => {
      if (msg) {
        try {
          const event = JSON.parse(msg.content.toString());

          const userResponse = await axios.get(
            `${AUTH_SERVICE_URL}/users/${event.userId}`
          );
          const username = userResponse.data.username;

            console.log(`[Notification] New recipe created: "${event.title}"
by ${username} (ID: ${event.userId}, Recipe ID: ${event.recipeId})`);

          channel.ack(msg);

        } catch (error) {
            console.error('[Notifications] Error processing RecipeCreated
event:', error);
          channel.reject(msg, false);
        }
      }
    });

    await channel.assertQueue('interactions_events', { durable: true });

    channel.consume('interactions_events', async (msg) => {
      if (msg) {
        try {
          const event = JSON.parse(msg.content.toString());

          if (event.type === 'LikeCreated') {
            const { userId, recipeId } = event.data;
            const recipeResponse = await axios.get(
              `${RECIPES_SERVICE_URL}/recipes/${recipeId}`
            );
            const recipe = recipeResponse.data;

            const authorResponse = await axios.get(
              `${AUTH_SERVICE_URL}/users/${recipe.userId}`
            );
            const authorUsername = authorResponse.data.username;

            console.log(`[Notification] New like on recipe "${recipe.title}" by user ID ${userId} (author: ${authorUsername})`);

            channel.ack(msg);
          }

        } catch (error) {
          console.error('[Notifications] Error processing interaction event:', error);
          channel.reject(msg, false);
        }
      }
    });

    console.log('[Notifications Service] Waiting for messages...');

  } catch (error) {
    console.error('[Notifications Service] Fatal error:', error);
    process.exit(1);
  }
}

start();
```

Заключение

В рамках лабораторной работы успешно внедрено асинхронное межсервисное взаимодействие через RabbitMQ. Реализована Event-Driven архитектура с двумя типами событий:

RecipeCreated - публикуется Recipes Service при создании рецепта

LikeCreated - публикуется Interactions Service при лайке рецепта

Создан consumer-only сервис Notifications Service, который обрабатывает эти события и логирует уведомления (в продакшене это были бы email/push/WebSocket).