САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №6

Выполнил:

Цой Степан

Группа

К3440

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

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

Ход работы

В файл docker-compose.yml добавлен сервис RabbitMQ с портами для AMQP протокола и веб-интерфейса управления.

```yaml
rabbitmq:
  image: rabbitmq:3-management
  ports:
    - "5672:5672"
    - "15672:15672"
  environment:
    - RABBITMQ_DEFAULT_USER=admin
    - RABBITMQ_DEFAULT_PASS=admin123
  healthcheck:
    test: rabbitmq-diagnostics -q ping
    interval: 10s
    timeout: 5s
    retries: 10
    start_period: 40s
  networks:
    - recipe-network
```

```
C:\Users\stepa\recipe-service>docker-compose up -d
[+] Running 5/5
 ✓ Container recipe-service-rabbitmq-1            Healthy                                                  6.4s
 ✓ Container recipe-service-interaction-service-1 Started                                                  6.9s
 ✓ Container recipe-service-auth-service-1        Started                                                  6.8s
 ✓ Container recipe-service-content-service-1     Started                                                  7.0s
 ✓ Container recipe-service-gateway-1             Started                                                  7.4s

C:\Users\stepa\recipe-service>docker-compose ps
NAME                                    IMAGE                                  COMMAND                  SERVICE              CR
EATED          STATUS                    PORTS
recipe-service-auth-service-1           recipe-service-auth-service            "docker-entrypoint.s…"   auth-service         25
 minutes ago   Up 3 seconds              0.0.0.0:3001->3001/tcp
recipe-service-content-service-1        recipe-service-content-service         "docker-entrypoint.s…"   content-service      25
 minutes ago   Up 3 seconds              0.0.0.0:3002->3002/tcp
recipe-service-gateway-1                recipe-service-gateway                 "docker-entrypoint.s…"   gateway              25
 minutes ago   Up 2 seconds              0.0.0.0:3000->3000/tcp
recipe-service-interaction-service-1    recipe-service-interaction-service     "docker-entrypoint.s…"   interaction-service  25
 minutes ago   Up 3 seconds              0.0.0.0:3003->3003/tcp
recipe-service-rabbitmq-1               rabbitmq:3-management                  "docker-entrypoint.s…"   rabbitmq             25
 minutes ago   Up 9 seconds (healthy)    4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.
0.0.0:15672->15672/tcp
```

В каждом микросервисе создан файл rabbitmq.ts, содержащий логику подключения к брокеру сообщений, создания каналов и объявления exchanges. (Пример Auth Service)

```ts
export const connectRabbitMQ = async () => {
  const maxRetries = 5;
  const retryDelay = 3000;

  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      const rabbitmqUrl = process.env.RABBITMQ_URL || "amqp://rabbitmq:5672";
      console.log(`🔄 RabbitMQ connection attempt ${attempt}/${maxRetries}...`);

      connection = await amqp.connect(rabbitmqUrl);
      channel = await connection.createChannel();

      await channel.assertExchange("user_events", "topic", { durable: true });
      await channel.assertExchange("recipe_events", "topic", { durable: true });

      console.log("✅ Auth Service connected to RabbitMQ");
      return channel;
    } catch (error: any) {
      console.error(
        `❌ RabbitMQ connection attempt ${attempt} failed:`,
        error.message
      );

      if (attempt === maxRetries) {
        throw error;
      }

      console.log(`⏳ Retrying in ${retryDelay / 1000} seconds...`);
      await new Promise((resolve) => setTimeout(resolve, retryDelay));
    }
  }
```

```ts
export const publishToExchange = async (
  exchange: string,
  routingKey: string,
  message: any
) => {
  try {
    if (!channel) {
      console.log(
        "⚠️ RabbitMQ channel not available, skipping event publishing"
      );
      return;
    }
    channel.publish(
      exchange,
      routingKey,
      Buffer.from(JSON.stringify(message)),
      {
        persistent: true,
      }
    );
    console.log(`📤 Event published: ${exchange} - ${routingKey}`);
  } catch (error: any) {
    console.error("Error publishing event:", error.message);
  }
};
```

Создана система событий для асинхронного взаимодействия между сервисами. Auth Service публикует события пользователей, Content Service - события рецептов.

```
export const publishUserUpdated = async (user: User) => {
  const event = {
    type: "USER_UPDATED",
    data: {
      user_id: user.user_id,
      username: user.username,
      email: user.email,
      profile_photo: user.profile_photo,
      bio: user.bio,
      updated_at: new Date().toISOString(),
    },
    timestamp: new Date().toISOString(),
  };

  await publishToExchange("user_events", "user.updated", event);
};
```

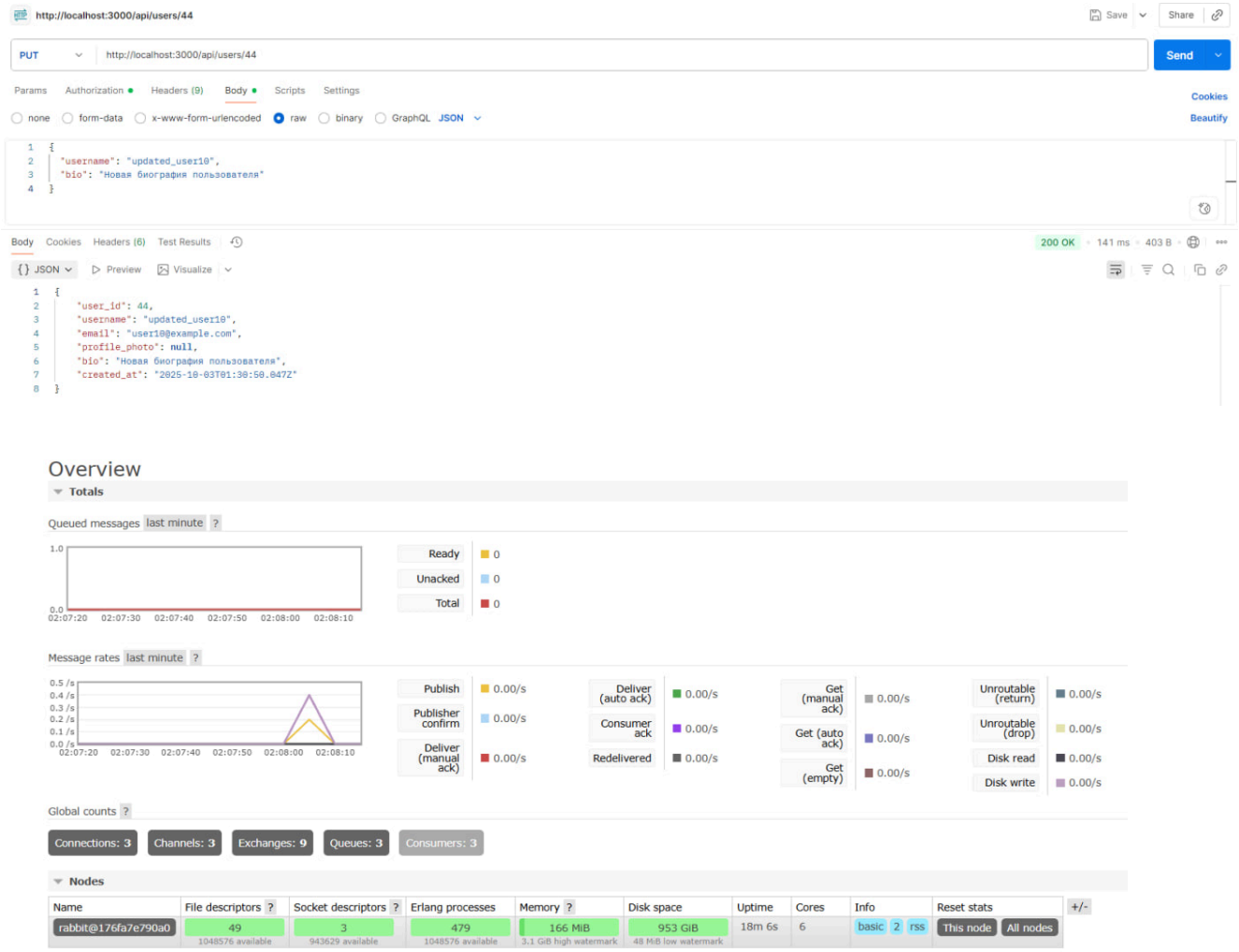
В контроллеры добавлены вызовы публикации событий после выполнения основных операций с данными.

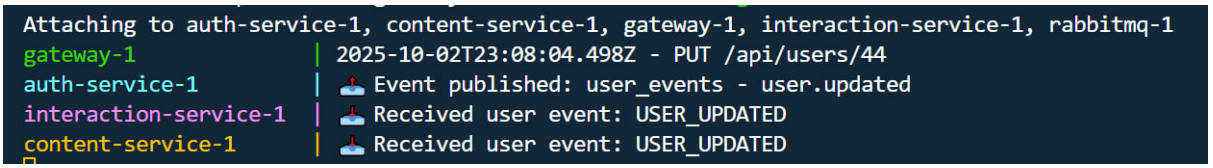
```
  await publishUserUpdated(user);

  res.json({
    user_id: user.user_id,
    username: user.username,
    email: user.email,
    profile_photo: user.profile_photo,
    bio: user.bio,
    created_at: user.created_at,
  });
} catch (error: any) {
  if (error.code === "23505") {
    return res.status(400).json({ message: "Username already exists" });
  }
  res
    .status(500)
    .json({ message: error.message || "Internal server error" });
}
```

Сервисы Content и Interaction настроены как потребители событий, подписанные на соответствующие routing keys.

```
channel.consume(userQueue.queue, (msg: any) => {
  if (msg) {
    try {
      const event = JSON.parse(msg.content.toString());
      console.log("📥 Received user event:", event.type);
      channel.ack(msg);
    } catch (error: any) {
      console.error("Error processing user event:", error.message);
      channel.nack(msg);
    }
  }
});

channel.consume(recipeQueue.queue, (msg: any) => {
  if (msg) {
    try {
      const event = JSON.parse(msg.content.toString());
      console.log("📥 Received recipe event:", event.type);

      if (event.type === "RECIPE_DELETED") {
        console.log(`Recipe ${event.data.recipe_id} was deleted`);
      }

      channel.ack(msg);
    } catch (error: any) {
      console.error("Error processing recipe event:", error.message);
      channel.nack(msg);
    }
```

Проведено тестирование системы для проверки корректной работы межсервисного взаимодействия через RabbitMQ.





Вывод

В результате лабораторной работы успешно реализовано асинхронное межсервисное взаимодействие с использованием RabbitMQ.