САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа №6

Выполнил:

Захарчук Александр

Группа К3441

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

Целью данной работы являлась переработка взаимодействия между микросервисами user-service, recipe-service и social-service с целью снижения связности, повышения масштабируемости и отказоустойчивости системы. Для этого было реализовано асинхронное взаимодействие сервисов с использованием брокера сообщений и событийно-ориентированной архитектуры.

Ход работы

Изначально система представляла собой набор из трёх микросервисов, взаимодействующих преимущественно через HTTP API:

- user-service - управление пользователями, аутентификация, подписки;
- recipe-service - управление рецептами и ингредиентами;
- social-service - лайки и комментарии.

Все сервисы:

- имели собственные базы данных PostgreSQL;
- были доступны через NGINX (API Gateway);
- логически зависели друг от друга через идентификаторы пользователей и рецептов.

Для устранения указанных проблем была выбрана событийно-ориентированная архитектура с использованием брокера сообщений RabbitMQ.

В файл docker-compose.yaml был добавлен сервис RabbitMQ (рисунок 1).

```yaml
rabbitmq:
  image: rabbitmq:3-management
  ports:
    - "5672:5672"
    - "15672:15672"
  networks:
    - recipes_network_test
```

Рисунок 1 - RabbitMQ в docker-compose

Также во все сервисы были добавлены переменные окружения:

RABBITMQ_URL=amqp://rabbitmq:5672

RABBITMQ_EXCHANGE=recipes.events

В каждом сервисе был реализован модуль инициализации брокера сообщений (рисунок 2).

```javascript
import amqp from "amqplib";

let channel;

export async function initRabbitMQ(url, exchange) {
  const connection = await amqp.connect(url);
  channel = await connection.createChannel();

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

export function getChannel() {
  return channel;
}
```

Рисунок 2 - Модуль инициализации брокера

Инициализация выполняется при старте сервиса.

Для публикации событий был реализован универсальный продюсер (рисунок 3).

```
import { v4 as uuid } from "uuid";
import { getChannel } from "./rabbitmq";

export async function publishEvent(routingKey, eventType, payload) {
  const event = {
    eventId: uuid(),
    eventType,
    occurredAt: new Date().toISOString(),
    payload,
  };

  getChannel().publish(
    process.env.RABBITMQ_EXCHANGE,
    routingKey,
    Buffer.from(JSON.stringify(event)),
    { persistent: true }
  );
}
```

Рисунок 3 - Универсальный продюсер

После успешного создания пользователя публикуется событие UserCreate (рисунок 4).

```
const user = await usersRepo.create(data);

await publishEvent(
  "user.created",
  "UserCreated",
  {
    userId: user.id,
    username: user.username,
    email: user.email,
  }
);
```

Рисунок 4 - Создание пользователя

При удалении пользователя публикуется событие UserDeleted (рисунок 5).

```
await usersRepo.delete(userId);

await publishEvent(
  "user.deleted",
  "UserDeleted",
  { userId }
);
```

Рисунок 5 - Удаление пользователя

Recipe-service подписывается на событие удаления пользователя и удаляет связанные с ним рецепты (рисунок 6).

```
channel.consume(queue, async (msg) => {
  const event = JSON.parse(msg.content.toString());

  if (event.eventType === "UserDeleted") {
    await recipesRepo.deleteByAuthorId(event.payload.userId);
  }

  channel.ack(msg);
});
```

Рисунок 6 - Обработка удаления пользователя

При получении события RecipeDeleted social-service удаляет связанные лайки и комментарии (рисунок 7).

```
if (event.eventType === "RecipeDeleted") {
  await likesRepo.deleteByRecipeId(event.payload.recipeId);
  await commentsRepo.deleteByRecipeId(event.payload.recipeId);
}
```

Рисунок 7 - Обработка удаления рецепта

Аналогично обрабатывается удаление пользователя (рисунок 8).

```
if (event.eventType === "UserDeleted") {
  await likesRepo.deleteByUserId(event.payload.userId);
  await commentsRepo.deleteByUserId(event.payload.userId);
}
```

Рисунок 8 - Обработка удаления пользователя

Таким образом, каскадные операции выполняются асинхронно, без прямых HTTP-вызовов.

Вывод

В ходе выполнения работы было реализовано асинхронное взаимодействие между микросервисами системы обмена рецептами с использованием брокера сообщений RabbitMQ.

В результате:

- снижена связность между микросервисами;
- устранены каскадные операции через HTTP;
- повышена отказоустойчивость и масштабируемость системы;