

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэкенд разработка

Отчет

Лабораторная работа 6

Выполнил:

Фролова Кристина

К3439

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

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

Ход работы

В качестве брокера я выбрала Apache Kafka. В моём проекте он будет выступать в роли передатчика сообщений об изменениях агрегата User, чтобы сервисы обновляли собственную копию данных таблицы known_users, тем самым уменьшая межсервисное синхронное взаимодействие, например, на операции верификации пользователя. На рисунке 1 отображены доменные события, на рисунке 2 - класс продюсера, который отправляет ивенты в топик. На рисунке 3 продемонстрирован класс консюмера, который слушает события в топике и на основе типа ивента принимает решение об операции в реплицирующей таблице.

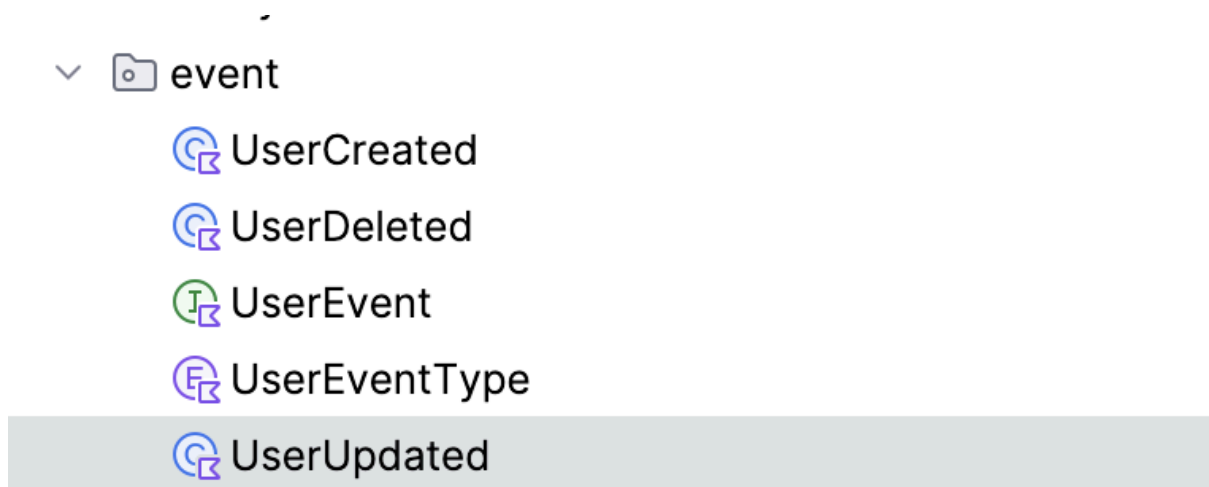


Рисунок 1 - Доменные события агрегата User

```

7  @Service
8  open class KafkaUserProducerImpl(
9      private val kafkaTemplate: KafkaTemplate<String, UserEvent>
10 ) : KafkaUserProducer {
11
12     companion object { 1 Usage
13         const val TOPIC = "user.events" 1 Usage
14     }
15
16     override fun send(event: UserEvent) {
17         kafkaTemplate.send(TOPIC, key = event.id, data = event)
18     }
19 }

```

Рисунок 2 - Класс продюсера

```

@Component
open class UserEventsConsumer(
    private val knownUsersService: KnownUsersService
) {

    @KafkaListener(
        topics = ["user.events"],
        groupId = "${spring.kafka.consumer.group-id}"
    )
    open fun onUserEvent(event: UserEvent) {
        when (event.type) {
            UserEventType.CREATED -> knownUsersService.create(KnownUser( id = UUID.fromString( name = event.id), event.updatedAt))
            UserEventType.UPDATED -> knownUsersService.update(KnownUser( id = UUID.fromString( name = event.id), event.updatedAt))
            UserEventType.DELETED -> knownUsersService.delete( id = UUID.fromString( name = event.id))
        }
    }
}

```

Рисунок 3 - Класс консьюмера

На рисунках 4 и 5 продемонстрирован пример работы очереди. При регистрации пользователя user-service отправляет сообщение в очередь, которая слушается chat-service. Получив ивент о создании пользователя, сервис создаёт во внутренней бд запись об этом пользователе. Таким

образом, если сервису понадобится верифицировать юзера, например, на существование. то ему не нужно будет обращаться к другому сервису.

The screenshot shows a REST client interface for the endpoint `POST /v1/auth/register`. The request body is a JSON object with the following structure:

```
{  "password": "string",  "mail": "string",  "firstName": "string",  "lastName": "string",  "phone": "string"}
```

The response is a 200 status code with the following JSON body:

```
{  "id": "f3e81be2-d7e1-44c4-bbb1-583161bd0120",  "mail": "string",  "firstName": "string",  "lastName": "string",  "phone": "string",  "createdAt": "2026-01-12T21:38:17.838331+03:00",  "updatedAt": "2026-01-12T21:38:17.838463+03:00"}
```

The interface includes a 'Curl' section with the following command:

```
curl -X 'POST' \  'http://localhost:8084/v1/auth/register' \  -H 'accept: */*' \  -H 'Content-Type: application/json' \  -d '{  "password": "string",  "mail": "string",  "firstName": "string",  "lastName": "string",  "phone": "string"  }'
```

The 'Request URL' is `http://localhost:8084/v1/auth/register`. The 'Server response' section shows the status code 200 and the response body.

Рисунок 4 - Реализованный запрос в user-service на регистрацию

The screenshot displays a PostgreSQL query editor interface. At the top, the connection string is `public.known_users/chat/postgres@chat`. Below the connection bar, there is a toolbar with various icons for file operations, query execution, and settings. The query editor shows the following SQL query:

```
1 SELECT * FROM public.known_users
2 ORDER BY id ASC
```

Below the query editor, there are tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table with the results of the query. The table has two columns: `id` (type: `[PK] uuid`) and `updated_at` (type: `timestamp with time zone`). The results show one row with the following values:

	id [PK] uuid	updated_at timestamp with time zone
1	f3e81be2-d7e1-44c4-bbb1-583161bd0120	2026-01-12 18:38:17.897833+00

Рисунок 5 - known_users в chat-service

Вывод

В ходе работы была подключена и настроена очередь сообщений между сервисами.