**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Лабораторная работа № 3
# “Микросервисы”

Выполнил:

Коротин А.М.

K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

**Задача**

Необходимо реализовать отдельный микросервис, выполняющий какую-либо содержательную функцию из всего арсенала функций вашего приложения.

**Ход работы**

В отдельный микросервис была вынесена логика аутентификации. Во время проектирования архитектуры рассматривались следующие варианты реализации работы с БД:

1) Database-per-service подход — каждый сервис оперирует своей собственной БД,
2) Shared-database подход — каждый сервис оперирует одной общей базой данных.

Выбор пал на использование 1ого подхода с использованием брокера сообщений Apache Kafka для асинхронной горизонтальной коммуникации в системе.

Архитектура сервиса аутентификации подразумевала генерацию доменных событий, одно из которых генерируются при регистрации нового пользователя. Обработчиком этого события сделаем функцию, отправляющую данные нового пользователя в специальный топик брокера сообщений (рисунок 1).

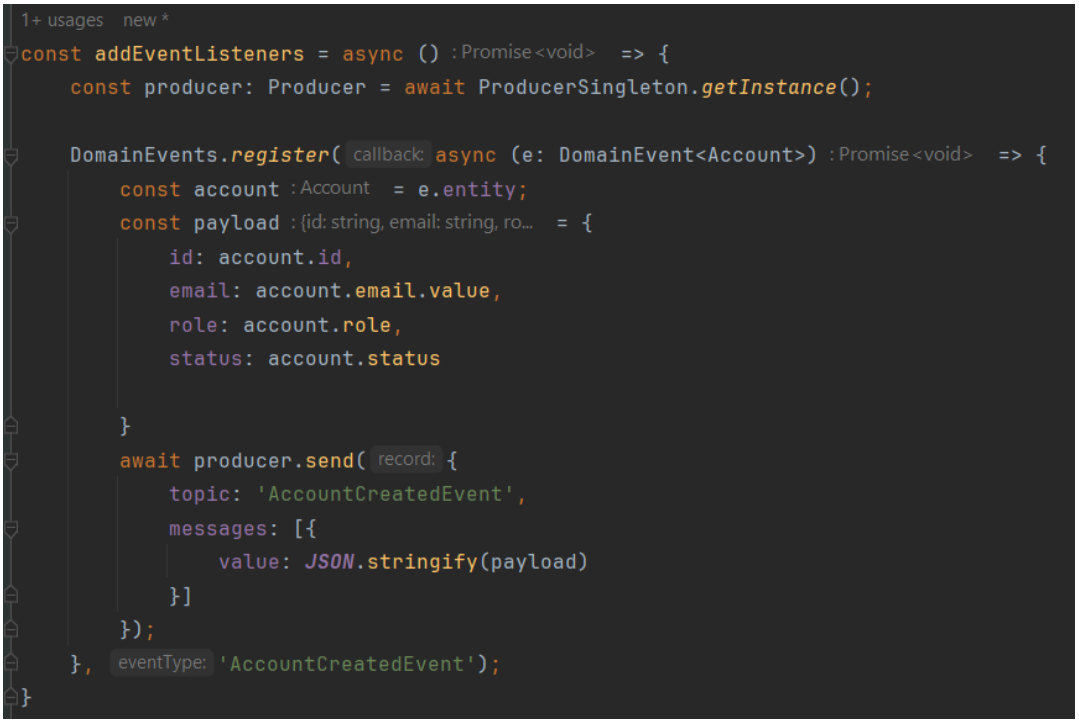


Рисунок 1 — Функция-обработчик доменного события создания аккаунта

В сервисе умных устройств же создадим соответствующий Kafka-Consumer, который будет реагировать на поступающие сообщения. На каждое сообщение о регистрации он будет создавать соответствующую запись в локальной БД для поддержания консистентности данных (рисунок 2).

```
1+ usages   new *
const userCreatedCallback = async (message: string) : Promise<void>  => {
    const properties = JSON.parse(message);
    const repository : Repository<AccountModel>  = sequelize.getRepository(AccountModel);

    await repository.create(properties);
}


1+ usages   new *
export const bootstrapKafka = async () : Promise<void>  => {
    const consumer : Consumer  = await ConsumerSingleton.getInstance();
    await consumer.subscribe( subscription: { topic: 'AccountCreatedEvent', fromBeginning: true })
    await consumer.run( config: {
        eachMessage: async ({message}: EachMessagePayload) : Promise<void>  => {
            const value : string  = message.value?.toString() as string;
            await userCreatedCallback(value);
        }
    })
}
```

Рисунок 2 — Функция-обработчик поступающих сообщений

На рисунке 3 приведена файловая структура проекта с использованием микросервисной архитектуры.

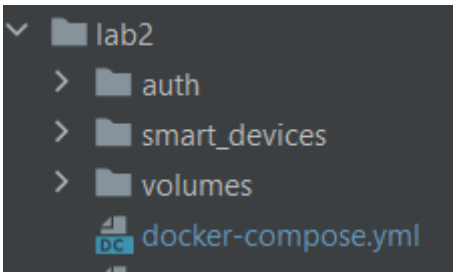


Рисунок 3 — Файловая структура

В микросервисе smart_devices была объявлена репликационная sequelize-модель аккаунта пользователя, предназначенная только для чтения (рисунок 4).

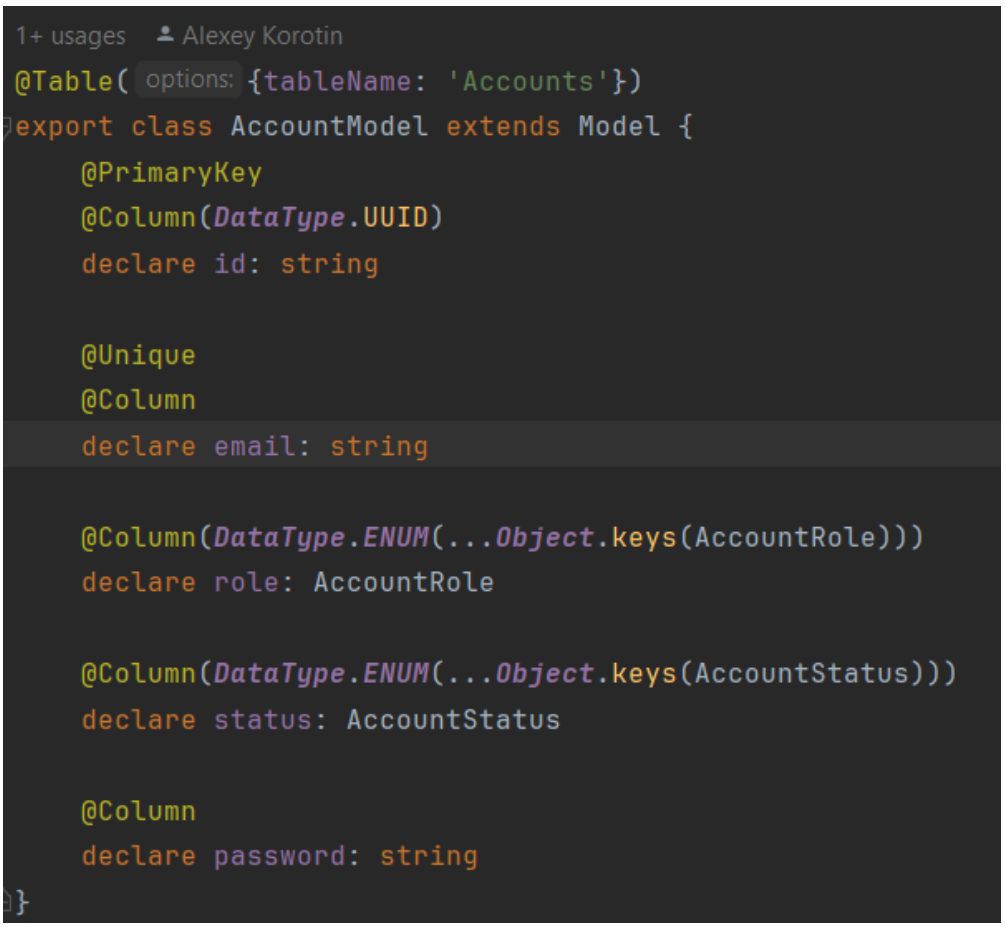


Рисунок 4 — Репликационная модель аккаунта

Именно эта модель будет использоваться для дальнейших отношений с другими моделями сервиса.

Помимо этого, в микросервисе аутентификации был реализован функционал refresh-токенов для удобства использования сервиса (рисунок 5).

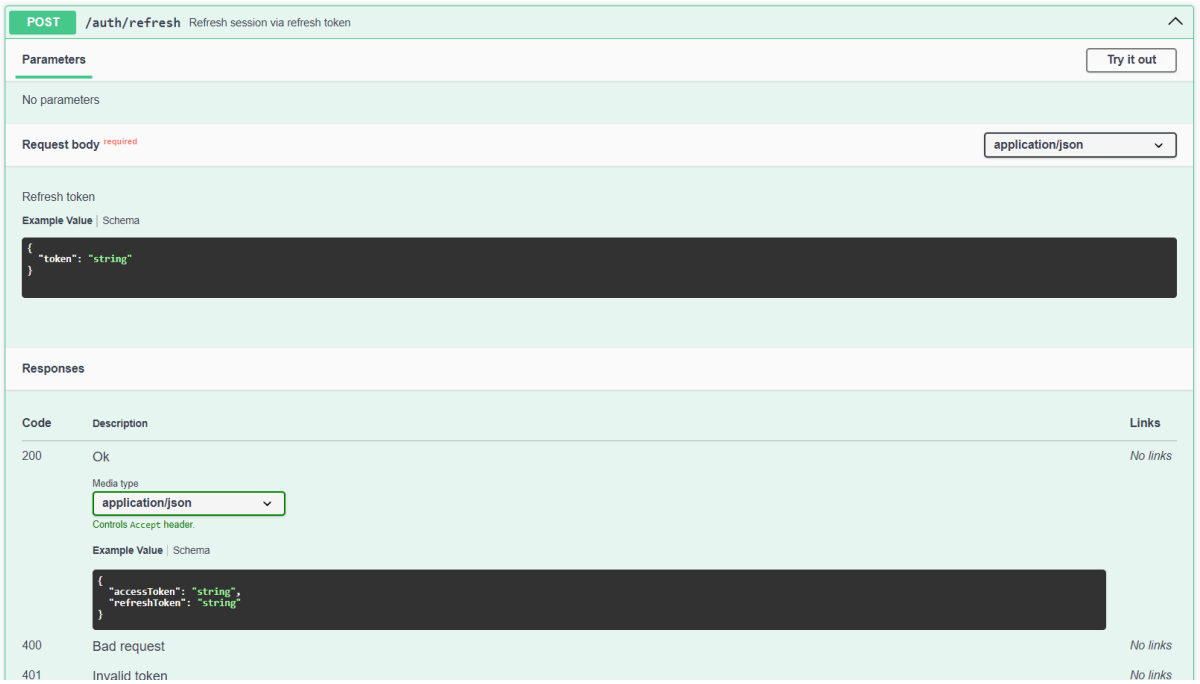


Рисунок 5 — Эндпоинт для возобновления сессии при помощи refresh-токена

## Вывод

В ходе выполнения лабораторной работы были изучены основы микросервисной архитектуры. Также было произведено разбиение ранее написанного приложения на микросервисы, взаимодействующие между собой