**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бек-энд разработка

# Отчет
# Лабораторная работа №2
# “Реализация RESTful API”

Выполнил:
Стукалов Артем
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

**Задание:**

В рамках данной лабораторной работы Вам предложено выбрать один из нескольких вариантов. Выбранный вариант остается единым на весь курс и будет использоваться в последующих лабораторных работах.

По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript.

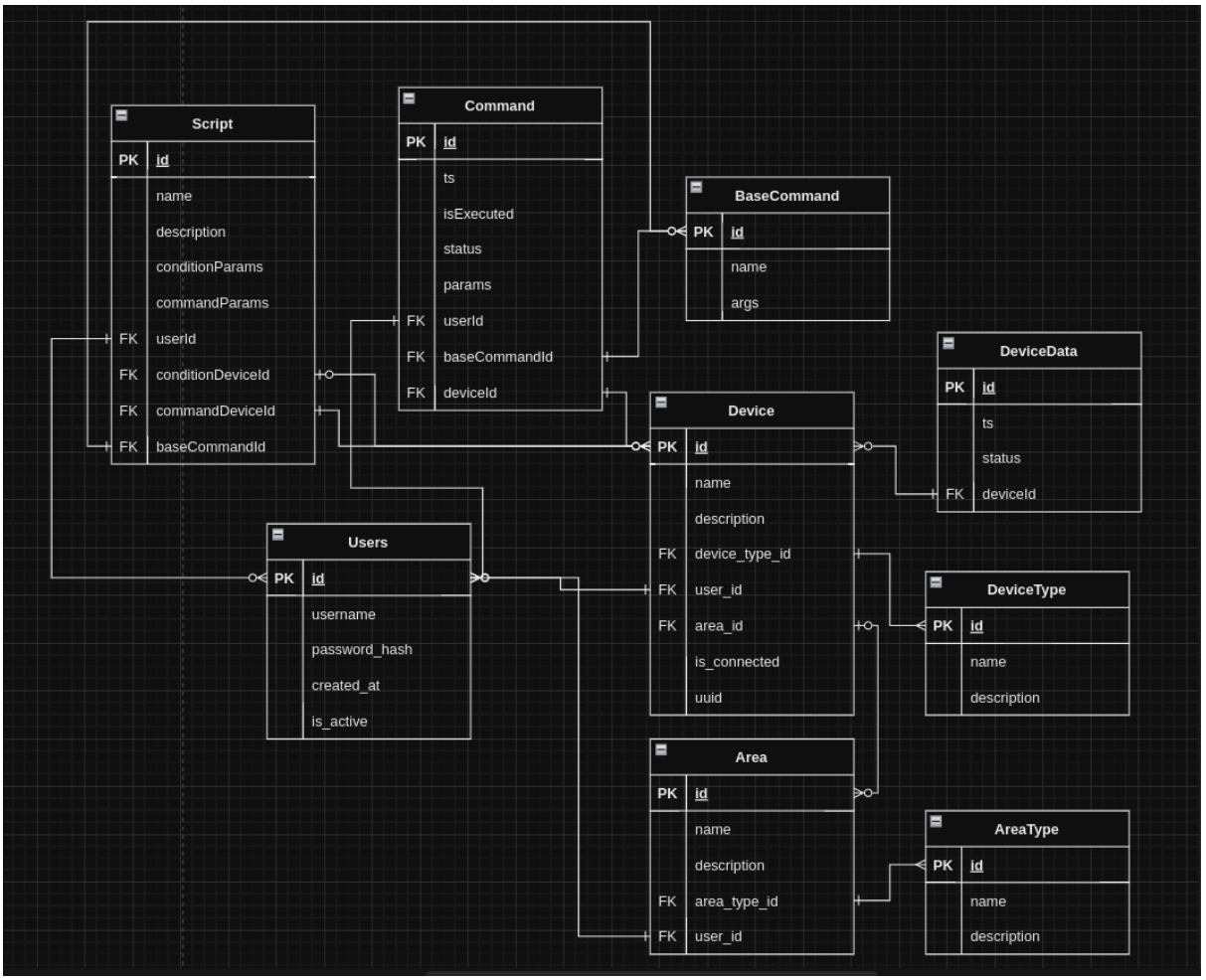
Выбранный сервис: Сервис для управления умным домом. Требуемый функционал: регистрация, авторизация, создание профиля, добавление нового устройства, настройка устройства, создание сценариев, запуск сценариев по триггерам/либо по времени (запуск сценариев можно сымитировать, главное, чтобы о запуске сохранилась информация в логах).

**Реализация:**

АПИ построено по схеме controller + service + routing. Все это также поддерживает возможность версионирования.

**Схема БД**

В качестве модели БД была сделана следующая схема:



Таблицы User, Device, Area - базовые таблицы, представляющие собой пользователя, устройства и комнаты, которые есть в системе.
DeviceData - талица логов изменения состояния устройства. Информация о состоянии хранится в формате json.

Command и Script - таблицы представляющие команды, которые можно передать устройствам и сценарии которые можно создать. Основная информация про параметры запуска команд и сценариев также хранится в формате json.

В json выносятся все данные, по которым нам не нужно осуществлять поиск в наших таблицах. Такой подход позволяет нам получить гибкую структуру, в рамках которой у нас есть некоторый формат выполнения команд, версию которых можно менять без изменения структуры БД. В данном случае остается лишь поддержать новые версии на конечных устройствах.

На данный момент есть поддержка 2 типов сценариев. В качестве демонстрации, сценарий имеет всего одно условие и одну команду, которую надо выполнить. Первый вариант это триггер по времени, который реализован через node-cron. Второй вариант это реагирование на изменение каких-то данных от других устройств.

**Сервисы**

В рамках приложения можно выделить 6 основных сервисов:

- scripts
- users
- areas
- commands
- deviceData
- devices

Каждый из них отвечает за выполнение каких-то основных функций по работе с определенной областью задач. Сервисы изолированы и не взаимодействуют друг с другом. Также все методы сервисов никогда не выкидывают ошибки, но могут их возвращать. Обработка этих ошибок отдается на уровень контроллеров.

**Контроллеры**

Соответственно контроллеров также 6. Контроллер является местом выполнения группы более сложных задач чем в сервисах. По сути они занимаются реализацией бизнес логики. Именно тут можно использовать методы разных сервисов и объединять их. Также каждый контроллер уже может выкидывать ошибки, так как контроллеры предполагается напрямую связывать с роутингом, что автоматически добавляет слой обработки ошибок.

**Роутинг**

Небольшая часть проекта, которая занимается неймингом энд поинтов и валидацией данных прежде чем они попадут в контроллер.

**Валидация данных**

Так как проект реализован на базе темплейта для ЛР1, в качестве валидатора данных используется zod в связке с Prisma, что позволяет автоматически генерировать zod объекты на основе нашей схемы БД. Такой подход упрощает работу с валидацией, так как все, что нужно это комбинировать автоматически сгенерированные типы, что снижает возможность ошибки.

**Обработка сценариев**

Как было сказано раньше, поддерживаем 2 типа сценариев. Модуль сценариев представлен в виде синглтон класса.

Сценарии по таймеру:

При старте приложения выгружаем сценарий из бд и запускаем кроны. При удалении или обновлении сценариев, дергаем методы класса и обновляем сценарии.

Сценарии по триггеру:

При запуске приложения для всех сценариев создаем мапу с ключами deviceId  и значениями в виде параметров которые нужно мониторить и при срабатывании условий выполнять какие-то действия. При получении информации о DeviceData, вызываем метод класса, который обработает этот ивент. При изменении сценариев также не забываем обновлять или удалять ненужные.