САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет
Лабораторная работа №2
“Тестирование, разработка и документирование RESTful API”

Выполнил:

Екушев Владислав

К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

1. По выбранному варианту необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Ход работы

Выбранный вариант: реализация приложения для управления умным домом.

Используем написанный шаблон на NestJS и Prisma (https://github.com/jarvis394/nestjs-prisma-boilerplate).

В ходе выполнения работы было принято решение поменять базу данных с Prisma на MongoDB для того, чтобы было удобнее хранить состояние девайсов.

Создаем схему девайса в MongoDB:

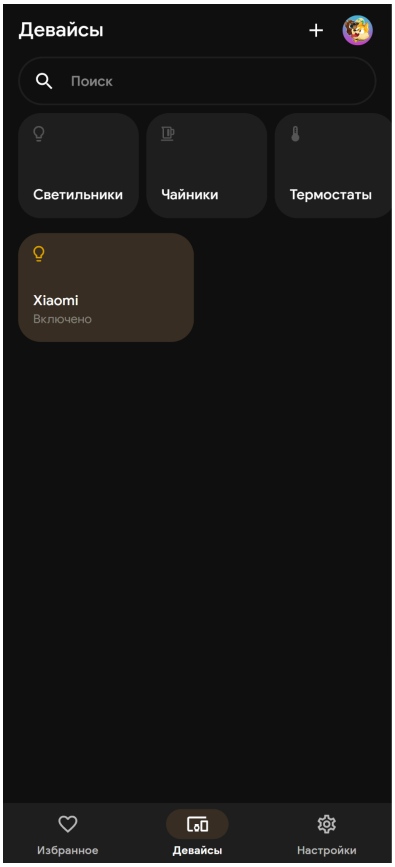
```ts
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose'
import mongoose, { HydratedDocument } from 'mongoose'
import {
  DeviceCapabilityType,
  DeviceCapabilityByType,
  DeviceType,
} from '@smart-home/shared'

export type DeviceDocument = HydratedDocument<Device>

@Schema()
export class Device {
  @Prop({ required: true })
  name: string

  @Prop({
    required: true,
    enum: Object.values(DeviceType),
  })
  type: `${DeviceType}`

  @Prop({ required: true })
  state: 0 | 1

  @Prop({ required: false, default: false })
  favorite: boolean

  @Prop({ type: mongoose.Schema.Types.ObjectId })
  userId: string

  @Prop({ required: true, type: mongoose.Schema.Types.Mixed })
  capabilities: {
    [Type in DeviceCapabilityType]?: DeviceCapabilityByType<Type>
  }
}

export const DeviceSchema = SchemaFactory.createForClass(Device)
```

Делаем контроллеры для девайсов:

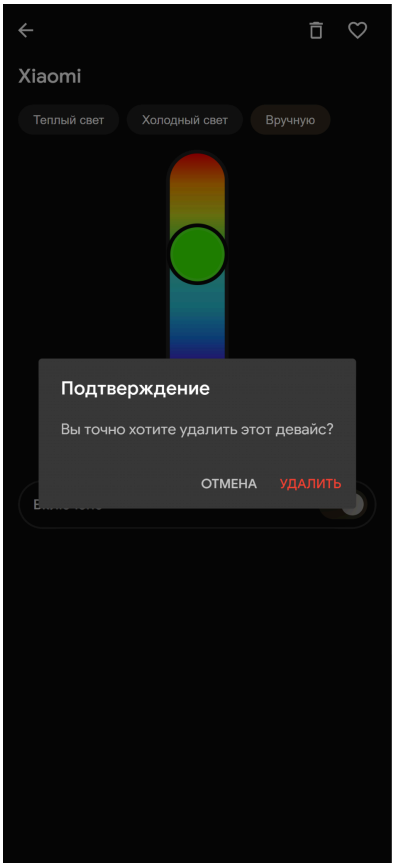
```ts
import {
  Body,
  Controller,
  Get,
  Param,
  Post,
  Request,
  UseGuards,
} from '@nestjs/common'
import { DevicesService } from './devices.service'
import { JwtAuthGuard } from '../auth/strategies/jwt.strategy'
import { RequestWithUser } from '../auth/auth.controller'
import {
  AddDeviceReq,
  AddDeviceRes,
  DeviceDeleteRes,
  DevicesGetRes,
  FavoriteDeviceRes,
  ToggleDeviceOnOffRes,
} from '@smart-home/shared'
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger'

@ApiTags('devices')
@Controller('devices')
export class DevicesController {
  constructor(private readonly devicesService: DevicesService) {}

  @UseGuards(JwtAuthGuard)
  @Get()
  @ApiBearerAuth()
  async getDevices(@Request() req: RequestWithUser): Promise<DevicesGetRes> {
    const devices = await this.devicesService.getDevices(req.user.userId)
    return { devices }
  }

  @UseGuards(JwtAuthGuard)
  @Get('favorites')
  @ApiBearerAuth()
  async getFavoriteDevices(
    @Request() req: RequestWithUser
  ): Promise<DevicesGetRes> {
    const devices = await this.devicesService.getFavoriteDevices(
      req.user.userId
    )
    return { devices }
  }

  @UseGuards(JwtAuthGuard)
  @Get(':id/favorite')
  @ApiBearerAuth()
  async toggleFavorite(
    @Request() req: RequestWithUser,
    @Param('id') id: string
  ): Promise<FavoriteDeviceRes> {
    const state = await this.devicesService.toggleFavorite(req.user.userId, id)
    return { state }
  }

  @UseGuards(JwtAuthGuard)
  @Get(':id/onOff/toggle')
  @ApiBearerAuth()
  async toggleOnOff(
    @Request() req: RequestWithUser,
    @Param('id') id: string
  ): Promise<ToggleDeviceOnOffRes> {
    const state = await this.devicesService.toggleOnOff(req.user.userId, id)
    return { state }
  }

  @UseGuards(JwtAuthGuard)
  @Get(':id/delete')
  @ApiBearerAuth()
  async delete(
    @Request() req: RequestWithUser,
    @Param('id') id: string
  ): Promise<DeviceDeleteRes> {
    const state = await this.devicesService.delete(req.user.userId, id)
    return { ok: state }
  }

  @UseGuards(JwtAuthGuard)
  @Post('add')
  @ApiBearerAuth()
  async addDevice(
    @Request() req: RequestWithUser,
    @Body() newDevice: AddDeviceReq
  ): Promise<AddDeviceRes> {
    const device = await this.devicesService.addDevice(
      req.user.userId,
      newDevice
    )
    return { device }
  }
}
```

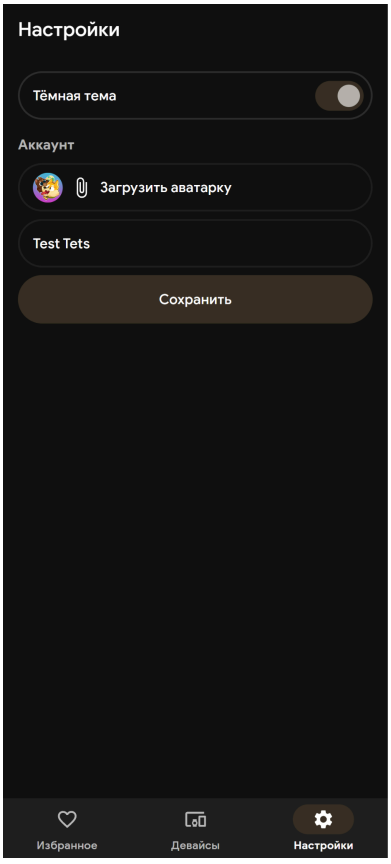
Сервис девайсов:

```ts
import { ForbiddenException, Injectable } from '@nestjs/common'
import { AddDeviceReq, DeviceType, Device as IDevice } from '@smart-home/shared'
import { Device, DeviceDocument } from './schemas/device.schema'
import { Model } from 'mongoose'
import { InjectModel } from '@nestjs/mongoose'
import { UserService } from '../user/user.service'

@Injectable()
export class DevicesService {
  constructor(
    @InjectModel(Device.name) private readonly deviceModel: Model<Device>,
    private userService: UserService
  ) {
    this.deviceModel = deviceModel
  }

  serializeDevice(deviceDocument: DeviceDocument): IDevice {
    return {
      id: deviceDocument.id,
      userId: deviceDocument.userId,
      // eslint-disable-next-line @typescript-eslint/ban-ts-comment
      //@ts-expect-error
      capabilities: deviceDocument.capabilities,
      favorite: deviceDocument.favorite,
      name: deviceDocument.name,
      state: deviceDocument.state,
      type: deviceDocument.type as DeviceType,
    }
  }

  async getDevices(userId: string): Promise<IDevice[]> {
    const devices = await this.deviceModel.find({ userId })
    return devices.map((device) => this.serializeDevice(device))
  }

  async getFavoriteDevices(userId: string): Promise<IDevice[]> {
    const devices = await this.deviceModel.find({ userId, favorite: true })
    return devices.map((device) => this.serializeDevice(device))
  }

  async delete(userId: string, deviceId: string): Promise<boolean> {
    const result = await this.deviceModel.deleteOne({ _id: deviceId, userId })

    return result.acknowledged
  }

  async toggleFavorite(userId: string, deviceId: string): Promise<boolean> {
    const result = await this.deviceModel.findOneAndUpdate(
      { _id: deviceId, userId },
      [{ $set: { favorite: { $eq: [false, '$favorite'] } } }]
    )

    return !result?.favorite || false
  }

  async toggleOnOff(userId: string, deviceId: string): Promise<boolean> {
    const result = await this.deviceModel.findOne({ _id: deviceId, userId })
    if (!result?.capabilities.on_off) {
      throw new ForbiddenException('Unsupported device feature')
    }

    result.capabilities.on_off.state.value =
      !result.capabilities.on_off.state.value

    await this.deviceModel.findOneAndUpdate(
      { _id: deviceId, userId },
      {
        capabilities: result.capabilities,
      },
      { new: true }
    )

    return result.capabilities.on_off.state.value
  }

  async addDevice(userId: string, data: AddDeviceReq): Promise<IDevice> {
    const newDevice = new this.deviceModel()
    newDevice.userId = userId
    newDevice.capabilities = data.capabilities
    newDevice.name = data.name
    newDevice.state = data.state
    newDevice.type = data.type

    const result = await this.deviceModel.create(newDevice)

    await this.userService.addDevice(result)

    return this.serializeDevice(result)
  }
}
```


Девайсы
Поиск
Светильники
Чайники
Термостаты
Xiaomi
Включено
Избранное
Девайсы
Настройки


Xiaomi
Теплый свет
Холодный свет
Вручную
Включено


Xiaomi
Теплый свет
Холодный свет
Вручную
Подтверждение
Вы точно хотите удалить этот девайс?
ОТМЕНА
УДАЛИТЬ


Настройки
Тёмная тема
Аккаунт
Загрузить аватарку
Test Tets
Сохранить
Избранное
Девайсы
Настройки

Вывод

В ходе лабораторной работы было реализовано приложение для управления умным домом. Поддержаны функции регистрации/авторизации, смены имени и аватарки, добавления/удаления устройств, добавления устройств в избранное и изменения состояния устройств.

Результат представлен в репозитории – https://github.com/jarvis394/smart-home