САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

ДЗ3 - документирование API средствами swagger и Postman

Выполнил:

Сергеев Виктор

K3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- реализовать автодокументирование средствами swagger;
- реализовать документацию API средствами Postman.

Ход работы

Для документирования API контроллеры моделей были переписаны на routing-controllers-openapi. Было принято решение остановиться на этом варианте, поскольку эта библиотека позволяет сделать автодокументирование с помощью встроенных методах, а также работает на Windows (в отличии от tsoa).

Для каждого контроллера каждой модели были написаны dto для создания, обновления и для ответа, а также написаны функции для перемещения данных в модель из dto и наоборот. Данные dto также будут являться схемами данных в OpenAPI конфигурации и swagger. Ниже представлен dto для регистрации пользователя.

```
export class RegisterUserDto {
    @IsDefined({message: "username is required"})
    @IsString({message: "username is invalid string"})
    @Matches(usernameRegex, {message: "username contains invalid symbols"})
    @Length(8, 50, {message: "username length must be between 8 and 50 characters"})
    public readonly username: string;

    @IsDefined({message: "password is required"})
    @IsString({message: "password is invalid string"})
    @Length(8, 100, {message: "password length must be between 8 and 100 characters"})
    public readonly password: string;

    @IsDefined({message: "password_confirm is required"})
    @IsString({message: "password_confirm is invalid string"})
    @Length(8, 100, {message: "password_confirm length must be between 8 and 100 characters"})
    public readonly password_confirm: string;
}
```

Рисунок 1 - класс RegisterUserDto

Для валидации данных используется библиотека class-validator.

Далее требовалось описать контроллеры методами routing-controllers. Ниже представлен пример описания api-endpoint-а регистрации пользователя в AuthController.

```
    @OpenAPI({summary: "register a new user"})
    @ResponseSchema(UserResponseDto, {
        statusCode: 201,
        description: "successful registration"
    })
    @ResponseSchema(ErrorDto, {
        statusCode: 400,
        description: "registration failure"
    })
    @Post("/register")
    async register (
        @Body({ type: RegisterUserDto }) body: RegisterUserDto,
        @Res() res: Response
    ): Promise<void> {
        if (await this.service.getEntityByUsername(body.username)) {
            res.status(400).json({detail: "username already exists"});
            return;
        }

        const hash = hashPassword(body.password)
        const dto = {
            "username": body.username,
            "password": hash
        }
        try {
            const entity = await this.service.createEntity(dto);
            const responseDto = toUserResponseDto(entity);
            res.status(201).json(responseDto);
        } catch (error) {
            res.status(400).json({error: error.message});
        }
    }
```

Рисунок 2 - метод AuthController.register

Аналогичным были описаны все методы всех контроллеров в приложении. Наконец, swagger подключается в express приложение и формируется автодокументация по адресу /docs. Ниже представлена сама документация swagger.

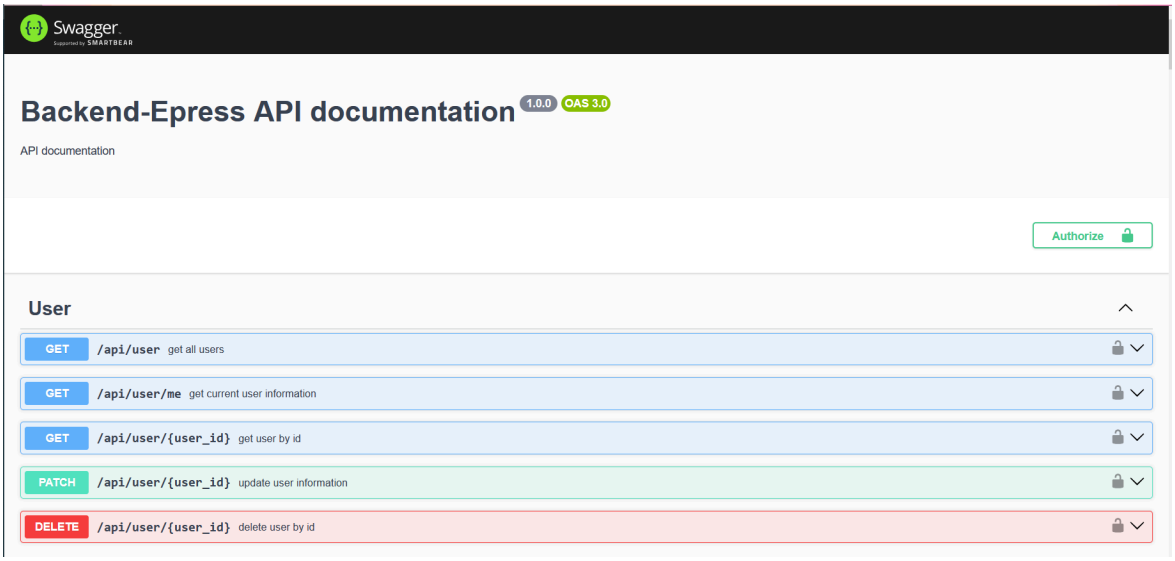

Рисунок 3 - страницы документации swagger

Вторым заданием было сделать документацию средствами postman. Для этого были в одной коллекции были собраны все методы и были собраны примеры ответов для этих методов. Ниже представлена коллекция API и примеры ответов для одного из контроллеров.

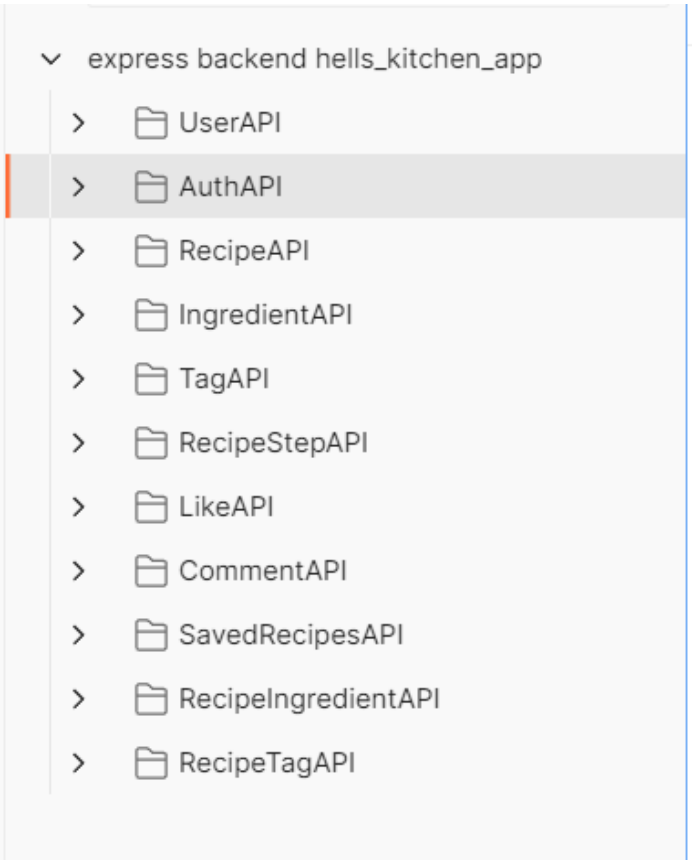

Рисунок 4 - коллекция методов

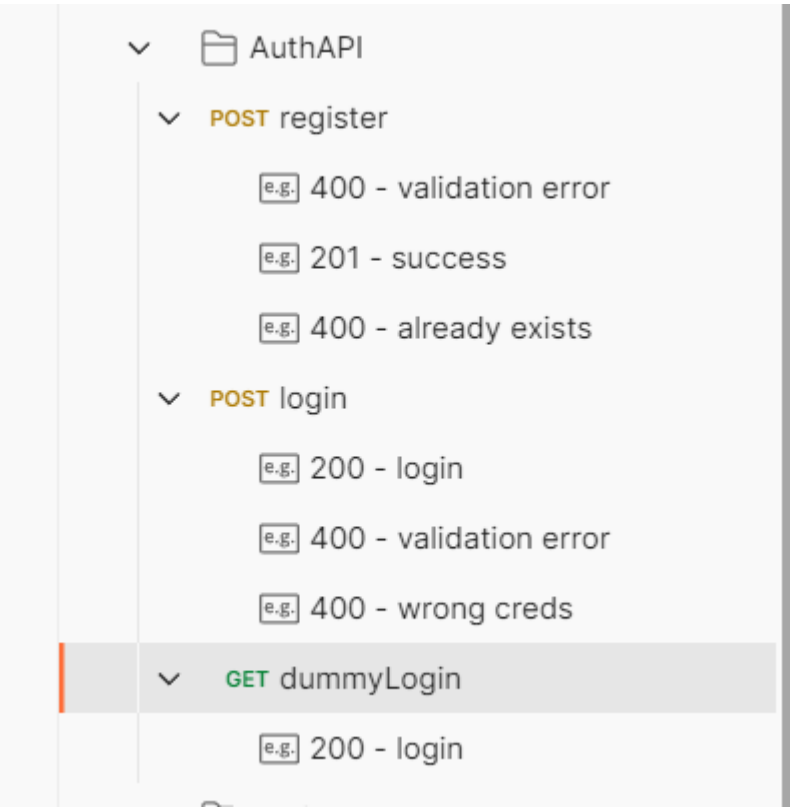

Рисунок 5 - документация в Postman для контроллера авторизации

Вывод

В процессе работы была реализована автодокументация API в формате OpenAPI средствами routing-controllers-openapi, а также реализована документация API в Postman.