САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №3

Выполнил:

Акулов Даниил

К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

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

Ход работы

Swagger

В проекте используется генерация спецификации OpenAPI из метаданных routing-controllers и class-validator.

index.ts file:

```
    class App {
        public port: number;
        public controllersPath: string;

        private app: express.Application;

        constructor(port = SETTINGS.API_PORT,
                                                controllersPath   =
SETTINGS.APP_CONTROLLERS_PATH) {
            this.port = port;
            this.controllersPath = controllersPath;
            this.app = this.configureApp();
        }

        private configureApp(): express.Application {
            let app = express();

            app.use(cors());
            app.use(express.json());

            const options = {
                routePrefix: SETTINGS.APP_API_PREFIX,
                controllers: [__dirname + this.controllersPath],
                validation: true,
                classTransformer: true,
                defaultErrorHandler: false,
            };
```

```
            app = useExpressServer(app, options);
            app = useSwagger(app, options);

            app.use(errorHandler);

            return app;
        }

        public start(): void {
            dataSource
                .initialize()
                .then(() => {
                                console.log('Data Source has been
initialized!');
                    this.app.listen(this.port, () => {
                        console.log(
                                                `Running server on
http://localhost:${SETTINGS.API_PORT}`,
                        );
                    });
                })
                .catch((err) => {
                            console.error('Error during Data Source
initialization:', err);
                });
        }
    }
```

swagger.ts file:

```
    export function useSwagger(
        app: Express,
        options: RoutingControllersOptions,
    ): Express {
        try {
            const schemas = validationMetadatasToSchemas({
                                    classTransformerMetadataStorage:
defaultMetadataStorage,
                refPointerPrefix: '#/definitions/',
            });
```

```
            const storage = getMetadataArgsStorage();

                const spec = routingControllersToSpec(storage,
options, {
                components: {
                    schemas,
                    securitySchemes: {
                        bearerAuth: {
                            type: 'http',
                            scheme: 'bearer',
                            bearerFormat: 'JWT',
                        },
                    },
                },
                info: {
                    title: 'Sport service API documentation',
                    version: '1.0.0',
                },
            });

                                app.use('/docs',    swaggerUi.serve,
swaggerUi.setup(spec));

            return app;
        } catch (error) {
            console.error('Swagger Error:', error);
            return app;
        }
    }
```

Декораторы в контроллерах влияют на описание маршрутов:

• @Get("/") — метод HTTP и путь

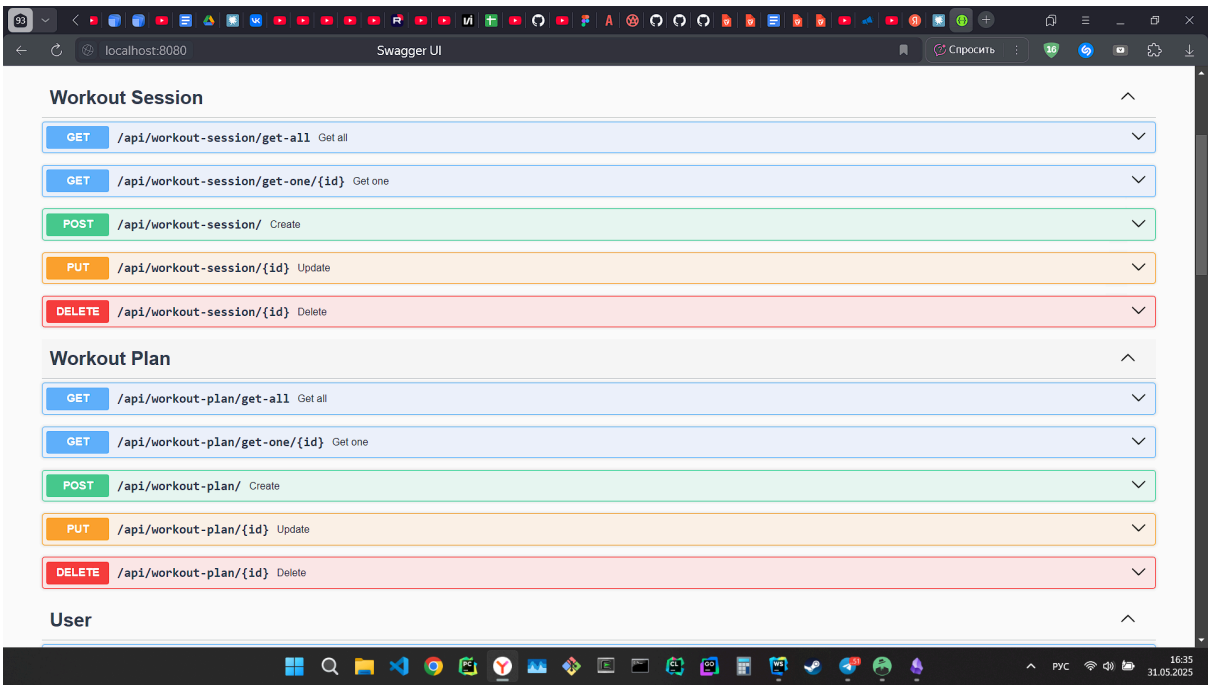
• @OpenAPI(...) — описание для Swagger (summary, description, теги, параметры)

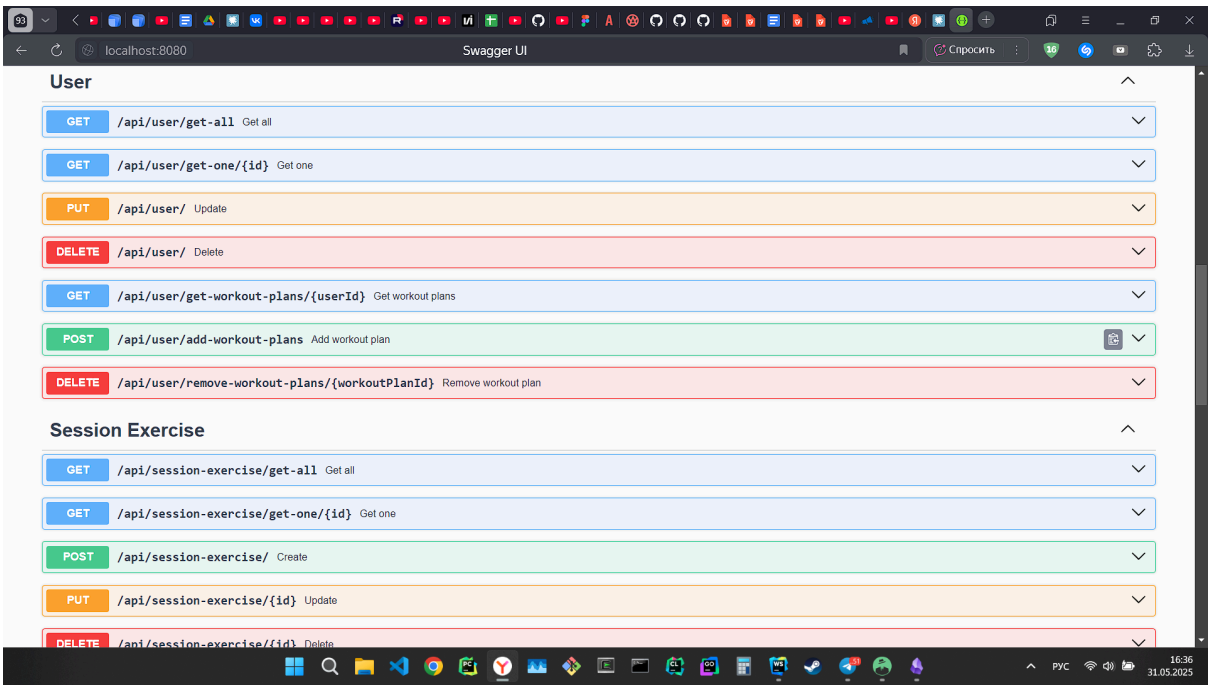
• @ResponseSchema(...) — тип возвращаемых данных (генерирует 200 OK и структуру JSON)

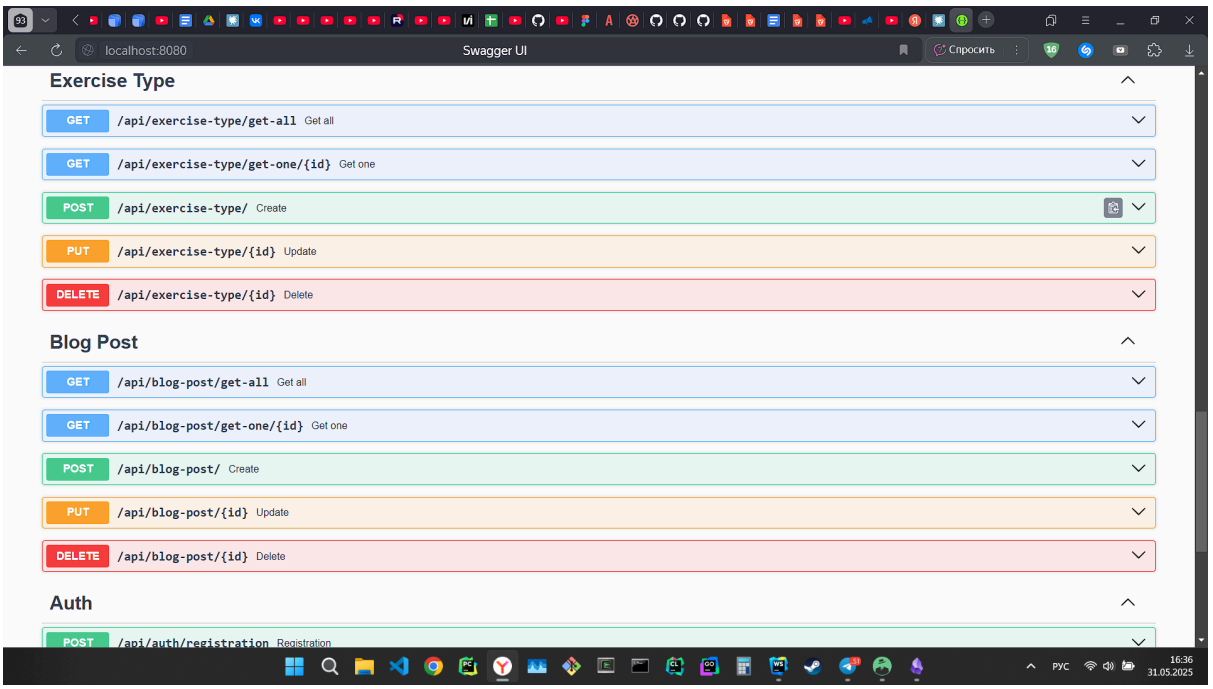
Пример контроллера:

```
@JsonController('/auth')
export class AuthController {
    @OpenAPI({})
    @Post('/registration')
    async registration(
        @Body() body: RegistrationDto,
        @Res() res: Response,
    ) {
        const {email, password, name} = body
            const candidate = await userRepo.findOne({where: {email}})
        if (candidate) {
                                                    throw ApiError.badRequest(errorMessages.emailIsTaken)
        }
        const hashPassword = await bcrypt.hash(password, 6)
          const userEntity = userRepo.create({email, password: hashPassword, name});
        const user = await userRepo.save(userEntity);
        const token = generateAccessToken(user.id, email)
        return res.json({token, user})
    }
```

Сгенерированная документация:

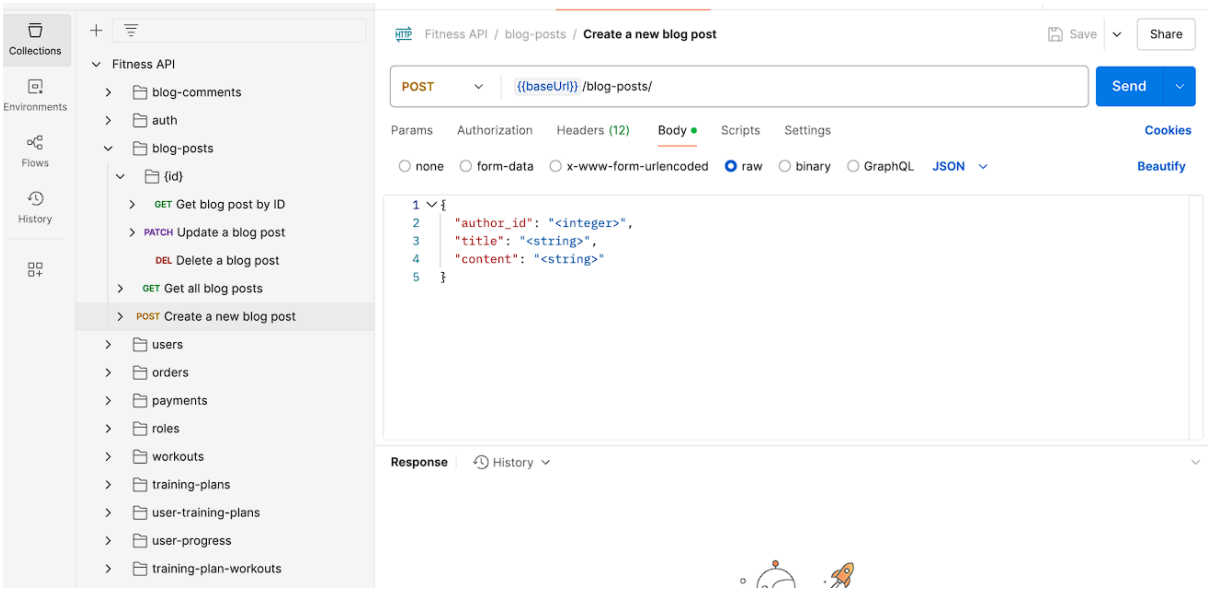






Postman

Дополнительно была вручную написана коллекция запросов в Postman, позволяющая протестировать API.



Вывод

В ходе практической работы была реализована документация REST API двумя способами: автоматически с помощью OpenAPI и вручную с помощью Postman. Это обеспечило прозрачность и удобство использования.