

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашняя работа 3

Выполнил:

Беломытцев Андрей

К3339

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

## Задача

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

## Ход работы

Для реализации документации было решено использовать tsoa.

### Файл tsoa.json

```
{
  "entryFile": "src/app.ts",
  "noImplicitAdditionalProperties": "throw-on-extras",
  "controllerPathGlobs": ["src/controllers/*.ts"],
  "spec": {
    "outputDirectory": "src",
    "specVersion": 3,
    "securityDefinitions": {
      "jwt": {
        "type": "apiKey",
        "name": "Authorization",
        "in": "header"
      }
    }
  },
  "routes": {
    "routesDir": "src",
    "authenticationModule": "./src/authentication.ts"
  }
}
```

Для генерации необходимого кода используется команда

```
npx tsoa spec-and-routes
```

Для подключения роутов tsoa в app.ts добавил

```
import swaggerUi from 'swagger-ui-express'
import swaggerDocument from './swagger.json'
import { RegisterRoutes } from './routes'

RegisterRoutes(app)
app.use('/docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument))
```

Для работы с документацией изменил аутентификацию (authentication.ts)

```
import { Request } from "express";
import jwt from "jsonwebtoken";
import config from './config'
import { AppDataSource } from './AppDataSource'
import { User } from './models/User'

const repository = AppDataSource.getRepository(User)
```

```

export function expressAuthentication(req: Request, securityName: string,
scopes?: string[]): Promise<any> {
  const token = req.headers.authorization?.split(' ')[1]
  return new Promise((resolve, reject) => {
    if (!token) {
      return reject(new Error("No token provided"));
    }
    jwt.verify(token, config.JWT_SECRET_KEY, async function (err: any,
decoded: any) {
      if (err) {
        return reject(err);
      } else {
        if (scopes){
          const user = await repository.findOne({ where: { username:
decoded.username }, relations: ['role'] })
          for (let scope of scopes) {
            if (!user || !user.role || ![user.role.name].includes(scope)) {
              reject(new Error("You do not have permission"));
            }
          }
        }
        return resolve(decoded);
      }
    })
  })
}

```

Написал контроллеры подходящие для tsoa. Рассмотрим на примере каналов. В них кроме CRUD реализовано подключение к API Ютуба для получения данных о канале и видео с канала при добавлении. Файл `controllers/Channel.ts`

```

import { AppDataSource } from "../AppDataSource"
import { Channel } from '../models/Channel'
import config from '../config';
import { Video } from '../models/Video'
import { User } from '../models/User'
import { Category } from '../models/Category'
import { Theme } from '../models/Theme'
import { Controller, Get, Post, Put, Delete, Route, Tags, Body, Path,
Security, Request } from 'tsoa'
import { ChannelCreateDto, ChannelDto } from '../dto/Channel';

const repository = AppDataSource.getRepository(Channel)

```

### Функция для получения видео с канала

```

const getVideos = async (ytid: string, maxResults: number = 50) => {
  const uploads = 'UULF' + ytid.slice(2)
  const videos: any = await (await
fetch(`https://www.googleapis.com/youtube/v3/playlistItems?part=snippet%2CcontentDetails&maxResults=${maxResults}&playlistId=${uploads}&key=${config.YT_API_KEY}`)).json()
  var videosList: Video[] = []
  for(var m of videos['items']){
    var m = m['snippet']
    videosList.push({
      'id': m['resourceId']['videoId'],

```

```

        'channel': m['channelId'],
        'title': m['title'],
        'publishedAt': m['publishedAt'],
        'thumbnail': m['thumbnails']['maxres' in m['thumbnails'] ? 'maxres' :
'medium']['url'],
        'description': m['description'],
    } as Video)
}
return videosList
}

```

## Функция для получения данных о канале

```

const getChannel = async (ytid: string, lang: string, category: string,
theme: string, username: string): Promise<Channel> => {
    const response = await
fetch(`https://youtube.googleapis.com/youtube/v3/channels?part=statistics,sn
ippet&id=${ytid}&key=${config.YT_API_KEY}`)
    if (!response.ok) throw new Error('Network response was not ok')
    const data: any = await response.json()
    var stat = data['items'][0]['statistics']
    var icon = data['items'][0]['snippet']['thumbnails']
    // var ytid = data['items'][0]['id']

    var videosList = await getVideos(ytid)

    var channel = {
        id: ytid,
        url: 'https://www.youtube.com/channel/' + ytid,
        title: data['items'][0]['snippet']['title'],
        views: parseInt(stat["viewCount"]),
        subs: parseInt(stat["subscriberCount"]),
        videos: parseInt(stat["videoCount"]),
        lang: lang,
        category: await AppDataSource.getRepository(Category).findOneBy({ name:
category }),
        theme: await AppDataSource.getRepository(Theme).findOneBy({ name: theme
}),
        iconDefault: icon['default']['url'],
        iconMedium: icon['medium']['url'],
        iconHigh: icon['high']['url'],
        description: data['items'][0]['snippet']['description'],
        isApproved: false,
        user: await AppDataSource.getRepository(User).findOneBy({ username:
username }),
        videosList: videosList,
    } as Channel

    return channel
}

```

## Класс для работы API и документации

```

@Tags('Channel')
@Route('channel')
export class ChannelController extends Controller {
    @Get()
    public async get(): Promise<ChannelDto[]> {
        return (await repository.find({ relations: ['category', 'theme']
})).map(({ user, videosList, reviews, ...rest }) => rest)
    }
}

```

```

    }

    /**
     * @example id "UCHnyfMqiRRGlu-2MsSQLbXA"
     */
    @Get('{id}')
    public async getOne(@Path() id: string): Promise<ChannelDto | null> {
        return await repository.findOne({ where: { id }, relations: ['category',
'theme'] })
    }

```

## Для добавления каналов требуется авторизоваться

```

@Post()
@Security('jwt')
public async create(@Body() body: ChannelCreateDto, @Request() req: any):
Promise<ChannelDto> {
    var channel = await getChannel(
        body.id,
        body.lang,
        body.theme,
        body.category,
        req.user.username
    )
    return await repository.save(channel)
}

```

## Редактировать и удалять информацию могут только администраторы

```

@Put('{id}')
@Security('jwt', ['admin'])
public async update(@Path() id: string, @Body() body:
Partial<ChannelCreateDto>): Promise<ChannelDto> {
    const x = await repository.findOneBy({ id })
    if (!x) {
        this.setStatus(404)
        throw new Error('Not found')
    }
    const updated: any = {...body}
    if (body.category) {
        updated.category = await
AppDataSource.getRepository(Category).findOneBy({ name: body.category })
    }
    if (body.theme) {
        updated.theme = await AppDataSource.getRepository(Theme).findOneBy({
name: body.theme })
    }
    repository.merge(x, updated)
    return await repository.save(x)
}

@Delete('{id}')
@Security('jwt', ['admin'])
public async remove(@Path() id: string) {
    const r = await repository.delete(id)
    if (r.affected === 0) {
        this.setStatus(404)
        throw new Error('Not found')
    }
    return r
}

```

```
    }  
  }
```

## DTO для каналов в файле dto/Channel.ts

```
import { CategoryDto } from '../Category'  
import { ThemeDto } from '../Theme'  
  
export interface ChannelCreateDto {  
  id: string,  
  lang: 'en' | 'ru',  
  category: string,  
  theme: string,  
}  
  
export interface ChannelDto {  
  id: string  
  url: string  
  title: string  
  views: number  
  subs: number  
  videos: number  
  lang: string  
  iconDefault: string  
  iconMedium: string  
  iconHigh: string  
  description: string  
  isApproved: boolean  
  timeCreate: Date  
  timeUpdate: Date  
  category: CategoryDto  
  theme: ThemeDto  
}
```

Остальные (User, Role, Video, Category, Theme, Review) реализованы аналогичным образом.

Открыть документацию можно по адресу /docs

Для создания документации с помощью Postman использовал расширение для VS Code.

## Вывод

В результате было реализовано автодокументирование средствами swagger при помощи tsoa, в том числе написаны DTO. Так же была реализована документация API средствами Postman при помощи расширения для VS Code.