САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:
Кривцов Павел
Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург
2023 г.

Задача
Необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate) для реализации платформы по составлению списка просмотренных фильмов и их оценок.

Ход работы
Оставим сущности, связанные с пользователем, с предыдущей лабораторной работы.

Добавим модели:
Режиссер:

```
@Entity()
export class Director {
    @PrimaryGeneratedColumn()
    id: number

    @Column( options: {unique: true})
    name: string

    @OneToMany( typeFunctionOrTarget: type => Movie, inverseSide: movie => movie.director)
    movies: Movie[];
}
```

Жанр:

```
@Entity()
export class Genre {
    @PrimaryGeneratedColumn()
    id: number

    @Column( options: {unique: true})
    name: string

    @OneToMany( typeFunctionOrTarget: type => Movie, inverseSide: movie => movie.genre)
    movies: Movie[];
}
```

Фильм:

```
@Entity()
@Unique( name: 'title_year_director', fields: ["title", "year", "director"])
export class Movie {
    @PrimaryGeneratedColumn()
    id: number

    @Column()
    title: string

    @Column( options: 'int')
    year: number

    @ManyToOne( typeFunctionOrTarget: type => Genre, inverseSide: genre => genre.movies)
    genre: Genre;

    @ManyToOne( typeFunctionOrTarget: type => Director, inverseSide: director => director.movies)
    director: Director;

    @OneToMany( typeFunctionOrTarget: () => Watchlist, inverseSide: watchlist => watchlist.movie)
    watchlist: Watchlist[];

    @BeforeInsert()
    @BeforeUpdate()
    checkYear() {
        const today = new Date()
        if (this.year > today.getFullYear() + 100 || this.year < 1888)
            throw new MovieError('Wrong movie year!')
    }
}
```

Watchlist:

```
@Unique( name: 'movie_user_constraint', fields: ["movie", "user"])
@Entity()
export class Watchlist {
    @PrimaryGeneratedColumn()
    id: number

    @Column( options: {
        nullable: true,
    })
    rate: number;

    @ManyToOne( typeFunctionOrTarget: () => Movie, inverseSide: (movie : Movie ) => movie.watchlist)
    movie: Movie;

    @ManyToOne( typeFunctionOrTarget: () => User, inverseSide: (user : User ) => user.watchlist)
    user: User;
}
```

Реализуем для них основные сервисы и контроллеры для CRUD-операций
Отличаться от всех будет сервис для watchlist’а, который должен обращаться к другим сервисам (user и movie):

```
const repository = AppDataSource.getRepository(Watchlist)
const userService = new UserService()
const movieService = new MovieService()

class WatchlistService {
    async getById(id: number, withMovies : boolean = false, withUser : boolean = false) {
        const watchlist = await repository.findOne( options: {
            relations: {
                movie: withMovies,
                user: withUser
            },
            where: {id}
        })
        if (watchlist) return watchlist
        throw new WatchlistError(`Watchlist with id = ${id} not found`)
    }

    async getAll() {
        return await repository.find( options: {
            relations: {
                movie: true
            }
        })
    }

    async getAllByUsername(username: string) {
        console.log(username)
        const user = await userService.getByUsername(username)
        return await repository.find( options: {
            relations: {
                user: false,
                movie: {
                    genre: true,
                    director: true
                }
            },
            where: {
                user: {
                    username: username
                }
            }
        })
    }
```

В роутах добавим middleware checkJWT, где это необходимо:

```
import express from 'express'
import WatchlistController from '../controllers/watchlist.controller'
import {checkJWT} from "../middlewares/checkJWT";

const router: express.Router = express.Router()

const controller = new WatchlistController()

router.route( prefix: '/').get(controller.getAll)

router.route( prefix: '/').post(checkJWT, controller.create)

router.route( prefix: '/:id').delete(checkJWT, controller.delete)

router.route( prefix: '/:id').put(checkJWT, controller.update)

router.route( prefix: '/my').get(checkJWT, controller.getAllByUsername)

export default router

```

Вывод

В ходе работы был создан был создан монолитный сервис, к которому можно направить все следующие запросы: