**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

# **Дисциплина:** Бек-энд разработка

Отчет

## Лабораторная Работа №2

Выполнил:

Кобелев Л.К.

K33401

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

## Задача

Разработать Rest API на boilerplate из ЛР1.

## Ход работы

Сделаем API для сервиса про игры. Определим следующие модели:

- Игра;
- Жанры (тэги игры);
- Классы жанров (тэгов);

Воспользуемся CSR-паттерном и рассмотрим часть API.

Модель `Game`:

```
export class Game {
    @PrimaryGeneratedColumn()
    id: number

    @Column( options: {
        unique: true,
    })
    gameId: number

    @Column()
    title: string

    @Column( options: {
        nullable: true,
    })
    description: string

    @Column( options: {
        nullable: true,
    })
    nameSlug: string

    @Column()
    reviews: number

    @Column( options: {
        type: 'date',
    })
    releaseDate: string

    @ManyToMany( typeFunctionOrTarget: (type) => Tag,  inverseSide: (tag :Tag ) => tag.games,  options: {
        cascade: true,
    })
    @JoinTable()
    tags: Tag[]
}
```

Эндпоинты:

```
import express from 'express'
import GameController from '../controllers/GameController'

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

const gameController = new GameController()

router.route( prefix: '/').get(gameController.getAllGames)

router.route( prefix: '/:id').get(gameController.getGame)

router.route( prefix: '/').post(gameController.createGame)

router.route( prefix: '/:id').delete(gameController.deleteGame)

router.route( prefix: '/:id').put(gameController.updateGame)

export default router
```

Соответственно есть:

- Получение всех игр;
- Получение конкретной игры;
- Добавление новой;
- Удаление конкретной;
- Обновление конкретной.

Рассмотрим `Game Controller`

```
class GameController {
    getAllGames = async (request: Request, response: Response) => {
        const tagsQuery = request.query.tag
        const sortQuery = request.query.sort || 'reviews'
        const orderQuery = request.query.order || 'DESC'
        const limitQuery = request.query.limit || 50
        const offsetQuery = request.query.offset || 0
        let searchQuery = request.query.search
        let usernameQuery = request.query.username

        if (searchQuery) {
            searchQuery = stringClean.cleanString(String(searchQuery))
        } else {
            searchQuery = '%'
        }

        if (usernameQuery) {
            usernameQuery = String(usernameQuery).trim()
        } else {
            usernameQuery = ''
        }

        let tagsIds = null
        if (tagsQuery) {
            tagsIds = convertor.convertQueryToNumbers(tagsQuery)
        }

        const allGames = await gameService.getFilteredGames(
            tagsIds,
            String(sortQuery),
            String(orderQuery),
            Number(limitQuery),
            Number(offsetQuery),
            searchQuery,
            usernameQuery
        )
            String(orderQuery),
            Number(limitQuery),
            Number(offsetQuery),
            searchQuery,
            usernameQuery
        )
        return response.send(allGames)
    }

    getGame = async (request: Request, response: Response) => {
        const id = Number(request.params.id)
        const game = await gameService.getById(id)
        return response.send(game)
    }

    createGame = async (request: Request, response: Response) => {
        const {
            gameId,
            title,
            description,
            nameSlug,
            reviews,
            releaseDate,
            tags,
        } = request.body
        const results = await gameService.create(
            gameId,
            title,
            description,
            nameSlug,
            reviews,
            releaseDate,
            tags
        )
        return response.send(results)
    }
```

Рассмотрим подробнее логику метода `getAllGames`, кажется, она самая интересная.

1. Получаем необходимые `query` параметры. Если нет, то ставим дефолтные;
2. Проверяем некоторые специфичные параметры на наличи. Например, если нет поиска, то ставим ‘%’ иначе “очищаем” строку;
3. Обращаемся к сервису и передаем ему все необходимые параметры;
4. Возвращаем результат работы сервиса.

`GameService.getFilteredGames()`

```
async getFilteredGames(
    tagsIds: number[] | null,
    sortOption: string,
    directionOption: string,
    limit: number,
    offset: number,
    searchString: string,
    usernameString: string
) {
    const gameSorting = new GameSorting(sortOption, directionOption)
    let gamesWithCount = await gameRepository.readAll(searchString)
    let games = gamesWithCount[0]

    if (tagsIds) {
        const tags: Tag[] = await tagService.getAllByIds(tagsIds)
        for (let i = 0; i < tags.length; i++) {
            const tagGames = tags[i].games
            games = games.filter(
                ({ id : number }) =>
                    tagGames.findIndex((game : Game ) => game.id === id) > -1
            )
        }
    }

    if (usernameString != '') {
        let userId
        if (usernameString.startsWith('https://steamcommunity.com/id/')) {
            usernameString = usernameString
                .split( separator: '/')
                .filter((item : string ) => item)
                .slice(-1)[0]
            userId = await steamService.GetSteamID(usernameString)
        } else if (
            usernameString.startsWith(
                'https://steamcommunity.com/profiles/'
```

```
            usernameString.startsWith(
                'https://steamcommunity.com/profiles/'
            )
        ) {
            userId = usernameString
                .split( separator: '/')
                .filter((item : string ) => item)
                .slice(-1)[0]
        } else {
            userId = await steamService.GetSteamID(usernameString)
        }
        if (userId) {
            const userGames = await steamService.GetUserGames(userId)
            if (userGames) {
                games = games.filter(
                    ({ gameId : number }) =>
                        userGames.findIndex(
                            (game) => game.appid === gameId
                        ) > -1
                )
            }
        }
    }

    const ids = games.map((game : Game ) => game.gameId)

    gamesWithCount = await gameRepository.readByIdsAndCount(
        ids,
        limit,
        offset,
        gameSorting
    )

    return gamesWithCount
}
```

Логика следующая:

1. Определяем параметры сортировки (отдельный интерфейс);
2. Получаем первый батч игр просто по поиску, считываем их количество;
3. Если есть жанры (тэги), то проходимся по ним циклом, получаем игры каждого жанра и ищем пересечения;
4. Если человек ввел `username`, то обращаемся к сервису работы с внешним `Steam API` - получаем игры пользователя и ищем объединение;
5. По итогу получаем набор игр, мапаем с `gameId` и получаем финальный результат, который возвращаем.

В ходе работы мы несколько раз обращаемся к `GameRepository`

```
async readByIdsAndCount(
    ids: number[],
    limit: number,
    offset: number,
    gameSorting: GameSorting
) {
    return await gameRepository.findAndCount( options: {
        where: {
            gameId: In( value: [...ids]),
        },
        order: {
            [gameSorting.parameter]: gameSorting.direction,
        },
        take: limit,
        skip: offset,
    })
}
```

Так, например, выглядит метод для чтения всех и возвращения кол-ва.

По такому принципу реализованы и все остальные методы API.

Весь их список: https://documenter.getpostman.com/view/23539417/2s93m4XiCL

## Вывод

Реализован RestAPI на ts + typeorm.