САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет
по лабораторной работе №1
«Boilerplate на express + sequelize / TypeORM + typescript»

Выполнила:

Киреева М.С.

Группа
К3333

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Написать свой boilerplate на express + sequelize / TypeORM + typescript.

Необходимо реализовать разделение на:

- модели
- контроллеры
- роуты
- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

Для упрощения работы с проектом будем использовать makefile со следующими командами

```
.PHONY: migrate
migrate:
    npx sequelize-cli db:migrate

.PHONY: init
init:
    npm init

.PHONY: install_d
install_d:
    npm install express -S
    npm install sequelize -S
    npm install sqlite3 -S
    npm install body-parser -S
    npm install cors -S
    npm install sequelize-typescript -S
    npm install tcs -S
    npm install typeorm -S
    npm install reflect-metadata -S
    npm install bcryptjs -S
    npm install passport -S
    npm install passport-jwt -S
    npm install uuid -S
    npm install --save-dev @types/bcrypt
    npm install --save-dev @types/cors
    npm install --save-dev @types/express
    npm install --save-dev @types/express-session
    npm install --save-dev @types/flat
    npm install --save-dev @types/node
    npm install --save-dev @types/passport
    npm install --save-dev @types/passport-jwt
    npm install --save-dev @types/styled-components
    npm install --save-dev @types/styled-system
    npm install --save-dev @types/validator
```

```
.PHONY: start
start:
	npx nodemon dist/index.js

.PHONY: lint
lint:
	npx eslint . --ext .ts

.PHONY: build
build:
	npx tcs

.DEFAULT_GOAL := init
```

Создадим файл package-lock.json командой npm init

Создадим свой файл с переменными окружения .env

```
# DATABASE
NAME = "user_db"
USERNAME = "root"
DIALECT = "sqlite"
PASSWORD = ""
STORAGE = "db.sqlite"


# JWT
ACCESS_TOKEN_LIFITIME = 300000
REFRESH_TOKEN_LIFITIME = 3600000

# SERVER
HOST = 127.0.0.1
PORT = 8000
```

Настроим файлы зависимостей и конфигураций:

Package.json

```
{
  "name": "l1",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "passport": "^0.6.0",
    "passport-jwt": "^4.0.1",
    "reflect-metadata": "^0.1.13",
    "sequelize": "^6.31.0",
    "sequelize-typescript": "^2.1.5",
    "sqlite3": "^5.1.6",
    "tcs": "^10.0.2",
    "typeorm": "^0.3.15",
    "uuid": "^9.0.0"
  },
  "description": "",
  "devDependencies": {
    "@types/bcrypt": "^5.0.0",
    "@types/cors": "^2.8.13",
    "@types/dotenv": "^8.2.0",
    "@types/express": "^4.17.17",
    "@types/express-session": "^1.17.7",
    "@types/flat": "^5.0.2",
```

.eslintrc.js (команда npm init @eslint/config)

```
module.exports = {
    "env": {
        "browser": true,
        "es2021": true
    },
    "extends": [
        "eslint:recommended",
        "plugin:react/recommended",
        "plugin:@typescript-eslint/recommended"
    ],
    "overrides": [
    ],
    "parser": "@typescript-eslint/parser",
    "parserOptions": {
        "ecmaVersion": "latest",
        "sourceType": "module"
    },
    "plugins": [
        "react",
        "@typescript-eslint"
    ],
    "rules": {
    }
}
```

tsconfig.json

~/Downloads/l1/tsconfig.json

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "strictPropertyInitialization": false,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "allowJs": true,
  },
}
```

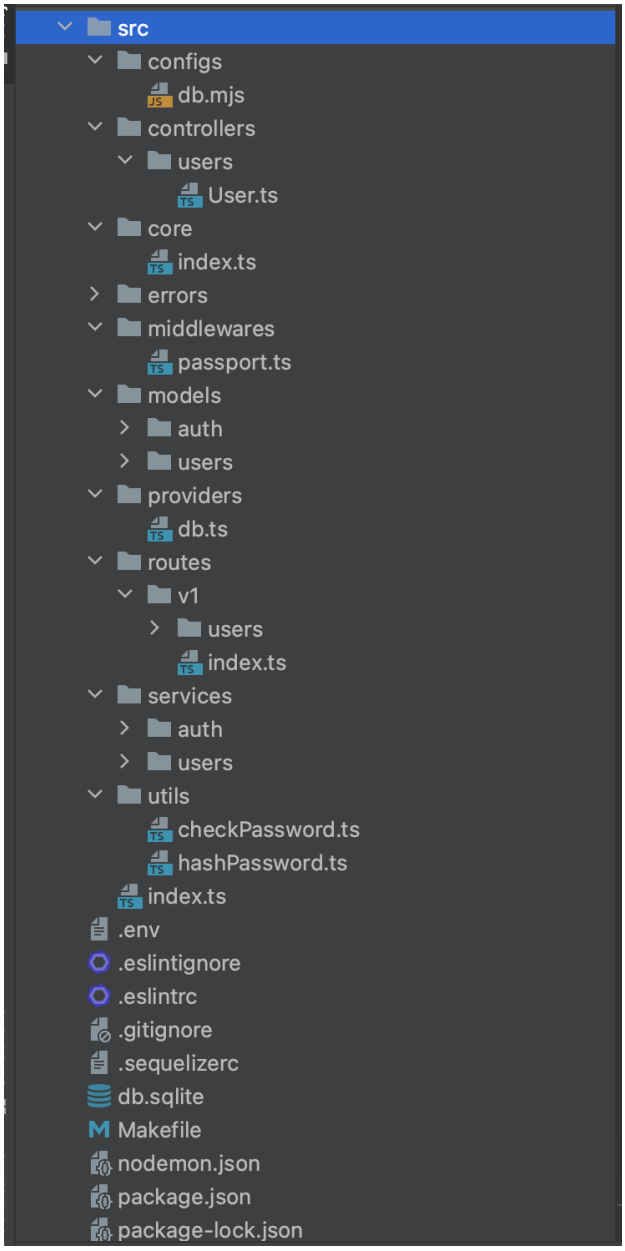
nodemon.json

```
{
  "watch": [
      "src"
  ],
  "ext": "ts",
  "ignore": [
      "node_modules",
      "src/**/*.spec.ts"
  ],
  "exec": "ts-node ./src/index.ts"
}
```

.sequelizerc

```
const path = require('path')

module.exports = {
  'config': path.resolve('src', 'configs/db.js'),
  'models-path': path.resolve('src', 'models'),
  'seeders-path': path.resolve('src', 'seeders'),
  'migrations-path': path.resolve('src', 'migrations')
}
```

Структура проекта:



- configs - файлы конфигурации (для подключения к БД)
- controllers – контроллеры
- core – точка входа в приложение
- index.ts - точка входа для запуска приложения
- middlewares - аутентификация с использованием Passport.js
- models - модели для модели пользователей и токенов.
- providers - провайдеры, управляющие зависимостями
- routes - маршруты
- services - службы, отвечающие за логику работы приложения
- utils - утилиты, которые используются во всем приложении

Модели:

Модель хранения токена

```
import { Table, Column, Model, Unique, AllowNull, ForeignKey } from 'sequelize-typescript'
import User from '../users/User'

5+ usages
@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull( allowNull: false)
    @Column
    no usages
    token: string

    @ForeignKey( relatedClassGetter: () => User)
    @Column
    1 usage
    userId: number
}

5+ usages
export default RefreshToken
```

Модель пользователя

Контроллеры:

```
2 usages
get = async (request: any, response: any) => {
    try {
        const user: User | UserError = await this.userService.getById(
            Number(request.params.id)
        )

        response.send(user)
    } catch (error: any) {
        response.status(404).send({ "error": error.message })
    }
}
```

```
2 usages
post = async (request: any, response: any) => {
    const { body } = request

    try {
        const user : User|UserError = await this.userService.create(body)

        response.status(201).send(user)
    } catch (error: any) {
        response.status(400).send({ "error": error.message })
    }
}
```

```
me = async (request: any, response: any) => {
    response.send(request.user)
}

2 usages
auth = async (request: any, response: any) => {
    const { body } = request

    const { email, password } = body

    try {
        const { user, checkPassword } = await this.userService.checkPassword(email, password)

        if (checkPassword) {
            const payload = { id: user.id }

            console.log('payload is', payload)

            const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)

            const refreshTokenService = new RefreshTokenService(user)

            const refreshToken = await refreshTokenService.generateRefreshToken()

            response.send({ accessToken, refreshToken })
        } else {
            throw new Error('Login or password is incorrect!')
        }
    } catch (e: any) {
        response.status(401).send({ "error": e.message })
    }
}
```

```
getAll = async (request: any, response: any) => {
    try {
        const users = await this.userService.getAll()

        response.send(users)
    } catch (error: any) {
        response.status(404).send({ "error": error.message })
    }
}
```

```
refreshToken = async (request: any, response: any) => {
    const { body } = request

    const { refreshToken } = body

    const refreshTokenService = new RefreshTokenService()

    try {
        const { userId, isExpired } = await refreshTokenService
            .isRefreshTokenExpired(refreshToken)

        if (!isExpired && userId) {
            const user = await this.userService.getById(userId)

            const payload = { id: user.id }

            const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)

            const refreshTokenService = new RefreshTokenService(user)

            const refreshToken = await refreshTokenService.generateRefreshToken()

            response.send({ accessToken, refreshToken })
        } else {
            throw new Error('Invalid credentials')
        }
    } catch (e) {
        response.status(401).send({ 'error': 'Invalid credentials' })
    }
}
```

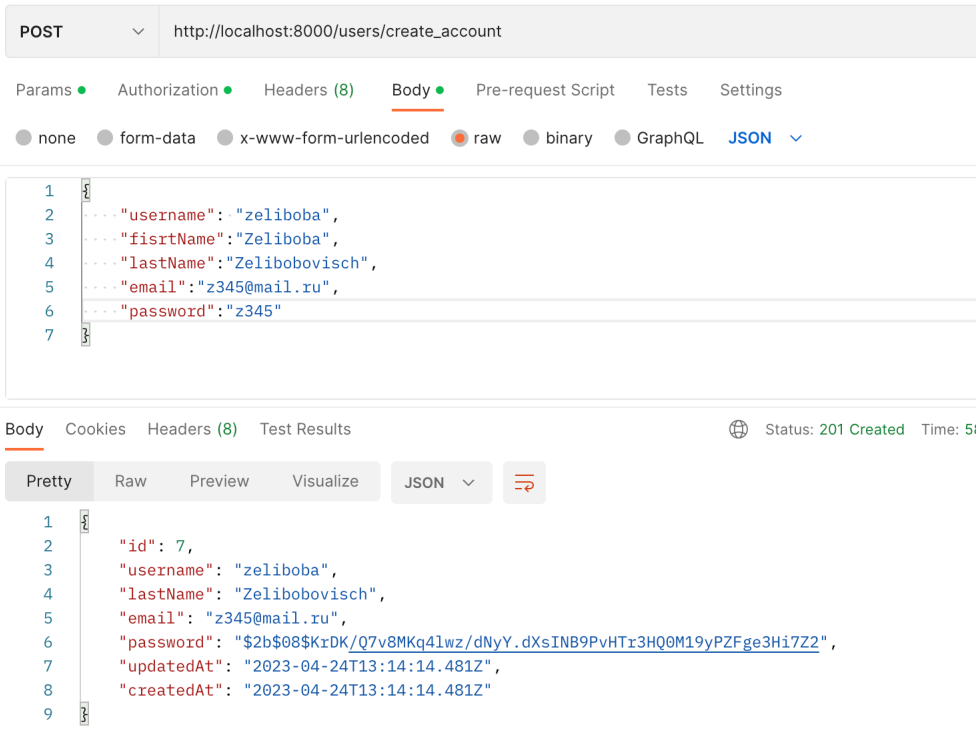
```
1 usage
deleteById = async (request: any, response: any)=> {
    try {
        const { id } = request.params;

        const deletedCount = await User.destroy( options: {
            where: {id: id}
        });

        if (deletedCount === 0) {
            throw new UserError(`User with id ${id} not found`);
        }
        else {
            response.send(`User with id ${id} was deleted`)
        }

        response.status(204).send();
    } catch (error: any) {
        response.status(404).send({ "error": error.message })
    }
}
```

Routes

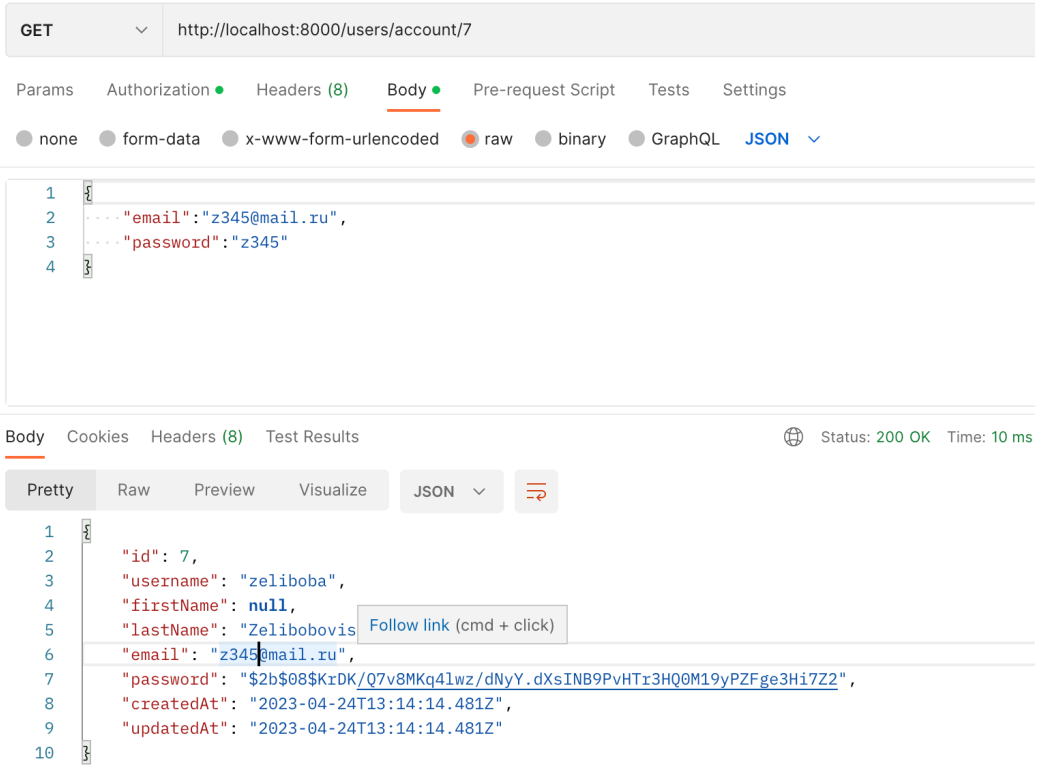
```
import express from "express"
import UserController from "../../../controllers/users/User"
import passport from "../../../middlewares/passport"

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

const controller: UserController = new UserController()

router.route( prefix: '/create_account')
    .post(controller.post)

router.route( prefix: '/auth')
    .get(passport.authenticate( strategy: 'jwt',  options: { session: false }), controller.me)

router.route( prefix: '/account/:id')
    .get(controller.get)

router.route( prefix: '/login')
    .post(controller.auth)

router.route( prefix: '/refresh')
    .post(controller.refreshToken)

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

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

2 usages
export default router
```

Проверка

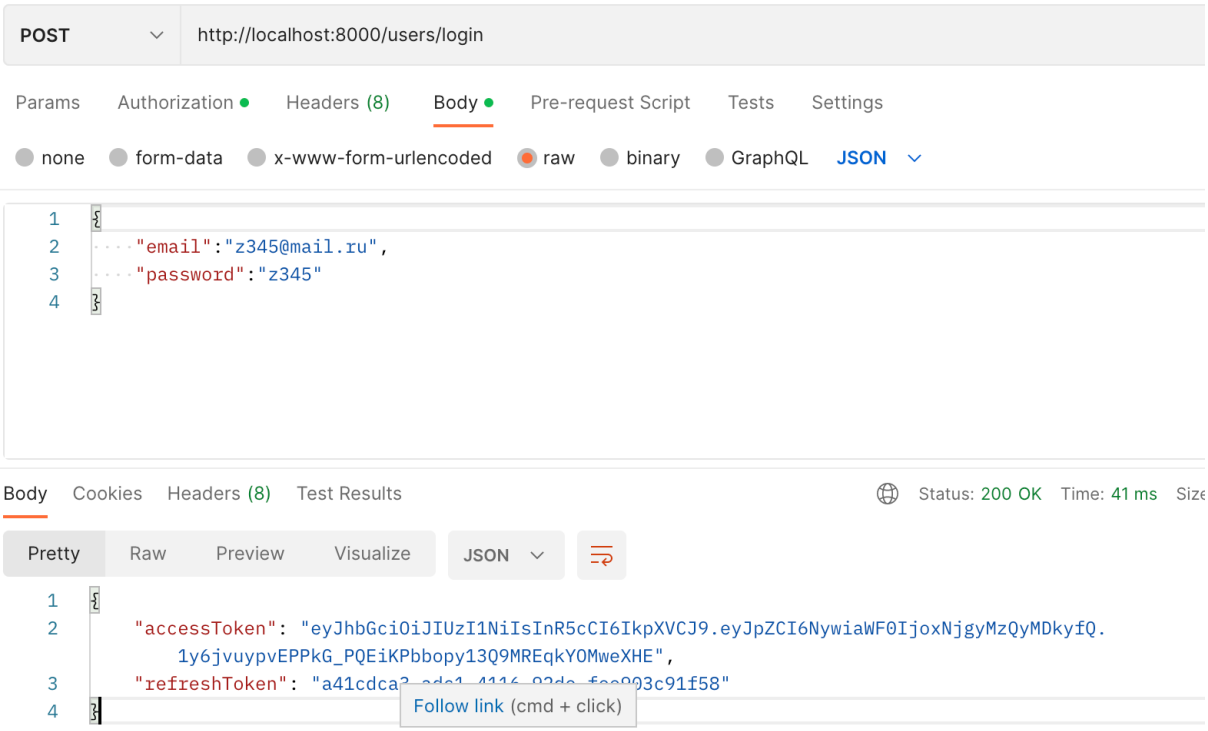
Создание пользователя



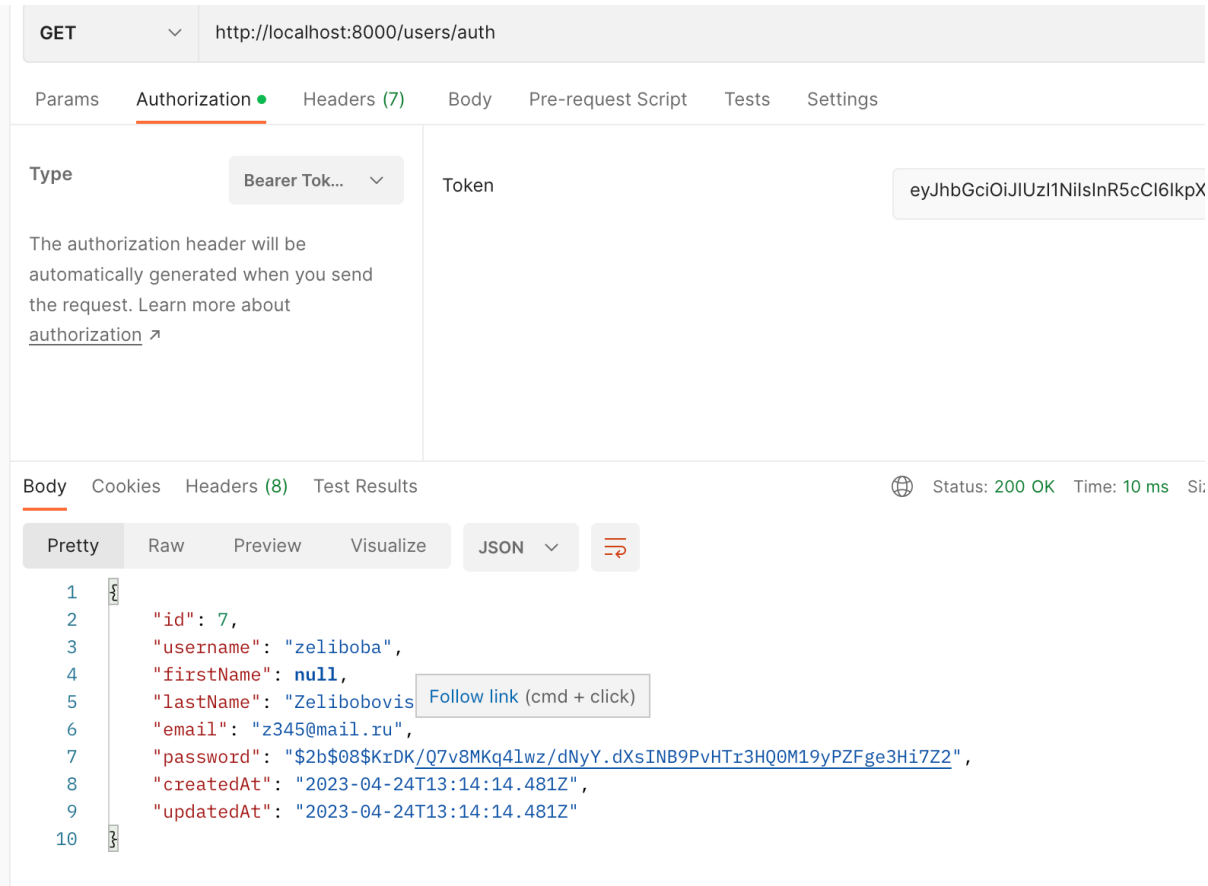
Поиск по id



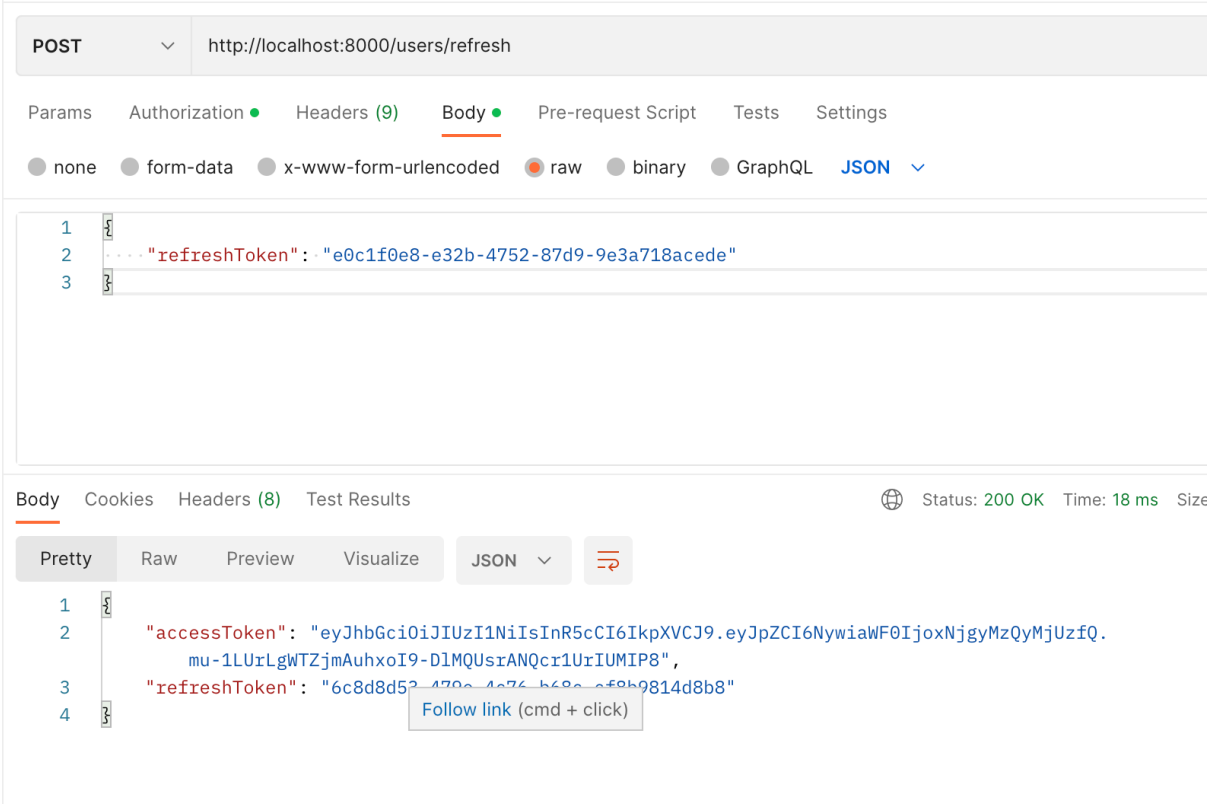
Вход в профиль



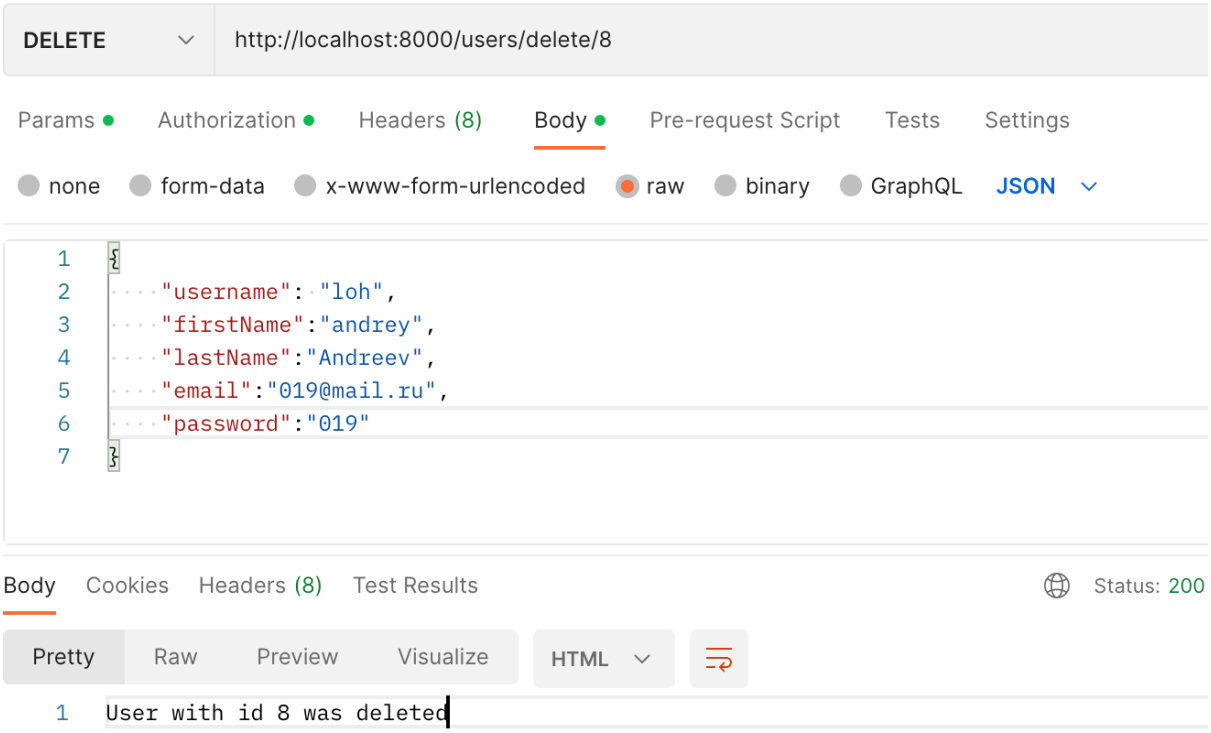
Поиск по токену



Обновление токена



Удаление аккаунта



Вывод

В ходе работы с использованием express + sequelize / TypeORM + typescrip был создан boilerplate, который в дальнейшем может использоваться как шаблон для новых проектов.