САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Таначев Егор

Группа K33412

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

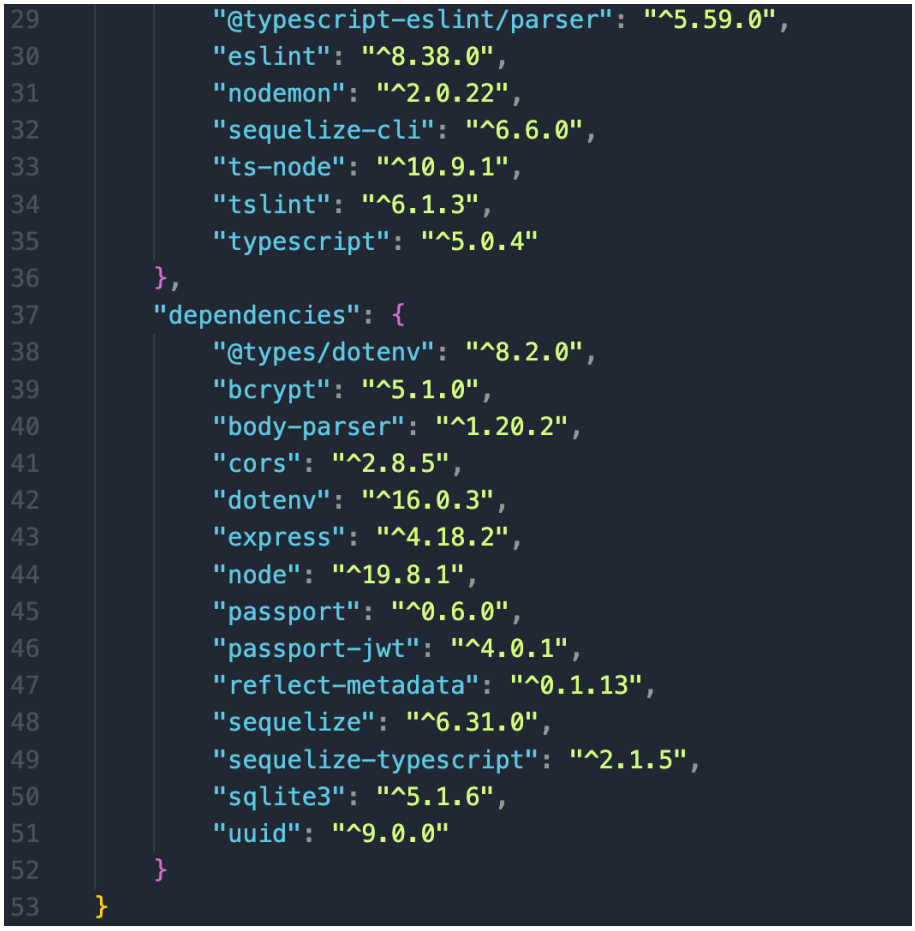
Написать boilerplate на express + sequelize + typescript с явным разделением на:

- Модели;
- Контроллеры;
- Роуты;
- Сервисы для работы с моделями (паттерн “репозиторий”)

Ход работы

Для начала работы необходимо проинициализировать проект, установить nodemon автоматического перезапуска проекта при изменении и составить Makefile. Файл с конфигурацией проекта показан на Рисунке 1.

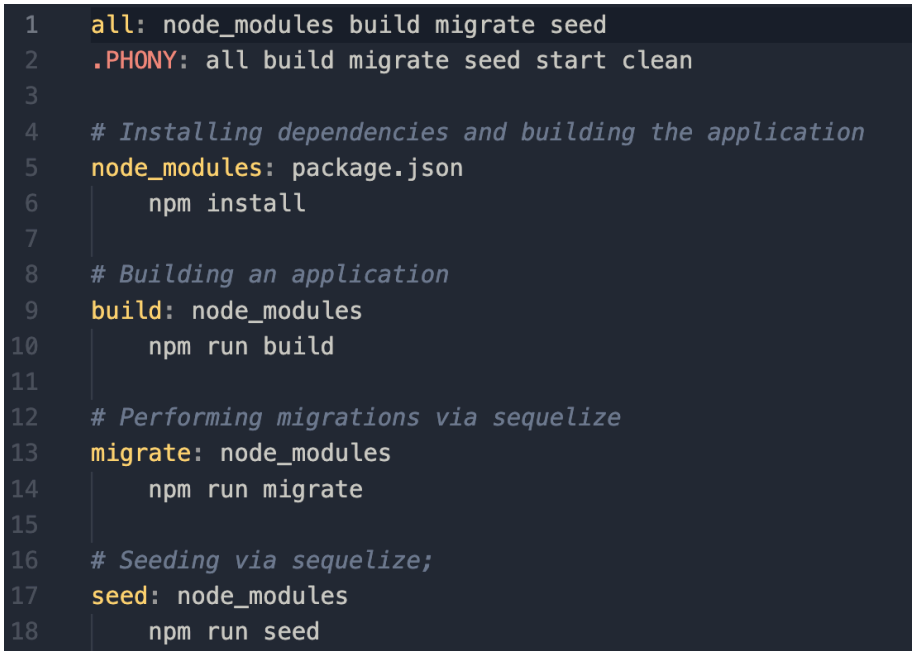
```
{
    "name": "lw1",
    "version": "1.0.0",
    "description": "laboratorywork01",
    "main": "index.js",
    ▷ Отладка
    "scripts": {
        "prestart": "npm run build",
        "start": "nodemon dist/index.js",
        "build": "npx tsc",
        "lint": "npx eslint . --ext .ts",
        "migrate": "npx sequelize db:migrate"
    },
    "author": "Tanachev Egor",
    "license": "ISC",
    "devDependencies": {
        "@types/bcrypt": "^5.0.0",
        "@types/cors": "^2.8.13",
        "@types/express": "^4.17.17",
        "@types/express-session": "^1.17.7",
        "@types/flat": "^5.0.2",
        "@types/node": "^18.15.11",
        "@types/passport": "^1.0.12",
        "@types/passport-jwt": "^3.0.8",
        "@types/styled-components": "^5.1.26",
        "@types/styled-system": "^5.1.16",
        "@types/uuid": "^9.0.1",
        "@types/validator": "^13.7.15",
        "@typescript-eslint/eslint-plugin": "^5.59.0",
```



```
        "@typescript-eslint/parser": "^5.59.0",
        "eslint": "^8.38.0",
        "nodemon": "^2.0.22",
        "sequelize-cli": "^6.6.0",
        "ts-node": "^10.9.1",
        "tslint": "^6.1.3",
        "typescript": "^5.0.4"
    },
    "dependencies": {
        "@types/dotenv": "^8.2.0",
        "bcrypt": "^5.1.0",
        "body-parser": "^1.20.2",
        "cors": "^2.8.5",
        "dotenv": "^16.0.3",
        "express": "^4.18.2",
        "node": "^19.8.1",
        "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",
        "uuid": "^9.0.0"
    }
}
```

Рисунок 1 – Файл package.json

Makefile показан на Рисунке 2. Для запуска используется *make (команда)*.



```
all: node_modules build migrate seed
.PHONY: all build migrate seed start clean

# Installing dependencies and building the application
node_modules: package.json
    npm install

# Building an application
build: node_modules
    npm run build

# Performing migrations via sequelize
migrate: node_modules
    npm run migrate

# Seeding via sequelize;
seed: node_modules
    npm run seed
```

```
19
20  # Checking the code for compliance with the linter rules
21  lint: node_modules
22      npm run lint
23
24  # Launching the application
25  start: node_modules
26      npm start
27
28  # Cleaning generated files
29  clean:
30      rm -rf node_modules db.sqlite dist
```

Рисунок 2 – Makefile

В работе необходимо было задать грамотную структуру, как показано на Рисунке 3.

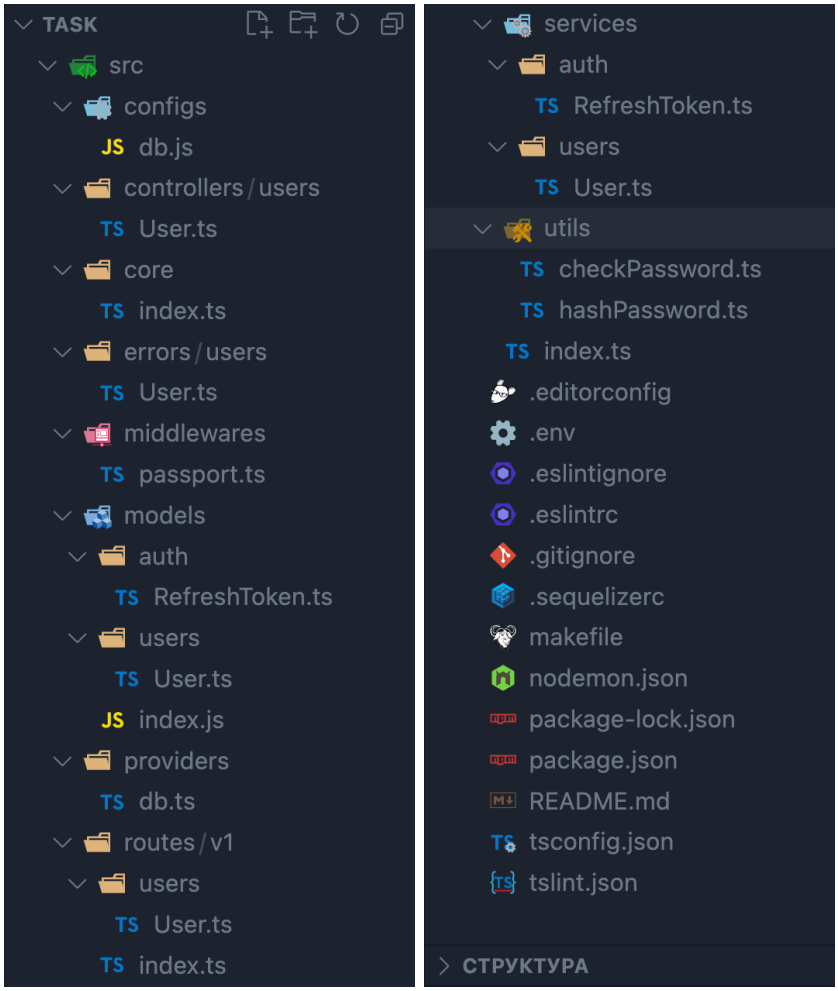


Рисунок 3 – Структура проекта

Опишем модель Пользователя, класс UserService, класс UserController и роутинг, как показано на Рисунках 4-7.

```ts
import { Table, Column, Model, Unique, AllowNull, Befor
import hashPassword from '../../utils/hashPassword'

@Table
class User extends Model {
    @AllowNull(false)
    @Column
    firstName: string

    @AllowNull(false)
    @Column
    lastName: string

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string

    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const { password } = instance

        if (instance.changed('password')) {
            instance.password = hashPassword(password)
        }
    }
}

export default User
```

Рисунок 4 – Модель User

```ts
import User from '../../models/users/User'
import UserError from '../../errors/users/User'
import checkPassword from '../../utils/checkPassword'

class UserService {
    async getById(id: number) : Promise<User> {
        const user = await User.findByPk(id)

        if (user) return user.toJSON()

        throw new UserError('Not found')
    }

    async create(userData: any) : Promise<User|UserError> {
        try {
            const user = await User.create(userData)

            return user.toJSON()
```

```
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new UserError(errors)
        }
    }

    async checkPassword(email: string, password: string) : Promise<any> {
        const user = await User.findOne({ where: { email } })

        if (user) return { user: user.toJSON(), checkPassword: checkPassword(user, password) }

        throw new UserError('Incorrect login/password!')
    }

    async getAll() {
        const users = await User.findAll()

        if (users) return users

        throw new UserError('Not found')
    }

    async getByEmail(email: string) {
        const user = await User.findOne({where: {email}})

        if (user) return user.toJSON()

        throw new UserError('Not found')
    }
}

export default UserService
```

Рисунок 5 – Класс UserService

```
import User from '../../models/users/User'
import UserService from '../../services/users/User'
import UserError from '../../errors/users/User'
import jwt from 'jsonwebtoken'
import { jwtOptions } from '../../middlewares/passport'
import RefreshTokenService from '../../services/auth/RefreshToken'

class UserController {
    private userService: UserService

    constructor() {
        this.userService = new UserService()
    }

    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 })
        }
    }
```

```
26
27      post = async (request: any, response: any) => {
28          const { body } = request
29
30          try {
31              const user : User|UserError = await this.userService.create(body)
32
33              response.status(201).send(user)
34          } catch (error: any) {
35              response.status(400).send({ "error": error.message })
36          }
37      }
38
39      me = async (request: any, response: any) => {
40          response.send(request.user)
41      }
42
43      auth = async (request: any, response: any) => {
44          const { body } = request
45
46          const { email, password } = body
47
48          try {
49              const { user, checkPassword } = await this.userService.checkPassword(email, passwo
50
51              if (checkPassword) {
52                  const payload = { id: user.id }
53
54                  const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
55
56                  const refreshTokenService = new RefreshTokenService(user)
57
58                  const refreshToken = await refreshTokenService.generateRefreshToken()
59
60                  response.send({ accessToken, refreshToken })
61              } else {
62                  throw new Error('Login or password is incorrect!')
63              }
64          } catch (e: any) {
65              response.status(401).send({ "error": e.message })
66          }
67      }
68
69      refreshToken = async (request: any, response: any) => {
70          const { body } = request
71
72          const { refreshToken } = body
73
74          const refreshTokenService = new RefreshTokenService()
75
76          try {
77              const { userId, isExpired } = await refreshTokenService
78                  .isRefreshTokenExpired(refreshToken)
79
80              if (!isExpired && userId) {
81                  const user = await this.userService.getById(userId)
82
83                  const payload = { id: user.id }
84
85                  const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
86
```

```
                const refreshTokenService = new RefreshTokenService(user)

                // tslint:disable-next-line:no-shadowed-variable
                const refreshToken = await refreshTokenService.generateRefreshToken()

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

    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 })
        }
    }

    getByEmail = async (request: any, response: any) => {
        try {
            const user = await this.userService.getByEmail(
                request.params.email
            )

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

Рисунок 6 – Класс UserController

```
router.route('/')
    .post(controller.post)

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

router.route('/email/:email')
    .get(controller.getByEmail)

router.route('/all')
    .get(controller.getAll)

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

router.route('/profile')
    .get(passport.authenticate('jwt', { session: false }), controller.me)

router.route('/refresh')
    .post(controller.refreshToken)
```

Рисунок 7 – Роутинг

Протестируем работу boilerplate в Postman, как показано на Рисунках 8-11.

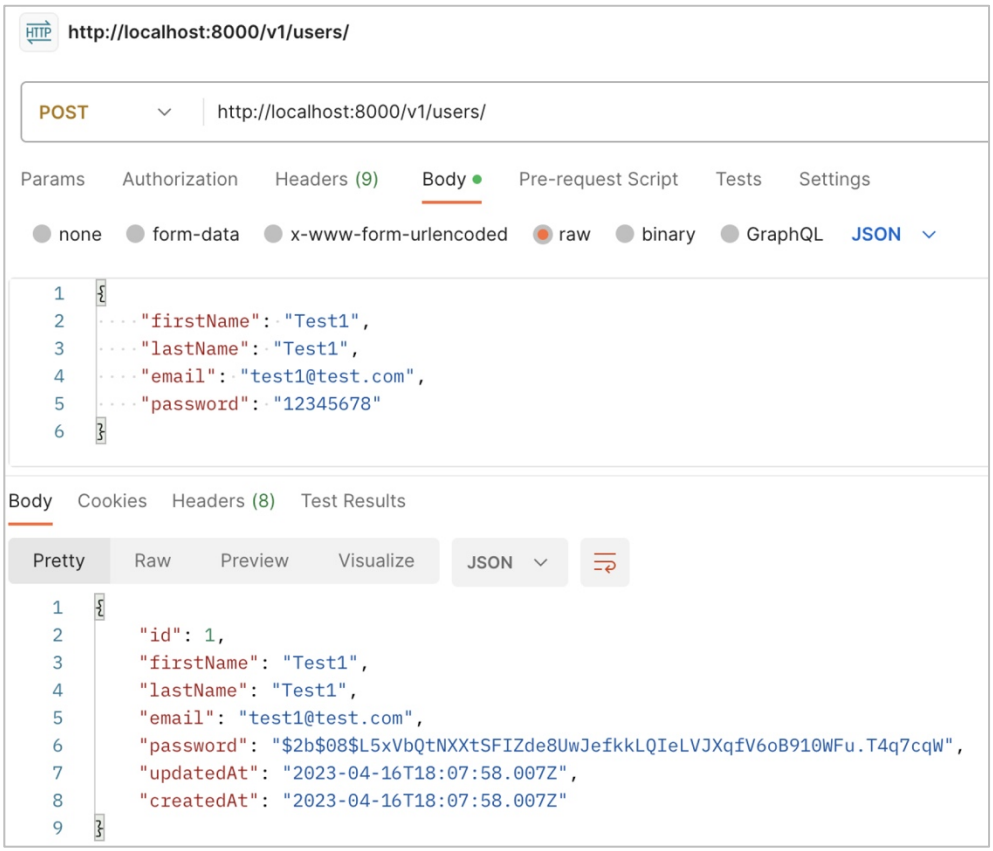


Рисунок 8 – Создание пользователя

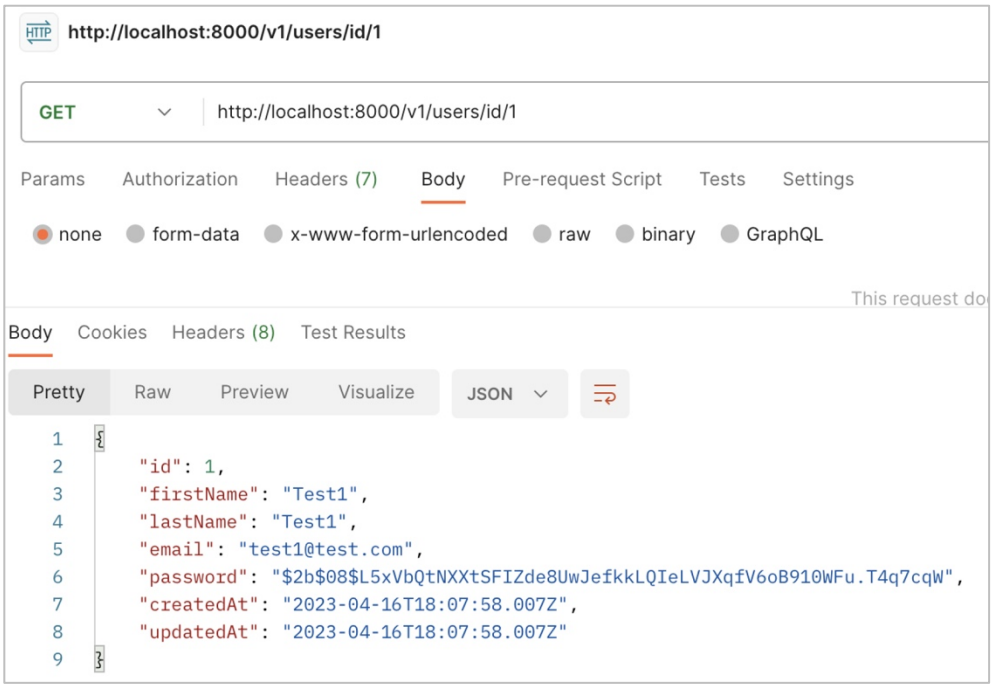


Рисунок 9 – Выдача пользователя по id

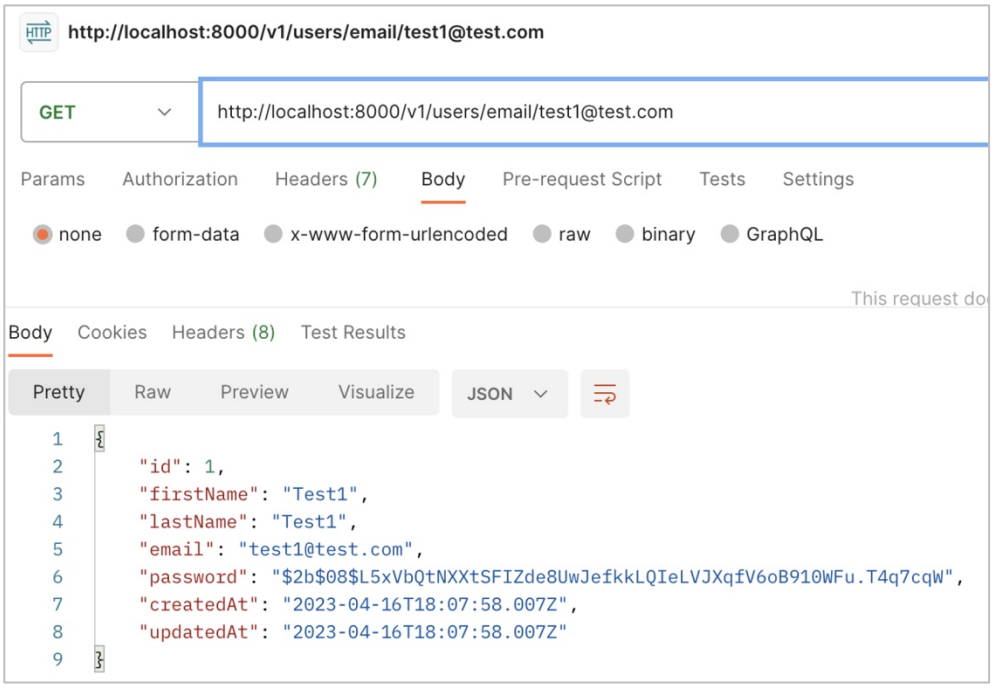


Рисунок 10 – Выдача пользователя по email

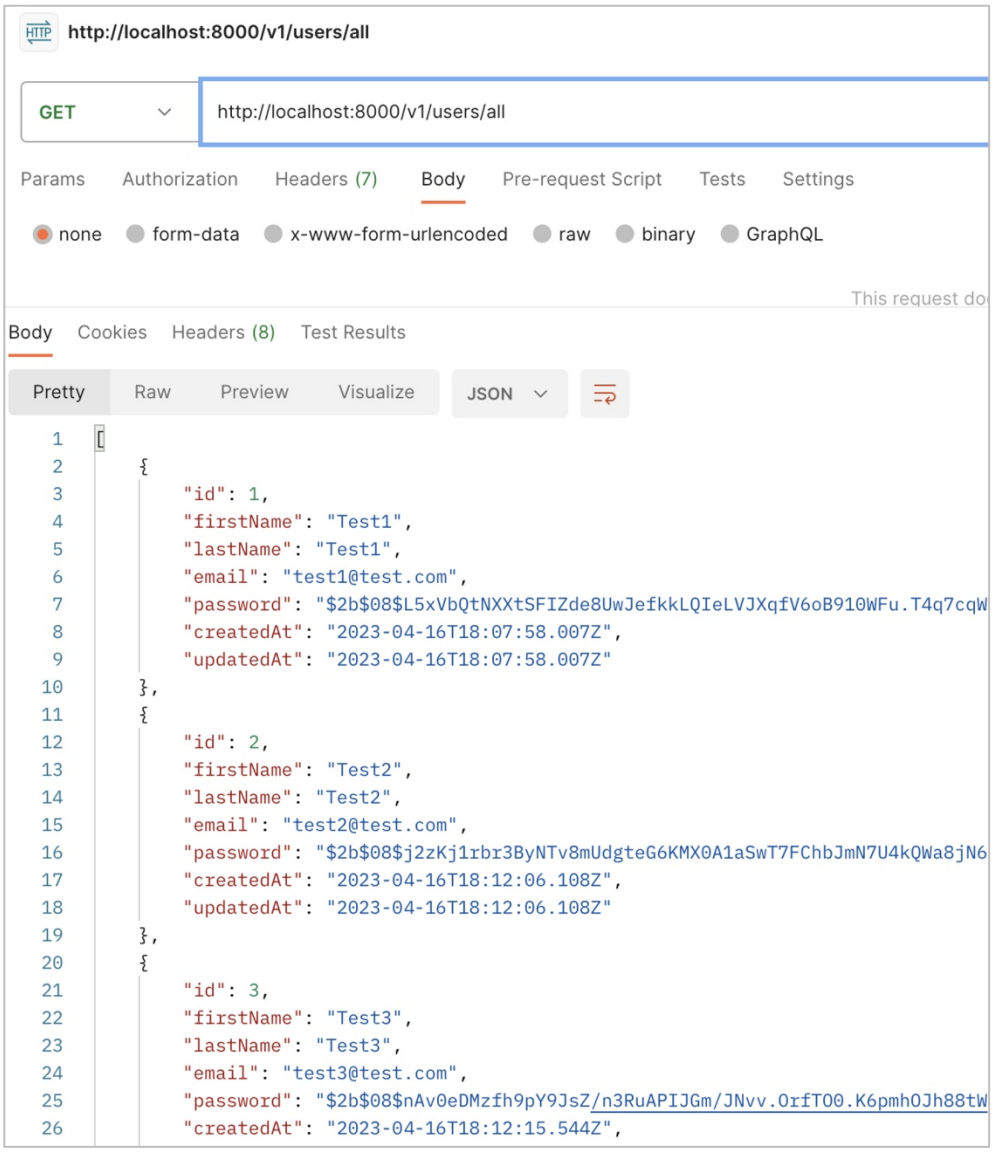


Рисунок 11 – Выдача всех пользователей

Вывод

В результате работы был разработан собственный boilerplate на основе фреймворка Express и библиотек Sequelize с использованием TypeScript. Boilerplate был структурирован с явным разделением на модели, контроллеры, роуты и сервисы, реализующие паттерн "репозиторий".