САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1
«Boilerplate на express + sequelize /TypeORM + typescript»

Выполнила:
Коник А. А.
Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Необходимо написать свой boilerplate на express + sequelize / TypeORM +typescript.

Должно быть явное разделение на:

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

Ход работы

package.json

```
{
    "name": "boilerplate-lr1",
    "version": "1.0.0",
    "description": "Simple boilerplate for express RESTful API, using TypeScript and Sequelize ORM",
    "main": "index.js",
    "scripts": {
        "prestart": "npm run build",
        "start": "nodemon dist/index.js",
        "build": "npx tsc",
        "lint": "npx eslint . --ext .ts",
        "migrate": "npx sequelize db:migrate"
    },
    "author": "",
    "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"
}
```

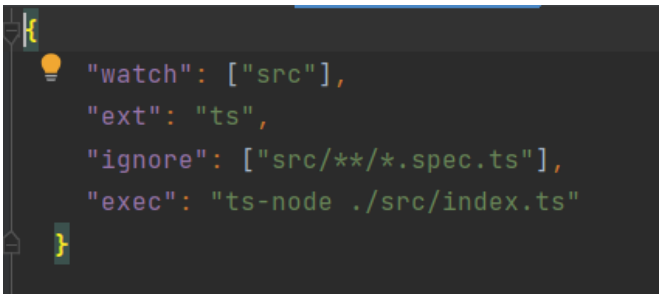
.eslintrc.js - файл конфигурации ESLint

```
{
    "root": true,
    "env": {
        "node": true
    },
    "parser": "@typescript-eslint/parser",
    "plugins": [
        "@typescript-eslint"
    ],
    "extends": [
        "eslint:recommended",
        "plugin:@typescript-eslint/eslint-recommended",
        "plugin:@typescript-eslint/recommended"
    ],
    "rules": {
        "@typescript-eslint/no-explicit-any": "off",
        "@typescript-eslint/no-non-null-assertion": "off"
    }
}
```

.sequelizerc - файл конфигурации sequelize

```
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')
}
```

nodemon.json – файл конфигурации пакета nodemon

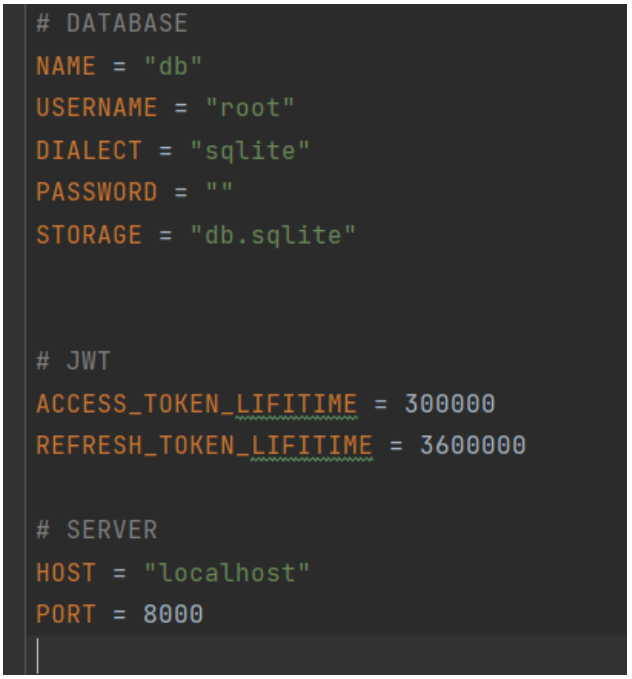


```
{
  "watch": ["src"],
  "ext": "ts",
  "ignore": ["src/**/*.spec.ts"],
  "exec": "ts-node ./src/index.ts"
}
```

tsconfig.json - файл конфигурации TypeScript

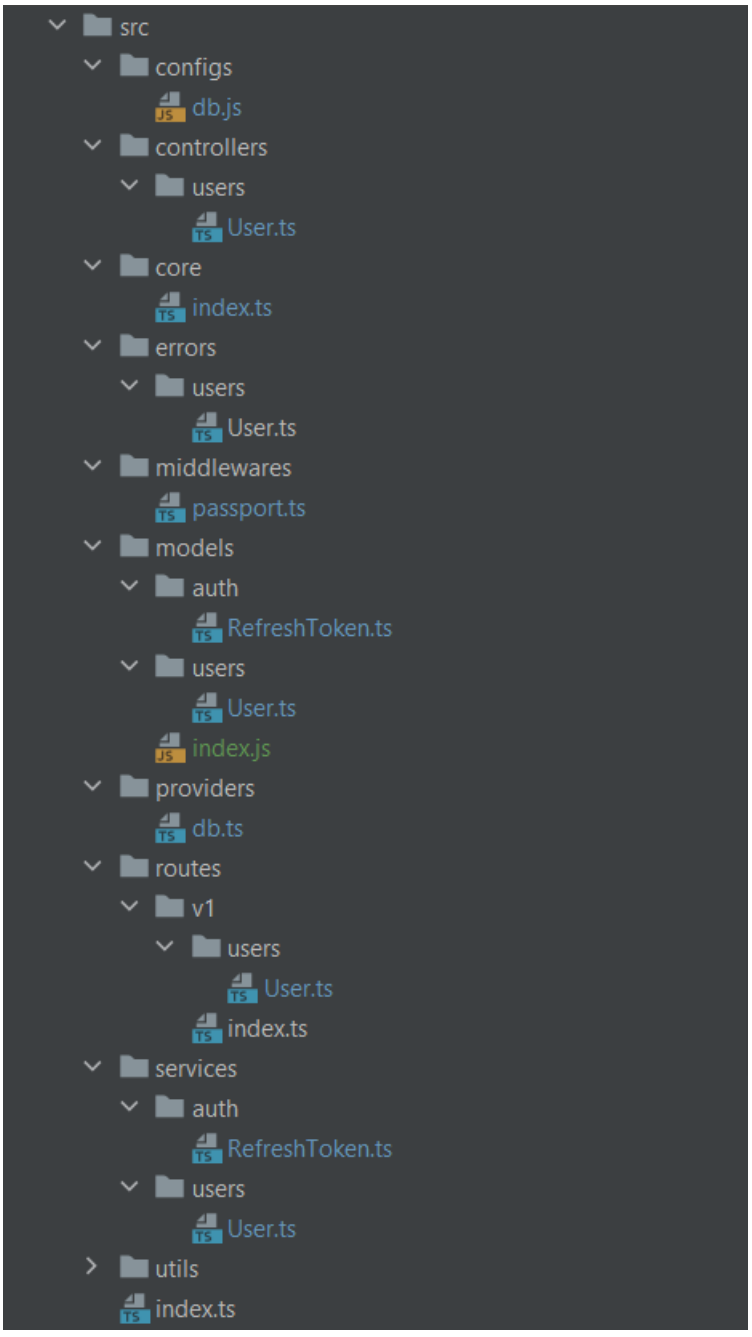
```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "strictPropertyInitialization": false,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  }
}
```

.env – файл с переменными окружения



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


# JWT
ACCESS_TOKEN_LIFITIME = 300000
REFRESH_TOKEN_LIFITIME = 3600000

# SERVER
HOST = "localhost"
PORT = 8000
```

Структура приложения:



Где

- **core** – точка входа в приложение;
- **configs** – файлы конфигурации (файл для подключения к БД);
- **controllers** – контроллеры, отвечающие за логику обработки http-запросов;
- **models** – модели sequelize;
- **providers** – точки доступа к данным;
- **routes** – описание маршрутов;
- **services –** службы, которые содержат запросы к базе данных и

возвращают объекты или выдают ошибки;

•**utils** – вспомогательные файлы, которые используются в приложении;

•**middlewares** - содержит аутентификацию с использованием passport.ts.

Модели:

User.ts – модель пользователя

```
7+ usages  Anastasia Konik
@Table
class User extends Model {
    @AllowNull( allowNull: false)
    @Column
    no usages  Anastasia Konik
    name: string

    @AllowNull( allowNull: false)
    @Column
    1 usage  Anastasia Konik
    lastname: string

    @Unique
    @Column
    3 usages  Anastasia Konik
    email: string

    @AllowNull( allowNull: false)
    @Column
    5+ usages  Anastasia Konik
    password: string

    1 usage  Anastasia Konik
    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const { password } = instance

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

RefreshToken.ts - модель хранения токенов

```
5+ usages  Anastasia Konik
@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull( allowNull: false)
    @Column
    1 usage  Anastasia Konik
    token: string

    @ForeignKey( relatedClassGetter: () => User)
    @Column
    3 usages  Anastasia Konik
    userId: number
}
```

Контроллеры:

```
class UserController {
    Anastasia Konik
    private userService: UserService

    1 usage  Anastasia Konik
    constructor() {
        this.userService = new UserService()
    }

    2 usages  Anastasia Konik
    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  Anastasia Konik
    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 })
        }
    }
```

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

2 usages  Anastasia Konik
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 }

            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 })
    }
}
```

```
2 usages  Anastasia Konik
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)

            // tslint:disable-next-line:no-shadowed-variable
            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' })
    }
}
```

```
2 usages  Anastasia Konik
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 })
    }
}

2 usages  Anastasia Konik
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 })
    }
}
}

3 usages  Anastasia Konik
export default UserController
```

Методы класса UserController:

- get: находит пользователя по id;
- post: создание нового пользователя;
- me: возвращает данные о пользователе;
- auth: генерирует новый токен доступа и токен обновления, если пользователь залогинился;
- refreshToken: генерирует новый JWT токен;
- getAll: получает информацию по всем пользователям;
- getByEmail: находит пользователя по его почте.

Services:

User.ts

```
class UserService {
    5+ usages  Anastasia Konik
    async getById(id: number) : Promise<User> {
        const user = await User.findByPk(id)

        if (user) return user.toJSON()

        throw new UserError('Not found!')
    }

    4 usages  Anastasia Konik
    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)
        }
    }
```

```
    2 usages  Anastasia Konik
    async checkPassword(email: string, password: string) : Promise<any> {
        const user = await User.findOne( options: { where: { email } })

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

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

    3 usages  Anastasia Konik
    async getAll() {
        const users = await User.findAll()

        if (users) return users

        throw new UserError('Users not found')
    }

    3 usages  Anastasia Konik
    async getByEmail(email: string) {
        const user = await User.findOne( options: {where: {email}})

        if (user) return user.toJSON()

        throw new UserError('User with this email not found')
    }
}
```

RefreshToken.ts

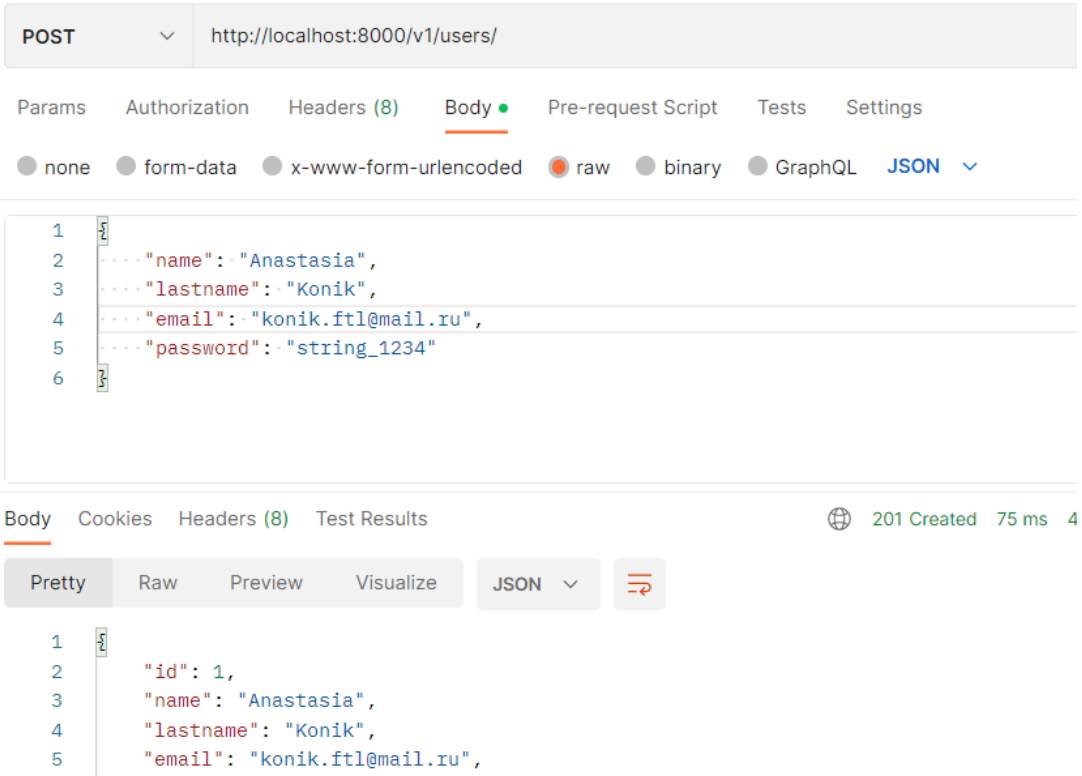
```
5+ usages  Anastasia Konik
class RefreshTokenService {
    Anastasia Konik
    private user: User | null

    3 usages  Anastasia Konik
    constructor(user: User | null = null) {
        this.user = user
    }

    4 usages  Anastasia Konik
    generateRefreshToken = async () : Promise<string> => {
        const token = randomUUID()

        const userId = this.user?.id

        await RefreshToken.create( values: { token, userId })

        return token
    }
```

```
    2 usages  Anastasia Konik
    isRefreshTokenExpired = async (token: string) : Promise<{ userId: number|null, isExpired: boolean }> => {
        const refreshToken = await RefreshToken.findOne( options: { where: { token } })

        if (refreshToken) {
            const tokenData = refreshToken.toJSON()

            const currentDate = new Date()
            const timeDelta = currentDate.getTime() - tokenData.createdAt.getTime()

            if (timeDelta > 0 && timeDelta < parseInt(process.env.REFRESH_TOKEN_LIFETIME!,  radix: 10)) {
                return { userId: tokenData.userId, isExpired: false }
            }

            return { userId: null, isExpired: true }
        }

        return { userId: null, isExpired: true }
    }
}

4 usages  Anastasia Konik
export default RefreshTokenService
```

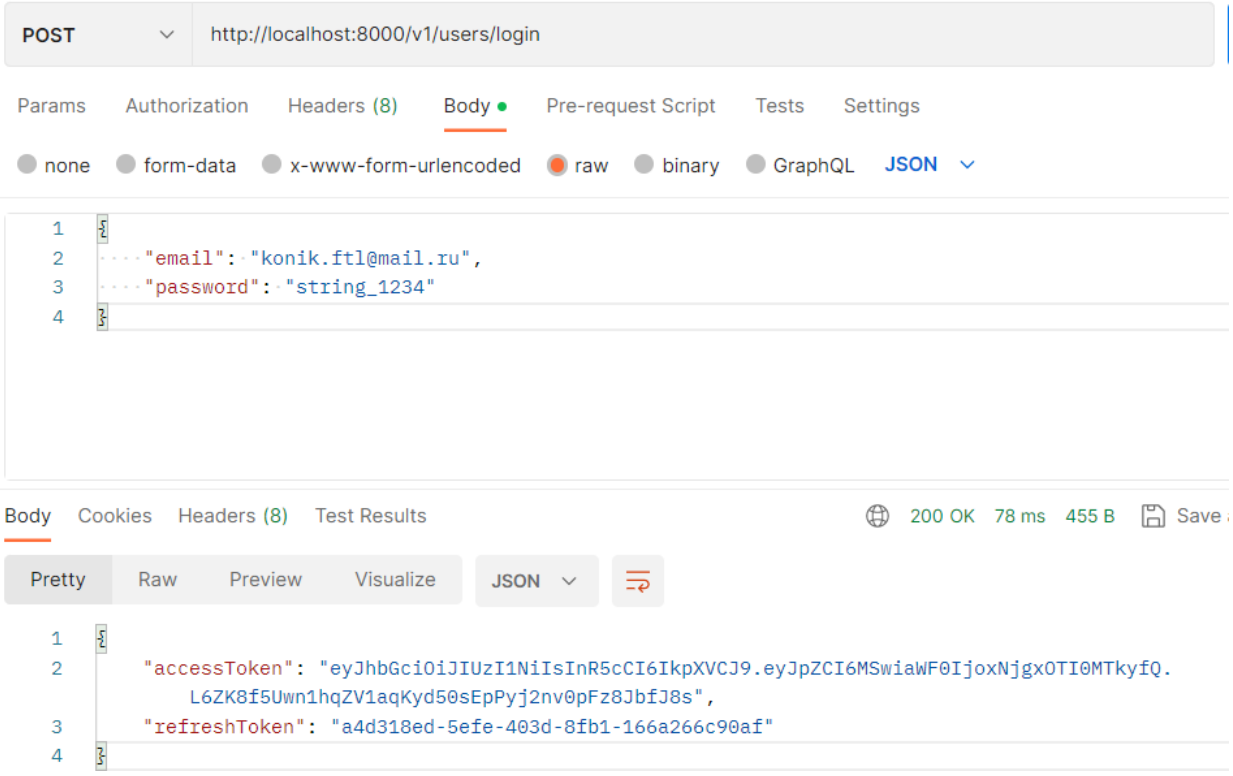
Routes:

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

const controller: UserController = new UserController()

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

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

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

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

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

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

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

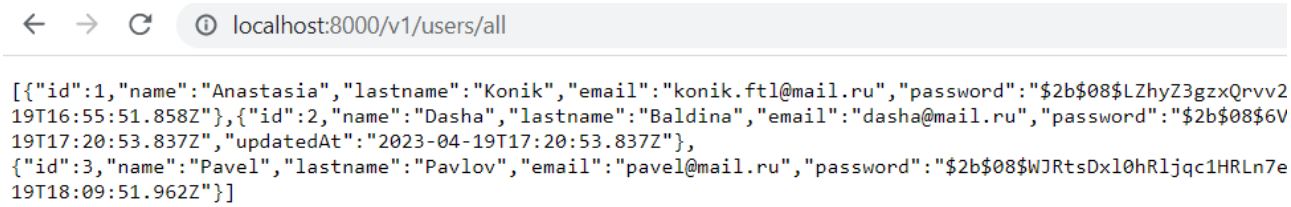
Регистрация пользователя:



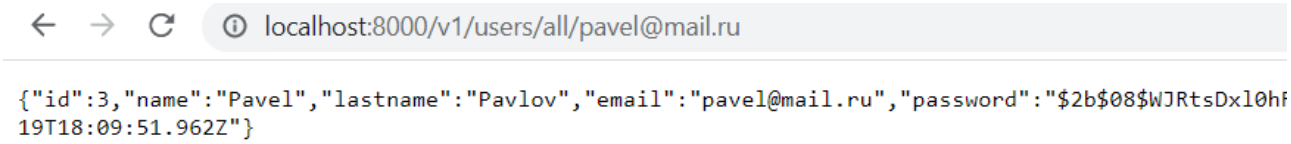
Авторизация:



Получение всех пользователей:



Получение пользователя по его почте:



Вывод

В ходе лабораторной работы был разработан свой boilerplate с продуманной архитектурой и реализованной логикой авторизации спомощью express, sequelize и typescript.