

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)**

Факультет Инфокоммуникационных технологий (ИКТ)

Образовательная программа Мобильные и сетевые технологии

О Т Ч Е Т

Back-end лабораторная работа №1

Тема задания: «Boilerplate на express + sequelize /TypeORM + typescriptL»

Выполнил: Мамедов Тогрул, группа К33402

Проверил: Добряков Давид Ильич

Санкт-Петербург
2023

Задача

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

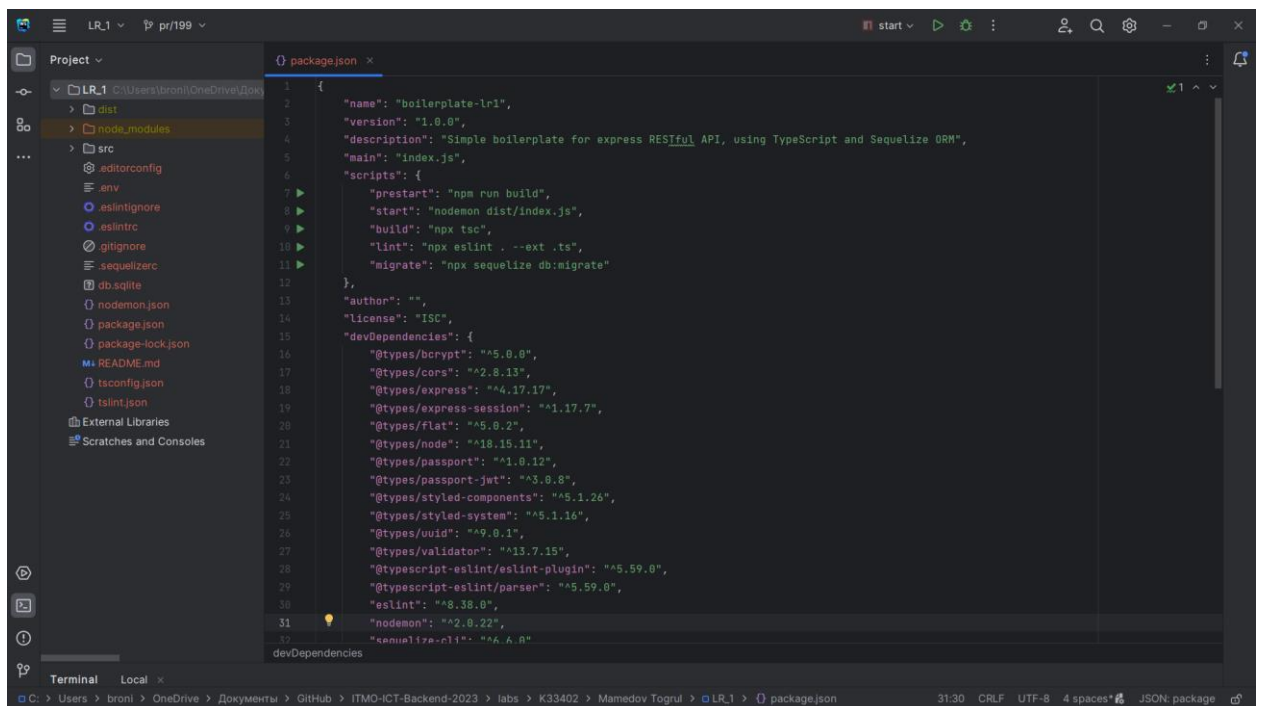
Должно быть явное разделение на:

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

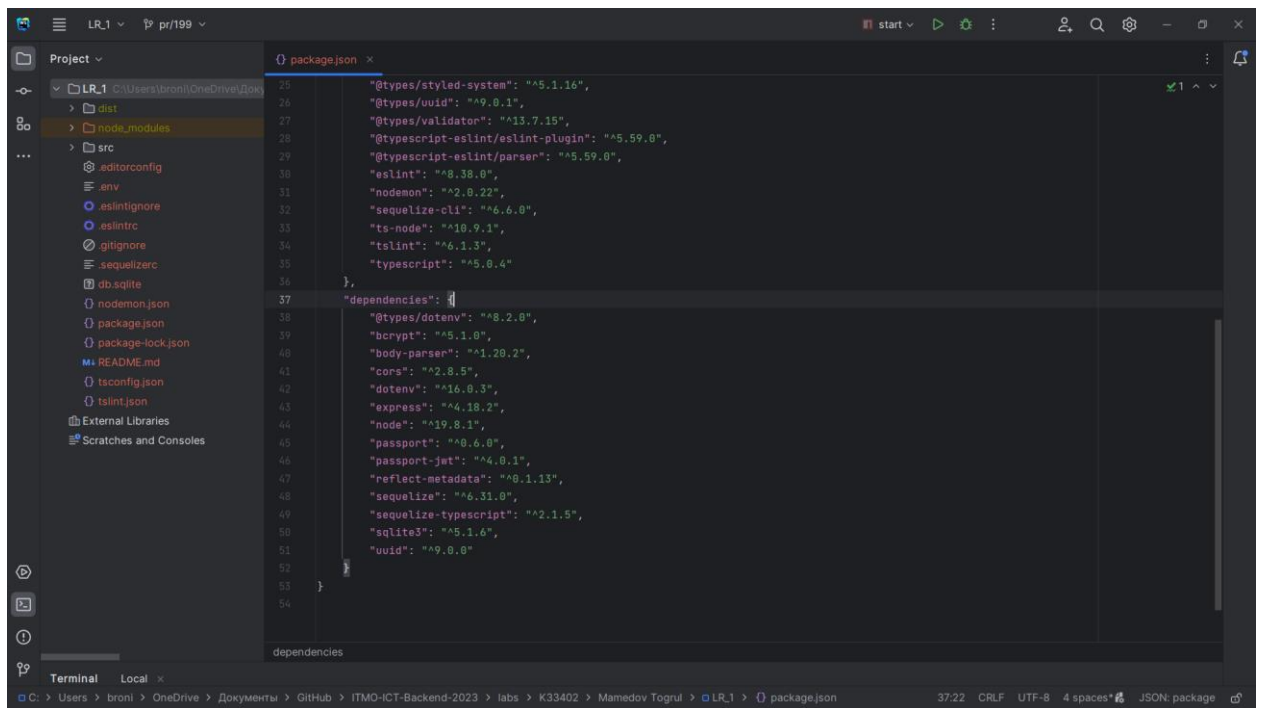
Ход работы

package.json

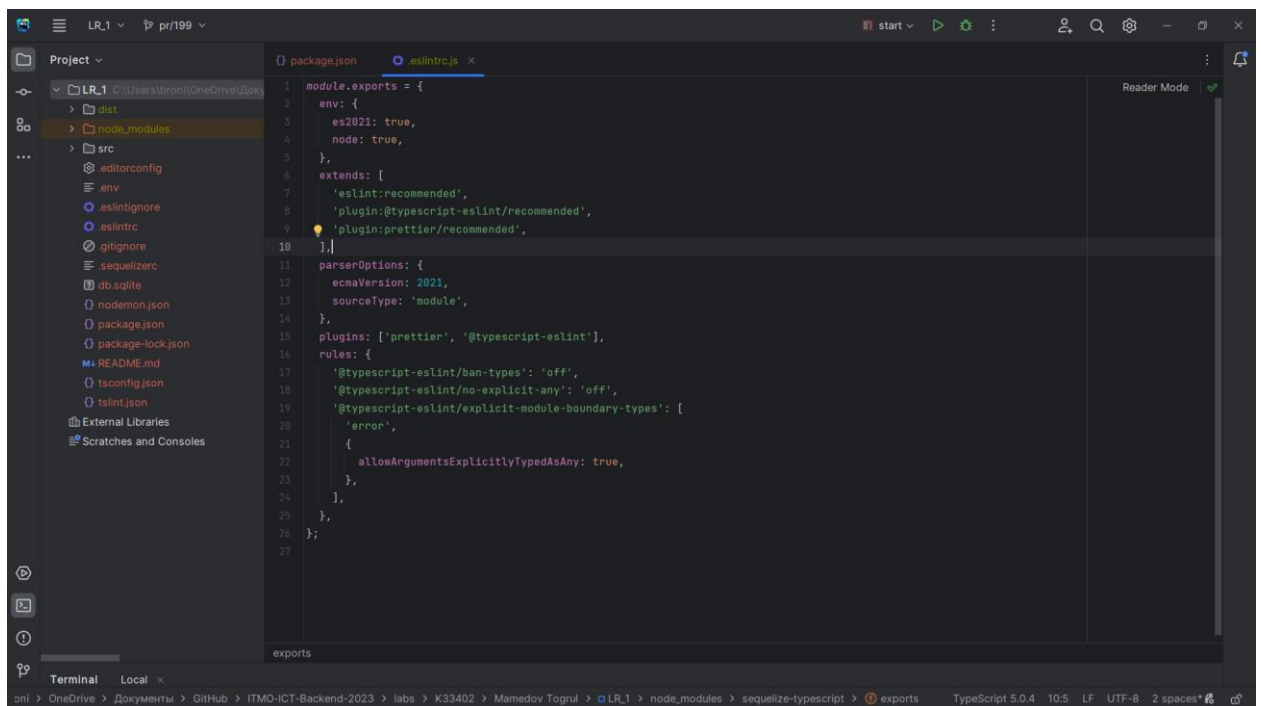
.eslintrc.js - файл конфигурации ESLint



```
1 {
2   "name": "boilerplate-lr1",
3   "version": "1.0.0",
4   "description": "Simple boilerplate for express RESTful API, using TypeScript and Sequelize ORM",
5   "main": "index.js",
6   "scripts": {
7     "prestart": "npm run build",
8     "start": "nodemon dist/index.js",
9     "build": "npx tsc",
10    "lint": "npx eslint . --ext .ts",
11    "migrate": "npx sequelize db:migrate"
12  },
13  "author": "",
14  "license": "ISC",
15  "devDependencies": {
16    "@types/bcrypt": "^5.0.0",
17    "@types/cors": "^2.8.13",
18    "@types/express": "^4.17.17",
19    "@types/express-session": "^1.17.7",
20    "@types/flat": "^5.0.2",
21    "@types/node": "^18.15.11",
22    "@types/passport": "^1.0.12",
23    "@types/passport-jwt": "^3.0.8",
24    "@types/styled-components": "^5.1.26",
25    "@types/styled-system": "^5.1.16",
26    "@types/uuid": "^9.0.1",
27    "@types/validator": "^13.7.15",
28    "@typescript-eslint/eslint-plugin": "^5.59.0",
29    "@typescript-eslint/parser": "^5.59.0",
30    "eslint": "^8.38.0",
31    "nodemon": "^2.0.22",
32    "sequelize-cli": "^6.6.0"
33  }
34 }
```

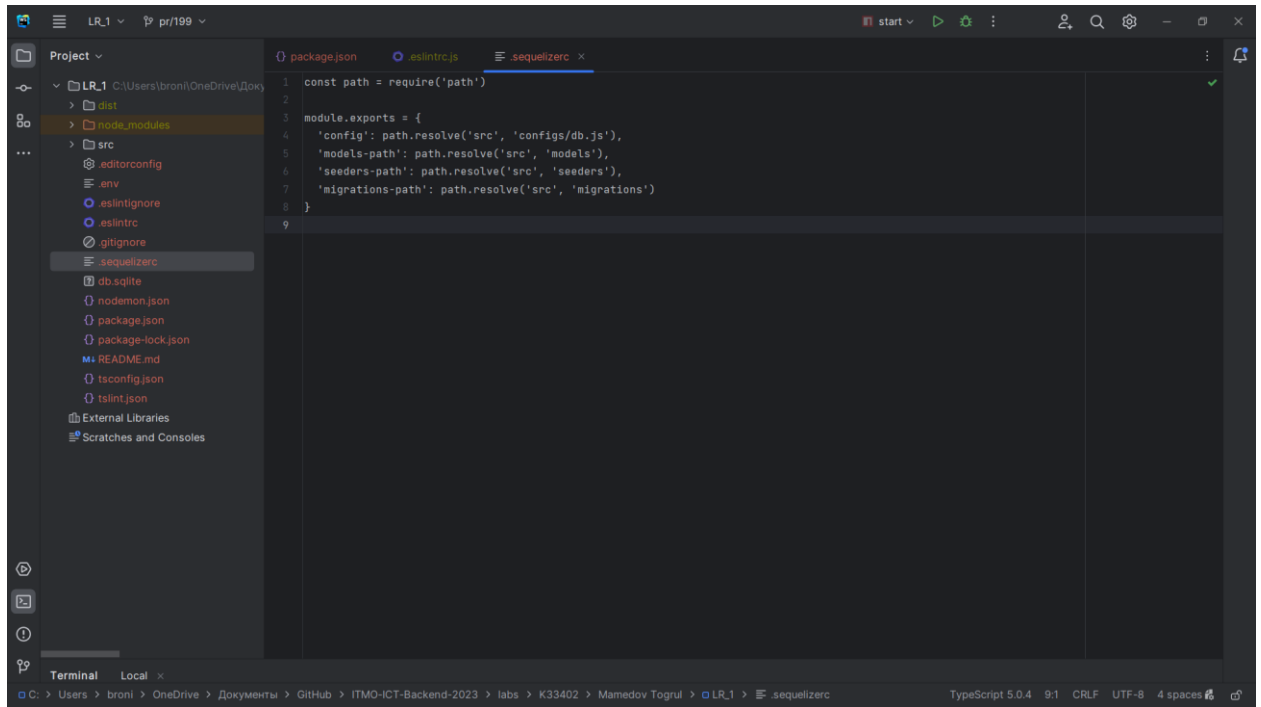


.eslintrc.js - файл конфигурации ESLint



.sequelizerc - файл конфигурации sequelize

0

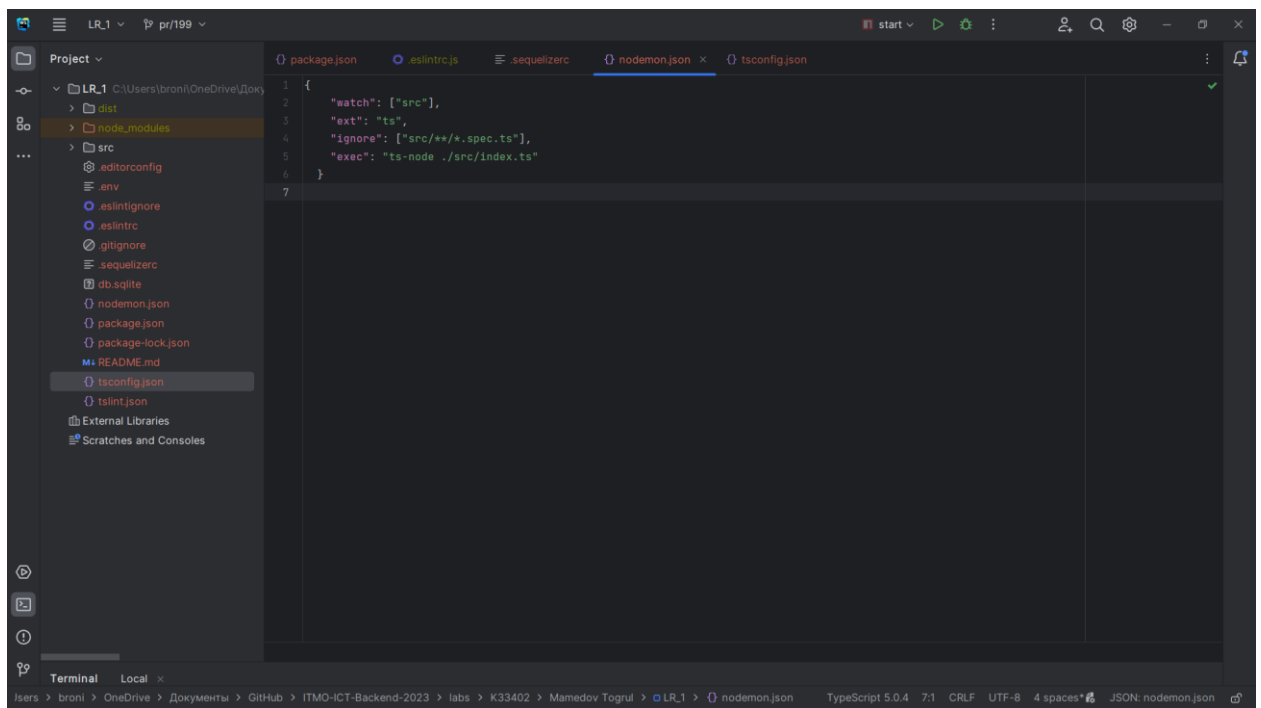


The screenshot shows the VS Code editor with the `.sequelizerc` file open. The file contains the following code:

```
1 const path = require('path')
2
3 module.exports = {
4   'config': path.resolve('src', 'configs/db.js'),
5   'models-path': path.resolve('src', 'models'),
6   'seeders-path': path.resolve('src', 'seeders'),
7   'migrations-path': path.resolve('src', 'migrations')
8 }
9
```

The left sidebar shows the project structure with `.sequelizerc` selected under the `node_modules` directory. The status bar at the bottom indicates the file is `.sequelizerc` in TypeScript 5.0.4.

nodemon.json – файл конфигурации пакета nodemon

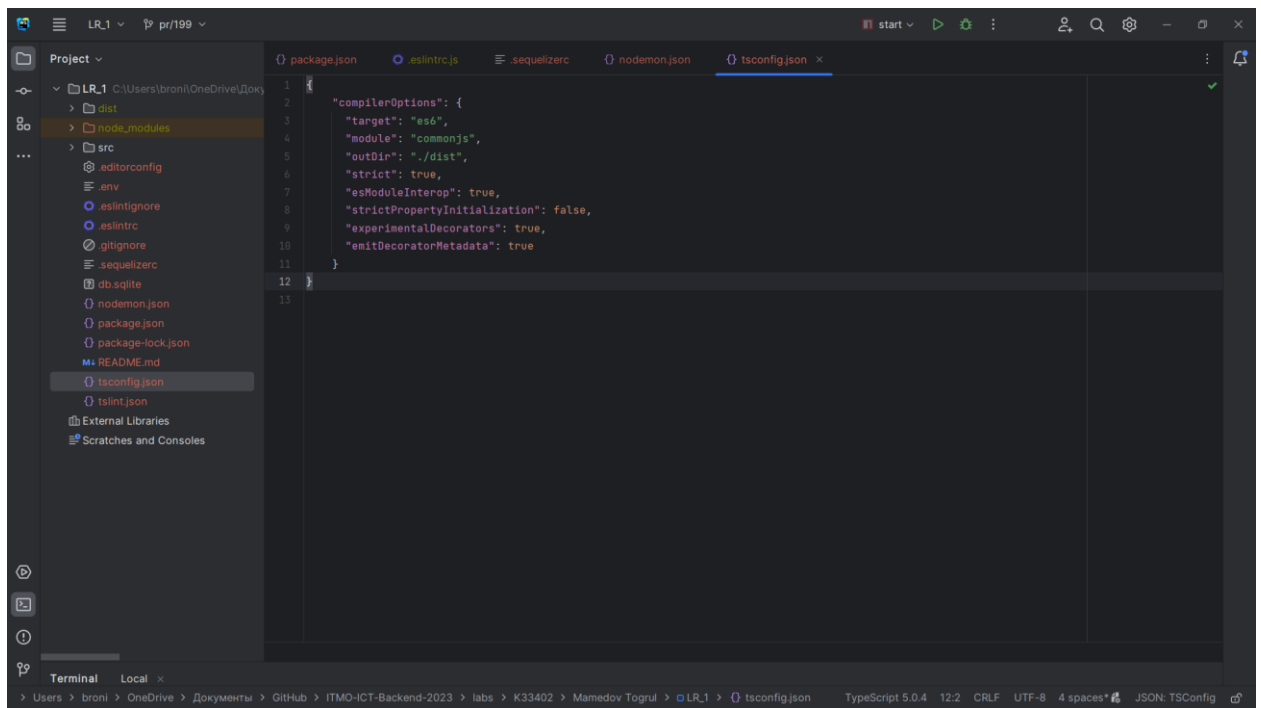


The screenshot shows the VS Code editor with the `nodemon.json` file open. The file contains the following code:

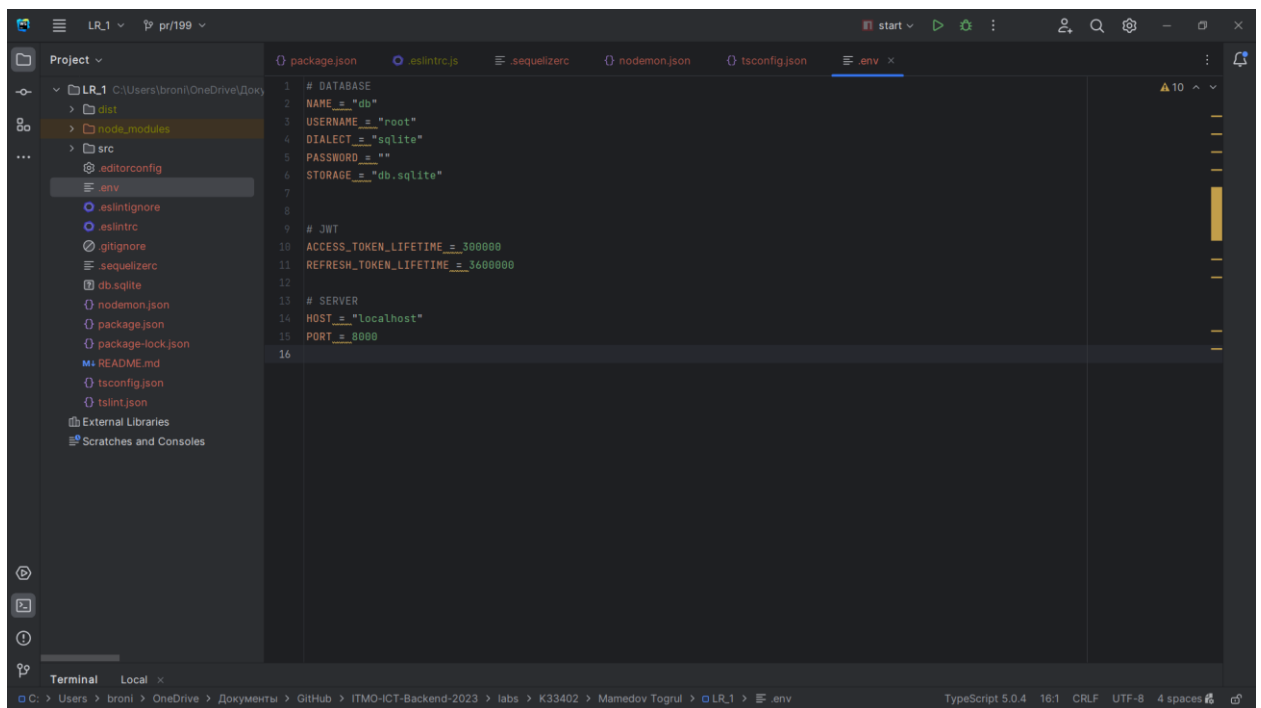
```
1 {
2   "watch": ["src"],
3   "ext": "ts",
4   "ignore": ["src/**/*.spec.ts"],
5   "exec": "ts-node ./src/index.ts"
6 }
7
```

The left sidebar shows the project structure with `nodemon.json` selected under the `node_modules` directory. The status bar at the bottom indicates the file is `nodemon.json` in TypeScript 5.0.4.

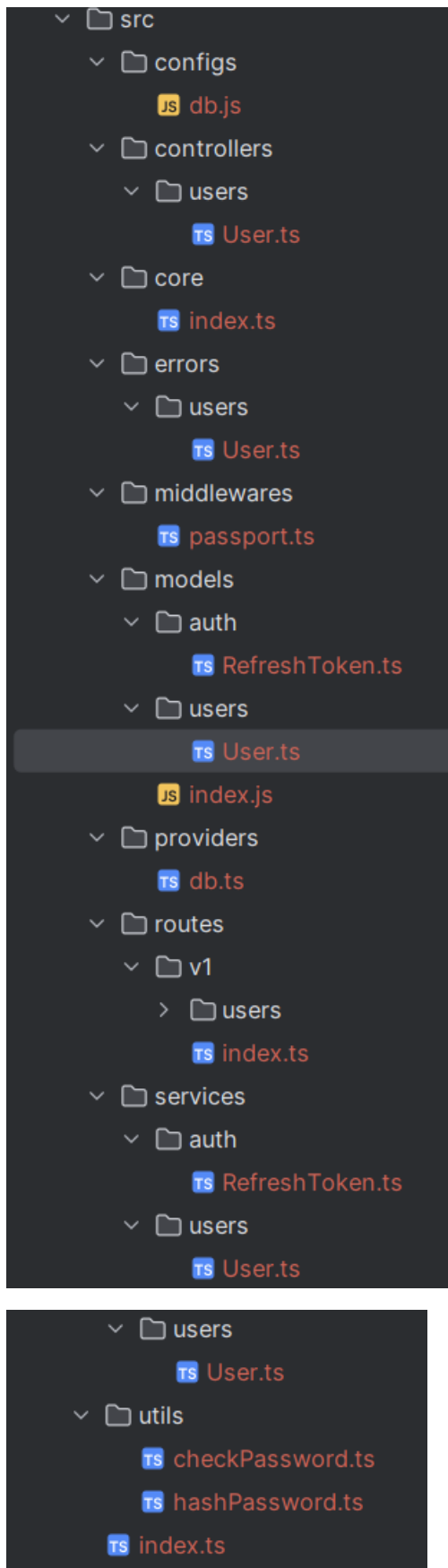
tsconfig.json - файл конфигурации TypeScript



.env – файл с переменными окружения



Структура приложения:



Где:

core – точка входа в приложение;

configs – файлы конфигурации (файл для подключения к БД);

controllers – контроллеры, отвечающие за логику обработки httpзапросов;

models – модели sequelize;

providers – точки доступа к данным;

routes – описание маршрутов;

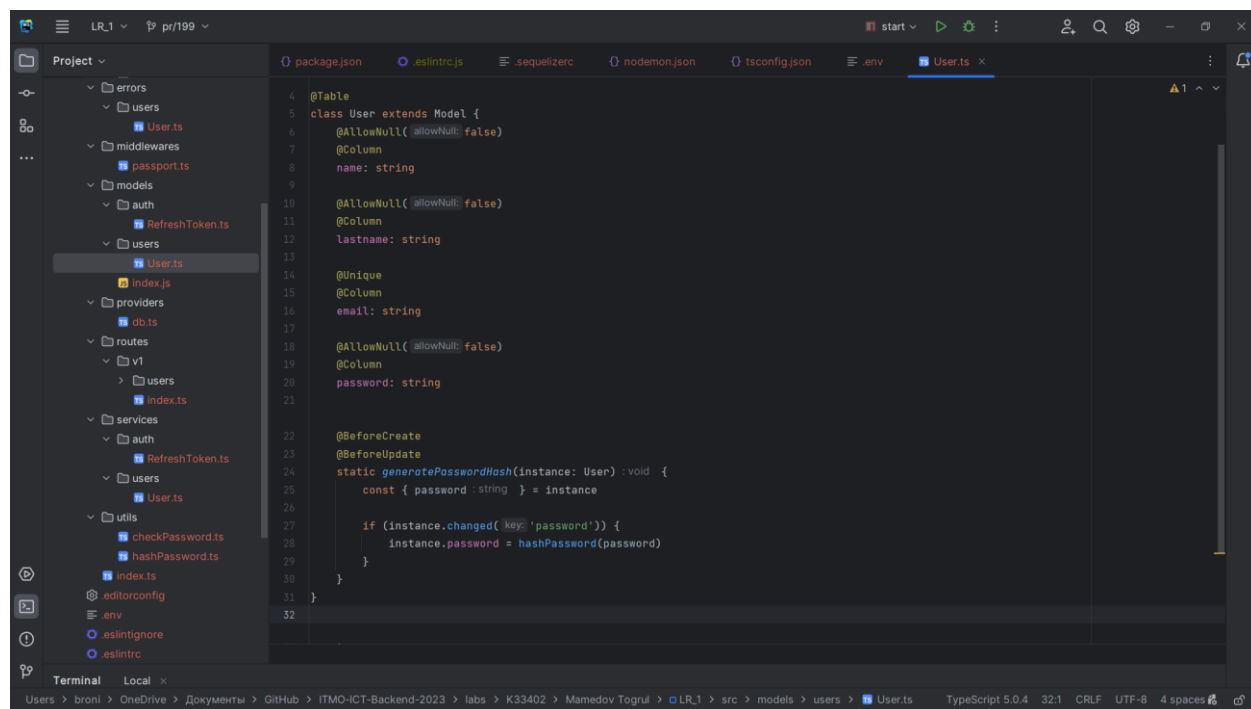
services – службы, которые содержат запросы к базе данных и возвращают объекты или выдают ошибки;

utils – вспомогательные файлы, которые используются в приложении;

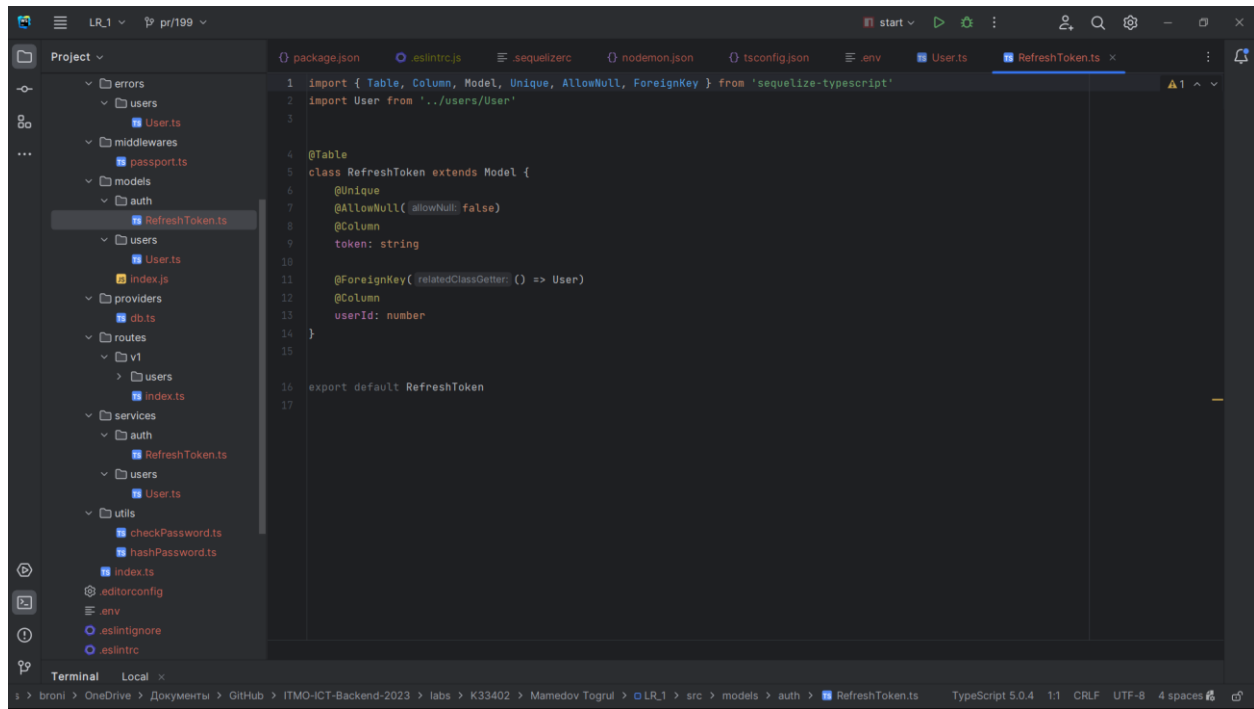
middlewares - содержит аутентификацию с использованием passport.ts.

Модели:

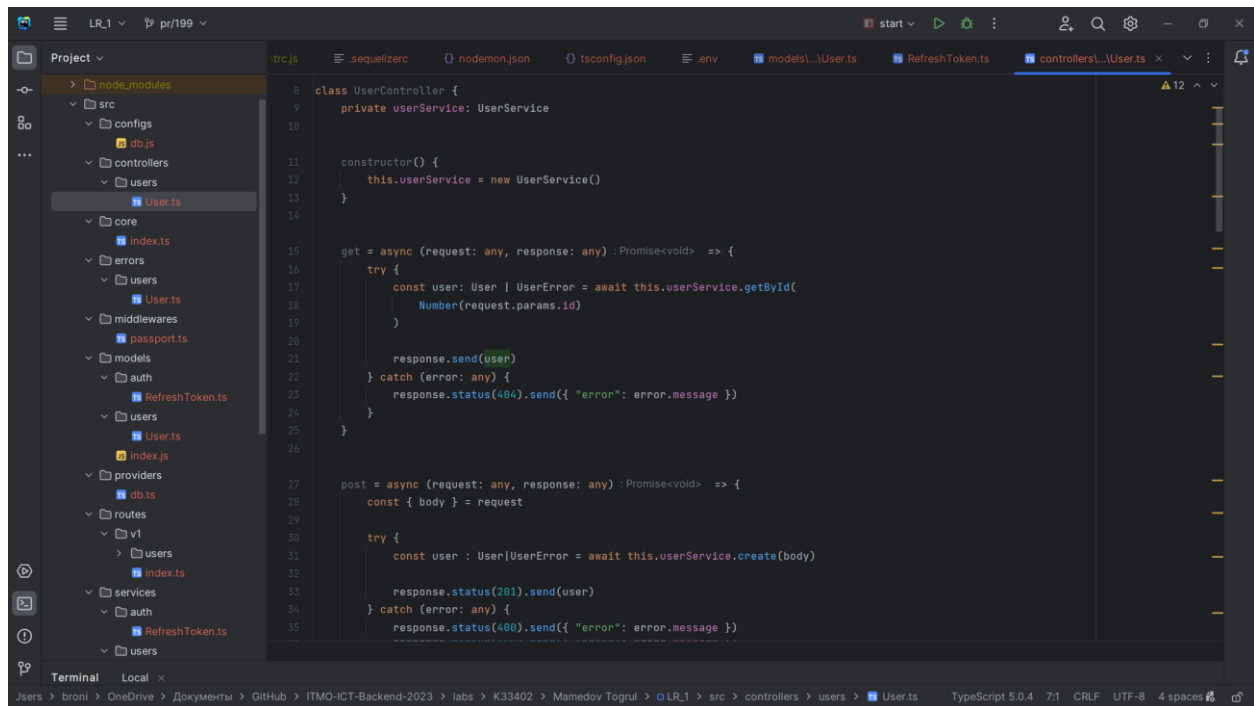
User.ts – модель пользователя



RefreshToken.ts - модель хранения токенов

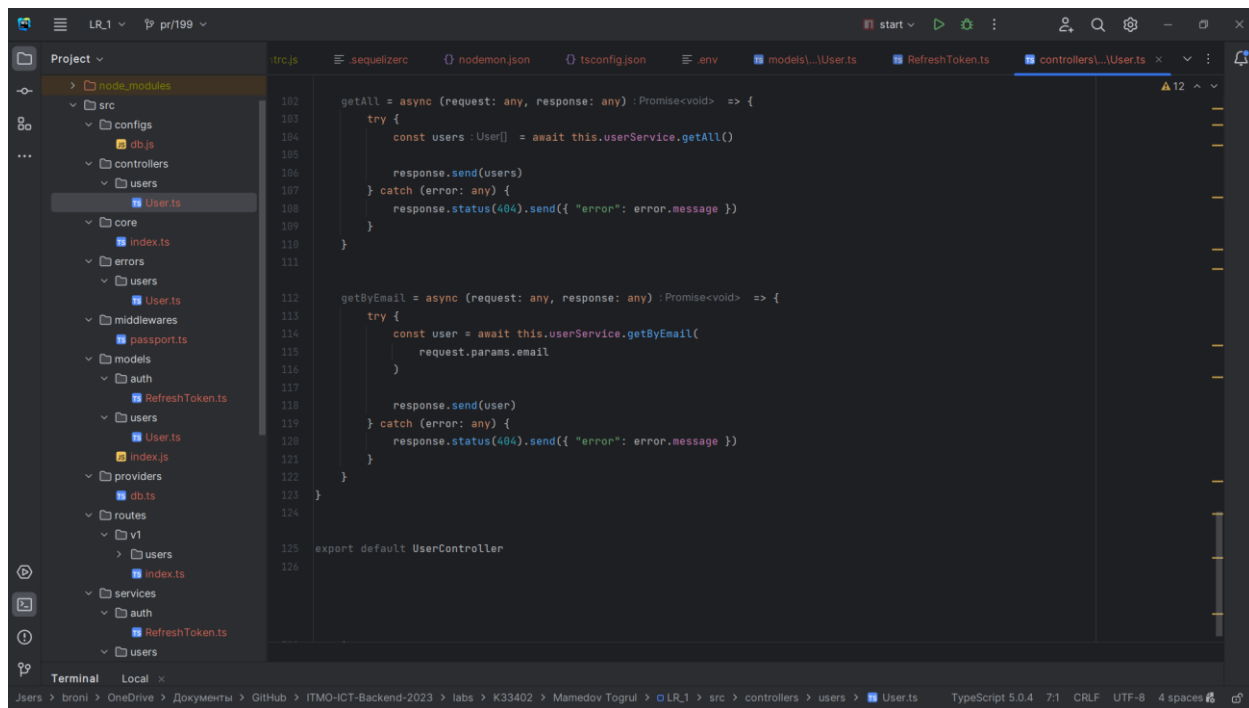


Контроллеры:




```
39 me = async (request: any, response: any) : Promise<void> => {
40     response.send(request.user)
41 }
42
43 auth = async (request: any, response: any) : Promise<void> => {
44     const { body } = request
45
46     const { email, password } = body
47
48     try {
49         const { user, checkPassword } = await this.userService.checkPassword(email, password)
50
51         if (checkPassword) {
52             const payload : (id: any) = { id: user.id }
53
54             const accessToken : string = jwt.sign(payload, jwtOptions.secretOrKey)
55
56             const refreshTokenService : RefreshTokenService = new RefreshTokenService(user)
57
58             const refreshToken : string = 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
```

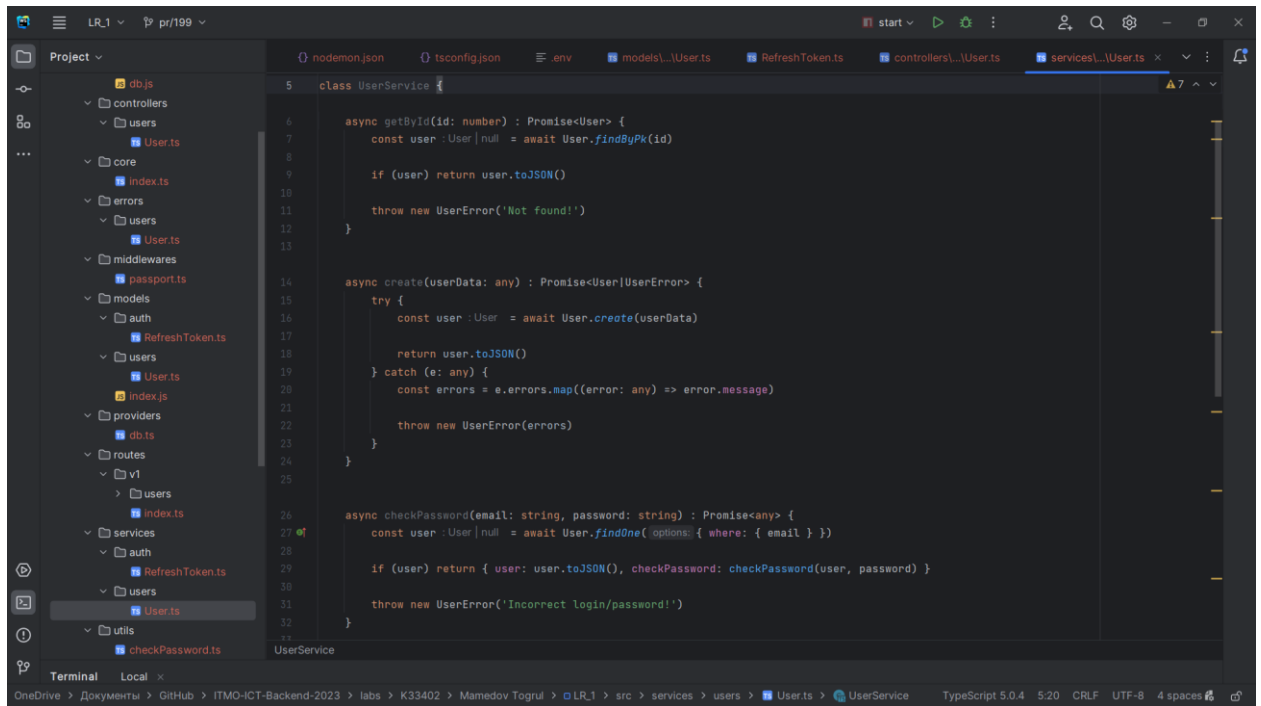
```
69 refreshToken = async (request: any, response: any) : Promise<void> => {
70     const { body } = request
71
72     const { refreshToken } = body
73
74     const refreshTokenService : RefreshTokenService = new RefreshTokenService()
75
76     try {
77         const { userId : number | null, isExpired : boolean } = await refreshTokenService
78             .isRefreshTokenExpired(refreshToken)
79
80         if (!isExpired && userId) {
81             const user : User = await this.userService.getById(userId)
82
83             const payload : (id: any) = { id: user.id }
84
85             const accessToken : string = jwt.sign(payload, jwtOptions.secretOrKey)
86
87             // tslint:disable-next-line:no-shadowed-variable
88             const refreshTokenService : RefreshTokenService = new RefreshTokenService(user)
89
90             // tslint:disable-next-line:no-shadowed-variable
91             const refreshToken : string = await refreshTokenService.generateRefreshToken()
92
93             response.send({ accessToken, refreshToken })
94         } else {
95             throw new Error('Invalid credentials')
96         }
97     } catch (e) {
98         response.status(401).send({ 'error': 'Invalid credentials' })
99     }
100 }
```



Методы класса UserController:

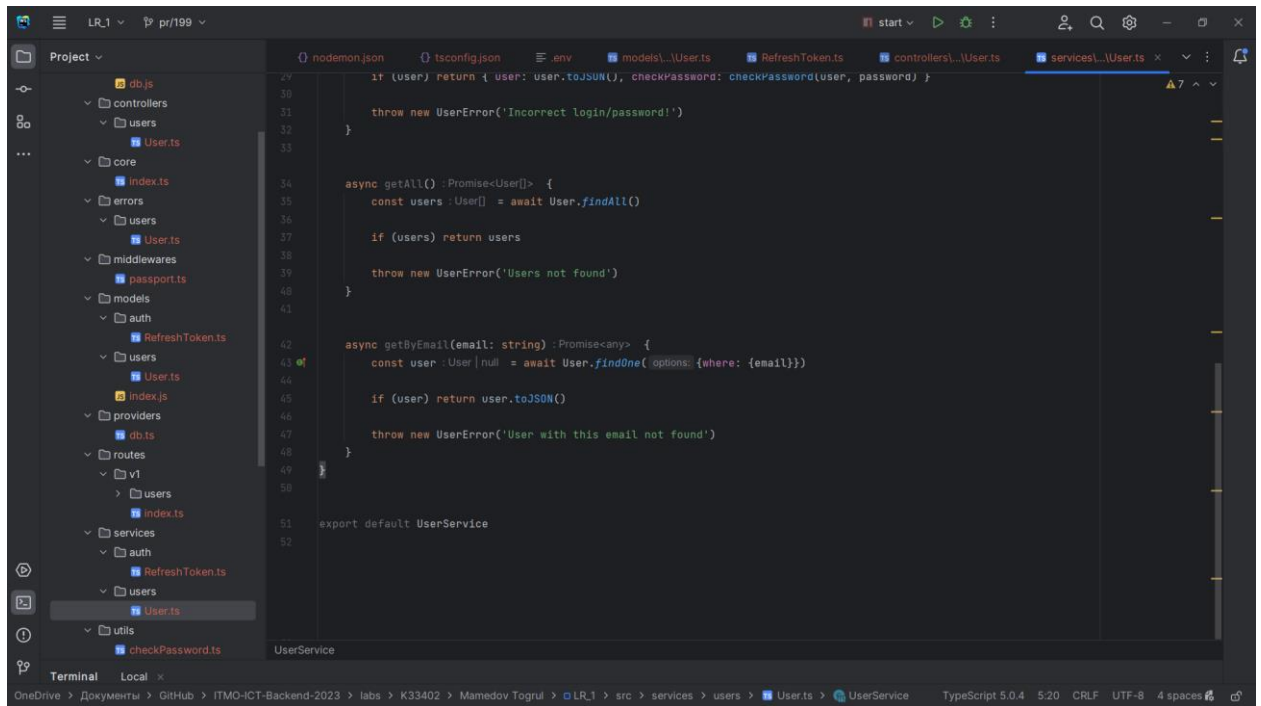
- get: находит пользователя по id;
- post: создание нового пользователя;
- me: возвращает данные о пользователе;
- auth: генерирует новый токен доступа и токен обновления, если пользователь залогинился;
- refreshToken: генерирует новый JWT токен;
- getAll: получает информацию по всем пользователям;
-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

Services:



The screenshot shows a VS Code editor window with a project named 'LR_1' and a pull request 'pr/199'. The file explorer on the left shows a directory structure with files like 'db.js', 'controllers', 'core', 'errors', 'middlewares', 'models', 'providers', 'routes', 'services', 'utils', and 'checkPassword.ts'. The main editor displays the 'UserService' class in 'services\...\User.ts'. The class has three methods: 'getById', 'create', and 'checkPassword'. The 'getById' method takes an 'id' and returns a 'Promise<User>'. The 'create' method takes 'userData' and returns a 'Promise<User|UserError>'. The 'checkPassword' method takes 'email' and 'password' and returns a 'Promise<any>'. The class is exported as 'UserService'.

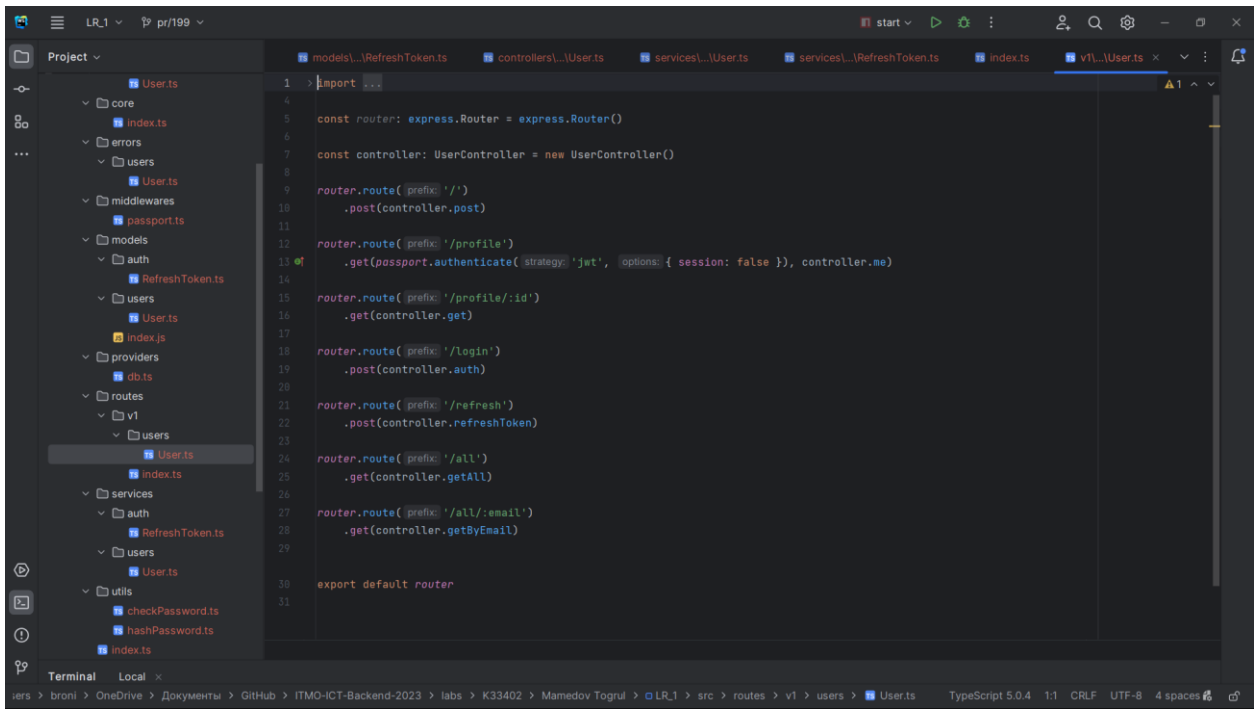
```
5 class UserService {
6
7   async getById(id: number) : Promise<User> {
8     const user : User | null = await User.findByIdPk(id)
9
10    if (user) return user.toJSON()
11
12    throw new UserError('Not found!')
13  }
14
15  async create(userData: any) : Promise<User|UserError> {
16    try {
17      const user : User = await User.create(userData)
18
19      return user.toJSON()
20    } catch (e: any) {
21      const errors = e.errors.map((error: any) => error.message)
22
23      throw new UserError(errors)
24    }
25  }
26
27  async checkPassword(email: string, password: string) : Promise<any> {
28    const user : User | null = await User.findOne({ options: { where: { email } }})
29
30    if (user) return { user: user.toJSON(), checkPassword: checkPassword(user, password) }
31
32    throw new UserError('Incorrect login/password!')
33  }
34}
35
36export default UserService
```



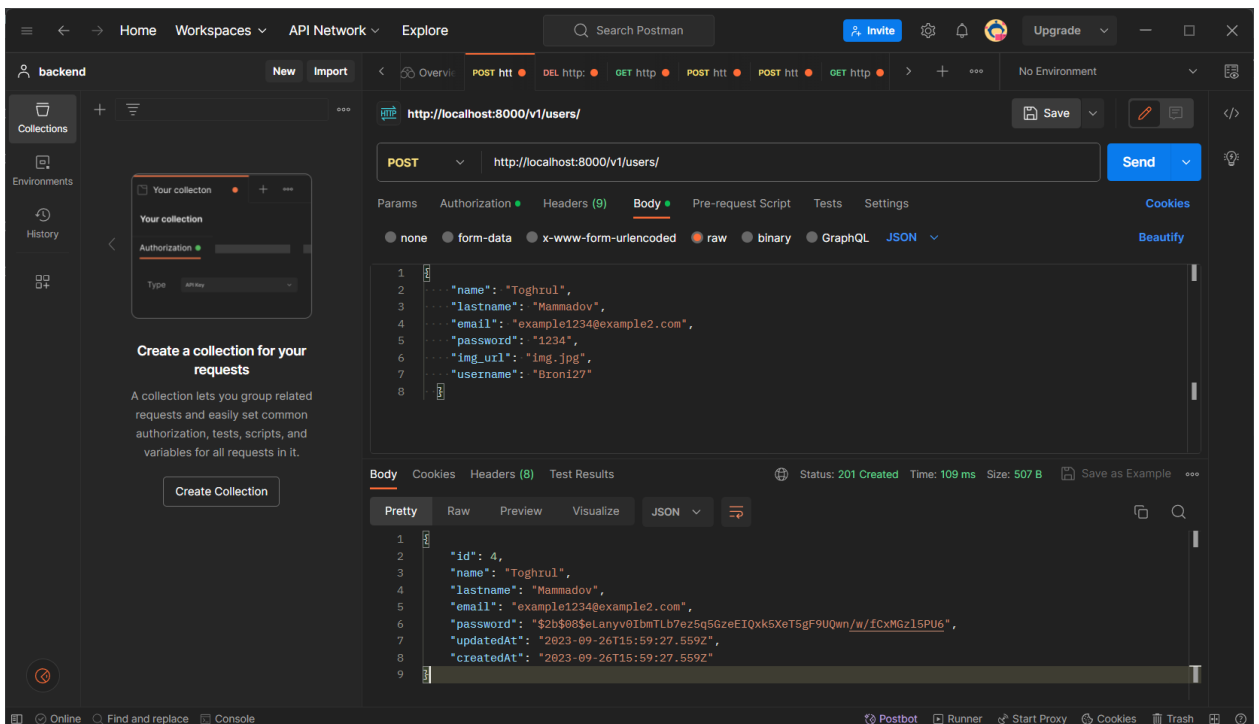
The screenshot shows a VS Code editor window with a project named 'LR_1' and a pull request 'pr/199'. The file explorer on the left shows a directory structure with files like 'db.js', 'controllers', 'core', 'errors', 'middlewares', 'models', 'providers', 'routes', 'services', 'utils', and 'checkPassword.ts'. The main editor displays the 'UserService' class in 'services\...\User.ts'. The class has three methods: 'getAll', 'getByEmail', and 'export default'. The 'getAll' method returns a 'Promise<User[]>'. The 'getByEmail' method takes an 'email' and returns a 'Promise<any>'. The class is exported as 'UserService'.

```
29
30
31 if (user) return { user: user.toJSON(), checkPassword: checkPassword(user, password) }
32
33 throw new UserError('Incorrect login/password!')
34 }
35
36 async getAll() : Promise<User[]> {
37   const users : User[] = await User.findAll()
38
39   if (users) return users
40
41   throw new UserError('Users not found')
42 }
43
44 async getByEmail(email: string) : Promise<any> {
45   const user : User | null = await User.findOne({ options: { where: { email } }})
46
47   if (user) return user.toJSON()
48
49   throw new UserError('User with this email not found')
50 }
51
52 export default UserService
```

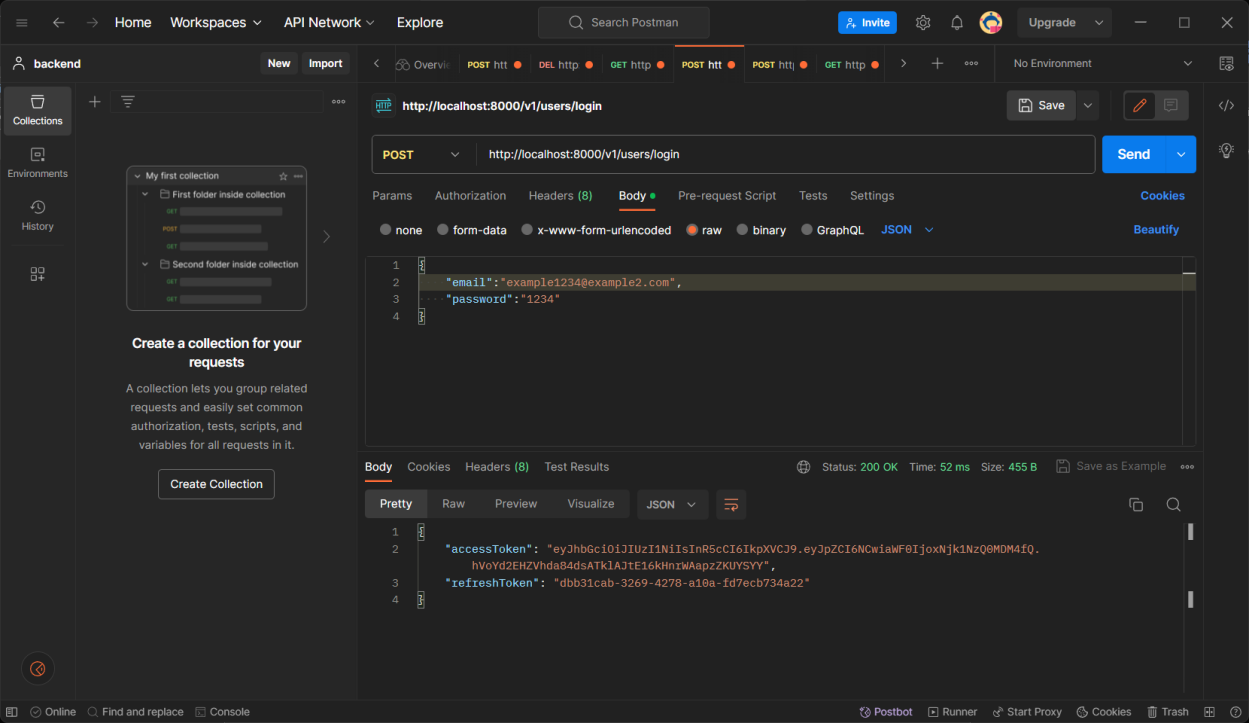
RefreshToken.ts



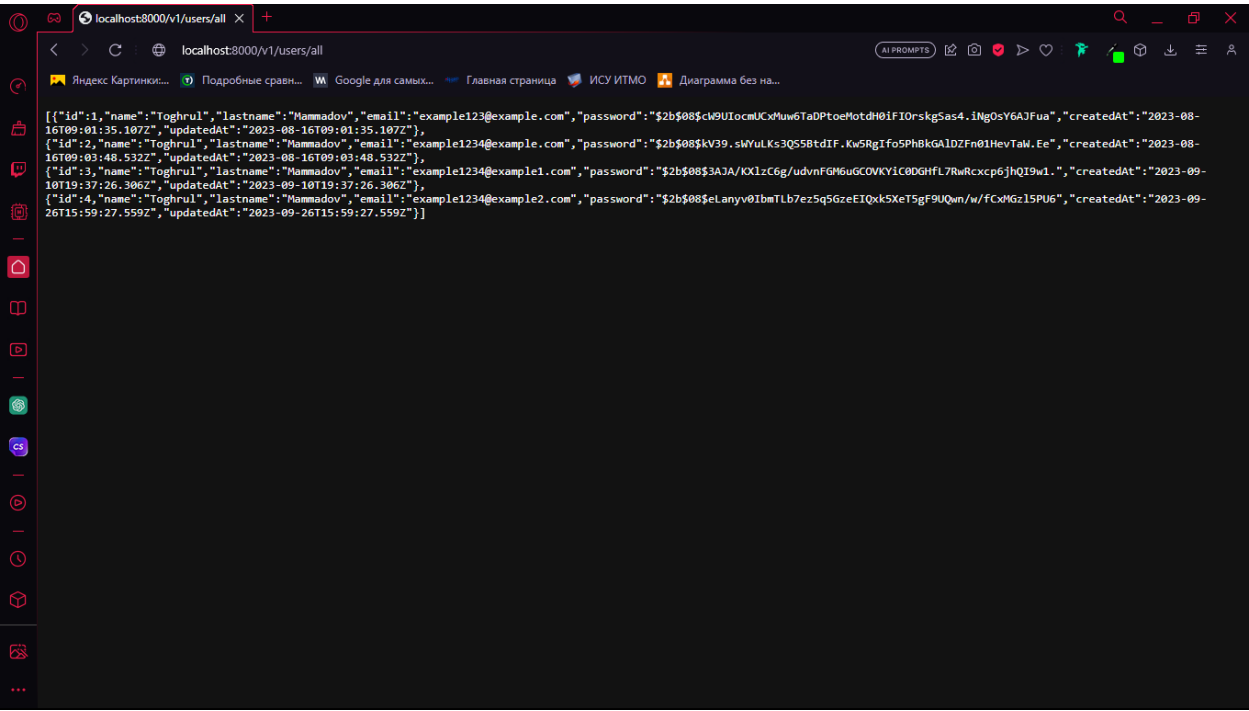
Регистрация пользователя:



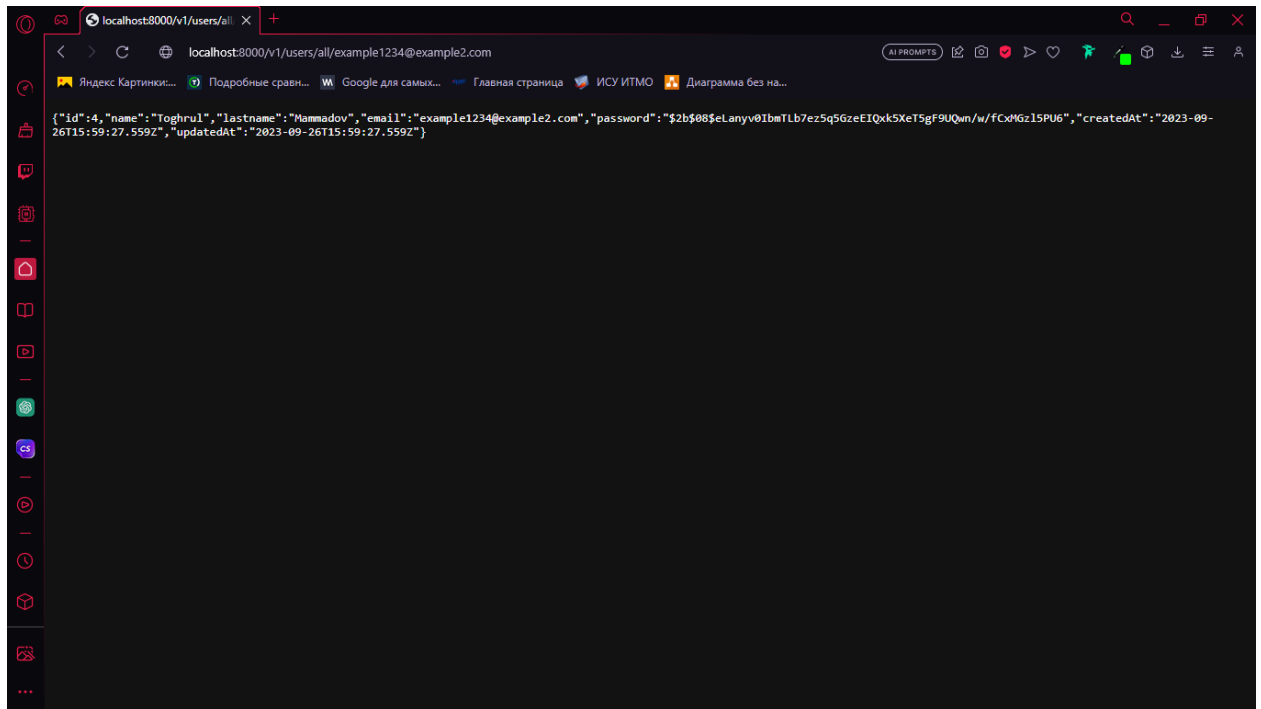
Авторизация:



Получение всех пользователей:



Получение пользователя по его почте:



Вывод

Во время выполнения лабораторной работы был создан собственный стандартный шаблон (boilerplate) с продуманной структурой и реализовал систему аутентификации с использованием Express, Sequelize и TypeScript.