Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

ОТЧЕТ

по Лабораторной работе № 2

по теме: «Express + TypeORM + TypeScript»
по дисциплине: Бек-энд разработка

Специальность:
09.03.03 Мобильные и сетевые технологии

Проверил:
Добряков Д. И. _____
Дата: «__» _____ 202__г.
Оценка _____

Выполнил:
студент группы K33401
Чернов Е. К.

Санкт-Петербург
2023

ЦЕЛЬ РАБОТЫ

Получить практические навыки по созданию *backend*-а, используя *Express*, *TypeORM* и *TypeScript*.

ВЫПОЛНЕНИЕ

1 Структура проекта

Проект имеет следующую структуру (рисунок 1):

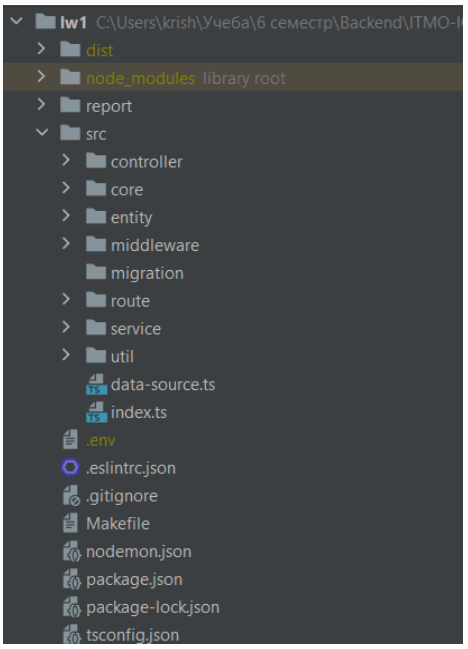


Рисунок 1 - Структура проекта

2 Подключение БД

Подключение к базе данных *«PostgreSQL»* (рисунок 2):

```
data-source.ts
import * as dotenv from "dotenv";
import {DataSource} from "typeorm";

export const AppDataSource = new DataSource( options: {
    type: "postgres",
    host: String(process.env.DB_HOST),
    port: parseInt(process.env.DB_PORT),
    username: String(process.env.DB_USERNAME),
    password: String(process.env.DB_PASSWORD),
    database: String(process.env.DB_DATABASE),
    synchronize: true,
    logging: false,
    entities: ["dist/entity/*.js", "src/entity/*.ts"],
    migrations: ["src/migration/*.js"]
})
```

Рисунок 2 - Файл *«data-source.ts»*

3 Роуты

Роуты для пользователя (рисунок 3):

UserRouter.ts

```ts
import ...

// Create router and controller
const userRouter: express.Router = express.Router()
const userController: UserController = new UserController()

// User routes
userRouter.route( prefix: '/list')
    .get(userController.getAllUsers)

userRouter.route( prefix: '/specific')
    .get(userController.getUser)

userRouter.route( prefix: '/register')
    .post(userController.postSignupUser)

userRouter.route( prefix: '/login')
    .post(userController.postLoginUser)

userRouter.route( prefix: '/update')
    .post(userController.updateUser)

userRouter.route( prefix: '/delete')
    .delete(userController.deleteUser)

export default userRouter
```

Рисунок 3 - Файл *«UserRouter.ts»*

Роуты для обновления токена (рисунок 4):

TokenRouter.ts

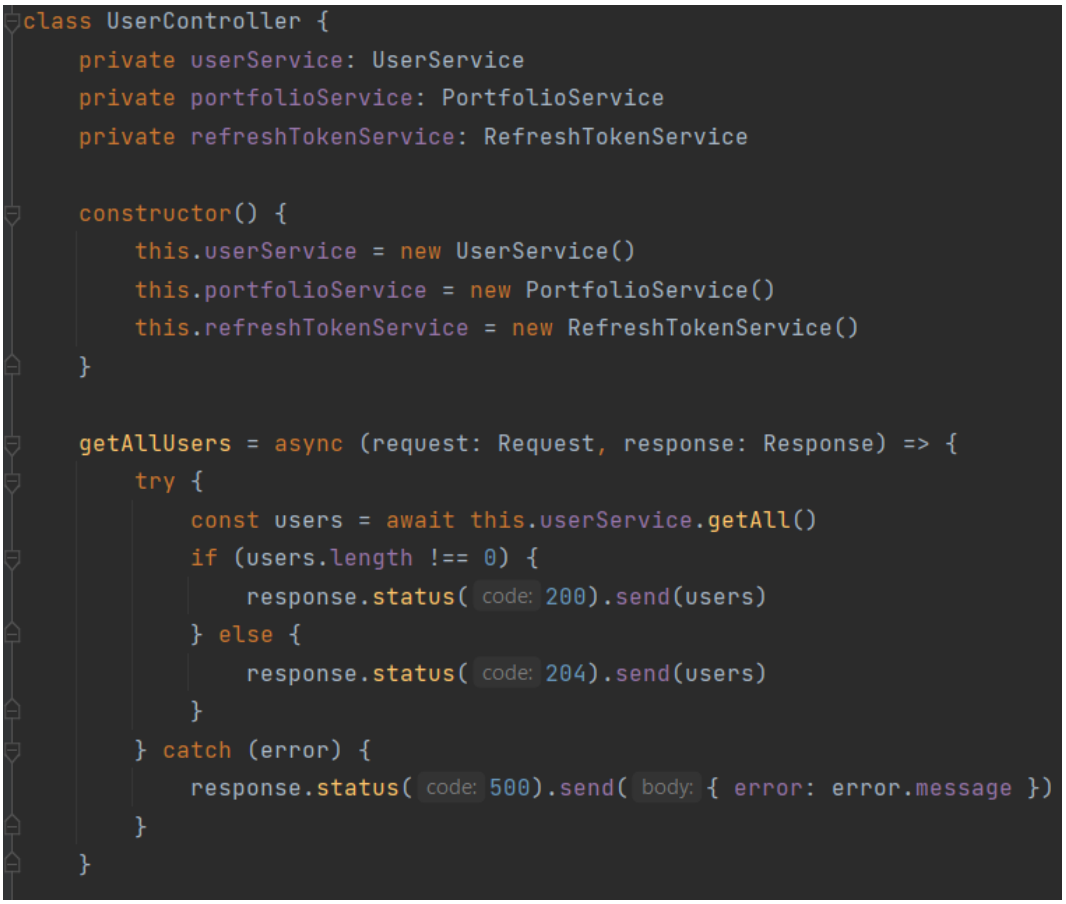
```ts
import ...

// Create router and controller
const tokenRouter: express.Router = express.Router()
const tokenController: TokenController = new TokenController()

// Token routes
tokenRouter.route( prefix: '/update')
    .get(tokenController.update)

export default tokenRouter
```

Рисунок 4 - Файл *«TokenRouter.ts»*

Остальные роуты для портфолио и монет прописаны по аналогии.

4 Контроллеры

Пример контроллера для пользователя:



```
getUser = async (request: Request, response: Response) => {
    try {
        const accessToken = request.headers.authorization.split( separator: " ")[1]
        const decoded = checkToken(accessToken)
        if (decoded.isExpired) {
            return response.status( code: 401).send( body: "Access token was expired")
        }
        const userId = decoded.payload.sub.toString()
        const user = await this.userService.get(userId)
        response.status( code: 200).send(user)
    } catch (error) {
        response.status( code: 500).send( body: { error: error.message })
    }
}

postSignupUser = async (request: Request, response: Response) => {
    try {
        const { body } = request
        const user = await this.userService.create(body)
        const tokens = generateTokens(user.id)
        await this.refreshTokenService.create(user, tokens.refreshToken)
        console.log("Create access and refresh tokens")
        response.status( code: 201).send(tokens)
    } catch (error) {
        response.status( code: 500).send( body: { error: error.message })
    }
}
```

```
postLoginUser = async (request: Request, response: Response) => {
    try {
        const { body } = request
        const { email, password } = body
        const user = await this.userService.login(email, password)
        const refreshToken = await this.refreshTokenService.get(user)
        const { isExpired } = checkToken(refreshToken)
        if (isExpired) {
            const tokens = generateTokens(user.id)
            await this.refreshTokenService.update(user, tokens.refreshToken)
            console.log("Update refresh token and create access token")
            return response.status( code: 200).send(tokens)
        }
        const tokens = generateTokens(user.id, refreshToken)
        console.log("Create access token")
        response.status( code: 200).send(tokens)
    } catch (error) {
        response.status( code: 500).send( body: { error: error.message })
    }
}
```

```
updateUser = async (request: Request, response: Response) => {
    try {
        const { body } = request
        const accessToken = request.headers.authorization.split( separator: " ")[1]
        const decoded = checkToken(accessToken)
        if (decoded.isExpired) {
            return response.status( code: 401).send( body: "Access token was expired")
        }
        const userId = decoded.payload.sub.toString()
        await this.userService.update(userId, body)
        response.status( code: 200).send( body: "Success")
    } catch (error) {
        response.status( code: 500).send( body: { error: error.message })
    }
}

deleteUser = async (request: Request, response: Response) => {
    try {
        const accessToken = request.headers.authorization.split( separator: " ")[1]
        const decoded = checkToken(accessToken)
        if (decoded.isExpired) {
            return response.status( code: 401).send( body: "Access token was expired")
        }
        const userId = decoded.payload.sub.toString()
        await this.userService.delete(userId)
        response.status( code: 200).send( body: "Success")
    } catch (error) {
        response.status( code: 500).send( body: { error: error.message })
    }
}
```

5 Сервисы

Сервис для пользователя:

```
class UserService {
    private userRepository: Repository<User>

    constructor() {
        this.userRepository = AppDataSource.getRepository(User)
    }

    async getAll() {
        return await this.userRepository.find()
    }

    async get(userId: string) {
        return await this.userRepository.findOneBy( where: {
                id: userId
        })
    }

    async create(userData: object) {
        const user = await this.userRepository.create(userData)
        return await this.userRepository.save(user)
    }
```

```
    async login(email, password) {
        const user = await this.userRepository.findOneBy( where: {
                email: email
        })
        const isMatch = checkPassword(password, user.password)
        if (isMatch) {
            return user
        }
        throw new Error("Password isn't correct")
    }

    async update(userId: string, userData: object) {
        return await this.userRepository.update( criteria: {id: userId}, userData)
    }

    async delete(userId) {
        return await this.userRepository.delete(userId)
    }
}

export default UserService
```

6 Средний слой

Функция для генерации токенов:

generateTokens.ts

```ts
import ...

export default (userId: string, refreshToken="") => {
    if (!refreshToken) {
        refreshToken = jwt.sign(
            payload: {sub: userId, iss: "cryptocoin"},
            process.env.SECRET_KEY,
            options: {algorithm: "HS512", expiresIn: "7d"}
        )
    }
    return {
        accessToken: jwt.sign(
            payload: {sub: userId, iss: "cryptocoin"},
            process.env.SECRET_KEY,
            options: {algorithm: "HS512", expiresIn: "10m"}
        ),
        refreshToken: refreshToken
    }
}
```

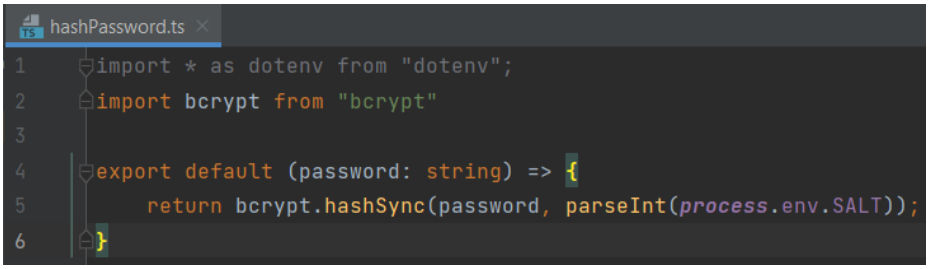
7 Утилиты

checkPassword.ts

```ts
import bcrypt from "bcrypt"

export default (checkPassword: any, password: string) => {
    return bcrypt.compareSync(checkPassword, password);
}
```

checkToken.ts

```ts
import ...

export default (token: string) => {
    try {
        return { payload: jwt.verify(token, process.env.SECRET_KEY), isExpired: false }
    } catch (error) {
        if ((error as Error).name == "TokenExpiredError") {
            return { payload: jwt.decode(token), isExpired: true }
        }
        throw error
    }
}
```



ВЫВОД

В ходе работы получили практические навыки по созданию *backend*-а, используя *Express*, *TypeORM* и *TypeScript*.