САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнила:

Мищенко Виолетта

Группа К33402

Проверил:
Добряков Д.И.

Санкт-Петербург

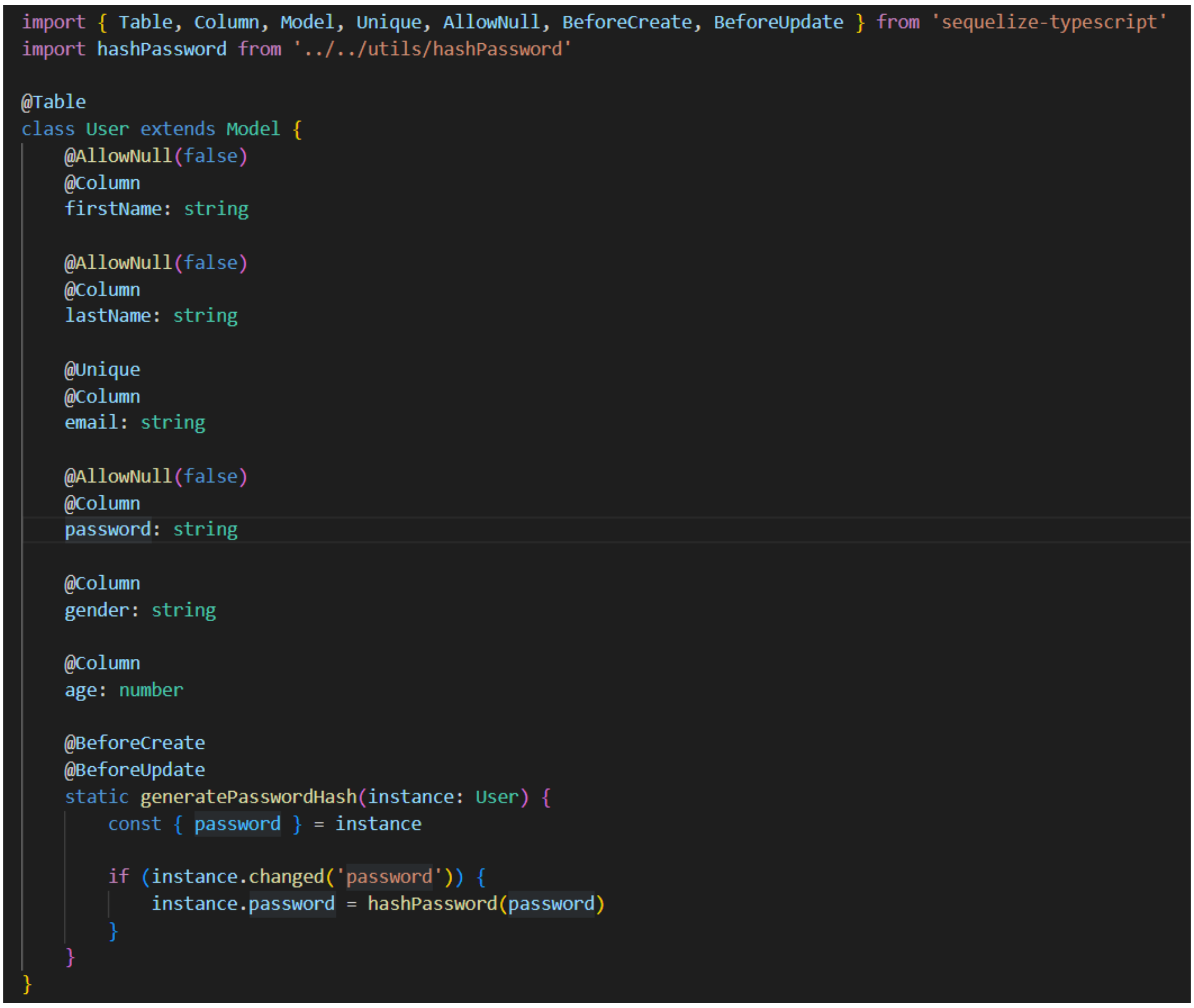
2022 г.

Цель работы: нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

Должно быть явное разделение на:

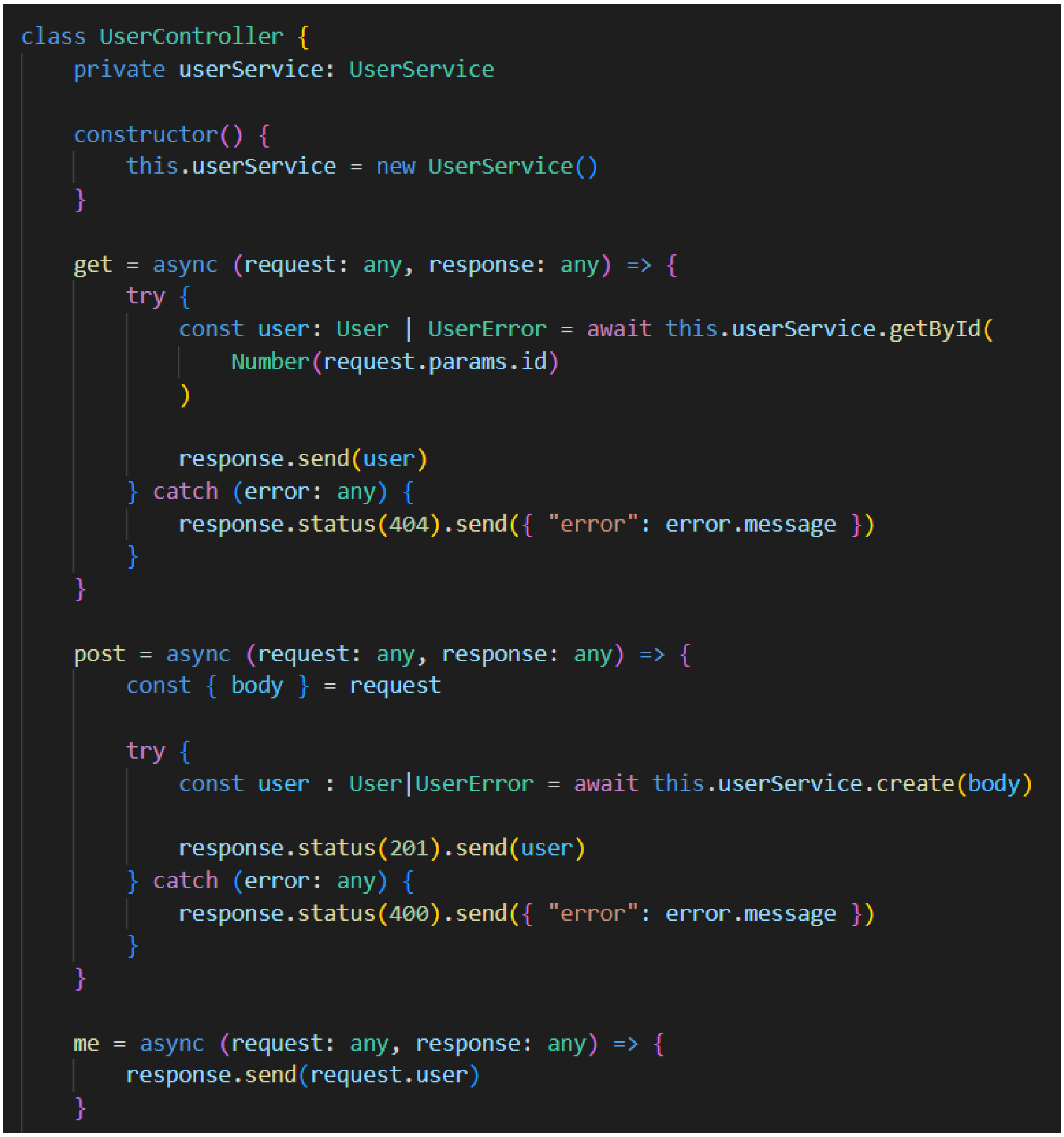
- модели
- контроллеры
- роуты
- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Реализованная модель пользователями с полями: firstname, lastname, email, password, gender, age

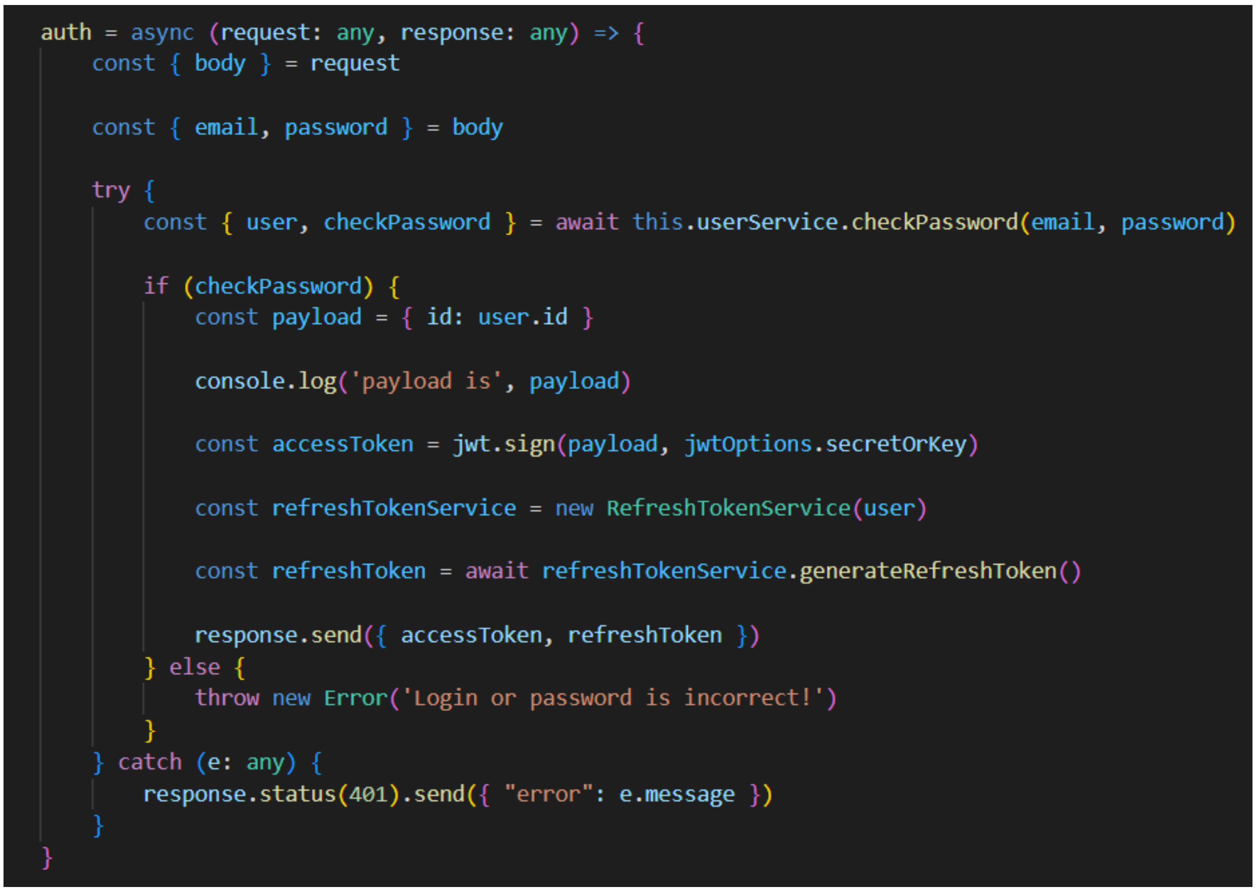


```
import { Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpdate } from 'sequelize-typescript'
import hashPassword from '../../utils/hashPassword'

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

    @AllowNull(false)
    @Column
    lastName: string

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string

    @Column
    gender: string

    @Column
    age: number

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

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

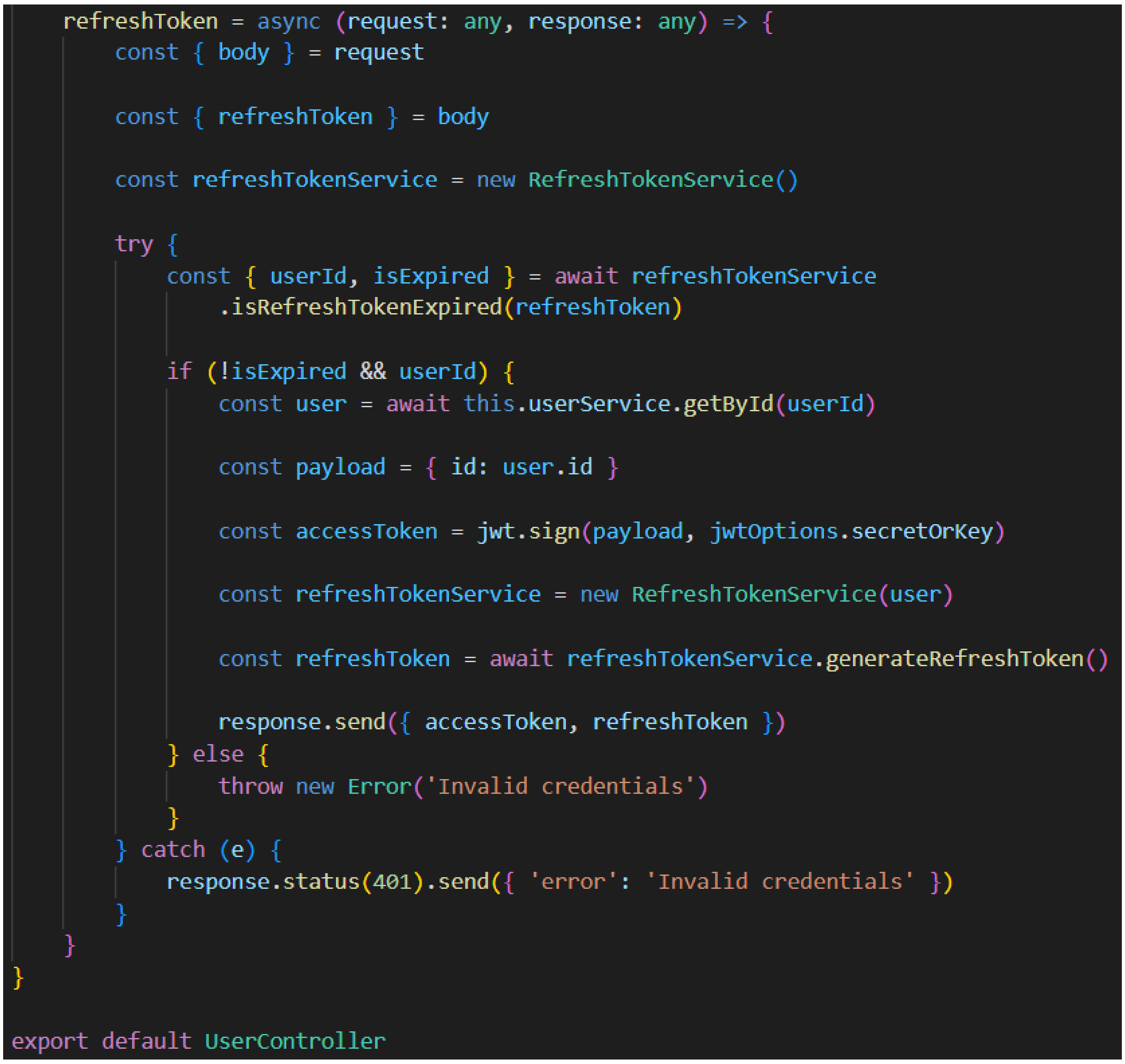
Контроллеры:



```
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 })
        }
    }

    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)
    }
```

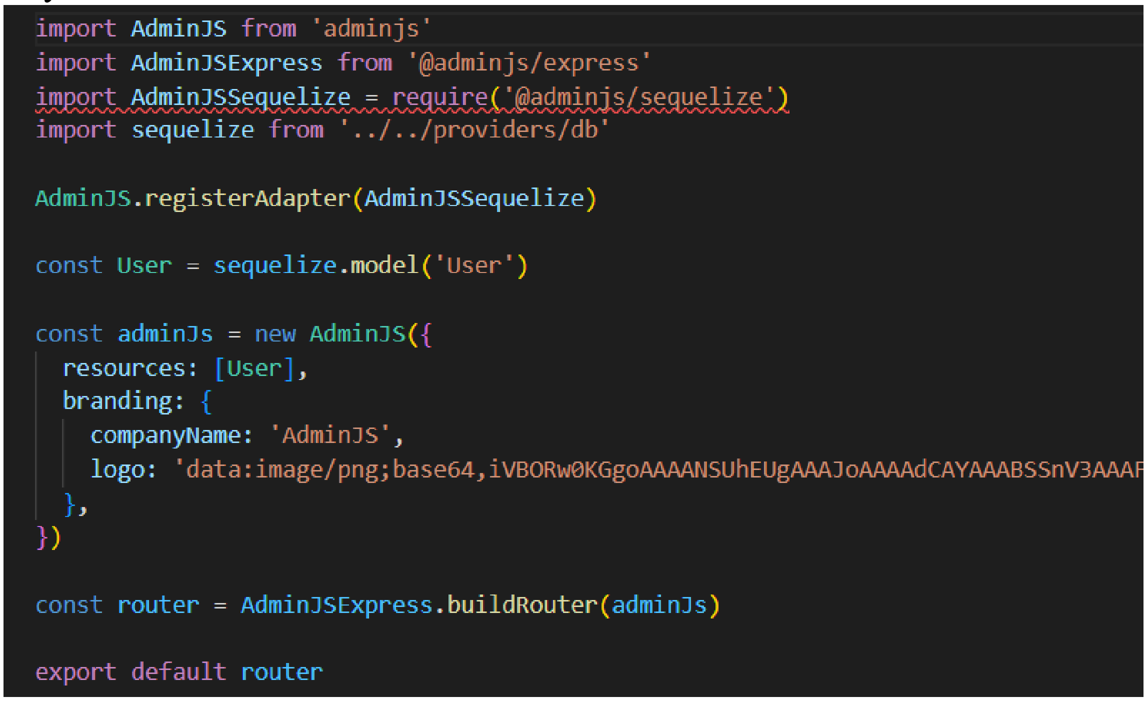


```
auth = async (request: any, response: any) => {
    const { body } = request

    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 })
    }
}
```

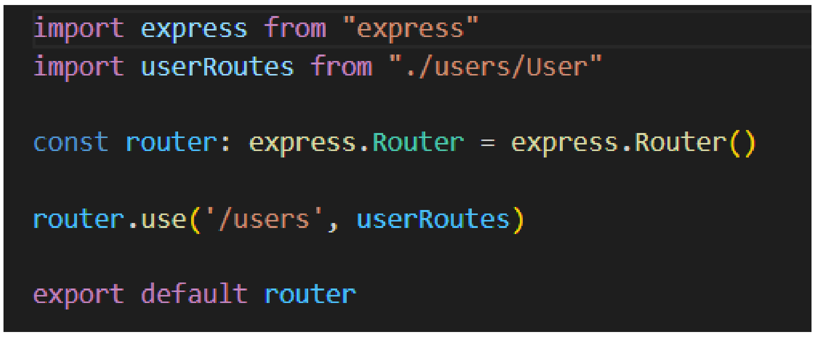


```
    refreshToken = async (request: any, response: any) => {
        const { body } = request

        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' })
        }
    }
}

export default UserController
```

Роуты:

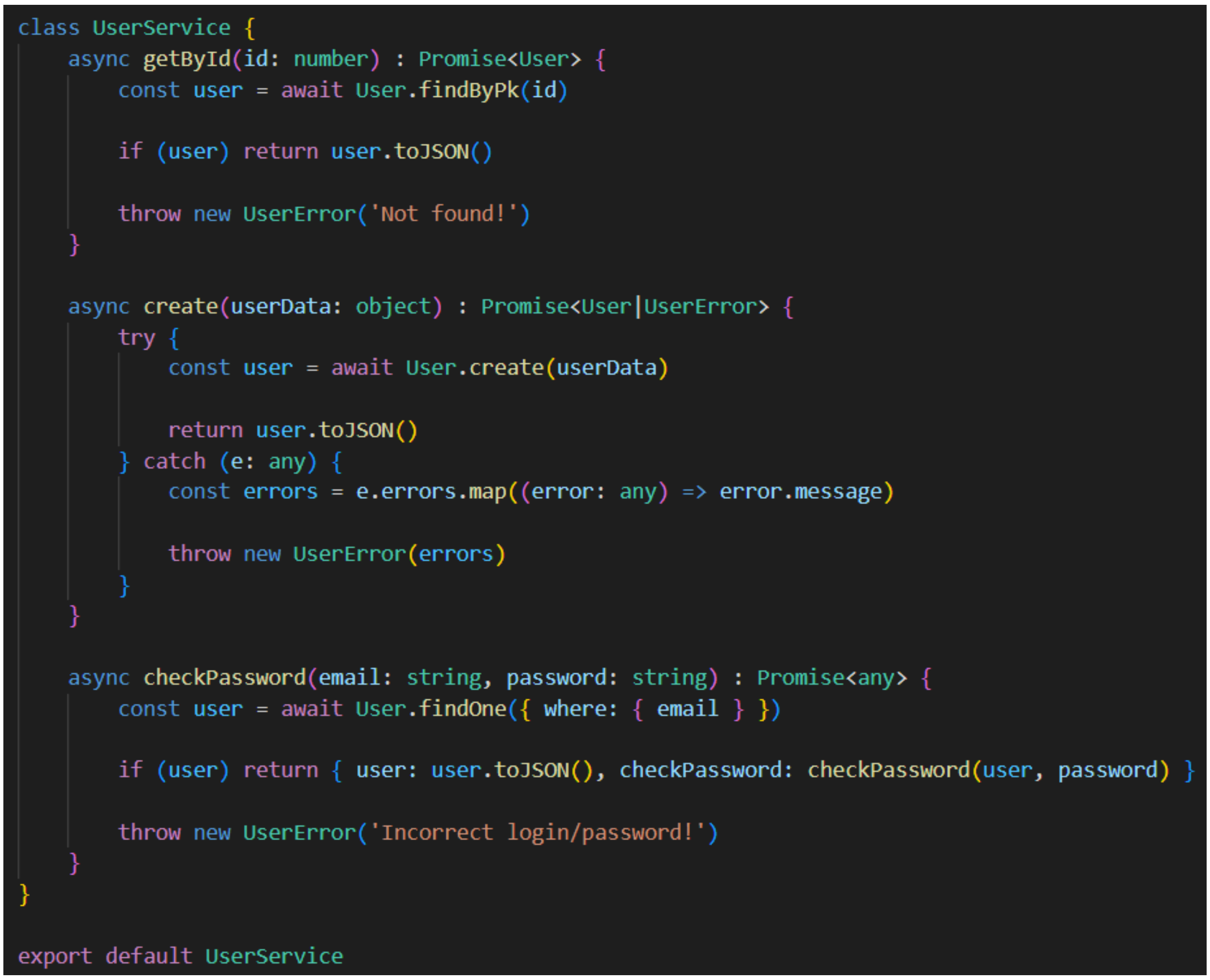


```
import AdminJS from 'adminjs'
import AdminJSExpress from '@adminjs/express'
import AdminJSSequelize = require('@adminjs/sequelize')
import sequelize from '../../providers/db'

AdminJS.registerAdapter(AdminJSSequelize)

const User = sequelize.model('User')

const adminJs = new AdminJS({
  resources: [User],
  branding: {
    companyName: 'AdminJS',
    logo: 'data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAJoAAAAdCAYAAABSSnV3AAAF
  },
})

const router = AdminJSExpress.buildRouter(adminJs)

export default router
```

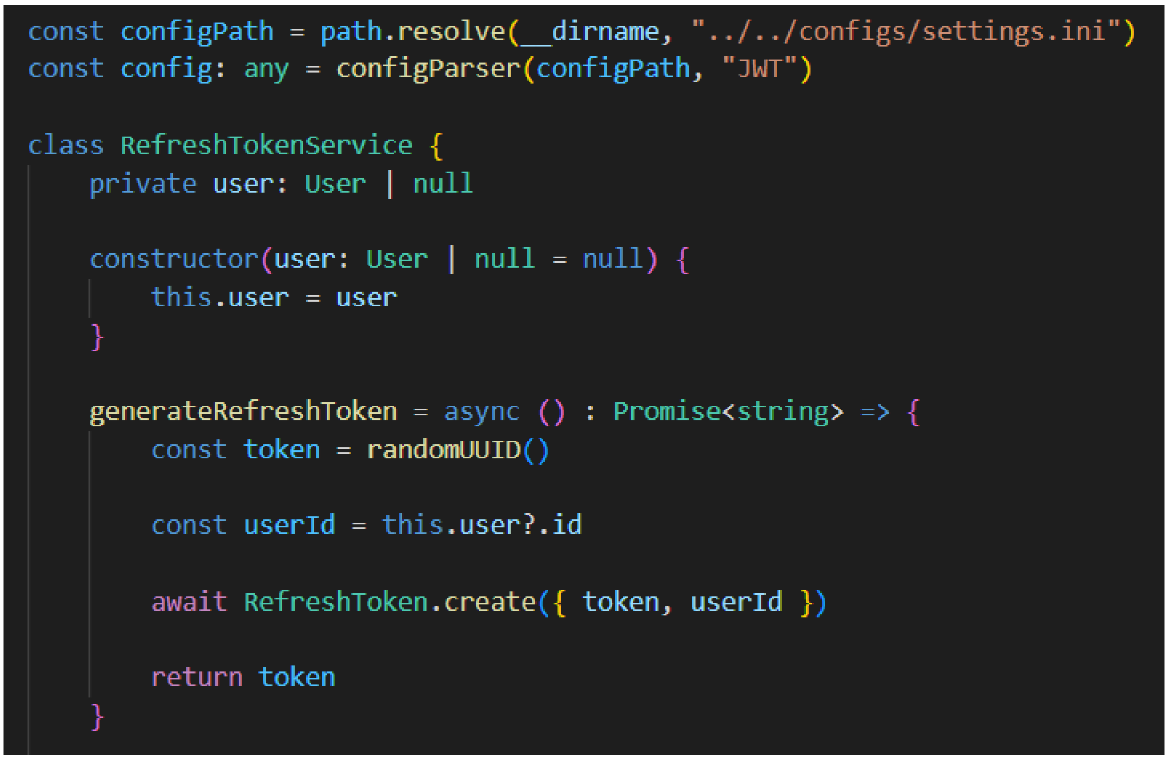


```
import express from "express"
import userRoutes from "./users/User"

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

router.use('/users', userRoutes)

export default router
```

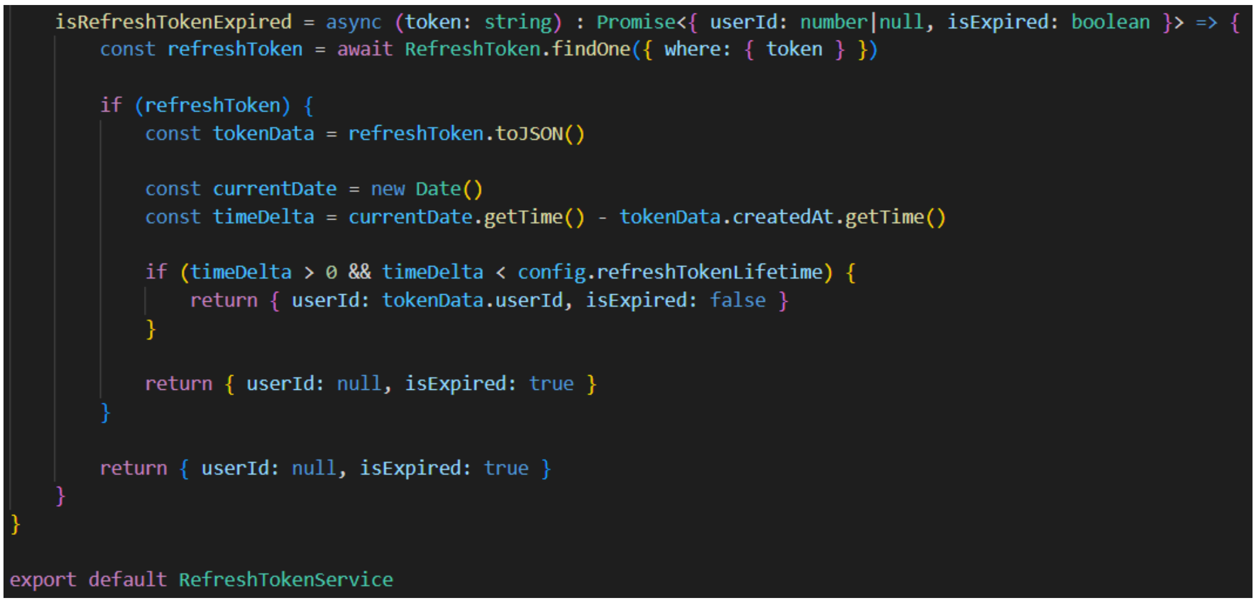
Сервисы:



```
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: object) : 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!')
    }
}

export default UserService
```



```
const configPath = path.resolve(__dirname, "../../configs/settings.ini")
const config: any = configParser(configPath, "JWT")

class RefreshTokenService {
    private user: User | null

    constructor(user: User | null = null) {
        this.user = user
    }

    generateRefreshToken = async () : Promise<string> => {
        const token = randomUUID()

        const userId = this.user?.id

        await RefreshToken.create({ token, userId })

        return token
    }
```



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

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

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

            if (timeDelta > 0 && timeDelta < config.refreshTokenLifetime) {
                return { userId: tokenData.userId, isExpired: false }
            }

            return { userId: null, isExpired: true }
        }

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

export default RefreshTokenService
```

Вывод: в ходе данной лабораторной работы были изучены express, sequelize, typescript и реализован boilerplate (шаблон) с нужными разделениями на модели, контроллеры, роуты и сервисы.