САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнила:
Барышева З. А.
Группа:
K33412

Проверил:
Добряков Д. И.

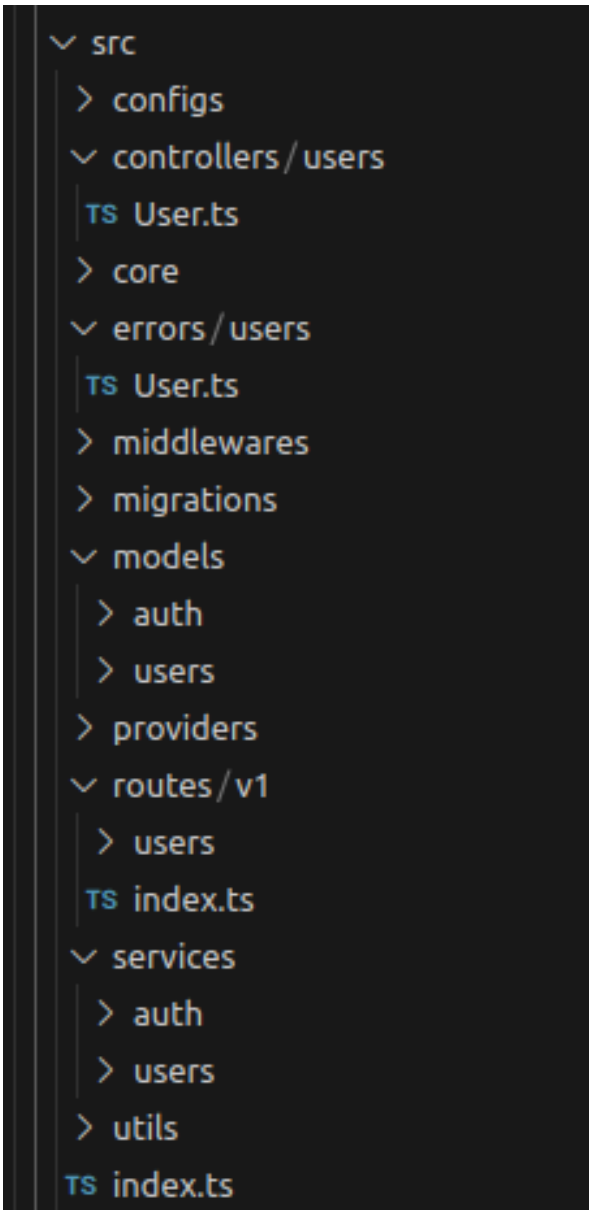
Санкт-Петербург

2023 г.

Задача:
Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.
Должно быть явное разделение на:

1. модели
2. контроллеры
3. роуты
4. сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы
Структура:



Модели:

1. Модель пользователя User

```
@Table
class User extends Model {
    @Column
    username: 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)
        }
    }
}
```

2. RefreshToken

```
@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull(false)
    @Column
    token: string

    @ForeignKey(() => User)
    @Column
    userId: number
}
```

Контроллеры:

1. User

   Пример некоторых функций:

```
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 })
        }
    }
```

Роуты:

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

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

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

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

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

Сервисы:

1. User

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

        if (user) return user.toJSON()

        throw new UserError('user 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('smth incorrect')
    }
}
```

2. RefreshToken

   Пример функции:

```
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
    }
```

Вывод

В ходе работы был реализован boilerplate, который будет использоваться в дальнейших работах.