САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1: Написание своего boilerplate

Выполнила:

Никифорова Кюннэй

Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

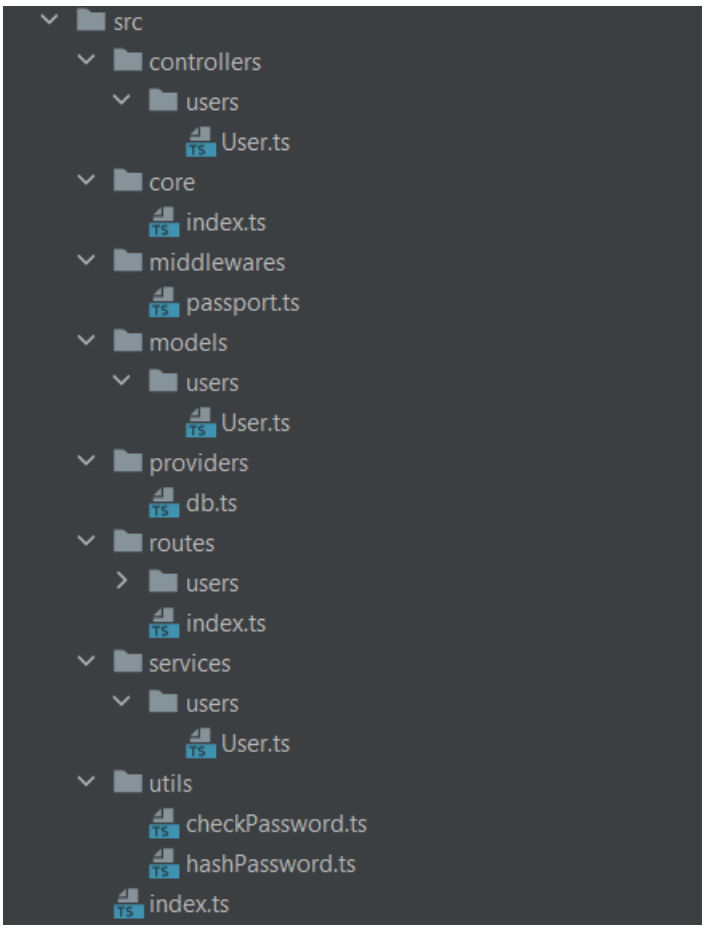
Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

Должно быть явное разделение на:

- Модели;
- Контроллеры;
- Роуты;
- Сервисы для работы с моделями (реализуем паттерн “репозиторий”).

Ход работы

1) Структура проекта:



2) *models/User.ts*

```
@Table
class User extends Model {
    @Column
    name: string

    @Unique
    @AllowNull(false)
    @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)
        }
    }
}

export default User
```

3) *routers/User.ts*

```
import express from "express"
import UserController from "../../controllers/users/User"
import passport from "../../middlewares/passport"

const userRoutes: express.Router = express.Router()
const controller: UserController = new UserController()

userRoutes.route('/')
    .post(controller.post)

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

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

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

userRoutes.route('/delete/:id')
    .delete(controller.deleteUser)

export default userRoutes
```

4) *services/User.ts*

```
import User from "../../models/users/User"
import checkPassword from '../../utils/checkPassword'

class UserService {
    async getById(id: number): Promise<User|Error> {
        const user = await User.findByPk(id)
        if (user == null) {
            throw new Error("Invalid identifier")
        }
        return user.toJSON()
    }

    async create(userData: any) : Promise<User> {
        try {
            const user = await User.create(userData)
            return user.toJSON()

        } catch (error: any) {
            const errors = error.errors.map((error: any) => error.message)
            throw new Error(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) }

        if (user == null) {
            throw new Error("Invalid identifier")
        }
    }

    async deleteById(id: number) {
        const user = await User.findByPk(id)
        if (user == null) {
            throw new Error("Invalid identifier")
        }
        return await User.destroy({ where: { id } })
    }
}

export default UserService
```

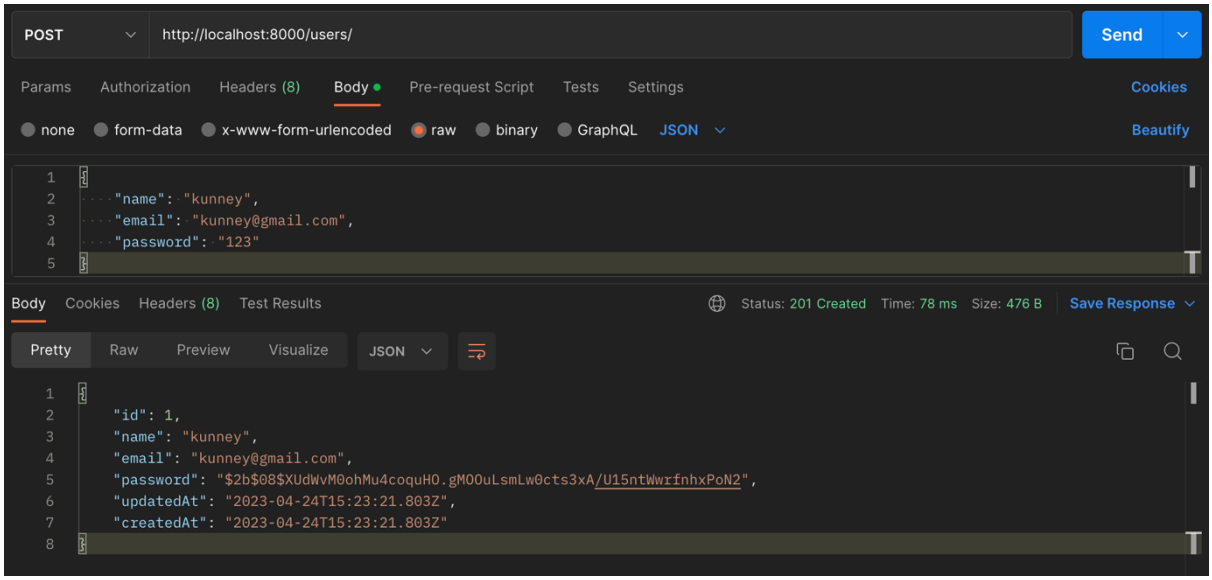
5) *controllers/User.ts*

```
import UserService from "../../services/users/User"
import { jwtOptions } from '../../middlewares/passport'
import jwt from 'jsonwebtoken'

class UserController {
    private userService: UserService;

    constructor() {
        this.userService = new UserService();
    }
```

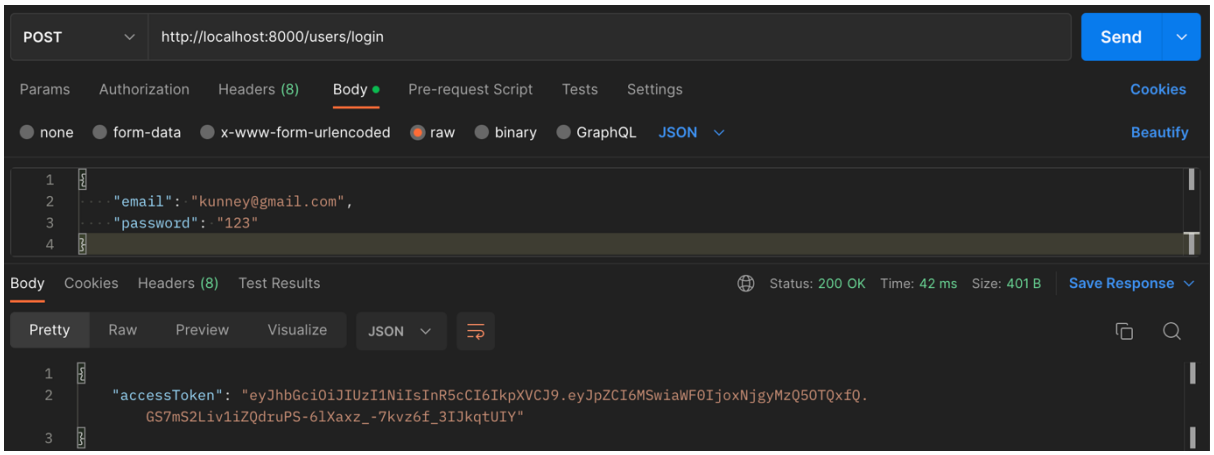
```
    get = async (request: any, response: any) => {
        try {
            const user = await this.userService.getById(
                Number(request.params.id)
            )
            response.send(user)

        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }

    post = async (request: any, response: any) => {
        const { body } = request
        try {
            const user = 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)
                response.send({accessToken: accessToken})
            } else {
                throw new Error('Login or password is incorrect!')
            }
        } catch (error: any) {
            response.status(401).send({ "error": error.message })
        }
    }

    deleteUser = async (request: any, response: any) => {
        try {
            const user = await this.userService.deleteById(
                Number(request.params.id)
            )
            response.status(204).send("User deleted")

        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
}

export default UserController
```

Результаты

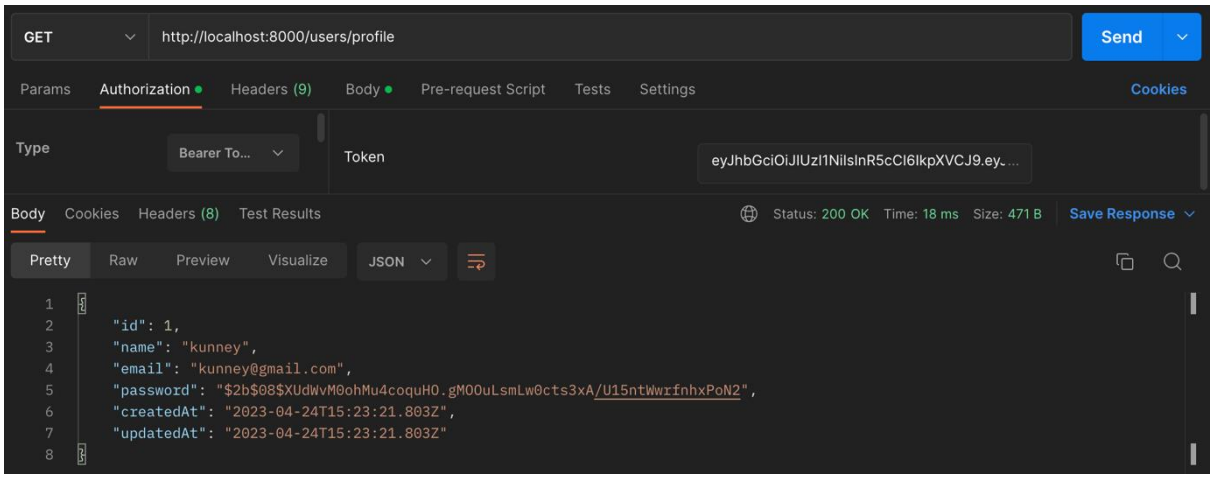
- Добавление пользователя:



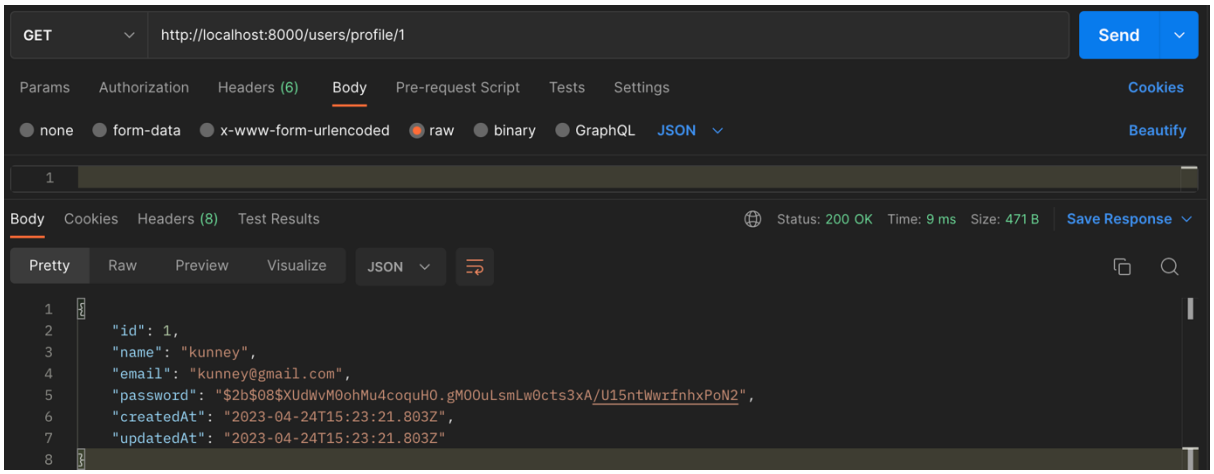
- Вход/получение токена:



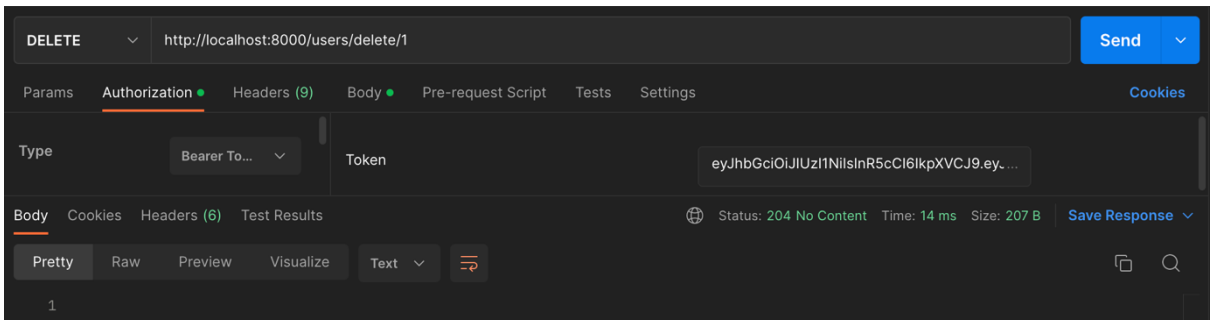
- Вход в профиль:



- Вывод профиля пользователя по id:



- Удаление профиля пользователя по id:



Вывод

В ходе данной лабораторной работы был разработан boilerplate с использованием фрейворка express, инструмента sequelize и языка программирования typescript, который в последующем может использоваться как шаблон для других новых проектов.