САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1: Написание своего boilerplate

Выполнил:

Попов Ньургун

К33412

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

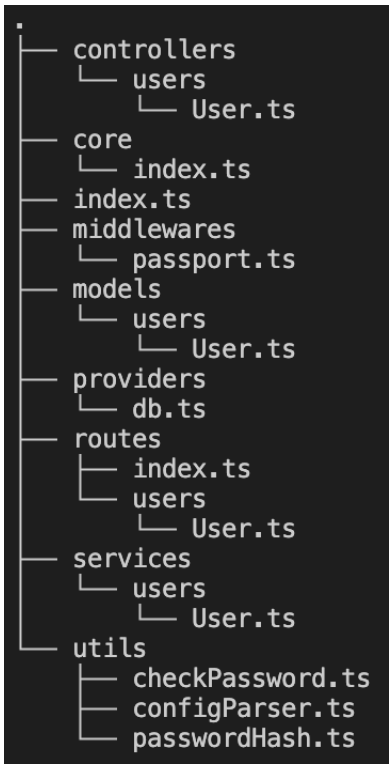
Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

Должно быть явное разделение на:

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

Ход работы

1. Структура проекта:



2. Переменное окружение:

```
PORT=8000
HOST=localhost
DB_NAME=database
JWT_SECRET=hello
JWT_LIFETIME=1h
```

3. Модель user:

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

@Table
class User extends Model {
    @PrimaryKey
    @Column
    id: number

    @Column
    firstName: string

    @Column
    lastName: 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 = passwordHash(password)
        }
    }
}

export default User
```

4. Контроллер user:

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

export class UserController {
    private userService: UserService;

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

    signup = async (request: any, response: any) => {
        const { body } = request
        try {
            await this.userService.createUser(body)
            response.status(200).send({ "status" : "OK" })
        }
        catch (error: any) {
            response.status(400).send({ "error" : error.message })
        }
```

```
    }

    login = 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 })
        }
    }

    me = async (request: any, response: any) => {
        response.send(request.user)
    }

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

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

5. Роут index:

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

const routes: express.Router = express.Router()
routes.use('/users', userRoutes)

export default routes
```

Роут user:

```
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('/signup')
    .post(controller.signup)

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

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

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

export default userRoutes
```

6. Сервисы user:

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

class UserService {
    async createUser(userData: any): Promise<User> {
        const user = await User.create(userData)
        return user.toJSON()
    }

    async getUser(id: number): Promise<User|Error> {
        const user = await User.findByPk(id)

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

    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")
        }
    }
}

export default UserService
```

7. Утилита checkPassword:

```
import bcrypt from "bcrypt"

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

Утилита configParser:

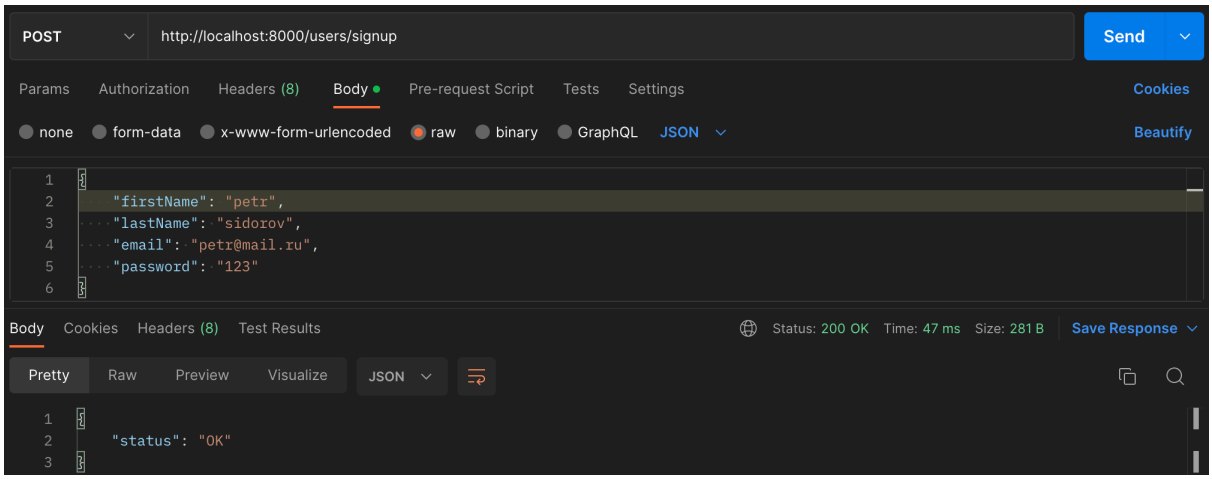
```
import ini from 'ini'
import fs from 'fs'

const configParser = (path: string, moduleName: string) : any => {
    const configFile: string = fs.readFileSync(path, 'utf-8')
    const parsedConfig: any = ini.parse(configFile)[moduleName]

    return parsedConfig
}

export default configParser
```

Утилита passwordHash:

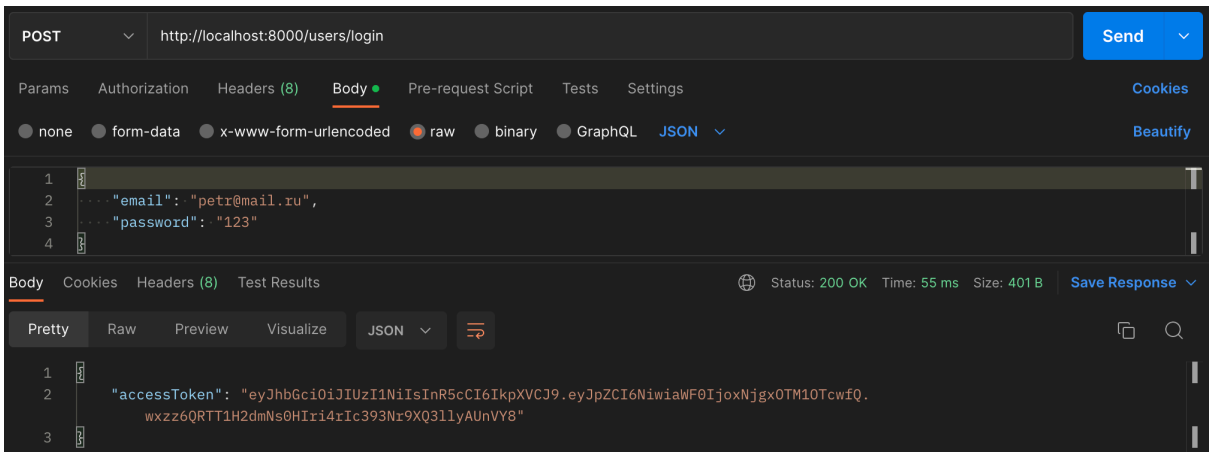
```
import bcrypt from 'bcrypt'

export default (password: string) : string => bcrypt.hashSync(password,
bcrypt.genSaltSync(8))
```

8. Вывод.

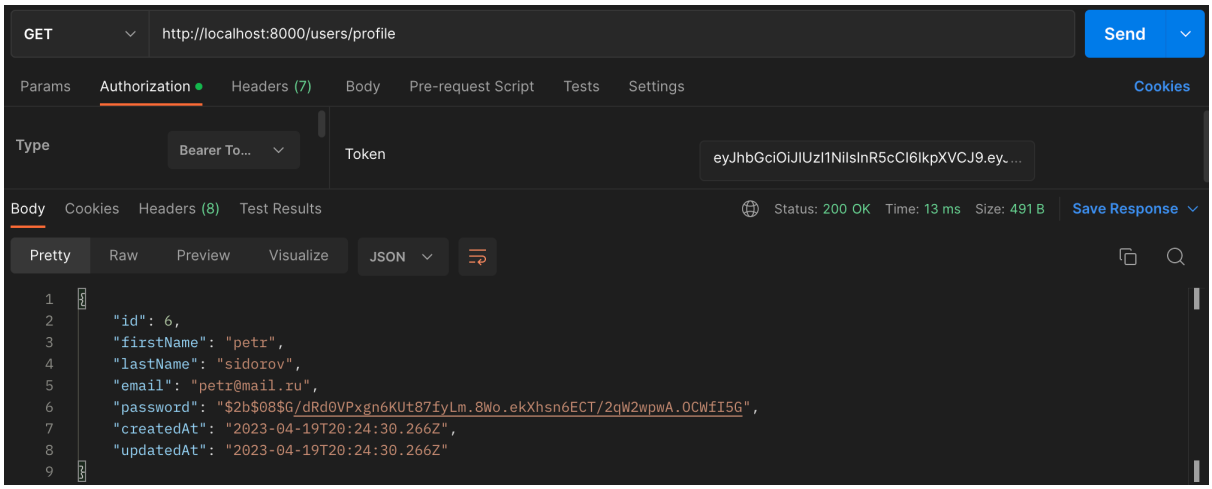
Добавление пользователя:



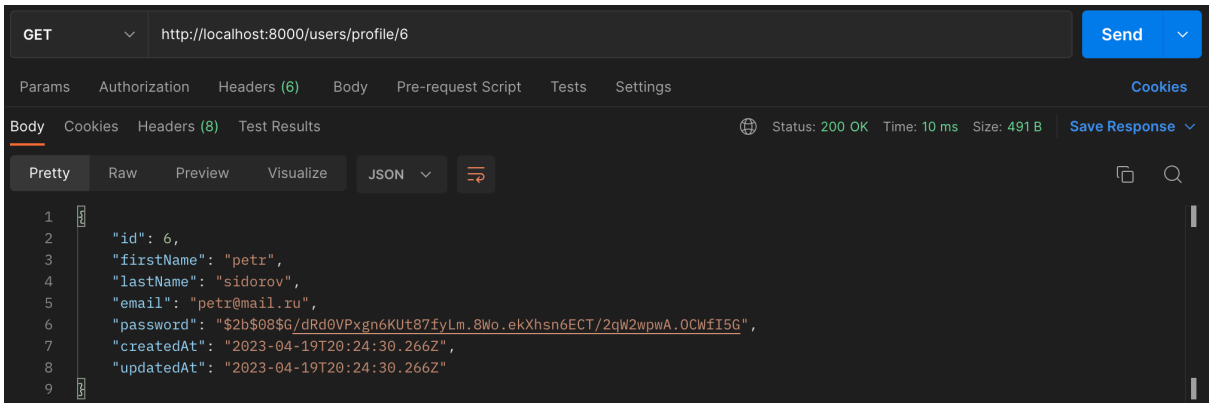
Вход/получение токена:



Вход в профиль:



Вывод профиля пользователя по id:



Вывод

В результате выполненной работы: бал создан собственный boilerplate на express + sequelize + typescript, был создан структурированный проект, который включает в себя: controller, core, middleware, model, provider, route, service и utils, также был добавлен makefile.