САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

О Т Ч Е Т

по лабораторной работе 1

Выполнила:
Бабан Виктория К33412

Проверил:
Добряков Д.И.

Санкт-Петербург
2023

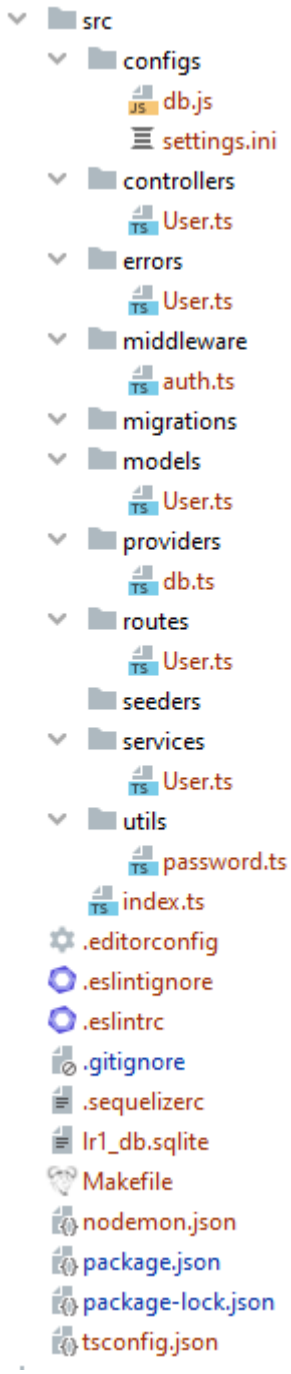
Цель работы:

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”)

Ход работы:

Написанный мною boilerplate имеет следующую структуру:



Рассмотрим коды файлов папки src:

- models/User.ts

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

    @Column
    lastName: string

    @Column
    firstName: string

    @Unique
    @AllowNull( allowNull: false)
    @Column
    email: string

    @Unique
    @AllowNull( allowNull: false)
    @Column
    username: string

    @AllowNull( allowNull: false)
    @Column
    password: string

}

export default User
```

- routes/User.ts

```
import UserController from "../controllers/User";
import express from "express";
import auth from "../middleware/auth"

const userRouter = express.Router()

const userController: UserController = new UserController()

userRouter.route( prefix: '/login').post(userController.login)
userRouter.route( prefix: '/register').post(userController.register)
userRouter.route( prefix: '/me').get(auth.auth, userController.me)
userRouter.route( prefix: '/reset').post(auth.auth, userController.updatePassword)

export default userRouter
```

- services/User.ts

```ts
import User from "../models/User"
import { hashPassword, checkPassword } from "../utils/password";
import UserError from "../errors/User";

class UserService {

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

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

    async get(email: string, password: string): Promise<User> {
        const user: User | null = await User.findOne( options: {
            where: {
               email: email,
            },
        })

        if (user != null && checkPassword(password, user.password)) {
            return user.toJSON()
        }
        throw new UserError("Invalid email/password")
    }

    async update(userData: any): Promise<User> {
        if (userData.id == undefined) {
            throw new UserError("Id is undefined")
        }

        await User.update(userData,  options: { where: {
            id: userData.id
        }})
        let user: User = await this.getById(userData.id)
        return user
    }

    async delete(id: number): Promise<void> {
        const user: User | null = await User.findByPk(id)
        if (user != null) {
            return user.destroy()
        }

        throw new UserError("Invalid identifier")
    }
}

export default UserService
```

- controllers/User.ts

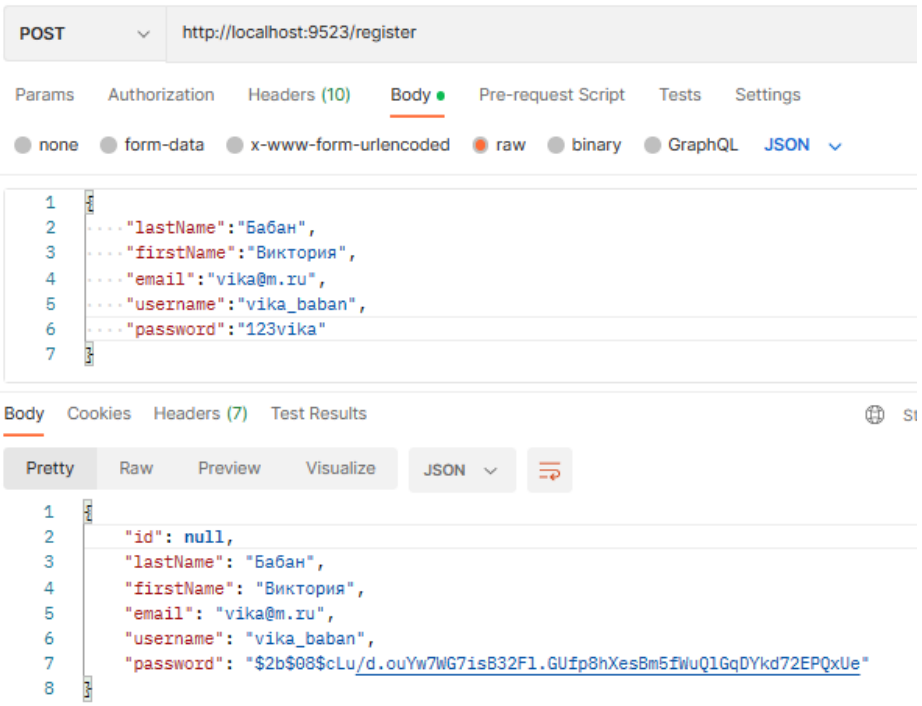
```
class UserController {
    private userService: UserService = new UserService()

    login = async (request: any, response: any) => {
        const { email, password } = request.body
        try {
            const user: User = await this.userService.get(email, password)
            const token = jwt.sign( payload: {
                id: user.id,
                username: user.username,
            }, secretOrPrivateKey: "sUbGuVE~t[)ByQDjcV?LCa_c4};LI-_n")

            response.send({
                    username: user.username,
                    email: user.email,
                    token
                })
        }
        catch (err) {
            response.status(400).send((err as UserError).message)
        }
    }

    register = async (request: any, response: any) => {
        const { body } = request
        try {
            const user = await this.userService.create(body)
            response.status(201).send(user)
        }
        catch (err) {
            response.status(400).send((err as UserError).message)
        }
    }

me = async (request: any, response: any) => {
    if (request.user === undefined) {
        response.sendStatus( code: 401)
    }
    else {
        try {
            const user = await this.userService.getById(request.user.id)
            response.send({
                id: user.id,
                lastName: user.lastName,
                firstName: user.firstName,
                username: user.username,
                email: user.email
            })
        }
        catch (err) {
            if (err as UserError)
                response.status(400).send((err as UserError).message)
            else
                response.sendStatus( code: 500)
        }
    }
}
```

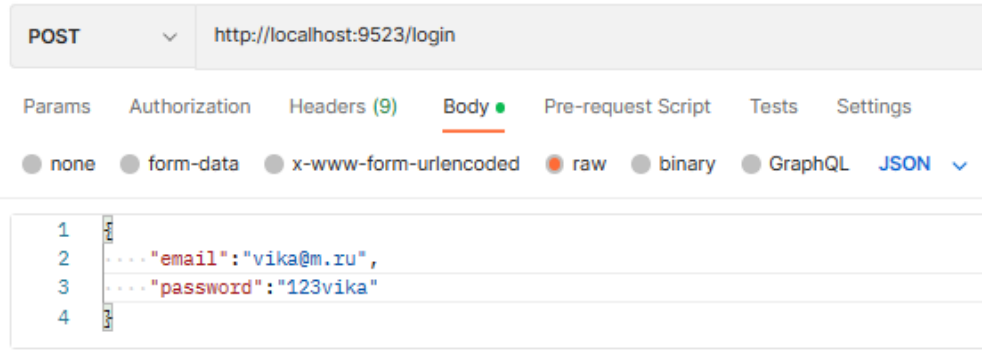
```
    updatePassword = async (request: any, response: any) => {
        const { oldPassword, newPassword } = request.body
        try {
            console.log(request.body)
            const user: User = await this.userService.getById(request.user.id)
            if (checkPassword(oldPassword, user.password)) {
                user.password = hashPassword(newPassword)
                await this.userService.update(user)
                response.status(200).send("Password change successfully")
            }
            else {
                response.sendStatus( code: 400)
            }
        }
        catch (err) {
            response.status(400).send((err as Error).message)
        }
    }
}

export default UserController
```

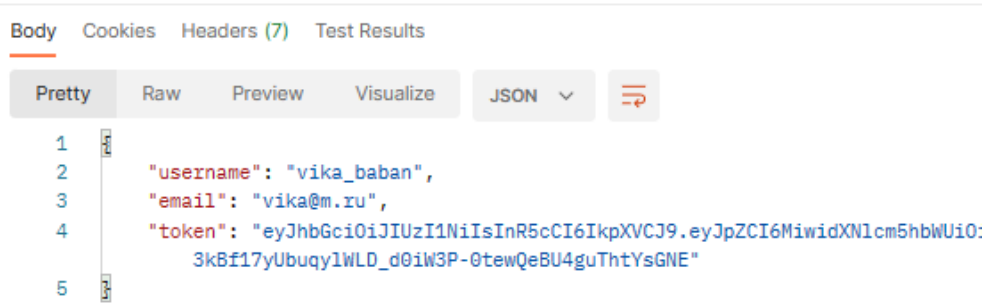
- utils/password.ts

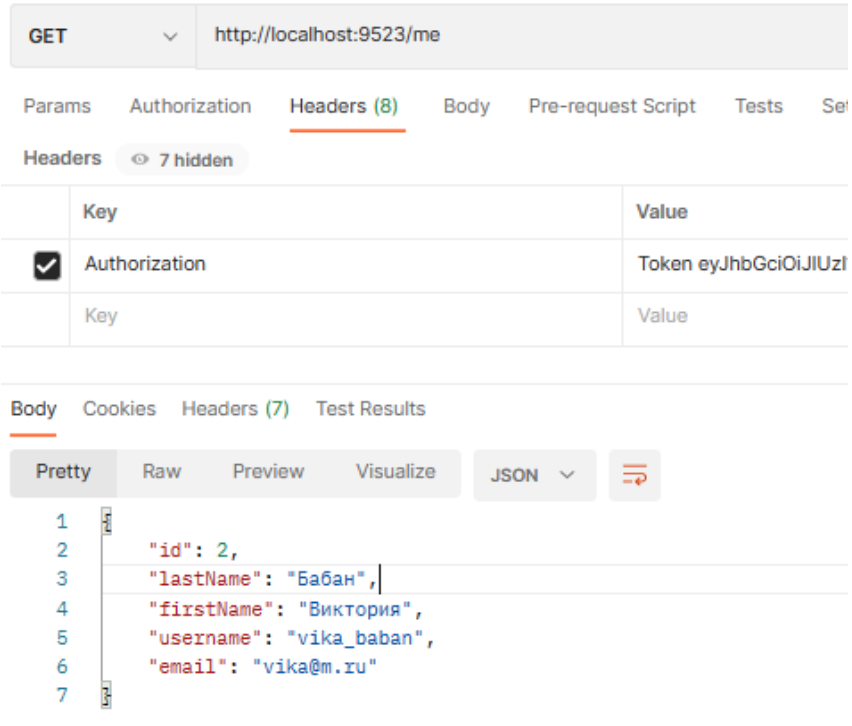
```
import bcrypt from "bcrypt";

function hashPassword(password: string): string {
    return bcrypt.hashSync(password, bcrypt.genSaltSync( rounds: 8))
}

function checkPassword(password: string, hash: string): boolean {
    return bcrypt.compareSync(password, hash)
}

export { hashPassword, checkPassword }
```

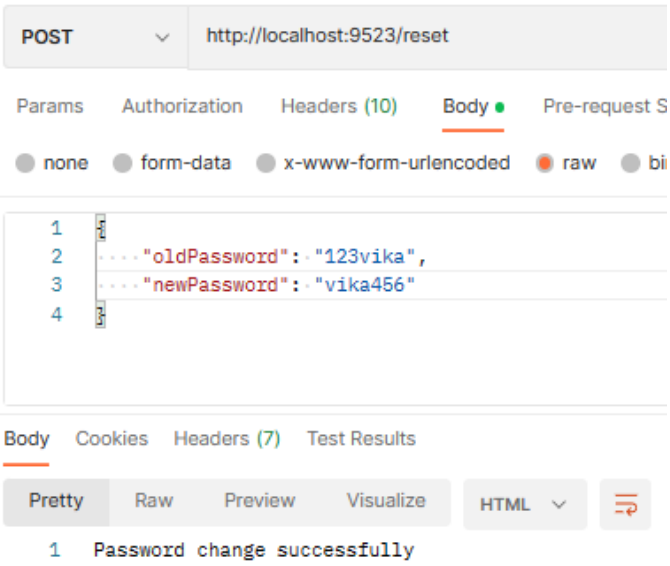
Примеры:




POST
http://localhost:9523/login
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"email":"vika@m.ru",
"password":"123vika"
}


Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"username": "vika_baban",
"email": "vika@m.ru",
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiO:
3kBf17yUbuqylWLD_d0iW3P-0tewQeBU4guThtYsGNE"
}


GET
http://localhost:9523/me
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Headers
7 hidden
Key
Value
Authorization
Token eyJhbGciOiJIUzI
Key
Value
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"lastName": "Бабан",
"firstName": "Виктория",
"username": "vika_baban",
"email": "vika@m.ru"
}



Вывод:

В результате выполнения данной работы мною был написан boilerplate, в котором описана модели пользователя и реализованы контроллеры для эндпоинтов.