САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Иконенко Данил

Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

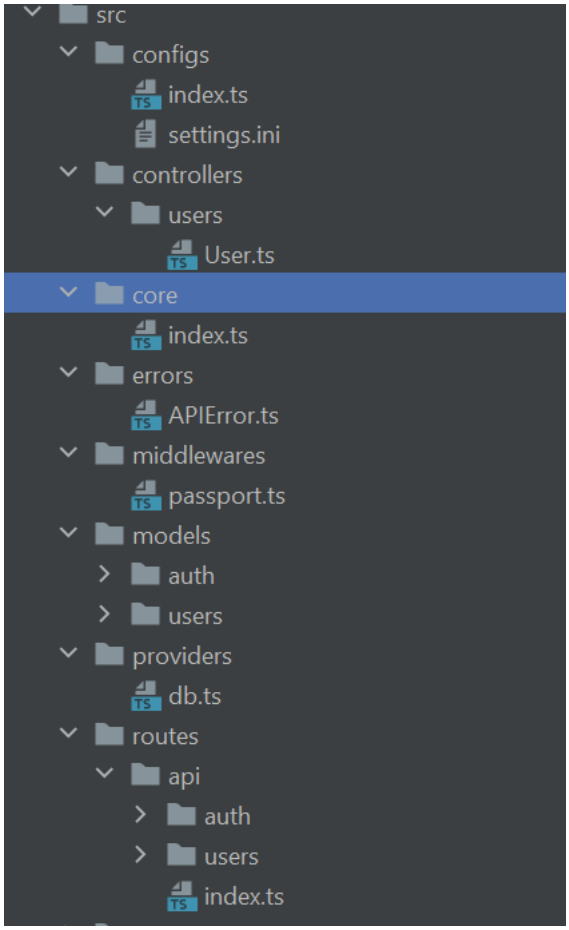
Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

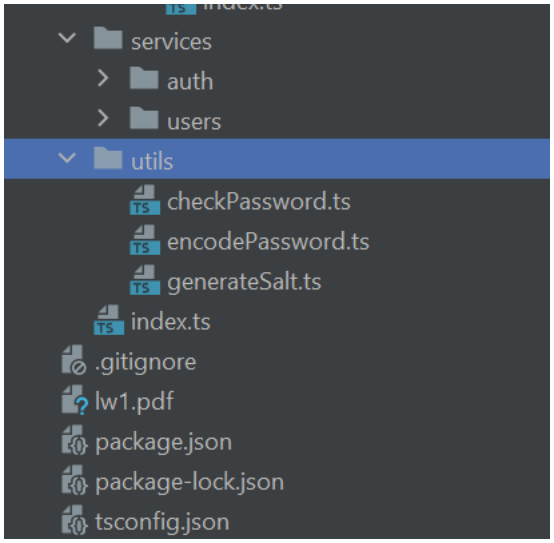
Должно быть явное разделение на:

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

Ход работы

Структура проекта:





Реализованы 2 модели: пользователь и refresh token и 5 эндпоинтов: логин, регистрация, создание пользователя, получение текущего пользователя, изменение текущего пользователя. Последние 2 эндпоинта требуют авторизации при помощи JWT.

Настройки проекта сохранены в файле .ini:

settings.ini

```
[DATABASE]
name = "my_db"
dialect = "sqlite"
username = "root"
password = ""
storage = "db.sqlite"
host = ""
port = ""
[JWT]
accessTokenLifetime = 900000
refreshTokenLifetime = 3600000
[SERVER]
port = 5000
host = "localhost"
```

Файл настроек парсится следующим кодом:

index.ts

```ts
const fs = require('fs')
const ini = require('ini')

const configFile = fs.readFileSync( path: 'src/configs/settings.ini', options: 'utf-8')
const dbConfig = ini.parse(configFile)['DATABASE']
const serverConfig = ini.parse(configFile)['SERVER']
const jwtConfig = ini.parse(configFile)['JWT']

export {
    dbConfig,
    serverConfig,
    jwtConfig
}
```

Вспомогательные функции для авторизации (генерация соли, хэширование пароля, проверка пароля):

generateSalt.ts

```ts
import {randomBytes} from 'crypto';

export default (): string => {
    return randomBytes( size: 16).toString( encoding: 'base64');
}
```

encodePassword.ts

```ts
import {createHash} from 'crypto';

export default (password: string, salt: string): string => {
    return createHash( algorithm: 'sha256').update( data: password + salt).digest( encoding: 'base64')
}
```

checkPassword.ts

```ts
import User from '../models/users/User'
import encodePassword from './encodePassword'

export default (user: User, password: string): boolean => {
    return user.password === encodePassword(password, user.salt)
}
```

Реализация модели пользователя с генерацией соли и хэшированием пароля:

User.ts

```ts
import ...

@DefaultScope( scopeGetter: () => ({
    attributes: ['id', 'firstName', 'lastName', 'email']
}))
@Table
class User extends Model {
    @AllowNull( allowNull: false)
    @Column
    firstName: string

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

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

    @AllowNull( allowNull: false)
    @Column

    @Column
    salt: string

    @BeforeCreate
    static setPasswordInitial(instance: User) {
        instance.salt = generateSalt()
        const {password} = instance
        instance.password = encodePassword(password, instance.salt)
        console.log(instance.salt)
        console.log(instance.password)
    }

    @BeforeUpdate
    static setPasswordUpdate(instance: User) {
        const {password} = instance
        if (instance.changed( key: 'password')) {
            instance.password = encodePassword(password, instance.salt)
        }
    }
}

export default User
```

Модель refresh токена:

RefreshToken.ts

```ts
import { Table, Column, Model, Unique, AllowNull, ForeignKey } from 'sequelize-typescript'
import User from '../users/User'

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

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

export default RefreshToken
```

Сервис для пользователя:

User.ts

```ts
import User from '../../models/users/User'
import APIError from '../../errors/APIError'
import checkPassword from '../../utils/checkPassword'


class UserService {
    async getById(id: number): Promise<User | APIError> {
        const user = await User.findByPk(id)
        if (user) {
            return user.toJSON()
        }
        throw new APIError('User not found')
    }

    async create(userData: any): Promise<User | APIError> {
        try {
            const user = await User.create(userData)
            return user.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new APIError(errors)
        }
    }
    async update(id: number, userData: any): Promise<User | APIError> {
        let user = await User.findByPk(id)
        if (user) {
            try {
                user = await user.update(userData)
                return user.toJSON()
            } catch (e: any) {
                const errors = e.errors.map((error: any) => error.message)
                throw new APIError(errors)
            }
        }
        throw new APIError('User not found')
    }

    async checkPassword(email: string, password: string): Promise<any> {
        const user = await User.findOne( options: {where: {email}})
        if (user) {
            return {user: user.toJSON(), checkPassword: checkPassword(user, password)}
        }
        throw new APIError('Incorrect credentials')
    }
}

export default UserService
```

Сервис для refresh токена:

RefreshToken.ts

```
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( values: {token, userId})
        return token
    }
    isRefreshTokenExpired = async (token: string): Promise<{ userId: number | null, isExpired: boolean }> =>
        const refreshToken = await RefreshToken.findOne( options: {where: {token}})
        if (refreshToken) {
            const tokenData = refreshToken.toJSON()
            const currentDate = new Date()
            const timeDelta = currentDate.getTime() - tokenData.createdAt.getTime()
            if (timeDelta > 0 && timeDelta < jwtConfig.refreshTokenLifetime) {
                return {userId: tokenData.userId, isExpired: false}
            }
            return {userId: null, isExpired: true}
        }
        return {userId: null, isExpired: true}
    }
}

export default RefreshTokenService
```

Контроллер для пользователей и авторизации:

```
class UserController {
    private userService: UserService

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

    // get = async (request: any, response: any) => {
    //     try {
    //         const user: User | APIError = await this.userService.getById(
    //             Number(request.params.id)
    //         )
    //         response.send(user)
    //     } catch (error: any) {
    //         response.status(404).send({'detail': error.message})
    //     }
    // }

```

```
    post = async (request: any, response: any) => {
        const {body} = request
        try {
            const user: User | APIError = await this.userService.create(body)
            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({'detail': error.message})
        }
    }

    // Get only allowed for myself
    get = async (request: any, response: any) => {
        const {user} = request
        if (user) {
            response.send(user)
        } else {
            response.status(401).send({'detail': 'Not authenticated'})
        }
    }
```

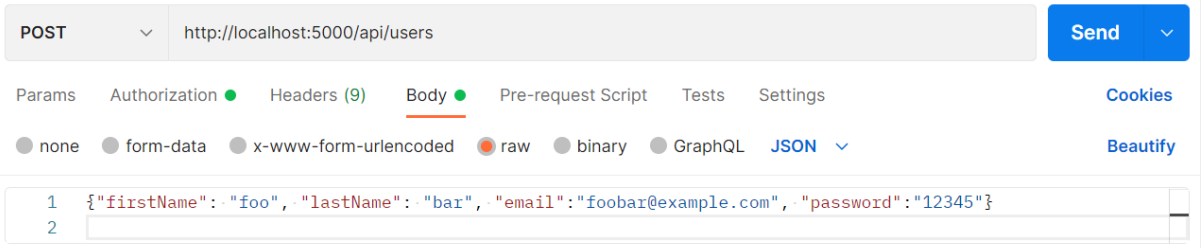
```
    // Put only allowed for myself
    put = async (request: any, response: any) => {
        const {body, user} = request
        if (user) {
            try {
                const updatedUser: User | APIError = await this.userService.update(user.id, body)
                response.status(200).send(updatedUser)
            } catch (error: any) {
                response.status(400).send({'detail': error.message})
            }
        } else {
            response.status(401).send({'detail': 'Not authenticated'})
        }
    }
```

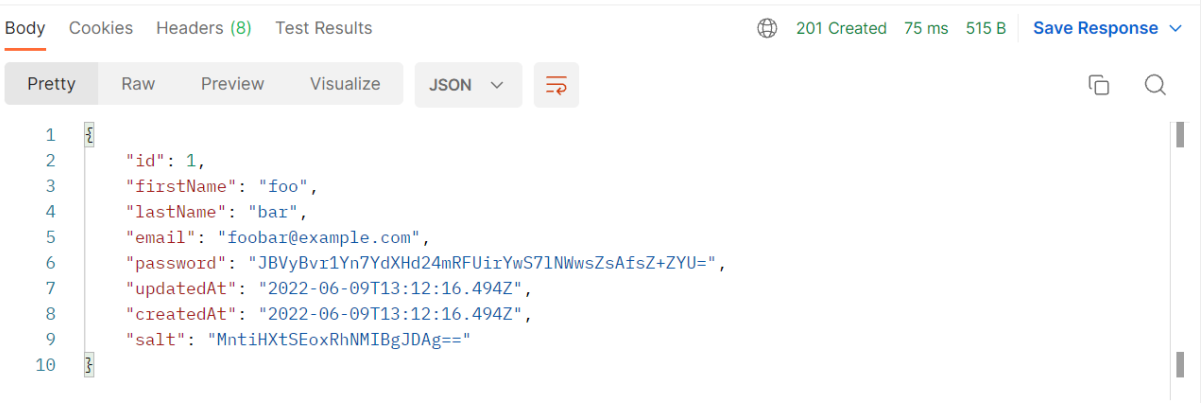
```
    login = async (request: any, response: any) => {
        const {body} = request
        const {email, password} = body
        try {
            const {user, checkPassword} = await this.userService.checkPassword(email, password)

            if (checkPassword) {
                const payload = {id: user.id}
                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
                const refreshTokenService = new RefreshTokenService(user)
                const refreshToken = await refreshTokenService.generateRefreshToken()

                response.send({accessToken, refreshToken})
            } else {
                response.status(400).send({'detail': 'Invalid credentials'})
            }
        } catch (e: any) {
            response.status(400).send({'detail': e.message})
        }
    }
```

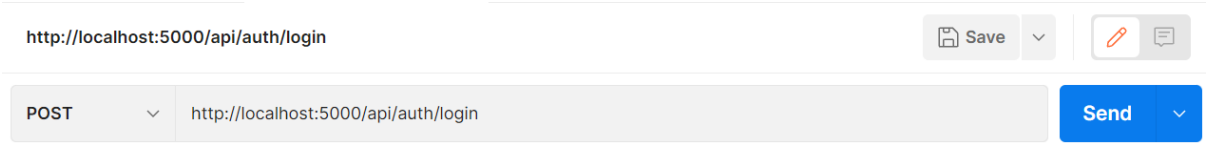
```
    refreshToken = async (request: any, response: any) => {
        const {body} = request
        const {refreshToken} = body
        const refreshTokenService = new RefreshTokenService()
        try {
            const {userId, isExpired} = await refreshTokenService
                .isRefreshTokenExpired(refreshToken)

            if (!isExpired && userId) {
                try {
                    const user = await this.userService.getById(userId)
                    // @ts-ignore
                    const payload = {id: user.id}
                    const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
                    // @ts-ignore
                    const refreshTokenService = new RefreshTokenService(user)
                    const refreshToken = await refreshTokenService.generateRefreshToken()
                    response.send({accessToken, refreshToken})
                } catch (e: any) {
                    response.status(400).send({'detail': e.message})
                }
```

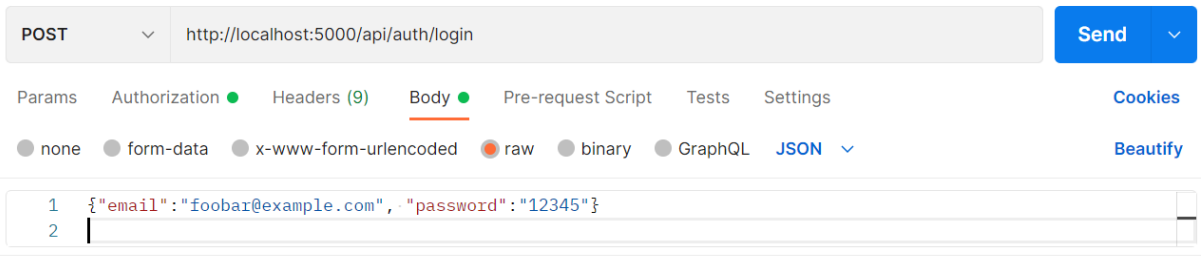
```
            } else {
                response.status(401).send({'error': 'Invalid credentials'})
            }
        } catch (e: any) {
            response.status(401).send({'error': e.message})
        }
    }
}

export default UserController
```

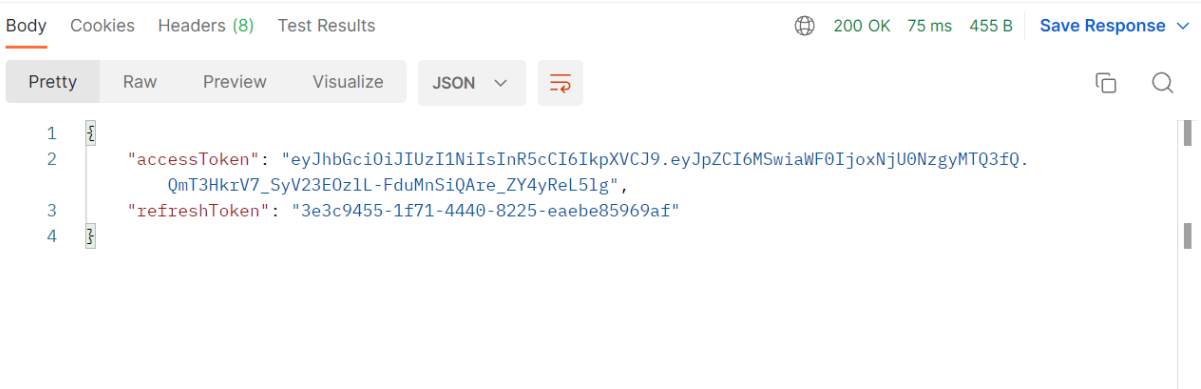
Работа эндпоинтов:

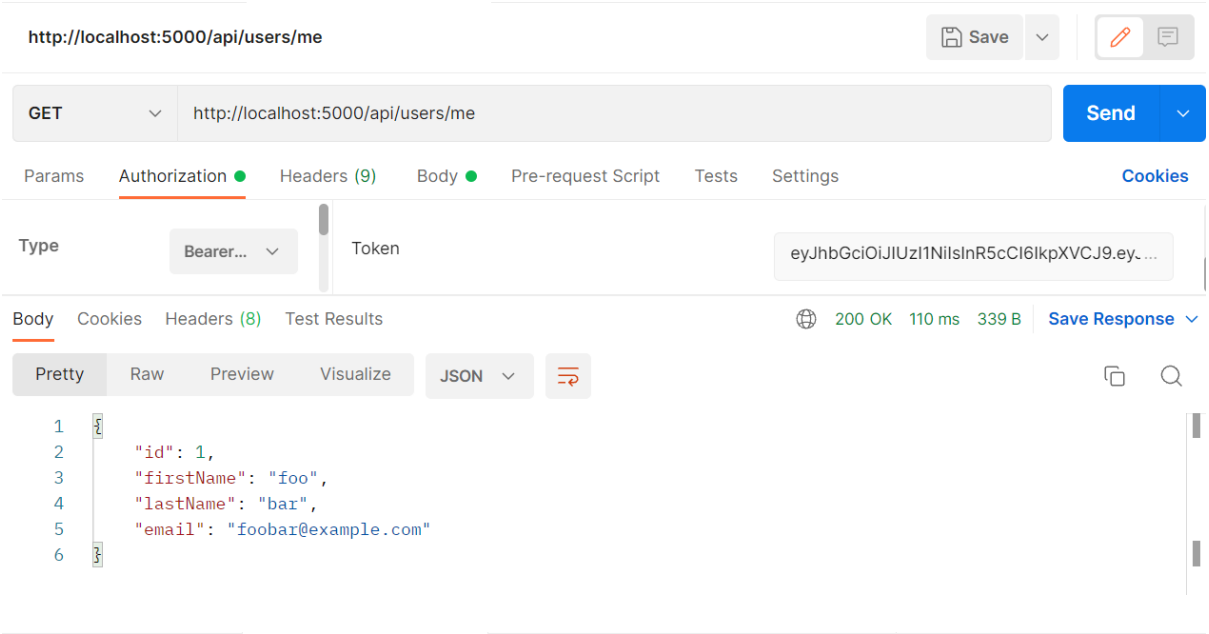

POST
http://localhost:5000/api/users
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{"firstName": "foo", "lastName": "bar", "email":"foobar@example.com", "password":"12345"}

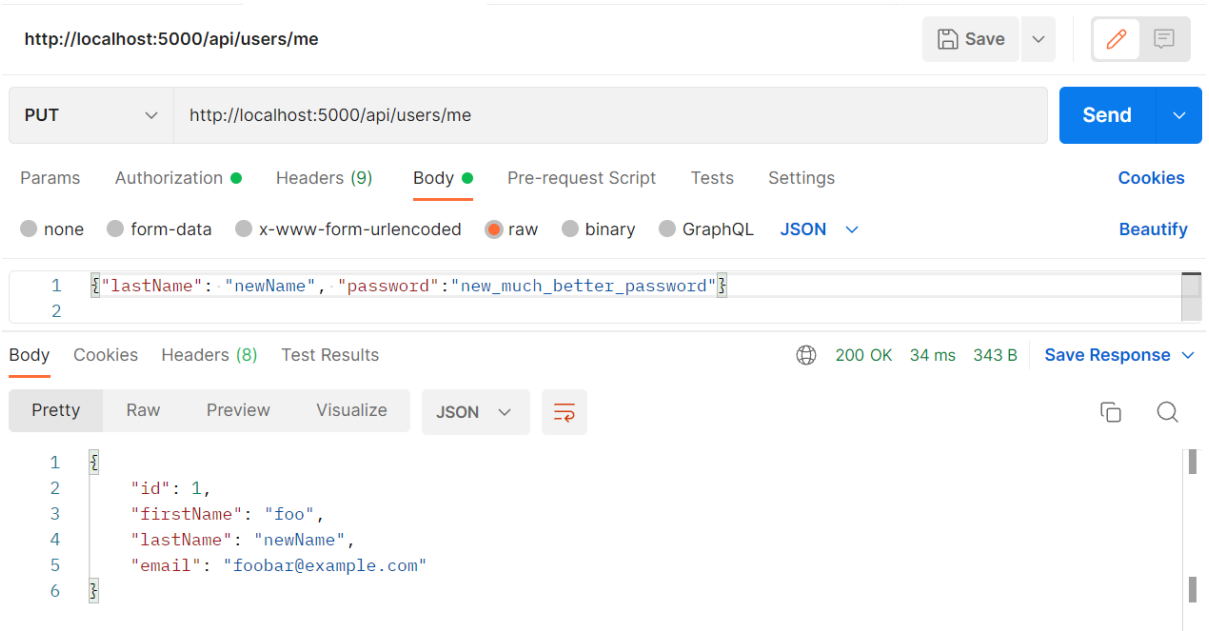

Body
Cookies
Headers (8)
Test Results
201 Created
75 ms
515 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"firstName": "foo",
"lastName": "bar",
"email": "foobar@example.com",
"password": "JBVyBvr1Yn7YdXHd24mRFUirYwS7lNWwsZsAfsZ+ZYU=",
"updatedAt": "2022-06-09T13:12:16.494Z",
"createdAt": "2022-06-09T13:12:16.494Z",
"salt": "MntiHXtSEoxRhNMIBgJDAg=="
}


http://localhost:5000/api/auth/login
Save


POST
http://localhost:5000/api/auth/login
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{"email":"foobar@example.com", "password":"12345"}


Body
Cookies
Headers (8)
Test Results
200 OK
75 ms
455 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNjU0NzgyMTQ3fQ.
QmT3HkrV7_SyV23EOzlL-FduMnSiQAre_ZY4yReL5lg",
"refreshToken": "3e3c9455-1f71-4440-8225-eaebe85969af"
}





Вывод

Я создал boilerplate проекта с реализованным функционалом пользователя и авторизации