САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Старовойтова Елизавета

Группа:

К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 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”)

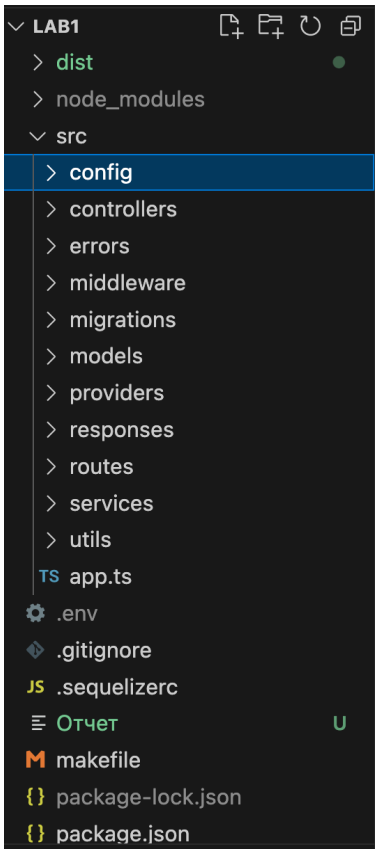
Ход работы

Для boilerplate было выбрано реализовать функциональность для пользователя: регистрацию, авторизацию и аунтентификацию с access и refresh токенами, хэширование пароля, просмотр информации о пользователе по id, просмотр всех пользователей, просмотр текущего пользователя, удаление пользователя по id, выход из приложения.

Первым делом инициализируем модель и создаем зависимости. Создаем файл tsconfig.json.

```
{
  "compilerOptions": {
    "strictPropertyInitialization": false,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "target": "es2016",
    "module": "commonjs",
     "baseUrl": "src",
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "skipLibCheck": true,
    "outDir": "./dist",
    "rootDir": "./src"
  }
}
```

Структура проекта.



Модель пользователя.

```
import {
    Table,
    Column,
    Model,
    Unique,
    AllowNull,
    PrimaryKey,
    AutoIncrement
} from 'sequelize-typescript'


@Table
class User extends Model {
    @PrimaryKey
    @AutoIncrement
    @Column
    id: Number
    @Column
    firstName: string
    @Column
    lastName: string

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
```

```
    password: string
}

export default User
```

Директория providers и файл sequelize.ts для инициализации подключения к бд с помощью Sequelize ORM.

```
import { Sequelize } from 'sequelize-typescript'
import { Dialect } from 'sequelize'
import models from '../models'

const initSequelize = async () => {
    const sequelize = new Sequelize({
        database: process.env.DB_DATABASE,
        dialect: process.env.DB_DIALECT as Dialect,
        username: process.env.DB_USERNAME,
        password: process.env.DB_PASSWORD,
        host: process.env.HOST,
        logging: console.log,
    })

    sequelize.addModels(models)

    try {
        await sequelize.authenticate();
        console.log('Connection has been established successfully.');
    } catch (error) {
        console.error('Unable to connect to the database:', error);
    }

    return sequelize
}

export default initSequelize
```

Директория utils и файл jwt.ts для работы с токенами.

```
import jwtConfig from "../config/jwt";
const jwt = require('jsonwebtoken');

class TokenType {
    static ACCESS = 'ACCESS'
    static REFRESH = 'REFRESH'
}

export interface JwtToken {
    sub: string,
    type: TokenType,
    iat: Number,
    exp: Number,
}

class Jwt {
    static generateAccessToken(sub: string, payload: Object = {}): string {
        return this._signToken(
            sub,
```

```
            payload,
            TokenType.ACCESS,
            jwtConfig.JWT_ACCESS_TOKEN_TTL,
        )
    }
    static generateRefreshToken(sub: string, payload: Object = {}): string {
        return this._signToken(
            sub,
            payload,
            TokenType.REFRESH,
            jwtConfig.JWT_REFRESH_TOKEN_TTL,
        )
    }
    static _signToken(sub: string, payload: Object, type: string, ttl: number):
string {
        const now = Date.now()
        const data = {
            sub: sub,
            type: type,
            iat: now,
            exp: now + ttl * 1000
        }
        return jwt.sign(data, jwtConfig.JWT_SECRET, { algorithm: jwtConfig.JWT_ALG })
    }
    static verify(token: string): JwtToken {
        return jwt.verify(token, jwtConfig.JWT_SECRET, {algorithms: ['HS256']})
    }
}

export default Jwt
```

Директория middleware и auth.ts для проверки наличия и валидности токенов в запросах.

```
import express, {NextFunction} from "express";
import {UnauthenticatedError} from "../errors";
import Jwt, {JwtToken} from "../utils/jwt";
import User from "../models/user";

const checkAccessToken = async (request: express.Request, response: express.Response,
next: NextFunction) => {
    try {
        request.user = await checkToken(request.cookies.jwt_access)
    } catch (e: any) {
        next(e)
    }
    next()
}

const checkRefreshToken = async (request: express.Request, response:
express.Response, next: NextFunction) => {
    try {
        request.user = await checkToken(request.cookies.jwt_refresh)
    } catch (e: any) {
        next(e)
    }
```

```
    next()
}

const checkToken = async (token: string|null): Promise<User> => {
    if (!token) {
        throw new UnauthenticatedError()
    }

    let data: JwtToken
    try {
        data = Jwt.verify(token)
    } catch (e: any) {
        throw new UnauthenticatedError()
    }

    // @ts-ignore
    const user: User = await User.findOne({where: { email: data.sub }});
    if (!user) {
        throw new UnauthenticatedError()
    }

    return user
}

export {checkAccessToken, checkRefreshToken}
```

Директория controllers для обработки запросов пользователей.

Для демонстрации authController.ts

```
import ApiResponse from '../responses/apiResponse'
import {NextFunction} from "express";
import express from 'express';
import jwtConfig from '../config/jwt';
import AuthService from '../services/authService';


class AuthController {

    register = async (request: express.Request, response: express.Response, next:
NextFunction) => {
        try {
            const user = await AuthService.register(request.body)
            ApiResponse.payload(response, user)
        } catch (e: any) {
            next(e)
        }
    }

    login = async (request: express.Request, response: express.Response, next:
NextFunction) => {
        try {
            const tokens = await AuthService.login(request.body)
            response.cookie('jwt_access', tokens.accessToken, { maxAge:
jwtConfig.JWT_ACCESS_TOKEN_TTL * 1000, httpOnly: true });
```

```
            response.cookie('jwt_refresh', tokens.refreshToken, { maxAge:
jwtConfig.JWT_REFRESH_TOKEN_TTL * 1000, httpOnly: true });
            ApiResponse.payload(response, tokens)
        } catch (e: any) {
            next(e)
        }
    }

    refresh = async (request: express.Request, response: express.Response, next:
NextFunction) => {
        try {
            const tokens = await AuthService.refreshToken(request.user)
            response.cookie('jwt_access', tokens.accessToken, { maxAge:
jwtConfig.JWT_ACCESS_TOKEN_TTL * 1000, httpOnly: true });
            response.cookie('jwt_refresh', tokens.refreshToken, { maxAge:
jwtConfig.JWT_REFRESH_TOKEN_TTL * 1000, httpOnly: true });
            ApiResponse.payload(response, tokens)
        } catch (e: any) {
            next(e)
        }
    }

    logout = async (request: express.Request, response: express.Response, next:
NextFunction) => {
        try {
            // await LogoutUseCase.run(request) todo revoke
            response.clearCookie('jwt_access')
            response.clearCookie('jwt_refresh')
            ApiResponse.success(response, 'ok')
        } catch (e: any) {
            next(e)
        }
    }

    me = async (request: express.Request, response: express.Response, next:
NextFunction) => {
        try {
            ApiResponse.payload(response, request.user)
        } catch (e: any) {
            next(e)
        }
    }

}

export default AuthController
```

Директория services для организации бизнес-логики.

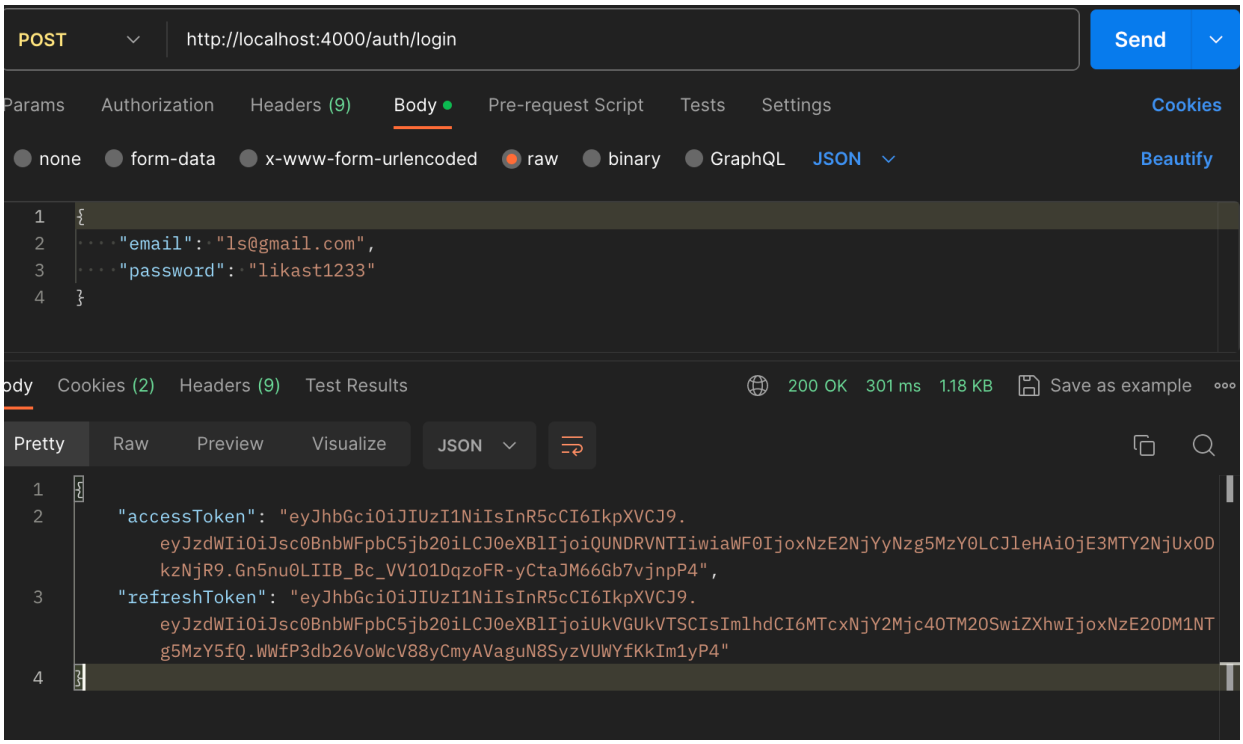
authService.ts для примера.

```
import User from "../models/user";
const bcrypt = require('bcrypt');
import Jwt from "../utils/jwt";
import {UnauthenticatedError} from "../errors";
```

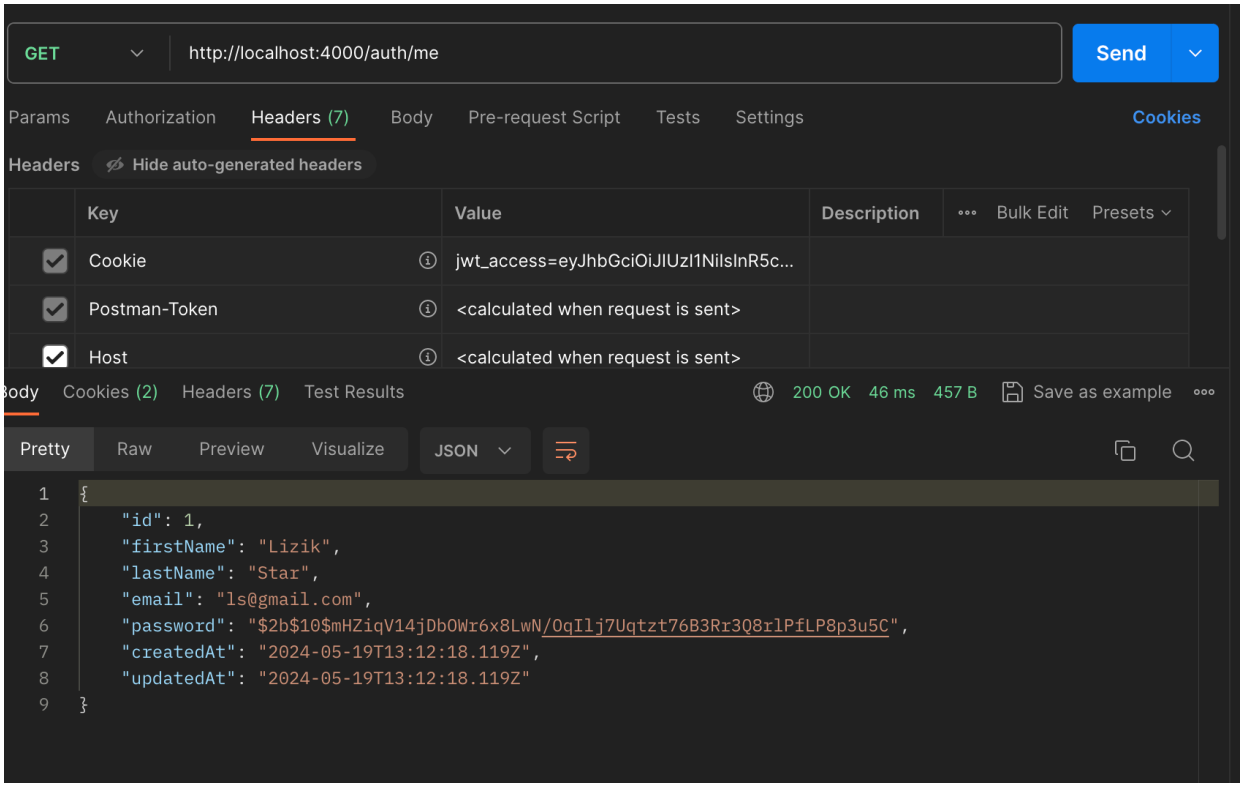
```
class AuthService {

    static async register(data: any): Promise<User> {
        return await User.create({
            firstName: data.firstName,
            lastName: data.lastName,
            email: data.email,
            password: await bcrypt.hash(data.password, 10),
        }, {returning: true})
    }

    static async login(data: any) {
        const user = await User.findOne({where: { email: data.email }});
        if (!user) {
            throw new Error('Not found user by email: ' + data.email)
        }
        if (!bcrypt.compareSync(data.password, user.password)) {
            throw new Error('Password is incorrect')
        }

        let accessToken = Jwt.generateAccessToken(data.email)
        let refreshToken = Jwt.generateRefreshToken(data.email)

        return {
            accessToken: accessToken,
            refreshToken: refreshToken,
        }
    }

    static async refreshToken(user: User|null) {
        if (!user) throw new UnauthenticatedError()
        let accessToken = Jwt.generateAccessToken(user.email)
        let refreshToken = Jwt.generateRefreshToken(user.email)

        return {
            accessToken: accessToken,
            refreshToken: refreshToken,
        }
    }
}

export default AuthService
```

Примеры работы эндпоинтов:

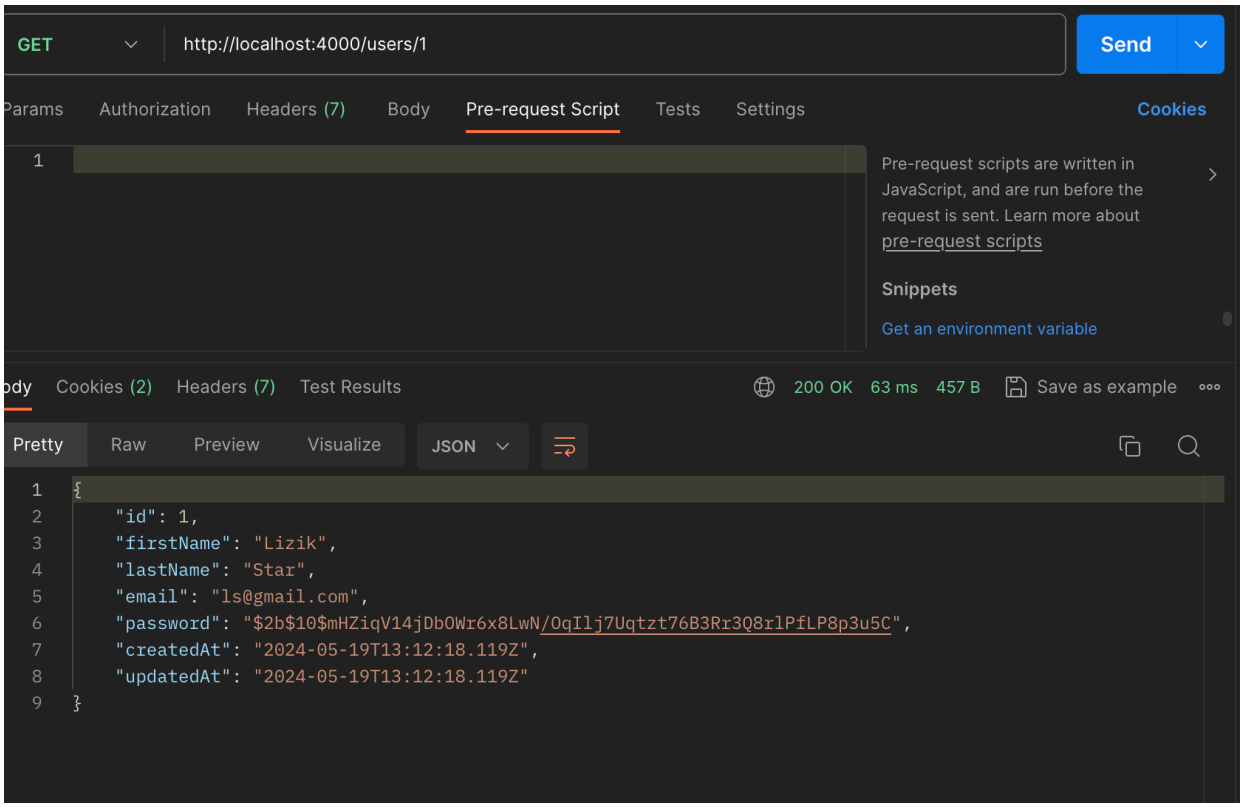
Login



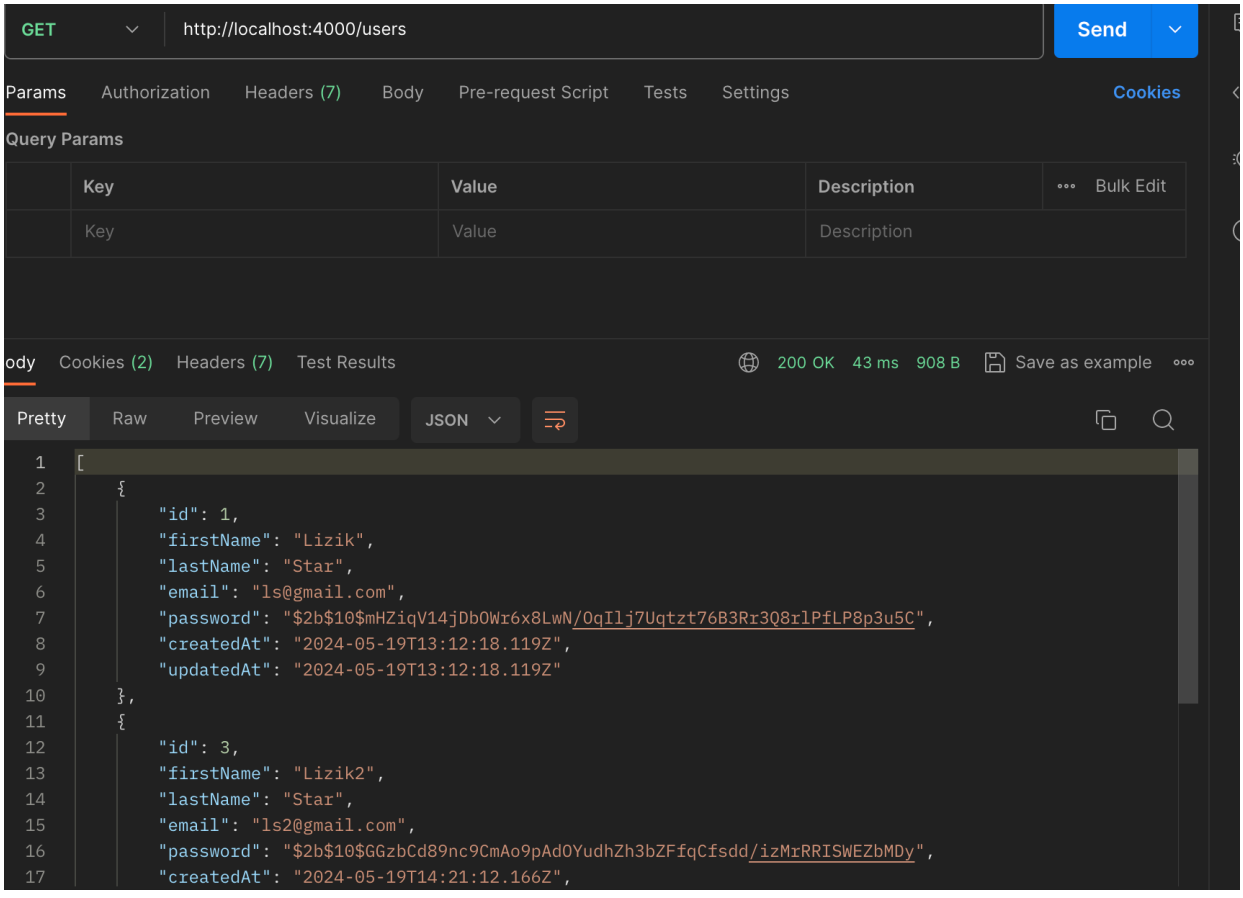
Информация о текущем пользователе.



Получение пользователя по id.



Получение всех пользователей.



Вывод:

В данной лабораторной работе я изучила новые технологии и написала свой boilerplate на express + sequelize.