**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Лабораторная работа № 2: REST, RESTful, SOAP, GraphQL.

Выполнил:

Безруков Андрей
Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

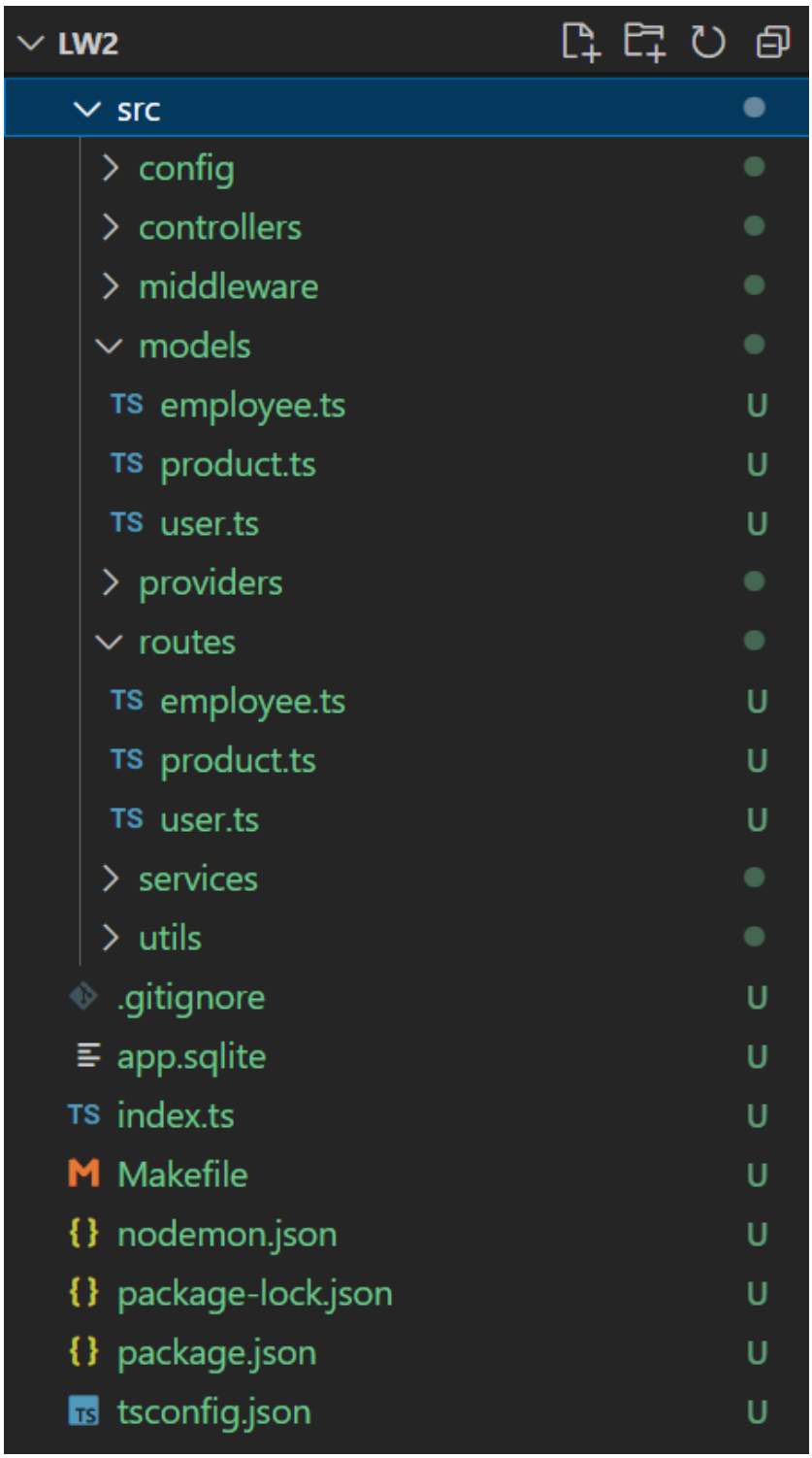
**Задача**

По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Вариант: онлайн-магазин Cute online shop API

**Ход работы**

Структура проекта:



Было реализовано 3 модели. Была использована библиотека sequelize-typescript.

- Пользователь – user.ts

```ts
import { Table, Model, PrimaryKey, Column, Unique, AllowNull } from "sequelize-typescript";

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

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

    @AllowNull(false)
    @Column
    password: string

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

export default User
```

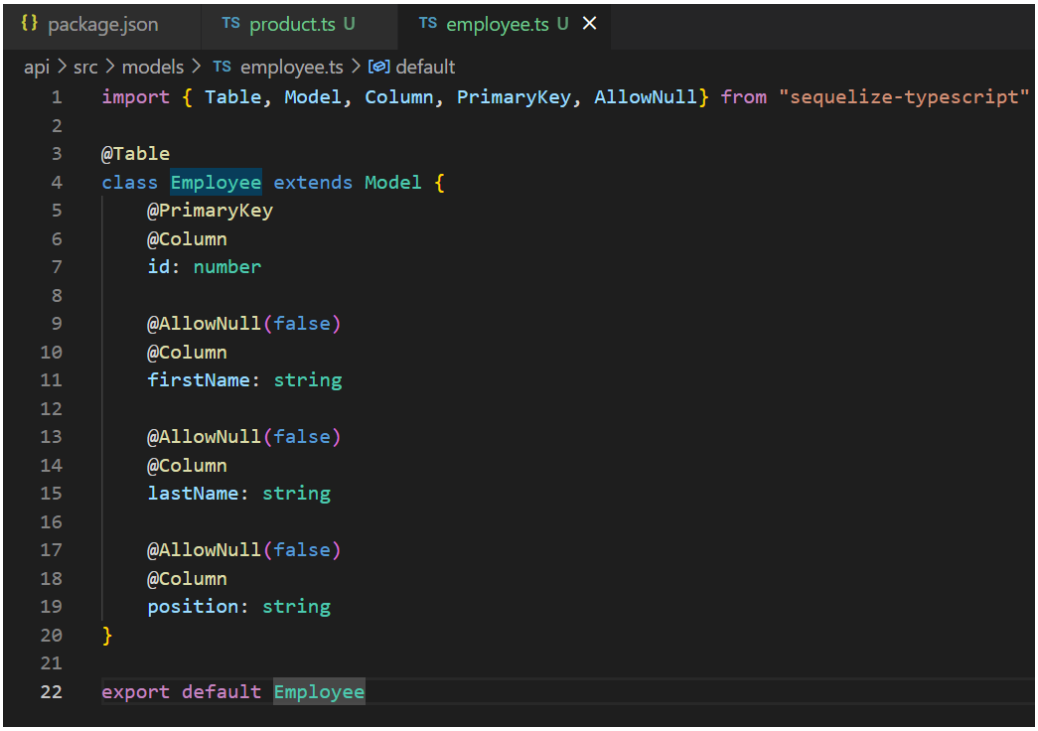
- Товар – product.ts

package.json | product.ts U

api > src > models > product.ts > ...

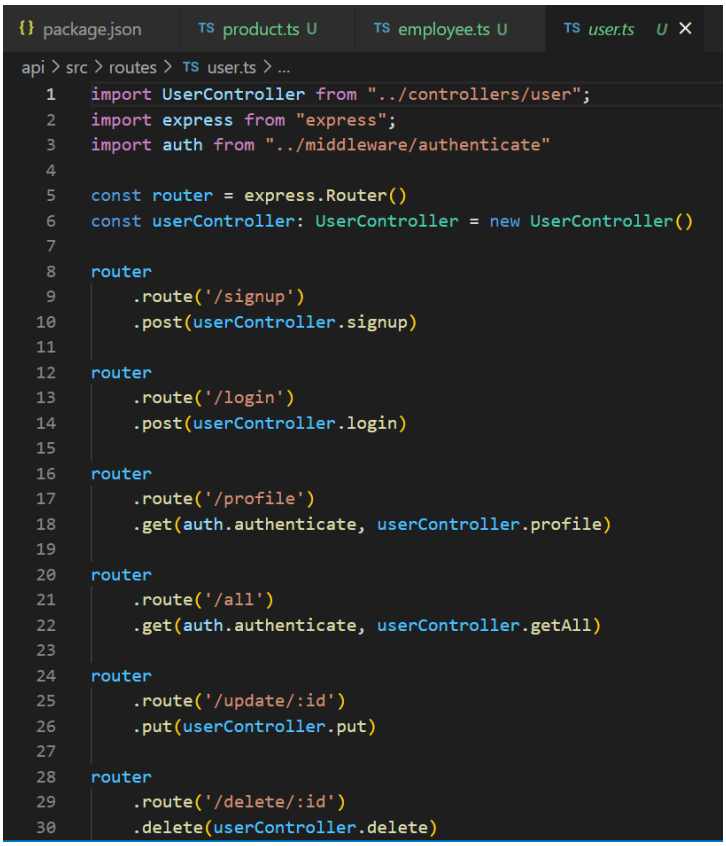
```ts
import { Table, Model, PrimaryKey, Column, AllowNull} from "sequelize-typescript";

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

    @AllowNull(false)
    @Column
    count: number

    @AllowNull(false)
    @Column
    name: string

    @AllowNull(false)
    @Column
    category: string

    @AllowNull(false)
    @Column
    price: number
}

export default Product
```

- Работник – employee.ts



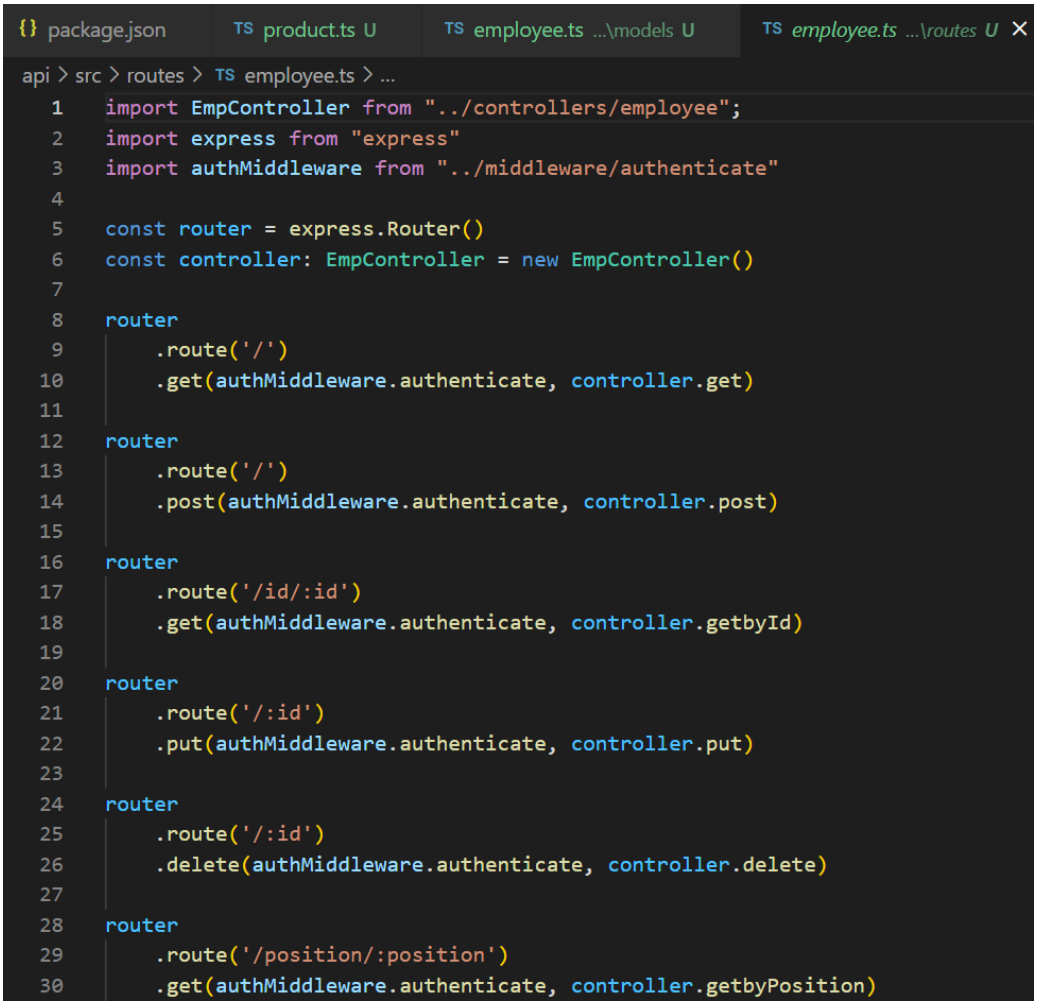

Роутинг и эндпоинты:

- Пользователь – регистрация, логин, получение профиля авторизованного, всех пользователей, обновление и удаление пользователей



- Товар – получение всех товаров/по id/по категории/по названию, добавление, изменение, удаление товаров, подсчет товаров


- Работник – получение всех работников/по id/по должности, изменение, добавление, удаление работников




Контролеры (пример для товаров)

```
import ProdService from "../services/product";

class ProdController {
    private productService: ProdService
    constructor() {
        this.productService = new ProdService()
    }

    get = async (request: any, response: any) => {
        try {
            const products = await this.productService.getAll()
            return response.json(products);
        } catch (e: any) {
            response.status(404).send({ "error": e.message })
        }
    }

    getbyId = async (request: any, response: any) => {
        try {
            const product = await this.productService.getById(request.params.id)
            return response.json(product);
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
```

Сервисы (пример для пользователей). Здесь также хэшируется пароль и указывается, что войти в систему можно и с username, и с email.

```ts
import User from "../models/user"
import { passwordHash } from "../utils/passwordHash";
import { Op } from "sequelize";

class UserService {
    async create(userInfo: any) {
        userInfo.password = passwordHash(userInfo.password)
        try {
            const user = await User.create(userInfo)
            return user
        } catch (e: any) {
            throw new Error(e)
        }
        }

    async get(username: string, password: string): Promise<User> {
        const hash: string = passwordHash(password);
        const user: User | null = await User.findOne({
            where: {
                [Op.or]: {
                    username: username,
                    email: username,
                },
                password: hash
            }
        })

        if (user == null) {
            throw new Error("Credentials are invalid")
        }
```

Функция хэширования пароля с использованием алгоритма sha512 – passwordHash.ts

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

function passwordHash(password: string): string {
    return createHash('sha512').update(password).digest('base64').toString()
}

export { passwordHash }
```

Хранение JWT-секрета – jwtsecret.ts

```
api > src > config > TS jwtsecret.ts > ...
1   const jwtsecret = "AZ95tw0Ionn7XPZZfN0NOml3d9FMfmSgXwovR9Ap6ryDIoGRU8EPHAB6iHsc0fb"
2
3   export default jwtsecret
```

Аутентификация реализована в файле authenticate.ts. Она происходит с использованием JWT-токена. Метод jwt.sign создает уникальную подпись, jwt.verify проверяет валидность токена.

```
5   class AuthMiddleware {
6
7       authenticate = async (request: any, response: any, next: any) => {
8           var authHeader = request.headers.authorization
9           if (authHeader) {
10              try {
11                  const token = authHeader.split(' ')[1]
12                  const user = await this.authenticateToken(token)
13                  request.user = user
14
15                  next()
16              }
17              catch (error: any) {
18                  response.sendStatus(401)
19              }
20
21          }
22          else {
23              response.sendStatus(401)
24          }
25      }
26
27      createToken = async (user: any): Promise<string> => {
28          return jwt.sign(lodash.pick(user, ['id', 'username']), jwtsecret)
29      }
30
31      authenticateToken = async (token: string): Promise<any> => {
32          return jwt.verify(token, jwtsecret)
33      }
```

**Вывод**

Было успешно реализовано API, все эндпоинты работают, так же была подключена авторизация с помощью токенов.