САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа № 1: Boilerplate на express + sequelize + typescript.

Выполнил:

Безруков Андрей
Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

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

Должно быть явное разделение на:

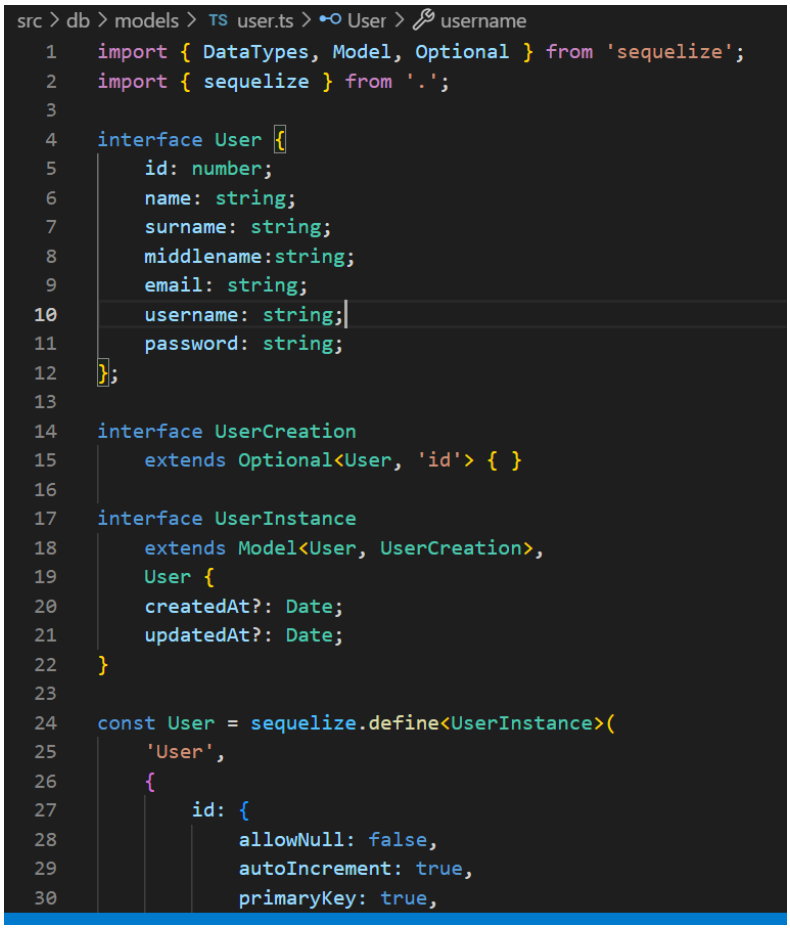
- модели
- контроллеры
- роуты
- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

Файл package.json отвечает за установление зависимостей, создание скриптов и другую информацию о проекте

```
{} package.json > ...
{
  "name": "lw1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "prestart": "npm run build",
    "start": "nodemon dist/index.js",
    "build": "npx tsc",
    "lint": "npx eslint . --ext .ts"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@types/express": "^4.17.17",
    "@types/node": "^18.14.6",
    "@types/node-fetch": "^2.6.2",
    "@typescript-eslint/eslint-plugin": "^5.54.0",
    "@typescript-eslint/parser": "^5.54.0",
    "dotenv": "^16.0.3",
    "eslint": "^8.35.0",
    "nodemon": "^2.0.21",
    "sequelize-cli": "^6.6.0",
    "ts-node": "^10.9.1",
    "tslint": "^6.1.3",
    "typescript": "^4.9.5"
  },
  "dependencies": {
    "express": "^4.18.2",
    "node-fetch": "^3.3.0",
    "sequelize": "^6.29.0",
    "sqlite3": "^5.1.4"
  }
}
```

Модель пользователя – models/user.ts включает в себя следующие поля: id, имя, фамилия, отчество, электронная почта, ник и пароль



```
src > db > models > TS user.ts > User > username
import { DataTypes, Model, Optional } from 'sequelize';
import { sequelize } from '.';

interface User {
    id: number;
    name: string;
    surname: string;
    middlename:string;
    email: string;
    username: string;
    password: string;
};

interface UserCreation
    extends Optional<User, 'id'> { }

interface UserInstance
    extends Model<User, UserCreation>,
    User {
    createdAt?: Date;
    updatedAt?: Date;
}

const User = sequelize.define<UserInstance>(
    'User',
    {
        id: {
            allowNull: false,
            autoIncrement: true,
            primaryKey: true,
            type: DataTypes.INTEGER,
            unique: true,
        },
        name: {
            allowNull: false,
            type: DataTypes.TEXT,
        },
        surname: {
            allowNull: false,
            type: DataTypes.TEXT,
        },
        middlename: {
            allowNull: true,
            type: DataTypes.TEXT,
        },
        email: {
            allowNull: false,
            type: DataTypes.TEXT,
        },
        username: {
            allowNull: false,
            type: DataTypes.TEXT,
        },
        password: {
            allowNull: false,
            type: DataTypes.TEXT,
        },
    }
);
```

Sequelize был использован при работе с sqlite. Файл models/index.ts

src > db > models > TS index.ts > ...

```ts
import { Sequelize } from 'sequelize';

const env = process.env.NODE_ENV || 'development';
const config = require(__dirname + '/../../config.json')[env];

const  sequelize = config.url
  ? new Sequelize(config.url, config)
  : new Sequelize(config.database, config.username, config.password, config);

export { Sequelize, sequelize };
```

CRUD-методы были реализованы через контроллер controllers/index.ts, который позволяет обрабатывать разные запросы пользователя, например, получить пользователя по электронной почте или удалить его.

src > controllers > TS index.ts > UserController > get

```ts
import UserService from "../services"

class UserController {
    private userService: UserService
    constructor() {
        this.userService = new UserService()
    }
    get = async (request: any, response: any) => {
        try {
            const users = await this.userService.getAll()
            return response.json(users);
        } catch (e: any) {
            response.status(404).send({ "error": e.message })
        }
    }
    getbyID = async (request: any, response: any) => {
        try {
            const user = await this.userService.getById(request.params.id)
            return response.json(user);
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
    getbyEmail = async (request: any, response: any) => {
        try {
            const user = await this.userService.getByEmail(request.params.email)
            return response.json(user);
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
```

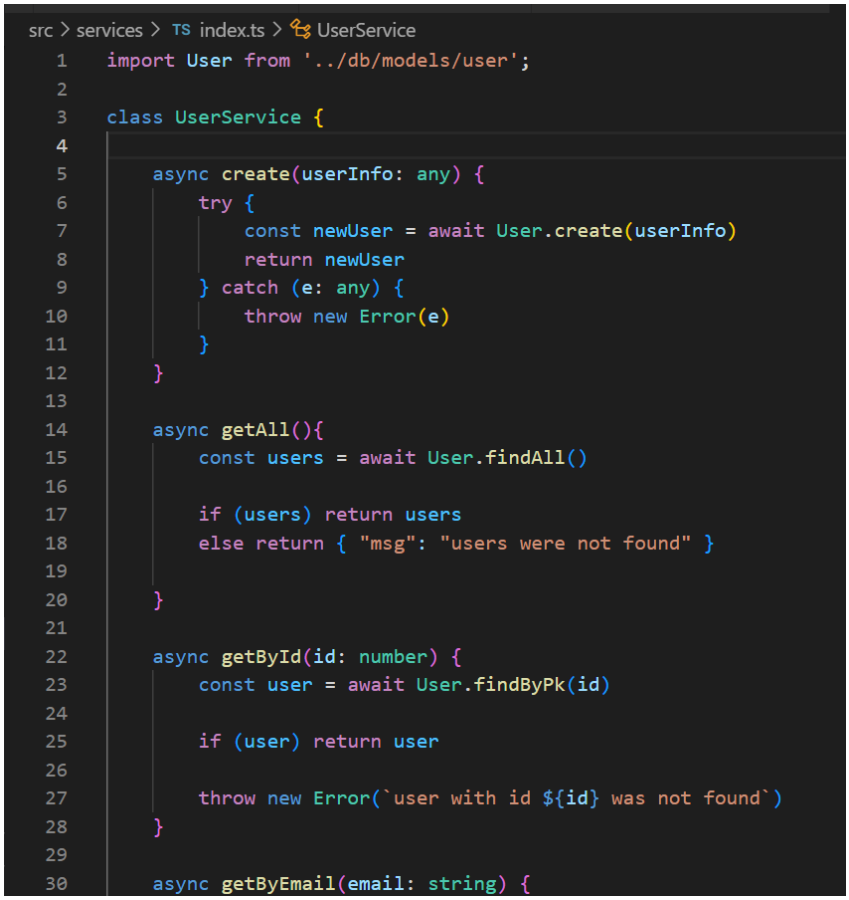
```
    }
    post = async (request: any, response: any) => {
        try {
            const { body } = request
            const user = await this.userService.create(body);
            return response.json({user, msg: "created" })
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }
    put = async (request: any, response: any) => {
        try {
            const { body } = request;
            const user = await this.userService.update(request.params.id, body)
            return response.json({user, msg: 'user was updated' })
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
    delete = async (request: any, response: any) => {
        try {
            const user = await this.userService.delete(request.params.id)
            return response.json({msg: 'user was deleted' })
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
}

export default UserController
```

Роутинг (эндпоинты) был реализован в routes/index.ts и опирается на контроллер

src > routes > TS index.ts > router

```
import express from "express"
import UserController from "../controllers"

const router = express.Router()

const userController = new UserController()

router.route('/users').get(userController.get)

router.route('/users/id/:id').get(userController.getbyID)

router.route('/users/email/:email').get(userController.getbyEmail)

router.route('/users').post(userController.post)

router.route('/users/:id').put(userController.put)

router.route('/users/:id').delete(userController.delete)

export default router
```

Сервисы для работы с моделями в services/index.ts

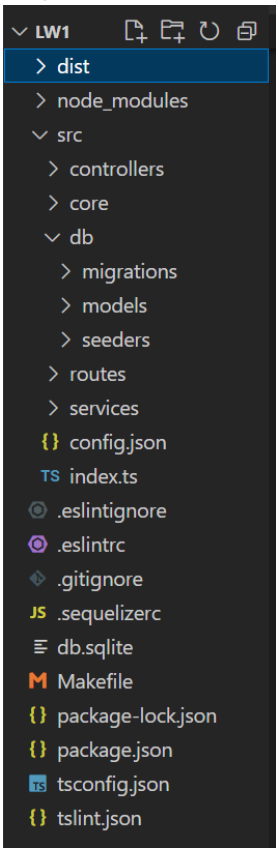


Также благодаря sequelize-cli были проведены миграции (db:migrate) и добавлены в папку migrations, добавлены сидеры (seed:generate) в папку seeders. Именно они позволили добавить пользователей.

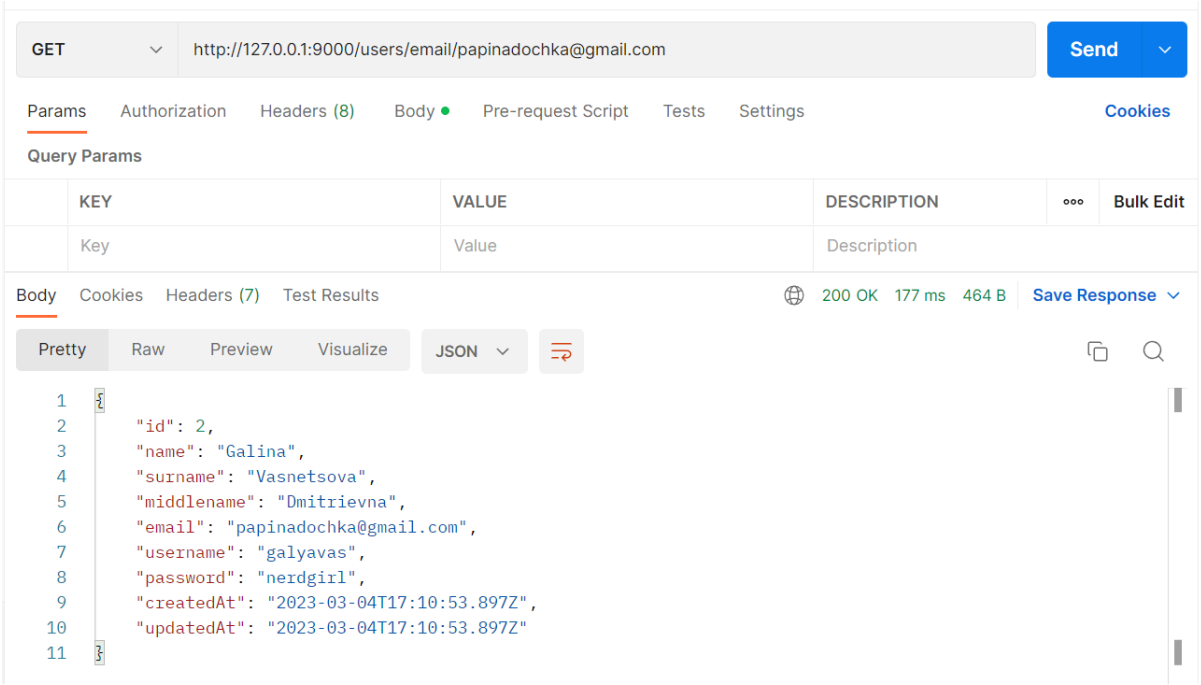
src > db > seeders > JS 20230304165946-user.js > ...

```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.bulkInsert('Users', [{
      name: 'Polina',
      surname: 'Savinova',
      middlename:'Andreevna',
      email: '@polisavgmail.com',
      username: 'polisav',
      password: 'crazygirl',
      createdAt: new Date(),
      updatedAt: new Date()
    },
    {
      name: 'Galina',
      surname: 'Vasnetsova',
      middlename: 'Dmitrievna',
      email: 'papinadochka@gmail.com',
      username: 'galyavas',
      password: 'nerdgirl',
      createdAt: new Date(),
      updatedAt: new Date()
    },
```

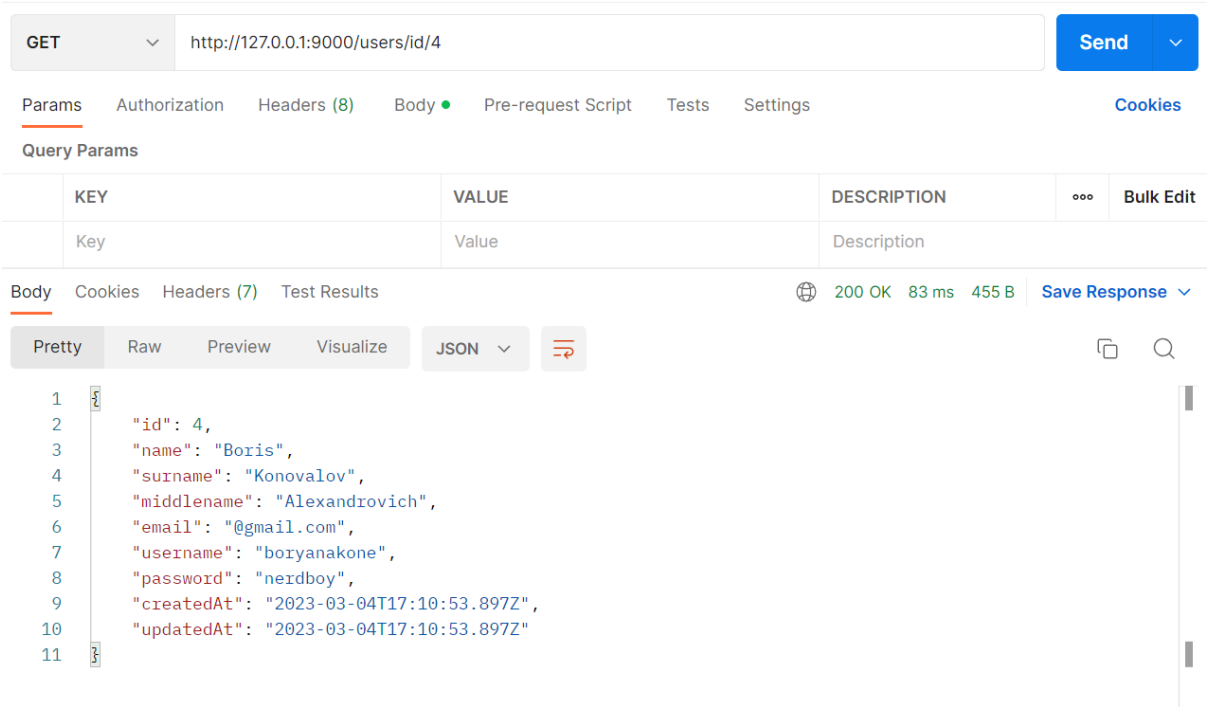
Структура проекта в итоге выглядит следующим образом – папки dist (собранный проект), node_modules, src (рабочая папка) и дополнительные файлы (гитигнор, база данных, установленные пакеты и др.)



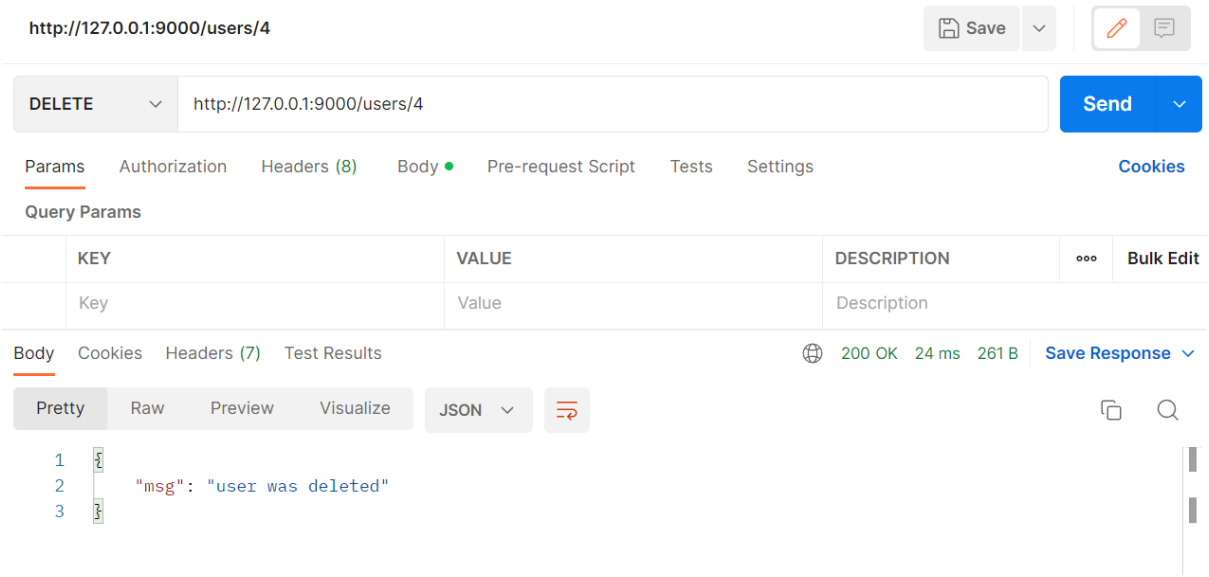
Пример get-запроса по электронной почте



Пример get-запроса по id



Пример delete-запроса



Вывод

Мною были получены навыки разработки boilerplate на express + sequelize + typescript. Этот шаблон проекта позволяет просматривать, добавлять, удалять, изменять данные о пользователях.