

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа 2

Выполнил:

Никитин Павел

Группа К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Продумать свою собственную модель пользователя

Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize

Написать запрос для получения пользователя по id/email

Ход работы

Реализовал задание на typescript с esm модулями, при компиляции файлы переводятся в cjs

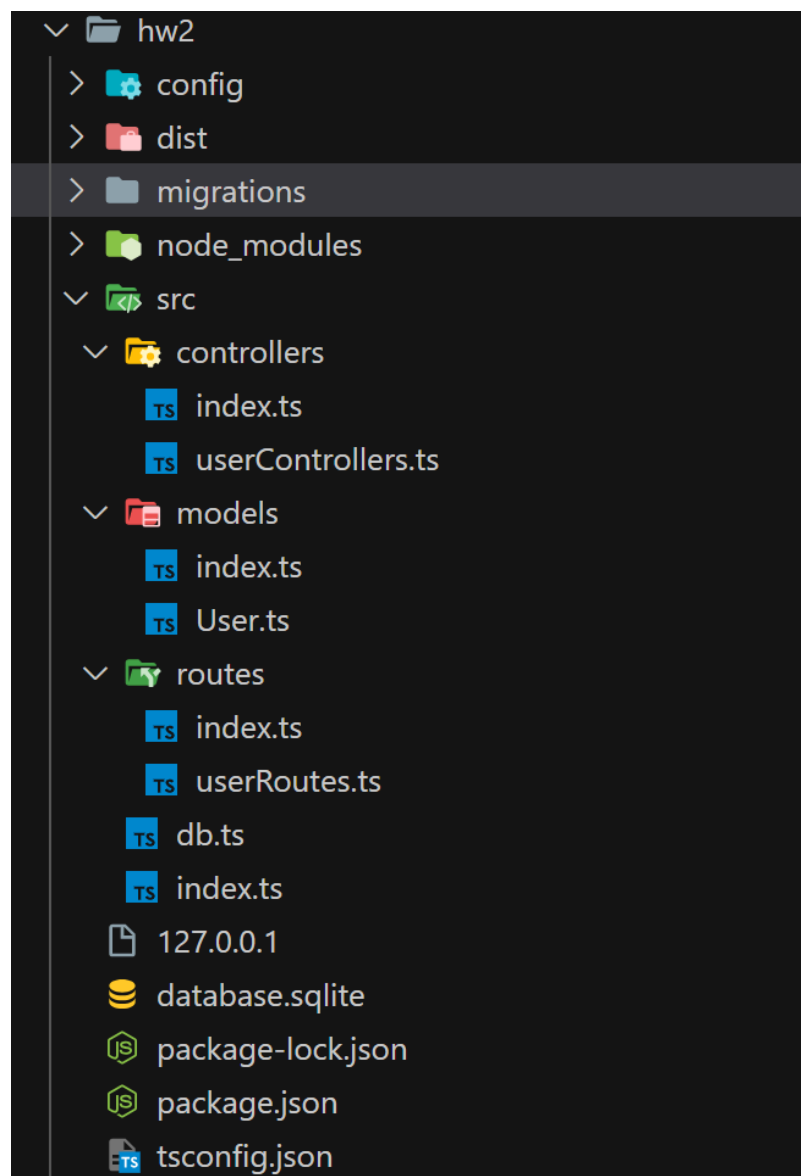


Рисунок 1 - файловая структура

В качестве базы данных использую sqlite3, прописал routes, для запросов к api

```
import express from "express";
import { UserController } from "../controllers";

const router = express.Router();

router.post("/", UserController.createUser);

router.get("/", UserController.getAllUsers);

router.get("/:id", UserController.getUserById);

router.get("/email/:email", UserController.getUserByEmail);

router.put("/:id", UserController.updateUser);

router.delete("/:id", UserController.deleteUser);

export { router };
```

Рисунок 2 - routes

Для каждого из файлов создается index.ts файл, откуда будет импортироваться весь функционал:

```
Nikitin Pavel > hw2 > src > routes > TS index.ts
1  export { router as UserRoutes } from "../userRoutes";
2
```

Рисунок 3 - пример index файла

Так выглядят обработчики api запросов

```
import { Request, Response } from "express";
import { User } from "../models/User";

type ControllerFunction = (req: Request, res: Response) => Promise<void>;

export const createUser: ControllerFunction = async (req, res) => {
  try {
    console.log(req.body);
    const user = await User.create(req.body);
    res.status(200).json(user);
  } catch (err: any) {
    res.status(400).json({ error: err.message });
  }
};

export const getAllUsers: ControllerFunction = async (req, res) => {
  try {
    const users = await User.findAll();
    res.status(200).json(users);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const getUserById: ControllerFunction = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    res.status(200).json(user);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const updateUser: ControllerFunction = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    await user.update(req.body);
    res.status(200).json(user);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const deleteUser: ControllerFunction = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    await user.destroy();
    res.status(200).end();
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const getUserByEmail: ControllerFunction = async (req, res) => {
  try {
    const userEmail = req.params.email;
    const user = await User.findOne({ where: { email: userEmail } });
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    res.status(200).json(user);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};
```

Рисунок 4 - контроллеры

Так выглядит модель пользователя

```
import { DataTypes, Model, ValidationError } from 'sequelize';
import { sequelize } from '../db';

interface UserAttributes {
  id: string;
  email: string;
  username: string;
  password: string;
}

export class User extends Model<UserAttributes> implements UserAttributes {
  public id!: string;
  public email!: string;
  public username!: string;
  public password!: string;
}

User.init(
  {
    id: {
      type: DataTypes.UUID,
      defaultValue: DataTypes.UUIDV4,
      primaryKey: true
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
    username: {
      type: DataTypes.STRING,
      allowNull: false
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false
    }
  },
  {
    sequelize,
    modelName: 'User'
  }
);
```

Рисунок 5 - пример модели

```

Nikitin Pavel > hw2 > package.json > {} devDependencies
1  {
2    "name": "hw2",
3    "version": "1.0.0",
4    "description": "",
5    "main": "src/index.js",
6    "scripts": {
7      "start": "tsc && node dist/index.js",
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "express": "^4.18.3",
14     "sequelize": "^6.37.1",
15     "sqlite3": "^5.1.7"
16   },
17   "devDependencies": {
18     "@types/express": "^4.17.21",
19     "sequelize-cli": "^6.6.2",
20     "typescript": "^5.4.2"
21   }
22 }
23

```

Рисунок 6 - package.json

```

Nikitin Pavel > hw2 > tsconfig.json > {} compilerOptions > esModuleInterop
1  {
2    "compilerOptions": {
3      "module": "commonjs",
4      "target": "ES6",
5      "moduleResolution": "Node",
6      "strict": true,
7      "esModuleInterop": true,
8      "skipLibCheck": true,
9      "outDir": "./dist",
10     "rootDir": "./src",
11     "baseUrl": "."
12   },
13   "include": ["src/**/*.ts"],
14   "exclude": ["node_modules", "**/*.spec.ts"]
15 }
16

```

Рисунок 7 - tsconfig.json

После запуска миграций и создания бд можем проверить работоспособность системы

```
$ curl -X POST -H "Content-Type: application/json" -d '{"email": "k", "username": "s", "password": "d", "st:3000/users"}' http://localhost:3000/users/1
```

Рисунок 8 - создание пользователя

```
NikitinPavel@LAPTOP-FLIG82Q1 MINGW64 ~/Documents/GitHub/ITMO-ICT-Backend-2024/homeworks/K33402 (master)
$ curl http://localhost:3000/users/1
{"id":1,"email":"example@example.com","username":"example","password":"password123","createdAt":"2024-03-17T21:56:28.147Z","updatedAt":"2024-03-17T21:56:28.147Z"}
```

Рисунок 9 - получение пользователя по id

```
NikitinPavel@LAPTOP-FLIG82Q1 MINGW64 ~/Documents/GitHub/ITMO-ICT-Backend-2024/homeworks/K33402 (master)
$ curl http://localhost:3000/users/email/example@example.com
{"id":1,"email":"example@example.com","username":"example","password":"password123","createdAt":"2024-03-17T21:56:28.147Z","updatedAt":"2024-03-17T21:56:28.147Z"}
```

Рисунок 10 - получение пользователя по email

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"email": "new@example.com", "username": "new_username", "password": "new_password"}' http://localhost:3000/users/1
{"id":1,"email":"new@example.com","username":"new_username","password":"new_password","createdAt":"2024-03-17T21:56:28.147Z","updatedAt":"2024-03-17T23:21:25.214Z"}
```

Рисунок 11 - обновление пользователя по id

```
NikitinPavel@LAPTOP-FLIG82Q1 MINGW64 ~/Documents/GitHub/ITMO-ICT-Backend-2024/homeworks/K33402 (master)
$ curl -X DELETE http://localhost:3000/users/1

NikitinPavel@LAPTOP-FLIG82Q1 MINGW64 ~/Documents/GitHub/ITMO-ICT-Backend-2024/homeworks/K33402 (master)
$ curl http://localhost:3000/users/1
{"error":"User not found"}
NikitinPavel@LAPTOP-FLIG82Q1 MINGW64 ~/Documents/GitHub/ITMO-ICT-Backend-2024/homeworks/K33402 (master)
$
```

Рисунок 12 = удаление по id

Вывод

В ходе работы я научился создавать routes в express и работать с базой данных sqlite3, проводить миграции, писать запросы в терминале при помощи curl.