

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бек-энд разработка

Отчет  
Лабораторная работа 1

Выполнила:  
Герулайтите Габриэля  
К33402

Проверил:  
Добряков Д. И.

Санкт-Петербург  
2022 г.

## Задача

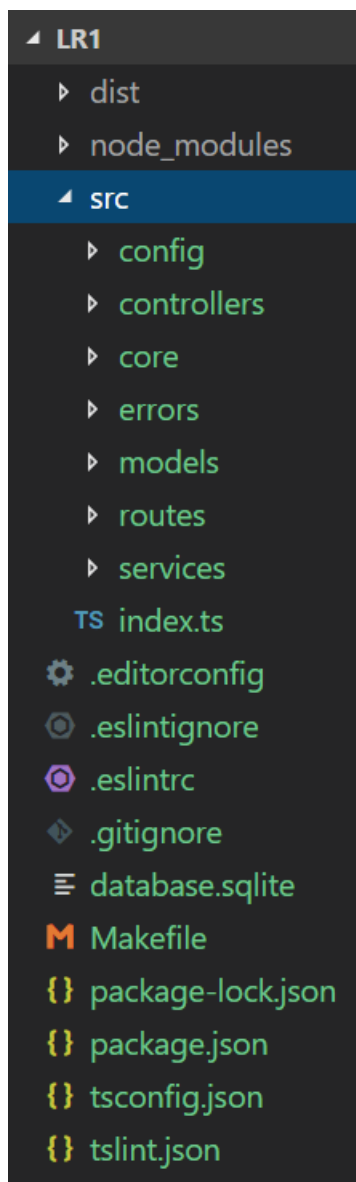
Написать свой boilerplate на express + sequelize + typescript.

Должно быть явное разделение на:

1. Модели
2. Контроллеры
3. Роуты
4. Сервисы для работы с моделями (реализуем паттерн “репозиторий”)

## Ход работы

Структура:



Нужные конфигурации и зависимости отображаемые в package.json

```
1  {
2    "name": "lr1",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "prestart": "npm run build",
8      "start": "nodemon dist/index.js",
9      "build": "npx tsc",
10     "lint": "npx eslint . --ext .ts",
11     "migrate": "npx sequelize-cli db:migrate"
12   },
13   "author": "",
14   "license": "ISC",
15   "devDependencies": {
16     "@types/express": "^4.17.13",
17     "@types/node": "^17.0.31",
18     "@types/uuid": "^8.3.4",
19     "@typescript-eslint/eslint-plugin": "^5.21.0",
20     "@typescript-eslint/parser": "^5.21.0",
21     "eslint": "^8.14.0",
22     "nodemon": "^2.0.15",
23     "sequelize-cli": "^6.4.1",
24     "ts-node": "^10.7.0",
25     "tslint": "^6.1.3",
26     "typescript": "^4.6.3"
27   },
28   "dependencies": {
29     "express": "^4.18.0",
30     "sequelize": "^6.19.0",
31     "sqlite3": "^5.0.6",
32     "uuid": "^8.3.2"
33   }
34 }
```

В качестве ORM я решила использовать sequelize.

Модель пользователя в src/models/index.ts

```

import { DataTypes, Model } from "sequelize"
import db from "../config/database_config";

interface Attributes {
  id: string;
  firstName: string;
  lastName: string;
  email: string;
}

class User extends Model<Attributes> {}

User.init(
  {
    id: {
      type: DataTypes.UUIDV4,
      allowNull: false,
      primaryKey: true
    },
    firstName: {
      type: DataTypes.STRING,
      allowNull: false
    },
    lastName: {
      type: DataTypes.STRING,
      allowNull: false
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
  },
  {
    sequelize: db,
  }
)

export default User

```

src/config/database\_config.ts

```

1  import { Sequelize } from 'sequelize';
2
3  const db = new Sequelize('app', '', '', {
4    storage: './database.sqlite',
5    dialect: 'sqlite',
6    logging: false,
7  });
8
9  export default db;

```

В директории controllers созданы контроллеры, использующие UserService для работы с пользователями – создание, вывод всех пользователей, удаление и обновление.

Src/controllers/user.ts

```
1  import User from "../models/index"
2  import { v4 as uuidv4 } from "uuid"
3  import UserService from "../services/user"
4  import router from "../routes/index"
5
6  class UserController {
7    private userService: UserService
8
9    constructor() {
10     this.userService = new UserService()
11   }
12
13   get = async (request: any, response: any) => {
14     try {
15       const records = await this.userService.listUsers()
16       return response.json(records);
17     } catch (error) {
18       response.status(404).send({ "error": 404 })
19     }
20   }
21
22   post = async (request: any, response: any) => {
23     const id = uuidv4()
24     console.log(request)
25     try {
26       const record = await this.userService.create({ ...request.body, id })
27       return response.json({ record, msg: 'Successfully create user' })
28     } catch (error) {
29       response.status(400).send({ "error": 404 })
30     }
31   }
32
33   getbyID = async (request: any, response: any) => {
34     try {
35       const record = await this.userService.getById(request.params.id)
36       return response.json(record);
37     } catch (error) {
38       response.status(404).send({ "error": 404 })
39     }
40   }
41
42   put = async (request: any, response: any) => {
43     try {
44       const record = await this.userService.updateUser(request.params.id, request.body)
45       return response.json({record, msg: 'Successfully update user' })
46     } catch (error) {
47       response.status(404).send({ "error": 404 })
48     }
49   }
50
51   delete = async (request: any, response: any) => {
52     try {
53       const record = await this.userService.deleteUser(request.params.id)
54       return response.json({msg: 'Successfully delete user' })
55     } catch (error) {
56       response.status(404).send({ "error": 404 })
57     }
58   }
59
60   }
61
62   export default UserController
```

Сервисы прописаны в src/services/user.ts

```
1  import { userInfo } from "os"
2  import UserError from "../errors/users/user"
3  import User from "../models/index"
4
5
6  class UserService {
7    async getById(id: string){
8      const user = await User.findByPk(id)
9
10     if (user) return user.toJSON()
11
12     throw new UserError('Not found!')
13   }
14 }
15
16 async create(user: any): Promise<User|Error>{
17   try {
18     const userData = await User.create(user)
19     return userData
20   } catch (e: any) {
21     const errors = e.errors.map((error: any) => error.message)
22     throw new UserError(errors)
23   }
24 }
25
26
27 async listUsers(){
28   const users = await User.findAll()
29
30   if (users) return users
31
32   throw new UserError('Not found!')
33 }
34
```

```
35   async updateUser(id:string, data: any) {
36     try {
37       const user = await User.findByPk(id)
38       if (user) {
39         const result = await (user.update(data))
40       }
41       return user
42     } catch (e: any) {
43       const errors = e.errors.map((error: any) => error.message)
44
45       throw new UserError(errors)
46     }
47   }
48
49   async deleteUser(id:string) {
50     try {
51       await User.destroy({where: {id:id}})
52     } catch (e: any) {
53       const errors = e.errors.map((error: any) => error.message)
54
55       throw new UserError(errors)
56     }
57   }
58 }
59
60
61 export default UserService
```

Все пути прописаны в роутере src/routes/index.ts

```
1 import express from "express"
2 import Controller from '../controllers/user'
3 import { Router } from 'express'
4
5
6 const router: express.Router = express.Router()
7
8 const controller = new Controller()
9
10 router.route('/read')
11   .get(controller.get)
12
13 router.route('/create')
14   .post(controller.post)
15
16 router.route('/user/:id')
17   .get(controller.getbyID)
18
19 router.route('/update/:id')
20   .put(controller.put)
21
22 router.route('/delete/:id')
23   .delete(controller.delete)
24
25 export default router
```

Проверим работу:

## Создание пользователя

The screenshot shows a REST client interface with a POST request to `http://localhost:8000/v1/create`. The request body is a JSON object with the following fields: `firstName` (Amy), `lastName` (Hay), and `email` (test3@gmail.com). The response status is 200 OK, and the response body is a JSON object with a `record` field containing the created user's details (including an ID and timestamps) and a `msg` field with the value "Successfully create user".

```
POST http://localhost:8000/v1/create

{
  "firstName": "Amy",
  "lastName": "Hay",
  "email": "test3@gmail.com"
}
```

```
{
  "record": {
    "firstName": "Amy",
    "lastName": "Hay",
    "email": "test3@gmail.com",
    "id": "6d891172-5c06-4fa5-adbc-7e6efaa42426",
    "updatedAt": "2022-05-17T15:10:35.058Z",
    "createdAt": "2022-05-17T15:10:35.058Z"
  },
  "msg": "Successfully create user"
}
```

## Вывод списка всех пользователей

GET http://localhost:8000/v1/read Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 [
2   ... "firstName": "Amy",
3   ... "lastName": "Hay",
4   ... "email": "test3@gmail.com"
5 ]
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 15 ms Size: 800 B Save Response

Pretty Raw Preview Visualize **JSON** Copy Search

```
14   "email": "test2@gmail.com",
15   "createdAt": "2022-05-14T03:47:55.813Z",
16   "updatedAt": "2022-05-17T14:37:34.342Z"
17 },
18 {
19   "id": "6d891172-5c06-4fa5-adbc-7e6efaa42426",
20   "firstName": "Amy",
21   "lastName": "Hay",
22   "email": "test3@gmail.com",
23   "createdAt": "2022-05-17T15:10:35.058Z",
24   "updatedAt": "2022-05-17T15:10:35.058Z"
25 }
26 ]
```

## Обновление информации пользователя

PUT http://localhost:8000/v1/update/6d891172-5c06-4fa5-adbc-7e6efaa42426 Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 [
2   ... "firstName": "Amy",
3   ... "lastName": "Haaaaaaaay",
4   ... "email": "test3@gmail.com"
5 ]
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 177 ms Size: 471 B Save Response

Pretty Raw Preview Visualize **JSON** Copy Search

```
1 {
2   "record": {
3     "id": "6d891172-5c06-4fa5-adbc-7e6efaa42426",
4     "firstName": "Amy",
5     "lastName": "Haaaaaaaay",
6     "email": "test3@gmail.com",
7     "createdAt": "2022-05-17T15:10:35.058Z",
8     "updatedAt": "2022-05-17T15:15:12.118Z"
9   },
10   "msg": "Successfully update user"
11 }
```



## Удаление пользователя

The screenshot displays a Postman interface for a DELETE request. The URL bar shows the endpoint: `http://localhost:8000/v1/delete/6d891172-5c06-4fa5-adbc-7e6efaa42426`. The 'Body' tab is selected, showing a JSON payload: 

```
{  "firstName": "Amy",  "lastName": "Haaaaaaaay",  "email": "test3@gmail.com"}
```

. The status bar at the bottom indicates a successful response: `Status: 200 OK`, `Time: 181 ms`, and `Size: 269 B`. The response body, shown in the 'Body' tab, is: 

```
{  "msg": "Successfully delete user"}
```

.

## Выводы:

В ходе лабораторной работы был написан свой boilerplate на express, sequelize и typescript. Проверена работа CRUD-методов в postman.