

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэкенд разработка

Отчет

Лабораторная работа 1

Выполнил:
Омар Сизей

Группа: К33412

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

В ходе работы был создан класс моделей:

```
1  import { Table, Column, Model } from 'sequelize-typescript'
2
3  @Table
4  export class Person extends Model {
5      @Column
6      name: string
7
8      @Column
9      surname: string
10
11     @Column
12     gmail: number
13 }
```

Использование моделей конфигурируется в config.ts:

```

config > ts config.ts > ...
1  import { Sequelize } from 'sequelize-typescript'
2  import { Person } from '../models/Person'
3
4  export const sequelize = new Sequelize({
5    database: 'example_db',
6    dialect: 'sqlite',
7    username: 'root',
8    password: '',
9    storage: ':memory:',
10   models: [Person],
11   repositoryMode: true
12 })

```

Запросы к БД реализуются через класс Сервис:

```

services > ts index.ts > ...
1  import { Person } from '../models/Person'
2  import { sequelize } from '../config/config'
3
4  export class MainService {
5
6    private repo = sequelize.getRepository(Person)
7
8    add(name_par: string, surname_par: string, gmail_par: number) {
9      this.repo.create({ name: name_par, surname: surname_par, gmail: gmail_par })
10    }
11
12    get() {
13      const data = this.repo.findAll()
14      return data
15    }
16  }

```

С сервисом взаимодействует Контроллер:

```

1  import { MainService } from '../services/index'
2
3  class ExampleController {
4
5      private service = new MainService()
6
7      post = async (request: any, response: any) => {
8          try {
9              const body = request.body
10             await this.service.add(body.name, body.surname, body.gmail)
11             response.send('Added')
12         } catch (error: any) {
13             response.status(400).send(error.message)
14         }
15     }
16
17     get = async (request: any, response: any) => {
18         try {
19             const data = await this.service.get()
20             response.send(data)
21         } catch (error: any) {
22             response.status(400).send(error.message)
23         }
24     }
25 }
26
27 export default ExampleController

```

Роуты:

```

routes > TS index.ts > ...
1  import express from "express"
2  import {
3      (alias) function express(): core.Express
4      (alias) namespace express
5      import express
6      Creates an Express application. The express() function is a top-level function
7      exported by the express module.
8  router
9      .route('/')
10     .get(exampleController.get)
11     .post(exampleController.post)
12
13 export default router

```

Получившийся Makefile:

```
.PHONY: init
init:
    npm i
    npm run build

.PHONY: run
run:
    npm start

.PHONY: migrate
migrate:
    npx sequelize-cli db:migrate

.DEFAULT_GOAL := init
```

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8000/user
- Body:** A JSON object with the following fields:

```
{  "name": "CR&",  "surname": "1234users",  "gmail": "cr7@gmail.com"}
```
- Response:** 200 OK, 462 ms, 231 B. The response body is "Added".

GET

http://localhost:8000/user

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (7)

Test Results

200 OK109 ms383 BSave Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

{

"id": 1,

"name": "CR&",

"surname": "1234users",

"email": "cr7@gmail.com",

"createdAt": "2023-06-13T13:15:42.522Z",

"updatedAt": "2023-06-13T13:15:42.522Z"

}

Activate Window
Go to Settings to a

Вывод

В этой работе был создан бойлерплейт на express, sequelize и typescript. Получившийся код разделен на логические части – модели, контроллеры, роуты, сервис.