САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Дао Куанг Ань

Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

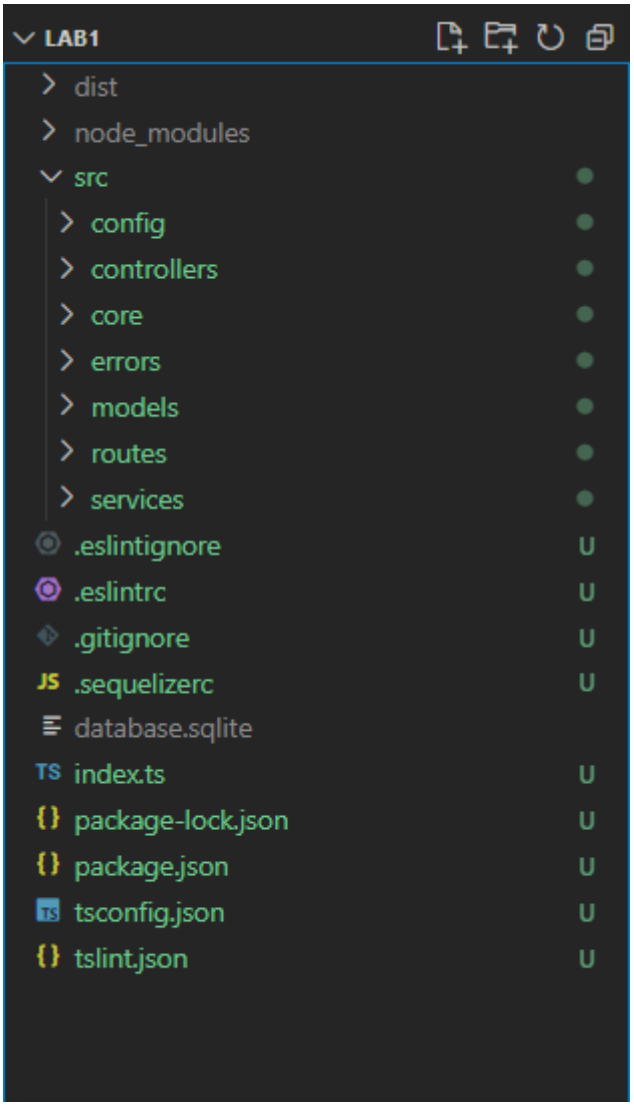
Задача

- Нужно написать свой boilerplate на express + sequelize + typescript.
- Должно быть явное разделение на:

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

Ход работы

1.Tree



2. controllers/index.ts

```
import User from "../models/index"
import { v4 as uuidv4 } from "uuid"
import UserService from "../services/user"


class Controller {
    private userService: UserService

    constructor() {
        this.userService = new UserService()
    }

    get = async (request: any, response: any) => {
        try {
            const records = await this.userService.listUsers()
            return response.json(records);
        } catch (e) {
             return response.json({ msg: 'fail to read', status: 500, route:
'/read'})
        }
    }
    post = async (request: any, response: any) => {
        const id = uuidv4()
        try {
             const record = await this.userService.create({ ...request.body,
id})
            return response.json({ record, msg: 'Successfully create user' })
        } catch (e) {
             return response.json({msg: "fail to create", status: 500, route:
'/create'})
        }
    }
```

```
    getbyID = async (request: any, response: any) => {

        try {

            const record = await this.userService.getById( request.params.id)

            return response.json(record);

        } catch (e) {

             return response.json({ msg: 'fail to read', status: 500, route:
'/user/:id'})

        }

    }

    put = async (request: any, response: any) => {

        try {

                                                const    record    =    await
this.userService.updateUser(request.params.id, request.body)

            return response.json({record, msg: 'Successfully update user' })

        } catch (e) {

             return response.json({msg: "fail to update", status: 500, route:
'/update'})

        }

    }

    delete = async (request: any, response: any) => {

        try {

                                                const    record    =    await
this.userService.deleteUser(request.params.id)

            return response.json({msg: 'Successfully deleted user' })

        } catch (e) {

             return response.json({msg: "fail to delete", status: 500, route:
'/delete'})

        }

    }

}



export default Controller
```

3. core/index.ts

```
import express from "express"

import { createServer, Server } from "http"

import routes from "../routes/index"

import db from '../config/config'

class App {

    public port: number

    public host: string


    private app: express.Application

    private server: Server


    constructor(port = 8000, host = "localhost") {

        this.port = port

        this.host = host


        this.app = this.createApp()

        this.server = this.createServer()

    }


    private createApp(): express.Application {

        const app = express()

        app.use(express.json())

        app.use('/v1', routes)


        return app

      }


    private createServer(): Server {

        const server = createServer(this.app)
```

```
        return server
    }



    public start(): void {
        db.sync().then(() => {
            this.server.listen(this.port, () => {
                console.log(`Connect to db`)
                console.log(`Running server on port ${this.port}`)
            })
        })

    }
}

export default App
```

4. models/index.ts

```
import { DataTypes, Model } from "sequelize"
import db from "../config/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,
        tableName: "todos"
    }
)

export default User
```

5.routes/index.ts

```
import express from "express"
import Controller from '../controllers/index'

const router: express.Router = express.Router()

const controller = new Controller()

router.route('/read')
  .get(controller.get)

router.route('/create')
  .post(controller.post)

router.route('/user/:id')
  .get(controller.getbyID)

router.route('/update/:id')
  .put(controller.put)

router.route('/delete/:id')
  .delete(controller.delete)
export default router
```

6.services/user.ts

```
import { userInfo } from "os"
import UserError from "../errors/users/user"
import User from "../models/index"



class UserService {
    async getById(id: string){
        const user = await User.findByPk(id)

        if (user) return user.toJSON()

        throw Error("getbyID service not working")

    }

    async create(user: any): Promise<User|Error>{
        try {
            const userData = await User.create(user)
            return userData
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new UserError(errors)
        }

    }

    async listUsers(){
        const users = await User.findAll()
```

```
        if (users) return users


        throw Error("listUsers service not working")
    }


    async updateUser(id:string, data: any) {
        try {
            const user = await User.findByPk(id)
            if (user) {
                user.update(data)
            }
            return user
        } catch {
            throw Error("updateUser service not working")
        }
    }


    async deleteUser(id:string) {
        try {
            await User.destroy({where: {id:id}})
        } catch {
            throw Error("updateUser service not working")
        }
    }


}


export default UserService
```

Вывод

- Написал свой boilerplate на express + sequelize + typescript.
- Были разделены директория:
    - модели
    - контроллеры
    - роуты
    - сервисы для работы с моделями