САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Беллазрег Анис

К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

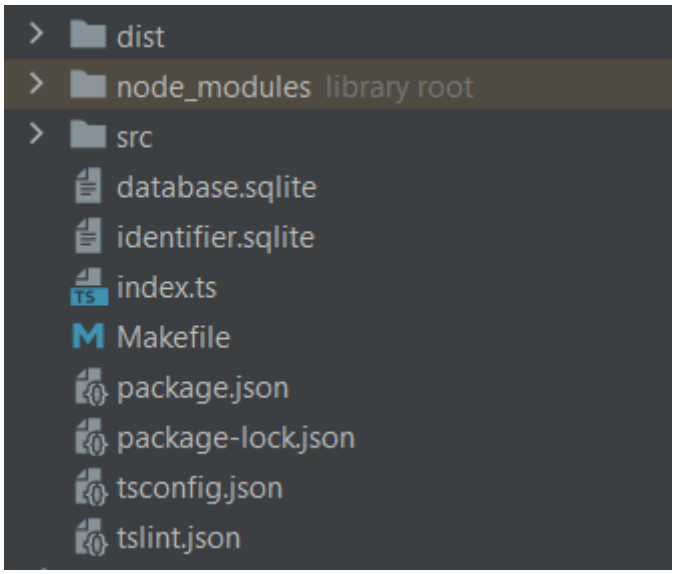
2023 г.

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

```ts
import { v4 as uuidv4 } from "uuid"
import UserService from "../services/user"

class UserController {
    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 (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
    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 (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }
```

```
    getbyID = async (request: any, response: any) => {
        try {
            const record = await this.userService.getById(request.params.id)
            return response.json(record);
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
    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 (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
    delete = async (request: any, response: any) => {
        try {
            const record = await this.userService.deleteUser(request.params.id)
            return response.json({msg: 'Successfully delete user' })
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
}

export default UserController
```

3. core/index.ts

```
import db from '../config/config'
class App {
    public port: number
    public host: string

    private app: express.Application
    private server: Server

    constructor(port :number = 8000, host :string = "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, listeningListener: () => {
                console.log(`Connect to database`)
                console.log(`Running server on port ${this.port}`)
            })
        })

    }
}

export default App
```

1.models/index.ts

```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(
     attributes: {
        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
          },
    },
     options: {
        sequelize:db,
        tableName: "todos"
    }
)

export default User
```

2.routes/index.ts

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

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

const controller = new Controller()

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

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

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

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

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

3.services/user.ts

```
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 new UserError('Not found!')

    }

    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 new UserError('Not found!')
    }

    async updateUser(id:string, data: any) {
        try {
            const user = await User.findByPk(id)
            if (user) {
                user.update(data)
            }
            return user
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new UserError(errors)
        }
    }

    async deleteUser(id:string) {
        try {
            await User.destroy( options: {where: {id:id}})
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new UserError(errors)
        }
    }

}

export default UserService
```

Вывод

В ходе лабораторной работе я написал свой boilerplate на express + sequelize + typescript.

Директория были разделены на:

- Модели;
- Контроллеры;
- Роуты;
- Сервисы для работы с моделями.