САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Рыбалко Олег

Группа K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 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”)

Составьте Makefile, который будет автоматизировать ваши рутинные действия, такие как:

- запуск приложения;
- установка зависимостей и сборка приложения.

Ход работы

1. Инициализируем модуль

   npm init

2. Установим зависимости

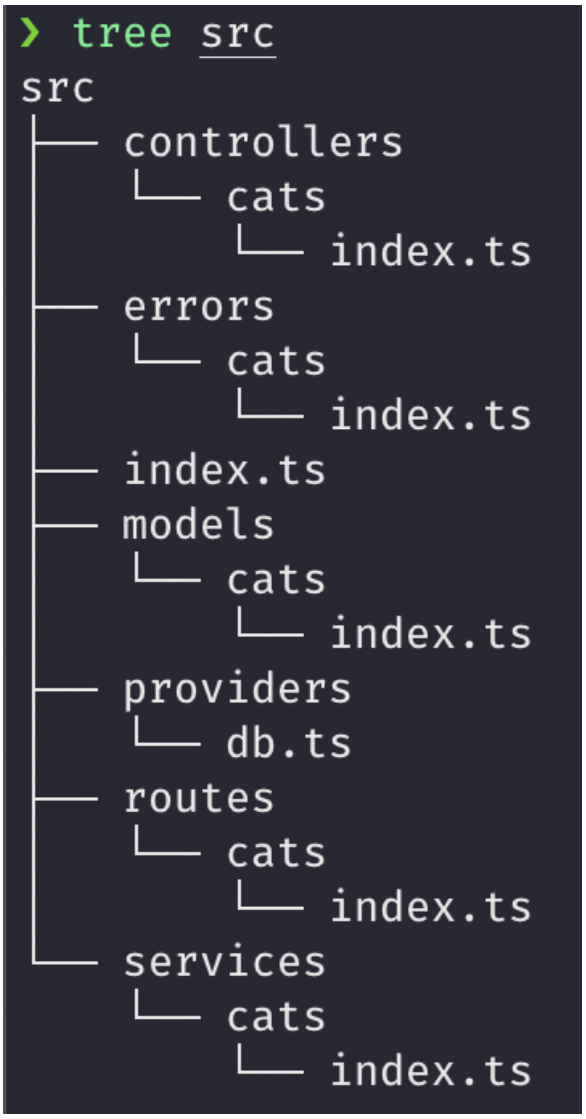
   npm i dotenv express sequelize-typescript sqlite3 @types/express

3. В файле package.json укажем type: module

```
{
  "name": "lab1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  ▷ Debug
  "scripts": {
    "build": "tsc",
    "start": "tsc && node dist/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "typescript": "^5.4.3"
  },
  "dependencies": {
    "@types/express": "^4.17.21",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "sequelize-typescript": "^2.1.6",
    "sqlite3": "^5.1.7"
  }
}
```

4. Создадим файл tsconfig.json

5. Создадим следующую структуру проекта.
   Бизнес-логика разделена на контроллеры, модели, роутеры и сервисы. Также есть директория для хранения ошибок и провайдеров (например для базы данных)



6. В файле src/index.ts создадим express приложение, подключим необходимые роутеры и запустим сервер на порту, указанном в файле .env

```
import express from 'express'
import catsRouter from './routes/cats/index.js'
import sequelize from './providers/db.js'
import dotenv from 'dotenv'

dotenv.config()
const app = express()
app.use(express.json())
app.use('/cats', catsRouter)

app.listen(process.env.PORT, () => {
  sequelize // to not delete after compilation
  console.log(`Listening on port ${process.env.PORT}`)
})
```

7. Создадим контроллер. В классе контроллера будет храниться объект сервиса, который будет использован для получения данных из базы.

```
import { Request, Response } from 'express'
import { Cat } from '../../models/cats/index.js'
import { CatsService } from '../../services/cats/index.js'

export class CatsController {
  service: CatsService

  constructor() {
    this.service = new CatsService()
  }

  get = async (req: Request, res: Response) => {
    try {
      res.send(this.service.get((req.body as Cat).identifier))
    } catch {
      res
        .status(404)
        .send({ error: 'cat with the specified identifier was not found' })
    }
  }
  post = async (req: Request, res: Response) => {
    try {
      res.send(this.service.create(req.body as Cat))
    } catch {
      res.status(400).send({ error: 'invalid data specified' })
    }
  }
}
```

8. Создадим сервис для получения кошек из базы данных при помощи sequelize. Для начала определим два метода для добавления кошки в базу и для получения модели по идентификатору

```
import { CatCreationError, CatNotFound } from '../../errors/cats/index.js'
import { Cat } from '../../models/cats/index.js'

export class CatsService {
  async create(catData: any): Promise<Cat> {
    try {
      return (await Cat.create(catData)).toJSON()
    } catch {
      throw new CatCreationError('failed to create a cat')
    }
  }

  async get(identifier: number): Promise<Cat> {
    try {
      return await Cat.findByPk(identifier)
    } catch {
      throw new CatNotFound(
        `cat with the identifier ${identifier} was not found}`
      )
    }
  }
}
```

9. Для создания роутера необходимо проинициализировать модель контроллера и подключить методы к роутеру

```
import { Router } from 'express'
import { CatsController } from '../../controllers/cats/index.js'

const router = Router()
const controller = new CatsController()

router.get('/', controller.get)
router.post('/', controller.post)

export default router
```

10. Создадим sequelize модель для кошки

```
import {
  Table,
  Column,
  Model,
  Unique,
  PrimaryKey,
  AutoIncrement,
} from 'sequelize-typescript'

@Table
export class Cat extends Model {
  @Unique
  @PrimaryKey
  @AutoIncrement
  @Column
  identifier: number

  @Column
  name: string

  @Column
  breed: string
}
```

Вывод

В данной лабораторной работе удалось написать boilerplate проект с использованием typescript + sequelize + express. Полученный проект можно использовать для создания следующих проектов с таким же стеком.