САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Таякин Даниил

Группа К33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

Должно быть явное разделение на:

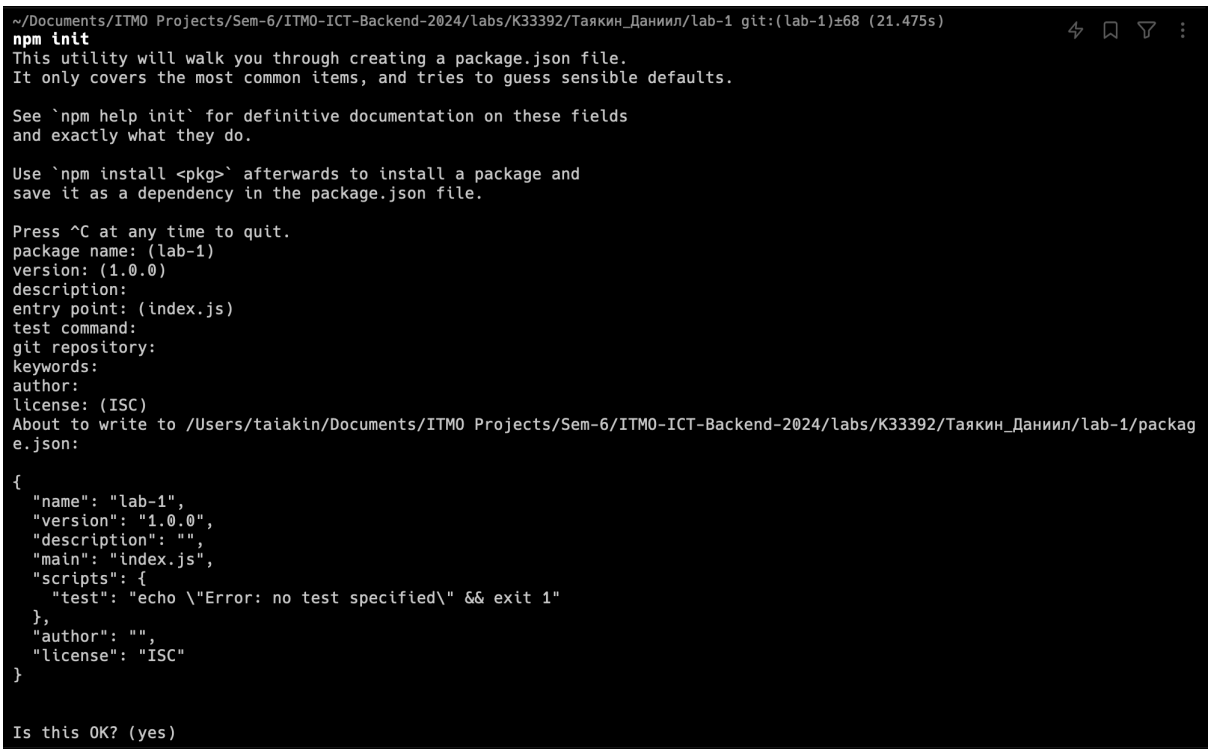
1. модели
2. контроллеры
3. роуты
4. сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Пример: https://github.com/kantegory/express-sequelize-boilerplate

Другие примеры можно поискать на github, набрав в поиске: "express boilerplate".

Ход работы

1. Инициализируем проект.



2. Устанавливаем зависимости.

```
~/Documents/ITMO Projects/Sem-6/ITMO-ICT-Backend-2024/labs/K33392/Таякин_Даниил/lab-1 git:(lab-1)±69 (6.995s)
npm i dotenv express sequelize-typescript sqlite3 @types/express
npm WARN deprecated @npmcli/move-file@1.1.2: This functionality has been moved to @npmcli/fs

added 222 packages, and audited 223 packages in 7s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

3. В package.json указываем тип module.

```
{
  "name": "lab-1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@types/express": "^4.17.21",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "sequelize-typescript": "^2.1.6",
    "sqlite3": "^5.1.7"
  }
}
```

4. Добавим TypeScript в проект.

```
~/Documents/ITMO Projects/Sem-6/ITMO-ICT-Backend-2024/labs/K33392/Таякин_Даниил/lab-1 git:(lab-1)±69 (1.411s)
npm install typescript --save-dev

added 1 package, and audited 224 packages in 1s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

5. Добавим tsconfig.json со следующим содержимым.

```json
{
    "compilerOptions": {
        "module": "NodeNext",
        "moduleResolution": "NodeNext",
        "target": "ES2020",
        "sourceMap": true,
        "outDir": "dist",
        "experimentalDecorators": true,
        "emitDecoratorMetadata": true
    },
    "include": ["src/**/*"]
}
```

6. Создадим структуру проекта.

   На рисунке ниже показана структура, которая делится на контроллеры, модели, роуты и сервисы. Также есть директории для ошибок и провайдеров (например, хранить базу данных).

```
~/Documents/ITMO Projects/Sem-6/ITMO-ICT-Backend-2024/
tree src
src
├── controllers
│   └── dogs
│       └── Dog.ts
├── errors
│   └── dogs
│       └── Dog.ts
├── index.ts
├── models
│   └── dogs
│       └── Dog.ts
├── providers
│   └── db.ts
├── routes
│   └── dogs
│       └── Dog.ts
└── services
    └── dogs
        └── Dog.ts

11 directories, 7 files
```

7. В src/index.ts создаем express приложение, в котором подключаем нужные роуты и запускаем сервер на порту, который указан в .env

файле.

```
import express from 'express'
import dogsRouter from './routes/dogs/Dog.js'
import sequelize from './providers/db.js'
import dotenv from 'dotenv'

dotenv.config()
const app = express()
app.use(express.json())
app.use('/dogs', dogsRouter)

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

8. Создаем контроллер в котором будет храниться сервис для получения данных из базы.

```
import { Request, Response } from 'express'
import { Dog } from '../../models/dogs/Dog.js'
import { DogsService } from '../../services/dogs/Dog.js'

export class DogsController {
  service: DogsService

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

  get = async (req: Request, res: Response) => {
    try {
      const dog: Dog = await this.service.get(
        Number(req.params.id)
      )

      res.send(dog)
    } catch {
      res.status(404).send({ error: 'Dog with the specified id was not found' })
    }
  }
  post = async (req: Request, res: Response) => {
    try {
      const dog: Dog = await this.service.create(req.body)
      res.status(201).send(dog)
    } catch {
      res.status(400).send({ error: 'Invalid data specified' })
    }
  }
}
```

9. Создаем сервис для получения собак из базы при помощи sequelize. Определяем два метода get и post для получения и создания собак

соответственно.

```
import { DogCreationError, DogNotFound } from '../../errors/dogs/Dog.js'
import { Dog } from '../../models/dogs/Dog.js'

export class DogsService {
  async create(data: any): Promise<Dog> {
    try {
      const dog = await Dog.create(data)

      return dog.toJSON()
    } catch (e: any) {
      const errors = e.errors.map((error: any) => error.message)

      throw new DogCreationError(errors)
    }
  }

  async get(id: number): Promise<Dog> {
    const dog = await Dog.findByPk(id)

    if (dog) return dog.toJSON()

    throw new DogNotFound(`Dog with id ${id} was not found`)
  }
}
```

10. Создаем роут и инициализируем контроллер для подключения методов к роуту.

```
import { Router } from 'express'
import { DogsController } from '../../controllers/dogs/Dog.js'

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

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

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

export default router
```

11. Создаем sequelize модель Dog

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

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

    @Column
    name: string

    @Column
    breed: string
}
```

Вывод

В данной лабораторной работе был написан boilerplate проект с использованием typescript, sequelize, express. Далее, можно использовать этот проект для создания подобных проектов.