САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчёт

Лабораторная работа №1 «Написание своего boilerplate»

Выполнил:

Шугинин Юрий

К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

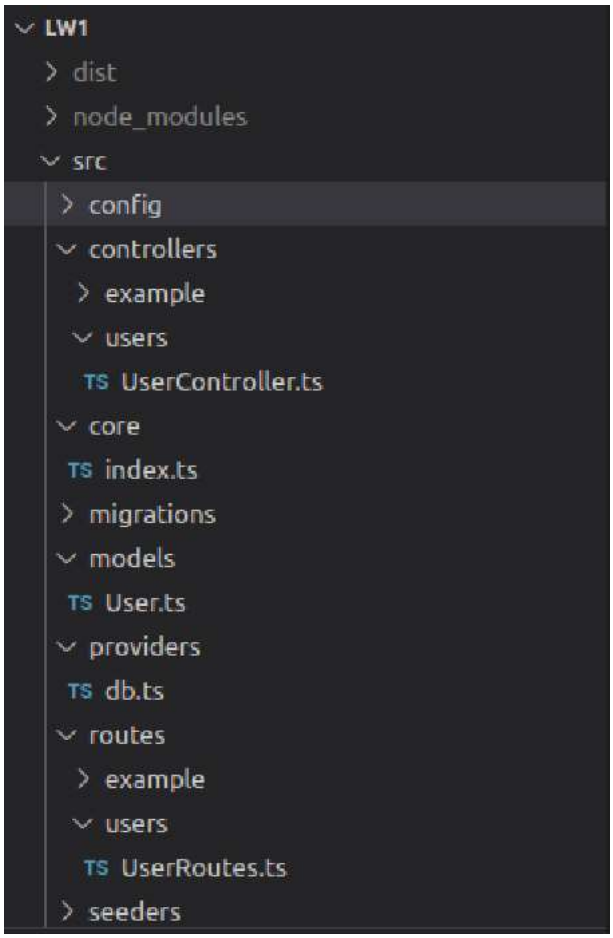
Написать свой boilerplate на Express + Sequelize + TypeScript.

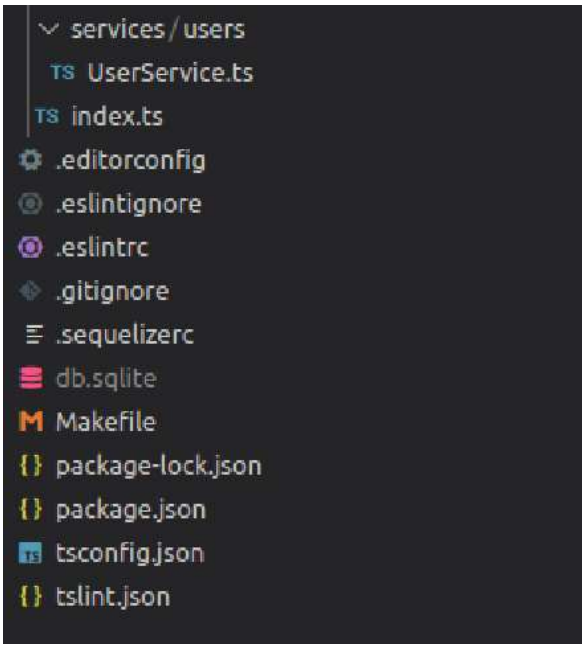
Должно быть явное разделение на:

- модели
- контроллеры;
- роуты;
- сервисы для работы с моделями (паттерн «Репозиторий»).

Ход работы

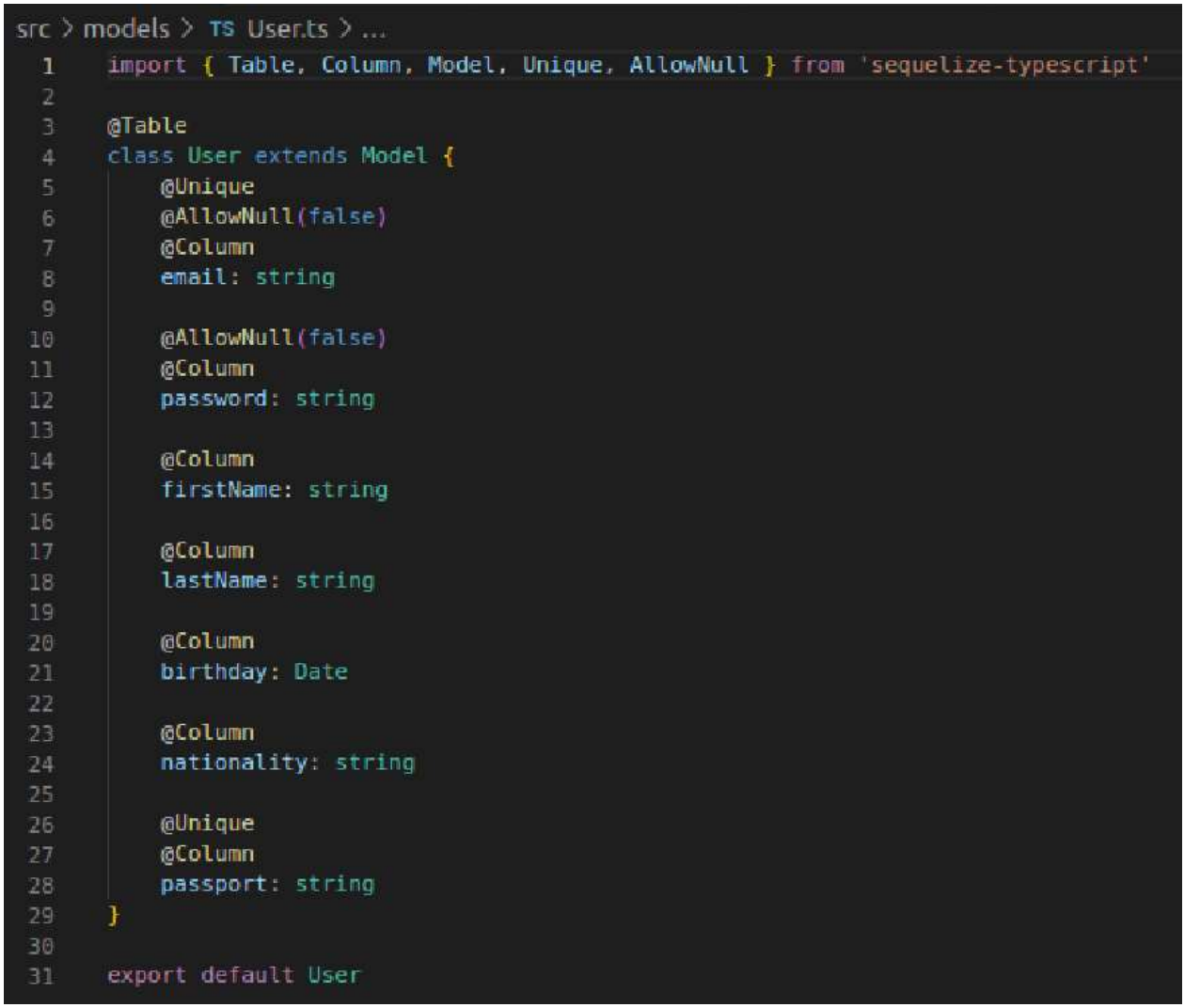
1) Структура проекта





2) Модель пользователя (файл “./src/models/User.ts”)

Для работы с ORM Sequelize на языке TypeScript была выбрана библиотека sequelize-typescript. Она обладает лаконичным и легко доступным в освоении синтаксисом описания модели. Отрицательной стороной библиотеки является отсутствие автоматической генерации файлов миграций, однако их можно написать самостоятельно, следуя документации.



3) Сервис для работы с моделью пользователя (файл “./src/services/users/UserService.ts”)

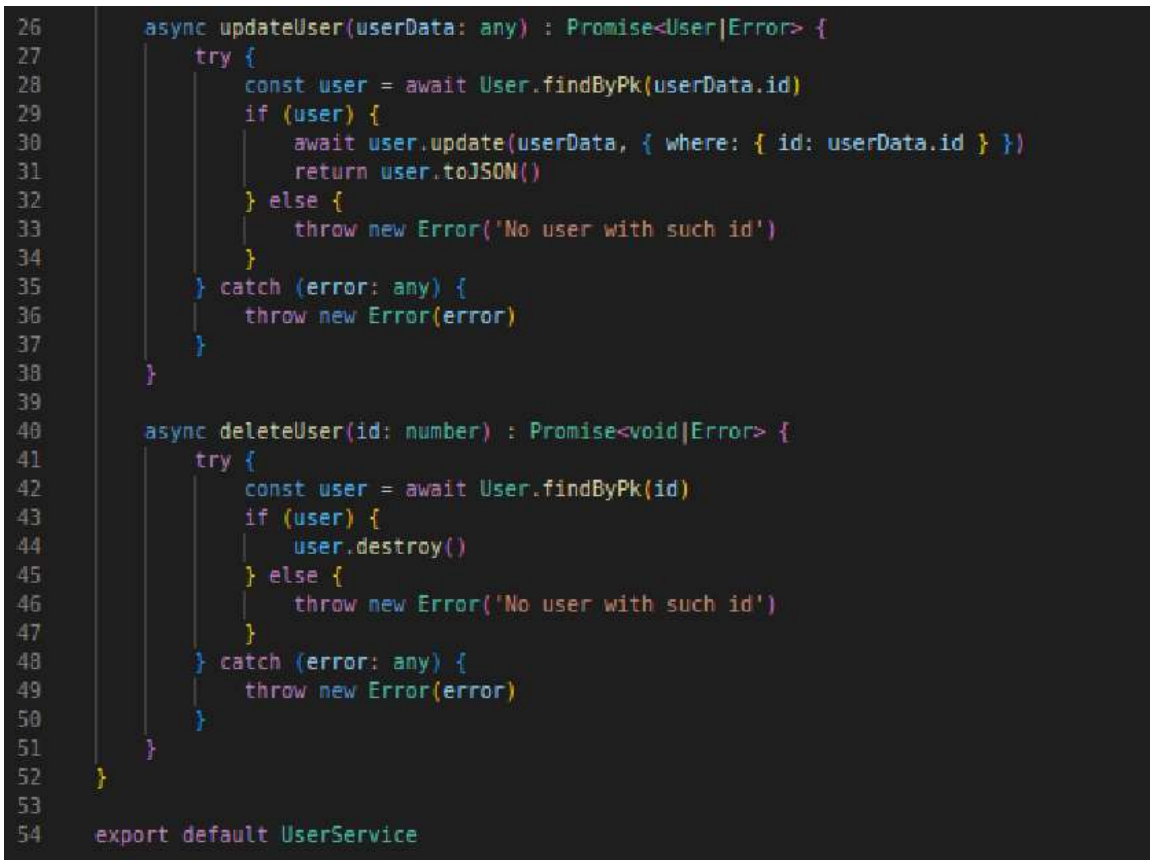
```
src > services > users > TS UserService.ts > ...
import User from "../../models/User"

class UserService {
    async getAllUsers() : Promise<User[]> {
        const users = await User.findAll()
        return users
    }

    async getUserByID(id: number) : Promise<User|Error> {
        const user = await User.findByPk(id);
        if (user) {
            return user.toJSON()
        }
        throw new Error('No User with such id')
    }

    async createUser(userData: any) : Promise<User|Error> {
        try {
            const user = await User.create(userData)
            return user.toJSON()
        } catch (error: any) {
            throw new Error(error)
        }
    }

```



```
    async updateUser(userData: any) : Promise<User|Error> {
        try {
            const user = await User.findByPk(userData.id)
            if (user) {
                await user.update(userData, { where: { id: userData.id } })
                return user.toJSON()
            } else {
                throw new Error('No user with such id')
            }
        } catch (error: any) {
            throw new Error(error)
        }
    }

    async deleteUser(id: number) : Promise<void|Error> {
        try {
            const user = await User.findByPk(id)
            if (user) {
                user.destroy()
            } else {
                throw new Error('No user with such id')
            }
        } catch (error: any) {
            throw new Error(error)
        }
    }
}

export default UserService
```

4) Контроллер для работы с моделью пользователя (файл “./src/controllers/users/UserController.ts”)

src > controllers > users > TS UserController.ts > ...

```ts
import User from '../../models/User'
import UserService from '../../services/users/UserService'

class UserController {
    private userService: UserService

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

    all = async (request: any, response: any) => {
        const users : User[] = await this.userService.getAllUsers()
        response.send(users)
    }

    get = async (request: any, response: any) => {
        try {
            const user : User | Error = await this.userService.getUserByID(request.params.id)
            response.send(user)
        } catch (error: any) {
            response.status(404).send(error.message)
        }
    }

    post = async (request: any, response: any) => {
        const userData = request.body
        try {
            const user : User | Error = await this.userService.createUser(userData)
            response.send(user)
        } catch (error: any) {
            response.status(400).send(error.message)
        }
    }

    update = async (request: any, response: any) => {
        const userData = request.body
        try {
            const user : User | Error = await this.userService.updateUser(userData)
            response.send(user)
        } catch (error: any) {
            response.status(400).send(error.message)
        }
    }

    delete = async (request: any, response: any) => {
        try {
            await this.userService.deleteUser(request.params.id)
            response.send('User deleted successfully')
        } catch (error: any) {
            response.status(404).send(error.message)
        }
    }
}

export default UserController
```

5) Роуты для работы с моделью пользователя (файл “./src/routes/users/UserRoutes.ts”)

src > routes > users > TS UserRoutes.ts > ...

```ts
import express from "express"
import UserController from '../../controllers/users/UserController'

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

const userController : UserController = new UserController()

router.route('/all')
    .get(userController.all)

router.route('/:id')
    .get(userController.get)
    .delete(userController.delete)

router.route('/')
    .post(userController.post)
    .put(userController.update)

export default router
```

6) Общая конфигурация приложения (файл “./src/core/index.ts”)

src > core > TS index.ts > default

```ts
import express from "express"
import { createServer, Server } from "http"
import routes from "../routes/example/index"
import userRoutes from "../routes/users/UserRoutes"
import sequelize from "../providers/db"
import { Sequelize } from 'sequelize-typescript'

class App {
  public port: number
  public host: string

  private app: express.Application
  private server: Server
  private sequelize: Sequelize

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

      this.app = this.createApp()
      this.server = this.createServer()
      this.sequelize = sequelize
  }

```

```
25      private createApp(): express.Application {
26          const app = express()
27          app.use(express.json())
28          app.use('/example', routes)
29          app.use('/users', userRoutes)
30
31          return app
32      }
33
34      private createServer(): Server {
35          const server = createServer(this.app)
36
37          return server
38      }
39
40      public start(): void {
41          this.sequelize
42              .sync()
43              .then(() => {
44                  console.log('Models synchronized successfully')
45              })
46              .catch((error) => console.log(error));
47
48          this.server.listen(this.port, () => {
49              console.log(`Running server on port ${this.port}`)
50          })
51      }
52  }
53
54  export default App
```

7) Проверка работы эндпоинтов

Рисунок 1 - Попытка получения информации о пользователе с id = 3

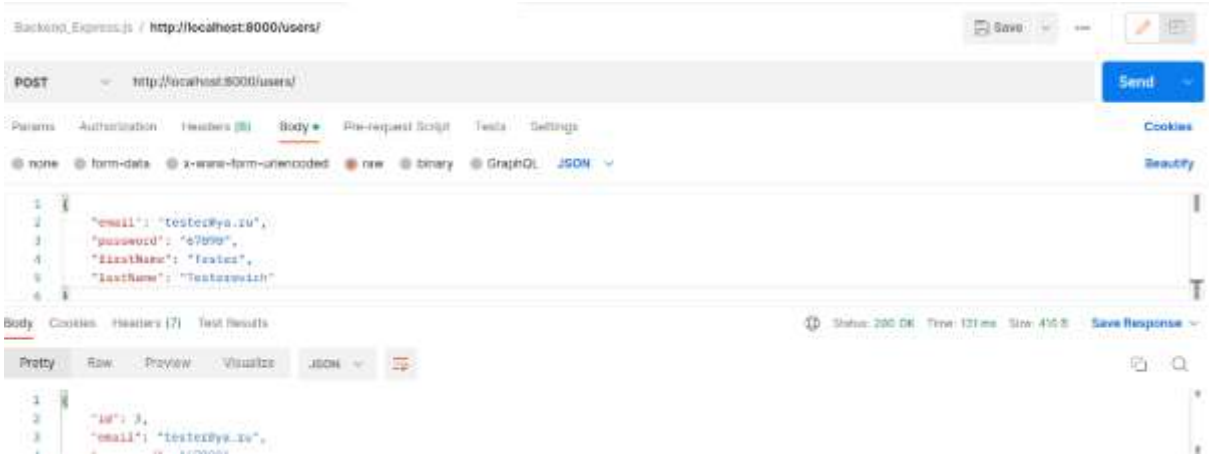


Рисунок 2 - Добавление пользователя

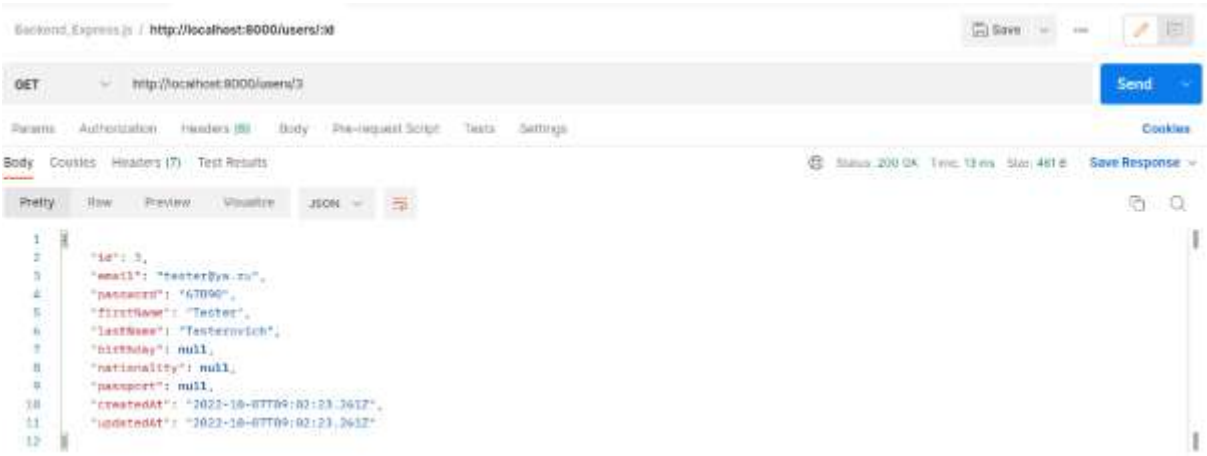


Рисунок 3 - Повторная попытка получения информации о пользователе с id = 3

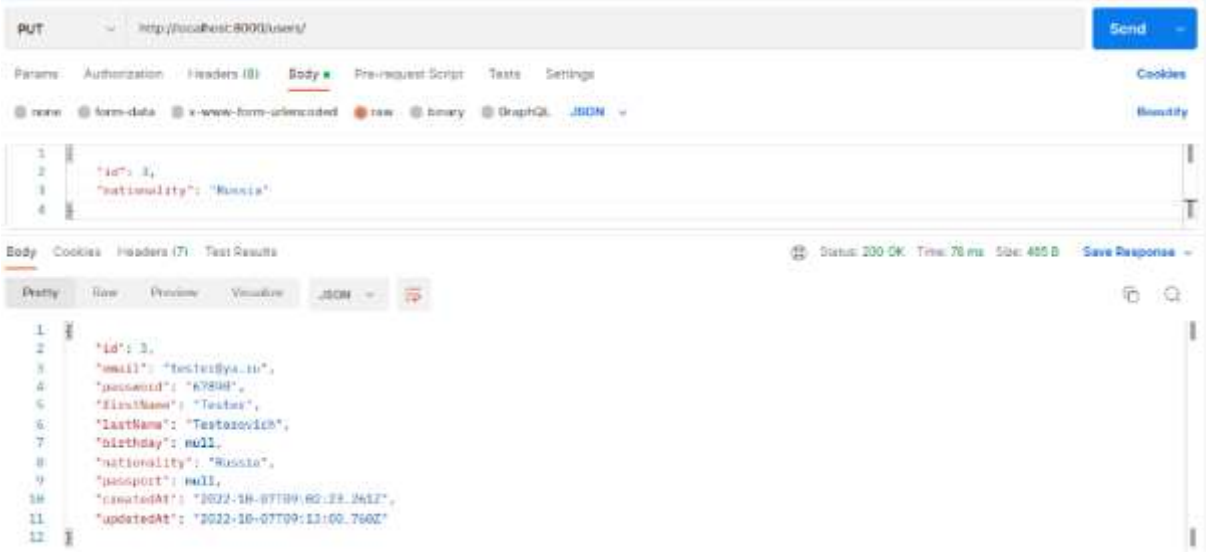


Рисунок 4 - Изменение информации о пользователе с id = 3

Вывод

В процессе выполнения лабораторной работы была изучена структура проекта, создаваемого с помощью микро-фреймворка Express, ORM Sequelize и языка программирования TypeScript, а также были получены практические навыки реализации шаблона (boilerplate), потенциально используемого в качестве стартовой точки при создании серверной части приложения с помощью упомянутых программных средств.