

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

ОТЧЕТ

по Лабораторной работе № 1

по теме: «Boilerplate на Express + TypeORM + TypeScript»

по дисциплине: Бек-энд разработка

Специальность:

09.03.03 Мобильные и сетевые технологии

Проверил:

Добряков Д. И. _____

Дата: «__» _____ 202__ г.

Оценка _____

Выполнил:

студент группы К33401

Чернов Е. К.

Санкт-Петербург

2023

ЦЕЛЬ РАБОТЫ

Получить практические навыки по созданию *backend*-а, используя *Express*, *TypeORM* и *TypeScript*.

ВЫПОЛНЕНИЕ

1 Проектирование модели проекта

Проект реализует платформу для отслеживания криптовалют.

Спроектируем базу данных для проекта (рисунок 1):

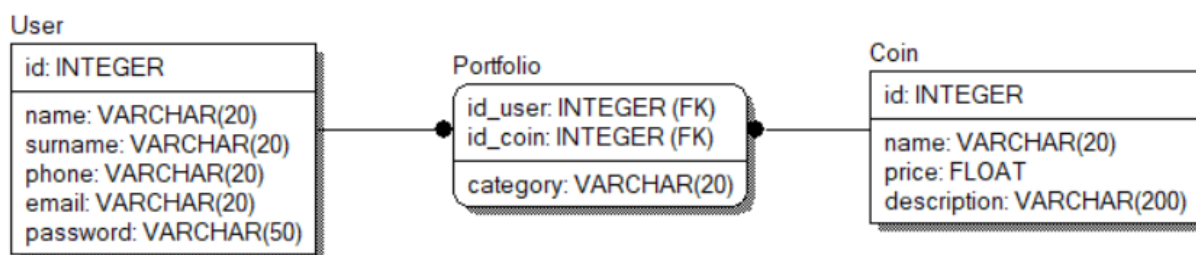


Рисунок 1: База данных проекта

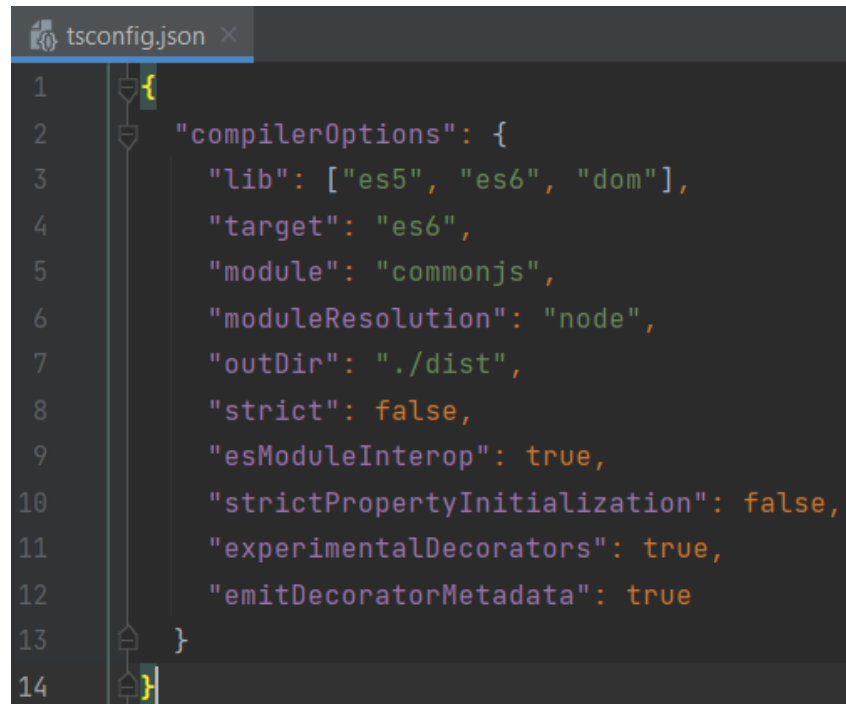
2 Создание архитектуры проекта

Инициализируем наш проект командой *npm init*.

Загружаем необходимые пакеты:

1. командой *npm i <package_name> -S*,
2. командой *npm i <package_name> -D*.

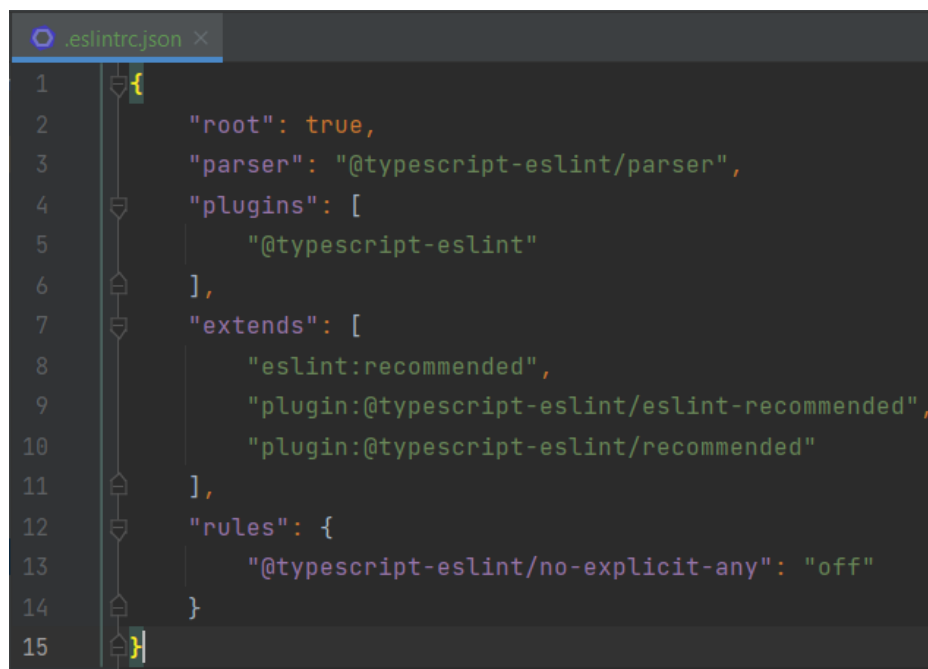
Генерируем «*tsconfig.json*» для *TypeScript*, для компиляции «*.ts*» файлов командой *tsc --init* (рисунок 2).



```
1 {
2   "compilerOptions": {
3     "lib": ["es5", "es6", "dom"],
4     "target": "es6",
5     "module": "commonjs",
6     "moduleResolution": "node",
7     "outDir": "./dist",
8     "strict": false,
9     "esModuleInterop": true,
10    "strictPropertyInitialization": false,
11    "experimentalDecorators": true,
12    "emitDecoratorMetadata": true
13  }
14 }
```

Рисунок 2 - Файл «*tsconfig.json*»

Генерируем «*tslint.json*» для настройки проверки соответствия стандартам кода *TypeScript* командой *npm init @eslint/config*.



```
1 {
2   "root": true,
3   "parser": "@typescript-eslint/parser",
4   "plugins": [
5     "@typescript-eslint"
6   ],
7   "extends": [
8     "eslint:recommended",
9     "plugin:@typescript-eslint/eslint-recommended",
10    "plugin:@typescript-eslint/recommended"
11  ],
12   "rules": {
13     "@typescript-eslint/no-explicit-any": "off"
14   }
15 }
```

Рисунок 3 - Файл «*tslint.json*»

Создаем структуру папок и добавляем необходимые файлы (отображено на рисунке 4):

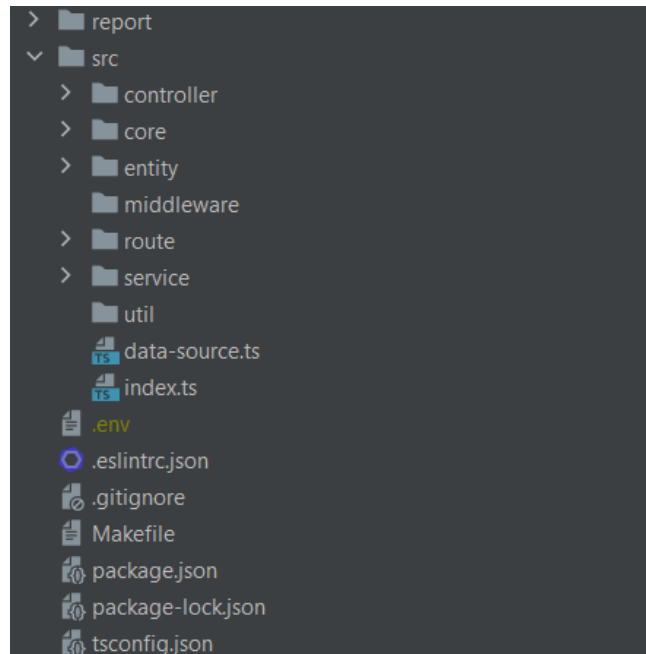


Рисунок 4 - Структура проекта

3 Создание моделей

Модель *User* (рисунок 5):

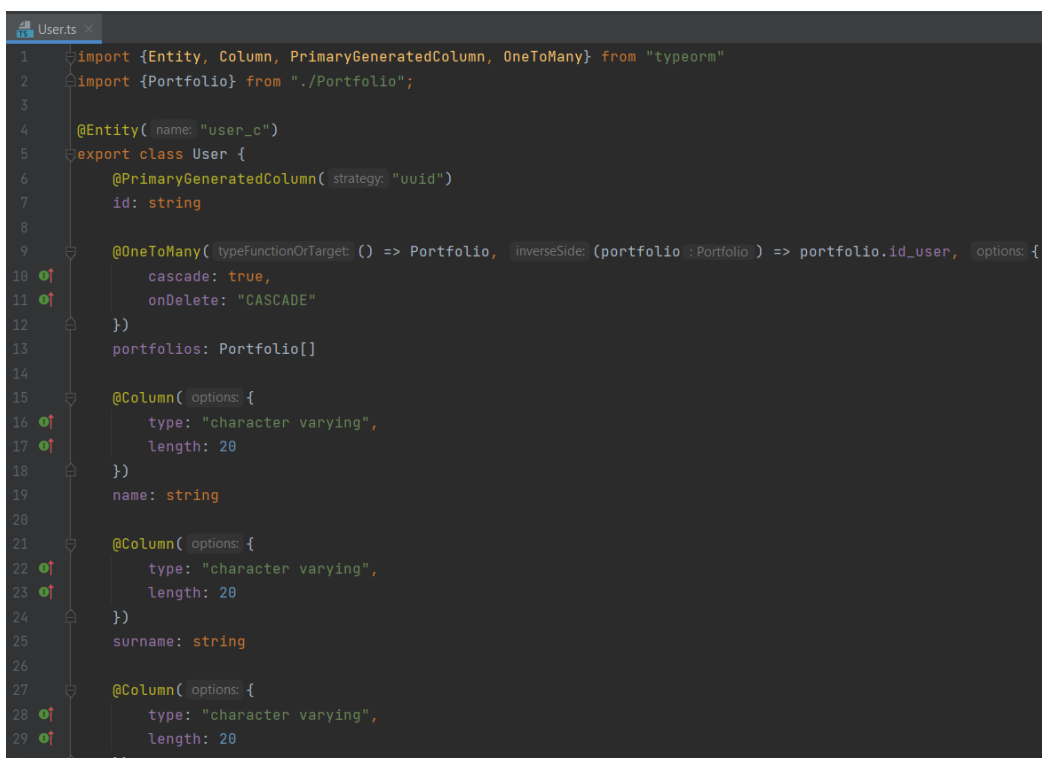


Рисунок 5 - Модель *User*

Модель *Coin* (рисунок 6):

```
Coins.ts
2 import {Portfolio} from "../Portfolio";
3
4 @Entity( name: "coin")
5 export class Coin {
6   @PrimaryGeneratedColumn( strategy: "uuid")
7   id: string
8
9   @OneToMany( typeFunctionOrTarget: () => Portfolio, inverseSide: (portfolio : Portfolio ) => portfolio.id_coin, options: {
10     cascade: true,
11     onDelete: "SET NULL"
12   })
13   portfolios: Portfolio[]
14
15   @Column( options: {
16     type: "character varying",
17     length: 20
18   })
19   name: string
20
21   @Column( options: "money")
22   price: number
23
24   @Column( options: {
25     type: "character varying",
26     length: 200,
27     nullable: true
28   })
29   description: string
30 }
```

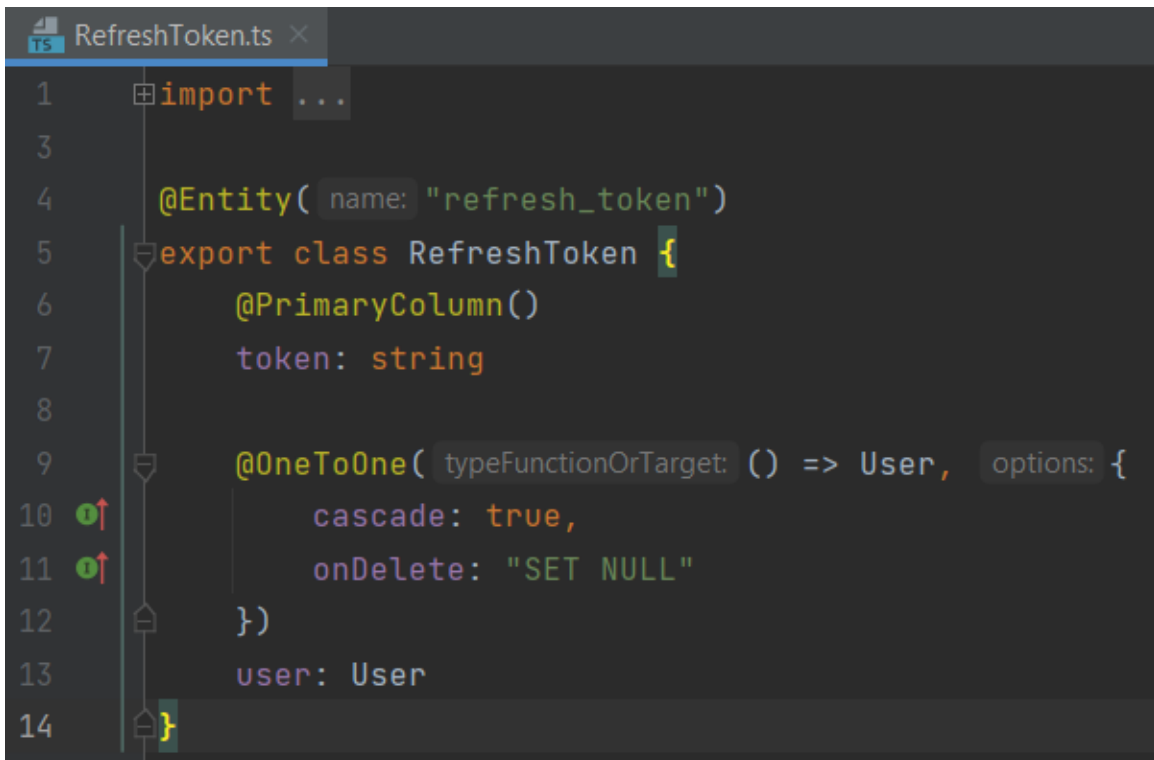
Рисунок 6 - Модель *Coin*

Модель *Portfolio* (рисунок 7):

```
Portfolio.ts
1 import ...
4
5 @Entity( name: "portfolio")
6 export class Portfolio {
7   @PrimaryGeneratedColumn( strategy: 'uuid')
8   id: string
9
10   // @PrimaryColumn()
11   @ManyToOne( typeFunctionOrTarget: () => User, inverseSide: (user : User ) => user.portfolios)
12   id_user: User
13
14   // @PrimaryColumn()
15   @ManyToOne( typeFunctionOrTarget: () => Coin, inverseSide: (coin : Coin ) => coin.portfolios)
16   id_coin: Coin
17
18   @Column( options: {
19     type: "character varying",
20     length: 20
21   })
22   category: string
23 }
```

Рисунок 7 - Модель *Portfolio*

Также добавили модель *RefreshToken* (рисунок 8):

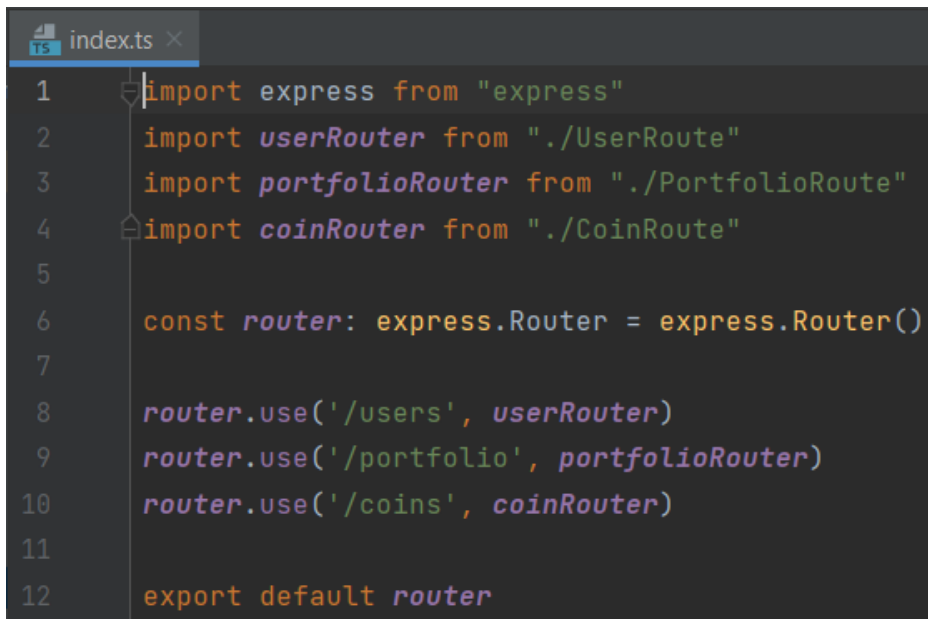


```
1  import ...
3
4  @Entity( name: "refresh_token")
5  export class RefreshToken {
6      @PrimaryColumn()
7      token: string
8
9      @OneToOne( typeFunctionOrTarget: () => User, options: {
10         cascade: true,
11         onDelete: "SET NULL"
12     })
13     user: User
14 }
```

Рисунок 8 - Модель *RefreshToken*

4 Создание маршрутизатора, определение шаблонов

Создаем файл «*index.ts*», который перенаправляет нас на каждый маршрутизатор (рисунок 9):



```
1  import express from "express"
2  import userRouter from "./UserRoute"
3  import portfolioRouter from "./PortfolioRoute"
4  import coinRouter from "./CoinRoute"
5
6  const router: express.Router = express.Router()
7
8  router.use('/users', userRouter)
9  router.use('/portfolio', portfolioRouter)
10 router.use('/coins', coinRouter)
11
12 export default router
```

Рисунок 9 - Файл «*index.ts*»

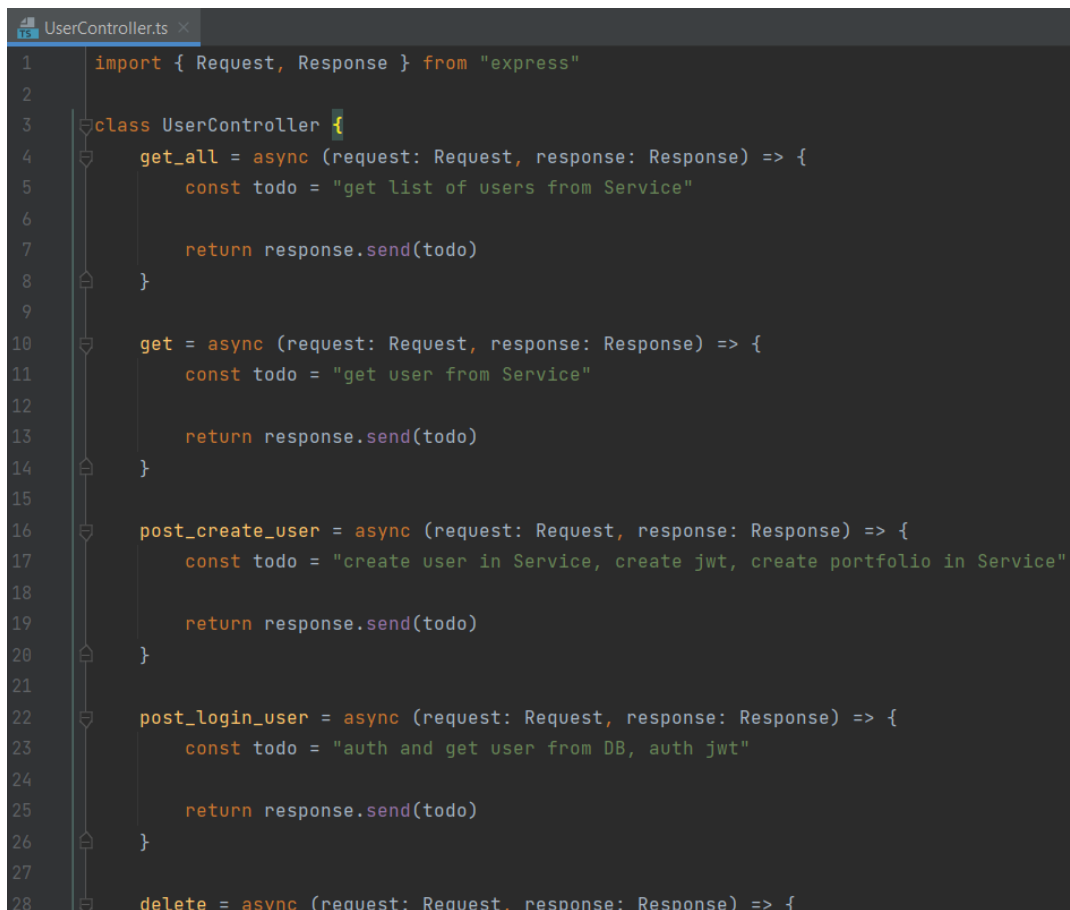
Далее расписываем каждый маршрутизатор по аналогии с приведенным ниже на рисунке 10:



```
1  import express from "express"
2  // import TestController from "../../controller/test/index"
3  import UserController from "../../controller/v1/UserController"
4  import portfolioRouter from "../PortfolioRoute";
5  // import passport from "../../../middleware/passport"
6
7  // Test...
13
14  // Create router and controller
15  const userRouter: express.Router = express.Router()
16  const userController: UserController = new UserController()
17
18  // User routes
19  userRouter.route( prefix: '/list')
20    .get(userController.get_all)
21
22  userRouter.route( prefix: '/specific')
23    .get(userController.get)
24
25  userRouter.route( prefix: '/registration')
26    .get(userController.post_create_user)
27
28  userRouter.route( prefix: '/login')
29    .get(userController.post_login_user)
30
31  portfolioRouter.route( prefix: '/delete')
32    .get(userController.delete)
33
34  export default userRouter
```

Рисунок 10 - Файл «*UserRouter.ts*»

Маршрутизаторы в свою очередь используют контроллеры и вспомогательный функционал (*middleware*). Пример шаблона контроллера, который работает с сервисами (рисунок 11):



```

1  import { Request, Response } from "express"
2
3  class UserController {
4    get_all = async (request: Request, response: Response) => {
5      const todo = "get list of users from Service"
6
7      return response.send(todo)
8    }
9
10   get = async (request: Request, response: Response) => {
11     const todo = "get user from Service"
12
13     return response.send(todo)
14   }
15
16   post_create_user = async (request: Request, response: Response) => {
17     const todo = "create user in Service, create jwt, create portfolio in Service"
18
19     return response.send(todo)
20   }
21
22   post_login_user = async (request: Request, response: Response) => {
23     const todo = "auth and get user from DB, auth jwt"
24
25     return response.send(todo)
26   }
27
28   delete = async (request: Request, response: Response) => {

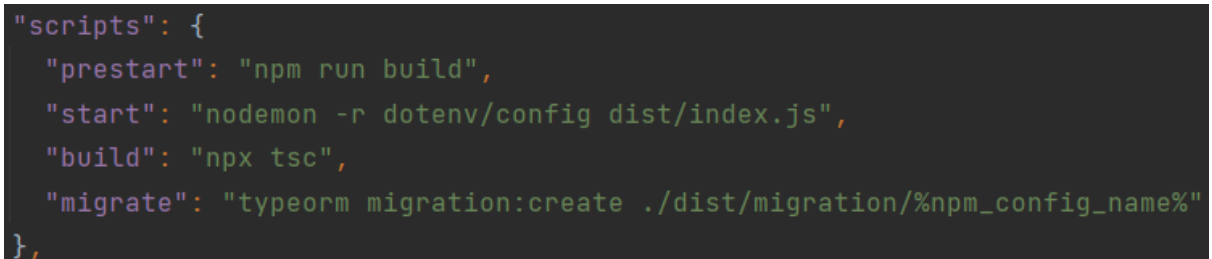
```

Рисунок 11 - Файл «*UserController.ts*»

Контроллеры ссылаются на сервисы, которые работают непосредственно с базой данных, таким образом мы реализуем паттерн «Репозиторий».

5 Создание *Makefile*

Для начала в файл «*package.json*» пропишем следующие команды (рисунок 12):



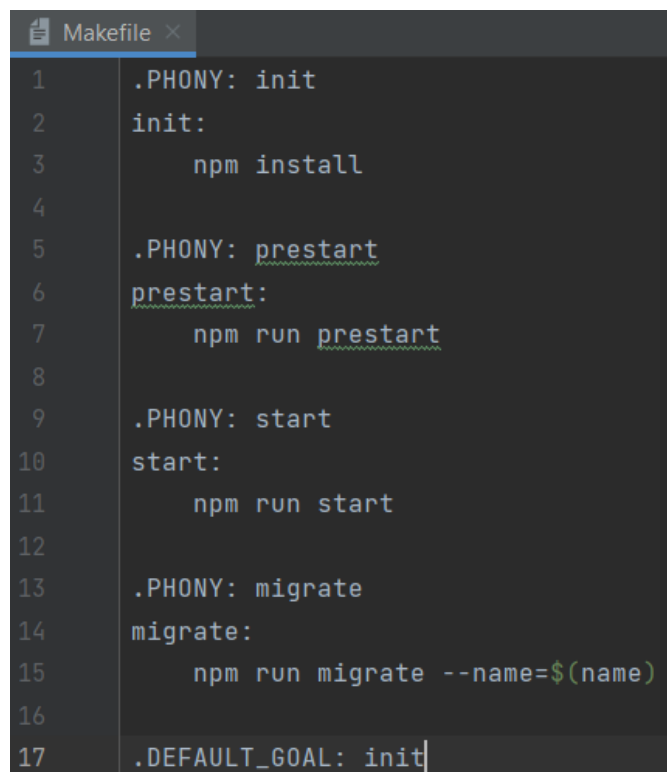
```

"scripts": {
  "prestart": "npm run build",
  "start": "nodemon -r dotenv/config dist/index.js",
  "build": "npx tsc",
  "migrate": "typeorm migration:create ./dist/migration/%npm_config_name%",
},

```

Рисунок 12 - *Scripts* в файле «*package.json*»

После заполним Makefile (рисунок 13):



```
1 .PHONY: init
2 init:
3     npm install
4
5 .PHONY: prestart
6 prestart:
7     npm run prestart
8
9 .PHONY: start
10 start:
11     npm run start
12
13 .PHONY: migrate
14 migrate:
15     npm run migrate --name=$(name)
16
17 .DEFAULT_GOAL: init
```

Рисунок 13 - Файл «*Makefile*»

ВЫВОД

В результате работы получил практические навыки по созданию *backend*-а, используя *Express*, *TypeORM* и *TypeScript*.