

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Буданцев Артём

К3333

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 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”)

## Ход работы

### 1. Инициализация проекта и установка зависимостей

***npm init***

***npm i -S <package>***

***npm i -D <package>***

### 2. Настройка конфиг файлов.

Создание tsconfig.json:

**npx tsc --init**

tsconfig.json имеет следующее содержимое:

```
{
  "compilerOptions": {
    /* Language and Environment */
    "target": "es6",
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    /* Modules */
    "module": "commonjs",
    "rootDir": "./src",
    "outDir": "./dist",
    "esModuleInterop": true,
    // "preserveSymlinks": true,
    "forceConsistentCasingInFileNames": true,
    /* Type Checking */
    "strict": true,
    /* Completeness */
    "skipLibCheck": true
  },
  "include": ["src/**/*.ts"]
}
```

Создаём файл env:

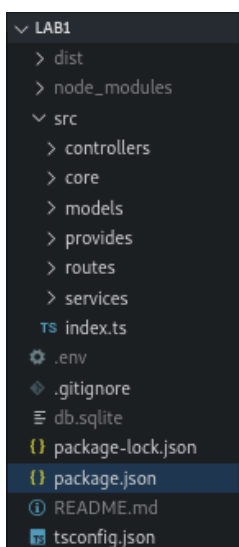
```
PORT = "8000"
HOST = "localhost"
```

Остальные переменные среды окружения задаём по аналогии.

### 3. Содержание package.json, написание кастомных команд

```
{
  "name": "lab1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "watch": "tsc -w",
    "build": "npx tsc",
    "start": "nodemon ./dist/index.js",
    "dev": "npm-run-all --parallel watch start"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@types/dotenv": "^8.2.0",
    "@types/express": "^4.17.17",
    "@types/ini": "^1.3.31",
    "@types/node": "^18.15.11",
    "@types/sequelize": "^4.28.14",
    "@types/validator": "^13.7.15",
    "dotenv": "^16.0.3",
    "nodemon": "^2.0.22",
    "npm-run-all": "^4.1.5",
    "reflect-metadata": "^0.1.13",
    "ts-node": "^10.9.1",
    "typescript": "^5.0.4"
  },
  "dependencies": {
    "@types/http-errors": "^2.0.1",
    "express": "^4.18.2",
    "http-errors": "^2.0.0",
    "sequelize": "^6.31.0",
    "sequelize-cli": "^6.6.0",
    "sequelize-typescript": "^2.1.5",
    "sqlite3": "^5.1.6"
  }
}
```

### 4. Создание структуры проекта



**controllers** - содержит классы контроллеры

**core** - класс с приложением

**models** - содержит описание моделей

**providers** - содержит класс, отвечающий за подключение к БД

**routes** - содержит роутинг

**services** - вспомогательные классы для взаимодействия с моделью

## 5. Создание роутинга

index.ts - общий файл с маршрутами

```
import express from "express"
import userRoutes from "../user/user"

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

router.use('/users', userRoutes)

export default router
```

Роутинг для пользователя:

```
import express from "express"
import UserController from "../../controllers/userController"

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

const controller: UserController = new UserController()

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

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

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

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

router.route("/:id")
  .delete(controller.delete)

export default router
```

## 6. Создание контроллера

```
get = async (req: express.Request, res: express.Response) => {
  res.type("json")
  const users = await this.userService.get();
  res.send(users)
}

getById = async (request: any, response: any) => {
  try {
    const user = await this.userService.getById(
      Number(request.params.id)
    )
    response.send(user)
  } catch (error: any) {
    response.status(404).send({ "error": error.message })
  }
}
```

```

post = async (req: express.Request, response: express.Response) => {
  response.type("json")
  const { body } = req
  try {
    const user = await this.userService.create(body)
    response.send(user)
  } catch (error: any) {
    response.status(404).send({ "error": error.message })
  }
};

```

```

patch = async (req: express.Request, response: express.Response) => {
  response.type("json")
  const { body } = req;
  const { id } = req.params

  try {
    const user = await this.userService.update(Number(id), body)
    response.status(200).send(user)
  } catch (error: any) {
    response.status(404).send({ "error": error.message })
  }
};

```

```

delete = async (req: express.Request, response: express.Response) => {
  response.type("json");
  const { id } = req.params;

  try {
    await this.userService.delete(Number(id))
    response.status(204).send()
  } catch (error: any) {
    response.status(404).send({ "error": error.message })
  }
};

```

**get** - Отправляет данные обо всех пользователях

**getById** - Отправляет данные о конкретном пользователе, используя в качестве параметра запроса id

**post** - Получает тело запроса и создает нового пользователя

**patch** - Получает тело запроса и изменяет пользователя, используя в качестве параметра запроса id

**delete** - Удаляет пользователя, используя в качестве параметра запроса id

## 7. Создание модели

### Модель пользователя

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

@Table
class User extends Model {
  @Unique
  @Column
  login!: string;

  @AllowNull(false)
  @Column
  email!: string;

  @AllowNull(false)
  @Column
  firstName!: string;

  @Column
  lastName!: string;

  @Column
  phone!: string;
}

export default User
```

## 8. Сервисы для работы с моделью

### GET

```
async getById(id: number) {
  const user = await User.findByPk(id)

  if (user) return user.toJSON()
}

async get() {
  const users = await User.findAll();

  if (users) return users
}
```

GET localhost:8000/users/5	Send	Status: 200 OK Size: 181 Bytes Time: 46 ms
Query Headers 2 Auth Body 1 Tests Pre Run	Response Headers 6 Cookies Results Docs	
Query Parameters		
<input type="checkbox"/> parameter	value	
		1 { 2   "id": 5, 3   "login": "bip016", 4   "email": "bip@mail.com", 5   "firstName": "igor", 6   "lastName": "borisov", 7   "phone": "9871", 8   "createdAt": "2023-04-17T08:22:30.230Z", 9   "updatedAt": "2023-04-17T08:22:30.230Z", 10 }

GET localhost:8000/users Send

Status: 200 OK Size: 367 Bytes Time: 9 ms

Query Headers 2 Auth Body 1 Tests Pre Run

Query Parameters

☐ parameter value

Response Headers 6 Cookies Results Docs

```
1 {
2   {
3     "id": 4,
4     "login": "bip014",
5     "email": "bip@mail.com",
6     "firstName": "igor1",
7     "lastName": "borisov1",
8     "phone": "9871",
9     "createdAt": "2023-04-16T22:34:06.004Z",
10    "updatedAt": "2023-04-16T22:34:06.004Z"
11  },
12  {
13    "id": 5,
14    "login": "bip016",
15    "email": "bip@mail.com",
16    "firstName": "igor",
17    "lastName": "borisov",
18    "phone": "9871",
19    "createdAt": "2023-04-17T08:22:30.230Z",
20    "updatedAt": "2023-04-17T08:22:30.230Z"
21  }
22 }
```

## POST

```
async create(userData: any) {
  try {
    const user = await User.create(userData)
    return user.toJSON()
  }
  catch (e: any) {
    const errors = e.errors.map((error: any) => error.message)
    throw new Error(errors)
  }
}
```

POST localhost:8000/users Send

Status: 200 OK Size: 181 Bytes Time: 34 ms

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "login": "iia016",
3   "email": "iia@mail.com",
4   "firstName": "ivan",
5   "lastName": "ivanov",
6   "phone": "12345"
7 }
```

Response Headers 6 Cookies Results Docs

```
1 {
2   "id": 6,
3   "login": "iia016",
4   "email": "iia@mail.com",
5   "firstName": "ivan",
6   "lastName": "ivanov",
7   "phone": "12345",
8   "updatedAt": "2023-04-17T08:48:10.333Z",
9   "createdAt": "2023-04-17T08:48:10.333Z"
10 }
```

## PATCH

```
async update(id: number, userData: any) {
  try {
    const user = await User.findByPk(id)

    if (user) {
      console.log(userData)

      const update_user = await user.update(userData)

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

    throw new Error(errors)
  }
}
```

PATCH localhost:8000/users/5 <span>Send</span>		Status: 200 OK Size: 177 Bytes Time: 51 ms
Query Headers 2 Auth <b>Body 1</b> Tests Pre Run		Response Headers 6 Cookies Results Docs
JSON XML Text Form Form-encode GraphQL Binary		
JSON Content <span>Format</span>		
<pre>1 { 2   "login": "aoi012", 3   "email": "aoi@mail.com", 4   "firstName": "alex", 5   "lastName": "ooo", 6   "phone": "6785" 7 }</pre>		<pre>1 { 2   "id": 5, 3   "login": "aoi012", 4   "email": "aoi@mail.com", 5   "firstName": "alex", 6   "lastName": "ooo", 7   "phone": "6785", 8   "createdAt": "2023-04-17T08:22:30.230Z", 9   "updatedAt": "2023-04-17T08:49:44.246Z" 10 }</pre>



## DELETE

```
async delete(id: number) {  
  try {  
    const user = await User.findByPk(id)  
  
    if (user) {  
      const deleted_user = await user.destroy({where: {id: id}})  
  
      return deleted_user  
    }  
  }  
  catch (e: any) {  
    const errors = e.errors.map((error: any) => error.message)  
  
    throw new Error(errors)  
  }  
}
```

DELETE localhost:8000/users/4 Send

Status: 204 No Content Size: 0 Bytes Time: 32 ms

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {  
2   "login": "aoi012",  
3   "email": "aoi@mail.com",  
4   "firstName": "alex",  
5   "lastName": "ooo",  
6   "phone": "6785"  
7 }
```

Response Headers 3 Cookies Results Docs

1

GET localhost:8000/users Send

Status: 200 OK Size: 361 Bytes Time: 8 ms

Query Headers 2 Auth Body 1 Tests Pre Run

Query Parameters

☐ parameter value

Response Headers 6 Cookies Results Docs

```
1 [  
2   {  
3     "id": 5,  
4     "login": "aoi012",  
5     "email": "aoi@mail.com",  
6     "firstName": "alex",  
7     "lastName": "ooo",  
8     "phone": "6785",  
9     "createdAt": "2023-04-17T08:22:30.230Z",  
10    "updatedAt": "2023-04-17T08:49:44.246Z"  
11  },  
12  {  
13    "id": 6,  
14    "login": "iia016",  
15    "email": "iia@mail.com",  
16    "firstName": "ivan",  
17    "lastName": "ivanov",  
18    "phone": "12345",  
19    "createdAt": "2023-04-17T08:48:10.333Z",  
20    "updatedAt": "2023-04-17T08:48:10.333Z"  
21  }  
22 ]
```

## Вывод

В ходе работы был создан boilerplate с использованием express, typescript и sequelize. Созданы контроллеры, роутинги, модели и сервисы для работы с моделью. Реализованы CRUD операции для работы с моделью. Подобный boilerplate можно использовать в дальнейшем для быстрого старта в других проектах