

**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

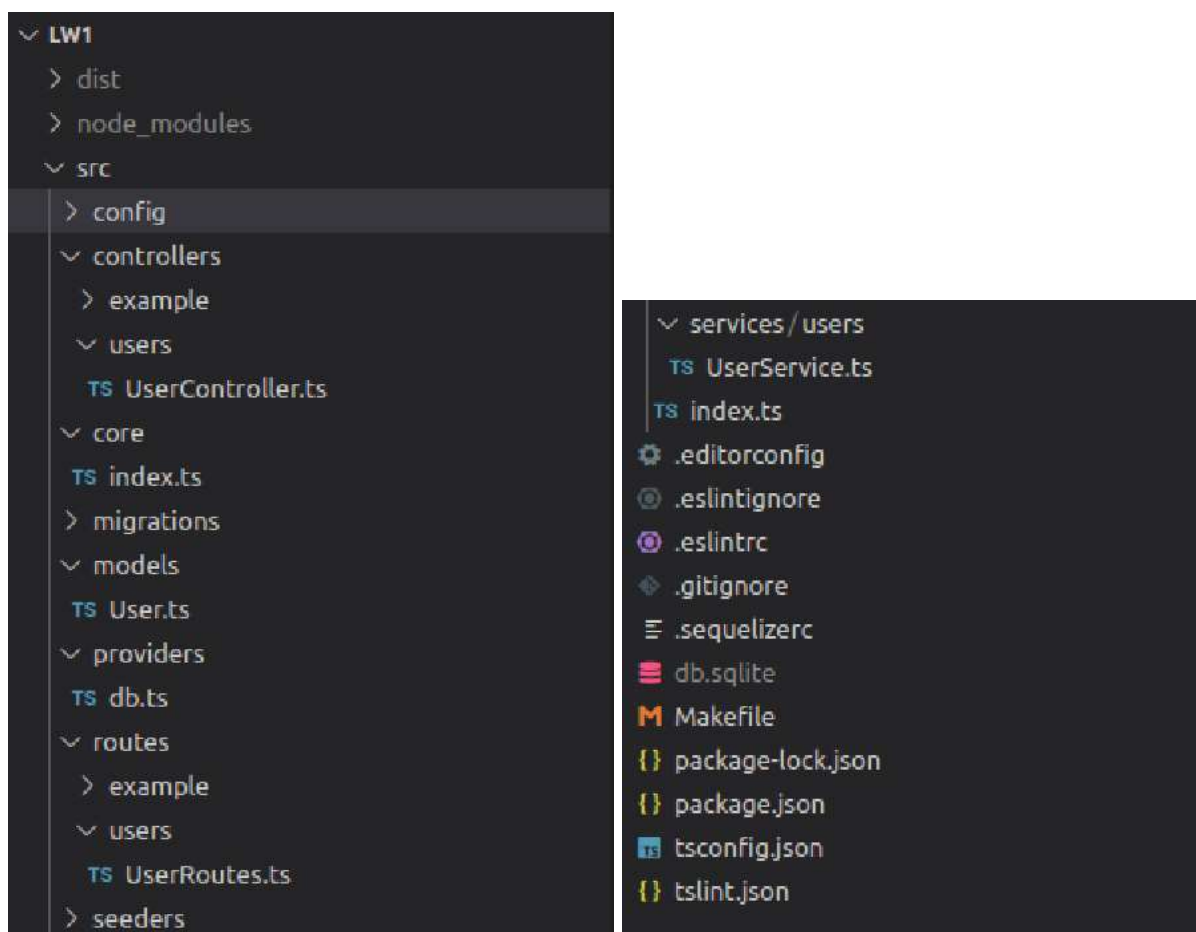
Написать свой boilerplate на Express + Sequelize + TypeScript.

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.

```
src > models > TS Users.ts > ...
1  import { Table, Column, Model, Unique, AllowNull } from 'sequelize-typescript'
2
3  @Table
4  class User extends Model {
5      @Unique
6      @AllowNull(false)
7      @Column
8      email: string
9
10     @AllowNull(false)
11     @Column
12     password: string
13
14     @Column
15     firstName: string
16
17     @Column
18     lastName: string
19
20     @Column
21     birthday: Date
22
23     @Column
24     nationality: string
25
26     @Unique
27     @Column
28     passport: string
29 }
30
31 export default User
```

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

```
src > services > users > TS UserService.ts > ...
1  import User from "../../models/User"
2
3  class UserService {
4    async getAllUsers() : Promise<User[]> {
5      const users = await User.findAll()
6      return users
7    }
8
9    async getUserByID(id: number) : Promise<User|Error> {
10     const user = await User.findByPk(id);
11     if (user) {
12       return user.toJSON()
13     }
14     throw new Error('No User with such id')
15   }
16
17   async createUser(userData: any) : Promise<User|Error> {
18     try {
19       const user = await User.create(userData)
20       return user.toJSON()
21     } catch (error: any) {
22       throw new Error(error)
23     }
24   }
25 }
```

```
26   async updateUser(userData: any) : Promise<User|Error> {
27     try {
28       const user = await User.findByPk(userData.id)
29       if (user) {
30         await user.update(userData, { where: { id: userData.id } })
31         return user.toJSON()
32       } else {
33         throw new Error('No user with such id')
34       }
35     } catch (error: any) {
36       throw new Error(error)
37     }
38   }
39
40   async deleteUser(id: number) : Promise<void|Error> {
41     try {
42       const user = await User.findByPk(id)
43       if (user) {
44         user.destroy()
45       } else {
46         throw new Error('No user with such id')
47       }
48     } catch (error: any) {
49       throw new Error(error)
50     }
51   }
52 }
53
54 export default UserService
```

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

```
src > controllers > users > TS UserController.ts > ...
1  import User from '../../../models/User'
2  import UserService from '../../../services/users/UserService'
3
4  class UserController {
5      private userService: UserService
6
7      constructor() {
8          this.userService = new UserService()
9      }
10
11     all = async (request: any, response: any) => {
12         const users : User[] = await this.userService.getAllUsers()
13         response.send(users)
14     }
15
16     get = async (request: any, response: any) => {
17         try {
18             const user : User | Error = await this.userService.getUserByID(request.params.id)
19             response.send(user)
20         } catch (error: any) {
21             response.status(404).send(error.message)
22         }
23     }
24
25     post = async (request: any, response: any) => {
26         const userData = request.body
27         try {
28             const user : User | Error = await this.userService.createUser(userData)
29             response.send(user)
30         } catch (error: any) {
31             response.status(400).send(error.message)
32         }
33     }
34
35     update = async (request: any, response: any) => {
36         const userData = request.body
37         try {
38             const user : User | Error = await this.userService.updateUser(userData)
39             response.send(user)
40         } catch (error: any) {
41             response.status(400).send(error.message)
42         }
43     }
44
45     delete = async (request: any, response: any) => {
46         try {
47             await this.userService.deleteUser(request.params.id)
48             response.send('User deleted successfully')
49         } catch (error: any) {
50             response.status(404).send(error.message)
51         }
52     }
53 }
54
55 export default UserController
```

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

```
src > routes > users > TS UserRoutes.ts > ...
1  import express from "express"
2  import UserController from '../../controllers/users/UserController'
3
4  const router: express.Router = express.Router()
5
6  const userController : UserController = new UserController()
7
8  router.route('/all')
9    .get(userController.all)
10
11 router.route('/:id')
12   .get(userController.get)
13   .delete(userController.delete)
14
15 router.route('/')
16   .post(userController.post)
17   .put(userController.update)
18
19 export default router
```

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

```
src > core > TS index.ts > [⌕] default
1  import express from "express"
2  import { createServer, Server } from "http"
3  import routes from "../routes/example/index"
4  import userRoutes from "../routes/users/UserRoutes"
5  import sequelize from "../providers/db"
6  import { Sequelize } from 'sequelize-typescript'
7
8  class App {
9    public port: number
10   public host: string
11
12   private app: express.Application
13   private server: Server
14   private sequelize: Sequelize
15
16   constructor(port = 8000, host = "localhost") {
17     this.port = port
18     this.host = host
19
20     this.app = this.createApp()
21     this.server = this.createServer()
22     this.sequelize = sequelize
23   }
24 }
```



```

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

```

## Вывод

В процессе выполнения лабораторной работы была изучена структура проекта, создаваемого с помощью 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.