

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа

Выполнила:

Олейникова Полина

Группа К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Нужно написать свой boilerplate на express + sequelize + typescript.

Должно быть явное разделение на:

- модели
- контроллеры
- роуты
- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

Написали файл для запуска сервера.

```
TS index.ts X
1 import express from "express"
2 import { createServer, Server } from "http"
3 import sequelize from "../providers/db"
4 import { Sequelize } from 'sequelize-typescript'
5 import bodyParser from "body-parser"
6 import cors from "cors";
7 import routes from "../routes/v1/index"
8
9 require('dotenv').config()
10
11 export default class App {
12   public port: number
13   public host: string
14
15   private app: express.Application
16   private server: Server
17   private sequelize: Sequelize
18
19   constructor(port = 8000, host = "localhost") {
20     this.port = Number(process.env.PORT) || port
21     this.host = process.env.HOST || host
22
23     this.app = this.createApp()
24     this.server = this.createServer()
25     this.sequelize = sequelize
26   }
27
28   private createApp(): express.Application {
29     const app = express()
30     app.use(cors())
31     app.use(bodyParser.json())
32     app.use('/v1', routes)
33     return app
34   }
35 }
```

Создали и подключились к БД.

```
JS config.js X TS db.ts
1 require('dotenv').config();
2
3 module.exports = {
4   "development": {
5     "username": process.env.DB_USERNAME,
6     "password": process.env.DB_PASSWORD,
7     "database": process.env.DB_NAME,
8     "host": process.env.HOST,
9     "dialect": "postgres"
10  },
11  "test": {
12    "username": process.env.DB_USERNAME,
13    "password": process.env.DB_PASSWORD,
14    "database": process.env.DB_NAME,
15    "host": process.env.HOST,
16    "dialect": "postgres"
17  },
18  "production": {
19    "username": process.env.DB_USERNAME,
20    "password": process.env.DB_PASSWORD,
21    "database": process.env.DB_NAME,
22    "host": process.env.HOST,
23    "dialect": "postgres"
24  }
25 }
26
TS db.ts X
1 import { Sequelize } from 'sequelize-typescript'
2 import User from '../models/users/User'
3 import { RefreshToken } from '../models/auth/RefreshToken'
4
5 require('dotenv').config();
6
7 const sequelize = new Sequelize({
8   "username": process.env.DB_USERNAME,
9   "password": process.env.DB_PASSWORD,
10  "database": process.env.DB_NAME,
11  "host": process.env.HOST,
12  "dialect": "postgres"
13 })
14 const models = [User, RefreshToken]
15 sequelize.addModels(models)
16
17 // sequelize
18 // .sync()
19 // .then(() => {
20 //   console.log('synced models')
21 // })
22 // .catch((e) => console.log(e));
23
24 async function testConnection() {
25   try {
26     await sequelize.authenticate();
27     console.log('connection has been established successfully.');
```

Обработали пользователя: создали модель, указали маршруты, добавили контроллер и сервис.

```
TS Users.ts X
1 import hashPassword from '../../utils/hashPassword'
2 import { Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpdate, PrimaryKey, DataType } from 'sequelize-typescript'
3 import { Optional } from "sequelize";
4
5 export type UserAttributes = {
6   id: string,
7   name: string,
8   email: string;
9   password: string;
10 };
11
12 export type UserCreationAttributes = Optional<UserAttributes, 'id'>;
13
14
15 @Table
16 export class User extends Model<UserAttributes, UserCreationAttributes> {
17   @PrimaryKey
18   @Column({
19     type: DataType.UUID,
20     defaultValue: DataType.UUIDV4,
21   })
22   id: string;
23
24   @Column
25   name: string;
26
27   @Unique
28   @Column
29   email: string
30
31   @AllowNull(false)
32   @Column
33   password: string
```

```
TS UserRoutes.ts X
1 import express from "express"
2 import UserController from "../../controllers/users/UserController"
3
4
5 const router: express.Router = express.Router()
6
7 const controller: UserController = new UserController()
8
9 router.get('/me', controller.me)
10 router.route('/')
11   .get(controller.get)
12   .patch(controller.update)
13   .delete(controller.delete)
14 router.route('/password') .patch(controller.changePassword)
15
16 export default router
```

```
TS UserControllers.ts X
1 import UserService from '../../services/users/UserService'
2
3 export default class UserController {
4   private userService: UserService
5
6   constructor() {
7     this.userService = new UserService()
8   }
9
10   get = async (request: any, response: any) => {
11     try {
12       const user = await this.userService.getAll()
13       response.status(201).send(user)
14     } catch (error: any) {
15       response.status(404).send({ "error": error.toString() })
16     }
17   }
18
19   update = async (request: any, response: any) => {
20     try {
21       const user = await this.userService.update(request.user, request.body)
22       response.status(201).send(user)
23     } catch (error: any) {
24       response.status(400).send({ "error": error.toString() })
25     }
26   }
27 }
```

```
TS UserService.ts X
1 import User, { UserAttributes } from '../../models/users/User'
2 import checkPassword from '../../utils/checkPassword'
3 import hashPassword from '../../utils/hashPassword'
4
5 class UserService {
6   async getById(id: number): Promise<User> {
7     try {
8       const user = await User.findPk(id)
9       if (!user) {
10         throw new Error("Not Found");
11       }
12       return user
13     } catch (error) {
14       throw error;
15     }
16   }
17
18   async getAll(): Promise<User[]> {
19     try {
20       const user = await User.findAll()
21       return user
22     } catch (error) {
23       throw error;
24     }
25   }
26 }
```

Также написали функции для хеширования пароля:

```
TS checkPassword.ts X
1 import bcrypt from "bcrypt"
2 import { UserAttributes } from "../models/users/User"
3
4 export default (user: UserAttributes, password: string) => {
5   return bcrypt.compareSync(password, user.password)
6 }
```

```
TS hashPassword.ts X
1 import bcrypt from 'bcrypt'
2
3 export default (password: string): string => bcrypt.hashSync(password, bcrypt.genSaltSync(8))
```

Сделали авторизацию и регистрацию, написали middleware

```
TS AuthService.ts X
23
24 async login(email: string, password: string): Promise<{ token: string, refreshToken: string }> {
25   try {
26     const user = await User.findOne({ where: { email } })
27     if (!user || !checkPassword(user, password)) {
28       throw new Error('Email or password is not correct');
29     }
30     const refreshTokenService = new RefreshTokenService(user)
31     const refreshToken = await refreshTokenService.generateRefreshToken()
32     const token = jwt.sign({ id: user.id }, SECRET_KEY, { expiresIn: '2 days' });
33     return { refreshToken: refreshToken, token: token };
34   } catch (error) {
35     throw error;
36   }
37 }
38
39 async refreshToken(refreshToken: string): Promise<{ token: string, refreshToken: string }> {
40   const refreshTokenService = new RefreshTokenService()
41   try {
42     const { userId, isExpired } = await refreshTokenService.isRefreshTokenExpired(refreshToken)
43     if (!isExpired && userId) {
44       const user = await User.findById(userId)
45       const refreshTokenService = new RefreshTokenService(user)
46       const refreshToken = await refreshTokenService.generateRefreshToken()
47       const accessToken = jwt.sign({ id: user.id }, SECRET_KEY, { expiresIn: '2 days' });
48       return { refreshToken: refreshToken, token: accessToken };
49     } else {
50       throw new Error('Invalid credentials')
51     }
52   } catch (error) {
53     throw error;
54   }
55 }
56 }
```

```
TS auth.ts X
1 import jwt, { Secret, JwtPayload } from 'jsonwebtoken';
2 import { Request, Response, NextFunction } from 'express';
3 import UserService from '../services/users/UserService';
4 import { UserAttributes } from '../models/users/User';
5
6 require('dotenv').config();
7
8 if (!process.env.SECRET_KEY) {
9   throw new Error('Missing SECRET_KEY in environment variables');
10 }
11
12 export const SECRET_KEY: Secret = process.env.SECRET_KEY;
13 export interface CustomRequest extends Request {
14   user: UserAttributes;
15 }
16
17 const userService = new UserService();
18
19 export const auth = async (req: Request, res: Response, next: NextFunction) => {
20   try {
21     const token = req.header('Authorization')?.replace('Bearer ', '');
22
23     if (!token) {
24       throw new Error('Missing token');
25     }
26
27     const decoded = jwt.verify(token, SECRET_KEY) as JwtPayload;
28     const userData = await userService.getById(decoded.id);
29
30     if (!userData) {
31       throw new Error('User not found');
32     }
33
34     (req as CustomRequest).user = userData;
35     next();
36   } catch (error) {
37     // Handle authentication error
38   }
39 }
```

В итоге получились работающие запросы:

POST http://localhost:8000/v1/register

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   ... "name": "name1",
3   ... "email": "email1",
4   ... "password": "password1"
5 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "6de77ba3-97e3-467e-9dfd-6b7133be0fc8",
3   "name": "name1",
4   "email": "email1",
5   "password": "$2b$08$Yebj78CK/u.z8Tf4UXU5x.Ee93GoifhEh0qK1RK8sctRVL/cXsW6",
6   "updatedAt": "2024-04-03T17:56:31.528Z",
7   "createdAt": "2024-04-03T17:56:31.528Z"
8 }
```

POST http://localhost:8000/v1/login

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   ... "email": "email1",
3   ... "password": "password1"
4 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "refreshToken": "e72et1c3-ab9f-4868-b14e-322c1ef3ec9d",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ3ZC1lZmUzNGRjOWM2LTMyYmUtdG1kNS1lM2ZjLWJhZjZlLnR5cCI6IkpXVCJ9.eyJ3ZC1lZmUzNGRjOWM2LTMyYmUtdG1kNS1lM2ZjLWJhZjZlLnR5cCI6IkpXVCJ9",
4 }
```

GET http://localhost:8000/v1/users/

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Type Bearer Token Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "e78dc9c6-36be-49d6-b3fc-baea9675b5cf",
3   "name": "name1",
4   "email": "email1",
5   "password": "$2b$08$Yebj78CK/u.z8Tf4UXU5x.Ee93GoifhEh0qK1RK8sctRVL/cXsW6",
6   "updatedAt": "2024-04-03T17:43:24.978Z"
7 }
```

GET http://localhost:8000/v1/users/me

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Type Bearer Token Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "6de77ba3-97e3-467e-9dfd-6b7133be0fc8",
3   "name": "name1",
4   "email": "email1",
5   "password": "$2b$08$Yebj78CK/u.z8Tf4UXU5x.Ee93GoifhEh0qK1RK8sctRVL/cXsW6",
6   "updatedAt": "2024-04-03T17:56:31.528Z",
7   "createdAt": "2024-04-03T17:56:31.528Z"
8 }
```

PATCH http://localhost:8000/v1/users/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   ... "name": "name3"
3 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "6de77ba3-97e3-467e-9dfd-6b7133be0fc8",
3   "name": "name3",
4   "email": "email1",
5   "password": "$2b$08$Yebj78CK/u.z8Tf4UXU5x.Ee93GoifhEh0qK1RK8sctRVL/cXsW6",
6   "updatedAt": "2024-04-03T17:56:31.528Z",
7   "createdAt": "2024-04-03T18:14:37.255Z"
8 }
```

PATCH http://localhost:8000/v1/users/password

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   ... "oldPassword": "password1",
3   ... "newPassword": "password2"
4 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": "Password have successful changed"
3 }
```

Вид в БД.

Tables (3)

- RefreshTokens
- SequelizeMeta
- Users
- Trigger Functions
- Types
- Views

Data Output Messages Notifications

	id	name	email	password	createdAt	updatedAt
	[PK] uuid	character varying (255)	character varying (255)	character varying (255)	timestamp with time zone	timestamp
1	6de77ba3-97e3-467e-9dfd-6b7133be0fc8	name1	email1	\$2b\$08\$DRYU2/g9mlE68Ze5mywceEpiWtQxWpXXM3mojBFwFomV7d3...	2024-04-03 20:56:31.528+03	2024-0

Вывод

В ходе данной работы был написан свой boilerplate на express + sequelize + typescript.