

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа

Выполнил:

Прокопец Семен

Группа К3339

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.






























Задача

Реализация boilerplate

Ход работы

Нужно написать свой boilerplate на nest.js + PrismaORM

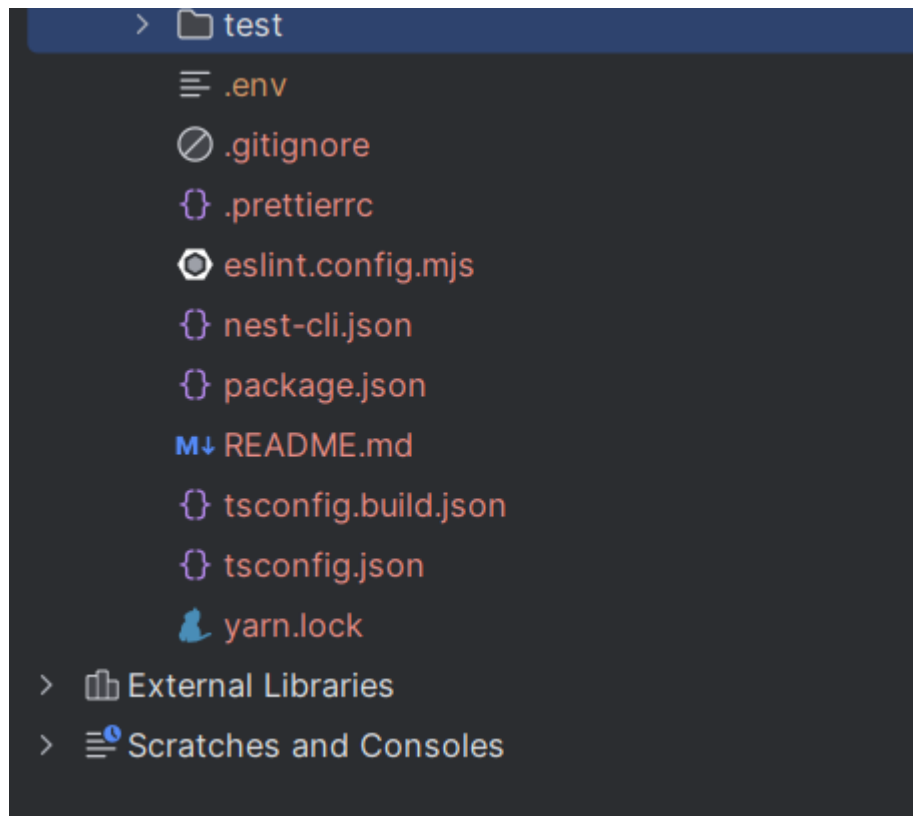
Структура:

- >  dist
- >  node_modules library root
- ✓  prisma
 -  schema.prisma
- ✓  src
 - ✓  application
 -  application.controller.ts
 -  application.dto.ts
 -  application.module.ts
 -  application.service.ts
 - ✓  company
 -  company.controller.ts
 -  company.dto.ts
 -  company.module.ts
 -  company.service.ts
 - ✓  conception
 -  guard.ts
 -  middleware.ts
 -  pipe.ts
 - ✓  education
 -  education.controller.ts
 -  education.dto.ts
 -  education.module.ts
 -  education.service.ts
 - ✓  Industry
 -  industry.controller.ts
 -  industry.dto.ts
 -  industry.module.ts
 -  industry.service.ts

- ▼ resume
 - TS resume.controller.ts
 - TS resume.dto.ts
 - TS resume.module.ts
 - TS resume.service.ts
- ▼ users
 - TS users.controller.ts
 - TS users.dto.ts
 - TS users.module.ts
 - TS users.service.ts
- ▼ vacancy
 - TS vacancy.controller.ts
 - TS vacancy.dto.ts
 - TS vacancy.module.ts
 - TS vacancy.service.ts
- > workExperiences
 - TS app.module.ts
 - TS main.ts
 - TS prisma.service.ts
 - TS types.ts

> test

- ≡ .env
- 🚫 .gitignore
- 📄 .prettierrc
- ⚙️ eslint.config.mjs
- 📄 nest-cli.json
- 📄 package.json
- M↓ README.md
- 📄 tsconfig.build.json



После ревью:

Добавил DTO для авторизации:

```
login.dto.ts x
1 import { IsEmail, IsString } from 'class-validator';
2 import { ApiProperty } from '@nestjs/swagger';
3
4 export class LoginDto { Show usages
5     @ApiProperty({
6     example: 'mem@ka.com',
7     description: 'Email пользователя',
8     })
9     @IsEmail()
10    email: string;
11
12    @ApiProperty({
13    example: '12345678',
14    description: 'Пароль пользователя',
15    })
16    @IsString()
17    password: string;
18 }
```

получение токена:

```
@Post({ path: 'login' }) no usages
@ApiOperation({ summary: 'Вход пользователя и получение JWT-токена' })
@ApiResponse({ status: 200, description: 'JWT-токен успешно получен' })
@ApiResponse({ status: 401, description: 'Неверный email или пароль' })
@ApiBody({ type: LoginDto })
async login(@Body() dto: LoginDto): Promise<{ token: string }> {
    const user :{id: number; email: string; password: st..} = await this.userService.validateUser(dto.email, dto
    const token :string = this.userService.generateToken(user);
    return { token };
}
}
```

Проверка токена в middlewere:

```
import {Injectable, UnauthorizedException} from "@nestjs/common";
import {JwtService} from "@nestjs/jwt";
```

```
@Injectable()

export class JwtMiddleware {

  constructor(private readonly jwtService: JwtService) {}

  use(req: any, res: any, next: () => void) {

    console.log(`Request URL: ${req.url}`);

    const authHeader = req.headers.authorization;

    if (!authHeader || !authHeader.startsWith('Bearer ')) {

      console.log('Token is missing or invalid');

      throw new UnauthorizedException('Отсутствует или неверный токен');

    }

    const token = authHeader.split(' ')[1];

    try {

      const payload = this.jwtService.verify(token, { secret: 'secret' });

      console.log('Token verified:', payload);

      req.user = payload;

      next();

    } catch (error) {

      console.error('Token verification failed:', error.message);

      throw new UnauthorizedException('Неверный или просроченный токен');

    }

  }

}
```

Получение токена по почте и паролю:

POST

/api/users/loginВход пользователя и получение JWT-токена

Parameters

No parameters

Request body^{required}

application/json

{
 "email": "user@example.com",
 "password": "password123"
}

Server response

CodeDetails

201
Undocumented

Response body

{
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIIOjIsImVtYmIsIm9kaWkiOiJjbGkiOjE1bnVybSImlhdCI6MTc0MTEwNDA3NTUyZWYyZXhwIjoxNzQ1MDc4NmAzfQ.hf1S2bHacK0wk5e1XhQFmw1-6pTZ19PV603XQ21vA2Y"
}

Response headers

access-control-allow-origin: *
connection: keep-alive
content-length: 187
content-type: application/json; charset=utf-8
date: Sat, 19 Apr 2025 15:05:18 GMT
etag: W/"bb-kgt9ceSpfNaMidtVFhlrIC6MIkQ"
keep-alive: timeout=5
x-powered-by: Express

Responses

CodeDescription

Хеширование пароля:

The screenshot shows the NestDB web interface. At the top, there's a toolbar with various icons for navigation and actions. Below the toolbar, the SQL query is displayed: `SELECT id, email, password, role, created_at`. The query is executed, and the results are shown in a table with 5 columns: `id`, `email`, `password`, `role`, and `created_at`. The first row of data is highlighted in blue.

	id	email	password	role	created_at
1	2	user@example.com	\$2b\$10\$tNC4xi07QbXGV1PvuNAPNehid.LyxPeGaTW8HFrEM/sA1NisTbDr0J4-19		14:33:17

On the right side, there's a sidebar showing the database structure. It includes a tree view with folders like `public` and `tables`, and a list of tables including `users`.

Вывод

Спроектировали и реализовали классное ПО