

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа

Выполнил:

Шайтор Илья

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

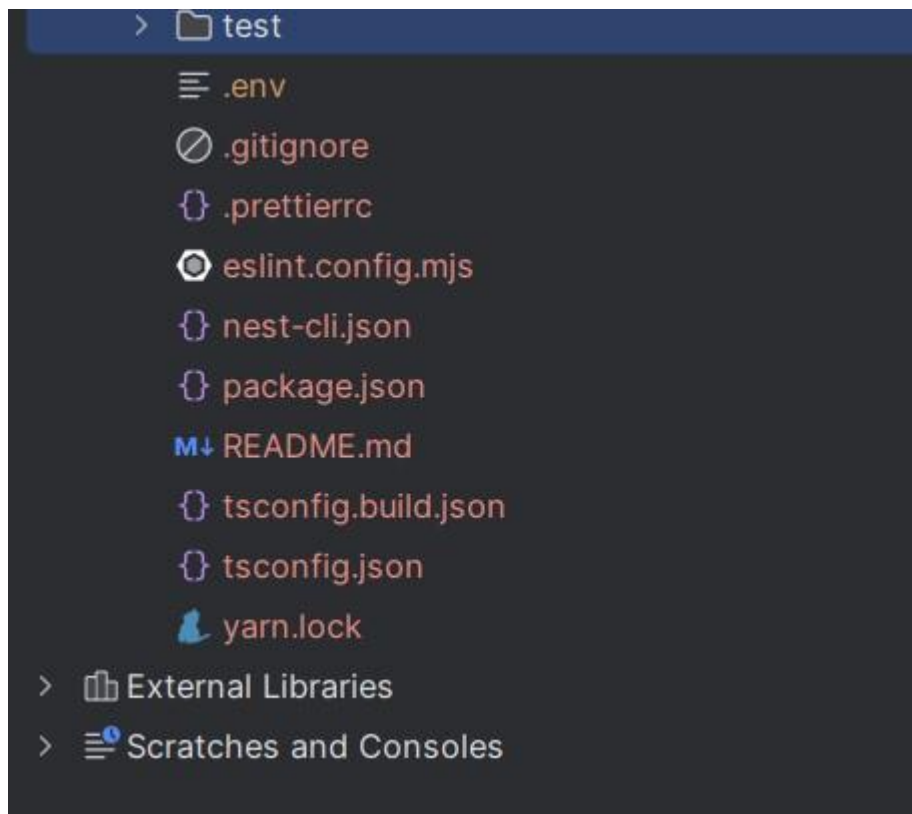
- ⚙️ eslint.config.mjs

- 📄 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/login Вход 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.eyJzdWIIOjI1bmVkbSIsImVtYWkiOiJodsdXNlckBleGFtcCw1LnVkbSIsIm1hdCI6MTY3MTEwMDAwMCwibmFA3NTExZHVudWZxLjoxNzQ1ODc4NDZAsFQ.HfAS2bHacK0uKSe1KHqPmw1-6pTZ19PY6O3XQ21vA2Y"  
}
```

Download

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-k6t9ZeSpfnAkidcvFh.fLCGM1KQ"  
keep-alive: timeout=5  
x-powered-by: Express
```

Responses

CodeDescriptionLinks

Хеширование пароля:

The screenshot shows the DBeaver SQL editor interface. The top toolbar contains icons for undo, redo, and other database operations. The SQL editor area displays the query: `SELECT * FROM users`. Below the query, the results are shown in a table with the following columns: `id`, `email`, `password`, `role`, and `created_at`. The first row of data is highlighted, showing the values: `2`, `user@example.com`, `$2b$10$tNC4x107Qbx6V1PvuNApNehid.LyxPeGaTW8HFrEM/sA1NisTbDr0`, `J4-19`, and `14:33:17`. The right sidebar shows the database structure, including the 'NestDB' database and its tables.

Вывод

Спроектировали и реализовали классное ПО