САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 5

Выполнила:

Казарян Тигран

Группа К3441

Проверил:

Добряков Д. И.

Санкт-Петербург

2026 г.

СОДЕРЖАНИЕ

Задачи

- Выделить самостоятельные модули в вашем приложении;
- Разделить своё API на микросервисы (минимум, их должно быть 3);
- Настроить сетевое взаимодействие между микросервисами.

Ход работы

В данной работе было выделено 3 основных микросервиса:

1. company-vacancy-service
2. industry-service
3. user-resume-service

Также было реализовано связующее звено - api-gateway.

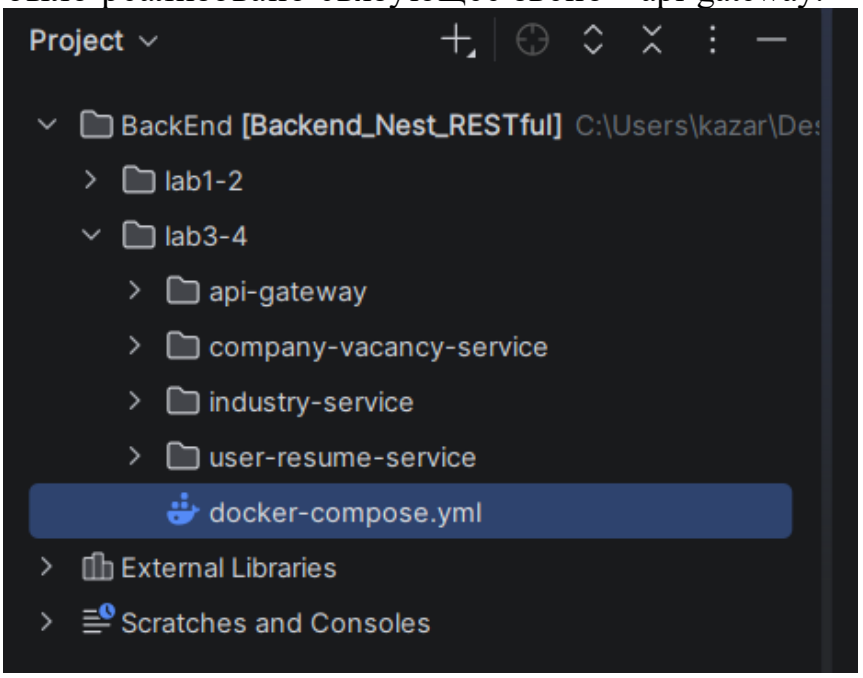

Рисунок 1 - Структура проекта

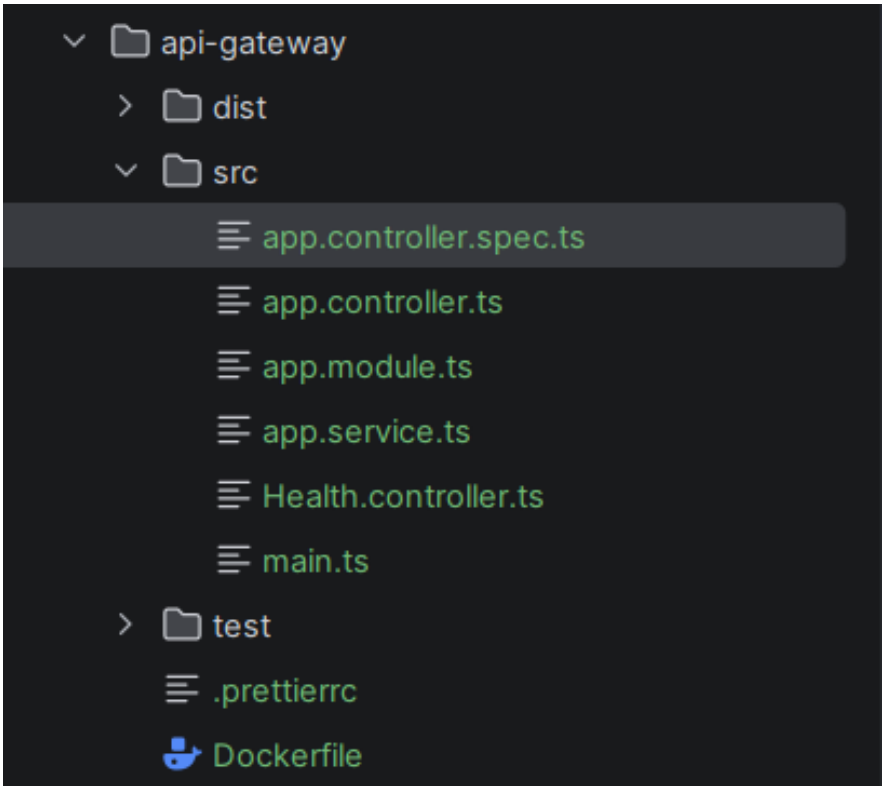

Рисунок 2 - Api-gateway

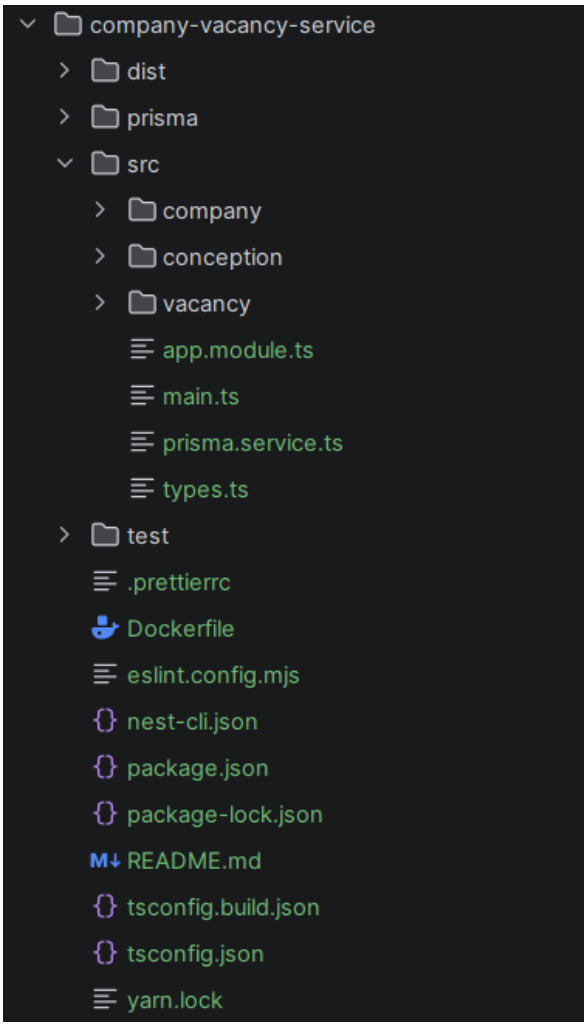

Рисунок 3 - company-vacancy-service

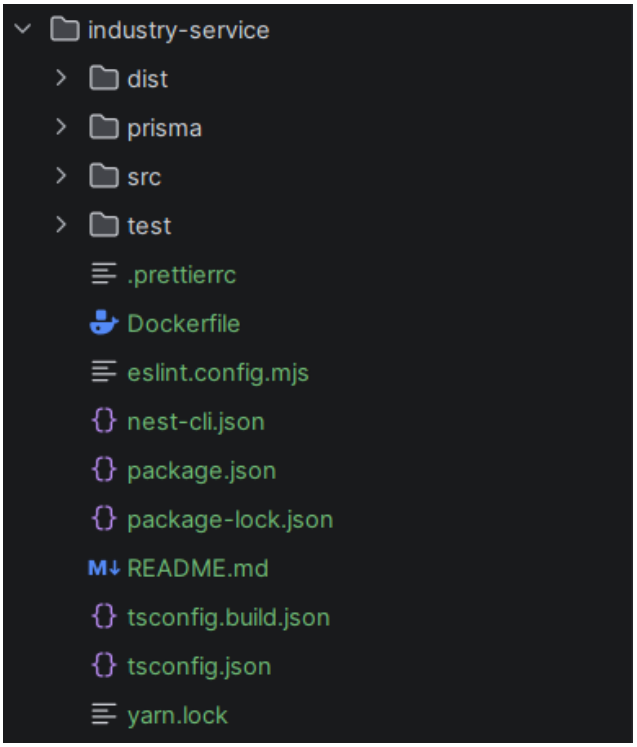

Рисунок 4 – industry-service

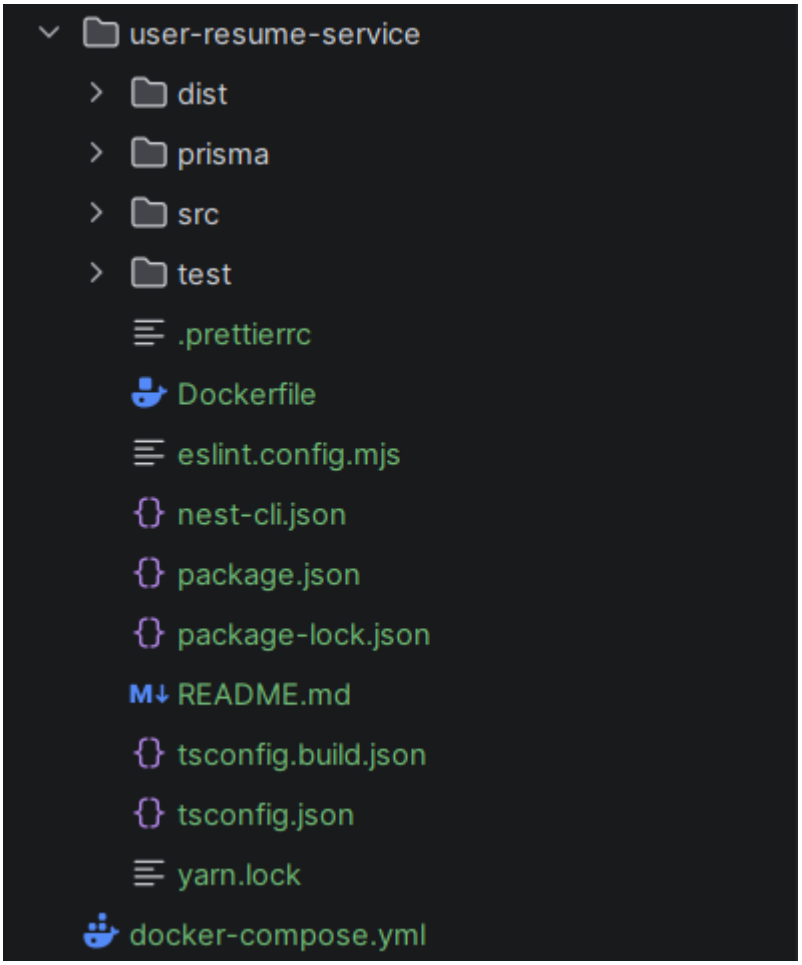

Рисунок 3 – user-resume-service

Листинг 1 – Реализация API Gateway Controller

```
import {
  Controller,
  Req,
  Res,
  Get,
  Post,
  Body,
  HttpStatus,
  ValidationPipe,
} from '@nestjs/common';
import { HttpService } from '@nestjs/axios';
import { Request, Response } from 'express';
import { ApiTags, ApiOperation, ApiBody, ApiResponse } from '@nestjs/swagger';
```

```
@Controller()
export class AppController {
  private readonly services = {
    user: 'http://localhost:3001',
    company: 'http://localhost:3002',
    industry: 'http://localhost:3003',
  };

  constructor(private readonly httpService: HttpService) {}

  @Get(':service/*')
  @ApiTags('gateway')
  @ApiOperation({ summary: 'Proxy GET-запросы к микросервисам' })
  @ApiResponse({ status: 200, description: 'Успешный ответ от микросервиса' })
  @ApiResponse({ status: 404, description: 'Сервис не найден' })
  async get(@Req() req: Request, @Res() res: Response) {
    const { service } = req.params;
    this.logRequest(req, service);

    if (!this.services[service]) {
      return res
          .status(HttpStatus.NOT_FOUND)
          .json({ error: `Service "${service}" not found` });
    }

    const url = this.buildUrl(req, service);
    try {
      const response = await this.httpService.axiosRef.get(url, {
        headers: req.headers,
      });
      this.proxyResponse(res, response);
    } catch (error) {
      this.handleError(res, error);
    }
  }
```

```
@Post(':service/*')
@ApiTags('gateway')
@ApiOperation({ summary: 'Proxy POST-запросы к микросервисам' })
@ApiBody({ description: 'Данные для отправки в микросервис' })
@ApiResponse({ status: 201, description: 'Ресурс успешно создан' })
@ApiResponse({ status: 400, description: 'Ошибка валидации или запроса' })
async post(
    @Req() req: Request,
    @Res() res: Response,
    @Body(new ValidationPipe({ transform: true })) body: any,
) {
  const { service } = req.params;
  this.logRequest(req, service, body);

  if (!this.services[service]) {
    return res
        .status(HttpStatus.NOT_FOUND)
        .json({ error: `Service "${service}" not found` });
  }

  const url = this.buildUrl(req, service);
  try {
    const response = await this.httpService.axiosRef.post(url, body, {
      headers: req.headers,
    });
    this.proxyResponse(res, response);
  } catch (error) {
    this.handleError(res, error);
  }
}
```

```
  private buildUrl(req: Request, service: string): string {
    const baseUrl = this.services[service];
    let path = req.url.replace(`/api/${service}`, '');

    if (!path || path === '/') path = '';

    const finalUrl = `${baseUrl}${path}`;
    console.log(`Proxying to: ${finalUrl}`);
    return finalUrl;
  }

  private proxyResponse(res: Response, response: any) {
    const setCookies = response.headers['set-cookie'];
    if (setCookies) {
      res.header('Set-Cookie', setCookies);
    }
    res.status(response.status).json(response.data);
  }

  private handleError(res: Response, error: any) {
    const status = error.response?.status || 500;
    const data = error.response?.data || { error: 'Internal Server Error' };
    console.error(` Error in gateway:`, error.message);
    res.status(status).json(data);
  }

  private logRequest(req: Request, service: string, body?: any) {
    console.log(` ${req.method} request to service: ${service}`);
    console.log(`   URL: ${req.originalUrl}`);
    if (body) {
      console.log(`   Body:`, JSON.stringify(body, null, 2));
    }
  }
}
```

```
119  @Controller('health')
120  export class AppHealthController {
121    @Get()
122    check() {
123      return {
124        status: 'Gateway is running',
125        timestamp: new Date(),
126      };
127    }
128  }
```

Вывод

Сначала проект был написан как один большой монолит - все функции находились в одном приложении, что делало его сложным для масштабирования и развертывания. Я переделал проект на микросервисную архитектуру, разделив его на четыре отдельных сервиса: API Gateway (раздает запросы), Company Vacancy Service (работает с вакансиями), Industry Service (хранит данные об индустриях) и User Resume Service (управляет резюме). Такой подход, построенный на NestJS, REST API и Docker Compose, позволяет легко обновлять отдельные сервисы, быстрее масштабировать систему, и если один сервис сломается, остальные продолжат работать.