**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Лабораторная работа 3
# Миграция написанного API на микросервисную архитектуру

Выполнила:

Едигарева Дарья

Группа
К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

**Задача**

Выделить самостоятельные модули в приложении.

Разделить API на микросервисы (≥3).

Настроить сетевое взаимодействие (gateway, внутренняя сеть).

Внедрить IoC/DI по контрактам.

**Архитектура**

**Сервисы**:

Auth+Users: регистрация, логин, хранение пользователей.

Vacancies: компании и вакансии, поиск.

Profiles: профили соискателей, резюме, опыт, образование.

**Внешняя маршрутизация через Nginx gateway:**

/api/auth/* → Auth

/api/vacancies/* → Vacancies

/api/profiles/* → Profiles

gateway/nginx.conf

```
events {}

http {

 server {

   listen 80;

   client_max_body_size 10m;


   rewrite ^/api/(auth|vacancies|profiles)/docs$ /api/$1/docs/ permanent;


   location = /register { return 307 /api/auth/register; }

   location = /login { return 307 /api/auth/login; }
```

```
    location = /docs { return 307 /api/auth/docs/; }
    location = /docs/ { return 307 /api/auth/docs/; }

    location /api/auth/ {
      proxy_pass http://auth_service:8001/;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /api/vacancies/ {
      proxy_pass http://vacancies_service:8002/;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /api/profiles/ {
      proxy_pass http://profiles_service:8003/;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
  }
}
```

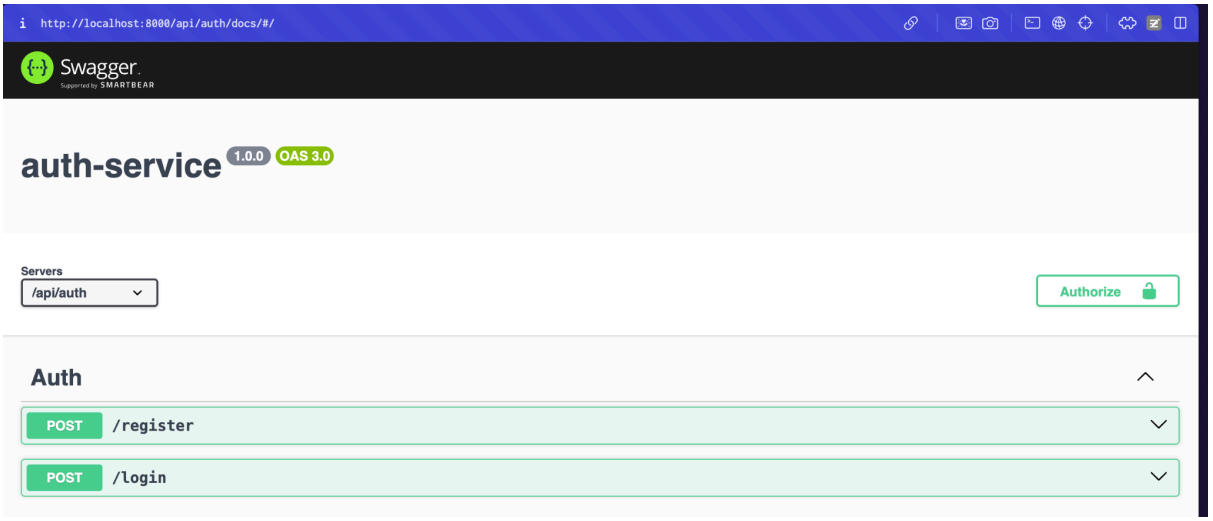
Настроено:

Проксирование /api/<svc>/ на соответствующий контейнер.

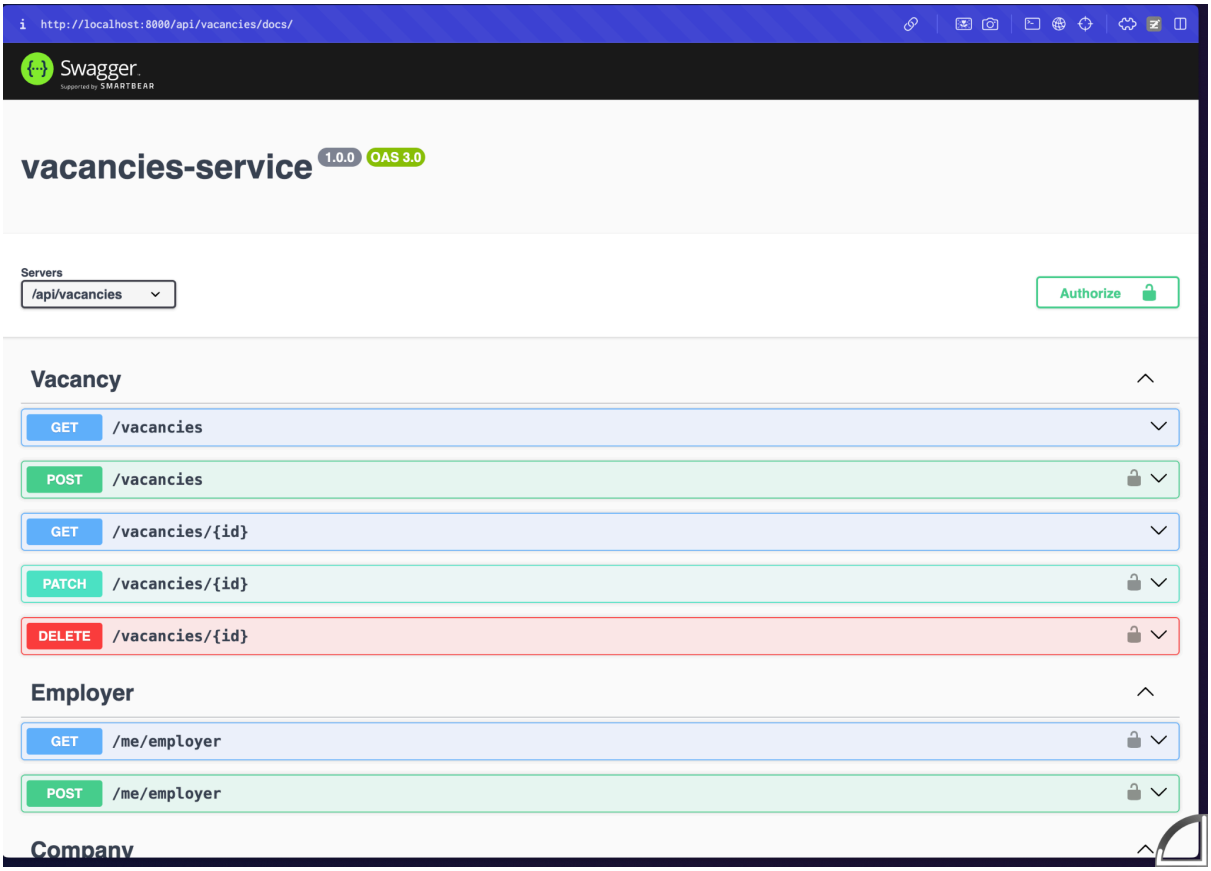
Приведение /api/<svc>/docs к /api/<svc>/docs/ (чтобы корректно работал Swagger UI).

Swagger через gateway доступен по:

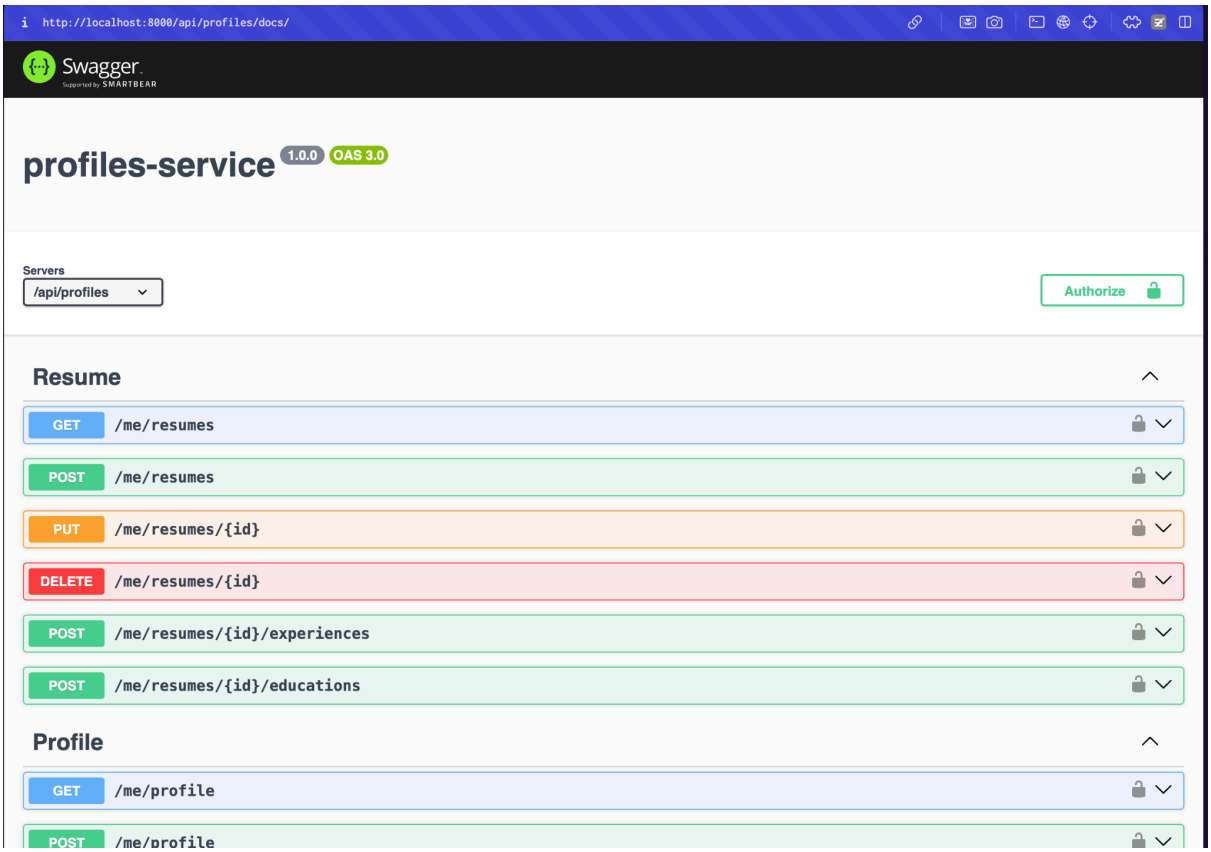
http://localhost:8000/api/auth/docs/




http://localhost:8000/api/vacancies/docs/




http://localhost:8000/api/profiles/docs/



Контейнеризация

Compose-файл: docker-compose.micro.yml:

```
services:

db:
  image: postgres:17
  container_name: backend_hw2_db
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
    POSTGRES_DB: postgres
  ports:
    - '5432:5432'
  volumes:
    - data:/var/lib/postgresql/data


auth-service:
  build:
    context: ./services/auth-service
    dockerfile: Dockerfile
  container_name: auth_service
  environment:
    NODE_ENV: production
    APP_HOST: 0.0.0.0
    APP_PORT: 8001
    APP_PROTOCOL: http
    DB_HOST: db
    DB_PORT: 5432
    DB_USER: postgres
```

```
    DB_PASSWORD: postgres
    DB_NAME: postgres
    JWT_SECRET_KEY: secret
  depends_on:
    - db
  expose:
    - '8001'


vacancies-service:
  build:
    context: ./services/vacancies-service
    dockerfile: Dockerfile
  container_name: vacancies_service
  environment:
    NODE_ENV: production
    APP_HOST: 0.0.0.0
    APP_PORT: 8002
    APP_PROTOCOL: http
    DB_HOST: db
    DB_PORT: 5432
    DB_USER: postgres
    DB_PASSWORD: postgres
    DB_NAME: postgres
    JWT_SECRET_KEY: secret
  depends_on:
    - db
  expose:
    - '8002'
```

```
profiles-service:
  build:
    context: ./services/profiles-service
    dockerfile: Dockerfile
  container_name: profiles_service
  environment:
    NODE_ENV: production
    APP_HOST: 0.0.0.0
    APP_PORT: 8003
    APP_PROTOCOL: http
    DB_HOST: db
    DB_PORT: 5432
    DB_USER: postgres
    DB_PASSWORD: postgres
    DB_NAME: postgres
    JWT_SECRET_KEY: secret
  depends_on:
    - db
  expose:
    - '8003'


gateway:
  image: nginx:alpine
  container_name: api_gateway
  volumes:
    - ./gateway/nginx.conf:/etc/nginx/nginx.conf:ro
  depends_on:
```

```
    - auth-service
    - vacancies-service
    - profiles-service
  ports:
    - '8000:80'


volumes:
 data:
```

Внутренние сервисы не публикуют порты наружу — только expose, доступ извне только через gateway.

Auth: expose: 8001

Vacancies: expose: 8002

Profiles: expose: 8003

Gateway: ports: 8000:80

**Данные и миграции**

Одна БД PostgreSQL, три схемы:

Auth — schema: "auth": services/auth-service/src/config/settings.ts

Vacancies — schema: "vacancies": services/vacancies-service/src/config/settings.ts

Profiles — schema: "profiles": services/profiles-service/src/config/settings.ts

Источники данных указывают schema и пути к миграциям:

(services/*/src/config/data-source.ts)

Миграции:

Есть во всех сервисах: services/*/src/migrations/*init*.ts

В миграциях используется uuid_generate_v4(). Требуется расширение:

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

**IoC/DI**

Контейнер: tsyringe; паттерн “интерфейс + Symbol-токен + реализация + регистрация + инъекция”.

Auth:

Контракт+токен: services/auth-service/src/services/auth.types.ts:

```
import { User } from '../models/User';

export interface IAuthService {

 register(email: string, password: string): Promise<User>;

 login(email: string, password: string): Promise<{ accessToken: string }>;

}

// Injection token for DI container

export const AUTH_SERVICE = Symbol('IAuthService');
```

Реализация: services/auth-service/src/services/auth.ts:

```
@singleton()

export class AuthService implements IAuthService {

 private readonly userRepo: Repository<User>;

 constructor() {

   this.userRepo = dataSource.getRepository(User);

 }

 public async register(email: string, password: string): Promise<User> {

   const existing = await this.userRepo.findOne({ where: { email } });
```

```
   if (existing) throw new Error('Email already in use');

   const hash = await bcrypt.hash(password, 10);

   const user = this.userRepo.create({ email, passwordHash: hash });

   return this.userRepo.save(user);

 }

 public async login(email: string, password: string): Promise<{ accessToken: string
}> {

   const user = await this.userRepo.findOne({ where: { email } });

   if (!user) throw new Error('Invalid credentials');

   const ok = await bcrypt.compare(password, user.passwordHash);

   if (!ok) throw new Error('Invalid credentials');

   const token = jwt.sign({ id: user.id, email: user.email },
SETTINGS.JWT_SECRET_KEY, { expiresIn: 60 * 5 });

   return { accessToken: token };

 }

}
```

Регистрация: services/auth-service/src/ioc.ts

```
export const iocContainer: IocContainer = {

 get<T>(controller: ServiceIdentifier<T>): T | Promise<T> {

   return container.resolve(controller as any);

 },

};



// Register interface-based bindings

container.register(AUTH_SERVICE, { useClass: AuthService });
```

Инъекция: services/auth-service/src/controllers/auth-controller.ts:

```
class AuthController extends Controller {
```

```
constructor(@inject(AUTH_SERVICE) private service: IAuthService) { super(); }
```

TSOA генерирует swagger.json и Express-роуты.

**Запуск**

Поднять контейнеры:

docker compose -f docker-compose.micro.yml up --build -d auth-service vacancies-service profiles-service gateway

Инициализация БД:

Включить расширение UUID в Postgres (однократно).

Выполнить миграции:

docker exec -it auth_service npm run migration:run:prod

docker exec -it profiles_service npm run migration:run:prod

**Вывод**

API разделено на три микросервиса (Auth, Vacancies, Profiles), маршрутизируется через Nginx-gateway, сервисы изолированы во внутренней сети Docker. Для управления зависимостями реализован IoC/DI по контрактам (интерфейс + токен). Документация доступна через gateway. Миграции и схемы БД разнесены по сервисам; для UUID требуется uuid-ossp. Архитектура стала модульной и расширяемой;