# САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

**Дисциплина:** Бэк-энд разработка Отчет

Реализация Миграция написанного API на микросервисную архитектуру

Выполнил: Петухов Семён

Группа К3339

Проверил: Добряков Д. И.

Санкт-Петербург 2025 г.

# Задача

* выделить самостоятельные модули в вашем приложении;
* провести разделение своего API на микросервисы (минимум, их должно быть 3);
* настроить сетевое взаимодействие между микросервисами.

# Ход работы

На основе реализованного ранее приложения было решена разделить приложение на 4 сервиса – user-service, skill-service, resume-service, vacancy-service

Разделив соответсвующим образом модели по сервисам, были реализованы изменения в роутах для корректной обработки запросов из gateway.

Были убраны связи м/у различными моделями разных систем в сущностях каждой системы, т.к. это требовала микросервисная архитектура. Такие ссылки были заменены на ID:number

import {  
 Entity,  
 PrimaryGeneratedColumn,  
 Column,  
} from "typeorm";  
  
export enum UserRole {  
 *SEEKER* = "соискатель",  
 *EMPLOYER* = "работодатель"  
}  
  
@Entity()  
export class User {  
 @PrimaryGeneratedColumn()  
 id!: number;  
  
 @Column({ type: 'varchar', length: 255 })  
 username!: string;  
  
 @Column({ type: 'varchar', unique: true, length: 255 })  
 email!: string;  
  
 @Column({ type: 'varchar', length: 255 })  
 password!: string;  
  
 @Column({  
 type: "enum",  
 enum: UserRole,  
 default: UserRole.*SEEKER* })  
 role!: UserRole;  
  
 @Column({ type: 'int', nullable: true })  
 companyId?: number | null; // внешний ID компании из Company-сервиса  
}

В контроллерах для было заменено прямое получение значений других систем, при помощи их моделей, на запрос по URL, при помощи axios

export const createUser = async (req: Request, res: Response) => {  
 try {  
 const { username, email, password, role, companyId } = req.body;  
  
 if (!username || !email || !password) {  
 return res.status(400).json({  
 message: "Missing required fields: username, email, or password",  
 });  
 }  
  
 const existingUser = await userRepo.findOneBy({ email: email.trim() });  
 if (existingUser) {  
 return res.status(409).json({ message: "User with this email already exists" });  
 }  
  
 if (role && !*Object*.values(UserRole).includes(role)) {  
 return res.status(400).json({ message: `Invalid role. Valid roles: ${*Object*.values(UserRole).join(", ")}` });  
 }  
 if (companyId) {  
 const companyResponse = await axios.get(`http://localhost:3003/vacancy-service/company/${companyId}`);  
 if (companyResponse.status !== 200 || !companyResponse.data) {  
 return res.status(400).json({ message: "Invalid company ID" });  
 }  
 }

К каждому роуты сервиса был добавлен префикс сервиса, чтобы можно было разделять модели (без такого разделения swagger не работал)

app.use('/user-service/api-docs', *swaggerUi*.*serve*, *swaggerUi*.setup(*swaggerSpec*));  
app.use("/user-service/user", *userRoutes*);

Остальные сервисы были реализованы аналогично.

Далее был настроен gateway для объединения сервисов.

Были созданы прокси связи для вызова URL из одного места.

import express from "express";  
import { createProxyMiddleware } from "http-proxy-middleware";  
  
const app = express();  
  
app.use("/user-service", createProxyMiddleware({target: "http://localhost:3001", changeOrigin: true,}));  
app.use("/resume-service", createProxyMiddleware({target: "http://localhost:3002", changeOrigin: true,}));  
app.use("/vacancy-service", createProxyMiddleware({target: "http://localhost:3003", changeOrigin: true,}));  
app.use("/skill-service", createProxyMiddleware({target: "http://localhost:3005", changeOrigin: true,}));  
  
app.listen(3000, () => {  
 *console*.log("API Gateway running on port 3000");  
});

# Вывод

По результатам работы система была мигрирована на микросервисную инфраструктуру