# Developer Recruitment Task

AI HR OS · Backend Engineer Assessment

NestJS | Node.js | PostgreSQL | Multi-Tenancy | Microservices | REST API

*Estimated time: 3–4 hours · Submit via GitHub Pull Request*

## What You'll Build

You will build a small but production-style backend service for the AI HR OS platform — a multi-tenant Job Posting API using NestJS, PostgreSQL, and clean microservice principles.

This task tests how you structure a NestJS service, handle multi-tenancy, write clean APIs, and work with a real database.

---

## The Task — Multi-Tenant Job Posting Service

### What to Build

Create a NestJS microservice called jobs-svc that allows multiple tenants (companies) to manage their job postings. Each tenant's data must be completely isolated.

### Core Features Required

| # | Feature | Details |
|---|---|---|
| 1 | Tenant Isolation | Every API call must include a tenant_id (via header or JWT claim). Tenants must never see each other's data. |
| 2 | Create a Job | POST /jobs — create a job posting with: title, department, location, employment_type (full-time/part-time/contract), salary_min, salary_max, description, status (draft/published). |
| 3 | List Jobs | GET /jobs — return only the calling tenant's jobs. Support optional query filters: status, department, location. |
| 4 | Get a Job | GET /jobs/:id — return a single job. Return 404 if not found or belongs to a different tenant. |
| 5 | Update a Job | PATCH /jobs/:id — partial update. Only the owning tenant can update. |
| 6 | Close a Job | DELETE /jobs/:id — soft delete (set status = 'closed', do not destroy the row). |

---

## Database Schema

Use PostgreSQL with TypeORM (or Drizzle). Create the following tables:

### tenants

```
id          UUID PRIMARY KEY DEFAULT gen_random_uuid()
name        VARCHAR(255) NOT NULL
```

```
slug        VARCHAR(100) UNIQUE NOT NULL   -- e.g. 'acme-corp'
created_at  TIMESTAMP DEFAULT NOW()
```

### jobs

```
id               UUID PRIMARY KEY DEFAULT gen_random_uuid()
tenant_id        UUID NOT NULL REFERENCES tenants(id)
title            VARCHAR(255) NOT NULL
department       VARCHAR(100)
location         VARCHAR(150)
employment_type  VARCHAR(50)  -- full-time | part-time | contract
salary_min       INTEGER
salary_max       INTEGER
description      TEXT
status           VARCHAR(50) DEFAULT 'draft'  -- draft | published | closed
created_at       TIMESTAMP DEFAULT NOW()
updated_at       TIMESTAMP DEFAULT NOW()
```

**✅ Important: Add a database-level index on (tenant_id) on the jobs table.**

---

## 🔒 Multi-Tenancy Requirement

Use the Shared Database, Shared Schema approach (single PostgreSQL DB, tenant_id column on every table). This is the approach used in the AI HR OS platform.

Implement a TenantGuard or TenantInterceptor in NestJS that:

- Reads the X-Tenant-ID header from every incoming request
- Attaches the tenant_id to the request context
- Returns HTTP 400 if the header is missing

Every service method that queries jobs must scope the query by tenant_id. Example:

```
this.jobRepo.find({ where: { tenant_id: tenantId, id: jobId } })
```

**⛔ A query that returns jobs across tenants is an automatic fail.**

---

## 📁 Suggested Project Structure

```
jobs-svc/
├── src/
│   ├── main.ts
│   ├── app.module.ts
│   ├── tenants/
│   │   ├── tenant.entity.ts
│   │   └── tenant.guard.ts          ← reads X-Tenant-ID header
│   ├── jobs/
│   │   ├── jobs.module.ts
│   │   ├── jobs.controller.ts
│   │   ├── jobs.service.ts
```

```
│   │   ├── job.entity.ts
│   │   └── dto/
│   │       ├── create-job.dto.ts
│   │       └── update-job.dto.ts
│   └── common/
│       └── interceptors/tenant.interceptor.ts
├── .env.example
├── docker-compose.yml              ← must work with docker-compose up
└── README.md
```

## ✓ Evaluation Criteria

| Area | What We Look For | Weight |
|---|---|---|
| Multi-Tenancy | tenant_id scoping on ALL queries, TenantGuard implemented correctly, no cross-tenant data leakage | 30% |
| NestJS Structure | Proper use of modules, controllers, services, DTOs, guards/interceptors, dependency injection | 25% |
| Database & TypeORM | Correct entities, relations, migrations or sync, index on tenant_id | 20% |
| API Design | RESTful routes, correct HTTP status codes, input validation with class-validator | 15% |
| Code Quality | Clean, readable code, no console.log in production paths, .env used for config | 10% |

## ★ Bonus (Optional — Not Required)

- Add JWT authentication — tenant_id extracted from JWT claim instead of header
- Write at least 2 unit tests using Jest for the JobsService
- Add a simple Swagger/OpenAPI spec (@nestjs/swagger)
- Use a database migration file instead of synchronize: true

## How to Submit

### Step 1 — Fork the Repo

- Go to: https://github.com/nexadev-io/ai-hr-os-assessment
- Click Fork → create your fork under your own GitHub account
- Clone it: git clone https://github.com/<your-username>/ai-hr-os-assessment.git

### Step 2 — Create Your Branch

```
git checkout -b feature/jobs-svc-<your-name>
```

Example: git checkout -b feature/jobs-svc-rahman-ali

### Step 3 — Build & Commit

- Build your service inside the jobs-svc/ folder
- Commit regularly with clear messages:

```
git commit -m "feat(jobs): add tenant guard and job entity"
git commit -m "feat(jobs): implement CRUD endpoints with tenant scoping"
git commit -m "feat(jobs): add input validation with class-validator"
```

### Step 4 — Add a README

Your jobs-svc/README.md must include:

- How to run locally (docker-compose up should be enough)
- How to test the endpoints (curl examples or Postman collection)
- Any design decisions or trade-offs you made

### Step 5 — Open a Pull Request

- Push your branch: git push origin feature/jobs-svc-<your-name>
- Go to your fork on GitHub → click Compare & pull request
- Target: nexadev-io/ai-hr-os-assessment → main

PR Title format:

```
[Assessment] Jobs Service — <Your Full Name>
```

PR Description — paste this template:

```
## What I Built Brief description of your implementation...  ## How to Run docker-compose up
## Test Tenant Isolation # Create jobs for Tenant A curl -X POST http://localhost:3000/jobs \
-H 'X-Tenant-ID: tenant-a-uuid' \   -H 'Content-Type: application/json' \   -d '{"title":
"Backend Dev", "status": "published"}'  ## Decisions / Trade-offs - Why I chose X over Y...  ##
Checklist - [ ] docker-compose up works - [ ] All 6 endpoints implemented - [ ] tenant_id
scoped on all queries - [ ] Input validation added - [ ] README complete
```

## Rules & Notes

- Do NOT commit .env files with real secrets — use .env.example
- docker-compose up must start the service and database with zero manual steps
- Use TypeScript — no plain JavaScript
- You can use any NestJS-compatible ORM (TypeORM recommended, Prisma accepted)
- Do NOT use any SaaS multi-tenancy library — implement it yourself so we see your thinking
- Submission deadline: 1 March 2026 — Late submissions will not be reviewed

**Questions? Email hello@nexadev.io**

*Good luck! We review every submission carefully.*