Backend & Infrastructure Take-Home Test

Org Chart Service & Containerized Deployment

Mission

Build a minimal backend service using **FastAPI** that manages **multiple organization charts**, including robust hierarchical operations and multi-tenant storage. The goal is to demonstrate clear backend logic, data-modeling that scales to *10,000 distinct org charts*, containerized deployment with **Docker** and **PostgreSQL**, and configuration-management best practices. Your solution should expose CRUD operations, hierarchy queries, and deletion rules that keep every chart valid at all times.

Time Expectation

- **Target:** ≤ **2 hours** for an experienced developer to finish core functionality, plus **½ hour** for documentation & polish (total ≤ 2.5 h).

- **Time Log:** Include an approximate breakdown of the time you spent on each major area (API logic, containerization, database setup, performance demo, documentation, etc.).

Contact for Questions

For clarification at any point, email **meni@thebizconquest.com**.

Requirements

1. Service Functionality (Multi-Org API with Hierarchy)

Data Model

Table	Columns	Notes
org_charts	`id` (PK, int), `name` (string)	Represents an independent organization.
employees	`id` (PK, int) `org_id` (FK → org_charts.id) `name` (string), `title` (string), `manager_id` (FK → employees.id, nullable)	Self-referencing hierarchy. (`manager_id=NULL` means CEO).

Core Rules

1. **Valid Hierarchy** – no cycles; every employee (except CEO) must have a manager in the same org.
2. **Delete with Re-parenting** – Deleting a non-CEO employee *must* atomically re-assign **all** of their direct (and indirect) reports to the deleted employee's own manager.
3. **CEO Protection** – The CEO **cannot be deleted**. To change the CEO, implement **`PUT /orgcharts/{org_id}/employees/{employee_id}/promote`** (or similar) that promotes an existing employee to CEO, automatically demoting/re-parenting the previous CEO.

API Endpoints (minimum)

```
Unset
POST   /orgcharts                      # create a new org chart
GET    /orgcharts                      # list org charts

POST   /orgcharts/{org_id}/employees   # create employee in an org
GET    /orgcharts/{org_id}/employees   # list employees in org
GET    /orgcharts/{org_id}/employees/{id}
PUT    /orgcharts/{org_id}/employees/{id}
DELETE /orgcharts/{org_id}/employees/{id}
```

```
# choose ONE hierarchy view:
GET    /orgcharts/{org_id}/employees/{id}/direct_reports
   or
GET    /orgcharts/{org_id}/employees/{id}/manager_chain

# CEO replacement
PUT    /orgcharts/{org_id}/employees/{id}/promote   # makes {id}
the new CEO
```

Use clear, conventional HTTP status codes (`400` for validation errors, `409` for constraint violations, etc.).

Scale Demonstration (10 k org charts)

- Provide a **seed script** (bash/python) that creates **10,000 org charts**, each with **5-15 employees** (CEO & 1-2 layers of hierarchy is sufficient).

- After seeding, the following queries **MUST finish in <2 seconds** on a typical developer laptop:
    - `GET /orgcharts/{org_id}/employees` (random org)
    - your chosen hierarchy endpoint (`direct_reports` *or* `manager_chain`)

A simple timing log in your README is fine, no formal benchmarks required.

2. Containerization & Deployment

- **Dockerfile** – builds your FastAPI app.
- **docker-compose.yml** – launches:

    - `api` service (FastAPI, uvicorn)
    - `db` service (PostgreSQL ≥14)

- Use `ENV` variables for DB host, port, user, pwd, DB name.
- Include an **init script** or **alembic migration** that runs automatically on container start-up.

3. Configuration for Environments

Show how *dev* vs. *production* settings would differ (e.g., `DEBUG`, DB creds) via environment variables. Mention how you would store secrets (Docker secrets, vault, AWS SM) in production.

4. Basic Security & Best Practices

- **No authentication** needed for the test, but structure code so it could be added later.
- Validate foreign keys and prevent cycles on `POST/PUT`.
- Never commit secrets.

5. Documentation & Explanation

Create a concise **README.md** covering:

1. **Quick-start** with docker-compose (build & run).
2. **DB initialization** & seeding the 10 k org charts.
3. **Hierarchy logic** – which endpoint you implemented and why.
4. **Deletion & CEO rules** – describe the transaction.
5. **Scaling notes** – indices you added, why queries remain fast.
6. **Performance evidence** – paste short timing output from your laptop.
7. **Potential enhancements** (as applicable).
8. **AI usage** – one line on where it helped, if applicable.
9. **Open issues** – add a list of missing parts, known bugs, etc.

Deliverables

- **Source Code** – FastAPI app & models.
- **Container Artifacts** – `Dockerfile`, `docker-compose.yml`, migration/DDL scripts.
- **Seed Script** – to populate 10 000 org charts.
- **Documentation** – `README.md` & time log.
- **Short video demo** – short recording of you explaining how your solution successfully performs at scale.
- **Bonus points** – **only** if you are confident your solution works, feel free to add extra things (UI/authentication/data analysis query) for extra credit.

Where to submit: push to a GitHub repo and send the link to **meni@thebizconquest.com**

Evaluation Criteria

Area	What We Look For
Backend Logic	Correct hierarchy operations, CEO protection, cycle prevention.
Multi-Tenant Design	Clean org isolation, efficient queries, indices.
Performance Demo	Queries on 10 k org charts meet target times.
Dockerization	Containers build & start with one command; env-var config.
Code Quality	Structure, readability, naming, typing, comments.
Documentation	Clear, minimal steps from clone → running tests.
Practicality	Completes within stated time, avoids unnecessary complexity.
Explanations	You must be able to explain all your decisions (AI said so is not a good explanation)

Suggested Road-map

- Ask your favorite AI developer tool to help create a structure for you
- Make sure you are happy with suggestions

```python
while not completed:
    implement missing parts/fix bugs/add your logic/consult AI
```

Tips

- Finish **core rules first**, then aim for clean code.
- Use migrations to avoid "it works on my machine" schema drift.
- Keep containers stateless; rely on DB for persistence.
- AI tools are **strongly encouraged**, use them smartly to stay within time.

Good luck—we're excited to see what you build!