📘 Tech Lead Assignment – TalentAdore

Welcome to the TalentAdore Tech Lead assignment! This task simulates the kind of technical decisions, architecture work, and coding quality expected from a Tech Lead. You'll design and implement a small microservice-based system with solid API structure, testing, and documentation.

Please do not spend more than 5–10 hours on the assignment, depending on your approach and experience level. AI tools are permitted and encouraged if they help you complete the assignment more effectively.

The assignment's broad scope means you're not expected to complete every task. Instead, it's designed to challenge you to prioritize, make thoughtful decisions, and clearly explain your reasoning.

We believe that great software is built with passion, collaboration, and attention to detail. While completing the assignment, we encourage you to focus on writing clean, maintainable code. Your ability to manage scope and quality is equally important as the end product.

You'll have time until the end of this week. At the latest on Sunday, please provide us the task with information about:

- The total time you spent
- A brief status summary of what you did
- Any tasks you skipped or deprioritized, and why

Submit this information to Oskari (oskari.valkama@talentadore.com), Muhammad (muhammad.naeem@talentadore.com, Owais (owais.afaq@talentadore.com) and Saara (saara.kyllonen@talentadore.com) via email.

Next, you'll have a discussion with Oskari & Muhammad about the task & our technical situation.

👇👇👇Good luck with the task! 👇👇👇

👨‍💼 Context

At TalentAdore, we're building an AI-powered recruitment platform. As a Tech Lead, you will mentor a team of developers and drive backend design decisions, DevOps practices, and cross-team collaboration.

This assignment lets you showcase your hands-on technical ability, architectural thinking, and leadership mindset.

🚀 The challenge

Build two separate microservices:

- A **Feedback Service** to manage feedbacks on organizations
- A **Member Service** to manage members of organizations

The services must:

- Be dockerized
- Expose RESTful APIs
- Have setup instructions and API documentation
- Include database seeding where necessary for testing/demo purposes

You must expose **all API endpoints under a single unified base URL** and **internally route requests** to the correct microservice.

🧱 Architecture Requirements

You must build **three containers**:

1. Comment-service

Handles:

    - Creating, retrieving, and soft-deleting feedback
    - Stores data in PostgreSQL or MariaDB

2. Member-service

- Handles:

    - Creating, retrieving, and soft-deleting members
    - Stores data in PostgreSQL or MariaDB

3. Gateway-service (entrypoint)

- All requests go through this service
- Routes internally to the appropriate service (e.g., using HTTP proxying)

📡 API Endpoints (Exposed via gateway-service)

All requests below go to the single base service (http://localhost:8000 or similar) and are routed internally.

✅ POST

Create feedback for the organization.

Request Body:

{

"feedback": "Great team culture, clear communication, and strong support for growth."

}

✅ GET

Get all **non-deleted** feedbacks for the organization.

✅ DELETE

Soft delete all feedbacks for the organization.

✅ POST

Create a new member.

Request Body:

```
{
 "first_name": "John",
 "last_name": "Doe",
 "login": "john123",
 "avatar_url": "https://example.com/avatar.jpg",
 "followers": 120,
 "following": 35,
 "title": "Senior Developer",
 "email": "john@example.com"
}
```


✅ **GET**

Get all non-deleted members for the organization, sorted by followers descending.

✅ **DELETE**

Soft delete all members of the organization.

✅ **Must Include**

Docker Setup

- Dockerfile for each service
- docker-compose.yml to spin up gateway, services, and databases

Routing

- All APIs are exposed via the gateway-service
- Internally proxy (e.g., via Flask routes, reverse proxy, or request forwarding)

Swagger API Docs

- Swagger/OpenAPI on gateway or each service
- At least document all routes at the gateway

README.md

- Setup instructions using Docker
- How to run tests
- How to access APIs and Swagger docs
- How to seed sample data

Secrets & Env

- Use .env and os.environ for config and secrets

Database Seeding

- Seed initial data (e.g., 1–2 orgs and members) for demo/test use

🌟 Bonus Points

- Add authentication/authorization for comment and members routes
- Include unit tests and integration tests between services