Django REST + Docker Test Assignment

Goal: Assess the candidate's ability to work under ambiguity, make reasonable assumptions, and deliver a working, containerized Django REST API quickly.

You are expected and encouraged to use your preferred **AI code assistant** for all your work.

THIS ASSIGNMENT MUST BE COMPLETED IN LESS THAN 6 HOURS OF WORK.

Scope of Work

Implement a Django REST API that provides endpoints for managing the following resources:

- **Accounts**
- **Organizations**
- **Users**
- **Teams**
- **One additional resource of your choice** (*choose something you find interesting/challenging; justify in the README*).

General Requirements

- **Permissions**: Implement proper permission logic for the endpoints (e.g., restrict who can create/update/delete resources).
- **Soft Deletion**: All resources must support soft deletion instead of hard deletion.
- **Database**: Use Postgres as the database (through Docker Compose).

Deliverables

Submit a Git repository containing:

1. **Docker setup**:
   - `Dockerfile` and `docker-compose.yml`.
   - Must be runnable with one command, exposing the service at **http://localhost:8000**.
2. **Django REST Framework setup**:
   - Models, migrations, and DRF endpoints for the required resources.
   - Reasonable permissions and validations.
3. **Test Suite**:
   - Implemented with **pytest**.
   - Must cover both happy-path and edge cases.
4. **README.md**:
   - Run instructions (build, run, test).
   - Clear assumptions made where requirements were ambiguous.
   - API usage examples (curl or HTTPie).

Evaluation Criteria

1. **Proactiveness**
   - How the candidate handles missing requirements and ambiguities.
   - Documenting assumptions and choices.
2. **Speed of Delivery**
   - Ability to ship a working solution in a short timeframe.
   - The number of hours the candidates needed to complete the assignment
3. **Code Quality & Organization**
   - Project structure, readability, adherence to Django/DRF conventions.
4. **Test Suite & Documentation**
   - Coverage, clarity, and usefulness of tests.
   - Quality of README and explanations.
5. **Proper Git Usage**
   - Meaningful commits, logical history, clean repository.