Тестовое задание: API организационной структуры

Модели

Department — подразделение

- id: int
- name: str (не пустой)
- parent_id: int | null (FK на Department, позволяет строить дерево)
- created_at: datetime

Employee — сотрудник

- id: int
- department_id: int (FK на Department)
- full_name: str (не пустой)
- position: str (не пустой)
- hired_at: date | null (опционально)
- created_at: datetime

Связи

- Department 1—N Employee
- Department 1—N Department (самоссылка через parent_id)

Методы API

1) Создать подразделение

POST /departments/

- Body:
  - name: str
  - parent_id: int | null (опционально)
- Response: созданное подразделение

---

2) Создать сотрудника в подразделении

POST /departments/{id}/employees/

- Body:
    - full_name: str
    - position: str
    - hired_at: date | null (опционально)
- Response: созданный сотрудник

---

3) Получить подразделение (детали + сотрудники + поддерево)

GET /departments/{id}

- Query:
    - depth: int (по умолчанию 1, максимум 5) — глубина вложенных подразделений в ответе
    - include_employees: bool (по умолчанию true)
- Response:
    - department (объект подразделения)
    - employees: [] (если include_employees=true, сортировка по created_at или full_name)
    - children: [] (вложенные подразделения до depth, рекурсивно)

---

4) Переместить подразделение в другое (изменить parent)

PATCH /departments/{id}

- Body:
    - name: str (опционально)
    - parent_id: int | null (опционально)
- Response: обновлённое подразделение

---

5) Удалить подразделение

DELETE /departments/{id}

- Query:

    - mode: str (cascade | reassign)
        - cascade — удалить подразделение, всех сотрудников и все дочерние подразделения
        - reassign — удалить подразделение, а сотрудников перевести в reassign_to_department_id
    - reassign_to_department_id: int (обязателен, если mode=reassign)
- Response: 204 No Content (или json-статус)

Логика и ограничения

- Нельзя создать сотрудника в несуществующем подразделении (404).
- name подразделения:
    - не пустой, длина 1..200
    - пробелы по краям должны триммиться (опционально, но приветствуется)
    - **в пределах одного parent** названия должны быть уникальны (например, два “Backend” в одном родителе нельзя)
- full_name:
    - не пустой, длина 1..200
- position:
    - не пустой, длина 1..200
- Нельзя сделать подразделение родителем самого себя.
- Нельзя создать цикл в дереве (например, переместить департамент внутрь своего поддерева) — возвращать 409 Conflict (или 400).
- GET /departments/{id} должен корректно отдавать дерево до depth
- При удалении в режиме cascade удаление должно быть каскадным на уровне БД/ORM.

⚙️ Технические требования

- Использовать **net/http**
- Работа с БД через **GORM**.
- Использовать **PostgreSQL**.
- Использовать миграции с помощью **goose**.
- Обернуть приложение в **Docker** и запустить через **docker-compose**.
- Приветствуется: логгирование, тесты (**httptest**/**testify**)

📦 Что нужно предоставить

- Ссылку на репозиторий (GitHub/GitLab).
- `README.md` с инструкцией по запуску (`docker-compose up`) и описанием проекта.

🧠 Критерии оценки:

- Архитектура проекта.
- Читаемость и качество кода.
- Корректность бизнес-логики (валидация, каскадное удаление).
- Работа с Docker и docker-compose.
- Наличие тестов и миграций (хотя бы одного теста и одной миграции).

⚠️ Частые ошибки:

1. Не структурированный проект (все в одном или двух файлах, отсутствие модульности).
2. Нечитаемый код (дублирование функций, смешивание бизнес-логики и работы с БД).
3. Отсутствие миграций
5. Нет описания запуска проекта в README.md.