Esercizio FastAPI Base

Relazioni 1:N e 1:1

Corso FastAPI per Principianti

Descrizione dello Scenario

Si vuole realizzare un semplice backend per la gestione del personale di un'azienda. Il sistema deve tracciare i dipartimenti, i dipendenti e i loro badge di accesso (tesserini).

Questo esercizio si focalizza sulla comprensione delle relazioni fondamentali senza la complessità delle tabelle di associazione.

1 Specifiche del Database

Il database è composto da 3 tabelle.

1. Le Tabelle

- **Department** (Dipartimento):
  - `id`: Integer, Primary Key.
  - `name`: String (es. "Amministrazione", "IT").
  - **Relazione**: Un dipartimento contiene molti dipendenti.
- **Employee** (Dipendente):
  - `id`: Integer, Primary Key.
  - `full_name`: String.
  - `role`: String (es. "Junior Dev", "Manager").
  - **Relazione A (1:N)**: Appartiene a **un solo** Department.
  - **Relazione B (1:1)**: Possiede **un solo** Badge personale.
- **Badge** (Tesserino):
  - `id`: Integer, Primary Key.
  - `code`: String (unico, es. "BDG-1234").
  - `expiration_date`: Date (o String).
  - **Relazione**: Appartiene a **un solo** Employee.

2 Schema delle Relazioni

Riepilogo Relazioni

1. **Department** ↔ **Employee**: Relazione **Uno a Molti** (One-to-Many).
   - 1 Dipartimento → N Dipendenti.
2. **Employee** ↔ **Badge**: Relazione **Uno a Uno** (One-to-One).
   - 1 Dipendente ↔ 1 Badge esatto.
   - Nessun badge può essere condiviso.

3 Requisiti dell'Esercizio

3.1 1. Configurazione e Modelli (`models.py`)

Definisci le classi SQLAlchemy. Presta particolare attenzione alla relazione 1:1 tra Employee e Badge.

Suggerimento: Relazione 1 a 1

In SQLAlchemy, per forzare una relazione 1:1 invece di 1:N, si usa il parametro `uselist=False` nella funzione `relationship()`.

```
class Employee(Base):
    # ... altri campi ...
    # uselist=False dice: "Non dammi una lista, dammi un oggetto singolo"
    badge = relationship("Badge", back_populates="owner", uselist=False)

class Badge(Base):
    # ...
    employee_id = Column(Integer, ForeignKey("employees.id"), unique=True)
    owner = relationship("Employee", back_populates="badge")
```

3.2 2. Schemi Pydantic (`schemas.py`)

Crea gli schemi per la validazione.

- **EmployeeResponse**: Quando richiedo un dipendente, voglio vedere automaticamente i dettagli del suo Badge (se ne ha uno) e il nome del Dipartimento.
- **DepartmentResponse**: Quando richiedo un dipartimento, voglio la lista dei dipendenti che ci lavorano.

3.3 3. Logica CRUD (`crud.py`)

Implementa le funzioni per:

- Creare un Dipartimento.
- Creare un Dipendente (associandolo a un dipartimento tramite ID).
- Creare un Badge (associandolo a un dipendente tramite ID).

- **Get Employee**: Restituire un dipendente con un Join automatico sul Badge.

3.4 4. API Endpoints (`main.py`)

Esponi le seguenti rotte:

- `POST /departments/`
- `POST /employees/`
- `POST /badges/`: Attenzione, se provo ad assegnare un badge a un dipendente che ne ha già uno, il database dovrebbe dare errore (se hai messo `unique=True` nel modello) o la logica dovrebbe impedirlo.
- `GET /employees/{id}`: Deve mostrare JSON annidato:

```
{
  "id": 1,
  "full_name": "Mario Rossi",
  "role": "Developer",
  "department_id": 2,
  "badge": {
    "code": "XYZ-999",
    "expiration_date": "2024-12-31"
  }
}
```