Backend Intern Take-Home Assignment

Role: Backend Intern

Technology Stack: Java, Spring Boot, Spring Data JPA, H2 Database

Focus Areas: RESTful APIs, Database Relationships (1:1, 1:N, N:M), JPA Annotations

Assignment: Task Management API

Build a RESTful API for a simple task management system using Java and Spring Boot.

Entity Relationship Overview

Your database should implement the following relationships:

```
User (1) -----→ (*) Task
  ↓
  |
  └──→ (*) Comment

Task (1) -----→ (*) Comment

Task (*) ←-----→ (*) Tag
     (Many-to-Many via join table)

Task (1) -----→ (1) TaskMetadata [Optional]
```

Key Points:

- Use `@OneToMany` and `@ManyToOne` for User-Task and Task-Comment relationships
- Use `@ManyToMany` with `@JoinTable` for Task-Tag relationship
- Use `@OneToOne` with `mappedBy` for Task-TaskMetadata relationship
- Consider cascade types carefully (e.g., `CascadeType.ALL` for comments when deleting task)
- Use `@JsonIgnore` or `@JsonManagedReference/@JsonBackReference` to prevent circular references in JSON responses

---

Requirements

Core Features (Must Have)

1. **Create a Task**
   - Fields: title (required), description (optional), status (TODO/IN_PROGRESS/COMPLETED), createdAt, updatedAt
   - Validate that title is not empty
2. **Get All Tasks**
   - Return a list of all tasks
   - Support filtering by status (query parameter)
3. **Get Task by ID**
   - Return a single task by its ID
   - Handle case when task is not found
4. **Update Task**
   - Allow updating title, description, and status
   - Update the `updatedAt` timestamp
5. **Delete Task**
   - Remove a task by ID

Technical Requirements

- Use **Spring Boot 3.x** (or 2.7+)
- Use **Spring Data JPA** for database operations
- Use **Use SQL database**
- Implement proper **HTTP status codes** (200, 201, 404, 400, etc.)
- Add basic **input validation**
- Write **clean, readable code** with proper naming conventions

Database Relationships (Important)

Implement the following JPA relationships to demonstrate your understanding of database modeling:

1. **One-to-Many: User → Tasks**
   - Create a `User` entity with fields: id, name, email
   - Each task belongs to one user (creator/assignee)
   - A user can have multiple tasks
   - Add endpoint: `GET /api/users/{userId}/tasks` - Get all tasks for a user
2. **One-to-Many: Task → Comments**
   - Create a `Comment` entity with fields: id, text, createdAt, task, user
   - Each task can have multiple comments
   - Each comment belongs to one task and one user
   - Add endpoints:
     - `POST /api/tasks/{taskId}/comments` - Add comment to a task
     - `GET /api/tasks/{taskId}/comments` - Get all comments for a task
3. **Many-to-Many: Tasks ↔ Tags**
   - Create a `Tag` entity with fields: id, name
   - Tasks can have multiple tags (e.g., "urgent", "bug", "feature")
   - Tags can be associated with multiple tasks
   - Add endpoints:
     - `POST /api/tags` - Create a new tag

- `POST /api/tasks/{taskId}/tags/{tagId}` - Assign tag to task
- `DELETE /api/tasks/{taskId}/tags/{tagId}` - Remove tag from task
- `GET /api/tasks?tag={tagName}` - Filter tasks by tag

4. **One-to-One: Task → TaskMetadata** (Optional)
   - Create `TaskMetadata` entity: id, estimatedHours, actualHours, priority
   - Each task has one metadata record
   - Use `@OneToOne` with cascade operations

Additional Bonus Points

- Add unit tests for service layer
- Implement pagination for GET all tasks
- Add API documentation (Swagger/OpenAPI)
- Custom exception handling with meaningful error messages
- Prevent orphan comments when a task is deleted

Deliverables

1. **Source Code**
   - Push your code to a **GitHub repository** (public or private)
   - Share the repository link
2. **README.md** should include:
   - Instructions to run the application
   - API endpoints documentation (or Postman collection)
   - **Database schema explanation** - Describe the relationships between entities
   - Any assumptions you made
   - Technologies/dependencies used
3. **Sample API Requests** (in README or Postman collection)

- Include examples for creating users, tasks, comments, and tags
- Show examples of filtering tasks by tags

Sample API Endpoints

```
Tasks:
POST   /api/tasks              - Create a new task
GET    /api/tasks              - Get all tasks (with optional ?status=TODO&ta
GET    /api/tasks/{id}         - Get task by ID
PUT    /api/tasks/{id}         - Update a task
DELETE /api/tasks/{id}         - Delete a task

Users:
POST   /api/users              - Create a new user
GET    /api/users              - Get all users
GET    /api/users/{userId}/tasks - Get all tasks for a specific user

Comments:
POST   /api/tasks/{taskId}/comments  - Add comment to a task
GET    /api/tasks/{taskId}/comments  - Get all comments for a task

Tags:
POST   /api/tags                           - Create a new tag
GET    /api/tags                           - Get all tags
POST   /api/tasks/{taskId}/tags/{tagId}    - Assign tag to task
DELETE /api/tasks/{taskId}/tags/{tagId}    - Remove tag from task
```

Sample Request/Response Examples

1. Create a Task

POST /api/tasks

```
Request:
{
  "title": "Complete assignment",
  "description": "Finish the backend intern assignment",
  "status": "TODO",
  "userId": 1
}
```

```
Response (201 Created):
{
  "id": 1,
  "title": "Complete assignment",
  "description": "Finish the backend intern assignment",
  "status": "TODO",
  "user": {
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com"
  },
  "tags": [],
  "createdAt": "2025-10-07T10:30:00",
  "updatedAt": "2025-10-07T10:30:00"
}
```

2. Add Comment to Task

POST /api/tasks/1/comments

```
Request:
{
  "text": "Started working on this task",
  "userId": 1
}

Response (201 Created):
{
  "id": 1,
  "text": "Started working on this task",
  "user": {
    "id": 1,
    "name": "John Doe"
  },
  "createdAt": "2025-10-07T11:00:00"
}
```

3. Assign Tag to Task

POST /api/tasks/1/tags/2

```
Response (200 OK):
{
  "id": 1,
```

```
  "title": "Complete assignment",
  "tags": [
    {
      "id": 2,
      "name": "urgent"
    }
  ]
}
```

Submission Guidelines

- Submit via email with:
  - GitHub repository link
  - Brief note about your approach (2-3 sentences)

Good luck! 🚀
