Project Requirements: Task Management System (TODO List) with Clean Architecture

Project Type: Spring Boot REST API with Clean Architecture

Goal: Practice Clean Architecture principles with a simple but complete CRUD application

Technical Stack

- **Framework**: Spring Boot 4.0.1
- **Language**: Java 21
- **Build Tool**: Gradle - Groovy
- **Database**: H2 (in-memory)
- **Dependencies**: Spring Web, Spring Data JPA, H2 Database, Lombok, Validation, DevTools

Architecture Requirements

Must follow Clean Architecture with 4 distinct layers:[2][1]

1. **Domain Layer** (innermost - no framework dependencies)
    - Contains pure business logic
    - No annotations from Spring or JPA
    - Independent of any framework
2. **Application Layer** (use cases)
    - Orchestrates domain logic
    - Contains use case implementations
    - Depends only on domain interfaces
3. **Infrastructure Layer** (external concerns)
    - JPA entities and repositories
    - Database implementations
    - Mappers between domain and persistence models
4. **Presentation Layer** (REST API)
    - REST controllers

- Request/Response DTOs
- HTTP concerns only

Domain Model

Entity: Task

- Properties:
  - `id` (Long) - unique identifier
  - `title` (String) - required, max 100 characters
  - `description` (String) - optional, max 500 characters
  - `status` (TaskStatus enum) - PENDING, IN_PROGRESS, COMPLETED, CANCELLED
  - `createdAt` (LocalDateTime) - timestamp of creation
  - `updatedAt` (LocalDateTime) - timestamp of last update

Business Rules to Implement in Domain:

1. Task can only be completed if status is PENDING or IN_PROGRESS
2. Task is considered overdue if PENDING for more than 7 days
3. Once CANCELLED, task cannot change status
4. Title is mandatory and cannot be empty
5. Completed tasks cannot be deleted

Use Cases to Implement

1. CreateTaskUseCase[3]

- Input: `CreateTaskRequest` (title, description)
- Output: `Task` (domain entity)
- Logic:
  - Validate title is not empty
  - Set status to PENDING
  - Set createdAt to current timestamp
  - Save and return task

2. UpdateTaskUseCase

- Input: taskId, `UpdateTaskRequest` (title, description)
- Output: `Task`
- Logic:
    - Find task by ID (throw TaskNotFoundException if not found)
    - Update title and description
    - Set updatedAt to current timestamp
    - Save and return updated task

3. CompleteTaskUseCase

- Input: taskId
- Output: `Task`
- Logic:
    - Find task by ID
    - Check if task can be completed (use domain method)
    - Change status to COMPLETED
    - Set updatedAt
    - Save and return task

4. ListTasksUseCase

- Input: optional status filter
- Output: List of `Task`
- Logic:
    - If status provided, filter by status
    - If no status, return all tasks
    - Return ordered by createdAt descending

5. DeleteTaskUseCase

- Input: taskId

- Output: void
- Logic:
    - Find task by ID
    - Check if task is not COMPLETED (business rule)
    - Delete task

Repository Interface (Domain Layer)

TaskRepository interface (must be in domain package, no JPA annotations):[2][3]

```
Methods needed:
- Task save(Task task)
- Optional<Task> findById(Long id)
- List<Task> findAll()
- List<Task> findByStatus(TaskStatus status)
- void deleteById(Long id)
- boolean existsById(Long id)
```

Infrastructure Implementation

JPA Implementation:[1]

- Create `TaskEntity` with JPA annotations (@Entity, @Table, @Id, etc)
- Create `JpaTaskRepository` interface extending `JpaRepository<TaskEntity, Long>`
- Create `TaskRepositoryImpl` that implements domain `TaskRepository` interface
- Use mapper to convert between `Task` (domain) and `TaskEntity` (JPA)
- `TaskMapper` should have methods: `toDomain(TaskEntity)` and `toEntity(Task)`

REST API Endpoints

Base path: `/api/tasks`

1. **POST /api/tasks** - Create new task
    - Request body: `{"title": "string", "description": "string"}`
    - Response: 201 Created with task object

2. **GET /api/tasks** - List all tasks
    - Query param: `status` (optional)
    - Response: 200 OK with array of tasks
3. **GET /api/tasks/{id}** - Get task by ID
    - Response: 200 OK with task object or 404 Not Found
4. **PUT /api/tasks/{id}** - Update task
    - Request body: `{"title": "string", "description": "string"}`
    - Response: 200 OK with updated task
5. **PATCH /api/tasks/{id}/complete** - Mark task as completed
    - Response: 200 OK with updated task
6. **DELETE /api/tasks/{id}** - Delete task
    - Response: 204 No Content or 400 if task is completed

DTOs Required

Request DTOs (in application layer):

- `CreateTaskRequest` - title, description (with validation annotations)
- `UpdateTaskRequest` - title, description (with validation annotations)

Response DTOs (in presentation layer):

- `TaskResponse` - id, title, description, status, createdAt, updatedAt
- Static factory method: `TaskResponse.from(Task task)`

Configuration

application.yml:

- H2 console enabled at `/h2-console`
- In-memory database with name `testdb`
- JPA show-sql enabled

- Server port 8080

UseCaseConfig class:

- @Configuration class that creates @Bean for each use case
- Injects TaskRepository into use case constructors
- Located in config package

Package Structure[1][2]

```
com.vinicius.cleanarch/
├─── domain/
│    ├─── entities/
│    │    ├─── Task.java
│    │    └─── TaskStatus.java (enum)
│    ├─── repositories/
│    │    └─── TaskRepository.java (interface)
│    └─── exceptions/
│         ├─── TaskNotFoundException.java
│         ├─── InvalidTaskException.java
│         └─── TaskCannotBeDeletedException.java
├─── application/
│    ├─── usecases/
│    │    ├─── CreateTaskUseCase.java
│    │    ├─── UpdateTaskUseCase.java
│    │    ├─── CompleteTaskUseCase.java
│    │    ├─── ListTasksUseCase.java
│    │    └─── DeleteTaskUseCase.java
│    └─── dto/
│         ├─── CreateTaskRequest.java
│         └─── UpdateTaskRequest.java
├─── infrastructure/
│    └─── persistence/
│         ├─── TaskEntity.java
│         ├─── JpaTaskRepository.java
│         ├─── TaskRepositoryImpl.java
│         └─── TaskMapper.java
├─── presentation/
│    ├─── controllers/
│    │    └─── TaskController.java
```

```
|    └──── dto/
|        └──── TaskResponse.java
└──── config/
    └──── UseCaseConfig.java
```

Exception Handling

Create `@RestControllerAdvice` class for global exception handling:

- `TaskNotFoundException` → 404 Not Found
- `InvalidTaskException` → 400 Bad Request
- `TaskCannotBeDeletedException` → 400 Bad Request
- `MethodArgumentNotValidException` → 400 Bad Request with validation errors

Important Constraints

Clean Architecture Rules to Enforce:[3][2][1]

1. Domain layer must have ZERO dependencies on Spring or JPA
2. Domain entities are POJOs (can use Lombok for boilerplate only)
3. Use cases depend only on domain repository interfaces, not implementations
4. Infrastructure implements domain interfaces, never the other way around
5. Controllers depend on use cases, not repositories directly
6. Mappers are only in infrastructure layer to keep domain pure

Testing Requirements (optional but recommended)

Create unit tests for:

- `CreateTaskUseCase` - test successful creation and validation failures
- Domain entity business rules (canBeCompleted(), isOverdue())
- Use Mockito to mock repository in use case tests
- No Spring context needed for use case tests (pure unit tests)