Actividad: API RESTful para Gestión de Tareas con Spring Boot y MongoDB.

Van a desarrollar una API RESTful completa para una aplicación de gestión de tareas tipo Google Keep, utilizando Spring Boot y MongoDB como base de datos. Esta API permitirá crear, leer, actualizar y eliminar tareas, así como organizarlas mediante etiquetas y categorías.

1. Configuración inicial del proyecto.

1.1. Crear el proyecto Spring Boot

- Accede a Spring Initializr
- Configura el proyecto con los siguientes parámetros:
  - **Project:** Maven
  - **Language:** Java
  - **Spring Boot:** última versión estable
  - **Group:** com.tuapellido.dam
  - **Artifact:** taskmanager-api
  - **Packaging:** Jar
  - **Java:** 17 o superior

1.2. Añadir las dependencias necesarias:

- Spring Web
- Spring Data MongoDB
- Lombok
- Validation (Spring Boot Starter Validation)
- Spring Boot DevTools

1.3. Descargar y descomprimir el proyecto en tu workspace

1.4. Configurar MongoDB

- Instala MongoDB Community Edition en tu equipo local O utiliza MongoDB Atlas (nube gratuita)
- Si usas MongoDB local: asegúrate de que el servicio esté corriendo en el puerto 27017
- Si usas MongoDB Atlas: crea un cluster gratuito y obtén la cadena de conexión

1.5. Configurar application.properties

```
# Configuración de MongoDB local
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=taskmanager

# O para MongoDB Atlas:
```

```
#    spring.data.mongodb.uri=mongodb+srv://usuario:password@cluster.mongodb.net/
taskmanager

# Configuración del servidor
server.port=8080

# Configuración de logs
logging.level.org.springframework.data.mongodb=DEBUG
```

2. Diseño y creación del modelo de datos.

2.1. Crear el paquete de modelos

- Crea el paquete: `com.tuapellido.dam.taskmanager.model`

2.2. Diseñar la entidad Task Crea la clase `Task.java` con los siguientes atributos:

- `id` (String): Identificador único generado por MongoDB
- `title` (String): Título de la tarea (obligatorio, máximo 100 caracteres)
- `description` (String): Descripción detallada (opcional, máximo 500 caracteres)
- `completed` (boolean): Estado de la tarea
- `priority` (enum Priority): BAJA, MEDIA, ALTA
- `tags` (List<String>): Lista de etiquetas
- `category` (String): Categoría de la tarea (Personal, Trabajo, Estudios, etc.)
- `createdAt` (LocalDateTime): Fecha de creación
- `updatedAt` (LocalDateTime): Fecha de última actualización
- `dueDate` (LocalDateTime): Fecha de vencimiento (opcional)

2.3. Aplicar anotaciones necesarias:

- `@Document(collection = "tasks")`: Para MongoDB
- `@Id`: Para el identificador
- `@NotBlank`, `@Size`, `@NotNull`: Para validaciones
- `@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`: De Lombok

2.4. Crear el enum Priority

- Crea `Priority.java` en el paquete model con los valores: BAJA, MEDIA, ALTA

3. Capa de repositorio.

3.1. Crear el paquete repository.

- Crea el paquete: `com.tuapellido.dam.taskmanager.repository`

3.2. Crear la interfaz TaskRepository

```
public interface TaskRepository extends MongoRepository<Task, String> {
    // Métodos de consulta personalizados
    List<Task> findByCompleted(boolean completed);
    List<Task> findByCategory(String category);
    List<Task> findByTagsContaining(String tag);
    List<Task> findByPriority(Priority priority);
    List<Task> findByTitleContainingIgnoreCase(String title);
    List<Task> findByDueDateBefore(LocalDateTime date);
}
```

4. Capa de servicio.

4.1. Crear el paquete service.

- Crea el paquete: `com.tuapellido.dam.taskmanager.service`

4.2. Crear la interfaz TaskService Define los métodos necesarios para:

- Crear una tarea
- Obtener todas las tareas
- Obtener una tarea por ID
- Actualizar una tarea
- Eliminar una tarea
- Buscar tareas por diferentes criterios (completadas, categoría, etiqueta, prioridad)
- Marcar tarea como completada/no completada

4.3. Implementar TaskServiceImpl

- Crea la clase `TaskServiceImpl` que implemente `TaskService`
- Anota con `@Service`
- Inyecta `TaskRepository` usando el constructor
- Implementa toda la lógica de negocio
- Gestiona el `updatedAt` automáticamente en las actualizaciones
- Lanza excepciones personalizadas cuando no se encuentre una tarea

5. Manejo de excepciones.

5.1. Crear el paquete exception.

- Crea el paquete: `com.tuapellido.dam.taskmanager.exception`

5.2. Crear excepción personalizada.

```
public class TaskNotFoundException extends RuntimeException {
    public TaskNotFoundException(String id) {
        super("No se encontró la tarea con ID: " + id);
    }
}
```

5.3. Crear clase para respuestas de error.

```
@Data
@AllArgsConstructor
public class ErrorResponse {
    private LocalDateTime timestamp;
    private int status;
    private String error;
    private String message;
    private String path;
}
```

5.4. Crear GlobalExceptionHandler.

- Anota con `@RestControllerAdvice`
- Maneja `TaskNotFoundException` (404)
- Maneja `MethodArgumentNotValidException` (400) para errores de validación
- Maneja `Exception` genérica (500)

6. Capa de controlador.

6.1. Crear el paquete controller.

- Crea el paquete: `com.tuapellido.dam.taskmanager.controller`

6.2. Crear TaskController.

- Anota con `@RestController` y `@RequestMapping("/api/tasks")`
- Inyecta `TaskService`
- Implementa los siguientes endpoints:

Método HTTP	Endpoint	Descripción
POST	/api/tasks	Crear nueva tarea
GET	/api/tasks	Obtener todas las tareas
GET	/api/tasks/{id}	Obtener tarea por ID
PUT	/api/tasks/{id}	Actualizar tarea completa
PATCH	/api/tasks/{id}/complete	Marcar como completada
PATCH	/api/tasks/{id}/incomplete	Marcar como no completada
DELETE	/api/tasks/{id}	Eliminar tarea
GET	/api/tasks/completed	Obtener tareas completadas
GET	/api/tasks/pending	Obtener tareas pendientes
GET	/api/tasks/category/{category}	Buscar por categoría
GET	/api/tasks/tag/{tag}	Buscar por etiqueta
GET	/api/tasks/priority/{priority}	Buscar por prioridad

Método HTTP	Endpoint	Descripción
GET	/api/tasks/search?title={title}	Buscar por título

6.3. Aplicar las anotaciones correctas:

- Usa `@Valid` para validar el cuerpo de las peticiones
- Devuelve códigos de estado HTTP apropiados: `@ResponseStatus`
- Utiliza `ResponseEntity` para mayor control de las respuestas

7. DTOs (Data Transfer Objects) - OPCIONAL pero recomendado.

7.1. Crear el paquete dto

- Si decides implementar DTOs, crea: `com.tuapellido.dam.taskmanager.dto`

7.2. Crear TaskRequestDTO y TaskResponseDTO

- Separa lo que recibes del cliente de lo que devuelves
- Implementa mappers para convertir entre entidades y DTOs

8. Pruebas y documentación.

8.1. Probar la API con Postman o Thunder Client

- Crea una colección con todas las peticiones
- Prueba todos los endpoints
- Verifica las validaciones
- Comprueba el manejo de errores

8.2. Documentar la API

- Crea un archivo README.md con:
    - Descripción del proyecto
    - Requisitos previos
    - Instrucciones de instalación
    - Documentación de todos los endpoints con ejemplos
    - Ejemplos de cuerpos de petición y respuesta en JSON

8.3. Poblar la base de datos con datos de prueba

- Crea al menos 10 tareas de ejemplo con diferentes estados, prioridades y categorías

9. Mejoras adicionales (OPCIONAL - Para nota de excelente)

- Implementar paginación en el listado de tareas
- Añadir filtros combinados (ej: tareas pendientes de alta prioridad)
- Implementar búsqueda avanzada con múltiples criterios

- Añadir un endpoint de estadísticas (total de tareas, completadas, por prioridad, etc.)
- Implementar ordenación (por fecha, prioridad, título)
- Añadir autenticación básica con Spring Security
- Documentar con Swagger/OpenAPI

Criterios de calificación

Funcionalidad (40%)

- **CRUD completo funcional (20%):** Todos los endpoints básicos funcionan correctamente
- **Consultas personalizadas (10%):** Búsquedas por categoría, etiqueta, prioridad funcionan
- **Validaciones (10%):** Las validaciones de campos se aplican correctamente

Arquitectura y código (30%)

- **Estructura del proyecto (10%):** Paquetes organizados correctamente por capas
- **Buenas prácticas (10%):** Código limpio, nombres descriptivos, principios SOLID
- **Manejo de excepciones (10%):** Excepciones personalizadas y GlobalExceptionHandler implementados

Base de datos (15%)

- **Configuración correcta (5%):** Conexión a MongoDB funcional
- **Modelo de datos (10%):** Entidad Task bien diseñada con validaciones apropiadas

Documentación (10%)

- **README completo (5%):** Instrucciones claras de instalación y uso
- **Documentación de API (5%):** Endpoints documentados con ejemplos

Pruebas (5%)

- **Testing funcional (5%):** Colección de Postman/Thunder Client con todas las peticiones probadas

Extras (hasta +10% adicional)

- Implementación de DTOs (+2%)
- Paginación y ordenación (+2%)
- Filtros avanzados (+2%)
- Estadísticas (+2%)
- Swagger/OpenAPI (+2%)

Entrega

Formato de entrega:

- Repositorio Git (GitHub, GitLab) con todo el código fuente
- Archivo README.md en la raíz del proyecto
- Colección de Postman exportada en formato JSON
- Documento PDF con capturas de pantalla de las pruebas en Postman

Recursos de apoyo

- Documentación oficial de Spring Boot: https://spring.io/projects/spring-boot
- Documentación de Spring Data MongoDB: https://spring.io/projects/spring-data-mongodb
- MongoDB University: https://university.mongodb.com/
- Postman Learning Center: https://learning.postman.com/

Consejos finales

1. **No dejes todo para el final:** Trabaja de forma incremental siguiendo los pasos
2. **Prueba constantemente:** Después de cada paso, verifica que todo funciona
3. **Haz commits frecuentes:** Usa Git desde el principio y haz commits significativos
4. **Pide ayuda cuando te bloquees:** Es mejor preguntar que perder horas atascado
5. **Lee los mensajes de error:** Aprende a interpretar los stacktraces de Java
6. **Consulta la documentación oficial:** Es tu mejor aliada