

Enunciado del ejercicio: API REST “Agenda de Tareas” (Spring Boot + Spring Data + Tests)

Desarrolla una aplicación **Spring Boot** que implemente una **API REST** para gestionar tareas sencillas de una agenda (título, realizado, fecha de vencimiento).

Requisitos funcionales

La aplicación debe permitir **crear, consultar, actualizar y eliminar** tareas, además de poder **marcar una tarea como realizada o no realizada**.

Modelo de datos

Cada tarea tendrá, como mínimo, los siguientes campos:

- Identificador único autogenerado.
- Título (obligatorio).
- Estado de finalización (por defecto, no realizada).
- Fecha límite (opcional).

Requisitos técnicos

- Implementar la solución con **arquitectura por capas** (controlador, servicio y repositorio).
- Persistencia mediante **Spring Data JPA**.
- Uso de una base de datos para desarrollo (por ejemplo, embebida).
- Validaciones de entrada (al menos, título obligatorio) y gestión adecuada de errores.

Pruebas

- Incluir **pruebas unitarias**:

1) Pruebas unitarias del servicio (TaskService)

- **Crear tarea OK**
 - Dado un título válido, se guarda y se devuelve la tarea con id asignado y done=false por defecto.
- **Crear tarea con título vacío**
 - Si title es null, vacío o solo espacios → lanza excepción de validación.

- **Obtener tarea existente**
 - Si el repositorio encuentra el id, el servicio devuelve la tarea.
- **Obtener tarea inexistente**
 - Si el repositorio no encuentra el id → lanza excepción “TaskNotFound”.
- **Actualizar tarea OK**
 - Si existe, actualiza campos (title, dueDate, done) y persiste; devuelve la tarea actualizada.
- **Actualizar tarea inexistente**
 - Si no existe → lanza “TaskNotFound” y **no** llama a save.
- **Marcar tarea como hecha/no hecha**
 - Si existe, cambia done a true/false y guarda.
 - Verificar que solo cambia el estado (si tu lógica lo exige).
- **Eliminar tarea existente**
 - Si existe, se elimina; verificar llamada a deleteById.
- **Eliminar tarea inexistente**
 - Si no existe → lanza “TaskNotFound” o comportamiento definido; verificar que no se elimina.
- **Listar tareas**
 - Devuelve la lista que entrega el repositorio (tamaño y contenido).

2) Pruebas del controlador (Web / API) con **MockMvc**

Enfocadas a HTTP, sin BD real (por ejemplo @WebMvcTest + mock del servicio):

- **POST crea tarea (201)**
 - Enviar JSON válido → responde 201 (o el código acordado) y devuelve cuerpo con campos esperados.
- **POST inválido (400)**

- Sin title o title vacío → 400 y mensaje/estructura de error correcta.
- **GET por id existente (200)**
 - Devuelve la tarea correcta.
- **GET por id inexistente (404)**
 - Respuesta 404 y error coherente.
- **PUT actualiza existente (200)**
 - Con body válido → devuelve tarea actualizada.
- **PUT inexistente (404)**
- **PATCH estado done (200)**
 - Cambia done y devuelve tarea actualizada (o respuesta definida).
- **DELETE existente (204)**
 - No devuelve body.
- **DELETE inexistente (404)**

Docker

- Dockerizar la aplicación para que pueda ejecutarse en un contenedor.

Docker Hub

- Publicar la imagen en **Docker Hub** en un repositorio accesible y proporcionar la referencia para poder descargarla y ejecutarla.