UD 06: COMPONENTES AVANZADOS

SEMANA 17: TESTING

Spring Boot nos proporciona una serie de utilidades y anotaciones para facilitar poder hacer pruebas de todo tipo en nuestras aplicaciones.

En esta práctica, vamos a utilizar las herramientas vistas en la teoría para realizar pruebas en cada una de nuestras capas.

TESTING UNITARIO EN LA CAPA DE SERVICIO

En este apartado, vamos a realizar **pruebas unitarias en la capa de servicio** de nuestra aplicación utilizando **JUnit** y **Mockito**. Estas pruebas permiten validar que el comportamiento de la lógica de negocio sea el esperado de forma rápida y aislada, sin requerir la conexión a una base de datos real u otros servicios externos.

Añade al proyecto la dependencia **spring-boot-starter-test**, que incluye herramientas como Spring Test, JUnit 5 o Mockito.

En la capa de servicio, a menudo tenemos lógica de negocio que depende de repositorios (para acceder a la base de datos) u otras capas. Para aislar el código de tu servicio y no requerir una base de datos real, utilizamos mocks de estas dependencias, con los que programamos su comportamiento.

Para escribir una prueba unitaria de la capa de servicio, tienes que seguir estos pasos:

1. Crear una clase de prueba con la anotación **@ExtendWith(MockitoExtension.class)**, que le indica a JUnit que use la extensión de Mockito.
2. Anotar la instancia real del servicio a probar con **@InjectMocks** e inyectar las dependencias simuladas que se hayan definido con **@Mock**.
3. Preparar los datos (entidades, DTOs) a utilizar en los tests, bajo un método anotado con **@BeforeEach**.
4. Escribir los métodos de prueba con la anotación **@Test**, que le dice a JUnit que son casos de prueba que se deben ejecutar.
5. Dentro de cada método de prueba, usar el método **when** de Mockito para especificar el comportamiento de los objetos simulados, es decir, qué deben devolver cuando se les llama con ciertos argumentos.

6. Dentro de cada método de prueba, usar los métodos **assert** para verificar que el resultado de la clase de servicio es el correcto, comparándolo con el valor esperado.

7. Además, usar el método **verify** para comprobar que los objetos simulados se han usado de la forma esperada, es decir, qué métodos se han llamado y con qué argumentos.

Estructuraremos las pruebas promoviendo el patrón **Given-When-Then**, que se define como:

- **Given (Escenario)**: Se especifica el escenario, las precondiciones.
- **When (Acción)**: Llama al método a probar.
- **Then (Resultado)**: Verifica que el resultado sea el esperado.

➢ Con esto, crea un test que valide la expulsión de Harry Potter de la escuela. Asegúrate de que el método delete se llama solo 1 vez.

TESTING INTEGRACIÓN EN EL REPOSITORIO

La anotación **@DataJpaTest** es una herramienta esencial en Spring Boot para realizar pruebas enfocadas en la capa de persistencia. Esta anotación configura un entorno de prueba que incluye el soporte de JPA, los repositorios de Spring Data JPA y una base de datos en memoria, generalmente H2, para facilitar el testing aislado de los repositorios.

➢ Usando una base de datos en memoria o TestContainers, crea un test que compruebe que, tras el borrado de un estudiante con mascota, la mascota también ha desaparecido, validando el borrado en cascada.

TESTING DE API

En las aplicaciones Spring Boot, es fundamental garantizar que los controladores REST funcionen correctamente. Para ello, se realizan tests de integración parcial que prueban el controlador en sí, aislándolo del resto de la aplicación. Una herramienta clave para este propósito es la anotación **@WebMvcTest**, que configura un entorno de prueba enfocado en la capa Web.

Al utilizar **@WebMvcTest**, se cargan únicamente los componentes relacionados con el contexto web, como los controladores, los conversores y el mapeo de endpoints, pero se excluyen los otros componentes, como los servicios. Esto permite probar el controlador de manera aislada y controlar sus dependencias mediante mocks.

Además, se utiliza la clase **MockMvc** para realizar solicitudes HTTP simuladas al controlador y verificar las respuestas. MockMvc permite enviar peticiones como GET, POST, PUT o DELETE, y comprobar aspectos como el código de estado HTTP, el contenido de la respuesta y

los encabezados.

Otro aspecto relevante es la verificación del JSON de respuesta. Con **jsonPath**, se pueden examinar campos específicos del JSON y validar que contengan los valores esperados. Esto es esencial para asegurarse de que el controlador serializa correctamente los objetos y devuelve la información adecuada.

- Crea un método que intente crear un Estudiante con anyo_curso igual a 10 y valide el código de retorno adecuado.
- Crea otro método que intente eliminar una Asignatura que tiene alumnos, retornando el código adecuado.