Proyecto CRUD REST MongoDB

Mockito en cada capa

Academia Java

Entregable

Natalia Esquivel Ocadiz

26 de septiembre 2025

Introducción:
En la presente entrega se continuó con la implementación de pruebas unitarias y de integración en el proyecto CRUD con MongoDB. Se aplicó Mockito para el testeo del RestController y se usó JUnit simple para validar la capa de modelo del Producto. Con estas pruebas se buscó asegurar y aumentar la calidad del código y asegurar el correcto funcionamiento de los distintos componentes de la aplicación.

Objetivos:

- Probar la capa de modelo de Producto con JUnit simple para validar constructores, getters y setters.
- Implementar pruebas con Mockito en el RestController para verificar el correcto funcionamiento y las respuestas de la API.
- Complementar las pruebas de la capa de servicio ya existentes, manteniendo el aislamiento mediante mocks.
- Verificar las veces que se llama a un método del Mock
- Probar que las excepciones implementadas sean lanzadas correctamente

Implementación:

En un proyecto Maven no es necesario añadir la dependencia Mockito, ya que está soportada por JUnit y se agrega en la librería por default.
Los tests usados tienen la siguiente estructura:

1. A excepción de la capa Modelo donde se usó JUnit solamente, en el caso de Producto se aplicaron pruebas directas sobre la clase Producto.

- Se verificó que los constructores inicialicen correctamente los atributos:

```
assertEquals(5, producto.getCantidad());
```

- Se probaron los getters y setters, asegurando que los valores se asignaran y recuperaran de forma correcta.

Estas pruebas garantizan que el modelo esté bien definido y libre de errores básicos que puedan propagarse a otras capas.

2. RestController

Se utilizó Mockito para simular la capa de servicio al invocar los endpoints del controlador.

- Anotaciones empleadas:
- @Mock: para mockear el ProductoService.
- @InjectMocks: para inyectar el mock dentro del controlador a probar.
- @BeforeEach: para inicializar el objeto Producto de prueba en cada test.
- Estructura de las pruebas:
- Set up: se definieron las respuestas esperadas con when(...).thenReturn(...).
- Execute: se llamó al endpoint correspondiente.
- Assert: se verificó el contenido del ResponseEntity.
- Verify: se comprobó que el servicio fuera invocado el número de veces esperado.

Ejemplo:

```
when(productoService.save(producto)).thenReturn(producto);

ResponseEntity<Producto> response = productoRestController.crearProducto(producto);

assertEquals(HttpStatus.OK, response.getStatusCode());

assertEquals("Laptop", response.getBody().getNombreProducto());

verify(productoService, times(1)).save(producto);
```

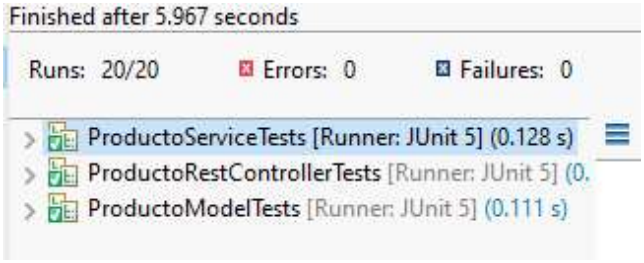
- Casos probados:
  - Creación de un producto vía POST.
  - Obtención de productos existentes.
  - Eliminación de un producto no válido
  - Comportamiento ante entradas no válidas o inexistentes.

3.Service

Se conservaron los métodos de prueba pasada, probando la capa de Servicio con el mock inyectado del repositorio (ProductoRepository)

Se probaron los métodos save, get, update y delete, validando tanto escenarios exitosos como de error.

Resultados:
Se realizaron 20 tests en total, y todos resultaron exitosos en lo esperado



Métricas de cobertura: Se logró un Coverage del 96.9%, 16.9% más del mínimo requerido el cual es bastante bueno.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
productos2	96.9 %	1,190	38	1,228
src/main/java	87.3 %	214	31	245
com.mongodb.crud.productos	37.5 %	3	5	8
com.mongodb.crud.productos.service	100.0 %	69	0	69
com.mongodb.crud.productos.model	100.0 %	81	0	81
com.mongodb.crud.productos.rest	70.1 %	61	26	87
src/test/java	99.3 %	976	7	983
com.mongodb.crud.test	99.3 %	976	7	983
ProductoModelTests.java	100.0 %	175	0	175
ProductoRestControllerTests.java	100.0 %	400	0	400
ProductoServiceTests.java	98.3 %	401	7	408

Reflexiones:

El uso combinado de JUnit y Mockito permitió cubrir distintas capas de la aplicación:

A nivel de modelo, se aseguraron las bases de la entidad Producto

A nivel de controlador, se simuló la comunicación con el servicio sin necesidad de levantar el servidor

A nivel de servicio, se aisló la lógica de negocio del repositorio real.

Esta combinación permitió detectar posibles fallos de integración de forma temprana y reforzó el entendimiento de cómo se deben estructurar pruebas en cada capa. Como una mejora futura, se realizarían pruebas de integración que verifiquen el flujo completo de la aplicación.