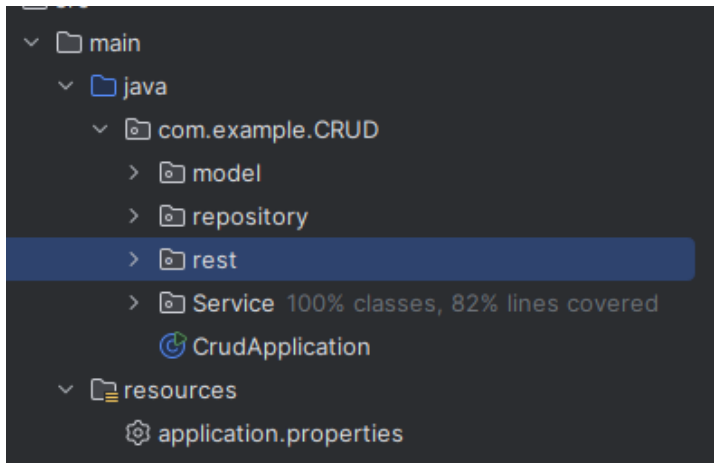


## Documentación del Proyecto CRUD con Spring Boot, MongoDB y Mockito

Este proyecto consiste en la creación de un sistema CRUD (Crear, Leer, Actualizar, Eliminar) para gestionar Spartans (inspirados en el universo de Halo), utilizando Spring Boot, MongoDB como base de datos y pruebas unitarias con JUnit y Mockito.

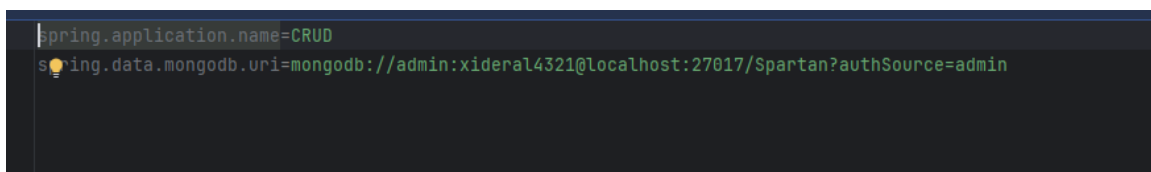
### 1. Arquitectura del Sistema

El sistema se compone de las siguientes capas principales:







### 2. Configuración del Proyecto

La conexión a MongoDB se realiza mediante el archivo `application.properties`, donde se especifica la URI de conexión:



Y dentro de Docker tenemos el contenedor corriendo

<input type="checkbox"/>		mongodb-container	8ffe4d68db94	<a href="#">mongo:8</a>	<a href="#">27017:27017</a> 
<input type="checkbox"/>		mysql-container	7fdb56747882	<a href="#">mysql:latest</a>	<a href="#">3306:3306</a> 

### 3. Endpoints del CRUD

El controlador REST expone los siguientes endpoints:

- **POST /api/spartans** → Crear un Spartan.
- **GET /api/spartans** → Listar todos los Spartans.
- **GET /api/spartans/{id}** → Obtener un Spartan por ID.
- **PUT /api/spartans/{id}** → Actualizar un Spartan.
- **DELETE /api/spartans/{id}** → Eliminar un Spartan.

```
@PostMapping no usages
public ResponseEntity<Spartan> createSpartan(@RequestBody Spartan spartan) {
    Spartan savedSpartan = spartanService.save(spartan);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedSpartan);
}

@GetMapping no usages
public ResponseEntity<List<Spartan>> getAllSpartans() {
    List<Spartan> spartans = spartanService.findAll();
    return ResponseEntity.ok(spartans);
}

@GetMapping("/{id}") no usages
public ResponseEntity<Spartan> getSpartanById(@PathVariable String id) {
    Optional<Spartan> spartan = spartanService.findById(id);
    return spartan.map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
}

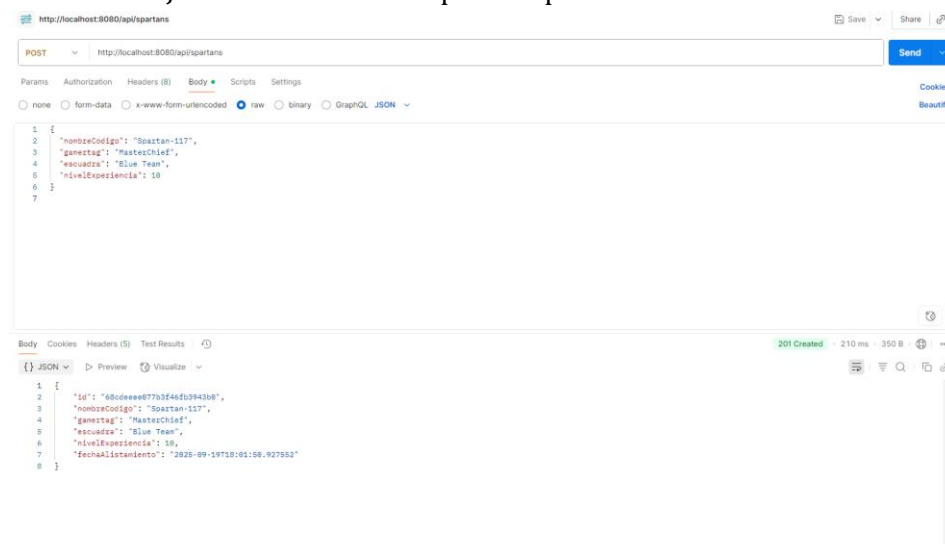
@PutMapping("/{id}") no usages
public ResponseEntity<Spartan> updateSpartan(@PathVariable String id, @RequestBody Spartan spartan) {
    try {
        Spartan updatedSpartan = spartanService.update(id, spartan);
        return ResponseEntity.ok(updatedSpartan);
    } catch (RuntimeException e) {

```

### 4. Pruebas de Caja Negra

Se realizaron pruebas de caja negra utilizando Postman para verificar que los endpoints funcionen correctamente. Ejemplo:

1. Abrir Postman.
2. Probar el servicio levantado en el puerto 8080.
3. Enviar un JSON válido en el cuerpo de la petición POST:



Con esto validamos que el sistema recibe y responde correctamente a entradas válidas.

## 5. Pruebas Unitarias con Mockito

Se implementaron pruebas unitarias en la capa de servicio utilizando **JUnit 5** y **Mockito**. En estas pruebas se 'mockea' el repositorio para simular la interacción con MongoDB, validando la lógica de negocio.

Ejemplo de test implementado:

- Verificar que `save` asigne valores por defecto.

```
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class SpartanServiceImplTest {

    @Mock 10 usages
    private SpartanRepository spartanRepository;

    @InjectMocks 5 usages
    private SpartanServiceImpl spartanService;

    private Spartan spartan; 13 usages

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        spartan = new Spartan();
        spartan.setId("1");
        spartan.setNombreCodigo("Spartan-117");
        spartan.setGanertag("MasterChief");
        spartan.setEscuadra("Blue Team");
        spartan.setNivelExperiencia(10);
        spartan.setFechaAlistamiento(LocalDate.now());
    }
}
```

- Probar `findAll` devolviendo una lista mockeada.

```
@Test
void testFindAll() {
    when(spartanRepository.findAll()).thenReturn(Arrays.asList(spartan));
    List<Spartan> spartans = spartanService.findAll();
    assertEquals(expected: 1, spartans.size());
    verify(spartanRepository, times(wantedNumberOfInvocations: 1)).findAll();
}
```

- Validar que `update` modifique un Spartan existente.

```

@Test
void testUpdateSpartan() {
    when(spartanRepository.findById("1")).thenReturn(Optional.of(spartan));
    when(spartanRepository.save(any(Spartan.class))).thenReturn(spartan);

    Spartan updated = new Spartan();
    updated.setNombreCodigo("Spartan-999");
    updated.setGamertag("NewChief");
    updated.setEscuadra("Red Team");
    updated.setNivelExperiencia(20);

    Spartan result = spartanService.update(id: "1", updated);

    assertEquals(expected: "NewChief", result.getGamertag());
    assertEquals(expected: 20, result.getNivelExperiencia());
}

```

- Simular eliminación con `deleteById`.

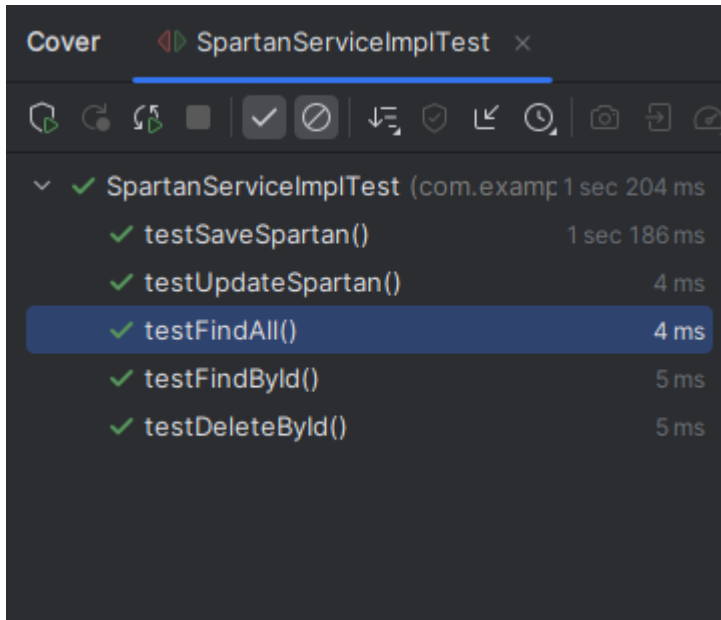
```

@Test
void testDeleteById() {
    doNothing().when(spartanRepository).deleteById("1");
    spartanService.deleteById("1");
    verify(spartanRepository, times(wantedNumberOfInvocations: 1)).deleteById("1");
}
}

```

## 6. Coverage de Pruebas

Tras ejecutar las pruebas, se obtuvo un coverage del **82% en líneas de código** en la clase `SpartanServiceImpl`, superando el mínimo requerido del 80%.



Esto asegura que la mayoría de los caminos lógicos fueron cubiertos durante la ejecución de los tests.

The screenshot shows the 'Coverage' window in the IntelliJ IDE. The title bar indicates the active test is 'SpartanServiceImplTest'. Below the title bar is a toolbar with icons for various actions. The main area displays a table with coverage statistics for the test class and its methods.

Element ^	Class, %	Method, %	Line, %	Branch, %
com.example.CRUD.Service	100% (1/1)	100% (5/5)	82% (14/17)	50% (3/6)
SpartanService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
SpartanServiceImpl	100% (1/1)	100% (5/5)	82% (14/17)	50% (3/6)