



CASO DE ESTUDIO N°2: EDUTECH INNOVATORS SPA

Maximiliano Hernández,
Renato Valenzuela, Diego
Zamora

Justificación de Herramientas y Framework utilizadas

JUnit5: Marco de pruebas unitarias estándar en Java.

- Justificación: Amplia comunidad, integración con Spring Boot, fácil de usar.
- Contribución: Asegura la ejecución adecuada de unidades individuales del código.

Mockito:

- Justificación: Permite la simulación de comportamientos en objetos (mocks).
- Contribución: Facilita pruebas unitarias aisladas al simular dependencias externas.

Postman:

- Justificación: Ideal para probar endpoints RESTful.
- Contribución: Verificación rápida y visual de la comunicación cliente-servidor.

Swagger (OpenAPI):

- Justificación: Estandariza y documenta APIs RESTful.
- Contribución: Permite a los desarrolladores y testers comprender e interactuar con los microservicios fácilmente.

MySQL Workbench:

- Justificación: Diseño y administración de bases de datos.
- Contribución: Facilitó la definición y prueba de esquemas de datos.

Spring boot:

- Justificación: Framework compatible con el lenguaje de programación utilizado (Java)
- Contribución: nos facilitó la creación rápida y sencilla de aplicación web y de backend, especialmente APIs REST.

Pruebas unitarias

Para hacer las pruebas unitarias se usó el framework Mockito el cual nos sirve para simular objetos (como Usuarios, Incidencias, Cursos, etc) llamados “mocks” y ver si el CRUD funciona correctamente sin la necesidad de hacer conexiones reales (a la base de datos en este caso) y para ejecutar estos testeos se ocupa como motor de pruebas el framework de JUnit.

```
Proyecto-Semestral > springboot-backend > src > test > java > com > proyecto > springboot > backend > springboot_backend > services > IncidenciaServiceImplTest.java > ...
1 package com.proyecto.springboot.backend.springboot_backend.services;
2
3 import static org.junit.jupiter.api.Assertions.*;
4 import static org.mockito.Mockito.times;
5 import static org.mockito.Mockito.verify;
6 import static org.mockito.Mockito.when;
7
8 import java.util.ArrayList;
9 import java.util.List;
10
11 import org.junit.jupiter.api.BeforeEach;
12 import org.junit.jupiter.api.Test;
13 import org.mockito.InjectMocks;
14 import org.mockito.Mock;
15 import org.mockito.MockitoAnnotations;
16
17 import com.proyecto.springboot.backend.springboot_backend.entities.Incidencia;
18 import com.proyecto.springboot.backend.springboot_backend.repository.IncidenciaRepository;
19
20
21 public class IncidenciaServiceImplTest {
22
23     @InjectMocks
24     private IncidenciaServiceImpl service;
25
26     @Mock
27     private IncidenciaRepository repository;
28
29     List<Incidencia> list = new ArrayList<Incidencia>();
30
31     @BeforeEach
32     public void init(){
33         MockitoAnnotations.openMocks(this);
34         this.chargeIncidencia();
35     }
36
37
38     @Test
39     public void findByAllTest(){
40         when(repository.findAll()).thenReturn(list);
41
42         List<Incidencia> response = service.findByAll();
43
44         assertEquals(expected:4, response.size());
45         verify(repository, times(wantedNumberOfInvocations:1)).findAll();
46     }
47
48     public void chargeIncidencia(){
49         Incidencia inc1 = new Incidencia(Long.valueOf(111), descripcion:"Primera prueba de Incidencias", estado:"Inactivo", prioridad:"Alta" );
50         Incidencia inc2 = new Incidencia(Long.valueOf(112), descripcion:"Segunda prueba de Incidencias", estado:"Inactivo", prioridad:"Baja" );
51         Incidencia inc3 = new Incidencia(Long.valueOf(113), descripcion:"Probando errores", estado:"Activo", prioridad:"Media" );
52         Incidencia inc4 = new Incidencia(Long.valueOf(114), descripcion:"Probando soluciones", estado:"Inactivo", prioridad:"Alta" );
53
54         list.add(inc1);
55         list.add(inc2);
56         list.add(inc3);
57         list.add(inc4);
58     }
59 }
60
```

¿Cómo nos aseguramos de la calidad del código desarrollado?

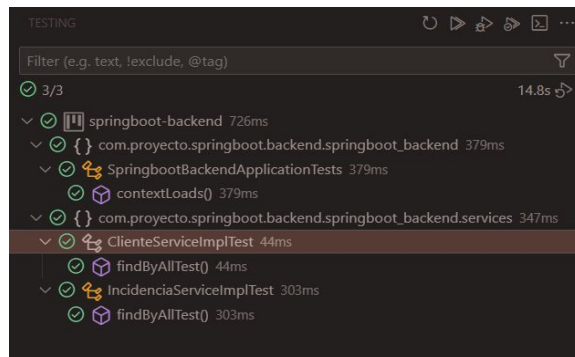
Para asegurarnos de la calidad del código en microservicios con pruebas unitarias usamos JUnit 5 e implementamos Mockito y MockMvc.

JUnit 5: es un framework de pruebas unitarias que nos ayudó a escribir y ejecutar pruebas automáticas para verificar que nuestro código funciona correctamente.

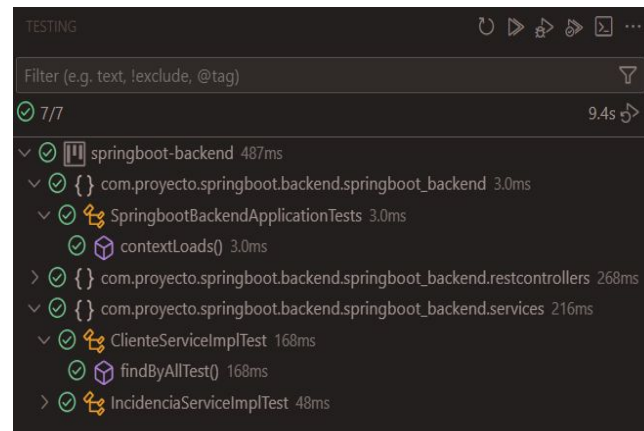
MockMvc: verifica nuestras rutas, respuestas y errores HTTP.

Mockito: simula las dependencias

Prueba de integración



Pruebas unitarias



Prueba de integración

Para hacer las pruebas de integración se usó una clase utilitaria que viene con Spring-Test llamada MockMvc el cual nos sirve para probar nuestra API Rest, osea, hace simulaciones de peticiones HTTP (Get, Post, Put, etc) sin la necesidad de levantar un servidor real.

Estas pruebas permiten verificar que los controladores, servicios y otros componentes funcionen correctamente de forma integrada. Dependiendo de la configuración, las peticiones pueden interactuar con una base de datos real o en memoria. Para ejecutar estos tests también se ocupa como motor de pruebas el framework de JUnit.

```
IncidenciaRestControllerTest.java u x Incidencia.java
Proyecto-Semestral > springboot-backend > src > test > java > com > proyecto > springboot > backend > springboot_backend > restcontrollers > IncidenciaRestControllerTest.java > IncidenciaRestControllerTest > crearIncidenciaTest()

20 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
21 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
22 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
23
24 import static org.mockito.ArgumentMatchers.any;
25 import org.springframework.http.MediaType;
26
27 @SpringBootTest
28 @AutoConfigureMockMvc
29 public class IncidenciaRestControllerTest {
30
31     @Autowired
32     private MockMvc mockMvc;
33
34     @Autowired
35     private ObjectMapper objectMapper;
36
37     @MockitoBean
38     private IncidenciaServiceImpl incidenciaServiceImpl;
39     private List<Incidencia> incidenciaLista;
40
41     @Test
42     public void verIncidenciaTest() throws Exception{
43         when(incidenciaServiceImpl.findAll()).thenReturn(incidenciaLista);
44         mockMvc.perform(get(uriTemplate:"/api/incidencia")
45             .contentType(MediaType.APPLICATION_JSON))
46             .andExpect(status().isOk());
47     }
48
49     @Test
50     public void verUnaIncidenciaTest(){
51         Incidencia unaIncidencia = new Incidencia(id:1L, descripcion:"Incidencia de prueba", estado:"Activo", prioridad:"Alta");
52         try{
53             when(incidenciaServiceImpl.findById(id:1L)).thenReturn(Optional.of(unaIncidencia));
54             mockMvc.perform(get(uriTemplate:"/api/incidencia/{id}"
55                 .contentType(MediaType.APPLICATION_JSON))
56                 .andExpect(status().isOk());
57         }
58         catch(Exception ex){
59             fail("El testing lanzo un error" + ex.getMessage());
60         }
61     }
62
63     @Test
64     public void incidenciaNoExisteTest() throws Exception{
65         when(incidenciaServiceImpl.findById(id:1L)).thenReturn(Optional.empty());
66         mockMvc.perform(get(uriTemplate:"/api/incidencia/{id}"
67             .contentType(MediaType.APPLICATION_JSON))
68             .andExpect(status().isNotFound());
69     }
70
71     @Test
72     public void crearIncidenciaTest() throws Exception{
73         Incidencia unaIncidencia = new Incidencia(id:null, descripcion:"Curso de matematicas con error en ver lista de alumnos", estado:"Terminado", prioridad:"Medio");
74         Incidencia otraIncidencia = new Incidencia(id:4L, descripcion:"Error al ver los videos subidos a las clases de ingles", estado:"En revision", prioridad:"Medio");
75         when(incidenciaServiceImpl.save(any(type:Incidencia.class))) thenReturn(otraIncidencia);
76         mockMvc.perform(post(uriTemplate:"/api/incidencia")
77             .contentType(MediaType.APPLICATION_JSON)
78             .content(objectMapper.writeValueAsString(unaIncidencia)))
79             .andExpect(status().isCreated());
80     }
81 }
82
```

Componentes de microservicio con OAS

-Edición del pom

-Se realizó la importación del swagger

-Se agregó:

“Operation”:describe un endpoint en términos de su propósito, parámetros de entrada y respuestas

“ApiResponse”:documentar las posibles respuestas que un endpoint puede devolver

“Tag”: Categorías para agrupar endpoints

```
@Tag(name = "Cursos", description = "Operaciones relacionadas con Cursos")
@RestController
@RequestMapping("api/cursos")
public class CursoController {

    @Autowired
    private CursoService service;

    @Operation(summary = "Obtener lista de cursos", description = "Devuelve todos los cursos disponibles")
    @ApiResponse(responseCode = "200", description = "Lista de cursos retornada correctamente",
        content = @Content(mediaType = "application/json",
            schema = @Schema(implementation = Curso.class)))
    @GetMapping
    public List<Curso> list(){
        return service.findAll();
    }

    @Operation(summary = "Obtener curso por ID", description = "Obtiene el detalle de un curso específico")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Curso encontrado",
            content = @Content(mediaType = "application/json", schema = @Schema(implementation = Curso.class))),
        @ApiResponse(responseCode = "404", description = "Curso no encontrado")
    })
    @GetMapping("/{id}")
    public ResponseEntity<?> verCurso(@PathVariable Long id){
        Optional<Curso> cursoOptional = service.findById(id);
        if (cursoOptional.isPresent()){
            return ResponseEntity.ok(cursoOptional.orElseThrow());
        }
        return ResponseEntity.notFound().build();
    }

    @Operation(summary = "Crear un nuevo curso", description = "Crea un curso con los datos proporcionados")
    @ApiResponse(responseCode = "201", description = "Curso creado correctamente",
        content = @Content(mediaType = "application/json", schema = @Schema(implementation = Curso.class)))
    @PostMapping
    public ResponseEntity<Curso> crear(@RequestBody Curso unCurso){
        return ResponseEntity.status(HttpStatus.CREATED).body(service.save(unCurso));
    }

    @Operation(summary = "Actualizar curso", description = "Actualiza la información de un curso existente")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Curso actualizado correctamente",
            content = @Content(mediaType = "application/json", schema = @Schema(implementation = Curso.class))),
        @ApiResponse(responseCode = "404", description = "Curso no encontrado")
    })
    @PutMapping
    public ResponseEntity<?> modificar(@PathVariable Long id, @RequestBody Curso unCurso){
        Optional<Curso> cursoOptional = service.findById(id);
        if (cursoOptional.isPresent()){
            Curso cursoExistente = cursoOptional.get();
            cursoExistente.setTitulo(unCurso.getTitulo());
            cursoExistente.setDescripcion(unCurso.getDescripcion());
            cursoExistente.setPublicado(unCurso.getPublicado());
            Curso cursoModificado = service.save(cursoExistente);
            return ResponseEntity.ok(cursoModificado);
        }
    }
}
```

Visor del Swagger

(Imagen de referencia, servicio más detallado)

Cursos Operaciones relacionadas con Cursos

GET /api/cursos Obtener lista de cursos

PUT /api/cursos Actualizar curso

POST /api/cursos Crear un nuevo curso

GET /api/cursos/{id} Obtener curso por ID

DELETE /api/cursos/{id} Eliminar curso

Cursos Operaciones relacionadas con Cursos

GET /api/cursos Obtener lista de cursos

PUT /api/cursos Actualizar curso

POST /api/cursos Crear un nuevo curso

GET /api/cursos/{id} Obtener curso por ID

DELETE /api/cursos/{id} Eliminar curso

Clientes Operaciones relacionadas con Clientes

GET /api/cliente Obtener lista de clientes

PUT /api/cliente Actualizar cliente

POST /api/cliente Crear un nuevo cliente

GET /api/cliente/{id} Obtener cliente por ID

DELETE /api/cliente/{id} Eliminar cliente

Usuarios Funcionalidades para el flujo de usuarios

GET /api/usuarios/{rut} Obtener usuario por rut

PUT /api/usuarios/{rut} Actualizar usuario

DELETE /api/usuarios/{rut} Eliminar usuario

GET /api/usuarios Obtener lista de usuarios

POST /api/usuarios Crear un nuevo usuario

Incidencias Operaciones relacionadas con Incidencias

GET /api/incidencia Obtener lista de incidencias

PUT /api/incidencia Actualizar incidencia

POST /api/incidencia Crear una nueva incidencia

GET /api/incidencia/{id} Obtener incidencia por ID

DELETE /api/incidencia/{id} Eliminar incidencia

Contenido Operaciones relacionadas con Contenido

GET /api/contenido Obtener lista de contenido

PUT /api/contenido Actualizar contenido

POST /api/contenido Crear un nuevo contenido

GET /api/contenido/{id} Obtener contenido por ID

DELETE /api/contenido/{id} Eliminar contenido