
## Departamento de Ciencias de la Computación

## Ingeniería de Software

# Aplicaciones Distribuidas

**Tarea Grupal**
Microservicios y Angular 17 - Plan de Pruebas

**NRC**: 14930

**Autores**:

Díaz Bautista Adriana Lizbeth
Masacela Atupaña Brandon Raúl
Pila Torres Johanna Jicela
Pilataxi Chisaguano Daniela del Pilar

*"La vida debe ser comprendida hacia atrás, pero debe ser vivida hacia delante."*

Søren Kierkegaard

Sangolquí, domingo 20 de julio de 2024

# ÍNDICE



# ÍNDICE DE FIGURAS

# 1. Objetivos de Plan de Pruebas

## *1.1. Objetivo General*

Garantizar la calidad y funcionalidad del proyecto Angular mediante la ejecución de pruebas end-to-end con Cypress, validando la integración correcta de los servicios UsuarioService, CursoService, y CursoUsuarioService, y asegurando que la tabla de usuarios se muestre correctamente en el archivo HTML utilizando Angular Material.

## *1.2. Objetivos Específicos*

- Los servicios deben devolver datos correctos y completos que se reflejen adecuadamente en la interfaz sin errores de integración..
- La tabla debe mostrar los datos de manera clara y precisa, con todas las funcionalidades de Angular Material funcionando como se espera.
- Asegurar que los datos presentados en la interfaz coinciden con los datos enviados y recibidos por los servicios, verificando la precisión en el flujo completo desde el backend hasta el frontend.

# 2. Alcance

- Realizar pruebas unitarios y de integración en BackEnd
- Validar componentes y servicio de Angular 17
- Validación de aplicación Angular con pruebas End-to-End

# 3. Estrategia de pruebas

Se realizarán las siguientes pruebas, mismas que se describen a continuación:

- Pruebas End-to-End utilizando Cypress.
- Pruebas de integración para verificar la interacción entre servicios y el frontend.
- Pruebas de validación para asegurar el correcto funcionamiento de la tabla de usuarios.

## *3.1. Pruebas Unitarias y de Integración para los microservicios backend JUnit*

En el desarrollo de microservicios backend, la implementación de pruebas unitarias y de integración es crucial para garantizar la estabilidad y la funcionalidad del sistema.

- Las pruebas unitarias, realizadas a nivel de componentes individuales, permiten verificar el comportamiento esperado de cada unidad de código.
- Las pruebas de integración se centran en la interacción entre los distintos microservicios y otros sistemas, asegurando que los componentes se integren correctamente y cumplan con los requisitos funcionales y de rendimiento.

La combinación efectiva de pruebas unitarias y de integración con JUnit ayuda a mantener un alto nivel de calidad y fiabilidad en las aplicaciones basadas en microservicios. Además se utilizan bibliotecas como Mockito para la simulación y MockMvc para las pruebas de integración.

Para realizar estas pruebas se consideraron cambios en las dependencias de Usuario, Curso y Principal del proyecto

**Principal**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>com.espe.app</groupId>
    <artifactId>app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging>

    <modules>
        <module>msvc-usuarios</module>
        <module>msvc-cursos</module>
    </modules>

    <properties>
        <java.version>17</java.version>
        <junit.jupiter.version>5.9.2</junit.jupiter.version>
        <mockito.version>4.6.1</mockito.version>
        <spring-cloud.version>2023.0.2</spring-cloud.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <!-- Otras dependencias -->

        <!-- JUnit 5 -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.jupiter.version}</version>
            <scope>test</scope>
        </dependency>
```

```
        <!-- Mockito -->
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>${mockito.version}</version>
            <scope>test</scope>
        </dependency>

        <!-- Spring Test -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

**Curso**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>com.espe.app</groupId>
                <artifactId>app</artifactId>
                <version>1.0-SNAPSHOT</version>
        </parent>
        <groupId>com.espe.msvc-cursos</groupId>
        <artifactId>msvc-cursos</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>msvc-cursos</name>
        <description>Demo project for Spring Boot</description>
        <url/>
        <licenses>
                <license/>
        </licenses>
        <developers>
                <developer/>
        </developers>
        <scm>
                <connection/>
                <developerConnection/>
                <tag/>
                <url/>
        </scm>
        <properties>
                <java.version>17</java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
```

```
		<dependency>
			<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>org.postgresql</groupId>
			<artifactId>postgresql</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>

		<!-- JUnit 5 -->
		<dependency>
			<groupId>org.junit.jupiter</groupId>
			<artifactId>junit-jupiter-engine</artifactId>
			<scope>test</scope>
		</dependency>

		<!-- Mockito -->
		<dependency>
			<groupId>org.mockito</groupId>
			<artifactId>mockito-core</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>
</project>
```

**Usuario**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
		 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
		 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<parent>
		<groupId>com.espe.app</groupId>
		<artifactId>app</artifactId>
		<version>1.0-SNAPSHOT</version>
	</parent>

	<groupId>com.espe.msvc.usuarios</groupId>
	<artifactId>msvc-usuarios</artifactId>
```

```
	<version>0.0.1-SNAPSHOT</version>
	<name>msvc-usuarios</name>
	<description>Demo project for Spring Boot</description>
	<properties>
		<java.version>17</java.version>
		<spring-cloud.version>2023.0.2</spring-cloud.version>
		<junit.jupiter.version>5.9.2</junit.jupiter.version>
		<mockito.version>4.6.1</mockito.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>
		</dependency>
		<dependency>
			<groupId>com.mysql</groupId>
			<artifactId>mysql-connector-j</artifactId>
			<scope>runtime</scope>
		</dependency>

		<!-- Dependencias para pruebas -->
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
		<dependency>
			<groupId>org.junit.jupiter</groupId>
			<artifactId>junit-jupiter-engine</artifactId>
			<version>${junit.jupiter.version}</version>
			<scope>test</scope>
		</dependency>
		<dependency>
			<groupId>org.mockito</groupId>
			<artifactId>mockito-core</artifactId>
			<version>${mockito.version}</version>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<dependencyManagement>
		<dependencies>
			<dependency>
				<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>
				<version>${spring-cloud.version}</version>
				<type>pom</type>
				<scope>import</scope>
			</dependency>
		</dependencies>
	</dependencyManagement>
```

```
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
</project>
```

### 3.1.1. Pruebas Unitarias

En esta prueba unitaria, se evalúan las funcionalidades del controlador `CursoController` mediante el uso de JUnit y Mockito para simular el comportamiento del servicio `CursoService`. Los métodos de prueba verifican que el controlador maneje correctamente las operaciones CRUD (listar, detalle, crear, editar y eliminar) y la asignación de usuarios a cursos. Cada prueba configura el entorno simulando respuestas del servicio, ejecuta la acción correspondiente del controlador y luego valida que la respuesta del controlador sea la esperada.

```
package com.espe.msvc_cursos.controllers;

import com.espe.msvc_cursos.models.Usuario;
import com.espe.msvc_cursos.models.entity.Curso;
import com.espe.msvc_cursos.servicies.CursoService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import java.util.Optional;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyLong;
import static org.mockito.Mockito.*;

class CursoControllerTest {

    @Mock
    private CursoService service;

    @InjectMocks
    private CursoController controller;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void listar() {
        // Arrange
        when(service.listar()).thenReturn(List.of(new Curso()));

        // Act
        var result = controller.listar();
```

```
        // Assert
        assertEquals(1, result.size());
    }

    @Test
    void detalle() {
        // Arrange
        Curso curso = new Curso();
        curso.setId(1L);
        when(service.porId(anyLong())).thenReturn(Optional.of(curso));

        // Act
        ResponseEntity<?> response = controller.detalle(1L);

        // Assert
        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertEquals(curso, response.getBody());
    }

    @Test
    void crear() {
        // Arrange
        Curso curso = new Curso();
        when(service.guardar(any(Curso.class))).thenReturn(curso);

        // Act
        ResponseEntity<?> response = controller.crear(curso);

        // Assert
        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertEquals(curso, response.getBody());
    }

    @Test
    void editar() {
        // Arrange
        Curso curso = new Curso();
        curso.setId(1L);
        when(service.porId(anyLong())).thenReturn(Optional.of(curso));
        when(service.guardar(any(Curso.class))).thenReturn(curso);

        // Act
        ResponseEntity<?> response = controller.editar(curso, 1L);

        // Assert
        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertEquals(curso, response.getBody());
    }

    @Test
    void eliminar() {
        // Arrange
        Curso curso = new Curso();
        curso.setId(1L);
        when(service.porId(anyLong())).thenReturn(Optional.of(curso));

        // Act
        ResponseEntity<?> response = controller.eliminar(1L);

        // Assert
        assertEquals(HttpStatus.NO_CONTENT, response.getStatusCode());
    }

    @Test
    void asignarUsuario() {
        // Arrange
        Usuario usuario = new Usuario();
```

```
        when(service.agregarUsuario(any(Usuario.class),
anyLong())).thenReturn(Optional.of(usuario));

        // Act
        ResponseEntity<?> response = controller.asignarUsuario(1L, usuario);

        // Assert
        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertEquals(usuario, response.getBody());
    }
}
```

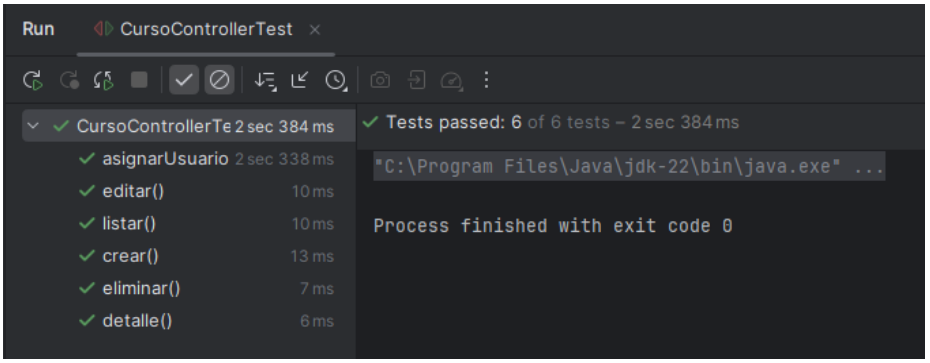


**Fig. 1**. Ejecución de Pruebas Unitarias en Curso

En esta prueba unitaria, se evalúan las operaciones del controlador `UsuarioController` utilizando JUnit y Mockito para simular el comportamiento del servicio `UsuarioService` y el objeto `BindingResult`. Las pruebas cubren funcionalidades CRUD, así como validar errores de entrada. Cada prueba configura el entorno simulado, ejecuta el método del controlador correspondiente y verifica que la respuesta del controlador sea correcta, asegurando que el controlador maneje adecuadamente los distintos escenarios y que las interacciones con el servicio.

```
package com.espe.msvc.usuarios.msvc_usuarios.controllers;

import com.espe.msvc.usuarios.msvc_usuarios.models.entity.Usuario;
import com.espe.msvc.usuarios.msvc_usuarios.services.UsuarioService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Optional;

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

class UsuarioControllerTest {

    @Mock
    private UsuarioService service;
```

```
    @Mock
    private BindingResult result;

    @InjectMocks
    private UsuarioController controller;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void listar() {
        // Arrange
        List<Usuario> usuarios = Arrays.asList(new Usuario(), new Usuario());
        when(service.listar()).thenReturn(usuarios);

        // Act
        List<Usuario> response = controller.listar();

        // Assert
        assertNotNull(response);
        assertEquals(2, response.size());
        verify(service, times(1)).listar();
    }

    @Test
    void detalle() {
        // Arrange
        Usuario usuario = new Usuario();
        when(service.porId(1L)).thenReturn(Optional.of(usuario));

        // Act
        ResponseEntity<?> response = controller.detalle(1L);

        // Assert
        assertNotNull(response);
        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertEquals(usuario, response.getBody());
        verify(service, times(1)).porId(1L);
    }

    @Test
    void crear() {
        // Arrange
        Usuario usuario = new Usuario();
        when(service.guardar(any())).thenReturn(usuario);

        // Act
        ResponseEntity<?> response = controller.crear(usuario, result);

        // Assert
        assertNotNull(response);
        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertEquals(usuario, response.getBody());
        verify(service, times(1)).guardar(any());
    }

    @Test
    void editar() {
        // Arrange
        Usuario usuario = new Usuario();
        when(service.porId(1L)).thenReturn(Optional.of(usuario));
        when(service.guardar(any())).thenReturn(usuario);

        // Act
        ResponseEntity<?> response = controller.editar(usuario, result, 1L);
```

```
        // Assert
        assertNotNull(response);
        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertEquals(usuario, response.getBody());
        verify(service, times(1)).porId(1L);
        verify(service, times(1)).guardar(any());
    }

    @Test
    void eliminar() {
        // Arrange
        Usuario usuario = new Usuario();
        when(service.porId(1L)).thenReturn(Optional.of(usuario));

        // Act
        ResponseEntity<?> response = controller.eliminar(1L);

        // Assert
        assertNotNull(response);
        assertEquals(HttpStatus.NO_CONTENT, response.getStatusCode());
        verify(service, times(1)).eliminar(1L);
    }

    @Test
    void validar() {
        // Arrange
        when(result.hasErrors()).thenReturn(true);

        // Act
        ResponseEntity<Map<String, String>> response =
UsuarioController.validar(result);

        // Assert
        assertNotNull(response);
        assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
    }
}
```

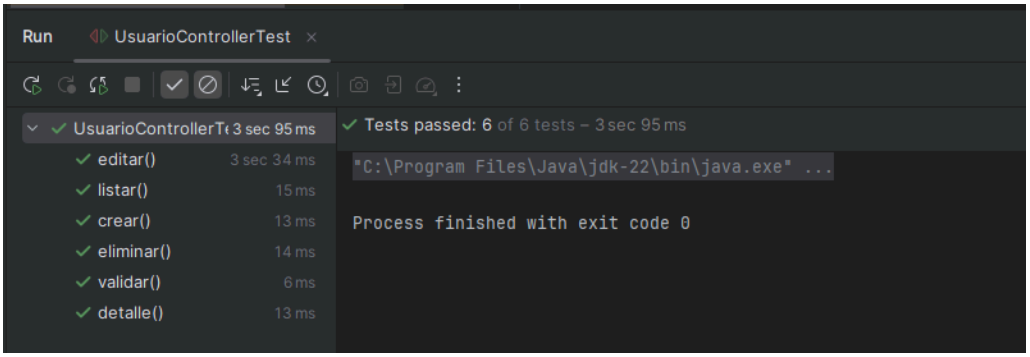


**Fig. 2**. Ejecución de Pruebas Unitarias en Usuario

### 3.1.2. Pruebas de Integración

En primera instancia queremos validar cómo el CursoController interactúa con CursoService y realiza operaciones HTTP correctamente, utilizando el enfoque mocks (@MockBean) para evitar depender de la implementación real del servicio, controlando el comportamiento del mismo y verificar cómo el controlador interactúa con él, en CusoControllerIntegrationTest

```
package com.espe.msvc_cursos.controllers;

import com.espe.msvc_cursos.models.Usuario;
import com.espe.msvc_cursos.models.entity.Curso;
import com.espe.msvc_cursos.servicies.CursoService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import java.util.Optional;

import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyLong;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@WebMvcTest(CursoController.class)
class CursoControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private CursoService service;

    /*@BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }*/

    @Test
    void listar() throws Exception {

mockMvc.perform(get("/curso/MostrarCursos").contentType(MediaType.APPLICATION_
JSON))
                .andExpect(status().isOk());
    }

    @Test
    void detalle() throws Exception {
        Curso curso = new Curso();
        curso.setId(1L);
        Mockito.when(service.porId(anyLong())).thenReturn(Optional.of(curso));

        mockMvc.perform(get("/curso/BuscarCursoPorID/1")
                        .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(1L));
    }

    @Test
    void crear() throws Exception {
        Curso curso = new Curso();
        curso.setId(1L);
        Mockito.when(service.guardar(any(Curso.class))).thenReturn(curso);

        mockMvc.perform(post("/curso/CrearCurso")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("{\"nombre\": \"Nuevo Curso\"}"))
```

```
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.id").value(1L));
    }

    @Test
    void editar() throws Exception {
        Curso curso = new Curso();
        curso.setId(1L);
        Mockito.when(service.porId(anyLong())).thenReturn(Optional.of(curso));
        Mockito.when(service.guardar(any(Curso.class))).thenReturn(curso);

        mockMvc.perform(put("/curso/ModificarCurso/1")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("{\"nombre\": \"Curso Actualizado\"}"))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.id").value(1L));
    }

    @Test
    void eliminar() throws Exception {
        Curso curso = new Curso();
        curso.setId(1L);
        Mockito.when(service.porId(anyLong())).thenReturn(Optional.of(curso));

        mockMvc.perform(delete("/curso/EliminarCurso/1")
                        .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isNoContent());
    }

    @Test
    void asignarUsuario() throws Exception {
        Usuario usuario = new Usuario();
        usuario.setId(1L);
        Mockito.when(service.agregarUsuario(any(Usuario.class),
anyLong())).thenReturn(Optional.of(usuario));

        mockMvc.perform(put("/asignar-usuario/1")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("{\"id\": 1}"))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.id").value(1L));
    }
}
```

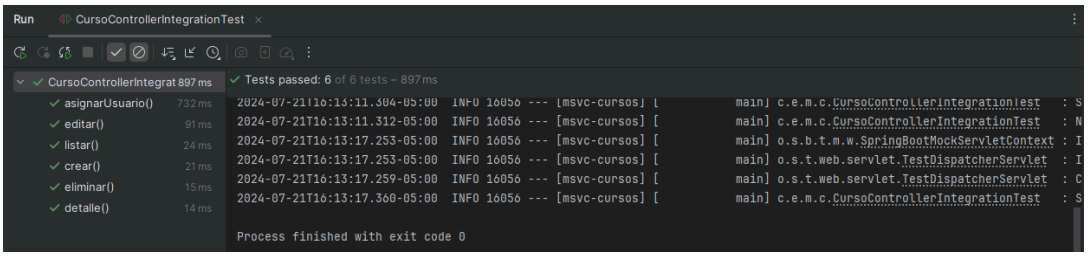


**Fig. 3**. Ejecución de Pruebas de Integración en Curso

Como segundo caso validamos la integración completa con la base de datos y verificamos la funcionalidad completa del sistema, utilizando el enfoque con una base de datos en memoria (@SpringBootTest) en UsuarioControllerIntegrationTest, asegurando que el controlador maneje correctamente las operaciones de CRUD sobre la base de datos.

```
package com.espe.msvc.usuarios.msvc_usuarios.controllers;

import com.espe.msvc.usuarios.msvc_usuarios.models.entity.Usuario;
import com.espe.msvc.usuarios.msvc_usuarios.repositories.UsuarioRepository;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

import static org.hamcrest.Matchers.*;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@SpringBootTest
@AutoConfigureMockMvc
class UsuarioControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private UsuarioRepository repository;

    @BeforeEach
    void setUp() {
        repository.deleteAll();
    }

    @Test
    void listar() throws Exception {
        repository.save(new Usuario());
        repository.save(new Usuario());

        mockMvc.perform(get("/usuario/MostrarUsuarios"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", hasSize(2)));
    }

    @Test
    void detalle() throws Exception {
        Usuario usuario = repository.save(new Usuario());

        mockMvc.perform(get("/usuario/BuscarUsuarioPorID/{id}",
usuario.getId()))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id", is(usuario.getId().intValue())));
    }

    @Test
    void crear() throws Exception {
        Usuario usuario = new Usuario();
        usuario.setNombre("Test User");
        usuario.setEmail("test@example.com");
        usuario.setPassword("password");

        mockMvc.perform(post("/usuario/CrearUsuario")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("{ \"nombre\": \"Test User\", \"email\":
\"test@example.com\", \"password\": \"password\" }"))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.nombre", is("Test User")))
```

```
                .andExpect(jsonPath("$.email", is("test@example.com")));
    }

    @Test
    void editar() throws Exception {
        Usuario usuario = repository.save(new Usuario());
        //usuario.setNombre("Updated User");
        usuario.setNombre("Original User");
        usuario.setEmail("original@example.com");
        usuario.setPassword("originalpassword");
        usuario = repository.save(usuario);

        mockMvc.perform(put("/usuario/ModificarUsuario/{id}", usuario.getId())
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("{ \"nombre\": \"Updated User\", \"email\":
\"updated@example.com\", \"password\": \"newpassword\" }"))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.nombre", is("Updated User")))
                .andExpect(jsonPath("$.email", is("updated@example.com")));
    }

    @Test
    void eliminar() throws Exception {
        Usuario usuario = repository.save(new Usuario());

        mockMvc.perform(delete("/usuario/EliminarUsuario/{id}",
usuario.getId()))
                .andExpect(status().isNoContent());
    }
}
```


**Fig. 4**. Ejecución de Pruebas de Integración en Usuario

## *3.2. Pruebas unitarias para los componentes y servicios de Angular con Jasmine y Karma*

Las pruebas unitarias son fundamentales para garantizar la calidad y fiabilidad del código en aplicaciones Angular. Utilizando Jasmine y Karma, se pueden crear y ejecutar pruebas que verifican el comportamiento individual de componentes y servicios.

- Jasmine es un marco de pruebas para JavaScript que facilita la escritura de tests legibles y expresivos
- Karma es un ejecutor de pruebas que permite correr estos tests en varios navegadores y plataformas de manera automatizada.

### 3.2.1. Prueba de Componentes y Servicio

Para realizar las pruebas de componentes y servicios es necesario considerar archivos de prueba (*.spec.ts) para cada componente y servicio. Estos archivos deben incluir las pruebas unitarias escritas en Jasmine y ejecutarlas con el comando ***ng test.***

**app.components.spec.ts**

```
import { TestBed } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [AppComponent],
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have the 'CursoUsuario' title`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('CursoUsuario');
  });

  it('should render title', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.nativeElement as HTMLElement;
    expect(compiled.querySelector('h1')?.textContent).toContain('Hello,
CursoUsuario');
  });
  it('Mi caso de prueba', () => {
    expect(1).toBe(1);
  });
});
```

**curso.service.spect.ts**

```
/* tslint:disable:no-unused-variable */

import { TestBed, async, inject } from '@angular/core/testing';
import { HttpClientTestingModule } from '@angular/common/http/testing';
import { CursoService } from './curso.service';

describe('Service: Curso', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule], // Agregar HttpClientTestingModule
      providers: [CursoService]
    });
  });

  it('should  be created', () => {
    const service: CursoService = TestBed.inject(CursoService);
    expect(service).toBeTruthy();
  });
});
```

**curso-usuario.service.spect.ts**

```
/* tslint:disable:no-unused-variable */

import { TestBed, async, inject } from '@angular/core/testing';
import { HttpClientTestingModule } from '@angular/common/http/testing';
import { CursoUsuarioService } from './curso-usuario.service';

describe('Service: CursoUsuario', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule], // Agregar HttpClientTestingModule
      providers: [CursoUsuarioService]
    });
  });

  it('should be created', () => {
    const service: CursoUsuarioService = TestBed.inject(CursoUsuarioService);
    expect(service).toBeTruthy();
  });
});
```

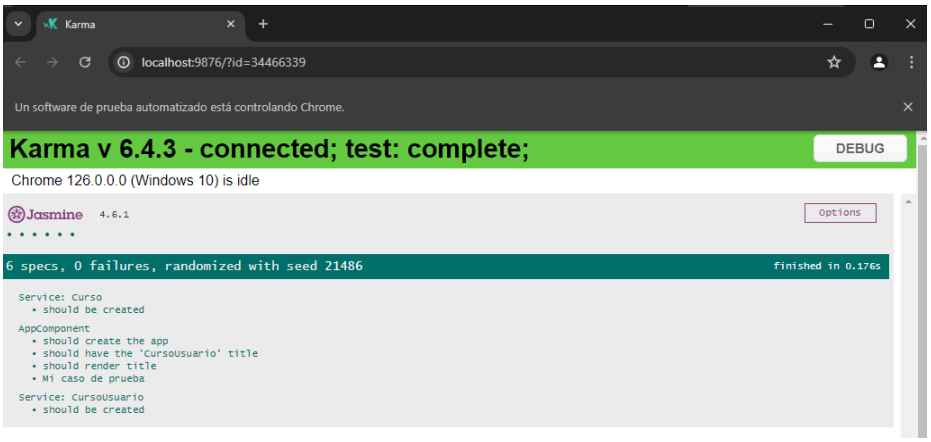


**Fig. 5**. Interfaz de Karma con Jasmine

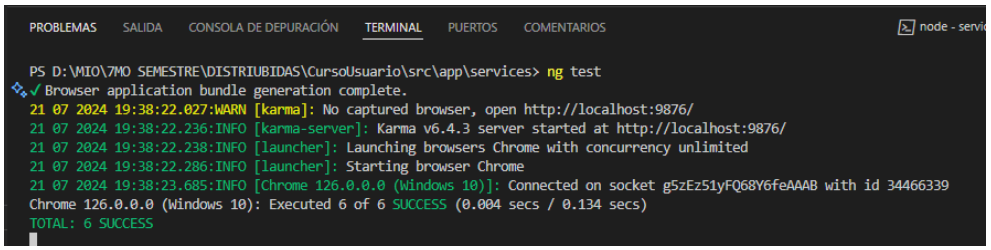


**Fig. 6**. Ejecución de Pruebas de componentes y servicios Angular

## *3.3. Pruebas end-to-end (E2E) para la aplicación Angular*

Las pruebas end-to-end (E2E) simulan escenarios de uso reales, verificando que todas las partes de la aplicación funcione correctamente en conjunto, desde la interfaz de

usuario hasta los servicios backend. Utilizando herramientas como Cypress, las pruebas E2E permiten detectar errores que pueden no ser evidentes en pruebas unitarias o de integración.

Para poder utilizarlo debemos instalar previamente ***cypress,*** una vez instalada la herramienta, nos crea un directorio donde podemos desarrollar las pruebas utilizando el comando ***ng e2e*** en la consola del proyecto

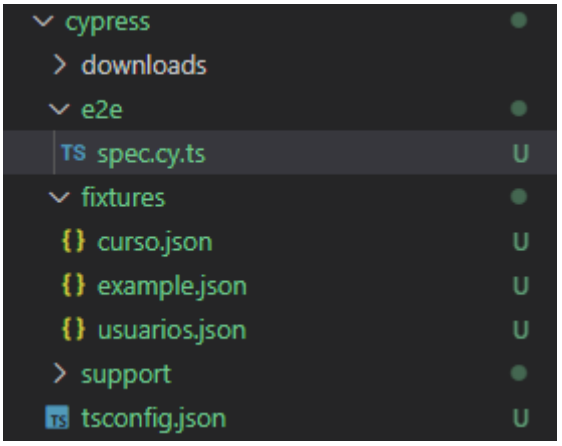


**Fig. 7**. Directorio de Cypress

En los fixtures se encuentran los ***.json*** en los cuales simulamos datos que deberían estar en nuestro proyecto para poder realizar las pruebas correspondientes. Finalmente en el archivo ***spec.cy.ts*** configuraremos la codificación para que realice estas pruebas considerando los archivos ***.json*** antes mencionados.

**spec.cy.ts**

```
describe('My First Test', () => {
  it('Visits the initial project page', () => {
    cy.visit('/')
    cy.contains('Bienvenido al Sistema de Gestión de Cursos')
  })
})

describe('UsuarioService Tests', () => {
  beforeEach(() => {
    cy.intercept('GET', 'http://localhost:8001/usuario/MostrarUsuarios', {
      fixture: 'usuarios.json'
    }).as('getUsuarios');
  });

  it('should display a list of usuarios', () => {
    cy.visit('/');
    cy.wait(5000); // Espera a que se cargue la lista de usuarios
    cy.wait('@getUsuarios'); // Espera a que se cargue la lista de usuarios

    cy.get('.mat-elevation-z8').should('exist');
    cy.get('.mat-elevation-z8 tbody tr').should('have.length', 1); // asume
que tienes 1 curso en el fixture
    cy.get('.mat-elevation-z8 tbody tr td').eq(1).should('contain', 'Usuario
1');
    cy.get('.mat-elevation-z8 tbody tr td').eq(2).should('contain',
```

```
'usuario1@example.com');
  });
});

describe('CursoService Tests', () => {
  beforeEach(() => {
    cy.intercept('GET', 'http://localhost:8002/curso/MostrarCursos', {
      fixture: 'curso.json'
    }).as('getCurso');
  });

  it('should display list of cursos', () => {
    cy.visit('/');
    cy.wait(5000); // Espera a que se cargue la lista de cursos
    cy.wait('@getCurso'); // Espera a que se cargue la lista de cursos
    cy.get('.mat-elevation-z8').should('exist');
    cy.get('.mat-elevation-z8 tbody tr').should('have.length', 1); // asume
que tienes 1 curso en el fixture
    cy.get('.mat-elevation-z8 tbody tr td').eq(1).should('contain', 'Curso de
Prueba');
  });
});
```

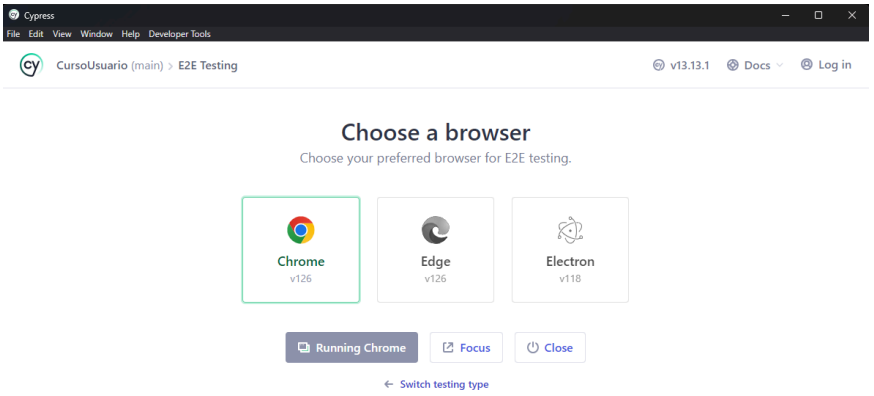


**Fig. 8**. Interfaz de Cypress

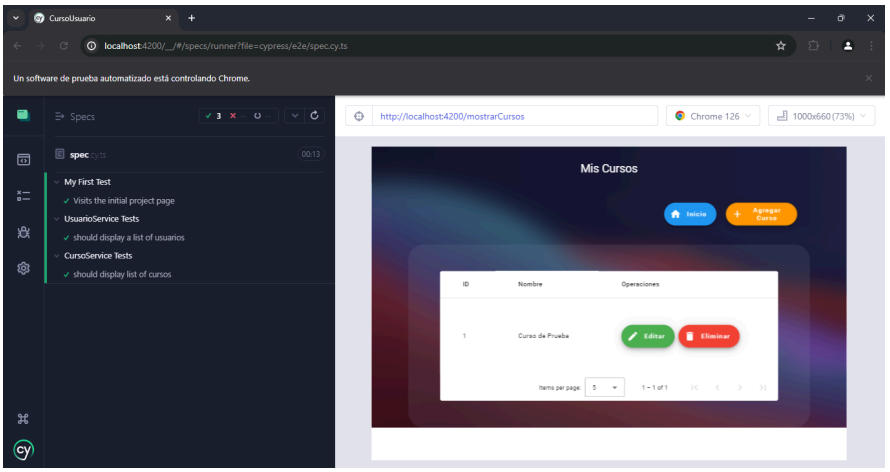


**Fig. 9**. Ejecución de Pruebas de aplicación

## 4. Criterios de Aceptación

- Todas las funcionalidades especificadas deben funcionar como se espera en la interfaz de usuario.
- La interfaz de usuario debe estar alineada con los diseños y especificaciones proporcionadas. Esto incluye el formato, disposición, y estilo de los componentes, como la tabla de usuarios, botones y mensajes de error.
- Los servicios y componentes deben integrarse de manera fluida, sin causar problemas de comunicación o errores entre ellos. La integración de UsuarioService, CursoService, y CursoUsuarioService debe funcionar correctamente sin fallos en la transferencia de datos.
- Las pruebas deben demostrar que la aplicación responde de manera eficiente a las interacciones del usuario.
- La aplicación debe manejar los errores de manera adecuada, mostrando mensajes claros y precisos cuando ocurren problemas.
- Los datos presentados en la interfaz deben ser precisos y reflejar correctamente la información proporcionada por los servicios. Cualquier discrepancia entre los datos esperados y los presentados debe ser identificada y corregida.

## 5. Recursos

- Cypress para pruebas end-to-end.
- Angular CLI para el entorno de desarrollo.
- Herramientas Jasmine y Karma para pruebas de componentes y servicios
- Herramientas de desarrollo de Angular Material para la interfaz de usuario.
- Herramienta JUnit 5 y Mockito para realizar pruebas unitarias y de integración.

## 6. Calendario de pruebas

- Inicio de Pruebas: 20-07-2024
- Pruebas de Integración y Validación de Tabla: 20-07-2024
- Pruebas de Manejo de Errores: 20-07-2024
- Revisión de Resultados y Ajustes: 21-07-2024
- Finalización de Pruebas: 21-07-2024

## 7. Responsabilidades

- Desarrolladores: Implementar y ejecutar las pruebas utilizando Cypress y herramientas anteriormente descritas para escribir pruebas específicas para los servicios y la tabla de usuarios, cursos e inscripciones.
- Equipo de QA: Revisar los resultados de las pruebas, validar que se cumplen los criterios de aceptación, y coordinar ajustes si es necesario.

## 8. Entregables

- Informes detallados sobre la ejecución de las pruebas, con métricas y resultados obtenidos.

## 9. Criterios de Éxito y Fracaso

- **Éxito**: El software se considera listo para su despliegue si todas las pruebas pasan sin errores críticos, la tabla de usuarios, cursos e inscripciones se muestran correctamente y el manejo de errores es adecuado.
- **Fracaso**: El software se considera no apto para el despliegue si se encuentran errores críticos que afectan la funcionalidad, integración o la interfaz de usuario, y no se resuelven antes de la fecha límite.

## 10. Conclusiones de Pruebas

Las pruebas realizadas para el proyecto han demostrado un sólido desempeño y una correcta implementación de las funcionalidades previstas. En el ámbito del backend, se llevaron a cabo pruebas unitarias e integración utilizando JUnit y Mockito, así como pruebas con mocks para validar el comportamiento de los microservicios. Estas pruebas han confirmado que los servicios funcionan de manera óptima, sin errores críticos y con una adecuada integración entre ellos. La cobertura de pruebas ha permitido identificar y resolver problemas potenciales antes de la implementación, asegurando la robustez y la fiabilidad del backend.

Por otro lado, en el entorno del frontend, se realizaron pruebas unitarias y de integración para los componentes y servicios de Angular utilizando Jasmine y Karma. Las pruebas end-to-end (E2E) también se ejecutaron para validar la aplicación Angular en escenarios reales. Los resultados fueron positivos, con todas las funcionalidades verificadas según los requisitos definidos y sin fallos significativos. La combinación de pruebas unitarias, de integración y E2E ha garantizado una experiencia de usuario consistente y libre de errores, corroborando la calidad y el desempeño esperado de la aplicación.