Aplicaciones Distribuidas

Universidad de las Fuerzas Armadas - ESPE

Sistema de Gestión de Estudiantes y Cursos

Plan de Pruebas Template

VERSIÓN: 1 FECHA DE REVISIÓN: 21/07/2024

La aprobación del presente Plan de Pruebas indica el entendimiento del propósito y contenido descrito en el presente documento, sus referencias y anexos. Esto implica que la aprobación del presente documento reconoce la aprobación de todos sus anexos. La firma de este documento, implica la conformidad de cada individuo con el mismo.

Tester 1		
Andrés Jácome	ajjácome2@gmail.com	099 859 8514
Firma		21/07/2024

Tester 2		
Brayan Patiño	bdpatino@espe.edu.ec	099 339 6358
Firma		21/07/2024

Tabla de Contenidos

Sección 1. Visión Global 1

Sección 2. Metodología de pruebas 2

2.1 Elementos de Prueba 2

2.2 Tipo de Pruebas ..... **¡Error! Marcador no definido.**

2.2.1 Pruebas Funcionales **¡Error! Marcador no definido.**

2.2.2 Pruebas de Datos **¡Error! Marcador no definido.**

2.2.3 Pruebas de Usuario **¡Error! Marcador no definido.**

2.2.4 Pruebas No Funcionales **¡Error! Marcador no definido.**

2.3 Fases de Pruebas..... **¡Error! Marcador no definido.**

Sección 3. Calendario de pruebas 4

Sección 4. Monitorización e informes de pruebas 4

4.1 Monitorización ..... **¡Error! Marcador no definido.**

4.2 Informes ..... **¡Error! Marcador no definido.**

Sección 5. Referencias 11

Sección 6. Glosario 12

Sección 7. Historial de Revisión 13

Sección 8. Apéndices 14

Sección 1. Visión Global

1.1 Enfoque de Pruebas del Proyecto

El proyecto consiste en el desarrollo de una aplicación Spring Boot que maneja operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre usuarios a través del controlador UsuarioController. Para asegurar la calidad y funcionalidad del controlador, se implementarán pruebas unitarias utilizando JUnit y Mockito.

1.2 Objetivos de las Pruebas

Los objetivos principales de las pruebas son:

1. **Validar la funcionalidad de los controladores:** Asegurarse de que todas las operaciones CRUD se ejecuten correctamente.
2. **Asegurar el manejo adecuado de errores:** Verificar que el controlador maneje apropiadamente los errores y retorne los códigos de estado HTTP correspondientes.
3. **Garantizar la integridad de los datos:** Confirmar que los datos de los usuarios se manejen correctamente durante las operaciones de creación, edición y eliminación.

1.3 Alcance de las Pruebas

El alcance de las pruebas incluye:

1. **Pruebas de funcionalidad:** Verificación de cada operación CRUD en el controlador UsuarioController.
2. **Pruebas de validación:** Confirmación de que los errores de validación se manejen correctamente.
3. **Pruebas de respuesta:** Validación de los códigos de estado HTTP y los mensajes de error retornados por el controlador.
4. **Pruebas de integración:** Asegurarse de que el controlador interactúe correctamente con el servicio de usuario.

1.4 Enfoque de Pruebas Adoptado

El enfoque de pruebas adoptado es el siguiente:

1. **Pruebas Unitarias:**
   - Utilizar JUnit para crear pruebas unitarias que validen cada operación del controlador UsuarioController.
2. **Pruebas Basadas en Casos de Uso:**
   - Definir casos de prueba basados en los casos de uso del sistema para asegurar que todas las rutas críticas de la aplicación sean verificadas.
3. **Revisión Continua:**
   - Revisar y actualizar los casos de prueba y los scripts de prueba continuamente para reflejar cualquier cambio en los requisitos del sistema o en la implementación del controlador.

Sección 2. Metodología de pruebas

2.1 Elementos de Prueba

En este proyecto, los elementos de prueba se centran en las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) realizadas por los controladores UsuarioController y CursoController de la aplicación Spring Boot. A continuación, se describen los elementos específicos de prueba para cada controlador:

UsuarioController

1. **Controlador UsuarioController:**
   - **Métodos:**
     - listar(): Lista todos los usuarios.
     - detalle(Long id): Obtiene los detalles de un usuario específico por su ID.
     - crear(Usuario usuario, BindingResult bindingResult): Crea un nuevo usuario.
     - editar(Usuario usuario, BindingResult bindingResult, Long id): Edita un usuario existente.
     - eliminar(Long id): Elimina un usuario por su ID.
   - **Entradas:**
     - Parámetros de solicitud (ID de usuario, datos de usuario).
     - Resultados de validación (BindingResult).
   - **Salidas:**
     - Respuestas HTTP con el estado correspondiente y el cuerpo de respuesta (datos del usuario o mensajes de error).
2. **Servicio UsuarioService:**
   - **Métodos:**
     - listar(): Retorna una lista de todos los usuarios.
     - porId(Long id): Retorna un usuario específico por su ID.
     - guardar(Usuario usuario): Guarda un nuevo usuario o actualiza uno existente.
     - eliminar(Long id): Elimina un usuario por su ID.
   - **Interacciones con el controlador:** Simuladas mediante Mockito para aislar y probar las funcionalidades del controlador.
3. **Validaciones:**
   - **Validaciones de datos de usuario:**
     - Requerimientos de campos obligatorios (por ejemplo, el nombre del usuario).
     - Formato de correo electrónico.
   - **Resultados de validación (BindingResult):**
     - Simulación de errores de validación para pruebas de manejo de errores.
4. **Respuestas HTTP:**
   - **Códigos de estado:**
     - 200 OK: Solicitud exitosa.
     - 201 Created: Recurso creado exitosamente.
     - 400 Bad Request: Error en la solicitud debido a datos inválidos.
     - 404 Not Found: Recurso no encontrado.
     - 204 No Content: Eliminación exitosa.

CursoController

1. **Controlador CursoController:**
   - **Métodos:**
     - listar(): Lista todos los cursos.
     - detalle(Long id): Obtiene los detalles de un curso específico por su ID.

        - crear(Curso curso, BindingResult bindingResult): Crea un nuevo curso.
        - editar(Curso curso, BindingResult bindingResult, Long id): Edita un curso existente.
        - eliminar(Long id): Elimina un curso por su ID.
        - asignarUsuario(Long cursoId, Long usuarioId): Asigna un usuario a un curso.
    - **Entradas:**
        - Parámetros de solicitud (ID del curso, datos del curso, ID del usuario).
        - Resultados de validación (BindingResult).
    - **Salidas:**
        - Respuestas HTTP con el estado correspondiente y el cuerpo de respuesta (datos del curso, confirmación de asignación, o mensajes de error).
2. **Servicio CursoService:**
    - **Métodos:**
        - listar(): Retorna una lista de todos los cursos.
        - porId(Long id): Retorna un curso específico por su ID.
        - guardar(Curso curso): Guarda un nuevo curso o actualiza uno existente.
        - eliminar(Long id): Elimina un curso por su ID.
        - asignarUsuario(Long cursoId, Long usuarioId): Asigna un usuario a un curso.
    - **Interacciones con el controlador:** Simuladas mediante Mockito para aislar y probar las funcionalidades del controlador.
3. **Validaciones:**
    - **Validaciones de datos de curso:**
        - Requerimientos de campos obligatorios (por ejemplo, el nombre del curso).
    - **Resultados de validación (BindingResult):**
        - Simulación de errores de validación para pruebas de manejo de errores.
4. **Respuestas HTTP:**
    - **Códigos de estado:**
        - 200 OK: Solicitud exitosa.
        - 201 Created: Recurso creado exitosamente.
        - 400 Bad Request: Error en la solicitud debido a datos inválidos.
        - 404 Not Found: Recurso no encontrado.
        - 204 No Content: Eliminación exitosa.

Sección 3. Calendario de pruebas

Actividad de Prueba	*Recursos Asignados*	*Prerrequisitos*	*Fecha de Inicio*	*Fecha de Fin*	*Entregables/Hitos*
Planificación de Pruebas	Testers	Revisión de Requisitos	17-07-2024	17-07-2024	Plan de Pruebas Finalizado
Preparación del Entorno de Pruebas	Testers	Infraestructura Configurada	17-07-2024	17-07-2024	Entorno de Pruebas Configurado
Diseño de Casos de Prueba	Testers	Plan de Pruebas Aprobado	17-07-2024	17-07-2024	Casos de Prueba Documentados
Implementación de Pruebas Unitarias	Testers	Código Fuente Completo	18-07-2024	18-07-2024	Pruebas Unitarias Implementadas
Ejecución de Pruebas Unitarias	Testers	Pruebas Unitarias Implementadas	19-07-2024	19-07-2024	Resultados de Pruebas Unitarias
Revisión y Corrección de Errores	Testers	Resultados de Pruebas Unitarias	19-07-2024	19-07-2024	Errores Corregidos
Diseño de Pruebas de Integración	Testers	Pruebas Unitarias Completadas	20-07-2024	20-07-2024	Casos de Prueba de Integración
Ejecución de Pruebas de Integración	Testers	Casos de Prueba de Integración	20-07-2024	20-07-2024	Resultados de Pruebas de Integración
Pruebas de Aceptación de Usuario (UAT)	Testers	Pruebas de Integración Completadas	21-07-2024	21-07-2024	Resultados de Pruebas de Aceptación
Revisión Final y Aprobación	Testers	Todas las Pruebas Completadas	21-07-2024	21-07-2024	Aprobación

Sección 4. Casos de pruebas

Casos de Prueba de la clase Usuario:

i. **Crear Usuario:**

Prueba con datos válidos.

```
@Test
void crear() {
    Usuario usuario = new Usuario();
    usuario.setId(1L);
    usuario.setNombre("Usuario1");
    usuario.setEmail("usuario1@example.com");
    usuario.setPassword("password");

    when(bindingResult.hasErrors()).thenReturn(false);

when(usuarioService.guardar(any(Usuario.class))).thenReturn(usuario);

    ResponseEntity<?> response = usuarioController.crear(usuario,
bindingResult);

    assertEquals(HttpStatus.CREATED, response.getStatusCode());
    assertEquals(usuario, response.getBody());
    verify(usuarioService, times(1)).guardar(any(Usuario.class));
    System.out.println("Test exitoso");
}
```

Prueba con datos inválidos (errores de validación).

```
@Test
void crearWithErrors() {
    Usuario usuario = new Usuario();

    when(bindingResult.hasErrors()).thenReturn(true);

when(bindingResult.getFieldErrors()).thenReturn(Collections.singletonLi
st(
            new org.springframework.validation.FieldError("usuario",
"nombre", "es obligatorio")
    ));

    ResponseEntity<?> response = usuarioController.crear(usuario,
bindingResult);

    assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
    Map<String, String> errores = (Map<String, String>)
response.getBody();
    assertTrue(errores.containsKey("nombre"));
    assertEquals("El campo nombre es obligatorio",
errores.get("nombre"));
    System.out.println("Test exitoso");
}
```

ii. **Listar Usuarios:**

```
@Test
void listar() {
    Usuario usuario1 = new Usuario();
    usuario1.setId(1L);
    usuario1.setNombre("Usuario1");
    usuario1.setEmail("usuario1@example.com");
    usuario1.setPassword("password");

    Usuario usuario2 = new Usuario();
    usuario2.setId(2L);
    usuario2.setNombre("Usuario2");
    usuario2.setEmail("usuario2@example.com");
    usuario2.setPassword("password");

    List<Usuario> usuarios = Arrays.asList(usuario1, usuario2);

    when(usuarioService.listar()).thenReturn(usuarios);

    List<Usuario> result = usuarioController.listar();

    assertNotNull(result);
    assertEquals(2, result.size());
    verify(usuarioService, times(1)).listar();
    System.out.println("Test exitoso");
}
```

iii. **Detalle de Usuario:**

Prueba con un ID válido.

```
@Test
void detalle() {
    Usuario usuario = new Usuario();
    usuario.setId(1L);
    usuario.setNombre("Usuario1");
    usuario.setEmail("usuario1@example.com");
    usuario.setPassword("password");


when(usuarioService.porId(anyLong())).thenReturn(Optional.of(usuario));

    ResponseEntity<?> response = usuarioController.detalle(1L);

    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(usuario, response.getBody());
    verify(usuarioService, times(1)).porId(anyLong());
    System.out.println("Test exitoso");
}
```

Prueba con un ID no existente.

```
@Test
void detalleNotFound() {
    when(usuarioService.porId(anyLong())).thenReturn(Optional.empty());

    ResponseEntity<?> response = usuarioController.detalle(1L);

    assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
    verify(usuarioService, times(1)).porId(anyLong());
    System.out.println("Test exitoso");
}
```

iv. **Editar Usuario:**

Prueba con datos válidos y usuario existente.

```
@Test
void editar() {
    Usuario usuario = new Usuario();
    usuario.setId(1L);
    usuario.setNombre("Usuario1");
    usuario.setEmail("usuario1@example.com");
    usuario.setPassword("password");

    Usuario usuarioDB = new Usuario();
    usuarioDB.setId(1L);
    usuarioDB.setNombre("Usuario1");
    usuarioDB.setEmail("usuario1@example.com");
    usuarioDB.setPassword("password");

    when(bindingResult.hasErrors()).thenReturn(false);

when(usuarioService.porId(anyLong())).thenReturn(Optional.of(usuarioDB)
);

when(usuarioService.guardar(any(Usuario.class))).thenReturn(usuario);

    ResponseEntity<?> response = usuarioController.editar(usuario,
bindingResult, 1L);

    assertEquals(HttpStatus.CREATED, response.getStatusCode());
    assertEquals(usuario, response.getBody());
    verify(usuarioService, times(1)).guardar(any(Usuario.class));
    System.out.println("Test exitoso");
}
```

Prueba con datos inválidos.

```
@Test
void editarNotFound() {
    Usuario usuario = new Usuario();
    usuario.setId(1L);
    usuario.setNombre("Usuario1");
    usuario.setEmail("usuario1@example.com");
    usuario.setPassword("password");
```

```
    when(bindingResult.hasErrors()).thenReturn(false);
    when(usuarioService.porId(anyLong())).thenReturn(Optional.empty());

    ResponseEntity<?> response = usuarioController.editar(usuario,
bindingResult, 1L);

    assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
    verify(usuarioService, times(0)).guardar(any(Usuario.class));
    System.out.println("Test exitoso");
}
```

v. **Eliminar Usuario:**

Prueba con datos válidos.

```
@Test
void eliminar() {
    Usuario usuario = new Usuario();
    usuario.setId(1L);
    usuario.setNombre("Usuario1");
    usuario.setEmail("usuario1@example.com");
    usuario.setPassword("password");


when(usuarioService.porId(anyLong())).thenReturn(Optional.of(usuario));

    ResponseEntity<?> response = usuarioController.eliminar(1L);

    assertEquals(HttpStatus.NO_CONTENT, response.getStatusCode());
    verify(usuarioService, times(1)).eliminar(anyLong());
    System.out.println("Test exitoso");
}
```

Prueba con datos inválidos.

```
@Test
void eliminarNotFound() {
    when(usuarioService.porId(anyLong())).thenReturn(Optional.empty());

    ResponseEntity<?> response = usuarioController.eliminar(1L);

    assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
    verify(usuarioService, times(0)).eliminar(anyLong());
    System.out.println("Test exitoso");
}
```

Casos de Prueba de la clase Curso:

i. **Crear Curso:**

```
@Test
void crear() {
    // Arrange
    Curso curso = new Curso();
    when(bindingResult.hasErrors()).thenReturn(false);
```

```
    when(cursoService.guardar(any(Curso.class))).thenReturn(curso);

    // Act
    var response = cursoController.crear(curso, bindingResult);

    // Assert
    assertEquals(HttpStatus.CREATED, ((ResponseEntity<?>)
response).getStatusCode());
    verify(cursoService).guardar(any(Curso.class));
    System.out.println("Test exitoso");
}
```

ii. **Listar Cursos:**

```
void listar() {
    // Arrange
    Curso curso = new Curso(); // Asegúrate de inicializar
correctamente los datos del curso

when(cursoService.listar()).thenReturn(Collections.singletonList(curso)
);

    // Act
    var response = cursoController.listar();

    // Assert
    assertEquals(1, response.size());
    verify(cursoService).listar();
    System.out.println("Test exitoso");
}
```

iii. **Detalle de Curso:**

```
void detalle() {
    // Arrange
    Curso curso = new Curso();
    when(cursoService.porId(anyLong())).thenReturn(Optional.of(curso));

    // Act
    var response = cursoController.detalle(1L);

    // Assert
    assertEquals(HttpStatus.OK, ((ResponseEntity<?>)
response).getStatusCode());
    verify(cursoService).porId(anyLong());
    System.out.println("Test exitoso");
}
```

iv. **Editar Curso:**

```
@Test
void editar() {
    // Arrange
    Curso curso = new Curso();
    when(bindingResult.hasErrors()).thenReturn(false);
```

```
    when(cursoService.porId(anyLong())).thenReturn(Optional.of(curso));
    when(cursoService.guardar(any(Curso.class))).thenReturn(curso);

    // Act
    var response = cursoController.editar(curso, bindingResult, 1L);

    // Assert
    assertEquals(HttpStatus.CREATED, ((ResponseEntity<?>)
response).getStatusCode());
    verify(cursoService).guardar(any(Curso.class));
    System.out.println("Test exitoso");
}
```

v. **Eliminar Curso:**

```
@Test
void eliminar() {
    // Arrange
    when(cursoService.porId(anyLong())).thenReturn(Optional.of(new
Curso()));

    // Act
    var response = cursoController.eliminar(1L);

    // Assert
    assertEquals(HttpStatus.NO_CONTENT, ((ResponseEntity<?>)
response).getStatusCode());
    verify(cursoService).eliminar(anyLong());
    System.out.println("Test exitoso");
}
```

vi. **Asignar Curso:**

```
@Test
void asignarUsuario() {
    // Arrange
    Usuario usuario = new Usuario();
    when(cursoService.agregarUsuario(any(Usuario.class),
anyLong())).thenReturn(Optional.of(usuario));

    // Act
    var response = cursoController.asignarUsuario(usuario, 1L);

    // Assert
    assertEquals(HttpStatus.CREATED, ((ResponseEntity<?>)
response).getStatusCode());
    verify(cursoService).agregarUsuario(any(Usuario.class), anyLong());
    System.out.println("Test exitoso");
}
```

Sección 6. Referencias

Nº Documento	*Título del Documento*	*Fecha*	*Autor*
1	**Plan de Pruebas del Sistema de Gestión de Usuarios**	2024-07-10	Equipo de Desarrollo y QA
2	**Guía de Pruebas Unitarias y de Integración en Java**	2024-06-25	Documentación Técnica del Proyecto
3	**Manual de Usuario del Sistema de Gestión de Cursos**	2024-06-30	Equipo de Documentación Técnica
4	**Documentación Técnica de la Arquitectura del Sistema**	2024-06-15	Arquitecto de Software
5	**Especificación de Requisitos Funcionales**	2024-05-20	Analista de Requisitos
6	**Informe de Resultados de Pruebas Unitarias**	2024-07-21	Equipo de QA
7	**Estrategia de Pruebas para la Aplicación de Usuario**	2024-07-05	Líder de Pruebas
8	**Normas y Estándares de Pruebas en Desarrollo de Software**	2024-07-01	Comité

Sección 7. Glosario

- **API (Interfaz de Programación de Aplicaciones)**: Un conjunto de reglas y herramientas para construir software y aplicaciones, que permite la interacción entre diferentes componentes o servicios.

- **BindingResult**: Una interfaz de Spring Framework que encapsula los errores de validación ocurridos durante la vinculación de datos entre un formulario y un objeto de dominio.

- **Controller**: En el contexto de aplicaciones web, es una clase que maneja las solicitudes del usuario y coordina la respuesta adecuada a través de la lógica de negocio.

- **JUnit**: Un marco de pruebas unitarias para Java que permite ejecutar pruebas para verificar el comportamiento de los métodos de una aplicación, asegurando que el código funcione como se espera.

- **Mock**: Un objeto simulado que imita el comportamiento de un objeto real en un entorno controlado para propósitos de prueba, permitiendo verificar el comportamiento de unidades de código sin depender de implementaciones reales.

- **Pruebas Unitarias**: Tipo de prueba de software que verifica el funcionamiento de una unidad o componente específico del código de manera aislada, asegurando que cada componente funcione de acuerdo con sus especificaciones.

- **Pruebas de Integración**: Tipo de prueba que verifica la interacción entre diferentes componentes del sistema para asegurar que funcionen correctamente juntos y se comuniquen de manera efectiva.

- **ResponseEntity**: Una clase de Spring Framework que representa la respuesta HTTP que se devuelve desde un controlador, incluyendo el cuerpo de la respuesta, los encabezados y el código de estado HTTP.

- **Spring Boot**: Un marco de trabajo de Java que facilita el desarrollo de aplicaciones basadas en Spring, proporcionando configuraciones automáticas y convenciones que simplifican la creación de aplicaciones independientes y listas para producción.

- **Usuario**: En el contexto del sistema, una entidad que representa a una persona o entidad que interactúa con el sistema, almacenando información relevante como identificación, nombre y otros datos personales.

- **ValidationException**: Una excepción que se lanza cuando los datos proporcionados no cumplen con los criterios de validación establecidos, indicando que los datos no son válidos para el proceso en curso.

- **XML (Lenguaje de Marcado Extensible)**: Un formato de archivo que se utiliza para almacenar y transportar datos en un formato estructurado y legible por humanos y máquinas, facilitando la interoperabilidad entre sistemas.

- **HTTP (Protocolo de Transferencia de Hipertexto)**: Un protocolo de comunicación utilizado para la transferencia de datos en la web, permitiendo la comunicación entre clientes y servidores a través de solicitudes y respuestas.

Sección 8. Historial de Revisión

[Identificar los cambios al Plan de Pruebas.]

Versión	Fecha	Nombre	Descripción
1	17/07/2024	Testers	Creación inicial Plan de Pruebas. Sección 1.
1	18/07/2024	Testers	Adición de Sección 3.
1	19/07/2024	Testers	Inclusión de Sección 5.
1	20/07/2024	Testers	Revisión y ajuste Sección 4.
1	21/07/2024	Testers	Finalización plan de pruebas

Sección 9. Apéndices

[Incluir cualquier apéndice relevante.]