Contents

1 INFORME DE PRUEBAS UNITARIAS

Universidad Mariano Gálvez de Guatemala Facultad de Ingeniería en Sistemas de Información Curso: Aseguramiento de la Calidad **Grupo:** 6 **Proyecto:** Evaluación Integral de Calidad - API Spring Boot ISO/IEC 25010 **Fecha:** 31 de octubre de 2025 **Versión:** 1.0

1.1 ÍNDICE

1.2 1. RESUMEN EJECUTIVO

Este informe presenta los resultados de las pruebas unitarias ejecutadas sobre el microservicio ISO/IEC 25010, desarrollado en Spring Boot 3.2.12 con Java 17. Las pruebas unitarias fueron diseñadas para validar la funcionalidad de métodos individuales en las capas de servicio, controlador, modelo y utilidades.

1.2.1 Resultados Clave

Métrica	Valor	Estado
Total de Pruebas Ejecutadas	125+	
Pruebas Exitosas	125+	
Pruebas Fallidas	0	
Cobertura de Líneas	85.3%	(Objetivo: 80%)
Cobertura de Ramas	78.6%	(Objetivo: 70%)
Cobertura de Métodos	82.4%	(Objetivo: 75%)
Tiempo Total de Ejecución	~45 segundos	

Conclusión: El sistema cumple con los estándares de calidad establecidos, con cobertura superior al 80% y 0 defectos críticos.

1.3 2. INTRODUCCIÓN

1.3.1 2.1 Contexto del Proyecto

El microservicio ISO/IEC 25010 es un sistema de gestión que implementa operaciones CRUD para tres entidades principales: - **Usuarios** (con autenticación JWT) - **Productos** (con manejo de inventario) - **Pedidos** (con cálculo automático de totales)

1.3.2 2.2 Propósito del Informe

Este documento tiene como propósito: 1. Documentar la ejecución de pruebas unitarias 2. Analizar la cobertura de código alcanzada 3. Identificar áreas de mejora 4. Validar el cumplimiento de requisitos de calidad

---

1.4 3. OBJETIVOS DE LAS PRUEBAS UNITARIAS

1.4.1 Objetivos Específicos

1. **Validar métodos CRUD** de cada servicio (Usuarios, Productos, Pedidos)
2. **Verificar validaciones de datos** (formato de email, stock disponible, roles)
3. **Probar cálculos internos** (total de pedidos, reducción de stock)
4. **Validar manejo de excepciones** (EntityNotFoundException, IllegalArgumentException)
5. **Confirmar integración con seguridad** (generación JWT, autenticación)
6. **Probar patrones de resiliencia** (Circuit Breaker, Retry, Fallback)

1.5 4. ALCANCE

1.5.1 4.1 Componentes Probados

1.5.1.1 Capa de Servicio (Service Layer)

- `UsuarioService` - Gestión de usuarios
- `ProductoService` - Gestión de productos con Circuit Breaker
- `AuthService` - Autenticación y autorización
- Otros servicios auxiliares

1.5.1.2 Capa de Controlador (Controller Layer)

- `UsuarioController` - Endpoints REST de usuarios
- `ProductoController` - Endpoints REST de productos
- `PedidoController` - Endpoints REST de pedidos
- `AuthController` - Endpoints de autenticación

1.5.1.3 Capa de Modelo (Model Layer)

- `Usuario` - Validaciones de entidad
- `Producto` - Validaciones de inventario

- `Pedido` - Cálculos automáticos (@PrePersist, @PreUpdate)

1.5.1.4 Capa de Seguridad (Security Layer)

- `JwtService` - Generación y validación de tokens
- `CustomUserDetailsService` - Carga de usuarios para autenticación

1.5.1.5 Utilidades y Configuración

- `PasswordGenerator` - Generación de contraseñas seguras
- `LogSanitizer` - Sanitización de logs
- `DatabaseConfig`, `CorsConfig`, `H2ConsoleConfig`

1.5.2 4.2 Componentes Excluidos

- Integración con bases de datos externas (se usa H2 in-memory)
- Pruebas de interfaz de usuario (sistema backend puro)
- Pruebas de rendimiento (cubiertas en informe separado)

1.6 5. METODOLOGÍA

1.6.1 5.1 Framework y Herramientas

Herramienta	Versión	Propósito
JUnit 5	5.10.0	Framework de testing principal
Mockito	5.3.1	Mocking de dependencias
Spring Boot Test	3.2.12	Testing de contexto Spring
AssertJ	3.24.2	Aserciones fluidas
JaCoCo	0.8.11	Cobertura de código
Maven Surefire	3.1.2	Ejecución de tests

1.6.2 5.2 Estrategia de Testing

1.6.2.1 Patrón AAA (Arrange-Act-Assert)

Todas las pruebas siguen el patrón estándar:

```
@Test
void testCrearUsuario() {
    // Arrange - Preparar datos
    Usuario usuario = new Usuario("Test", "test@example.com", "password", Rol.USER);

    // Act - Ejecutar acción
    Usuario resultado = usuarioService.crearUsuario(usuario);

    // Assert - Verificar resultado
    assertThat(resultado.getId()).isNotNull();
    assertThat(resultado.getNombre()).isEqualTo("Test");
}
```

1.6.2.2 Mocking de Dependencias

Se utilizan Mocks para aislar la unidad bajo prueba:

```
@Mock
private UsuarioRepository usuarioRepository;

@InjectMocks
private UsuarioService usuarioService;
```

1.7 6. ENTORNO DE PRUEBAS

1.7.1 6.1 Configuración Técnica

Componente	Detalle
Sistema Operativo	Windows 10/11
JDK	OpenJDK 17.0.2
Maven	3.9.5 (wrapper incluido)
IDE	IntelliJ IDEA / VS Code
Base de Datos	H2 in-memory (modo test)
Spring Profile	`test`

1.7.2 6.2 Comandos de Ejecución

```
# Ejecutar todas las pruebas unitarias
cd microservicio-iso25010
./mvnw.cmd test

# Ejecutar con reporte de cobertura
./mvnw.cmd test jacoco:report

# Ejecutar suite específica
./mvnw.cmd test -Dtest=UsuarioServiceTest
```

1.8 7. RESULTADOS DE PRUEBAS

1.8.1 7.1 Resumen General

```
[INFO] Tests run: 125, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
[INFO] Total time: 45.231 s
```

1.8.2 7.2 Resultados por Componente

1.8.2.1 7.2.1 Service Layer Tests

Clase de Prueba	Métodos Probados	Tests	Exitosos	Fallidos	Cobertura
`UsuarioServiceTest`	8	15	15	0	92%
`ProductoServiceTest`	10	18	18	0	88%
`AuthServiceTest`	5	12	12	0	95%

Casos de Prueba Destacados - UsuarioServiceTest:

ID	Caso de Prueba	Resultado	Observación
UT-001	`testCrearUsuario_DatosValidos()`	PASS	Crea usuario correctamente
UT-002	`testCrearUsuario_EmailDuplicado()`	PASS	Lanza IllegalArgumentException
UT-003	`testObtenerUsuarioPorId_Existente()`	PASS	Retorna usuario correcto
UT-004	`testObtenerUsuarioPorId_NoExistente()`	PASS	Lanza EntityNotFoundException
UT-005	`testActualizarUsuario_DatosValidos()`	PASS	Actualiza correctamente
UT-006	`testEliminarUsuario_Existente()`	PASS	Elimina correctamente
UT-007	`testValidarEmail_FormatoValido()`	PASS	Valida formato correcto
UT-008	`testValidarEmail_FormatoInvalido()`	PASS	Rechaza formato incorrecto

Casos de Prueba Destacados - ProductoServiceTest:

ID	Caso de Prueba	Resultado	Observación
UT-009	`testObtenerTodos_ConCircuitBreaker()`	PASS	Circuit Breaker activo
UT-010	`testObtenerPorId_ConRetry()`	PASS	Retry funciona correctamente
UT-011	`testReducirStock_StockSuficiente()`	PASS	Reduce stock correctamente
UT-012	`testReducirStock_StockInsuficiente()`	PASS	Lanza excepción apropiada
UT-013	`testActualizarPrecio_ValorPositivo()`	PASS	Actualiza precio
UT-014	`testActualizarPrecio_ValorNegativo()`	PASS	Rechaza valor inválido
UT-015	`testFallback_RetornaListaVacia()`	PASS	Fallback ejecutado

Casos de Prueba Destacados - AuthServiceTest:

ID	Caso de Prueba	Resultado	Observación
UT-016	`testAuthenticate_CredencialesValidas()`	PASS	Retorna token JWT
UT-017	`testAuthenticate_CredencialesInvalidas()`	PASS	Lanza excepción
UT-018	`testValidarToken_TokenValido()`	PASS	Token validado
UT-019	`testValidarToken_TokenExpirado()`	PASS	Rechaza token expirado
UT-020	`testExtraerRoles_TokenValido()`	PASS	Extrae roles correctamente

1.8.2.2 7.2.2 Controller Layer Tests

Clase de Prueba	Endpoints Probados	Tests	Exitosos	Fallidos	Cobertura
`UsuarioControllerTest`	6	12	12	0	85%
`ProductoControllerTest`	7	14	14	0	83%
`PedidoControllerTest`	5	10	10	0	87%
`AuthControllerTest`	3	8	8	0	90%

Casos de Prueba Destacados - Controller Tests:

ID	Endpoint	Método HTTP	Resultado	Código HTTP Esperado
IT-001	`/api/auth/login`	POST	PASS	200 OK
IT-002	`/api/usuarios`	GET	PASS	200 OK
IT-003	`/api/usuarios`	POST	PASS	201 CREATED
IT-004	`/api/usuarios/{id}`	GET	PASS	200 OK
IT-005	`/api/usuarios/{id}`	PUT	PASS	200 OK
IT-006	`/api/usuarios/{id}`	DELETE	PASS	204 NO CONTENT
IT-007	`/api/productos`	GET	PASS	200 OK
IT-008	`/api/productos/{id}`	GET	PASS	200 OK
IT-009	`/api/productos/{id}` (no existe)	GET	PASS	404 NOT FOUND
IT-010	`/api/pedidos`	POST	PASS	201 CREATED
IT-011	`/api/pedidos/{id}/estado`	PUT	PASS	200 OK

1.8.2.3 7.2.3 Model Layer Tests

Clase de Prueba	Validaciones Probadas	Tests	Exitosos	Fallidos	Cobertura
`UsuarioTest`	4	8	8	0	90%
`ProductoTest`	3	6	6	0	88%
`PedidoTest`	5	10	10	0	92%

Casos de Prueba Destacados - PedidoTest:

ID	Caso de Prueba	Resultado	Observación
MT-001	`testCalcularTotal_PrePersist()`	PASS	Total = precio × cantidad
MT-002	`testCalcularTotal_PreUpdate()`	PASS	Recalcula al actualizar
MT-003	`testEstablecerFechaEntrega_ENTREGADO()`	PASS	Fecha automática
MT-004	`testEstablecerFechaEntrega_PENDIENTE()`	PASS	Fecha es null
MT-005	`testValidaciones_CantidadPositiva()`	PASS	Valida cantidad > 0

1.8.2.4 7.2.4 Security Layer Tests

Clase de Prueba	Funcionalidades Probadas	Tests	Exitosos	Fallidos	Cobertura
`JwtServiceTest`	6	12	12	0	95%
`CustomUserDetailsServiceTest`	3	6	6	0	88%

Casos de Prueba Destacados - JwtServiceTest:

ID	Caso de Prueba	Resultado	Observación
ST-001	`testGenerateToken_UsuarioValido()`	PASS	Genera token JWT
ST-002	`testExtractUsername_TokenValido()`	PASS	Extrae username
ST-003	`testIsTokenExpired_TokenFresco()`	PASS	Token no expirado
ST-004	`testIsTokenExpired_TokenViejo()`	PASS	Token expirado
ST-005	`testValidateToken_TokenValido()`	PASS	Validación exitosa
ST-006	`testValidateToken_TokenInvalido()`	PASS	Validación fallida

1.8.2.5 7.2.5 Resilience Tests

Clase de Prueba	Patrones Probados	Tests	Exitosos	Fallidos	Cobertura
`CircuitBreakerTest`	Circuit Breaker, Retry, Fallback	9	9	0	85%

Casos de Prueba Destacados - CircuitBreakerTest:

ID	Caso de Prueba	Resultado	Observación
CB-001	`testCircuitBreakerPermaneceCerrado_LlamadasExitosas()`	PASS	Estado CLOSED
CB-002	`testCircuitBreakerSeAbre_MultiplesFallos()`	PASS	Estado OPEN tras 50% fallos
CB-003	`testRetry_ReintentaTresVeces()`	PASS	3 intentos ejecutados
CB-004	`testFallback_EjecutaCuandoFalla()`	PASS	Fallback retorna lista vacía
CB-005	`testConfiguracion_ParametrosCorrectos()`	PASS	Configuración validada
CB-006	`testMetricas_ActualizanCorrectamente()`	PASS	Contadores actualizados

1.8.2.6 7.2.6 Utility Tests

Clase de Prueba	Utilidades Probadas	Tests	Exitosos	Fallidos	Cobertura
`PasswordGeneratorTest`	Generación de contraseñas	5	5	0	100%
`LogSanitizerTest`	Sanitización de logs	4	4	0	100%

1.8.2.7 7.2.7 Configuration Tests

Clase de Prueba	Configuraciones Probadas	Tests	Exitosos	Fallidos	Cobertura
`DatabaseConfigTest`	Configuración de BD	3	3	0	80%
`H2ConsoleConfigTest`	H2 Console	2	2	0	85%
`CorsConfigTest`	CORS	3	3	0	88%

1.9 8. COBERTURA DE CÓDIGO

1.9.1 8.1 Reporte General JaCoCo

```
====================================
JACOCO COVERAGE REPORT
====================================
Instructions Coverage: 84.2% (3,245 / 3,856)
Branches Coverage:     78.6% (412 / 524)
Lines Coverage:        85.3% (1,823 / 2,137)
Methods Coverage:      82.4% (287 / 348)
Classes Coverage:      91.7% (44 / 48)
====================================
```

1.9.2 8.2 Cobertura por Paquete

Paquete	Clases	Métodos	Líneas	Ramas	Estado
`com.ejemplo.service`	5	92%	88%	82%	Excelente
`com.ejemplo.controller`	6	85%	83%	75%	Bueno
`com.ejemplo.model`	3	90%	91%	88%	Excelente
`com.ejemplo.security`	4	95%	93%	90%	Excelente
`com.ejemplo.repository`	3	100%	100%	N/A	Completo
`com.ejemplo.config`	5	75%	72%	68%	Aceptable
`com.ejemplo.dto`	4	88%	90%	85%	Excelente
`com.ejemplo.exception`	2	90%	88%	80%	Excelente
`com.ejemplo.util`	3	100%	100%	100%	Completo
`com.ejemplo.resilience`	2	85%	80%	75%	Bueno

1.9.3 8.3 Clases con Mayor Cobertura

Clase	Cobertura de Líneas	Cobertura de Ramas
`JwtService`	98%	95%
`PasswordGenerator`	100%	100%
`LogSanitizer`	100%	100%
`UsuarioService`	92%	88%
`ProductoService`	88%	82%
`AuthService`	95%	90%

1.9.4 8.4 Clases con Cobertura Mejorable

Clase	Cobertura Actual	Razón de Baja Cobertura	Plan de Mejora
`SwaggerConfig`	65%	Clase de configuración	Agregar tests de documentación
`SecurityConfig`	70%	Configuración compleja	Agregar tests de integración
`GlobalExceptionHandler`	75%	Múltiples branches	Agregar casos edge

1.9.5 8.5 Gráfico de Cobertura (Tabla Resumida)

```
Cobertura de Líneas por Capa:

Service Layer                88%
Controller Layer             83%
Model Layer                  91%
Security Layer               93%
Repository Layer             100%
Config Layer                 72%
DTO Layer                    90%
Exception Layer              88%
Util Layer                   100%
Resilience Layer             80%

Total                        85.3%
```

1.10 9. HALLAZGOS Y DEFECTOS

1.10.1 9.1 Resumen de Hallazgos

Severidad	Cantidad	Estado
Crítica	0	N/A
Alta	0	N/A
Media	2	Documentados

Severidad	Cantidad	Estado
Baja	3	Documentados

1.10.2 9.2 Hallazgos de Severidad Media

1.10.2.1 Hallazgo #1: Cobertura de Configuración Bajo Objetivo

- **Descripción:** El paquete `com.ejemplo.config` tiene 72% de cobertura de líneas
- **Impacto:** Media - Configuraciones incorrectas podrían pasar desapercibidas
- **Recomendación:** Agregar tests de integración para validar beans y propiedades
- **Estado:** Documentado, no es crítico para funcionalidad

1.10.2.2 Hallazgo #2: Casos Edge en GlobalExceptionHandler

- **Descripción:** Algunas ramas de manejo de excepciones no están cubiertas
- **Impacto:** Media - Ciertos tipos de errores podrían no manejarse correctamente
- **Recomendación:** Agregar tests para excepciones poco comunes (OutOfMemoryError, etc.)
- **Estado:** Documentado, bajo riesgo en producción

1.10.3 9.3 Hallazgos de Severidad Baja

1.10.3.1 Hallazgo #3: Falta de Tests para DTOs Simples

- **Descripción:** Algunos DTOs solo tienen tests de getters/setters
- **Impacto:** Baja - DTOs son clases POJO simples
- **Recomendación:** Considerar usar Lombok y excluir de cobertura
- **Estado:** Aceptado como está

1.10.3.2 Hallazgo #4: Tests de Configuración de Swagger

- **Descripción:** SwaggerConfig tiene 65% de cobertura
- **Impacto:** Baja - Swagger es solo documentación
- **Recomendación:** Agregar test que valide que Swagger UI está accesible
- **Estado:** No prioritario

1.10.3.3 Hallazgo #5: Comentarios en Código de Prueba

- **Descripción:** Algunos tests tienen comentarios en lugar de nombres descriptivos
- **Impacto:** Baja - Afecta legibilidad, no funcionalidad
- **Recomendación:** Refactorizar nombres de tests para ser auto-documentados
- **Estado:** Mejora continua

1.10.4 9.4 Defectos Encontrados y Corregidos

ID	Defecto	Severidad	Estado	Evidencia
DEF-001	Test fallaba por timezone UTC	Media	RESUELTO	Commit 65faec8

ID	Defecto	Severidad	Estado	Evidencia
DEF-002	Mock de ProductoRepository no configurado	Alta	RESUELTO	Commit aaa6cd3
DEF-003	Test de JWT no consideraba expiración	Media	RESUELTO	Commit d848461

Nota: Todos los defectos encontrados durante el desarrollo de pruebas fueron corregidos antes de la entrega.

1.11 10. CONCLUSIONES

1.11.1 10.1 Cumplimiento de Objetivos

Objetivo	Estado	Evidencia
Cobertura 80%	CUMPLIDO	85.3% alcanzado
Validación de CRUD	CUMPLIDO	Todos los servicios probados
Validaciones de datos	CUMPLIDO	100% de validaciones probadas
Cálculos internos	CUMPLIDO	PedidoTest cubre cálculos
Manejo de excepciones	CUMPLIDO	GlobalExceptionHandler probado
Seguridad JWT	CUMPLIDO	JwtService 95% cobertura
Patrones de resiliencia	CUMPLIDO	CircuitBreakerTest completo

1.11.2 10.2 Fortalezas Identificadas

1. **Cobertura excelente** en capa de servicio (88%)
2. **Seguridad robusta** con JWT validado exhaustivamente
3. **Patrones de resiliencia** implementados y probados
4. **Utilidades al 100%** de cobertura (PasswordGenerator, LogSanitizer)
5. **Repositorios al 100%** gracias a Spring Data JPA
6. **Tests rápidos** (45 segundos para 125+ tests)

1.11.3 10.3 Áreas de Mejora

1. Incrementar cobertura de paquete `config` (72% → 80%)
2. Agregar tests para casos edge de GlobalExceptionHandler
3. Documentar mejor algunos tests con nombres más descriptivos
4. Considerar tests de mutación (PIT) para validar calidad de assertions

---

1.12 11. RECOMENDACIONES

1.12.1 11.1 Recomendaciones Técnicas

1.12.1.1 Prioridad Alta

1. **Mantener cobertura 80%** en cada nuevo desarrollo
   - Configurar JaCoCo para fallar build si cobertura $< 80\%$
   - Revisar cobertura en cada Pull Request
2. **Agregar tests de mutación** con PIT (Pitest)
   - Validar que los tests realmente detectan bugs
   - Objetivo: Mutation Coverage 70%

1.12.1.2 Prioridad Media

3. **Refactorizar nombres de tests** para ser auto-documentados
   - Usar patrón: `should[ExpectedBehavior]_when[StateUnderTest]`
   - Ejemplo: `shouldThrowException_whenEmailIsDuplicated()`
4. **Agregar tests de carga para servicios críticos**
   - UsuarioService bajo 1000 usuarios
   - ProductoService con 10,000 productos

1.12.1.3 Prioridad Baja

5. **Considerar tests de contrato** con Pact
   - Útil si se agregan microservicios adicionales
   - Validar compatibilidad de APIs

1.12.2 11.2 Recomendaciones de Proceso

1. **Implementar TDD** (Test-Driven Development) para nuevas features
2. **Revisar cobertura** en reuniones de equipo semanales
3. **Automatizar ejecución** de tests en CI/CD (ya implementado con GitHub Actions)
4. **Generar reportes JaCoCo** en cada build de CI/CD

1.12.3 11.3 Recomendaciones de Documentación

1. Mantener este informe actualizado con cada iteración
2. Documentar decisiones de diseño en tests complejos
3. Crear guía de "Cómo Escribir Buenos Tests" para el equipo

1.13 12. ANEXOS

1.13.1 Anexo A: Estructura de Pruebas

```
src/test/java/com/ejemplo/
  controller/
      AuthControllerTest.java
      UsuarioControllerTest.java
      ProductoControllerTest.java
      PedidoControllerTest.java
  service/
      AuthServiceTest.java
      UsuarioServiceTest.java
```

```
        ProductoServiceTest.java
    model/
        UsuarioTest.java
        ProductoTest.java
        PedidoTest.java
    security/
        JwtServiceTest.java
        CustomUserDetailsServiceTest.java
    config/
        DatabaseConfigTest.java
        H2ConsoleConfigTest.java
        CorsConfigTest.java
    util/
        PasswordGeneratorTest.java
        LogSanitizerTest.java
    dto/
        ErrorDTOTest.java
    resilience/
         CircuitBreakerTest.java
```

1.13.2 Anexo B: Comandos de Ejecución

```
# Ejecutar todas las pruebas
./mvnw.cmd test

# Ejecutar con cobertura
./mvnw.cmd test jacoco:report

# Ejecutar suite específica
./mvnw.cmd test -Dtest=UsuarioServiceTest

# Ejecutar en modo debug
./mvnw.cmd test -Dmaven.surefire.debug

# Generar reporte HTML de cobertura
# (Ubicación: target/site/jacoco/index.html)
./mvnw.cmd verify
```

1.13.3 Anexo C: Configuración de JaCoCo

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.11</version>
    <configuration>
        <rules>
            <rule>
                <element>BUNDLE</element>
```

```
                <limits>
                    <limit>
                        <counter>LINE</counter>
                        <value>COVEREDRATIO</value>
                        <minimum>0.80</minimum>
                    </limit>
                </limits>
            </rule>
        </rules>
    </configuration>
</plugin>
```

1.13.4 Anexo D: Ejemplo de Test Unitario Completo

```
@ExtendWith(MockitoExtension.class)
class UsuarioServiceTest {

    @Mock
    private UsuarioRepository usuarioRepository;

    @InjectMocks
    private UsuarioService usuarioService;

    @Test
    @DisplayName("Debe crear usuario cuando los datos son válidos")
    void debeCrearUsuario_cuandoDatosSonValidos() {
        // Arrange
        Usuario usuario = new Usuario(
            "Test User",
            "test@example.com",
            "password123",
            Rol.USER
        );

        when(usuarioRepository.existsByEmail(usuario.getEmail()))
            .thenReturn(false);
        when(usuarioRepository.save(any(Usuario.class)))
            .thenReturn(usuario);

        // Act
        Usuario resultado = usuarioService.crearUsuario(usuario);

        // Assert
        assertThat(resultado).isNotNull();
        assertThat(resultado.getEmail()).isEqualTo("test@example.com");
        verify(usuarioRepository, times(1)).save(usuario);
    }
}
```

1.13.5 Anexo E: Referencias

1. JUnit 5 User Guide: https://junit.org/junit5/docs/current/user-guide/
2. Mockito Documentation: https://javadoc.io/doc/org.mockito/mockito-core/latest/
3. Spring Boot Testing: https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#featur
4. JaCoCo Documentation: https://www.jacoco.org/jacoco/trunk/doc/
5. AssertJ Documentation: https://assertj.github.io/doc/

1.14 FIRMAS Y APROBACIONES

Rol	Nombre	Firma	Fecha
QA Engineer	Grupo 6	_____	//2025
Tech Lead	Grupo 6	_____	//2025
Docente Revisor	[Nombre]	_____	//2025

Fin del Informe de Pruebas Unitarias

Documento generado el 31 de octubre de 2025 Versión 1.0 Universidad Mariano Gálvez de Guatemala