INFORME DE TESTING WIS

Grupo 1-C1.020 | https://github.com/PDJ6975/Acme-ANS

Nombre	Correo Corporativo
Antonio Rodríguez Calderón	antrodcal@alum.us.es
Adrián Ramírez Gil	adrramgil@alum.us.es
Jianwu Hu	jiahu3@alum.us.es
Pablo Castrillón Mora	pabcasmor1@alum.us.es
Pablo Olivencia Moreno	pabolimor@alum.us.es

17 DE FEBRERO DE 2025

DISEÑO Y PRUEBAS II

Tabla de Contenido

1. Introducción

El objetivo de este informe es reflejar los conocimientos adquiridos sobre el testing de sistemas de información web (WIS), abarcando conceptos clave, tipos de pruebas, mejores prácticas y herramientas utilizadas en el proceso de validación y verificación del software.

2. Conceptos Generales sobre Testing

El testing es el proceso de ejecución del software con entradas específicas para verificar que el comportamiento del sistema cumple con los requerimientos establecidos. Existen dos objetivos principales:

1. Verificación: Validar que el sistema está construido correctamente ("Did you build the thing right?").
2. Validación: Comprobar que el sistema cumple con las necesidades del usuario ("Did you build the right thing?").

2.1 Indicadores de Desempeño Definidos

- SUT (System Under Test): Módulo o componente que se está probando.
- Test Suite: Conjunto de pruebas que verifican el comportamiento de un sistema.
- Test Case: Caso de prueba individual con entrada y salida esperada.
- Bug: Error o defecto encontrado en el software.
- Debugging: Proceso de identificar y corregir errores.

3. Tipo de testing en WIS

3.1. Testing Unitario

- Pruebas de menor granularidad.
- Se centran en clases, servicios, entidades y validadores.
- Son pruebas de caja blanca que dependen del conocimiento de la implementación.
- Se ejecutan rápidamente y deben ser independientes.
- Uso de test doubles para simular dependencias externas.

3.2. Testing de Integración

- Evalúa la comunicación entre diferentes módulos.
- Se prueba la interacción entre controladores, servicios y repositorios.
- En Spring Boot, se usa `@DataJpaTest` para probar interacciones con bases de datos sin cargar toda la aplicación.

3.3. Testing de Aceptación y End-to-End (E2E)

- Simulan escenarios completos de usuario.
- Se pueden realizar de forma externa (probando la aplicación en su totalidad) o interna (simulando llamadas HTTP con `MockMvc`).
- Se ejecutan en un entorno de pruebas con `@SpringBootTest`.

4. Estrategias de cobertura en Testing

Para garantizar una alta calidad en las pruebas, se deben seguir estrategias que maximicen la cobertura del código:

- Estrategia por sentencias: Asegurar que todas las líneas de código sean ejecutadas al menos una vez.
- Estrategia por condicionales: Probar todas las ramas de una estructura de control.
- Estrategia por bucles: Ejecutar los bucles con 0, 1 y varias iteraciones.
- Estrategia por valores frontera: Validar los límites de los datos de entrada.

5. Uso de Test Doubles

Para aislar componentes en pruebas unitarias, se usan diferentes tipos de test doubles:

- Mocks: Simulan objetos reales verificando su comportamiento.
- Stubs: Devuelven valores predefinidos.
- Fakes: Implementaciones livianas de APIs.
- Spies: Guardan información sobre cómo fueron llamados.
- Dummies: Objetos de relleno que no tienen funcionalidad.

En Spring Boot, `@MockBean` permite sustituir dependencias con mocks dentro del contexto de pruebas.

6. Herramientas de Testing en WIS

- JUnit 5: Framework principal para pruebas unitarias en Java.
- Mockito: Framework para crear mocks y stubs.
- MockMvc: Simula llamadas HTTP a controladores en pruebas de integración.
- AssertJ: Permite hacer aserciones más legibles y expresivas.

7. Buenas Prácticas en Testing

- Principio FIRST;
    - Fast: Las pruebas deben ejecutarse rápidamente.
    - Independent: No deben depender de otras pruebas.
    - Repeatable: Deben producir los mismos resultados en cada ejecución.
    - Self-checking: Deben validarse a sí mismas.
    - Timely: Deben escribirse al mismo tiempo que el código.
-
- No abusar de los mocks: Un exceso de mocks puede dificultar el mantenimiento.
- Mantener las pruebas simples y bien estructuradas.

8. Conclusión

El testing en sistemas de información web es un componente esencial para garantizar la calidad del software. Aplicando las estrategias adecuadas y utilizando las herramientas correctas, se puede asegurar que la aplicación cumple con los requerimientos funcionales y no funcionales. Un enfoque estructurado y disciplinado en las pruebas contribuye a reducir errores, mejorar la mantenibilidad y ofrecer una mejor experiencia a los usuarios finales.